Summer 2021

LSTM-Based Model For Human Brain Decisions Using EEG Signals Analysis

Lorela Bano

LSTM-BASED MODEL FOR HUMAN BRAIN DECISIONS USING EEG SIGNALS ANALYSIS

by

LORELA BANO

(Under the Direction of Ionut Emil Iacob)

ABSTRACT

As machine learning models become more sophisticated, and biometric data becomes more readily available through new non-invasive technologies, it becomes increasingly possible to gain access to interesting biometric data that could revolutionize Human Computer Interaction. In this research, we propose a framework to assess and quantify human preference (like or dislike) on presenting various external visual stimuli. Our framework relies on an Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) based model and on electroencephalogram (EEG) signals analysis to predict Like or Dislike preference of human subjects when presented with various marketing images.

INDEX WORDS: Electroencephalography, RNN, LSTM, Fourier transform, BCI

2009 Mathematics Subject Classification: 62H30, 68H10

LSTM-BASED MODEL FOR HUMAN BRAIN DECISIONS USING EEG SIGNALS ANALYSIS

by

LORELA BANO

B.S., University of Tirana, Albania, 2018

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

LSTM-BASED MODEL FOR HUMAN BRAIN DECISIONS USING EEG SIGNALS ANALYSIS

by

LORELA BANO

Major Professor:	Ionut Emil Iacob
Committee:	Felix Hamza-Lup
	Goran Lesaja

Electronic Version Approved:
July 2021

ACKNOWLEDGMENTS

I consider it an honor having Dr. Ionut Emil Iacob as my thesis advisor. Thank you for being a great professor and advisor, for your help, patience, and support. A special thank you to Dr. Enka Lakuriqi and Dr. Jimmy Dillies for giving me the opportunity to pursue my master's degree at Georgia Southern University. I would like to thank my committee members Dr. Felix Hamza-Lup and Dr. Goran Lesja, for their time and their valuable feedback. I would also like to acknowledge all professors I had the chance to take classes with during my studies. I am very thankful to my family and friends for their emotional support.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS 2

LIST OF TABLES 5

LIST OF FIGURES 6

CHAPTER

1 INTRODUCTION 9

1.1 Overview 11

2 NEURAL NETWORKS 12

2.1 Introduction to Neural Networks 12

2.2 A Simple Neural Network 13

2.3 BackPropagation 16

2.4 Recurrent Neural Networks 18

2.4.1 BackPropagation Through Time 20

2.4.2 Vanishing and exploding gradient problems 22

2.5 Long short-term memory (LSTM) 23

3 DATA DESCRIPTION 26

3.1 Neuromarketing 26

3.2 Electroencephalogram (EEG) 26

3.3 Fourier Transformation 27

3.4 The experiment 29

4 THE RNN-BASED MODEL FOR THE BINARY CLASSIFICATION OF THE BRAIN WAVE SIGNALS 32

4.1 Data aggregation and brain wave components extraction 33

4.1.1 Data preprocessing and notations 33

4.1.2 Filtering the brain wave components 36

4.2 The RNN-based classification model 38

5 EXPERIMENTAL RESULTS 41

5.1 Data statistics 41

5.1.1 Distributions of means and standard deviations for each brain wave component 42

5.1.2 Two data features: means and standard deviations 48

5.2 Single brain wave signal components classification 51

5.3 Binary classification using multiple brain wave signal components 55

6 CONCLUSION AND FUTURE WORK 61

REFERENCES 63

APPENDICES

A FILTERING AND SCALING 67

B DATA STATS 71

C RNN MODEL 83

LIST OF TABLES

Table Page

5.1 Mean accuracies for each brain wave component used for analysis . . . 54

LIST OF FIGURES

Figure Page

2.1 A biological neuron and an artificial neuron [37] 12

2.2 A simple neural network [6] 13

2.3 The graph of Sigmoid activation function [28] 15

2.4 The graph of Tanh activation function [28] 15

2.5 Gradient Descent [26] 18

2.6 A diagram of a simple RNN [14] 19

2.7 An unfolded RNN 20

2.8 A visualization of the vanishing gradient problem [20] 22

2.9 The gates of a LSTM cell [30] 24

2.10 LSTM block [21] 24

3.1 The sum of two simple waves with different frequencies in time domain and in frequency domain 29

3.2 Left: Sensor layout. Right: EPOC+ sensor and its accessories [38] . . . 29

3.3 A participant in the experiment looking at a product image while wearing EPOC+ [38]. 30

3.4 Images introduced to the participants [38] 31

4.1 Brain wave sample signals captured by the BCI sensors 33

4.2 Sample normalized brain wave component 'Delta' 35

4.3 Brain wave signals pre-processing 35

4.4 A sample 'Like' signal FFT spectrum 37

4.5 Brain wave components from a 'Like' signal 38

4.6 The LSTM block [21] 39

5.1 Means and standard deviation distributions for the 'Delta' brain wave component 43

5.2 Means and standard deviation distributions for the 'Theta' brain wave component 44

5.3 Means and standard deviation distributions for the 'Alpha' brain wave component 45

5.4 Means and standard deviation distributions for the 'Beta' brain wave component 46

5.5 Means and standard deviation distributions for the 'Gamma' brain wave component 47

5.6 Means vs standard deviations for the 'Delta' brain wave component . . 48

5.7 Means vs standard deviations for the 'Theta' brain wave component . . 49

5.8 Means vs standard deviations for the 'Alpha' brain wave component . . 49

5.9 Means vs standard deviations for the 'Beta' brain wave component . . 50

5.10 Means vs standard deviations for the 'Gamma' brain wave component . 50

5.11 Classification accuracies using the 'Delta' brain wave component . . . 52

5.12 Classification accuracies using the 'Theta' brain wave component . . . 52

5.13 Classification accuracies using the 'Alpha' brain wave component . . . 53

5.14 Classification accuracies using the 'Beta' brain wave component 53

5.15 Classification accuracies using the 'Gamma' brain wave component . . 54

5.16 The training process of an LSTM model 56

5.17 Classification accuracies using all brain wave components 57

5.18 Classification accuracies using four brain wave components 57

5.19 Classification accuracies using three brain wave components 58

5.20 Classification accuracies using three brain wave components and the first part of the signal [1,255] 58

5.21 Classification accuracies using three brain wave components and the middle part of the signal [125,425] 59

5.22 Classification accuracies using three brain wave components and the last part of the signal [255,512] 59

CHAPTER 1

INTRODUCTION

A lot of money are used on marketing campaigns, especially when introducing new products in the market. Many of these campaigns fail to seize the consumer's attention or stick in their memory. Some old marketing techniques like surveys, focus groups, or face-to-face interviews may not be effective because:

- They do not take into consideration the subconscious side of decision making.
- Other factors like time or peer pressure can influence how consumers feel about a particular product.
- The wording of questions can encourage the consumers towards the answer the marketers would like to hear.

A new revolutionary marketing form that overcomes the above-listed issues is neuromarketing. It is considered revolutionary because it combines marketing with neuroscience. What makes neuromarketing valuable is that it can assess information beyond the consciousness level to understand and analyze consumer behavior through neural activity. Some of the different methodologies used in the neuromarketing field are:

- Magnetoencephalography (MEG), is a brain imaging methodology that explores and registers the brain's magnetic activity.
- Functional magnetic resonancen imaging (fMRI), is a technique used to show what part of the brain gets activated during a specific mental activity by noticing the changes in blood oxygenation.
- Electroencephalogram (EEG), is a method that records the electrical activity of brain cell groups.

- Eye-tracking (ET), is a method that captures eye position and eye movement during a particular activity.

During the last decade, neuromarketing has been grabbing the attention of many researchers. Montague conducted the first neuromarketing experiment in 2003 [24]. Some individuals were asked to drink Coca-Cola or Pepsi while an fMRI was used to scan their brains. This study could not reveal how the human brain deals with brand choice, but it showed that if the individuals knew or did not know the brand they were consuming, different parts of their brain lit up. Later, Ambler [3] showed that there is a correlation between shopping decisions and brain imaging. This experiment had 18 subjects that looked at three product images of different brands at a time and were asked what brand they preferred. Baldo in 2015 [4] proposed an approach based on EEG signals to forecast product performance in the footwear industry. The EEG signals of 40 participants were recorded while looking at different shoe images on a screen. The participants were asked whether they would buy the shoes or not and fill a report on how much they liked a product on a scale of 1-5. This study revealed that the prediction accuracy using brain data was 20% higher than using self-report-based methods. Murugappan [27], in 2014 proposed a neuromarketing system for predicting the most preferred automobile brand out of four brands Toyota, Audi, Proton, and Suzuki, in Malaysia. The EEG signals of 12 individuals were captured while they looked at videos of four products of each brand, and later, they were asked if they liked or disliked the product and what emotions they felt watching a video of each product. The results suggested that the Toyota brand was highly preferred. Farashi [12], in 2019 has used the power of the EEG data to find the most critical brain regions for differentiating preferences and predicting decision-making for different mobile phone brands. The experiment results gave 87% accuracy for predicting consumer's decision-making and 63% accuracy for distinguishing between "Like" and "Dislike."

1.1 OVERVIEW

In this work, data from the EEG signals of 25 individuals while they looked at different product images was analyzed in order to build a flexible classification model able to distinguish a consumer preference in terms of "Like" and "Dislike" based on their brain waves. The data was captured using a Brain-Computer Interface (BCI) device that has 14 channels located in AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4. The individuals wore the BCI device while looking at various product images for 4 seconds each, and then they gave their preference regarding the images [38]. In [32], Fast Fourier Transform was used to get the five frequency bands (Alpha, Beta, Gamma, Theta, Delta) from EEG signals. We have scaled and normalized the data in our work and have used an LSTM based model for classifying brainwaves in a binary classification "Like" or "Dislike." Our model intends to overcome the shortcomings of the two analyses mentioned above. Even though the model in [32] reached a high accuracy, it can be considered impractical because the brainwaves of all individuals are used for the model training. We aimed to overcome this limitation by creating a classification model that groups the brain waves per human subject. The model takes brain waves from 24 subjects for the training part and tests to predict the preference of the 25th subject that was previously unknown to the model. In [38] analyses, all brain wave components are considered on various models; however, our experiment showed that combining all five brain wave components does not necessarily give higher results for classification.

A similar experiment was conducted with different data in [3]. The method used was fMRI and showed that the first part of a brain signal is related to problem-recognition and the other part is related to decision-making. Considering that brain waves are expected to "encode" more information than "Like" and "Dislike," our model can analyze only a fraction of the brainwave signal. From our experiment, this method produces better classification results.

CHAPTER 2

NEURAL NETWORKS

2.1 Introduction to Neural Networks

There are billions of neurons in the human brain. The neuron is a nerve cell, and it is the most crucial component of the nervous system. Neurons are responsible for capturing information and signals and transmitting this information from one neuron to another until it arrives at a specific part of the brain. Before transmitting the information, the neuron's role is to process it and decide if it should pass beyond this point. In 1943, Warren S. McCulloch and Walter Pitts presented for the first time the idea of an artificial neuron [25]. It took years for their work to be recognized; however, their development inspired other researchers to develop the primitive version of neural networks. Warren S. McCulloch and Walter Pitts paved the way for the sophisticated and complex neural network models of today's world. In 1958, Frank Rosenblatt proposed a binary classification algorithm called Perception, a mimic of the biological neuron [33].

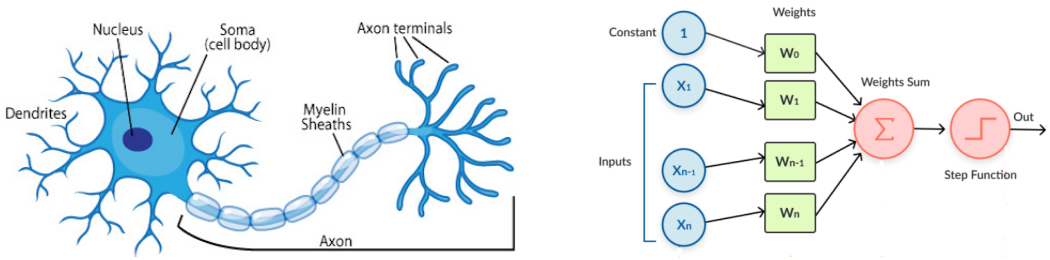

Figure 2.1: A biological neuron and an artificial neuron [37]

The neurons in Figure 2.1 function in a similar way. In the biological neuron, the information flows through the dendrites; it is processed in the nucleus, flows out by the axon, and is transmitted to the other neurons by synapses. Just like in the biological neuron, in its mathematical representation, inputs represent the dendrites. Inputs can be what we feel, see, touch. Weights can be considered like synapses. The application of the ac-

tivation function to the weighted sum is the step where the information gets processed. Then it flows out as an output signal. This output can be binary, categorical, or continuous. Multiple perceptrons stacked in several layers are called multilayer perceptrons [2]. Multilayer perceptions that only have one hidden layer are called "vanilla" neural networks. A vanilla neural network is a feed-forward neural network. In feed-forward neural networks, the inputs X_1,..,X_n are independent, meaning that they do not share any knowledge. Each input moves only in one direction, and it associates to a single output. This type of neural network is used when the output does not need to know any particular information from the preceding inputs. Feed-forward neural networks are used in different science fields, for example, in pattern recognition, and pattern classification, signal processing, image processing etc [15].

2.2 A Simple Neural Network

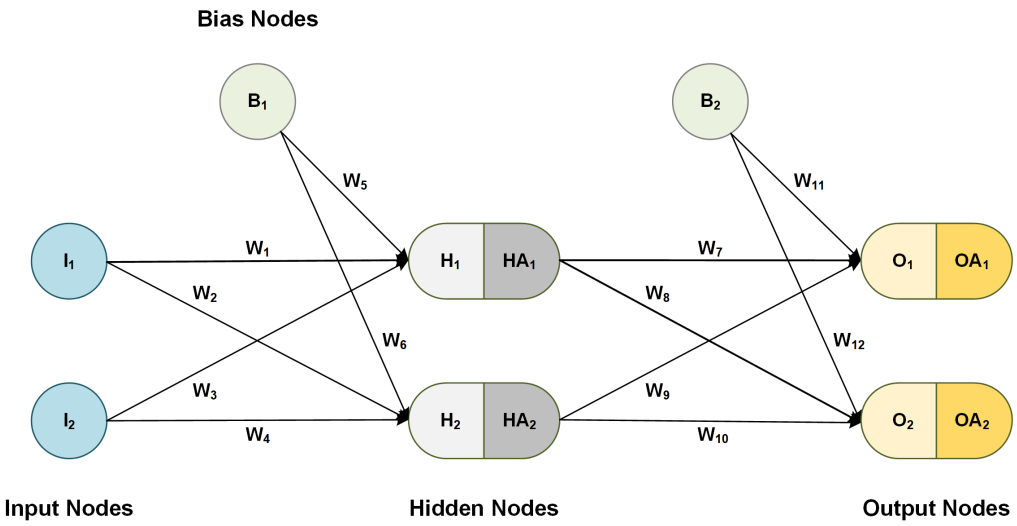

Figure 2.2: A simple neural network [6]

The first layer in the simple neural network in Figure 2.2 represents the input layer that consists of two inputs, the second one represents the hidden layer, and the last layer

represents the possible outcomes that in the above figure consists of two outputs. Each input layer connects to a hidden layer, and each hidden layer connects to the output layer. The NN in Figure 2.2 has only a hidden layer for demonstration, but usually, NN are more complex and have multiple hidden layers. In this case, the first hidden layer's output is considered an input for the second one and so on. The connections of the layers are associated with weights. Weights reflect the amount of input that should be considered. High weight values show the high significance of the corresponding input in the result. Each hidden node connects to a bias B_1, and each output is connected to another bias B_2. These connections also associate with weights. Biases B_1 and B_2 are vectors that are different for each layer, which add a change to the value of the output O [4]. A bias can be considered as an intercept term. If the activation function in the hidden neuron were linear, the bias would be the intercept [6]. The hidden layer takes in the sum of the weighted inputs and produces an output after applying an activation function to that sum. The role of Activation functions is to make neural networks non-linear. The reason why we need to use non-linearіy is because the real-world problems are very complex to be solved by linear regression. If we did not use non-linearity in Neural Network, despite the number of hidden layers that the Neural Network would have, it would behave like a single-layer perceptron. This means that we would get another linear function as the summation of all the hidden layers [7].

Activation functions are used to map the output values to a wanted range. Some of the most used activation functions are sigmoid functions, tanh functions, and softmax functions.

- Sigmoid activation function

  The Sigmoid function is a S-shaped graph that transforms values in the range (0,1) Figure 2.3. The Sigmoid function can be used for binary classification.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

- Tanh activation function

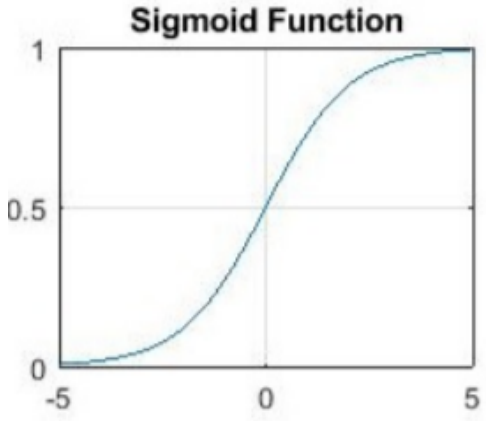

Figure 2.3: The graph of Sigmoid activation function [28]

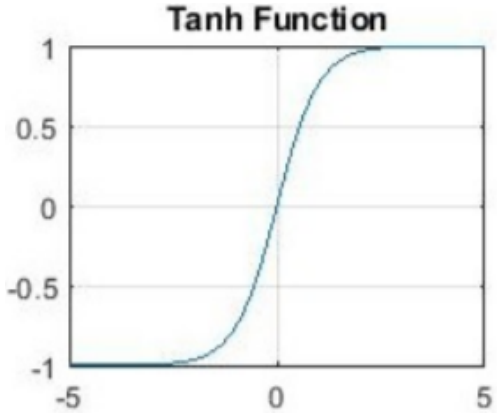

Figure 2.4: The graph of Tanh activation function [28]

Tanh is an S-shaped graph symmetric about the origin, which maps the values into the range (-1,1) as seen in Figure 2.4. Since the model is zero-centered, the outputs will be close to zero. Therefore, during optimization, their weight swings will be slight, and the model will learn faster. Tanh is a performant activation function that can be applied to the hidden state. The tanh formula is given in Equation 2.2.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.2}$$

where e is the Euler's number (e=2.718281828) and x is any real number.

- Rectified Linear Unit (ReLU) activation function.

  ReLu activation function is often applied to the hidden layers. It is faster learning, and it offers better performance than sigmoid and tanh activation functions [9].ReLU only activates the neuron if the value of the input is positive; this makes the function assure faster computations. ReLU can be one of the vanishing gradient problem solutions since it rectifies values close to zero to be zero.

$$f(x) = max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- Softmax activation function

  The Softmax activation function is applied to the output. It maps the values into the range (0,1). It is usually used for classification problems, and it provides the outputs as probabilities. An advantage of this activation function is that it can manage multiple classes. Suppose the likelihood of one class increases, the likelihood of the other class will decrease by the same amount since the probabilities have to sum up to one. The class that gives the best prediction is going to be the class that yields the highest probability. The mathematical formula for the Softmax activation function is given in Equation 2.3.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \text{ where i =1,...,k} \tag{2.3}$$

2.3 BACKPROPAGATION

One of the most popular questions, when introduced to NN, is adjusting the weights to have the model that we need. An answer to this question is by using Backpropagation.

Definition 1. [Backpropagation] A common method of training a neural network in which the initial system output is compared to the desired output, and the system is adjusted until the difference between the two is minimized.

As stated by Sathyanarayana [35]: "The goal of the backpropagation in neural networks is to find weights such that for every input vector in the training set, the neural network yields an output vector closely matching the targeted vector."

Initially, the weights are set as random. Then, it is checked how good the resulting neural network is by using a loss function. The Mean Squared Error (MSE) function is a commonly used loss function. It takes the average of the squared difference between the actual outputs and the target outputs. It would be ideal if the actual outputs would be equal to the target ones and the loss would be zero, but since this case is not that common, we aim to find an optimal solution by minimizing the loss function by improving the weights and the biases. If the output is linear, the optimal solution will be given by Equation 2.4.

$$Loss = min \quad \frac{1}{L}\sum_{i=1}^{L}(y_i - o_i)^2 \tag{2.4}$$

Where y_i is the actual output, o_i is the target output, and L is the number of samples in the training data.

If the output is binary, $y_i \in \{0, 1\}$, the optimal solution will be given by Equation (2.5). In this case, the objective is to minimize entropy, which is a measure for the disorder.

$$Loss = min \sum((y_i - 1) * log(1 - o_i) - y_i * log(o_i)) \tag{2.5}$$

A weight/bias will be changed by an amount proportional to the partial derivative of the loss function with respect to that weight/ bias using the gradient descend concept given by Equation 2.6 and demonstrated in Figure 2.5.

$$w^{t+1} = w^t - \eta * \frac{\partial\, Loss}{\partial\, w^t} \tag{2.6}$$

where w^{t+1} is the updated weight and η is the learning rate.

The same process repeats multiple times until we arrive at a desired level of accuracy. The partial derivatives, called Gradients, measure how the accuracy changes for small changes in weights and biases [35].

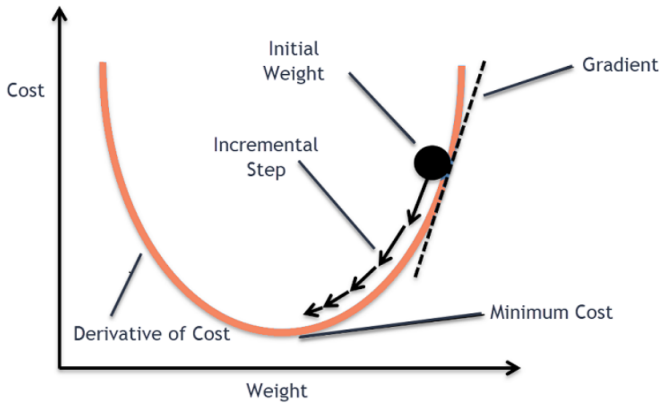

Figure 2.5: Gradient Descent [26]

In the graph in Figure 2.5, the horizontal axis represents the weights and the vertical axis the value of the loss or, said differently, the cost function. We start with an initial weight and find the corresponding point of this weight on the loss function. Since the goal is to adjust the weights to minimize the loss, we have to get closer to the local minimum in the function by taking the partial derivative of the loss function with respect to the weight. If the derivative is positive, the point in the loss function has to move towards the left, closer to the minimum. If it is negative, it has to move in the opposite direction.

2.4 Recurrent Neural Networks

In the simple feed-forward NN, the predictions only depend on the current input. What if we are using sequential data where the current inputs are dependent on the previous input? In this case, to understand the data and make predictions, we need information from the current input and the previous ones. This matter was solved in 1997 when Hochreiter and Schmidhuber presented the idea of recurrent neural networks (RNN)[5]. Contrary to the standard feed-forward neural networks, RNN has a cyclical hidden state referred to as the RNN's memory. This feature of RNN remembers information from the previous steps, and it gets modified every time RNN reads a new input. In Figure 2.6 an input is fed in

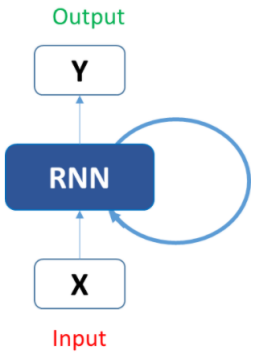

Figure 2.6: A diagram of a simple RNN [14]

the RNN, the hidden state is updated, and an output is produced. To better understand how RNN works, the simple RNN is unfolded in Figure 2.7. The unfolded RNN consists of copies of the same diagram. Each of them contains an input that can be a sequence, a hidden state, and an output that can classify or apply regression to the inputs. These diagrams represent the batches. The data used for training is separated into batches, and RNN trains one batch at a time. $X^{(t)}$ and $O^{(t)}$ stand for the input and output in time *t* or, said differently, the current input and output. The hidden state in time t is represented by $h^{(t)}$. The matrices U, W, and V, represent the weights of the RNN. As illustrated in Figure 2.7, matrix U represents the weights that transform the input to the hidden state. Matrix W represents the weights that transform the hidden state at time *t-1* to the hidden state at time *t*, and matrix V represents the weights that transform the hidden state to the output. At each cycle or time step happens the following: A vector x(t) is taken in as an input, and the weighted input is sent to the hidden state. In the hidden state at time *t* a memory state $h^{(t)}$ is produced that is an output of both the current input $X^{(t)}$ and the previous memory state $h^{(t-1)}$. The activation function that is mainly applied in the hidden state in RNN is the tanh function. The formula for calculating $h^{(t)}$ is given in Equation 2.7. The output $O^{(t)}$ is produced by applying the softmax activation function to the weighted $h^{(t)}$ like shown in Equation 2.8. The same function and the same set of parameters are used on the inputs and

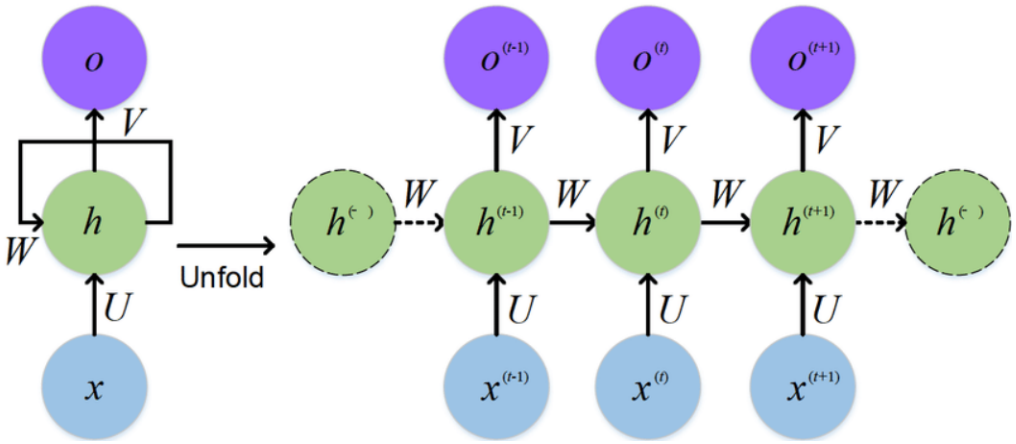

Figure 2.7: An unfolded RNN

hidden layers at every time step of the calculations to produce the output; hence RNN is less complex in terms of parameters than the other NN. The following formulas are used for calculating $h^{(t)}$ and $O^{(t)}$:

$$h^{(t)} = tanh(W * h^{(t-1)} + U * X^{(t)} + b^{(h)}) \tag{2.7}$$

$$o^{(t)} = softmax(V * h^{(t)} + b^{(o)}) \tag{2.8}$$

2.4.1 BackPropagation Through Time

In RNN, the process of backpropagation gets more complicated. Unlike in simple neural networks, in RNN, the error is calculated at each time step, and the total error is the sum of all these errors.

$$L = \sum_{t=1}^{n} L_t \tag{2.9}$$

Therefore, the backpropagation process happens at each of the individual time steps and across all of them until the start of the sequence. This is the reason why the backpropagation process in RNN is called backpropagation through time (BPTT).

Let us take into consideration the Equations 2.7 and 2.8 and the Unfolded RNN in Figure 2.7. As mentioned above, the weight matrices do not change for different time steps. This makes the BPTT more difficult than the backpropagation in Simple NN because every $h^{(i)}$ and $O^{(i)}$, where i represents the different time steps, is dependent on the same weight matrix. To update the weights, we have to find the partial derivatives of the loss function with respect to the weights.

$$\frac{\partial L}{\partial W} = \sum_{i=1}^{n} \frac{\partial L^{(i)}}{\partial W} \tag{2.10}$$

Let's find the partial derivative of the loss function at a time t:

$$\frac{\partial L^{(t)}}{\partial W} = \frac{\partial L^{(t)}}{\partial O^{(t)}} * \frac{\partial O^{(t)}}{\partial h^{(t)}} * \frac{\partial h^{(t)}}{\partial W} \tag{2.11}$$

For simplicity reasons, let us consider the output of the hidden state before applying the tanh activation function in Equation 2.7: $(W * h^{(t-1)} + U * X^{(t)}) = z$.

$$\frac{\partial h^{(t)}}{\partial W} = tanh'(z) * \left(h^{(t-1)} + W * \frac{\partial h^{(t-1)}}{\partial W} \right) \tag{2.12}$$

As can be seen in the above equation (2.12) both $h^{(t)}$ and $h^{(t-1)}$ depend on the same W. Hence:

$$\frac{\partial h(t)}{\partial W} = tanh'(z) * \left[h^{(t-1)} + W * tanh'(z) * \left(h^{(t-2)} + W * \frac{\partial h^{(t-2)}}{\partial W} \right) \right] \tag{2.13}$$

This pattern will be used to expand the partial derivatives until we arrive at time : t=1.

$$\frac{\partial h^{(t)}}{\partial W} = tanh'(z) * \left\{ h^{(t-1)} + W * tanh'(z) * \left[h^{(t-2)} + W * tanh'(z) \left(h^{(t-3)} + ... + W * \frac{\partial h^{(1)}}{\partial W} \right) \right] \right\}$$

To find the partial derivative (Equation 2.11), we have to plug in the Equation 2.4.1 in the place of the partial derivative of the hidden state at time t with respect to W.

2.4.2 VANISHING AND EXPLODING GRADIENT PROBLEMS

Two of the most significant issues when using BPTT across many time steps are the vanishing and the exploding gradient problems.

- Vanishing gradient problem

  Due to the usage of the Chain Rule in calculations, the gradients that come from the initial layers have to go through continuous multiplications. If they have small values (<1), these long-term components will exponentially fast to norm 0. During backpropagation, the weights get updated by an amount proportional to the gradient. If this amount is closer to 0, the updated weight will be very close to the previous one and far from the optimal weight. This issue weakens the model's ability to learn. Figure 2.8 is a visualization of the vanishing gradient problem. If the weight along the recurrent edge is less than one, the contribution of the input at the first time step to the output at the final time step will decrease exponentially fast as a function of the length of the time interval in between.

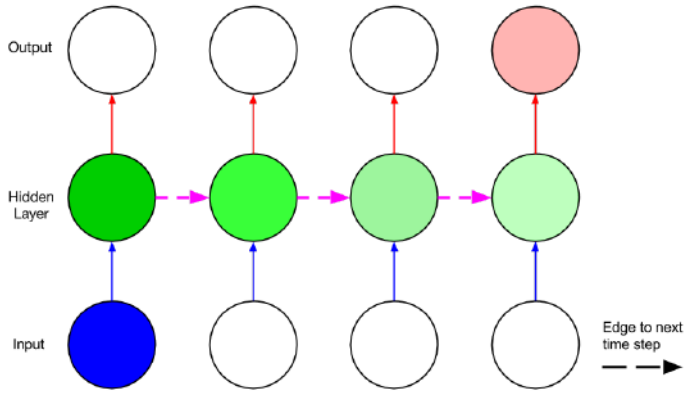

Figure 2.8: A visualization of the vanishing gradient problem [20] .

- Exploding gradient problem

  The exploding gradient problem is the opposite of the vanishing gradient problem. In

this case, the gradients have high values (>1); hence, there will be a large increase in the norm of the gradients during the training. The long term components will grow exponentially more than the short term components till they explode and crush the model [29].

2.5 LONG SHORT-TERM MEMORY (LSTM)

A solution that helps overcome the vanishing gradient issue was introduced by Hochreiter and Schmidhuber in 1997 [17]. Hochreiter and Schmidhuber proposed a new type of RNN called Long Short-Term Memory (LSTM), the architecture of which makes sure that the gradient problems will not occur in networks with many layers. LSTM is capable of learning long-distance dependencies and deals better than RNN in the prediction of complex tasks. Due to this ability, LSTM is now used widely in speech recognition [13], language modeling [39], translating [22], audio analyses [23] etc. Over the years, different improvements are made to the architectures of LSTM in different studies.

The main idea behind LSTM architecture is the concept of a gated cell. The architecture of this cell makes it possible for LSTM to deal with long-term dependencies by regulating the information that flows in and out of the cell, similar to the human brain. As it can be seen in Figure 2.9, there is a cell state and three gates inside of an LSTM cell. Each of the gates contains a sigmoid function, and its purpose is to add or remove information from the cell state. The LSTM cell's gates are the following:

- Forget gate (f_t). The forget gate is called this because it decides what information and how much information is irrelevant and must not be kept. The sigmoid function takes in information from the previous hidden layer (h_{t-1}) and information from the current input (x_t) and gives (f_t) as an output. The value of (f_t) ranges between 0 and 1. If f_t is 0, it means that the past information is all forgotten; if (f_t) is 1, it means that all the information from the past is relevant, and so it needs to be kept. The

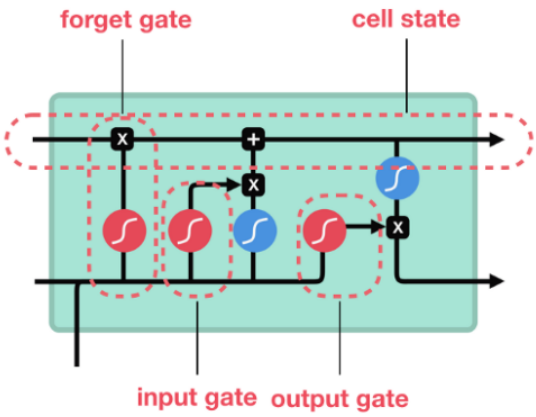

Figure 2.9: The gates of a LSTM cell [30]

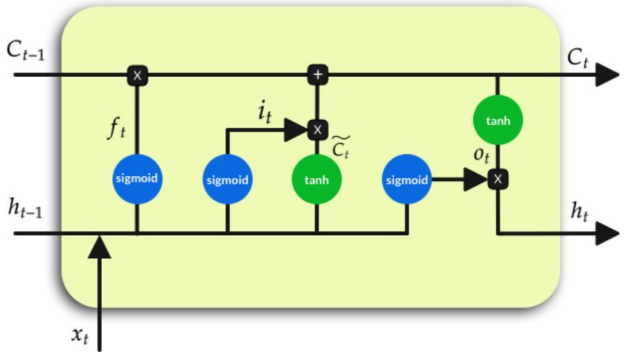

Figure 2.10: LSTM block [21]

mathematical representation of the forget gate is given in Equation 2.14:

$$f_t = \sigma(W_{xf}X_t + W_{hf}h_{t-1} + b_f) \tag{2.14}$$

where, W_{xf} and W_{hf} represent the weights and b_f the bias.

- Input gate (i_t). The input gate determines what new information is relevant in the current time step and needs to be passed through. It decides the updates that should be stored in the cell state. Firstly, the sigmoid function determines the values that will be updated by taking in the previous hidden state (h_{t-1}) and the current input

(X_t) and by giving the output (i_t) between 0 and 1.

$$i_t = \sigma(W_{xi}X_t + W_{hi}h_{t-1} + b_i) \tag{2.15}$$

Then, the candidate gate ($\tilde{C}_t$) is calculated using tanh as the activation function. ($\tilde{C}_t$) represents the vector created with the new candidate values to add to the cell state.

$$\tilde{C}_t = tanh(W_{xc}X_t + W_{hc}h_{t-1} + b_c) \tag{2.16}$$

The forget gate and the input gate are used to update the cell state (C_t). By multiplying component-wise, the forget gate and the previous cell state, the information that will be forgotten from the previous cell state is decided. By multiplying component-wise the input gate and the candidate gate, it is decided what new information needs to be considered from the current time step.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{2.17}$$

- Output gate (o_t). The output gate decides what output to generate from the current cell state. By doing so, it determines the next hidden state. The values of the previous hidden state (h_{t-1}) and the current input (X_t) are passed through a sigmoid function. In the meantime, the current cell state is passed through a tanh function. The new hidden state at time t is produced by multiplying component-wise the output generated in Equation 2.18 and the tanh output of the cell state.

$$o_t = \sigma(W_{xo}X_t + W_{ho}h_{t-1} + b_o) \tag{2.18}$$

$$h_t = o_t \odot tanh(C_t) \tag{2.19}$$

CHAPTER 3

DATA DESCRIPTION

3.1 NEUROMARKETING

Neuromarketing as a field of study is defined as the application of neuroscientific methods to analyze and understand human behavior in relation to markets and marketing exchanges [19]. It is formed by a group of techniques that seek to identify the brain areas activated during a marketing stimulus and the cognitive processes that occur in those areas, as well as the various related biological markers [11]. Neuromarketing is considered a revolutionary form of marketing because of its ability to access and assess information beyond the level of human consciousness. It is also a useful tool to help marketers understand how consumers make choices during the purchase process.

3.2 ELECTROENCEPHALOGRAM (EEG)

Electroencephalogram (EEG) is the recording of the spontaneous electrical activity of brain cell groups in the cerebral cortex, or the scalp surface [34]. The brain controls all physical and mental processes; hence the brain waves contain much physical, psychological, and pathological information. Studying and analyzing EEG signals has played an essential role in diagnosing epilepsy, seizure disorders, and sleep disorders. Nowadays, the fields where EEG signal analyses are being used are getting wider. A promising field that has attracted many researchers' attention is the use of EEG signals in neuromarketing. Researchers are using brain waves in order to study aspects of marketing by analyzing the consumer behavior, and advertisement phenomenon [38].

The EEG signals are captured in the time domain by placing EEG electrodes at various positions on the scalp. These signals are random in nature, and it is difficult to obtain information by looking at them in the time domain; therefore, signal processing techniques

are used to extract the features needed to analyze them. For feature extraction, EEG signals are usually decomposed into five distinct frequency bands that indicate different conditions:

- Delta (1 Hz - 4 Hz). Delta waves are the slowest waves in the human brain. They are associated with sleep and deep levels of relaxation and are most often found in infants. Delta waves are very noticeable in brain injuries and inability to think cases [1].

- Theta (4 Hz - 7 Hz). Theta waves are associated with the working memory [31]. These waves also occur during movement [16], and during cognitive tasks [18].

- Alpha (8 Hz - 13 Hz). Alpha waves refer to activities that are associated with relaxation [5]. High levels of Alpha mean deep-relaxation or problems in concentration, while low levels of Alpha connect to stress or anxiety symptoms.

- Beta (13 Hz - 22 Hz). Beta waves are associated with active and logical thinking. Beta's normal levels relate to problem-solving, focus, and memory, while higher levels of Beta relate to the inability to relax and depression [2].

- Gamma (32 Hz - 100 Hz). Gamma waves are frequently analysed in cognitive activities related to perception, attention, and memory [36] [10].

3.3 Fourier Transformation

Fourier Transformation is a famous mathematics function. In the 1800s, Joseph Fourier discovered that every function could be expressed as a sum of simple sine and cosine functions. Later, his discovery inspired other scientists who used Fourier's findings in different mathematics and engineering areas. In 1965, the mathematicians J. W. Cooley and J. W. Tukey invented the Fast Fourier Transform, an algorithm for Fourier Transform that diminished the computational time [9]. Nowadays, Fourier Transform is widely used in

telecommunication, hearing devices systems, medicine, optics, voice recognition, image processing.

The idea behind Fourier Transform that makes it so useful in the fields mentioned above is that it can transform a signal f(t) in the time domain to a signal in the frequency domain F(s). The Fourier Transform of a function of time is defined by:

$$F(s) = \int_{-\infty}^{\infty} f(t) e^{-2i\pi st} dt \tag{3.1}$$

The outcome is F(s), a function of the frequency s. F(s) gives how much power f(t) contains at the frequency s. Also, by using the Inverse Fourier Transform, f(t) can be obtained from F(s):

$$f(t) = \int_{-\infty}^{\infty} F(s) e^{2i\pi st} ds \tag{3.2}$$

Fourier Transform is a signal processing method that can be used to analyze EEG signals. In order to use Fourier Transform in EEG signals, the signal is assumed to be stationary, meaning that the mean and the variance of the signal do not depend on the time component. By applying Fourier transform to a brain wave, we can extract the EEG frequency components. Analyzing a signal in the frequency domain is useful when it is distinguished by its frequency, not the time or space.

A new complex wave can be obtained by adding simple sine waves that can have different amplitudes and frequencies. The new complex wave will be decomposed to the previous simple waves if a Fourier Transformation is applied to it. The wave decomposition allows us to distinguish the amplitude and frequency components that were not obvious before in the complex wave. Figure 3.1 shows that it is easier to discern how many components and what components are used to create the complex wave by looking at the wave in the frequency domain rather than looking at the wave in the time domain.

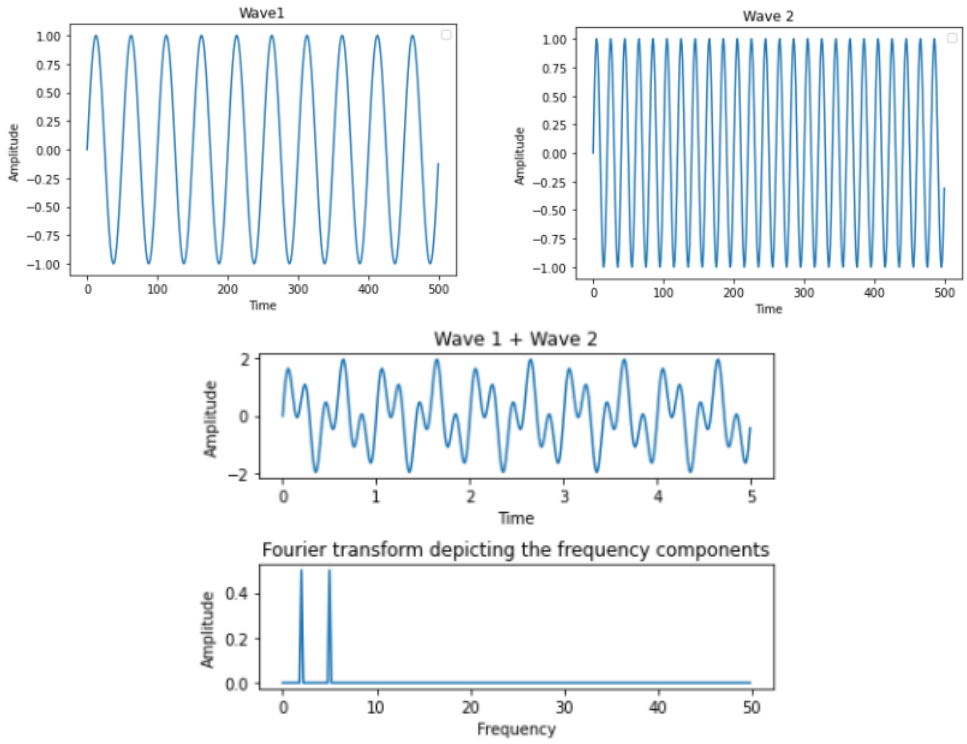

Figure 3.1: The sum of two simple waves with different frequencies in time domain and in frequency domain

3.4 THE EXPERIMENT

The experiment conducted has its focus on neuromarketing [38]. Twenty-five individuals whose ages variate from 18 to 38 years old have participated. A neuro-signal data acquisition wireless device called Emotive EPOC+ has been placed on the participant's head like in Figure 3.3.

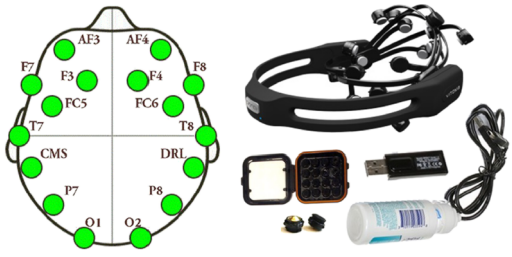

Figure 3.2: Left: Sensor layout. Right: EPOC+ sensor and its accessories [38]

Figure 3.3: A participant in the experiment looking at a product image while wearing EPOC+ [38].

The device has 14 channels located in AF3, F7, F3, FC5, T7, P7, O1, O2, P8, T8, FC6, F4, F8, AF4. EPOC+ is used to capture the participant's EEG signals while looking at a variety of product images. Then, the participants have given their preferences in terms of likes and dislikes for each image shown on the screen. As it can be seen in Figure 3.4, the images shown in this experiment are those of 14 products with three variations each, in total 42 images. Each of these 42 different images has been shown to the participants for 4 seconds.

Figure 3.4: Images introduced to the participants [38]

CHAPTER 4

THE RNN-BASED MODEL FOR THE BINARY CLASSIFICATION OF THE BRAIN WAVE SIGNALS

This work's main goal is to create a flexible classification model capable of deciding human subjects' preferences (*like* or *dislike*) on visualizing advertisement images of various products. Our model relies on analyzing one or more brain wave components ('Delta,' 'Theta,' 'Alpha,' 'Beta,' and 'Gamma') collected through a Brain-Computer Interface (BCI) device while 25 human subjects are exposed to the above mentioned visual stimuli [38].

The model we propose aims to overcome some of the shortcomings of the same dataset's previous analysis. The work in [32] shows that the data can be classified using Artificial Neural Networks (ANN) models with high accuracy. However, it considers the collection of brain wave data as a whole, without grouping it per human subjects. In the current work, we overcome this disadvantage and propose a classification model that corresponds to a practical situation where the model is created based on the data collected from volunteers and used to predict preferences for other subjects previously unknown to the model. The work in [38] performs the analysis of the same dataset (using various models) by considering all brain wave components. However, as our experimental results in Chapter 5 show, considering all components does not necessarily produce the best classification results. In our work, we overcome this disadvantage by proposing a flexible model capable of using one or more brain wave components. Moreover, as brain waves are expected to "encode" more than *like/dislike* information, it is expected that analyzing a fragment of the whole signal will produce better results than processing the whole sequence. Our model can accommodate partial signal analysis and the experimental results confirm that analyzing only a fraction of the signal does produce better classification accuracy.

This chapter is organized as follows. In Section 4.1 we describe the dataset used for

defining our classification model and for testing. The Recurrent Neural Network (RNN) model we construct in this work is introduced in Section 4.2.

4.1 Data aggregation and brain wave components extraction

4.1.1 Data preprocessing and notations

The brain wave signals were collected [38] from 25 human subjects, visualizing 42 advertising images, using 14 BCI sensors. However, from some of the subjects, some captures (while visualizing some images) were affected by errors. Consequently, out of 14,700 possible signals (from all subjects, images, and sensors), only 14,630 signals were collected. Figure 4.1 shows two sample signals, for 'Like' and 'Dislike' preferences, respectively. Each signal collected by a sensor was stored as 512 samples (that is, each signal was represented by a sequence of 512 numbers).

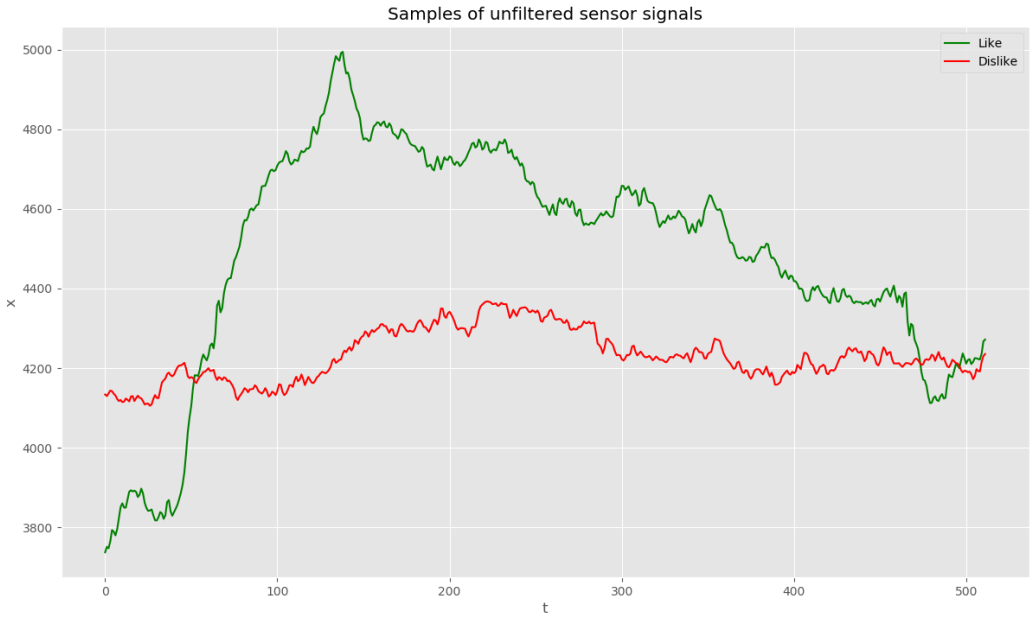

Figure 4.1: Brain wave sample signals captured by the BCI sensors

The dataset $\mathcal{D}$ of signals for this work was obtained from the above-described data as follows:

- Each sensor signal was filtered using the Fast Fourier Transform (FFT). Subsequently, 5 components were created for each signal, boosting the size of the database to 73,150 entries (each entry being a sequence of 512 numbers).

- The data was subsequently aggregated by image ID and wave component. That is, out of 14 components of the same type ('delta', 'theta', etc.) produced by 14 sensors while a subject was exposed to one image, a single component of the respective type was created. Consequently, the dataset size was reduced to 5,225 entries (each entry being a sequence of 512 numbers, representing one brain wave component collected from all sensors while a subject was exposed to one image).

- Each entry of the dataset was subsequently normalized to the range $(0, 1)$. Figure 4.2 shows samples of the 'Delta' component for 'Like' and 'Dislike' preferences, respectively.

- The set of 5,225 signals obtained as described above represents the dataset $\mathcal{D}$ of signals for performing the analysis described in this work.

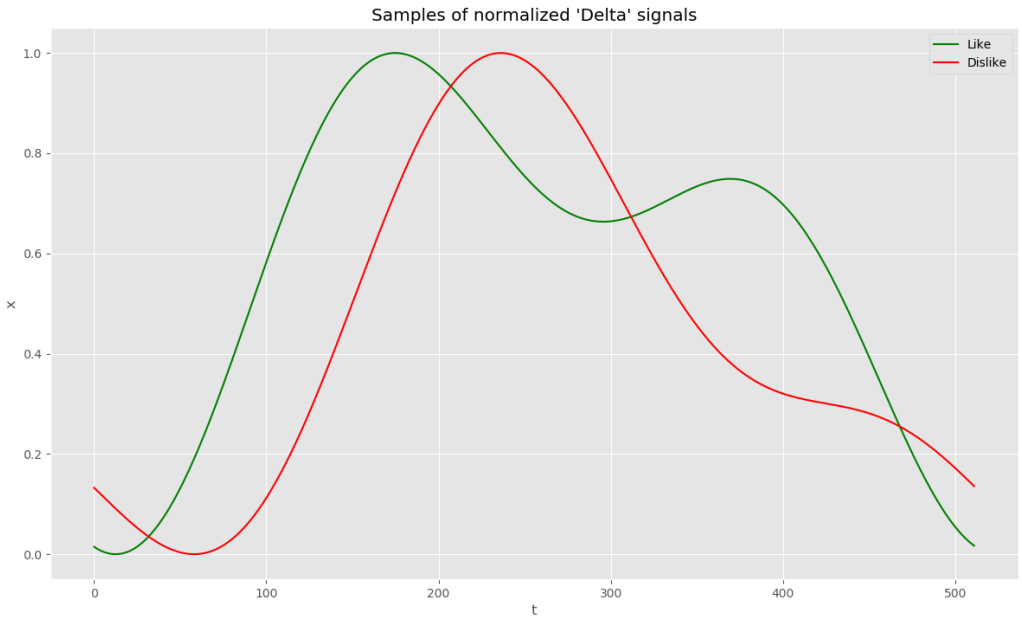

Figure 4.2: Sample normalized brain wave component 'Delta'

The whole data-preprocessing process is illustrated in Figure 4.3.

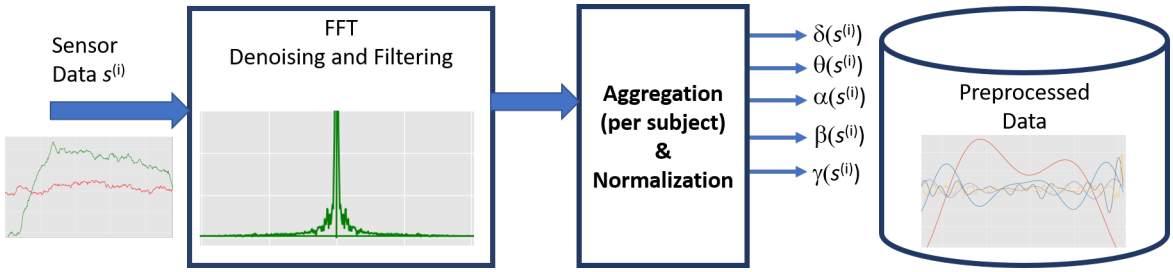

Figure 4.3: Brain wave signals pre-processing

For the rest of this thesis we will use the following notations:

- Each entry $x = x[k], k = 1, ..., 512$ in the dataset $\mathcal{D}$ is a sequence of 512 numbers in the range $(0, 1)$.

- We use indices s and i to identify the subject and the image, respectively, of each

sequence in $\mathcal{D}$. That is, x_{si} is an entry (sequence) in $\mathcal{D}$ corresponding to subject s and image i, with $s = 1, ..., 25$ and $i = 1, ..., 42$. (Note that an image is missing from a number of subject, making $i = 1, ..., 41$.)

- Each entry x has a type, $type(x) \in \{\delta, \theta, \alpha, \beta, \gamma\}$, corresponding to the brain wave component it represents. We will be using the notations δ_{si}, θ_{si}, etc. to denote the respective component for subject s and image i.

- Each entry x has a label and a class: $label(x) \in \{Like, Dislike\}$, $class(x) \in \{1, 0\}$, corresponding to the subject preference for the image corresponding to the given entry (signal).

By using these notations, we can therefore write that

$$\mathcal{D} = \{x_{si}[k] \mid x \in \{\delta, \theta, \alpha, \beta, \gamma\},\ s = 1, ..., 25,\ i = 1, ..., 42,\ k = 1, ..., 512\}$$

The FFT filtering of the original brain wave signals is described in the subsequent section.

4.1.2 FILTERING THE BRAIN WAVE COMPONENTS

We extract the brain wave components ($\delta, \theta, \alpha, \beta, \gamma$) using the Fast Fourier Transform (FFT) and Inverse Fourier Transform (IFT) as follows:

- The FFT is applied to each signal in the dataset $x_{si} \in \mathcal{D}$ (which is a sequence of 512 samples) to obtain the FFT coefficients of the signal:

$$X_{si}[n] = \sum_{k=0}^{511} x_{si}[k] e^{-j\frac{2\pi}{512}nk},\ n = 0, 1, ...$$

  Figure 4.4 shows the plotting of these coefficients (called the "signal spectrum") for a 'Like' brain wave signal like one in Figure 4.1. Each coefficient corresponds to a signal component of frequency $\frac{2\pi}{512}n$.

- The sequences of FFT coefficients corresponding to brain wave components frequency ranges are subsequently produced:

$$
\begin{aligned}
X_{si}^{(\delta)}[j] &= X_{si}[j], \text{ for } 1 \leq \frac{2\pi}{512} j < 4; \text{ else } X^{(\delta)}[j] = 0, \ j = 0, 1, ... \\
X_{si}^{(\theta)}[j] &= X_{si}[j], \text{ for } 4 \leq \frac{2\pi}{512} j < 8; \text{ else } X^{(\theta)}[j] = 0, \ j = 0, 1, ... \\
X_{si}^{(\alpha)}[j] &= X_{si}[j], \text{ for } 8 \leq \frac{2\pi}{512} j < 13; \text{ else } X^{(\alpha)}[j] = 0, \ j = 0, 1, ... \\
X_{si}^{(\beta)}[j] &= X_{si}[j], \text{ for } 13 \leq \frac{2\pi}{512} j < 32; \text{ else } X^{(\beta)}[j] = 0, \ j = 0, 1, ... \\
X_{si}^{(\gamma)}[j] &= X_{si}[j], \text{ for } 32 \leq \frac{2\pi}{512} j < 100; \text{ else } X^{(\gamma)}[j] = 0, \ j = 0, 1, ...
\end{aligned}
$$

  These coefficients are the FFT coefficients corresponding to the brain wave components $\delta, \theta, \alpha, \beta, \gamma$, respectively.

- The Inverse Fourier Transform (IFT) is then applied to each sequence of Fourier coefficients to create each brain wave component:

$$w_{si}[k] = Re\left(\frac{1}{512}\sum_{n=0}^{511} X^{(w)}[n] e^{j\frac{2\pi}{512}nk}\right)$$

  where $w \in \{\delta, \theta, \alpha, \beta, \gamma\}$

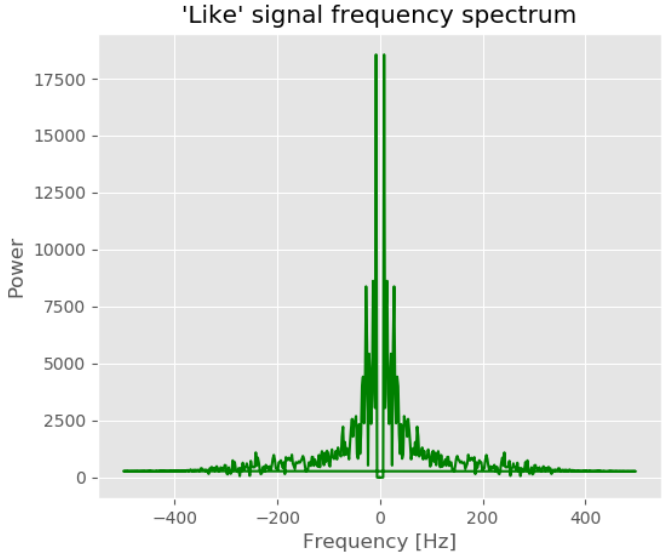

Figure 4.4: A sample 'Like' signal FFT spectrum

Sample signals of the extracted brain wave components using the steps described above are presented in Figure 4.5.

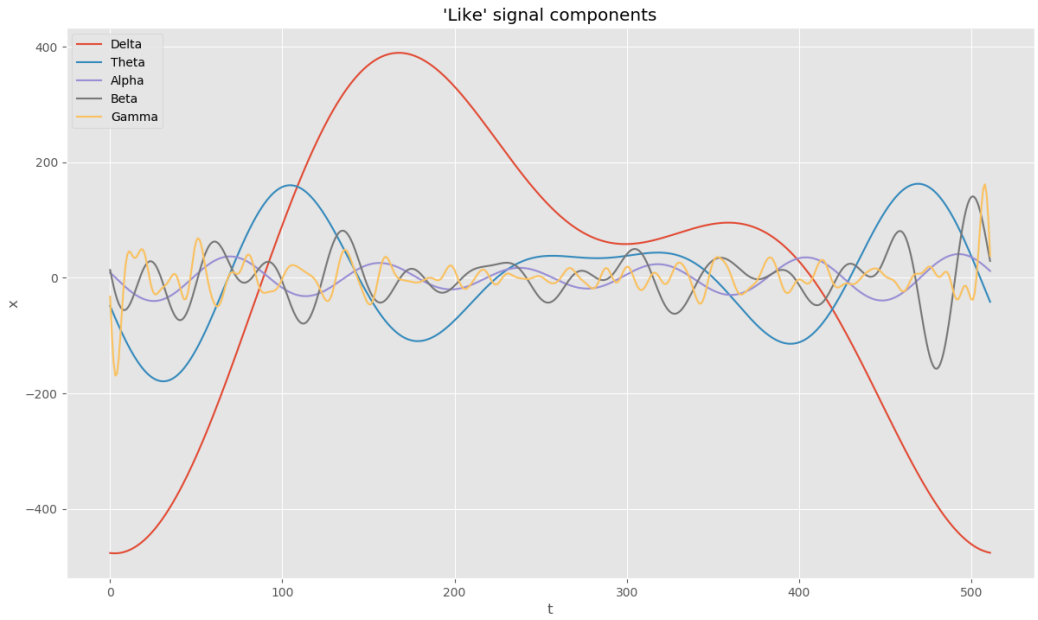

Figure 4.5: Brain wave components from a 'Like' signal

The Python code for the data pre-processing is given in Appendix C.

4.2 THE RNN-BASED CLASSIFICATION MODEL

Definition 2. A training set $\mathcal{T}_S \subset \mathcal{D}$ for some $S \in \{1, ..., 25\}$, is the set

$$\mathcal{T}_S = \{x_{si} \mid x \in P(\{\delta, \theta, \alpha, \beta, \gamma\}),\ s \neq S\}$$

A test set $\mathcal{S}_S \subset \mathcal{D}$ for some $S \in \{1, ..., 25\}$, is the set

$$\mathcal{S}_S = \{x_{Si} \mid x \in P(\{\delta, \theta, \alpha, \beta, \gamma\})\}$$

When the subject S is not important, we will simply use the notations $\mathcal{T}$ and $\mathcal{S}$, respectively.

Definition 3. [Classification Model] The brain wave binary classification model $\mathcal{M}$ on a dataset $\mathcal{D} = \mathcal{T} \cup \mathcal{S}$ is a mapping

$$\mathcal{M} : \mathbb{R}^{p \times 512} \rightarrow \{Like, Dislike\}$$

where $\mathcal{M}$ is subject of

$$\min \sum_{x \in \mathcal{T}, y = class(x)} (-y \log \mathcal{M}(x) - (1-y) \log(1 - \mathcal{M}(x)))$$

Definition 4. [Accuracy of Model] The classification accuracy of a model $\mathcal{M}$ on a dataset $\mathcal{D} = \mathcal{T} \cup \mathcal{S}$ is:

$$acc(\mathcal{M}) = \frac{\sum_{x \in \mathcal{S}} |class(x) - \mathcal{M}(x)|}{|\mathcal{S}|}$$

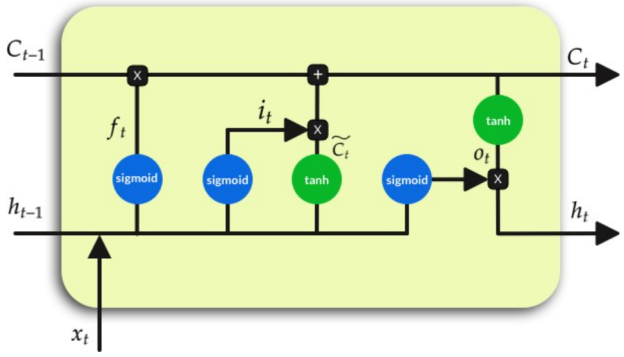

Figure 4.6: The LSTM block [21]

The model in Definition 3 is implemented using an LSTM variant (Figure 4.6) of a RNN (as described in Chapter 2, Figure 2.10).

The $\mathcal{M}(x_{si})$ function for an input $x_{si} \in \mathcal{D}$ computes the Like/Dislike decision recur-

sively as follows:

$$
\begin{aligned}
&time\ step\ 1: \\
&\mathbf{h_1} = ReLU(W_1\mathbf{x_{si}}[\mathbf{1}] + b_1) \\
&time\ step\ 2: \\
&\mathbf{h_2} = ReLU(W_1\mathbf{x_{si}}[\mathbf{2}] + Z\mathbf{h_1} + b_1) \\
&... \\
&time\ step\ 512: \\
&\mathbf{h_{512}} = ReLU(W_1\mathbf{x_{si}}[\mathbf{512}] + Z\mathbf{h_{511}} + b_1) \\
&\mathcal{M}(\mathbf{x_{si}}) = softmax(W_2 \cdot \mathbf{h_{512}} + b_2)
\end{aligned}
$$

where the hidden layer function h for the LSTM cell is computed as given by the equations (2.14)-(2.19) in Chapter 2.

After 512 time steps, the model computes a pair of probabilities, for Like and Dislike, respectively. The highest probability indicates the input signal classification decision.

The parameters of the LSTM models (number of hidden layers and nodes) are selected experimentally, and they are presented in Chapter 5 for each experimental result.

CHAPTER 5

EXPERIMENTAL RESULTS

We performed extensive experimental results using the model we introduced in Chapter 4 on the dataset described in Chapter 3, which was pre-processed as described in Chapter 4.

All our experimental results presented in this chapter were performed on a PC equipped with an Intel Core i7-4770 CPU @3.40GH. The complete Python code listings for preprocessing data and the experimental results are provided in the Appendix. Throughout this chapter we use the notations introduced in Section 4.1.

The rest of the chapter is organized as follows. We present some dataset statistics in Section 5.1. Then we present the brain waves binary classification results in Sections 5.2 and 5.3, for the single and multiple brain wave components, respectively.

5.1 DATA STATISTICS

We have performed some basic statistics (means and standard deviations) for each brain wave component. As indicated in [32], these statics may serve as indicators of the components that are likely to provide good classification information. We plotted the distributions of these statistics separately (each can be considered as a single data feature), and together (two data features). The results are presented below. However, we could not find any indicators that these data features can be used for components selection in our classification model.

The Python code for the results presented in this section is given in Appendix B.

5.1.1 Distributions of means and standard deviations for each brain wave component

The distributions of means and standard deviations for each brain wave component (Delta, Theta, Alpha, Beta, and Gamma) are presented in Figures 5.1 – 5.5, respectively. All these figures show consistent overlapping between distributions of means and standard deviations for both Like and Dislike signals, for each brain wave. Therefore, based on a single signal feature (mean or standard deviation), we cannot determine which of the brain waves are good candidates for performing classification.

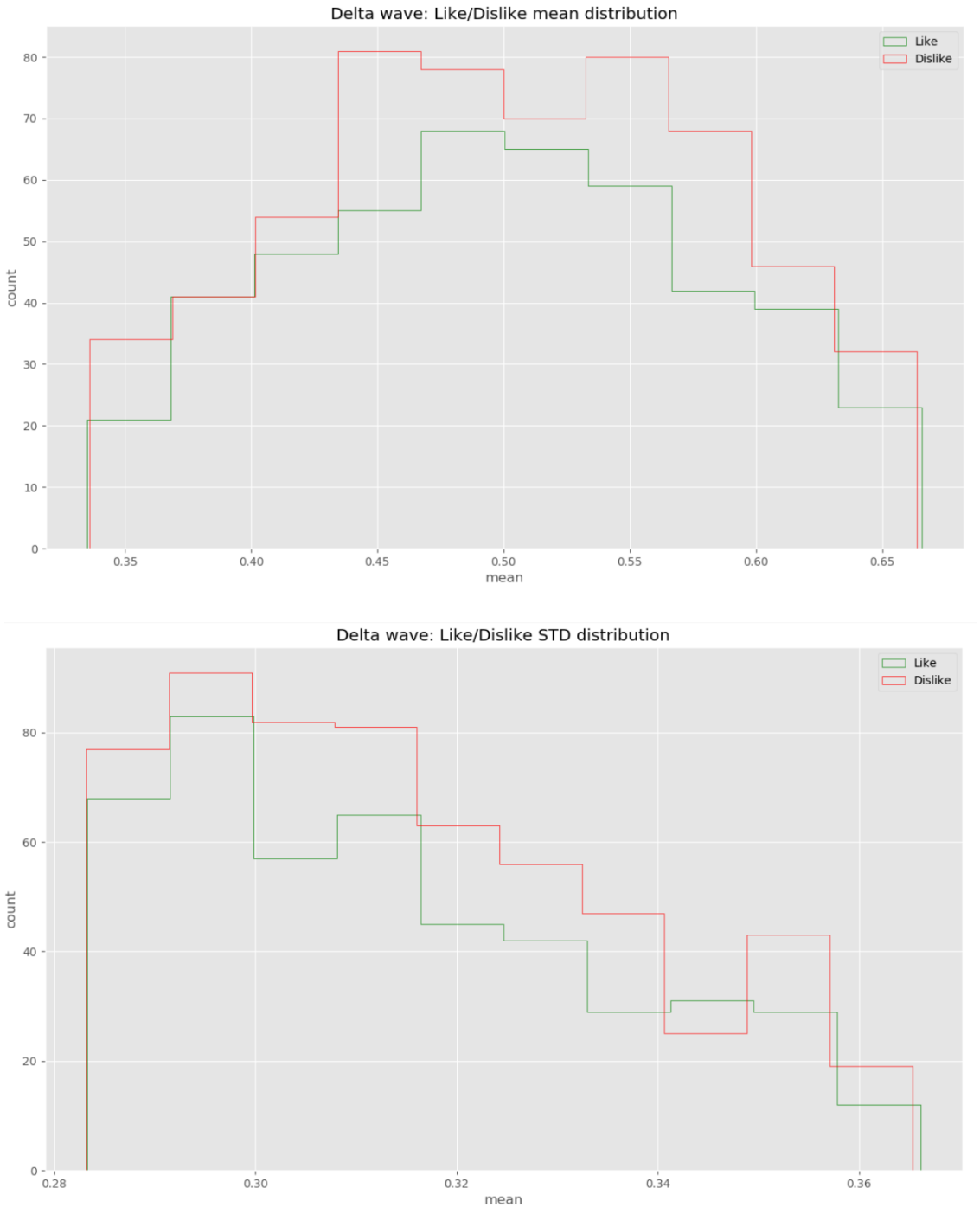

Figure 5.1: Means and standard deviation distributions for the 'Delta' brain wave component

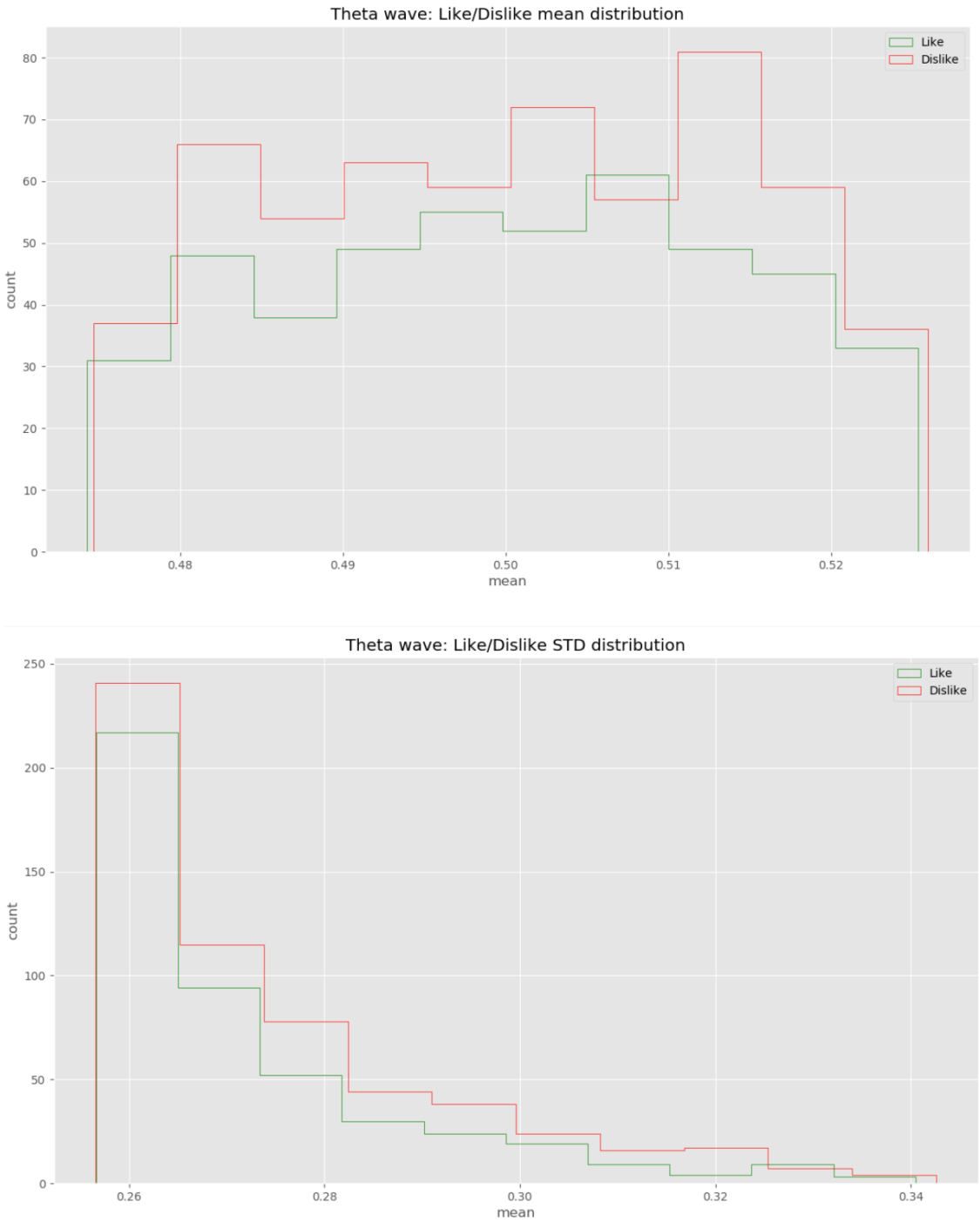

Figure 5.2: Means and standard deviation distributions for the 'Theta' brain wave component

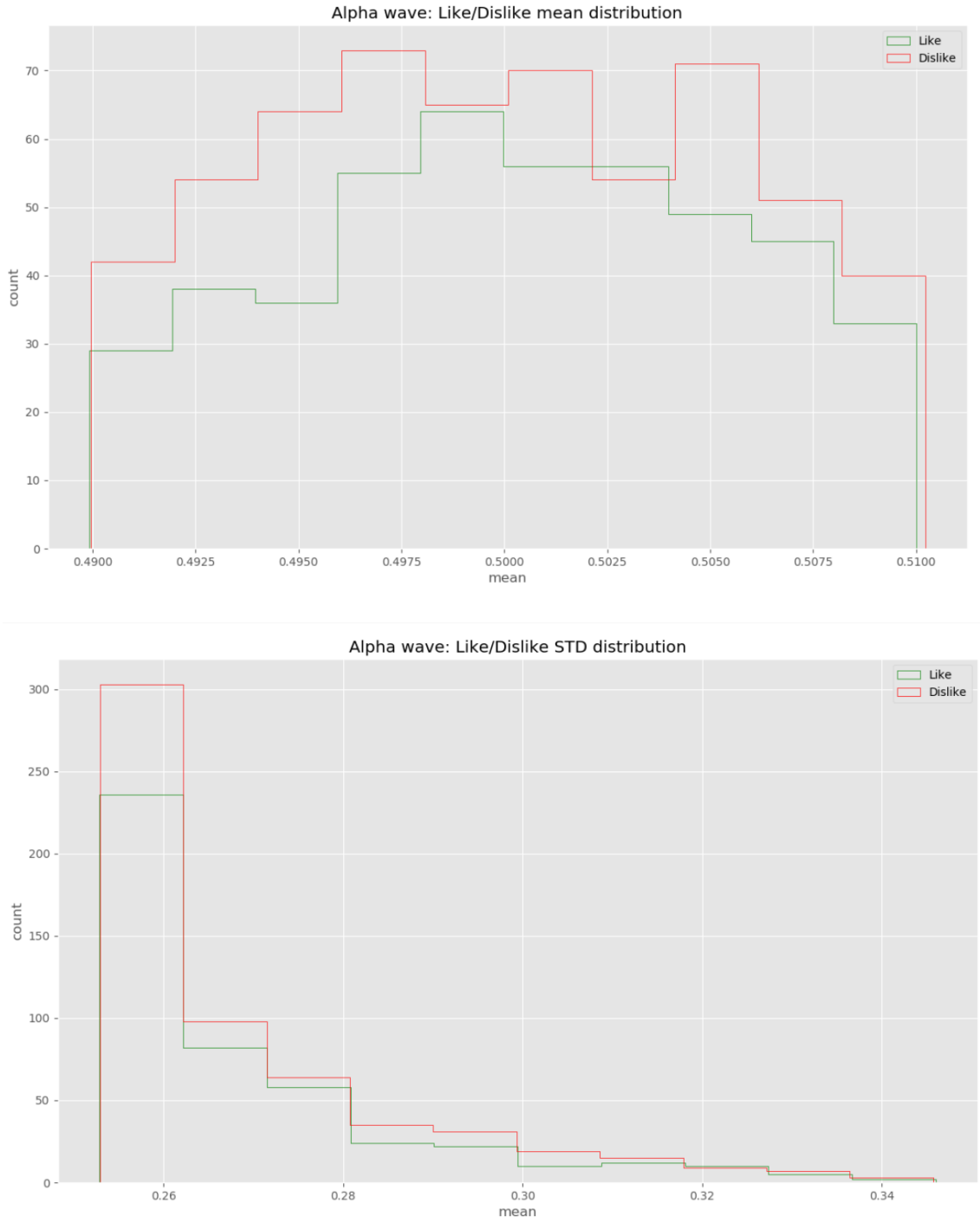

Figure 5.3: Means and standard deviation distributions for the 'Alpha' brain wave component

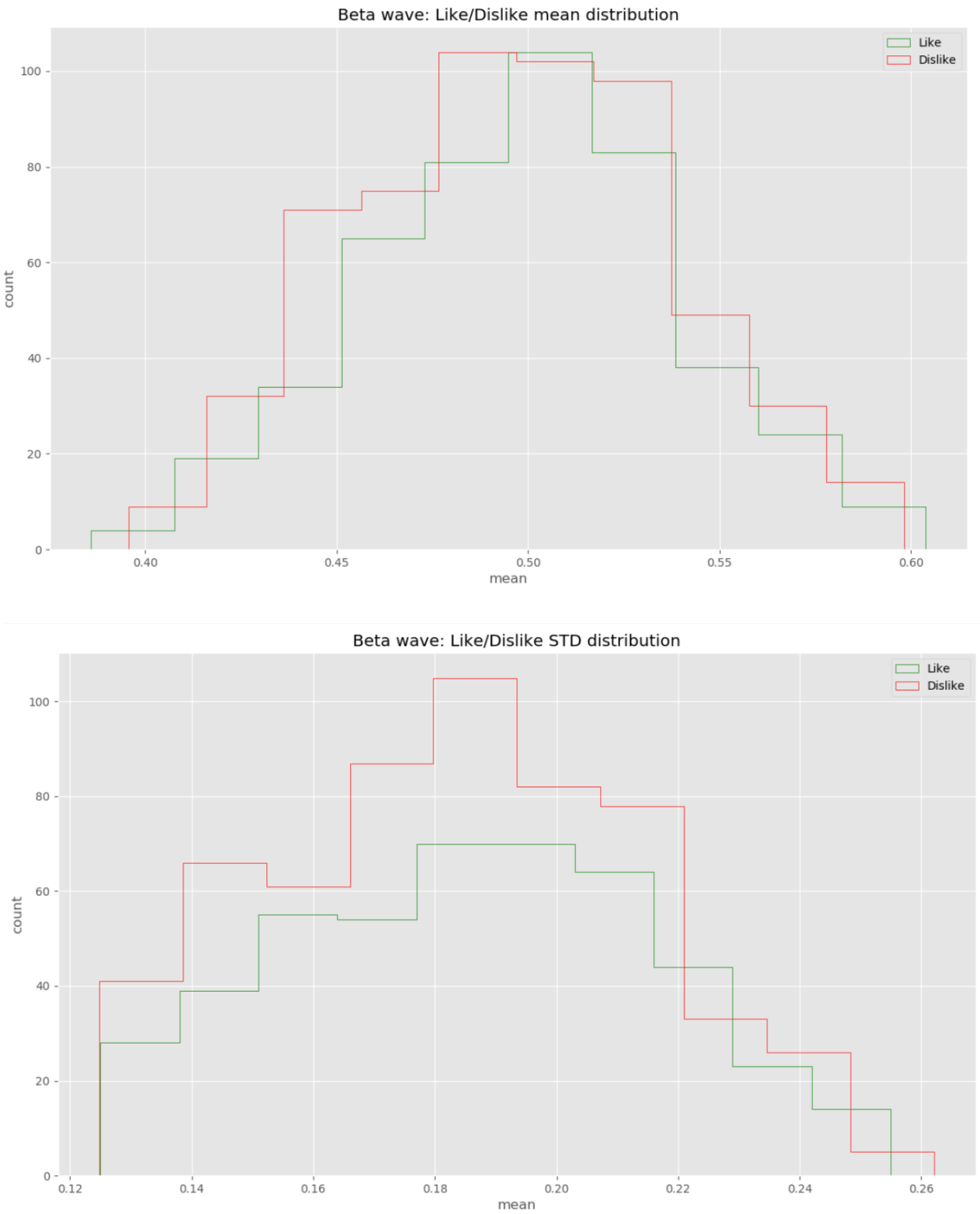

Figure 5.4: Means and standard deviation distributions for the 'Beta' brain wave component

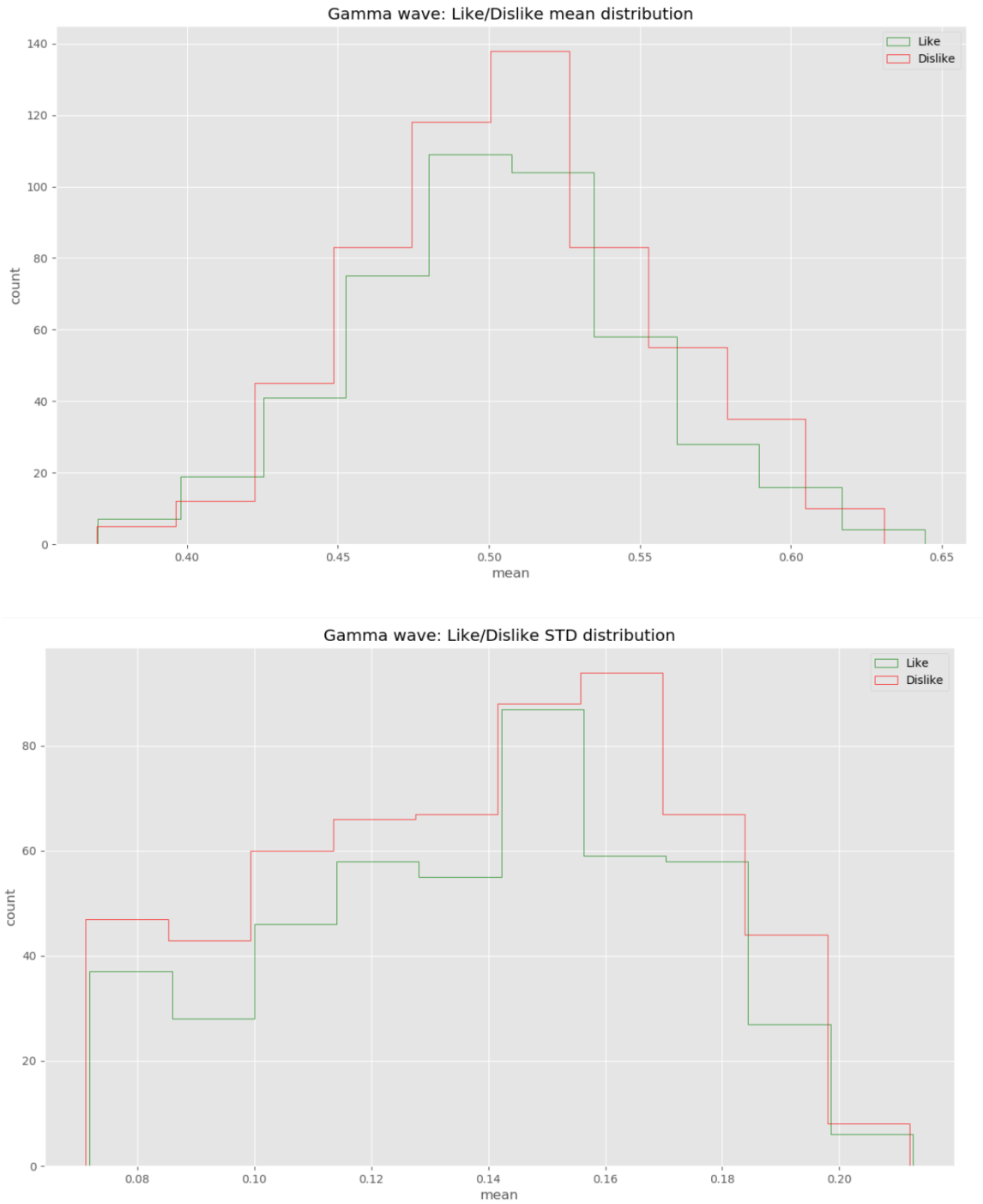

Figure 5.5: Means and standard deviation distributions for the 'Gamma' brain wave component

5.1.2 TWO DATA FEATURES: MEANS AND STANDARD DEVIATIONS

For this set of experiments we combined the mean and standard deviation of each brain wave signal. Figures 5.6 – 5.10 show the plottings of these two features, for each brain wave. While some of the plottings exhibit interesting distributions (apparently along parabolas), there is no indication, again, that the combination of mean and standard deviation would offer any indication of good signal candidates for classification.

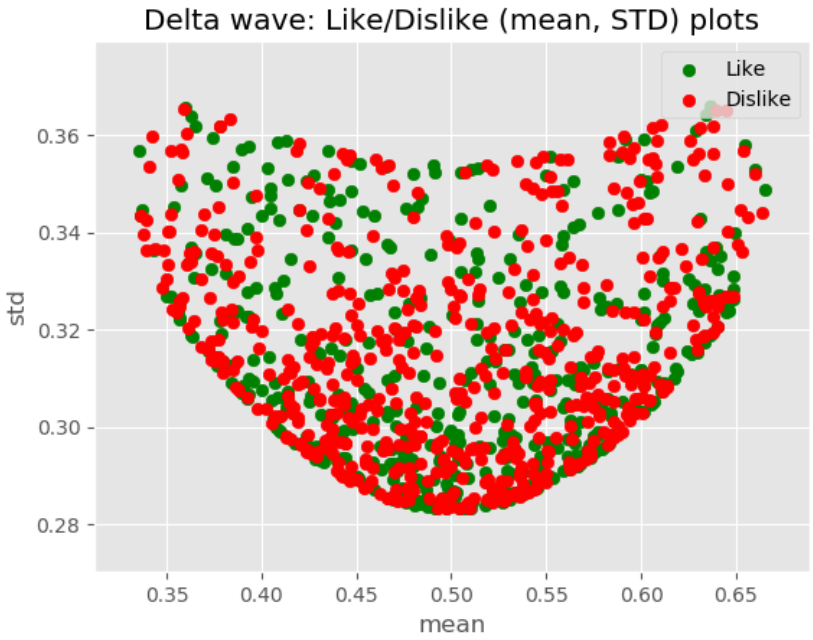

Figure 5.6: Means vs standard deviations for the 'Delta' brain wave component

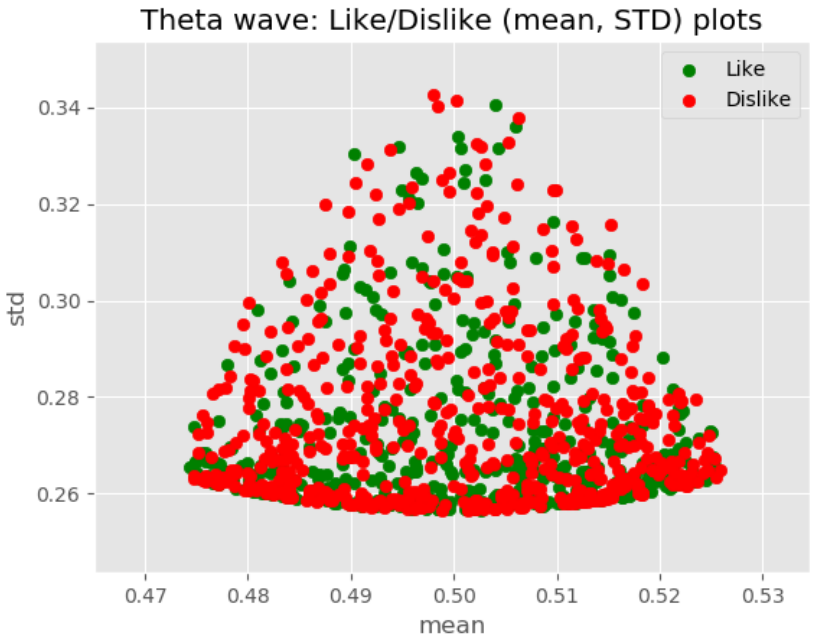

Figure 5.7: Means vs standard deviations for the 'Theta' brain wave component

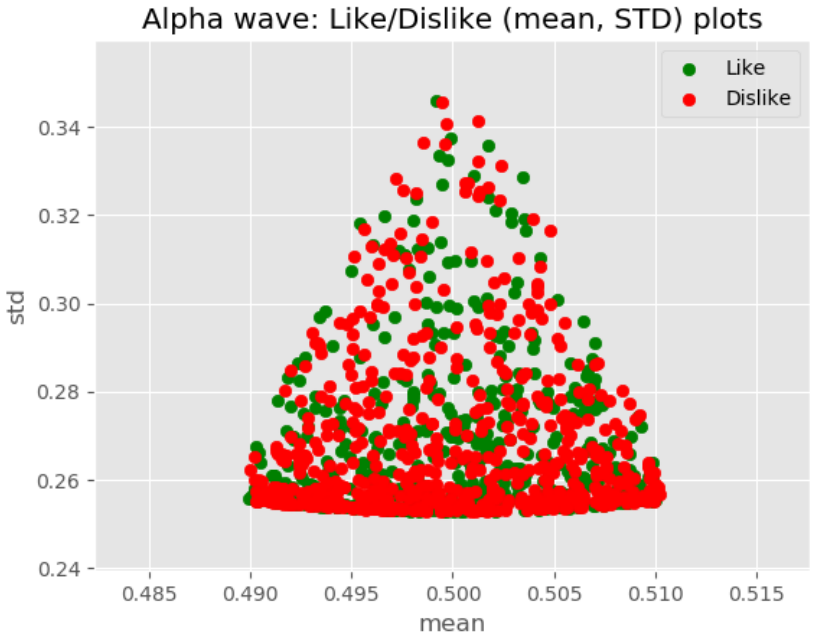

Figure 5.8: Means vs standard deviations for the 'Alpha' brain wave component

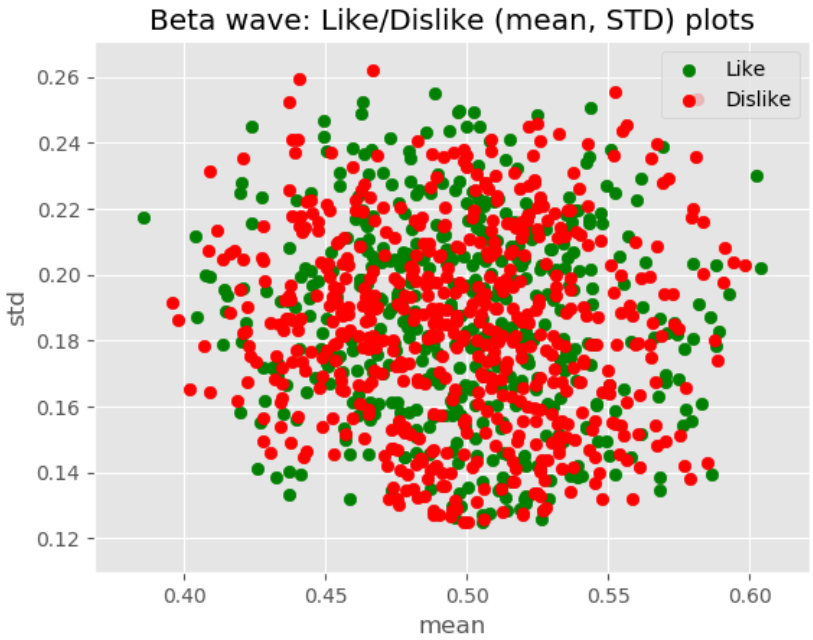

Figure 5.9: Means vs standard deviations for the 'Beta' brain wave component

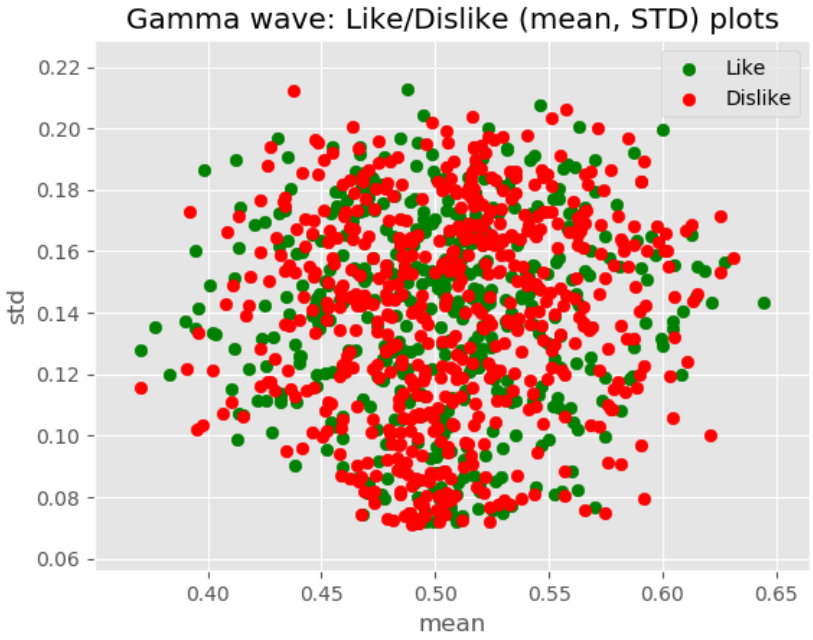

Figure 5.10: Means vs standard deviations for the 'Gamma' brain wave component

5.2 SINGLE BRAIN WAVE SIGNAL COMPONENTS CLASSIFICATION

The model's accuracies for analysing each brain wave component (one at the time) are presented in Figures 5.11 - 5.15. For each figure, the accuracy of the model for a subject s_k represents the accuracy of the model when the respective subject s_k data is the test data, whereas all other subjects' data represent the model's training data. For each brain wave component, a model was created for each subject $s_k,\ k = 1, ..., 25$, and each model's accuracy was recorded (for a total of 25 model accuracies per each brain wave component). Each figure also shows the mean accuracy of all 25 models when the respective brain wave components are being used for the analysis. More formally, for each brain wave component $x \in \{\delta, \theta, \alpha, \beta, \gamma\}$ and each subject s, the dataset $\mathcal{D} = \mathcal{T}_s \cup \mathcal{S}_s$ is composed of the training set

$$\mathcal{T}_s = \{x_{ki} \in \mathcal{D} \mid k = 1, ..., 25,\ k \neq s;\ i = 1, ..., 42\}$$

and the test set

$$\mathcal{S}_s = \{x_{ki} \in \mathcal{D} \mid k = 1, ..., 25,\ k = s;\ i = 1, ..., 42\}$$

(where indices k, i represent the subject and image, respectively).

The main purpose of this experiment was producing a quantitative comparison between using each individual brain wave component for analysis. Due to relative long amounts of time needed for running these experiments (typically 10-14 hours), we did not aim for obtaining the highes possible accuracies, which would require many neural network hidden nodes and/or multiple layers (and hence very time consuming). We used an LSTM model with one hidden layer and 256 nodes, trained in 150 epochs.

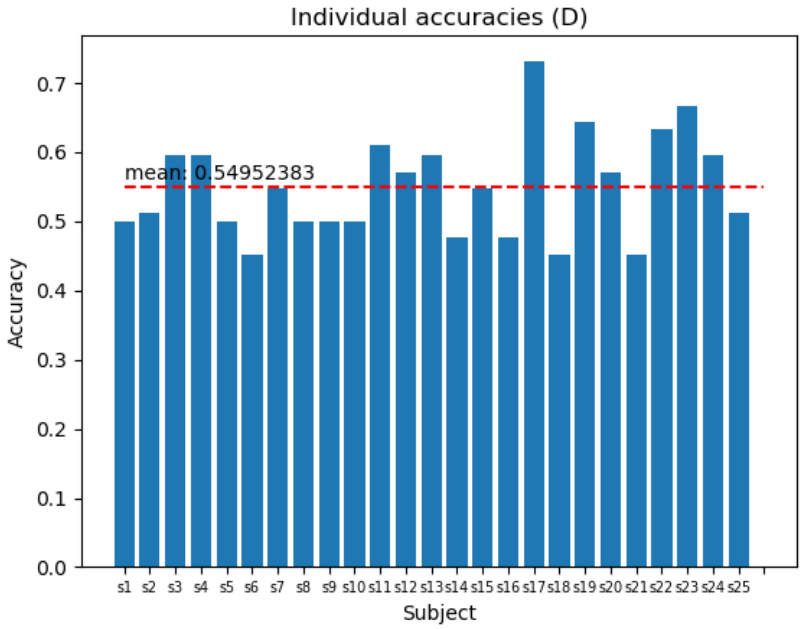

Figure 5.11: Classification accuracies using the 'Delta' brain wave component

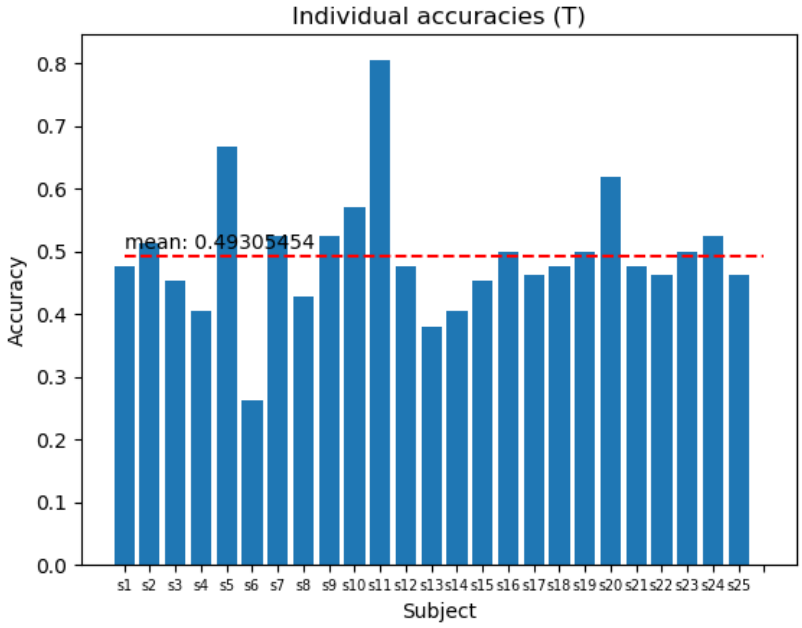

Figure 5.12: Classification accuracies using the 'Theta' brain wave component

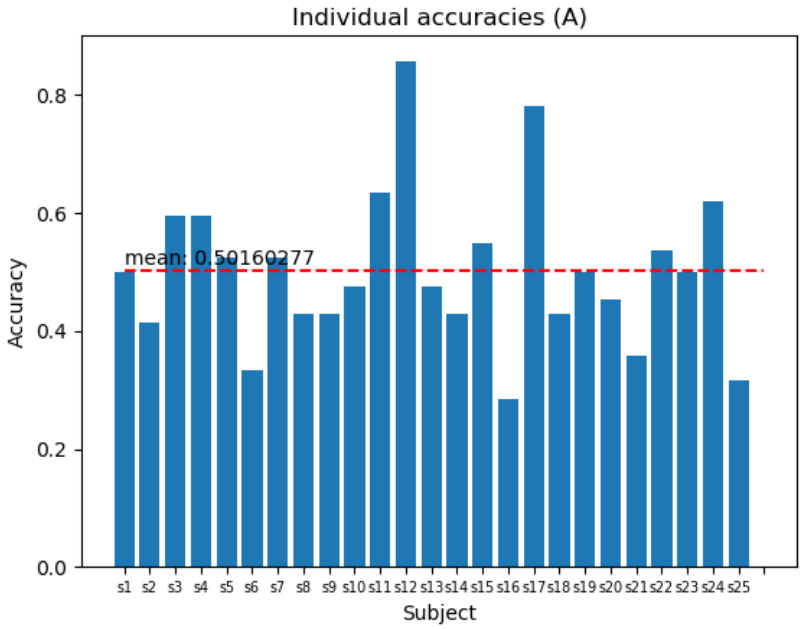

Figure 5.13: Classification accuracies using the 'Alpha' brain wave component

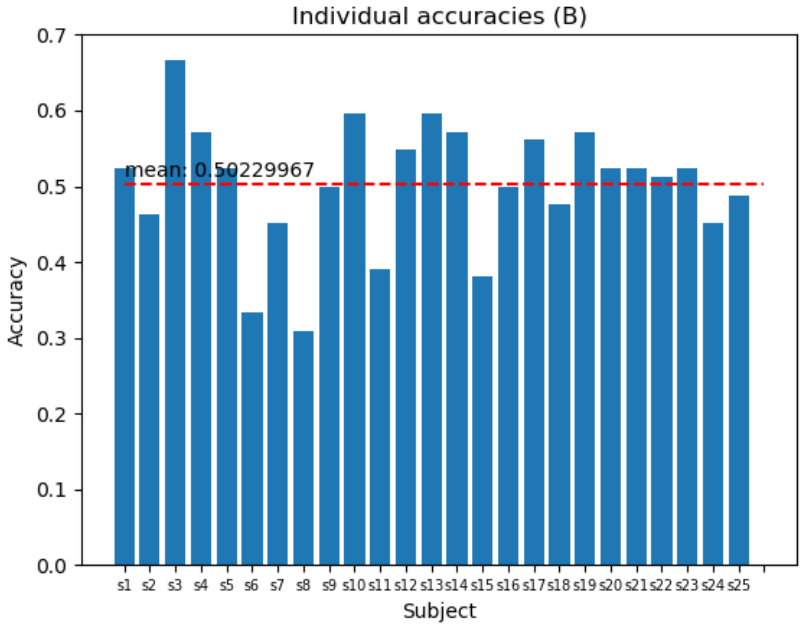

Figure 5.14: Classification accuracies using the 'Beta' brain wave component

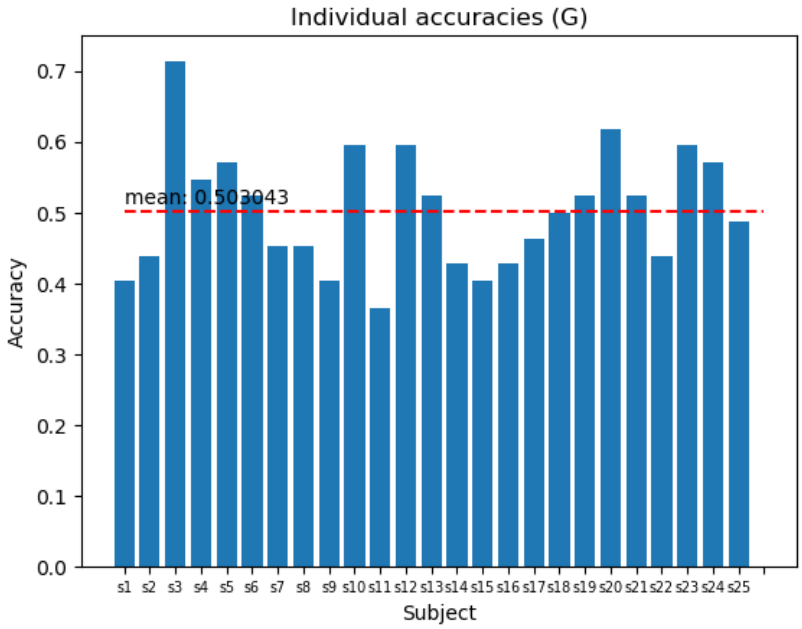

Figure 5.15: Classification accuracies using the 'Gamma' brain wave component

The mean accuracies when each brain wave components is being used for analysis are summarized in Table 5.1.

Component	Mean accuracy [%]
Delta	54.95
Theta	49.31
Alpha	50.16
Beta	50.23
Gamma	50.30

Table 5.1: Mean accuracies for each brain wave component used for analysis

The results in Figures 5.11-5.15 and Table 5.1 show the brain wave component 'Delta' as the most significant in performing the preference classification, with the other components performing about the same. Clearly, 'Delta' component is a first candidate in any

mix of components to be analyzed. However, some other components, in combination with 'Delta' may help raise the accuracies for the subjects who perform not so well for the 'Delta' component. For instance, 'Gamma' may contribute to raising accuracies for $s18$ or $s21$, which are lower for the 'Delta' component. We present experimental results for analyzing multiple components in the subsequent section.

The Python code for the results presented in this section is given in Appendix C.

5.3 Binary classification using multiple brain wave signal components

For this set of experiments we used various combination of brain waves to perform the binary classification. As before, due to the fact that each experiment was very time consuming (10-14 hours), we used a rather small LSTM model (one hidden layer, 256 nodes, 150 epochs for training) and we tried to experimentally determine which combination(s) of brain waves would produce the best classification results. In addition, given the well-known Neural Networks "appetite" for training data and their very non-convex cost function (see, for instance, [8]), we must remark that our relatively small training dataset is unlikely to produce the best model parameters even for such small LSTM model. Figure 5.16 illustrates the training process for one of the experiments in this section. While the accuracy shows asymptotic convergence to maximum, the evolution is not monotonic, as during the process multiple local minima are likely found.

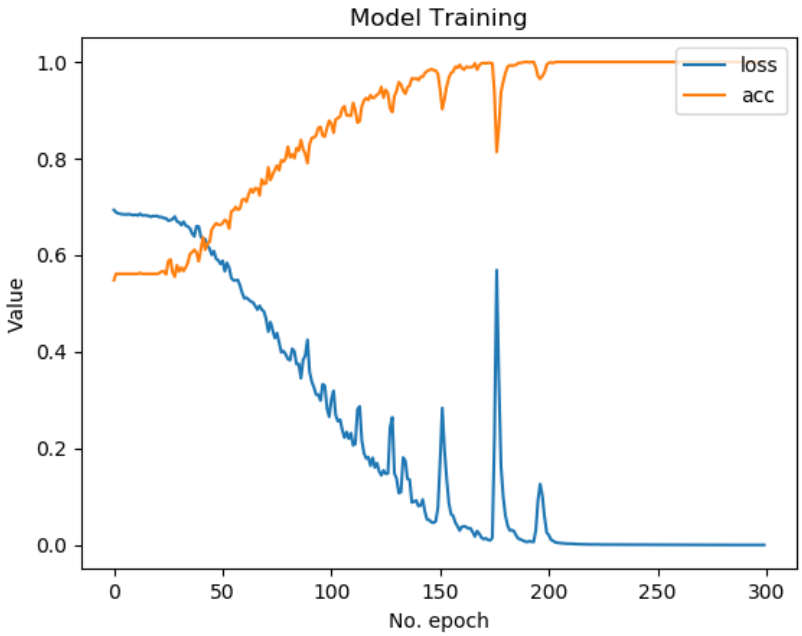

Figure 5.16: The training process of an LSTM model

The experiments were organized as follows:

- Find how many brain waves combined would produce best classification results.
- Find which bran waves combination is best.
- Find which part of the signals are best to be analyzed for the binary classification purpose.

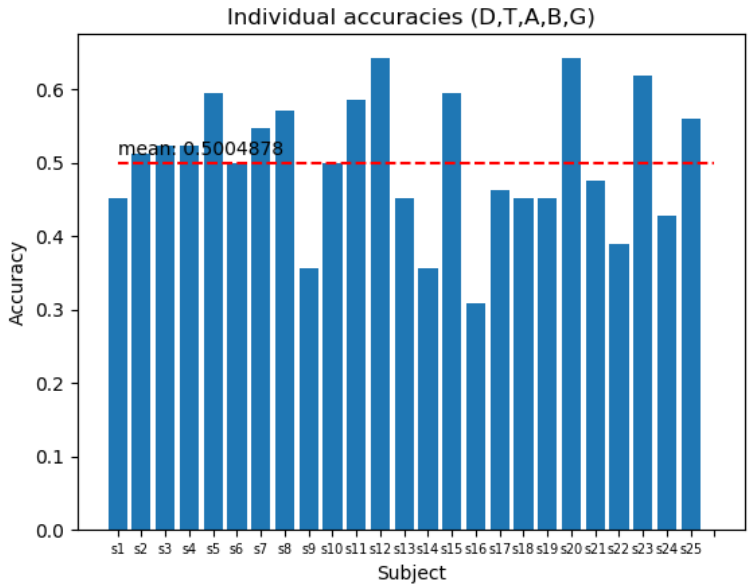

Figure 5.17: Classification accuracies using all brain wave components

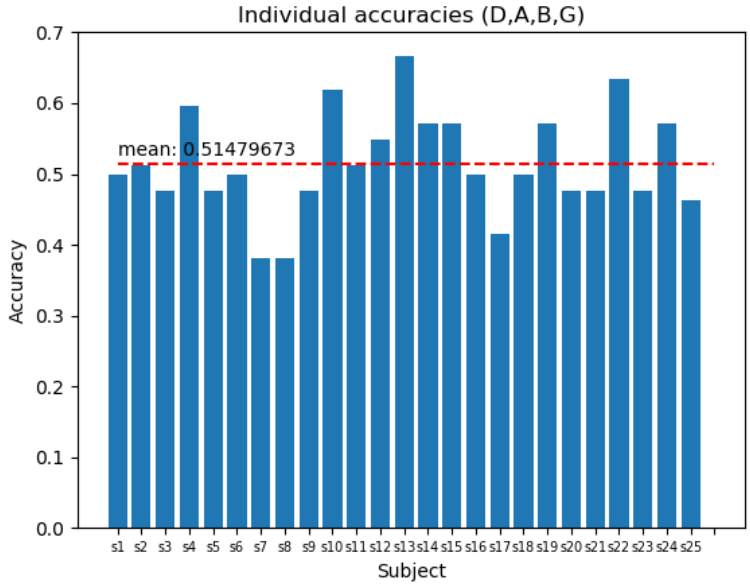

Figure 5.18: Classification accuracies using four brain wave components

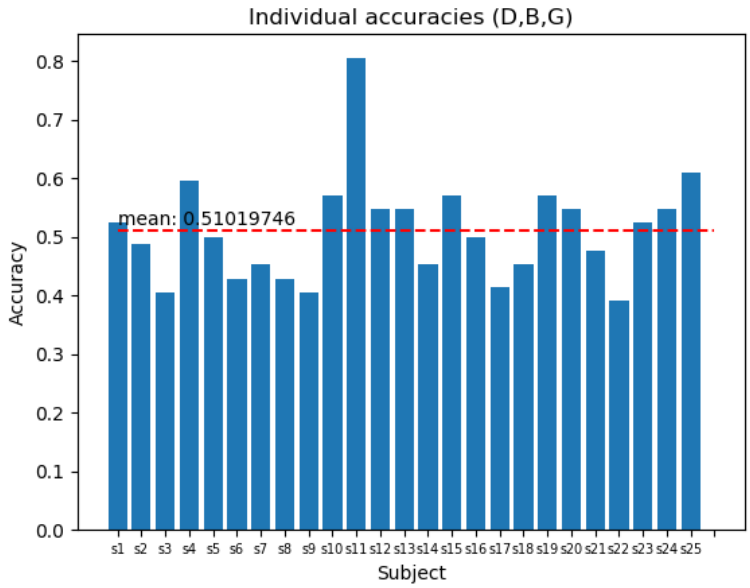

Figure 5.19: Classification accuracies using three brain wave components

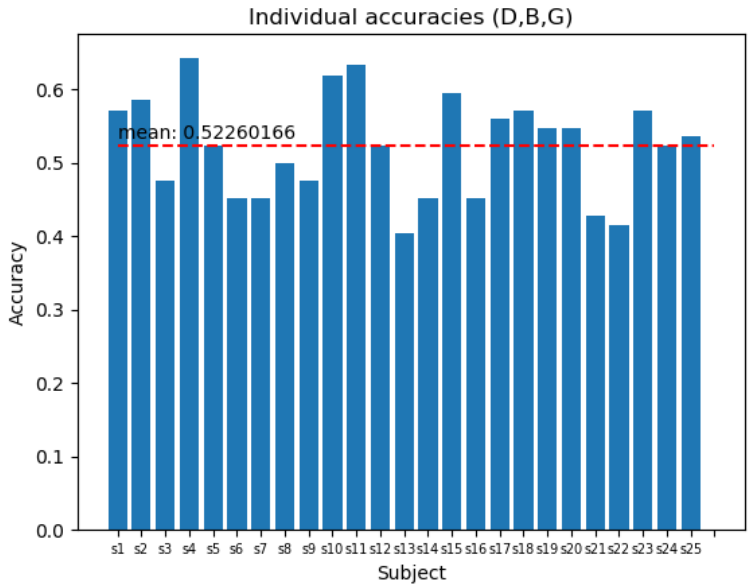

Figure 5.20: Classification accuracies using three brain wave components and the first part of the signal [1,255]

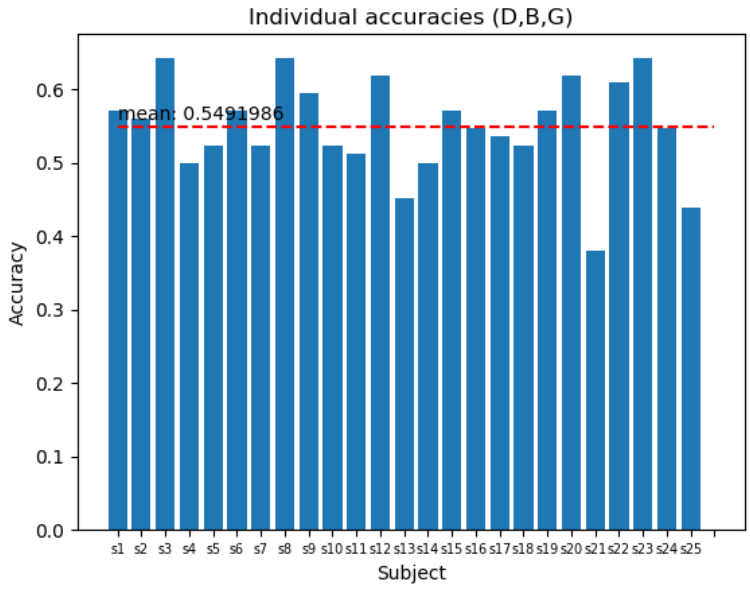

Figure 5.21: Classification accuracies using three brain wave components and the middle part of the signal [125,425]

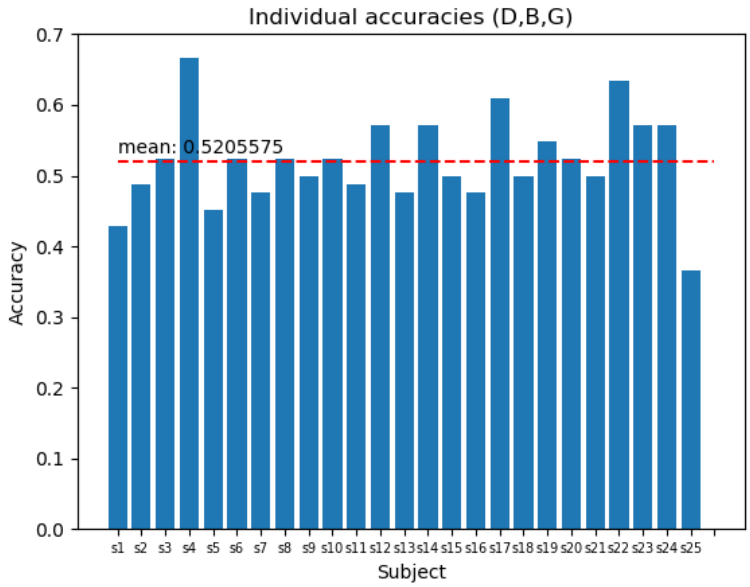

Figure 5.22: Classification accuracies using three brain wave components and the last part of the signal [255,512]

As the results in Figures 5.17 (all five brain waves), 5.18 (four brain waves), and 5.19 show, combining all brain waves does not necessarily yield the best accuracy. The results allow us to conjecture that four or even three brain waves can produce significantly better results. Unfortunately, determining which combination is best appears to be difficult to determine by means other than experimental.

For the rest of the experiments, we selected three brain waves (Delta, Beta, and Gamma) and analyzed different parts of them to determine which part would produce the best accuracy. By comparing the results shown in Figures 5.19 (the whole signal), 5.20 (the first part of the signal), 5.21 (the middle part of the signal), and 5.22 (the last part of the signal) we can clearly conclude that analyzing only a fragment of the whole signal (the middle part, in our experiments) produces the best results. This conclusion is consistent with the intuition that one brain takes a little bit of time to establish a preference, then the intensity of the brain signals fades out after the decision was established.

The Python code for the results presented in this section is given in Appendix C.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this work, we performed an analysis of EEG signals collected from 25 human volunteers who observed 42 commercial advertising images and recorded their preference *like* or *dislike* for each image observed. The original brain wave signals were collected by means of 14 sensors through a Brain-Computer Interface (BCI). We proposed an Artificial Neural Network (ANN) based model for automatically classifying a subject's preference for an advertising image. First, we decomposed each recorded brain wave signal into its five components, then aggregated all 14 components from each sensor signal. Our model has the flexibility of selecting one or more brain wave components for performing the classification.

The model we propose relies on analyzing brain wave components ('Delta,' 'Theta,' 'Alpha,' 'Beta,' and 'Gamma') collected through a Brain-Computer Interface (BCI) device while 25 human subjects are exposed to advertising images of various products [38]. Previous work [32] shows that the data can be classified using Artificial Neural Networks (ANN) models with high accuracy. However, the work in [32] considers the collection of brain wave data as a whole, without grouping it per human subjects. While the analysis produces good insights on performing such data classification, it would be impractical for detecting new human subjects' preferences. In this work, we overcome this disadvantage and propose a classification model trained on data collected from 24 (out of 25) human subjects, while the accuracy of the model is verified on the data collected from the 25th subject. This corresponds to a practical situation where the model is created based on the data collected from volunteers and used to predict preferences for other subjects, previously unknown to the model.

The extensive experimental results show clearly that the model's classification is more accurate if only a few components are selected for the analysis, rather than analyzing all

components as in previous work [38] using the same dataset.

The main contribution of this work is a flexible classification model capable of producing brain wave binary classification based on one or more brain wave components, while previous work [38] uses all brain wave components for such classification. As our experimental results show, using fewer brain wave components produces classification with higher accuracy. A non-experimental method for finding an optimal mix of brain wave components that produce the highest classification accuracy is not subject to this work and is left for future research. Moreover, our model can analyze only a fraction of the brain wave signal, and the experimental results show that this approach produces better classification results.

In our study we have learned that each brain wave signal contains significantly more information than the binary *like/dislike* we were looking for. Intuitively, when a human subject visualizes an image, the brain must "encode" much more information. One direction to explore in future work would be finding connections between the brain waves being recorded and the specific image presented to the human subject. That is, we leave as an open question whether image detection (multiclass classification) can be performed on the dataset we analyzed.

REFERENCES

[1] Priyanka A Abhang, Bharti W Gawali, and Suresh C Mehrotra, *Introduction to EEG and speech-based emotion recognition*, Academic Press, 2016.

[2] _____, *Technical aspects of brain rhythms and speech parameters*, Introduction to EEG-and Speech-Based Emotion Recognition (2016), 51–79.

[3] Tim Ambler, Sven Braeutigam, John Stins, Steven Rose, and Stephen Swithenby, *Salience and choice: neural correlates of shopping decisions*, Psychology & Marketing **21** (2004), no. 4, 247–261.

[4] Davide Baldo, Hirak Parikh, Yvonne Piu, and Kai-Markus Müller, *Brain waves predict success of new fashion products: a practical application for the footwear retailing industry*, Journal of Creating Value **1** (2015), no. 1, 61–71.

[5] Anusha Baskaran, Roumen Milev, and Roger S McIntyre, *The neurobiology of the EEG biomarker as a predictor of treatment response in depression*, Neuropharmacology **63** (2012), no. 4, 507–513.

[6] Edvin Beqari, *A Very Basic Introduction to Feed-Forward Neural Networks*, https://dzone.com/, Janary 2018.

[7] Christopher M Bishop, *Pattern recognition*, Machine learning **128** (2006), no. 9.

[8] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun, *The Loss Surfaces of Multilayer Networks*, 2015.

[9] James W Cooley and John W Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of computation **19** (1965), no. 90, 297–301.

[10] Emrah Düzel, Will D Penny, and Neil Burgess, *Brain oscillations and memory*, Current opinion in neurobiology **20** (2010), no. 2, 143–149.

[11] Vitor Costa Rozan Fortunato, Janaina de Moura Engracia Giraldi, and Jorge Henrique Caldeira de Oliveira, *A review of studies on neuromarketing: Practical results, techniques, contributions and limitations*, Journal of Management Research **6** (2014), no. 2, 201.

[12] Parnaz Golnar-Nik, Sajjad Farashi, and Mir-Shahram Safari, *The application of eeg power for the prediction and interpretation of consumer decision-making: a neuromarketing study*, Physiology & behavior **207** (2019), 90–98.

[13] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed, *Hybrid speech recognition with deep bidirectional LSTM*, 2013 IEEE workshop on automatic speech recognition and understanding, IEEE, 2013, pp. 273–278.

[14] Dishashree Gupta, *Introduction to Recurrent Neural Networks*, https://www. analyticsvidhya.com/, December 2017.

[15] Simon Haykin, *Self-organizing maps*, Neural networks-A comprehensive foundation, 2nd edition, Prentice-Hall (1999).

[16] Nora A Herweg and Michael J Kahana, *Spatial representations in the human brain*, Frontiers in human neuroscience **12** (2018), 297.

[17] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[18] Joshua Jacobs and Michael J Kahana, *Direct brain recordings fuel advances in cognitive electrophysiology*, Trends in cognitive sciences **14** (2010), no. 4, 162–171.

[19] Nick Lee, Amanda J Broderick, and Laura Chamberlain, *What is 'neuromarketing'? a discussion and agenda for future research*, International journal of psychophysiology **63** (2007), no. 2, 199–204.

[20] Zachary C Lipton, John Berkowitz, and Charles Elkan, *A critical review of recurrent neural networks for sequence learning*, arXiv preprint arXiv:1506.00019 (2015).

[21] Fernando Lopez, *From a LSTM cell to a Multilayer LSTM Network with PyTorch*, https:// towardsdatascience.com/from-a-lstm-cell-to-a-multilayer-lstm-network-with-pytorch-2899eb5696f3, 2020.

[22] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba, *Addressing the rare word problem in neural machine translation*, arXiv preprint arXiv:1410.8206 (2014).

[23] Erik Marchi, Giacomo Ferroni, Florian Eyben, Leonardo Gabrielli, Stefano Squartini, and Björn Schuller, *Multi-resolution linear prediction based features for audio onset*

detection with bidirectional LSTM neural networks, 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, 2014, pp. 2164–2168.

[24] Samuel M McClure, Jian Li, Damon Tomlin, Kim S Cypert, Latané M Montague, and P Read Montague, *Neural correlates of behavioral preference for culturally familiar drinks*, Neuron **44** (2004), no. 2, 379–387.

[25] Warren S McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5** (1943), no. 4, 115–133.

[26] Rekha Molala, *The Ascent of Gradient Descent*, https://blog.clairvoyantsoft.com/, September 2019.

[27] M Murugappan, Subbulakshmi Murugappan, Celestin Gerard, et al., *Wireless eeg signals based neuromarketing system using fast fourier transform (fft)*, 2014 IEEE 10th international colloquium on signal processing and its applications, IEEE, 2014, pp. 25–30.

[28] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall, *Activation functions: Comparison of trends in practice and research for deep learning*, arXiv preprint arXiv:1811.03378 (2018).

[29] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio, *On the difficulty of training recurrent neural networks*, International conference on machine learning, 2013, pp. 1310–1318.

[30] Michael Phi, *Illustrated Guide to LSTM's and GRU's: A step by step explanation*, https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21, 2018.

[31] Sridhar Raghavachari, Michael J Kahana, Daniel S Rizzuto, Jeremy B Caplan, Matthew P Kirschen, Blaise Bourgeois, Joseph R Madsen, and John E Lisman, *Gating of human theta oscillations by a working memory task*, Journal of Neuroscience **21** (2001), no. 9, 3175–3183.

[32] Lateef Rasheed, *Decision pattern detection from brain response to marketing stimuli*, https://digitalcommons.georgiasouthern.edu/etd/2185, 2020.

[33] Frank Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain.*, Psychological review **65** (1958), no. 6, 386.

[34] Neelam Rout, *Classifications & Misclassifications of EEG Signals using Linear and AdaBoost Support Vector Machines*, International Journal Of Advanced Research, Ideas And Innovations in Technology **1** (2014), no. 2, 1–6.

[35] Shashi Sathyanarayana, *A gentle introduction to backpropagation*, Numeric Insight **7** (2014), 1–15.

[36] Wolf Singer, *Synchronization of cortical activity and its putative role in information processing and learning*, Annual review of physiology **55** (1993), no. 1, 349–374.

[37] WWW, *Perceptrons and multi-layer perceptrons*, https://missinglink.ai/guides/neural-network-concepts/perceptrons-and-multi-layer-perceptrons-the-artificial-neuron-at-the-core-of-deep-learning/, November 2020.

[38] Mahendra Yadava, Pradeep Kumar, Rajkumar Saini, Partha Pratim Roy, and Debi Prosad Dogra, *Analysis of EEG signals and its application to neuromarketing*, Multimedia Tools and Applications **76** (2017), no. 18, 19087–19111.

[39] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, *Recurrent neural network regularization*, arXiv preprint arXiv:1409.2329 (2014).

APPENDIX A

FILTERING AND SCALING

```
# -*- coding: utf-8 -*-
"""
Listing 1: Scales and saves as a csv file

@author: Lorela Bano
"""

#%% imports
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

#%%-------------- read data -----------------
df = pd.read_csv('../data/AllNeuroMarketingEEGsFiltered.csv'
   , index_col = 0)
#remove the sensors and name columns
del df['Sensor']
del df['Name']

subjects = df.Id.unique()

#keep only the ones used for classification
freqBand = ['Delta', 'Theta', 'Alpha', 'Beta', 'Gamma']
freqBandAbbr = ['D', 'T', 'A', 'B', 'G']
```

```
#freqBand = ['Delta', 'Beta']
#freqBandAbbr = ['D', 'B']

# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))

#%%------ collect and prepare data ----------
dataset = pd.DataFrame()
datasetY = []
for i in range(len(freqBand)):
    wn = freqBand[i]
    wna = freqBandAbbr[i]
    dataL = df.loc[(df['Wave'] == wna) & (df['Like'] == True
       )].groupby(['Id','Wave'], as_index=False).agg('sum')#
       .loc[:,'X1':'X512']
    dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
       False)].groupby(['Id','Wave'], as_index=False).agg('
       sum')#.loc[:,'X1':'X512']
    datasetY = np.concatenate([datasetY, np.concatenate((np.
       repeat(True, dataL.shape[0]),np.repeat(False, dataD.
       shape[0])))])
    #normalize
    #dataL = scaler.fit_transform(dataL)
    #dataD = scaler.fit_transform(dataD)
    if (len(dataset) == 0):
        #dataset= np.vstack([dataL, dataD])
```

```
            dataset = pd.concat([dataL, dataD], axis=0)
            #np.reshape(dataset, (dataset.shape[0], 1, dataset.
               shape[1]))
        else:
            vstack = pd.concat([dataL, dataD], axis=0)
            #dataset = np.stack([dataset, np.vstack([dataL,
               dataD])], axis = 1)
            dataset = pd.concat([dataset, vstack], axis=0)

dataset.loc[dataset['Like'] > 0,'Like'] = True
dataset.loc[dataset['Like'] == 0,'Like'] = False

datasetX = scaler.fit_transform(dataset.loc[:,'X1':'X512'].
   transpose()).transpose()
#datasetX = dataset.loc[:,'X1':'X512']
subjectsX = [sid.rpartition("_")[0] for sid in dataset.Id]
#%%-------construct the new data frame ----------
#Fields: subjectsX, dataset.Id, dataset.Wave, dataset.Like,
   datasetX
newDF = pd.DataFrame(list(zip(subjectsX, dataset.Id, dataset
   .Wave, dataset.Like)))
newDF = pd.DataFrame(np.concatenate([newDF, datasetX], axis
   = 1),
                     columns = np.hstack((np.array(["Name",
                        "Id", "Wave", "Like"]), np.array(
                        dataset.columns[3:]))))
```

```
#%%-------- save in a csv file --------------------
newDF.to_csv('../data/
   AllNeuroMarketingEEGsFilteredScaledNames2.csv')
```

APPENDIX B

DATA STATS

```
# -*- coding: utf-8 -*-
"""
Listing 2: Brain signal types comparisons

@author: Lorela Bano
"""

#%% imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from matplotlib import colors

plt.style.use("ggplot")

#%%-------------- read data -----------------
alldf = pd.read_csv('../data/
   AllNeuroMarketingEEGsFilteredScaledNames2.csv', index_col
    = 0)
#remove the sensors and name columns
del alldf['Id']

freqBand = ['Delta', 'Theta', 'Alpha', 'Beta', 'Gamma']
```

```
freqBandAbbr = ['D', 'T', 'A', 'B', 'G']

# normalizer (optional use after aggregation)
scaler = MinMaxScaler(feature_range=(0, 1))

#%%--------  some sample plots --------------------
plt.figure()
#Get (some) like and dislike signals
yl = alldf.loc[(alldf['Wave']=='D') & (alldf['Like']==True),
   'X1':'X512'].values[0]
yd = alldf.loc[(alldf['Wave']=='D') & (alldf['Like']==False)
   ,'X1':'X512'].values[0]
plt.plot(yl, 'g', label = 'Like')
plt.plot(yd, 'r', label = 'Dislike')
plt.title("Samples of normalized 'Delta' signals")
plt.xlabel('t')
plt.ylabel('x')
plt.legend(loc="upper right")
plt.show()

#%%-------------------------------------------------
subjects = alldf.Name.unique()
#%%-------------------------------------------------
for cs in subjects:
    #cs = subjects[0]
    df = alldf[alldf['Name'] == cs]
```

```
del df['Name']


for i in range(len(freqBand)):
#for i in [0,3]:
#for i in [0]:
    wn = freqBand[i]
    wna = freqBandAbbr[i]
    #dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
        True)].groupby(['Wave']).agg('sum').loc[:,'X1':'
       X512']
    dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
       True)].loc[:,'X1':'X512']
    #normalize
    #dataL = scaler.fit_transform(dataL)
    #dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
        False)].groupby(['Wave']).agg('sum').loc[:,'X1
       ':'X512']
    dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
       False)].loc[:,'X1':'X512']
    #normalize
    #dataD = scaler.fit_transform(dataD)


    #compute means
    meansL = dataL.mean(axis = 1)
    meansD = dataD.mean(axis = 1)
```

```
#compute stds
stdL = dataL.std(axis = 1)
stdD = dataD.std(axis = 1)

#plot means distribution
plt.figure()
plt.hist(meansL, color = 'g', label = 'Like')
plt.hist(meansD, color = 'r', label = 'Dislike')
plt.title(wn + ' wave: Like/Dislike mean
   distribution')
plt.suptitle('[' + cs + ']')
plt.xlabel('mean')
plt.ylabel('count')
plt.legend(loc="upper right")
plt.show()

#plot stds distribution
plt.figure()
plt.hist(stdL, color = 'g', label = 'Like')
plt.hist(stdD, color = 'r', label = 'Dislike')
plt.title(wn + ' wave: Like/Dislike STD distribution
   ')
plt.suptitle('[' + cs + ']')
plt.xlabel('mean')
plt.ylabel('count')
plt.legend(loc="upper right")
```

```
        plt.show()

#%%-------------------------------------------------------------
for i in range(len(freqBand)):
#for i in [0,3]:
#for i in [0]:
    allmeansL = np.array([])
    allstdsL = np.array([])
    allmeansD = np.array([])
    allstdsD = np.array([])

    for cs in subjects:
        #cs = subjects[0]
        df = alldf[alldf['Name'] == cs]
        del df['Name']

        wn = freqBand[i]
        wna = freqBandAbbr[i]
        #dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
            True)].groupby(['Wave']).agg('sum').loc[:,'X1':'
           X512']
        dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
           True)].loc[:,'X1':'X512']
        #normalize
        #dataL = scaler.fit_transform(dataL)
```

```
        #dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
            False)].groupby(['Wave']).agg('sum').loc[:,'X1
            ':'X512']
        dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
           False)].loc[:,'X1':'X512']
        #normalize
        #dataD = scaler.fit_transform(dataD)

        #compute means
        meansL = dataL.mean(axis = 1)
        meansD = dataD.mean(axis = 1)
        allmeansL = np.concatenate([allmeansL, meansL])
        allmeansD = np.concatenate([allmeansD, meansD])

        #compute stds
        stdL = dataL.std(axis = 1)
        stdD = dataD.std(axis = 1)
        allstdsL = np.concatenate([allstdsL, stdL])
        allstdsD = np.concatenate([allstdsD, stdD])

#plot means distribution
plt.figure()
plt.hist(allmeansL, color = 'g', label = 'Like')
plt.hist(allmeansD, color = 'r', label = 'Dislike')
plt.title(wn + ' wave: Like/Dislike mean distribution')
plt.xlabel('mean')
```

```
    plt.ylabel('count')
    plt.legend(loc="upper right")
    plt.show()

    #plot stds distribution
    plt.figure()
    plt.hist(allstdsL, color = 'g', label = 'Like')
    plt.hist(allstdsD, color = 'r', label = 'Dislike')
    plt.title(wn + ' wave: Like/Dislike STD distribution')
    plt.xlabel('mean')
    plt.ylabel('count')
    plt.legend(loc="upper right")
    plt.show()

#%%------------ non filled histograms -------------
for i in range(len(freqBand)):
#for i in [0,3]:
#for i in [0]:
    allmeansL = np.array([])
    allstdsL = np.array([])
    allmeansD = np.array([])
    allstdsD = np.array([])

    for cs in subjects:
        #cs = subjects[0]
        df = alldf[alldf['Name'] == cs]
```

```
del df['Name']


wn = freqBand[i]
wna = freqBandAbbr[i]
#dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
    True)].groupby(['Wave']).agg('sum').loc[:,'X1':'
   X512']
dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
   True)].loc[:,'X1':'X512']
#normalize
#dataL = scaler.fit_transform(dataL)
#dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
    False)].groupby(['Wave']).agg('sum').loc[:,'X1
   ':'X512']
dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
   False)].loc[:,'X1':'X512']
#normalize
#dataD = scaler.fit_transform(dataD)


#compute means
meansL = dataL.mean(axis = 1)
meansD = dataD.mean(axis = 1)
allmeansL = np.concatenate([allmeansL, meansL])
allmeansD = np.concatenate([allmeansD, meansD])


#compute stds
```

```
    stdL = dataL.std(axis = 1)
    stdD = dataD.std(axis = 1)
    allstdsL = np.concatenate([allstdsL, stdL])
    allstdsD = np.concatenate([allstdsD, stdD])

#plot means distribution
plt.figure()
plt.hist(allmeansL, color = 'g', label = 'Like',
   histtype='step', stacked=True, fill=False)
plt.hist(allmeansD, color = 'r', label = 'Dislike',
   histtype='step', stacked=True, fill=False)
plt.title(wn + ' wave: Like/Dislike mean distribution')
plt.xlabel('mean')
plt.ylabel('count')
plt.legend(loc="upper right")
plt.show()

#plot stds distribution
plt.figure()
plt.hist(allstdsL, color = 'g', label = 'Like', histtype
   ='step', stacked=True, fill=False)
plt.hist(allstdsD, color = 'r', label = 'Dislike',
   histtype='step', stacked=True, fill=False)
plt.title(wn + ' wave: Like/Dislike STD distribution')
plt.xlabel('mean')
plt.ylabel('count')
```

```
    plt.legend(loc="upper right")
    plt.show()

#%%--collect and compute means and sdvs; plot the results--
for i in range(len(freqBand)):
#for i in [0,3]:
#for i in [0]:
    allmeansL = np.array([])
    allstdsL = np.array([])
    allmeansD = np.array([])
    allstdsD = np.array([])

    for cs in subjects:
        #cs = subjects[0]
        df = alldf[alldf['Name'] == cs]
        del df['Name']

        wn = freqBand[i]
        wna = freqBandAbbr[i]
        #dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
            True)].groupby(['Wave']).agg('sum').loc[:,'X1':'
           X512']
        dataL = df.loc[(df['Wave'] == wna) & (df['Like'] ==
           True)].loc[:,'X1':'X512']
        #normalize
        #dataL = scaler.fit_transform(dataL)
```

```
    #dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
        False)].groupby(['Wave']).agg('sum').loc[:,'X1
       ':'X512']
    dataD = df.loc[(df['Wave'] == wna) & (df['Like'] ==
       False)].loc[:,'X1':'X512']
    #normalize
    #dataD = scaler.fit_transform(dataD)

    #compute means
    meansL = np.array(dataL.mean(axis = 1))
    meansD = np.array(dataD.mean(axis = 1))
    allmeansL = np.concatenate([allmeansL, meansL])
    allmeansD = np.concatenate([allmeansD, meansD])

    #compute stds
    stdL = np.array(dataL.std(axis = 1))
    stdD = np.array(dataD.std(axis = 1))
    allstdsL = np.concatenate([allstdsL, stdL])
    allstdsD = np.concatenate([allstdsD, stdD])


#plot the results
plt.figure()
plt.scatter(allmeansL, allstdsL, color = 'g', label = '
   Like')
```

```
plt.scatter(allmeansD, allstdsD, color = 'r', label = '
   Dislike')
plt.title(wn + ' wave: Like/Dislike (mean, STD) plots')
plt.xlabel('mean')
plt.ylabel('std')
plt.legend(loc="upper right")
plt.show()
```

APPENDIX C

RNN MODEL

```
# -*- coding: utf-8 -*-
"""
Listing 3: Recurrent Neural Network (LSTM) Model


@author: Lorela Bano
"""
#%% imports
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#--------------------- NN -------------------------------
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import SimpleRNN
from tensorflow.keras.layers import LSTM
import seaborn as sns


#%%-------------- read data -------------------------------
df = pd.read_csv('../data/
   AllNeuroMarketingEEGsFilteredScaledNames2.csv', index_col
    = 0)
#remove the name column
#del df['Name']
```

```
subjects = df.Name.unique()
sabbrs = ['s' + str(i+1) for i in range(len(subjects))]
#%%-------------collect and prepare data -----------------
#keep only the ones used for classification
#freqBand = ['Delta', 'Theta', 'Alpha', 'Beta', 'Gamma']
#freqBandAbbr = ['D', 'T', 'A', 'B', 'G']
freqBand = ['Delta', 'Theta', 'Beta']
freqBandAbbr = ['D', 'T', 'A', 'B', 'G']

origdata = df.sort_values(by=['Name', 'Id', 'Wave'])
origdata = origdata[origdata['Wave'].isin(freqBandAbbr)]
#%%--------- run this if want to classify derivative
   --------

for i in range(4,4+510):
    origdata.iloc[:,i] = origdata.iloc[:,i+1] - origdata.
       iloc[:,i]

#%%-------- RNN params ----------

hidden_layers = 512
input_dim = len(freqBandAbbr)

np.random.seed(2020)
#method = "Simple RNN"
```

```
method = "LSTM"

NoEpochs = 300

dataWindow = [1,512]
dataWindowSize = dataWindow[1]-dataWindow[0] + 1
dataWindowRange = slice('X'+str(dataWindow[0]), 'X'+str(
    dataWindow[1]))

#%%------ perform classification ------

accs = []
#the test subject
si = 0

for si in range(len(subjects)):
#for si in range(1):
    ts = subjects[si]
    df_test = origdata[origdata['Name'] == ts]
    df_train = origdata[origdata['Name'] != ts]

    #df_train = pd.concat([#df_train, df_train,
    #                      df_train, df_train,
    #                      df_train, df_train], axis=0)

    testY = df_test[df_test['Wave'] == freqBandAbbr[0]].Like
```

```
testX = np.reshape(df_test.loc[:,dataWindowRange].
   to_numpy(), (int(df_test.shape[0]/len(freqBandAbbr)),
    len(freqBandAbbr), dataWindowSize))

trainY = df_train[df_train['Wave'] == freqBandAbbr[0]].
   Like
trainX = np.reshape(df_train.loc[:,dataWindowRange].
   to_numpy(), (int(df_train.shape[0]/len(freqBandAbbr))
   , len(freqBandAbbr), dataWindowSize))

seq_len = trainX.shape[2]
modelRNN = Sequential([
        #SimpleRNN(units=hidden_layers, input_shape=(
           input_dim,seq_len),
        #           activation="relu"),
        #LSTM(units=512, input_shape=(input_dim,seq_len)
           ,
        #           activation="relu", return_sequences=
           True),
        LSTM(units=hidden_layers, input_shape=(input_dim
           ,seq_len),
                   activation="relu"),
        #Dense(1, activation='sigmoid')
        Dense(2, activation='softmax')
    ])
```

```
    modelRNN.compile(loss='sparse_categorical_crossentropy',
        optimizer='adam', metrics=['accuracy'])
    modelRNN.summary()

    history = modelRNN.fit(trainX, trainY, epochs=NoEpochs,
       verbose=1)



    testres = modelRNN.evaluate(testX, testY)
    accs.append(testres[1])
    classification = modelRNN.predict(testX)

    res = classification.argmax(axis=1)

    # Plot history
#     figure = plt.figure()
#     plt.plot(history.history['loss'], label='loss')
#     plt.plot(history.history['acc'], label='acc')
#     plt.title('Model Training')
#     plt.ylabel('Value')
#     plt.xlabel('No. epoch')
#     plt.legend(loc="upper right")
#     plt.show()

##%%------ all accuracies --------
m = np.mean(accs)
```

```
figure = plt.figure()
plt.bar(sabbrs, accs, align='center', width=0.8)
plt.xticks(range(len(sabbrs)+1), sabbrs, size='x-small')
xs = np.array([i for i in range(len(sabbrs)+1)])
hl = np.array([m for i in range(len(sabbrs)+1)])
plt.plot(xs, hl, 'r--')
plt.text(0,m+0.01, 'mean: '+str(m))
plt.title('Individual accuracies (' + ",".join(freqBandAbbr)
    + ')')

#%%-----the confusion matrix ------
con_mat = tf.math.confusion_matrix(testY, res).eval(session=
   tf.compat.v1.Session())

#To normalize the result as from 0 to 1. Replace 'con_mat_df
    = pd.DataFrame(con_mat_norm,...)'
con_mat_norm = np.around(con_mat.astype('float') / con_mat.
   sum(axis=1)[:, np.newaxis], decimals=2)

con_mat_df = pd.DataFrame(con_mat_norm,
                     index = classes,
                     columns = classes)

figure = plt.figure()
sns.heatmap(con_mat_df, annot=True,cmap=plt.cm.Blues )#, fmt
   ='d')
```

```
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

#as counts
con_mat_df = pd.DataFrame(con_mat,
                          index = classes,
                          columns = classes)

figure = plt.figure()
sns.heatmap(con_mat_df, annot=True,cmap=plt.cm.Blues )#, fmt
    ='d')
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```