SCHOOL OF ELECTRICAL ENGINEERING
AND TELECOMMUNICATIONS

Investigating The Feasibility Of Using Non-Linear Least Squares For Sleep Spindle Classification In EEG

by

Jason Ha

Student ID: z5115925

Disseration as partial fulfillment for the degree
Bachelor of Engineering (Electrical Engineering)

Submitted: November 28, 2019
Supervisor: Beena Ahmed
Course Code: ELEC4953 (Thesis C)

Topic Title: Investigating the Feasibility Of Using Non-Linear Least Squares For Sleep Spindle Classification In EEG

Student Name: Jason Ha **Student ID:** z5115925

A. **Problem statement**

The goal of the thesis is to investigate an alternative method for sleep spindle classification that combines techniques from existing spindle detection algorithms as well machine learning techniques where the primary source of features comes directly from the QPS model and its associated parameters $a,b,c,d,e,f \in \mathbb{R}$. The ability for the QPS model to fit to sleep spindles via an NLLS algorithm suggests a way to train a machine learning model using the optimised QPS parameter values as input features.

B. **Objective**

Conduct a literature review on past and modern work on sleep spindle detection/classification. Implement a streamline and automatic algorithm to extract QPS parameters from small segments of Stage N2 epochs via *non-linear least squares* (NLLS) from a patient EEG to be used as features in a machine learning model. Analyze the results in order to assess the feasibility of the NLLS in the classification.

C. **My solution**

1.0 second moving frame to extract spindle and non-spindle samples
Using NLLS to perform regression of the QPS model to the raw captured spindle
Using the SDT ratio as the basis for parameter initialization for spindle and non-spindle samples.
Using a feedforward artificial neural network to perform binary classification.

D. **Contributions** (at most one per line, most important first)

Developed a streamline algorithm to extract QPS parameters via NLLS coupled with an artificial neural network.
Efficient implementation that does not require any form of 2-dimensional deep learning (convolutional neural network)
Verified the unfeasible nature of the NLLS in differentiating between spindles and non-spindle samples.
Compared method used in this study to existing methods from literature review.

E. **Suggestions for future work**

Coupling the QPS model and NLLS with more reliable spindle detection methods such as those involving time-frequency analysis
Developing an analytical method to QPS parameter estimation as an alternative to initializing parameters that need to be derived a priori.

While I may have benefited from discussion with other people, I certify that this report is entirely my own work, except where appropriately documented acknowledgements are included.

Signature:  Date: 28 / 11 / 2019

Pointers

List relevant page numbers in the column on the left. Be precise and selective: Don't list all pages of your report!

34	Problem Statement
35	Objective

Theory (up to 5 most relevant ideas)

39	Gradient Descent Algorithm
40-41	Levenberg-Marquadt Algorithm
42-47	Neural Networks and Backpropagation Algorithm
51-52	Multitaper Power Spectral Density Calculation
53-54	Dataset Processing & Feature Selection Via T-Test

Method of solution (up to 5 most relevant points)

48	Develop a script to extract stage N2 epochs from 8 hours PSG recordings
49-52	Develop an automatic algorithm to extract spindle/non-spindle samples from EEG using a moving frame acquisition method and use NLLS using SDT ratio as the initialisation criterion.

Contributions (most important first)

71-72	Determined that NLLS is effective when scorer annotations are used as the basis for parameter initialization
73-74	However, using other fair NLLS initialization schemes independent on annotations makes NLLS unreliable.
55-60	Showed the baseline distribution of spindle/non-spindle values in the ideal classification scenario.

My work

48-49	System block diagrams/algorithms/equations solved
55	Description of assessment criteria used
48-54	Description of procedure (e.g. for experiments)

Results

55-70	Succinct presentation of results
55-70	Analysis
71-74	Significance of results

Conclusion

77	Statement of whether the outcomes met the objectives
75-76	Suggestions for future research

Literature: (up to 5 most important references)

31 – 33	[30] A. J. Palliyali, M. N. Ahmed, and B. Ahmed
28 – 30	[39] P. M Kulkarni, Z. Xiao, E. J Robinson, A. Sagarwa Jami, J. Zhang, H. Zhou, S. E Henin,A. Liu, R. S Osorio, J. Wang, and Z. Sage Chen
28 – 30	[32] S. Devuyst, T. Dutoit, P. Stenuit, and M. Kerkhofs,

Acknowledgements

I would like to acknowledge my supervisor, Dr. Beena Ahmed for her guidance and support throughout the entire duration of this thesis. I want to thank you for all the constant support you provided during the year via the weekly meetings and emails back and forth, despite our whereabouts, in the pursuit of results. The ingenuity you provided in the times where progress seemed less than optimistic drove me further to improve on the methodology and constantly experiment. I also cannot thank you enough for the little conversations about life in general that we had in the weekly meetings. You truly kept my sanity in check and always preached that *we all must have our failures in order to learn and progress.* This is something I often overlook, but after this, I do not think I will ever forget these words.

I also want to acknowledge my friends for the support they have provided me throughout the duration of the thesis and throughout the entirety of my academic years. Finally, I want to acknowledge my family for their support and reassurance they provided throughout this thesis. I want to thank them for all the times they consoled me through the hardships that I encountered time and time again, for the times they had sit through my explanations of signal processing, statistics and data pre-processing as I vented through my frustrations. Many hardships were experienced and I want to thank them for helping me through the trials and tribulations that was this degree.

Abstract

In this final thesis report, a novel algorithm is developed and explored that uses the *Quadratic Parameter Sinusoid* model developed by Palliyali et al is used as the main source of features to be used to perform binary classification of sleep spindles found in Stage N2 sleep in EEG. These features are six coefficients that independently control the morphology and frequency characteristics of a gaussian-pulse waveform that mimics the non-stationary nature of sleep spindles. The feature extraction process requires the implementation of *non-linear least squares* (NLLS) regression via the Levenberg-Marquadt Algorithm that seeks the minima of the cost function as fast as possible. However, the NLLS is a sensitive process and requires proper parameter initialisation for optimal fitting of a statistical model. This study has shown that the NLLS is effective at separating spindles from non-spindles under the assumption that the location of spindles is accurately known but fails to statistically differentiate when using an unbiased power-based criterion in the NLLS parameter intialisation.

Contents

Abbreviations

AASM American Academy Of Sleep Medicine

ANN Artificial Neural Network

MLP Multilayer Perceptron

SVM Support Vector Machine

STFT Short Time Fourier Transform

DWT Discrete Wavelet Transform

NLLS Non-Linear Least Squares

QPS Quadratic Parameter Sinusoid

Chapter 1

Introduction

1.1 Definition Of A Sleep Spindle

Sleep spindles are rhythmic transients that are produced as a result of neuron activity in the thalamocortical system [1], a system that describes all forms of communication between the thalamus and the rest of the brain [2]. Such transients and oscillations are measured and visualised via *electroencephalographic* (EEG) recordings. Special attention was placed on oscillations that exhibited a waxing and waning structure which we were named *sleep spindles* [3].

The definition of a sleep spindle was first defined by Allan Rechtschaffen and Anthony Kales in 1968 [4] with continual revisions to the definition made in the subsequent years. The American Academy of Sleep Medicine (AASM) have standardised the definition of sleep spindles to be [5]:

"... oscillatory bursts on EEG of 11-16Hz sinusoidal waves, with duration of 0.5-2s and waxing and waning envelope"

Figure 1.1 shows how a sleep spindle would appear as on an epoch of a human EEG recording. How the recording is performed will be described in detail in *Section 1.2*.

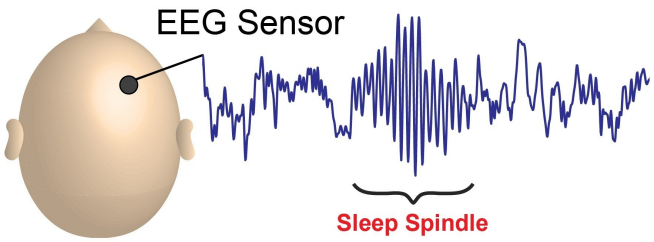


Figure 1.1: Sleep spindle in an EEG signal

1.2 Sleep Spindles As Neurological Markers

1.2.1 Sleep Stages

Sleep spindles are one of many types of oscillations and transients that occur on an EEG recording at various stages of sleep. The stages of sleep can be divided into 5 overall stages that periodically occur in cycles which can be summarised in Figure 1.2 in the form of a *hypnogram* [6]. Each stage of sleep can be graphed over the course of the total sleep time. The following hypnogram assumes that an individual has had an average of 8 hours of sleep.

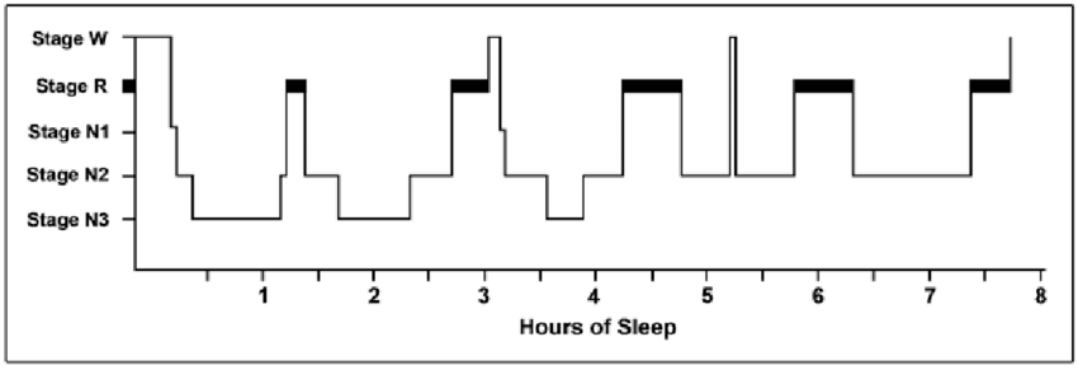
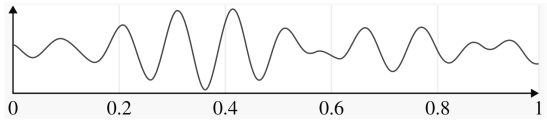


Figure 1.2: Sleep stages on a hypnogram for an average 8 hour sleep period [7].

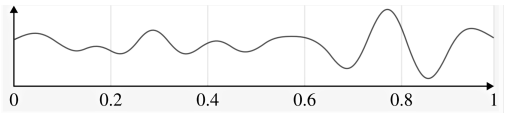
Stage 1 (N1)

This stage is a stage of light sleep. In this stage, medium to fast *alpha* and *theta* waves are produced which are oscillatory EEG signals with frequency ranges 8-13 Hz and 4-8 Hz respectively [8]. The individual is still alert in this stage of sleep and can be said to be in a state of drowsiness. Overall, the frequency spectrum of EEG signals in this stage are shifted

to lower frequencies alongside an increase in the amplitude of cortical waves. These alpha and beta waves are shown in Figure 1.3a and Figure 1.3b: [9].



(a) Alpha waves (8-13 Hz) [10]



(b) Theta Waves (4-8 Hz) [10]

Figure 1.3: Morphology of the alpha and beta waves in EEG

Stage 2 (N2)

The drowsy period of Stage 1 is followed by Stage 2 which is still a part of light sleep and is characterised by a further decrease in frequency of EEG waves and an increase in their amplitude. Intermittent and periodic bursts of neuron activity between 10-12 Hz with a period of 1-2 seconds occur in the EEG signal [9]. These signals are, of course, consistent with the AASM definition of sleep spindles that was defined in section 1.1.

Accompanying sleep spindles are biphasic slow waves that exhibit a sawtooth shape. Such waves have been named *K-complexes*. While their clinical significance is still indeterminate, it is believed to be prevent any form of intrusive arousal and promotes sleep maintenance. K-complexes generally last for 0.5 seconds and have a peak-to-peak amplitude of $75\mu V$ [11]. Figure 1.4 shows an EEG segment containing sleep spindles and an instance of a K-complex.

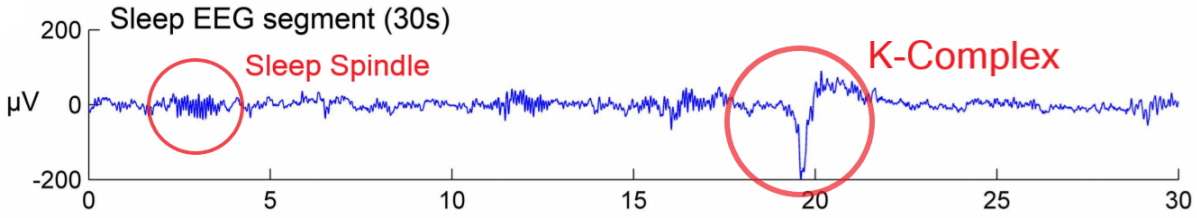


Figure 1.4: Sleep Spindles and a K-Complex [12]

Stage 3/4 (N3 and N4)

Stages 3 and 4 are grouped in the period of moderate to deep sleep. In Stage 3, the frequency of sleep spindle occurrences gradually decreases while the amplitude of low-frequency EEG components increases. This then progresses to the last stage of *non-Rapid Eye Movement* (or *non-REM*) sleep, Stage 4. The dominant oscillations that occur in Stage 4 are known as *delta waves*, fluctuations in EEG that have frequencies less than 3 Hz [8]. Stage 4 is, hence, also named *delta sleep* after the dominant EEG fluctuations in this stage or *Slow Wave Sleep* (SWS). Figure 1.5 shows the typical morphology for a delta wave.

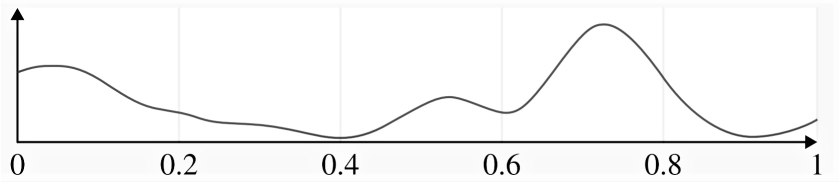


Figure 1.5: Delta Wave ($f < 4$ Hz) [10]

Clinical studies have shown that it is difficult for an individual to be awakened in the stage of deep sleep. Characterised with the greatest arousal thresholds relative to all other stages of sleep, it has been shown that sound disturbances greater than $100dB$ will not awaken an individual who is in SWS. [13]. Waking from SWS generally gives rise to moderate impairment in mental performance and indicates the *need* from SWS. This phenomenon is called *sleep inertia*.

REM Sleep

Rapid Eye Movement Sleep (or REM sleep) proceeds Stage 4. The REM sleep stage is indicated in red in Figure 1.2 and occurs in 90 minute cycles after falling asleep [14]. This stage is characterised by irregular and erratic breathing and an increase in the heart rate.

However, signals are sent from the brain to the spinal cord rendering skeletal movement atonic or without movement as a form of temporarily paralysis [13]. EEG recordings during REM sleep are similar to EEG signals during the wake state [11] and, hence, it is believed that the muscle paralysis is an evolutionary trait to ensure the individual does not injure

themselves during the wake state. Furthermore, REM sleep is critical for consolidating learned and processed information from the day prior to be stored in long-term memory. The stage is also associated with *dreaming* though this has not been yet fully understood.

1.2.2 Signals in EEG Activity & Recording Method

The recording of EEG signals is achieved through a process called *polysomnography* (or PSG). Often referred to as a *sleep study*, a PSG study is a multi-parametric test which measures different types of physiological signals during sleep [15]. A polysomnographic test can also measure the following types of signals (other than EEG).

- Electrocardiogram (ECG) - Heart rate and rhythm
- Electrooculogram (EOG) - Eye movements
- Electromyogram (EMG) - Chin and leg muscle activity
- Blood oxygen level (Oximetry)

Electroencephalographic measurements utilise devices that consist of electrodes with conductive material, amplifiers with filters, analog-to-digital (A/D) converters and a recording device such as a computer which stores and displays the signal measurements as graphs. [16]. A conductive gel or paste is used as an electrically conductive medium between the scalp, which can be thought of as an ionic solution [8] and the electrode.

These electrodes come in different forms such as needle electrodes, saline-based electrodes, reusable disc electrodes made of gold, silver or tin or in the form of an electrode cap. When capturing multi-channel signals where multiple electrodes are required an electrode cap is typically used [16] where the electrodes used are Ag-$AgCl$ electrodes used in conjunction with conductive paste.

10-20 System

The placement of the electrodes on the scalp is not arbitrary and follows the de-facto standard known as the *international 10/20 system* developed by psychologist, Herbert H. Jasper in 1958 [17]. The system describes the recommended positioning of the electrodes at certain cranial landmarks on the scalp [18] where the relative distances between the electrodes should allow for adequate coverage of all parts of the head and should be proportional to the size of the head and the scalp [17]. The electrode positioning convention in the 10-20 standard in shown in Figure 1.6.

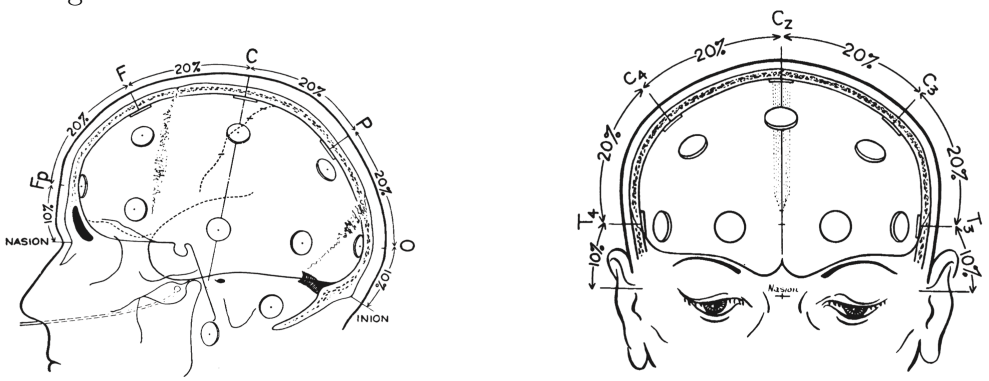


Figure 1.6: Side (LEFT) and front (RIGHT) view of electrode positioning using the 10-20 system.

The diagrams in Figure 1.6 indicate the relative positioning and distances of the electrodes as percentages but, more importantly, letters that indicate what *position* an electrode is located at with respect to the cranial landmarks. Table 1.1 indicates what each symbol refers to [19].

Letter/Symbol	**Lobe**
F	Frontal
Fp	Prefrontal
T	Temporal
C	Central
P	Parietal
O	Occipital

Table 1.1: Symbol assignment to electrodes positioned at particular lobes of the brain

Each letter has an associated subscript. Electrodes with subscript 'z' (which stands for 'zero') refer to electrodes positioned on the midline of the scalp (e.g. C_z). The Fp_z electrode is often allocated as the ground/reference electrode [16]. Subscripts $i \in \{2, 4, 6, 8...2n\}$ refer to electrodes positioned on right hemisphere (e.g. Fp_2) while subscripts $j \in \{1, 3, 5, 7...2n+1\}$ refer to electrodes positioned on the left hemisphere (e.g. T_3) [19]. Figure 1.7 shows the top view as a more complete view of the possible electrode positions on the scalp. The *general* electrode configuration for a system such as the 10-20 system is often referred to as a *montage* [16].

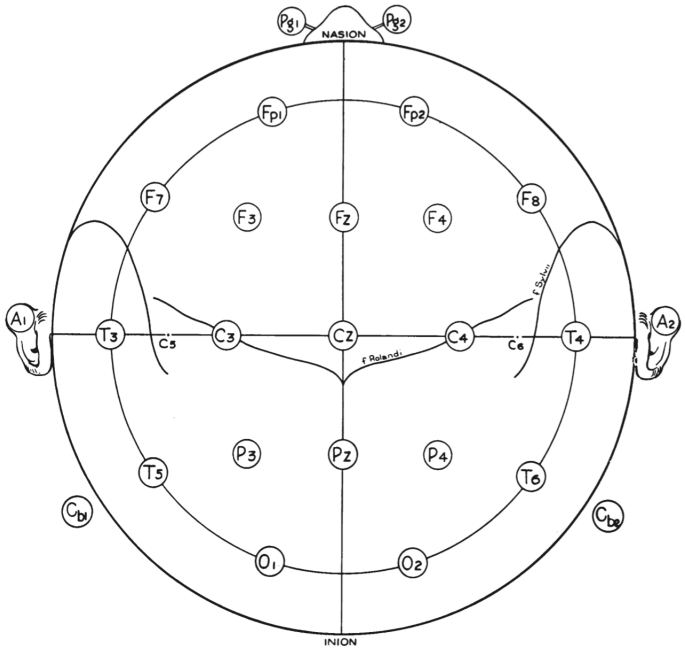


Figure 1.7: Top view of electrode positioning using the 10-20 system

The measurements taken from the PSG device/unit are visualised on a PC as a multi-channel PSG recording such as that shown in Figure X. The recordings are plotted as voltage-time graphs where the voltages are typically in μV while time is measured in seconds. Below are some other requirements for the measuring equipment [16]

[1]There is no such *central* lobe. The symbol C is used for identification purposes only.

- Analog filters in the unit should have a high-pass filter with a $3dB$ cutoff frequency between 0.1-0.7 Hz.
- A 12-bit A/D converter in the unit the samples at a frequency between 128-1024 Hz.
- Fast PC with adequate volume for the hard disc has the resulting recordings are gnerally 200MB+ for an 8-channel recording taken for 1 hour.

An example of a PSG recording is shown in Figure 1.8.

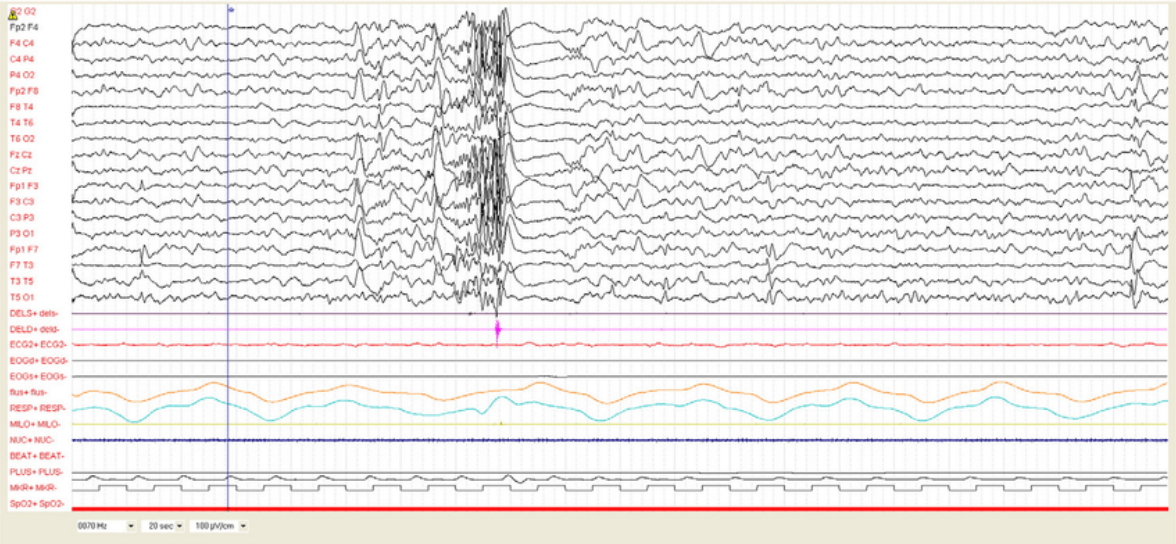


Figure 1.8: PSG recording of a 35 year old woman with Lennox-Gastaut syndrome (LGS) [20]

Importance of Polysomnography In Diagnosing Sleep Disorders

Polysomnography can be used to diagnose patients with particular sleeping disorders. While there are scenarios where sleep studies can be conducted without supervision by a specialist (*Limited Channel Monitoring*), PSG examinations are generally performed in a laboratory setting with trained PSG technicians (also referred to as *sleep technicians*) [15]. Sleep technicians are trained to test and calibrate equipment, understand electrode placement and are able to interpret certain events and anomalies in the recordings and relate these readings to the possible diagnosis of a sleeping disorder.

Polysomnography is typically used to diagnose *Obstructive Sleep Apnea Syndrome* (OSAS), which is the relaxation of the soft palatte muscles causes the airways to be block resulting in snoring and constant gasping of air [21]. Technicians will observe for events in the PSG recording that indicate when the patient has stopped breathing for 10 seconds or longer and the frequency of such events [21].

Other sleep disorders that can be diagnosed via PSG recordings include, but are not limited to: hyper ventilation syndromes, narcolepsy, chronic insomnia, sleep-related seizures etc.

1.2.3 Clinical Significance Of Sleep Spindles

A great deal of research has gone into understanding the rhythmic properties of sleep spindles through various methods such as brain and electrophysiological imaging as well as mathematical and computational modelling to name a few. However, a comprehensive understanding of their functional role in humans and animals still remains incomplete.

However, it is generally believed that these sleep spindles are linked to the individuals quality of sleep, cognition and memory consolidation and the maintenance of conciousness, alertness, and motor development [22]. Studies have also shown that while sleep spindles are consistently generated night-to-night, spindle activity varies considerably between individuals and has been suggested to be an electrophysiological fingerprint of an individual [23]. Spindle activity, however, has been shown to be correlated with the age of a person as well as whether the person has neurological disease or disorder

Effect Of Sleep Disorders On Sleep Spindles

Neurological disorders such as (but are not limited to) epilepsy and schizophrenia influence the characteristics of spindles such as their morphology and shape in NREM sleep. Epilepsies have a significant effect in the sleep-wake cycle and sleep architecture of an individual and occur primarily during NREM stage-2 sleep. Generally, patients who have seizures have a lower spindle frequency count [22]. More specifically, individuals with *generalised epilepsy*, where both hemispheres of the brain are affected from the onset of the epileptic event with awareness impaired [24], have significantly slower rate of spindle generation than those who only go through *partial seizures* [3]. Residual effects of medication such as antiepileptic drugs are a possible reason for the reduction in spindle frequency as well as physiological differences between those with partial seizures or generalised epilepsies.

Studies of patients diagnosed with schizophrenia have also shown a marked reduction in sleep spindle count in the prefrontal, centroparietal and temporal regions of the brain [22]. Along with the reduction in spindle count was a reduction in the amplitude and power in the sigma band (12-16 Hz) showing a strong correlation between the reduction in spindles and the severity of the schizophrenia [25]. These studies conclusively determined that antipsychotic medication [22] was not the main cause for the deficit in spindles but purely due to the abnormalities in the thalamocortical system, inhibiting the transfer of neurons to and from the thalamus responsible for cognition and memory consolidation [25].

Effect Of Age On Sleep Spindles

Ageing results in long-term degenerative changes to the brain. As such, this should influence the brain connectivity and, hence, the morphology and characteristics of sleep spindles [**?**]. The decrease in spindle density is not uniform across the entire cortex but prominent in the frontal and occipital lobes as the an individual ages as white matter in the brain reduces [26]. More importantly, ageing results in the decrease in volume of the thalamus, thus altering its connectivity with the cortex and, hence, spindle generation. This decrease in spindle density is shown in Figure 1.9 as a function of age. The general trend also shows that while there is a decrease in spindle count for older patients, the centre frequency of the spindles shifts to higher frequencies as one ages. This is shown in Figure 1.10 as a graph of spindle frequency as a function of age.

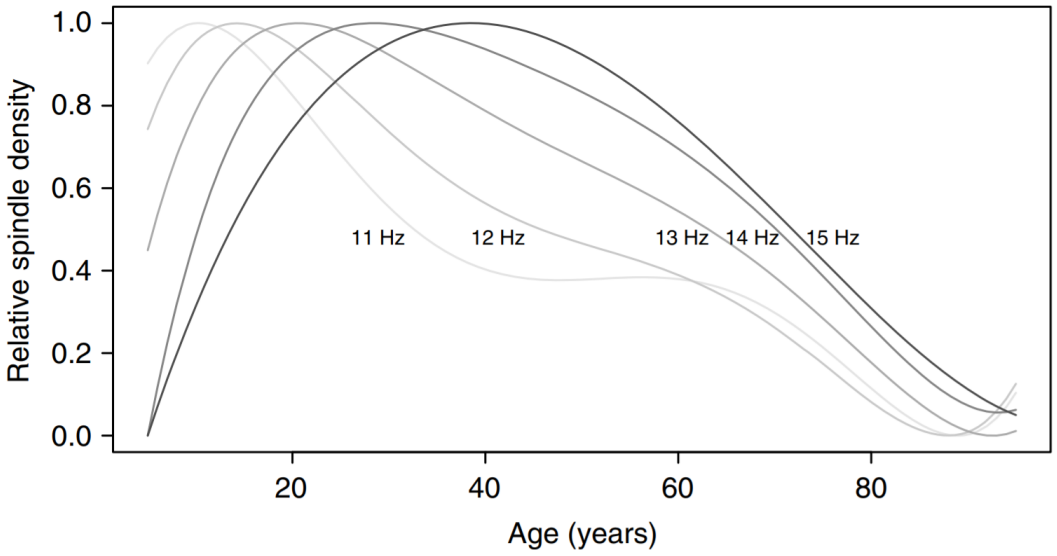


Figure 1.9: Spindle density (for different spindle frequencies) as a function of age [27]

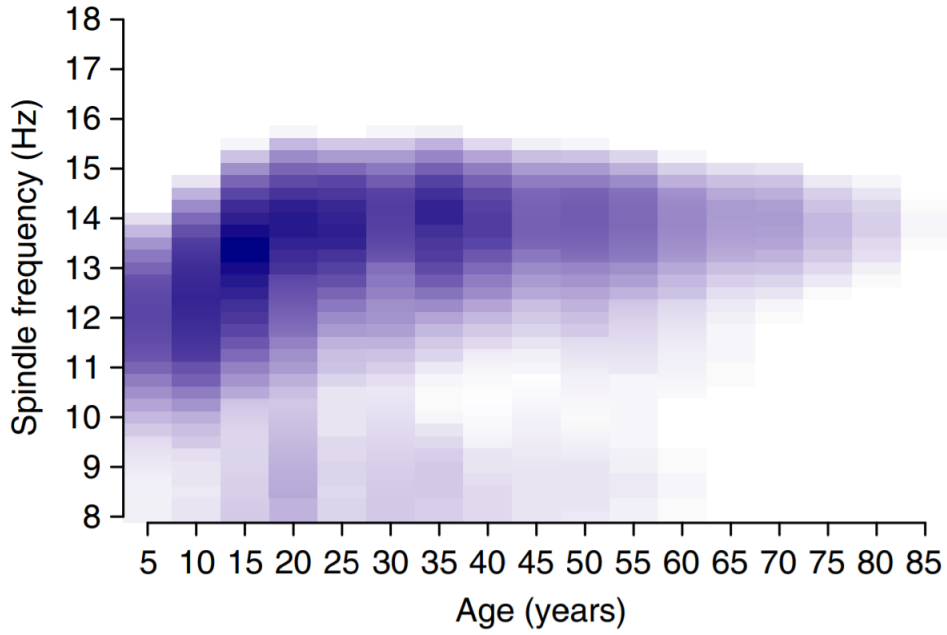


Figure 1.10: Heatmap showing spindle frequency at different ages [27]

1.2.4 Spindle Detection & Need For A Sleep Spindle Model

The relationship between sleep spindles and their morphological alterations as a result of neurological disorders such as seizures and epilepsy as well as diseases such as schizophrenia, Parkinson's and Alzheimer's has been gradually understood over the years. Hence, spindle detection has become an imperative task as they are representive as *biomarkers* for these neurological anomalies as well as acting as a supplementary diagnostic value [28]

At present, sleep technicians are used as the *gold standard* (experts) in spindle detection and manual detection of these spindles has become the commonly excepted method. However, the manual scoring of spindles, unsurprisingly is a tedious task, especially when analysing PSG recordings for numerous patients where each recording can span to 8 hours or more. The manual scoring of spindles also introduces human bias and error into the process which influences the accuracy of true positive spindle findings. The tedious nature of manual spindle detection has motivated numerous studies regarding the possible unbiased *automation* of spindle scoring and classification [29] that has continued till today.

Methods of automatic detection have utilised techniques such as *Short-Time Fourier Transforms* (STFT), wavelets, sparse optimisation and filter banks [30] that will be elaborated further in the literature review in *Chapter 3*. Recent developments in machine learning and deep learning algorithms have led to the use of artificial neural networks that have seen significant results. These methods and their outcomes will be discussed further in the literature review as well.

The other underlying issue is understanding the morphology of sleep spindles. Spindles are fundamentally non-stationary [30] meaning that their statistical features change over time. In this case, an EEG signal would qualify as a non-stationary signal as its frequency content changes over the course of the EEG recording (e.g. the frequency spectrum shifts to lower frequencies during Stage N3 and N4 sleep) [31]. Sleep spindles, in particular, vary in frequency from 11-16 Hz throughout the period of time they are generated in sleep. Furthermore, their

amplitude and phase characteristics are also not uniform throughout the entire sleep period. It is clear that their classification, let alone their detection is a non-trivial task that requires analytical techniques and/or mathematical models that can adequately describe their rich temporal features while keeping in mind their non-stationary nature.

A mathematical model that serves as a suitable candidate in describing the characteristics of such sleep spindles is the *Quadratic Parameter Sinusoid* (QPS). Introduced by Palliyali et. al [30], the proposed model is defined as a sinusoidal carrier modulated by a gaussian envelope whereby both the carrier and envelope are controlled by time-dependent quadratic polynomials:

$$s(t) = e^{(a+bt+ct^2)} \cdot cos(d + et + ft^2) \tag{1.1}$$

As the centrepiece of this thesis, the QPS carries the waxing and waning shape of a spindle whilst having the property of mimicing the minute frequency and phase perturbations that other existing models cannot replicate. The QPS is also able to produce asymmetry in its envelope structure that is common in most sleep spindles that occur naturally. We hope that such mathematical properties of this model are able to be utilised as robust learning features for a machine learning model that will be able to classify spindles and non-spindles in EEG. This will be formally discussed at the end of the literature review in *Chapter 3* after an in-depth analysis on past work and current state-of-the-art implementations of spindle detection and classification.

Chapter 2

Literature Review

Spindle detection and classification algorithms have been developed as early as the 1970s [32]. These methods have generally involved time-frequency signal processing methods of threshold-based conditions that allow for the extraction of temporal information on spindles on a small time-interval as the back-end process, some of which have their advantages and disadvantages. These algorithms may be coupled with some form of machine learning algorithm as the front-end analysis after data-pre-processing in order to perform either a final binary or multi-class classification with iterative training for a machine learning model.

2.1 Time-Frequency Analysis Methods

Spindles have time-varying frequency within the 11-16 Hz. This means that a single Fourier Transform of the whole time-domain EEG recording to the frequency domain gives us a limited perspective on the non-stationary frequency behaviour of sleep spindles within this band. We can see the time-independence from the equation for the Fourier transform for some discrete time-series signal, $x(t)$:

$$X(\omega) = \mathscr{F}\{x(t)\} = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt \tag{2.1}$$

However, an inherent issue with this is the loss of all temporal information. That is, the mapping from the time-domain to the frequency-domain does not indicate when each frequency

component of a signal occurs in a given signal. For this reason, the Fourier transformation is only useful in the case where the signal being analysed is stationary and periodic for all times and frequencies. This loss of temporal information is shown in the time-frequency diagrams in Figure 2.1.

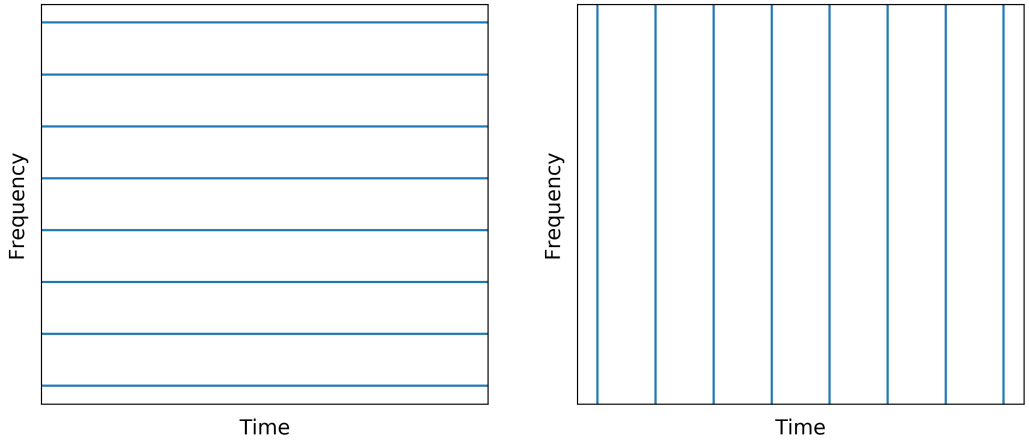


Figure 2.1: (LEFT) Only frequency information retained when Fourier transformation is performed. (RIGHT) Only temporal information is available when analysing the signal in the time-domain

2.1.1 Short-Time Fourier Transform (STFT)

To account for this for the lack of simultaneity in time and frequency information, one of the techniques used is the *Short-Time Fourier Transform*. A windowing function, $w(t)$ is used that is time-limited and time-shifted along the duration of the signal being analysed and multiplied with the signal for all time $t \in \mathbb{R}$. This process is essentially a time-domain convolution operation and can be expressed as per Equation 2.2.

$$y(t) = (x \star w)(t) \tag{2.2}$$

$$= \int_{-\infty}^{\infty} x(\tau) w(t - \tau) dt \tag{2.3}$$

In discrete-time where $t = nT_s$ where T_s is the sampling period, the convolution can be written alternatively as:

$$y[n] = \sum_{n=1}^{N} x[k]w[n-k] \tag{2.4}$$

The Fourier Transform can then be used for the time-limited segment of the signal that we want to obtain the frequency information of. The combined process of windowing and performing the Fourier Transform is named the *Short-Time Fourier Transform* or STFT. We can expressed this mathematically as:

$$X(\omega) = \mathbf{STFT}[x(t)] = \int_{-\infty}^{\infty} x(t)w(\tau - t)e^{-j\omega t}dt \tag{2.5}$$

A similar operation can be performed in discrete-time. The benefit of the STFT can be shown in the spectrogram in Figure 2.2 where both frequency and time information are simultaneously represented in the diagram. The window function also ensures that the frequency spectrum of the windowed signal does not exhibit spectral leakage. Plotting the frequency of the signal against time, a spectrogram is produced that shows the peak frequency at each time-interval based on a colour mapping where red spots are indicative of a large amplitude at that frequency in that time instance.

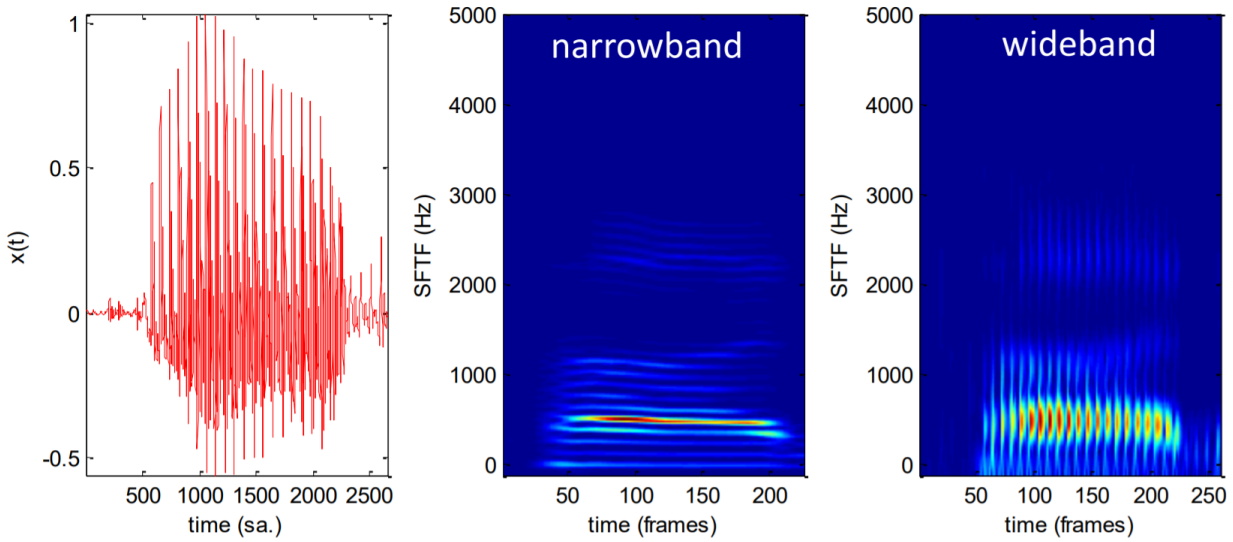


Figure 2.2: Spectrogram output of frequencies detected in a speech recording via STFT. The frequency resolution is dependent on the window size of $w(t)$ [33]

Past Work Using STFT For Sleep Spindle Detection/Classification

In 2002, Gorur et al [34] used the STFT for the purposes of feature extraction using the Hamming window to isolate time-localised sections of EEG data. The coeffcients generated between 2 - 64 Hz was used to generate a feature vector with an associated label for the feature vector based off of the visual stage annotations provided in the dataset (1 for spindles and -1 for non-spindles). The size of the window overlap was made less than the window length in order to generate more 'spindle' samples in the resultant dataset. The dataset was then used to train and test an MLP with an 32/60/1 architecture with the hyperbolic tangent (*tanh*) used as the activation function at the output to generate a ± 1 output for spindles and non-spindles.

The *tanh* function was also used as the activation function for the hidden layer neurons with backpropagation used to minimise the objective function which was the mean-squared error (MSE). After training the MLP using 1000 spindle and non-spindle samples and 1142 samples for the test dataset and using a 10-fold cross validation process for 13,000 epochs, they found that the training and test accuracies were 99.3% and 88.7% respectively. They then performed a similar experiment using the *Support Vector Machine* (SVM) model which picks two data points from the classes in the dataset to act as *support vectors* that best separates the classes via a hyperplane called a *kernel*. Using a Gaussian kernel and the same train-test split ratio for the dataset, they found a noticeably superior performance of 95.4% accuracy using the test dataset.

2.1.2 Continuous and Discrete Wavelet Transform

While the STFT can be used to determine the time-localised frequency content for a small segment of the signal uniformly, the *size* of the window is fixed for *all* frequencies. That is, the time and frequency resolution is fixed for the entire analysis. The time-frequency resolution for the STFT is shown in Figure 2.3.

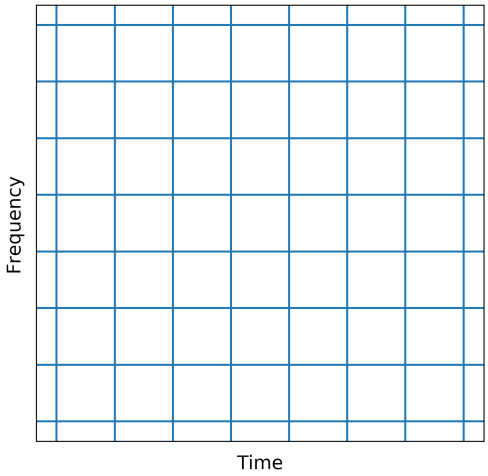


Figure 2.3: Time-frequency resolution for the STFT

The constraint between window size and frequency resolution is named the *Heisenberg Uncertainty Principle*. In the context of time-frequency transformations, this principle stipulates that it is not possible to achieve good time resolution and good frequency resolution [35].

The *wavelet transform* in its continuous and discrete variations accounts for the restriction in the window size by using a *wavelet* as the basis for the transform as opposed to sines and cosines. These wavelets are mathematical objects that exhibit a waxing and waning envelope with a carrier component as shown in Figure 2.4. Unlike the STFT, however, the ability of the wavelet to dilate offers variability in the time-resolution at different frequencies. The time-frequency resolution characteristics for the wavelet transform is shown in Figure 2.5.

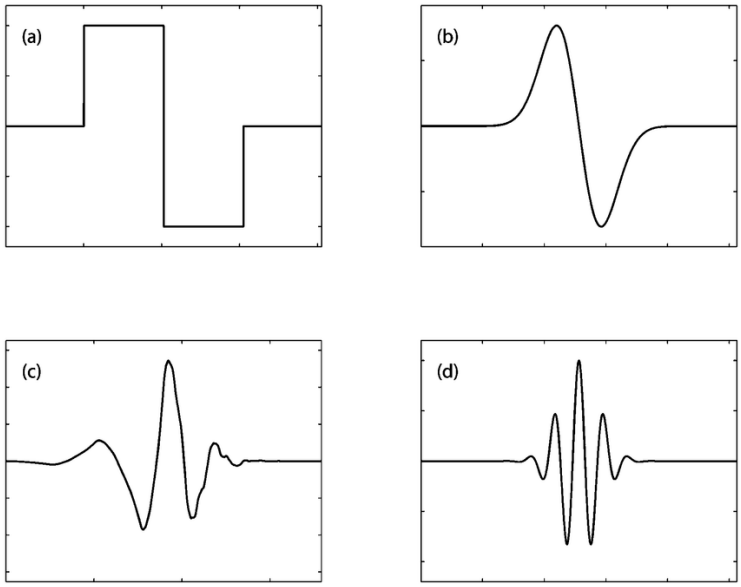


Figure 2.4: (a) Haar Wavelet, (b) Gaussian wavelet of order 1, (c), Daubechies wavelet of order 4, (d) Morlet wavelet [36].

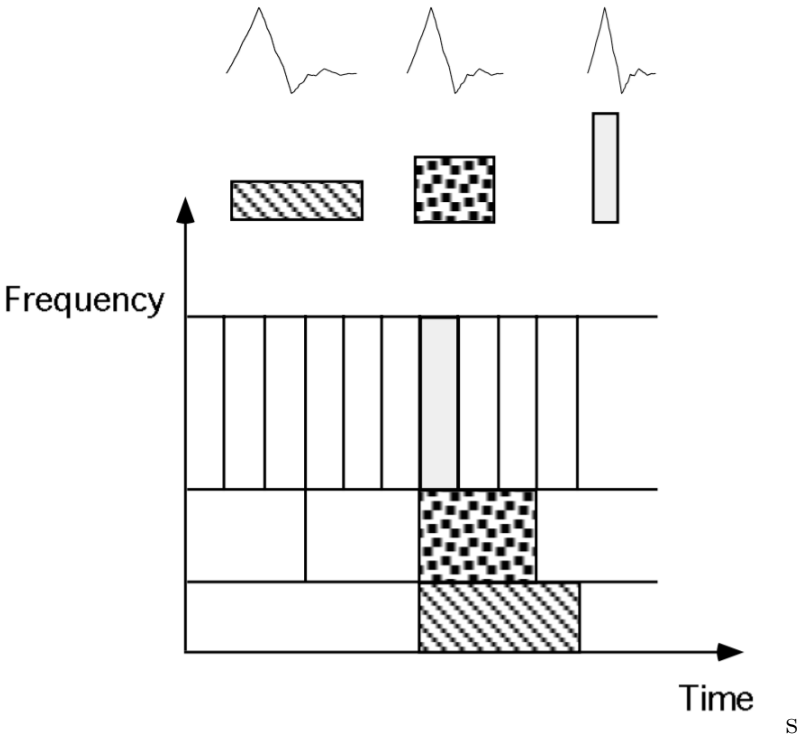


Figure 2.5: Time-frequency resolution when using Daubechies wavelets as windowing functions [37]

The wavelet is used in a similar fashion to the window function used in the STFT. Let $\psi_{a,b}(t)$ denote the wavelet kernel. Then, the wavelet transform in the continuous time-domain takes the general form shown in Equation 2.6

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\psi^* \left(\frac{t-b}{a} \right) \tag{2.6}$$

Where the * denotes complex conjugation $a \in \mathbb{R}$ is the dilation (scaling) factor, $b \in \mathbb{R}$ imposes a time-shift and the $\frac{1}{\sqrt{a}}$ factor ensures the wavelet is normalised given some dilation due to a. Where no scaling or time-shifting has been applied ($a = 1$ and $b = 0$), the wavelet is referred to as the *mother wavelet* or $\psi_{1,0}(t)$ while the *daughter wavelets* are time-shifted and scaled versions of the mother wavelet. The scaling of the wavelet is inversely related to its frequency [35]. The relationship is approximated with Equation 2.7.

$$a = \frac{F_c \cdot T_s}{f} \tag{2.7}$$

Where f is the frequency of the signal, T_s is the sampling period and F_c is the centre frequency of the basis function. Observations for high frequencies correspond to low scaling values. Then, the *Continuous Wavelet Transform* (CWT) operation is the convolution between the signal being analysed, $x(t)$ and the wavelet. The CWT is mathematically expressed via Equation 2.8.

$$W(a,b) = \int_{-\infty}^{\infty} x(t)\psi^* \left(\frac{t-b}{a} \right) dt \tag{2.8}$$

In essence, the CWT produces coefficients depending on the correlation between the frequency of the signal being analysed and the scale of the wavelet. The higher the correlation is, the larger the generated coefficient will be at that time instant for that scale/frequency. Like the STFT, a spectrogram can be generated that reflects the prominent frequencies in a time-segment of a signal.

Past Work Using The CWT/DWT For Sleep Spindle Detection/Classification

Though the CWT is not necessarily a novel tool currently, contemporary research has aimed to utilise the time-frequency resolution benefits of the CWT in new algorithms. Lajnef *et al* (2015) [38] to separate oscillatory components of an EEG signal (possibly containing sleep spindles) from transient components (which are most likely K-complexes that occur in the EEG signal) by first tuning the Q-factor of the wavelet and then performing a discrete wavelet transform (called a *Tuneable Q-Factor Wavelet Transform* or TQWT) and then using sparse optimisation to reconstruct the oscillatory and transient components using wavelet coefficients from the TQWT process. The reconstructed oscillatory components of the EEG are shown in Figure 2.6.

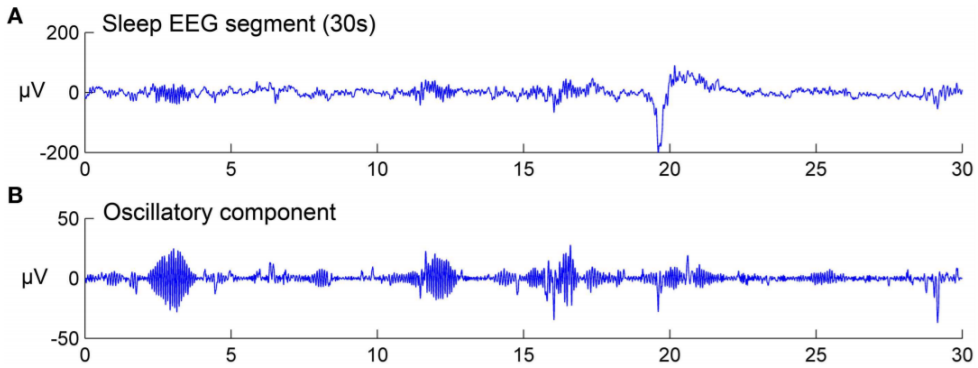


Figure 2.6: (A) A 30 second EEG segment. (B) The oscillatory component of the EEG segment reconstructed [38]

Lajnefet et al used a Q-factor of 5.5 to isolate frequencies in the AASM defined spindle frequency range (11-16Hz). A CWT was then used on the oscillatory components using the B-Spline wavelet to detect the spindles using maxima detection and showed the highest mean cross correlation coefficient across other wavelets used. Figure 2.7 shows the outcome of the process in the form of a spectrogram where the detected maxima signifying large wavelet coefficients at the particular frequency that matches the characteristics of the basis for the CWT.

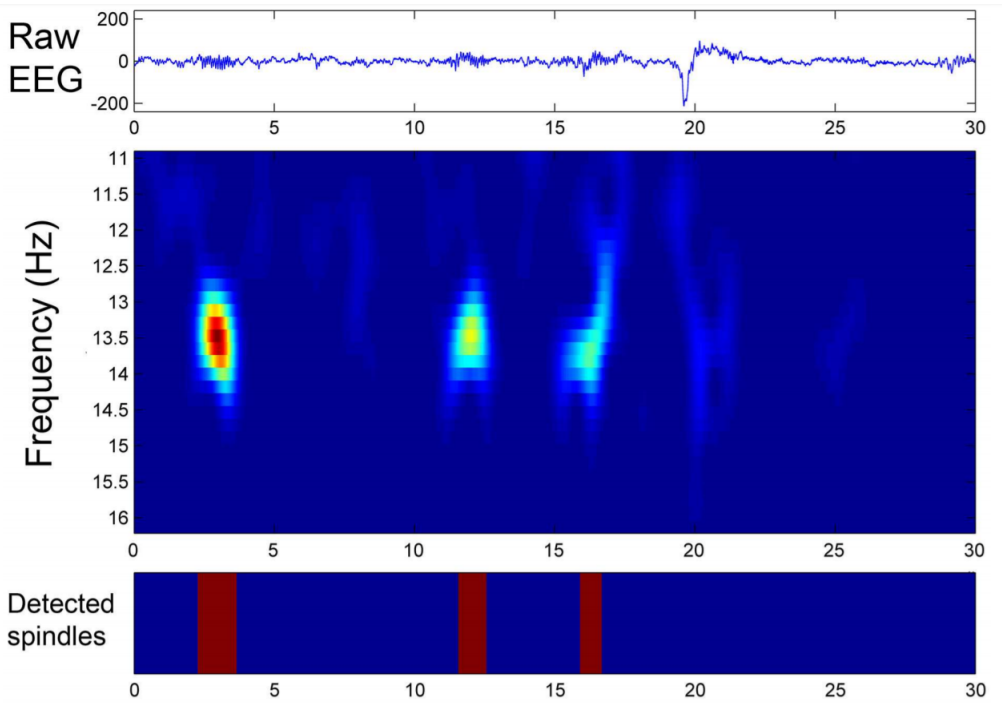


Figure 2.7: CWT used to detect sleep spindles from the oscillatory component obtained from the TQWT decomposition. [38]

2.2 Threshold-Based Detection/Classification Methods

Alternative methods have been developed that use adaptive threshold algorithms or energy criterion as the basis for spindle detection and/or classification. Devuyst et al [32] developed a heuristic method where they used the STFT along with a Hanning window to obtain the localised power spectral density (PSD, which we will denote by S) between 0.5 to 40 Hz, by squaring the magnitude of the STFT. The absolute spindle power in the window at each time-interval was then computed within the 11-15 Hz range. Equation 2.9 was then used to calculate the relative spindle power (RSP) which is the ratio of the absolute spindle power to the total bandpower in the 0.5 to 40 Hz range:

$$RSP(t) = \frac{\int_{11}^{15} S(f,t)df}{\int_{0.5}^{40} S(f,t)df} \tag{2.9}$$

The motivation for Equation 2.9 comes from the fact that it should be expected that locations where spindles are generated in the EEG should have a higher relative spindle power than anywhere else in the recording.

The database used was recorded from the *Sleep Laboratory of the Andre Vesale Hospital* and Belgium that comprised of six whole night recordings from 3 men and women with ages between 31 to 54 with different pathologies. Annotations from two expert scorers were used as the ground truth with 289 spindles detected by Scorer 1 and 409 by Scorer 2. The best possible metrics for this algorithm was a sensitivity of 70.20%, a false positive rate of 1.38% and a specificity of 98.62% when the *union* of the scorers was used. Using the intersection of the scorers as the basis for the ground truth showed an increase in the number of false positives. Devuyst et al concluded that though they do no deny the existence of more powerful algorithms, they encouraged future work on spindle detection to use the algorithm as a base for comparison.

Kulkarni et al [39] published a recent paper where they used the relative spindle power as a feature for a machine learning model to be used in real-time (online). They used the *Montreal Archive Of Sleep Studies* (MASS) and the DREAMS database as the source of spindle and non-spindle samples. This was performed by using a moving frame with a frame size of $T_w = 250$ ms that moved along the entire EEG signal with an overlap of 1 sample. In each frame several features such as raw EEG spindle and baseline obtained via an 11-16 Hz bandpass and notch filter were fed through a convolutional neural network and a recurrent network which was named *Network 1. Network 2* contained the same neural network models but the input features were the envelope of the extracted spindle and baseline. The relative spindle power compared to the delta (0.5-4 Hz) and the theta (4-7 Hz) components was computed via the following ratio in Equation 2.10. In this thesis, this ratio is used as an energy criterion and is named the *Spindle To Delta & Theta Ratio* or the *SDT Ratio*:

$$\text{SDT Ratio} = \frac{\text{spindle band}}{\text{delta band} + \text{theta band}} \tag{2.10}$$

The bandpowers for the spindle, delta and theta frequency bands were computed by computing the multitaper PSD. The output from Network 1 and 2 as well as the SDT ratio were fed into a fully connected network and a softmax activation function was used to produce a binary output that determined whether or not the input corresponded to a spindle or non-spindle.

Using the visual scorer annotations provided in each database, they generated synthetic spindles using the QPS model [30] by estimating values for each of the quadratic polynomials via non-linear regression and adding the broadband (non-spindle) components to the QPS spindle. These served as part of the training samples with appropriate labelling along with the spindles found via the frame acquisition procedure.

The result of their real-time online implementation named *SpindleNet* was assessed using different databases. Using Stage N2 epochs from MASS database with 5-fold cross validation, evaluated the models performance to have a sensitivity score of $90.17 \pm 2.16\%$, F1 score of

0.75 ± 0.05 and an AUC score of $98.97 \pm 0.13\%$. Using the DREAMS database as the unseen dataset used for testing, they calculated the final performance of machine learning model to have an AUC-ROC score of $95.97 \pm 0.96\%$ despite the differences in the spindle statistics between the two datasets.

2.3 Quadratic Parameter Sinusoid (QPS)

The *Quadratic Parameter Sinusoid* (QPS) developed by Palliyali et al [30] in 2015 was proposed as a mathematical model that could generate accurate approximations to the envelope and carrier characteristics of raw sleep spindles in the 11-16 Hz range found in EEG. The QPS is defined via Equation 2.11 as a sinusoidal carrier modulated by a gaussian envelope where both components are controlled by time-dependent quadratic polynomials:

$$s(t) = e^{(a+bt+ct^2)} \cdot cos(d + et + ft^2) \tag{2.11}$$

The analytic version of the QPS can be written for $s \in \mathbb{C}$ using the complex exponential as shown in Equation 2.12

$$s(t) = e^{(a+bt+ct^2)} \cdot e^{j(d+et+ft^2)} \tag{2.12}$$

Parameters a, b and c vary the log-amplitude of the envelope, the rate of change of the amplitude and the shape and duration of the Gaussian envelope respectively. Parameters d, e control the phase shift and the centre angular frequency of the sinusoidal carrier f generates linear perturbations about the centre frequency and is expected to be a small value. To show this, let $\theta(t)$ be the instantaneous phase of the QPS which can be expressed as:

$$\theta(t) = d + et + ft^2 \tag{2.13}$$

Differentiating the instantaneous phase gives the instantaneous angular frequency which can convert to the regular instantaneous frequency, $F(t)$ by division of 2π. This is expressed in Equation 2.14.

$$F(t) = \frac{1}{2\pi} \cdot \frac{d\theta}{dt} = \frac{1}{2\pi} \cdot (e + ft) \tag{2.14}$$

The choice of quadratic polynomials is such that the instantaneous frequency of the QPS is time-varying which mimics the non-stationary characteristic of sleep spindles. Figure 2.8 shows a generated QPS for mean values known a posteriori from Kulkarni [39] and Palliyali [30]. The effect of the changes in the QPS parameters to the morphology of the QPS is shown in Appendix A.

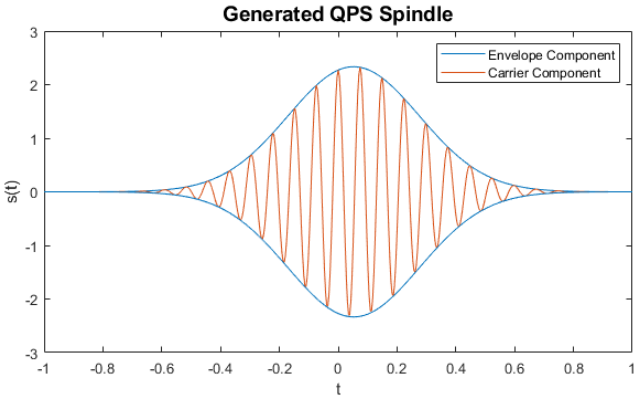


Figure 2.8: A generated QPS using parameter values known a posteriori.

Using the MASS C1-SS2 database as the source of spindle and non-spindle samples from 15 test subjects, their analyses involved assessing how well the the QPS model would fit to raw spindles via *Non-Linear Least Squares* (NLLS) regression after bandpass filtering the raw EEG in the 11-16 Hz range. Figure 2.9 shows the resultant best-fit QPS plotted with a sleep spindle found in the EEG and shows that the QPS model has the ability to perform sleep spindle reconstruction effectively. However, the performance of the NLLS is highly dependent on how close the inital approximations to the QPS parameters to the optimum. To reach the optimum as fast and close as possible, Palliyali et al used the *Levenberg-Marquadt Algorithm* which alters between the *Gauss-Newton* optimisation process and ordinary gradient descent depending on how far the initialisation is to the optimum. This is further discussed in the *Methodology* chapter of this thesis.

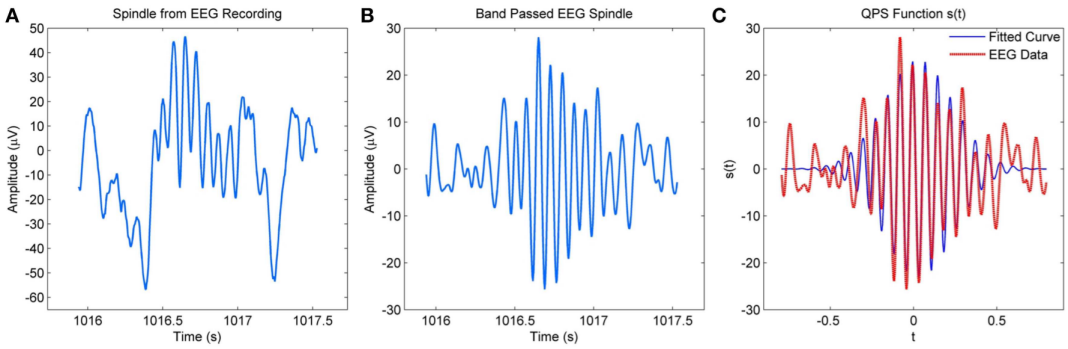


Figure 2.9: (A) Raw spindle from an EEG recording (B) EEG spindle after bandpass filtering (C) Fitted QPS curve onto the filtered spindle [30]

The QPS model was validated through a variety of experiments with simulated and real spindles extracted from the the PSG recordings from the MASS C1-SS2 database. The first experiment involved generating a QPS spindle with pre-set coeffcients for the quadratic polynomials and corrupting the QPS with delta noise in order to emulate a spindle extracted from raw EEG. After bandpass filtering the simulated spindle and performing NLLS regression with a new instance of the QPS, the results show that the mean estimated values exhibited less than a 3% difference for all parameters except for parameters d and f which had percentage differences of 9.7% and 15.6% respectively. Performing the same procedure for multiple synthetic spindles, the distributions showed the greatest variations for parameters c and f which is indicative of the poorer accuracy in their estimation during the NLLS regression.

Their next set of analyses showed that there was a large percentage error between the energy of a best-fit QPS spindle and a spindle extracted from the EEG of a patient from the MASS C1-SS2 database but comparatively smaller percentage error in the estimated frequency of the model to the raw spindle which suggests that the estimate energy from the fitted QPS is an unreliable representation of the real energy of the raw spindle.

Most notably, Palliyali et al also showed that the initialisation of the NLLS parameters prior to performing regression with the QPS plays a large role in the class separation of spindle samples from non-spindles. Using the expert scorer annotations provided in the database, the NLLS regression was performed using the known parameter values known a posteriori as the initial values for the NLLS for spindles found by the scorers and were set to zero for non-spindles manually detected. An independent T-test between QPS parameter values for the spindle and non-spindle samples showed significant statistical difference for *all* parameters at a significance level of 1%. However, it was found that a consistent NLLS initialisation for both spindles and non-spindles showed no difference between spindles and non-spindles. This result was taken into account in the *Methodology* section of this thesis.

Chapter 3

Problem Statement

Investigation of literature has shown a wide range of techniques that have been used to detect and/or classify sleep spindles as well as other oscillations and transients that occur in an EEG recording. We have explored the QPS as a suitable model for sleep spindles due to the range of parameters that capture the main and minute characteristics of sleep spindles. However, the literature has only shown singular aspects being achieved such as the sole use of machine learning models on spectrogram coefficient data for sleep classification or the use of wavelets to detect desired frequencies that provide little to no information on the morphological information of the spindles detected. Furthermore, literature associated with a solidified mathematical model for sleep spindles and their statistical fitting to EEG spindles has been minimal.

The goal of the thesis is to investigate an alternative method for sleep spindle classification that combines techniques from existing spindle detection algorithms as well machine learning techniques where the primary source of features comes directly from the QPS model and its associated parameters $\{a, b, c, d, e, f\} \in \mathbb{R}$. The ability for the QPS model to fit to sleep spindles via an NLLS algorithm suggests a way to train a machine learning model using the optimised QPS parameter values as input features.

Chapter 4

Thesis Objectives

While the problem statement seeks to determine whether or not the QPS model is a reliable candidate for feature extraction for a machine learning model to perform binary classification between spindles and non-spindles, the primary objective is to assess the reliability of the NLLS regression in the process of generating the estimative parameter values. The following milestones described below summarise the tasks that were required to be completed in order for the objective to be met.

1. Pre-process raw PSG recordings, perform data cleaning and extract the appropriate EEG channels from the recordings.

2. Develop a streamlined system that acquires spindle and non-spindle samples and generate QPS parameter values for each sample via a fair criterion and validate against expert scorer annotations.

3. Create a 50-50 dataset of parameter values for spindles and non-spindles with appropriate labelling

4. Build a machine learning model that performs binary classification on the spindle dataset from Step 4 that takes in the parameters as features and determines if the features correspond to a valid AASM defined sleep spindle.

5. Refine the machine learning model and perform feature selection to determine the best QPS parameters to classify spindles.

The milestones have gone through slight amendments throughout the duration of this study and more importantly, the thesis objective was shifted to primarily focus on the assessment of the NLLS in the performance of the machine learning model used. The methodology was also dynamically changed as methods used initially produced invalid results and needed to be adjusted so that the data acquisition was as fair as possible. Such amendments included the use of a moving frame to acquire spindle and non-spindle samples and using the expert scorer annotations purely for validation purposes and not for the NLLS initialisation.

Chapter 5

Proposed Solution

5.1 Formulating The Problem

As was explained in the literature review, The QPS can be used as a statistical model to estimate the parameter values that best fit the model to a sleep spindle detected in an EEG signal. However, the non-linear nature of the model means simple regression methods such as linear regression cannot be used, making the determination of the parameter values non-trivial.

Let the parameters be defined by the vector $\mathbf{p} = (a, b, c, d, e, f)^T \in \mathbb{R}^6$, let $\mathbf{y}(t) \in \mathbb{R}^n$ represent the raw EEG spindle and let $\hat{y}(t; \mathbf{p}) \in \mathbb{R}^n$ represent the QPS model with parameters in $\mathbf{p}$. The error or residual, $\mathbf{r} \in \mathbb{R}^n$ would be expressed as:

$$r_{(i)} = y_{(i)} - \hat{y}_{(i)} \tag{5.1}$$

For $i = 1, 2, 3...n$. We wish to minimise the sum of squared residuals (SSR), also called the *cost function*, in this non-linear regression problem [30]. The SSR, which we shall abbreviate to S for convenience for all subsequent sections, is expressed in Equations 5.2 and 5.3.

$$S = \sum_{i=1}^{n} \left(y_{(i)} - \hat{y}_{(i)}\right)^2 \tag{5.2}$$

$$= \sum_{i=1}^{n} r_{(i)}^2 \tag{5.3}$$

Equation 5.2 and 5.3 can also be written in vectorised form where the sum is essentially the dot product between the residual vector, $\mathbf{r}$ and its transpose $\mathbf{r}^T$. This is shown in Equations 5.4 and 5.5.

$$S = (\mathbf{y} - \hat{\mathbf{y}})^T(\mathbf{y} - \hat{\mathbf{y}}) \tag{5.4}$$

$$= \mathbf{r}^T \cdot \mathbf{r} \tag{5.5}$$

The SSR is continuous and differentiable and so the optimisation of the non-linear regression is achieved by finding the local minima of the SSR for each QPS parameter in $\mathbf{p}$. In other words, we seek for an optimal value for j^{th} parameter of $\mathbf{p}$ when the partial derivative of the SSR with respect to the j^{th} parameter equal to 0. This is expressed in Equation 5.6

$$\frac{\partial S}{\partial p_j} = 2\sum_{i=0}^{n} \left(y_{(i)} - \hat{y}_{(i)}\right) \cdot \left(-\frac{\partial \hat{y_{(i)}}}{\partial p_j}\right) = -2\sum_{i=0}^{n} \left(y_{(i)} - \hat{y}_{(i)}\right) J_{ij} = 0 \tag{5.6}$$

Where $J_{ij} = \frac{\partial \hat{y_{(i)}}}{\partial p_j}$ represents the i^{th} partial derivative for the j^{th} statistical parameter in the Jacobian matrix, $\mathbf{J}$, which contains the partial derivatives for all the parameters with which the statistical model is defined by. In vectorised form, the partial derivative can also be expressed as [40].

$$\frac{\partial S(\mathbf{p})}{\partial \mathbf{p}} = -2\mathbf{J}^T(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{0} \implies \mathbf{J}^T(\mathbf{y} - \hat{\mathbf{y}}) = \mathbf{0} \tag{5.7}$$

5.2 Optimisation Methods

5.2.1 Gradient (Steepest) Descent Algorithm

The steepest (gradient) descent is a general iterative algorithm that attempts to update parameter values in a downhill direction [30, 40] generally towards the minimum of the cost function. Figure 5.1 summarises how the gradient descent works in a simple picture assuming the cost function is a simple parabolic curve (based from Equation 5.2). Suppose parameter p_j has an initial value and is to be optimised. A learning rate, $\alpha \in \mathbb{R}$ is set prior to the optimisation and applied to Equation 5.6 that is used to update $p_{j(n)}$ to a new value $p_{j_{(n+1)}}$ in the following manner [40]:

$$p_{j_{(n+1)}} \longleftarrow p_{j_{(n)}} - \alpha \frac{\partial S}{\partial p_j} \tag{5.8}$$

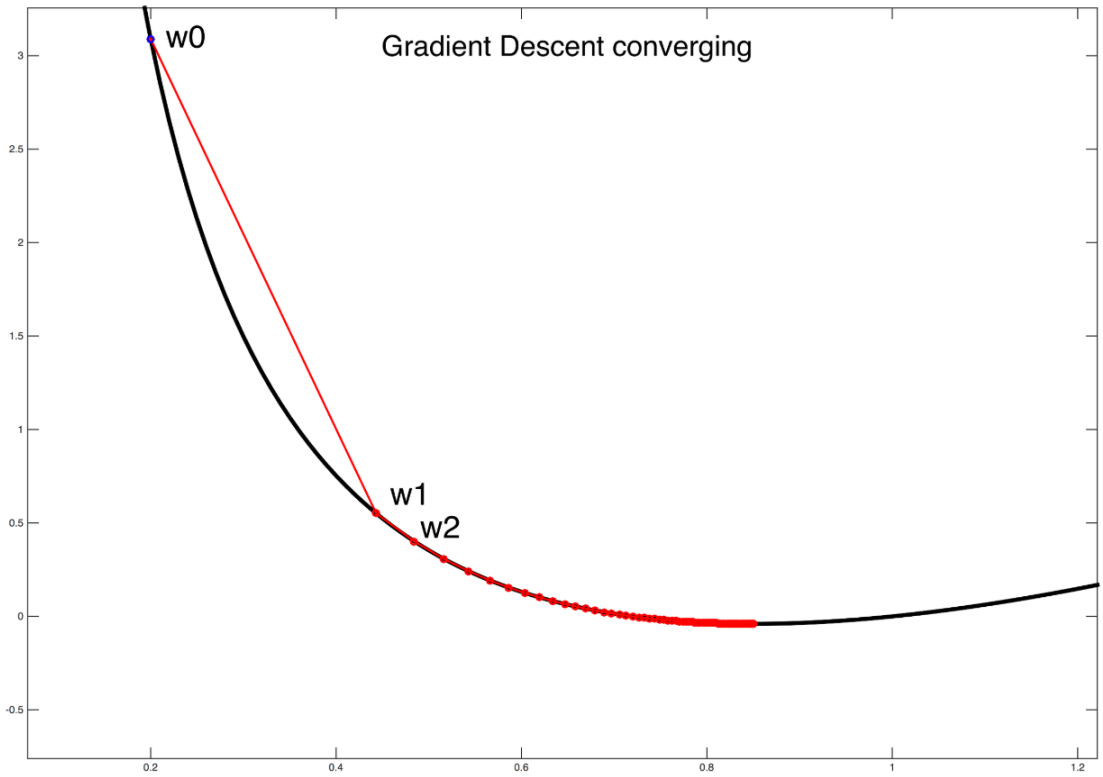


Figure 5.1: The gradient descent algorithm aims to iteratively converge towards the local minima of the cost function. The vertical axis represents the cost-function value while the horizontal axis represents the parameter value, p_j to be optimised

5.2.2 Gauss-Newton Algorithm

The Gauss-Newton (GN) algorithm is used in the case where cost function is approximately quadratic near the optimal parameter values [40]. These algorithms require an initial guess to the values of parameters, near to the optimum as much as possible, that aim to be that are then updated when seeking for the minima of the cost function. The GN algorithm aims to compute the *parameter update* which we denote as $\mathbf{h}$ that takes the initial estimate for the parameter value, $\mathbf{p}_n$ to the next value $\mathbf{p}_{n+1}$. This can be expressed as:

$$\mathbf{h} = \mathbf{p_{(n+1)}} - \mathbf{p_{(n)}} \tag{5.9}$$

The GN algorithm starts by approximating the value of the statistical model at the updated parameter value as a first-order (linear) Taylor series approximation as a function of the current parameter iteration [41]. This is expressed in Equation 5.10. We can think of this approximation as a secant of the cost function as shown in Figure 5.1.

$$\hat{y}_{(i)}(\mathbf{p_{n+1}}) \approx \hat{y}_{(i)}(\mathbf{p_n}) + \sum_{k=1}^{m} \frac{\partial \hat{y}_{(i)}(\mathbf{p_n})}{\partial p_k} \cdot h_k \tag{5.10}$$

The partial derivatives in Equation 5.10 form part of the Jacobian matrix for the parameter being updated which is essentially a vector. The sum can be simplified to be:

$$\hat{y}_{(i)}(\mathbf{p_{n+1}}) \approx \hat{y}_{(i)}(\mathbf{p_n}) + \sum_{k=1}^{m} J_{ik} h_k \tag{5.11}$$

For all parameters in the statistical model, this can be expressed in vectorised form:

$$\hat{\mathbf{y}}(\mathbf{p}_{n+1}) \approx \hat{\mathbf{y}}(\mathbf{p}_n) + \mathbf{Jh} \tag{5.12}$$

The goal is to find a closed-form for the parameter update with respect to the condition that we want to reach the local minima of the cost function as described by Equation 5.7. We begin with the partial derivative of the cost function at the next parameter iteration $\mathbf{p}_{n+1}$ which by substition into Equation 5.7 is expressed as:

$$\mathbf{J}^T(\hat{\mathbf{y}} - \hat{\mathbf{y}}(\mathbf{p_{n+1}})) = \mathbf{0} \tag{5.13}$$

Now, $\hat{\mathbf{y}}(\mathbf{p}_{n+1})$ is related to the parameter update as shown by Equation 5.12. Substituting Equation 5.12 into Equation 5.13, we have:

$$\mathbf{J}^T(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}_n) - \mathbf{Jh}) = \mathbf{0} \tag{5.14}$$

The parameter update can now be solved for as a function of the value of the model at the current parameter iteration and the value of the raw data as well. This is expressed in the form shown in Equation 5.15

$$\mathbf{h} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J^T}\,(\mathbf{y} - \mathbf{y}(\mathbf{p}_n)) \tag{5.15}$$

5.2.3 Levenberg-Marquadt Algorithm

The Levenberg-Marquadt (LM) algorithm was briefly explained in *Chapter 2* as an algorithm that jumps between the GN algorithm and the gradient-descent method depending on the proximity of the initial parameter estimates to the optimum. Traversing between the two methods uses a *damping factor*, λ that controls the way the cost reduction is achieved [30, 40]. Mathematically, the LM algorithm is a slight modification to Equation 5.15 as shown in Equation 5.16 where the parameter update via the LM algorithm is denoted by $\mathbf{h}_{lm}$

$$\mathbf{h}_{lm} = (\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I})^{-1}\mathbf{J^T}\,(\mathbf{y} - \mathbf{y}(\mathbf{p}_n)) \tag{5.16}$$

The choice for the size of λ is decided by the following two-case scenario:

- **Large** λ: When the LM algorithm is to perform the optimisation using the gradient descent method. This is done initially to begin acceleration towards the optimum.
- **Small** λ: When the LM algorithm is to behave closer to the GN algorithm when the parameters are near the optimum.

Fortunately, the computation of the cost function minima can be achieved with in-built Python libraries such as the *NumPy* and *SciPy* libraries, specifically the `linalg` class, that contains the `leastsq` method. The method utilises the `lmdif` and `lmder` LM algorithms developed originally by researchers More. J *et al* [42]. An alternative open source Python library, `lmfit` performs the NLLS regression in the same way but allows for parameters to be assigned to labels instead of floats.

Once the LM algorithm has been used to optimally fit the QPS model to the EEG spindle, the values of $\{a, b, c, d, e, f\}$ can then be obtained. The LM algorithm will be repeatedly for numerous spindles extracted from raw EEG data. This is further outlined and explained in chapter 6.

5.3 Building The Machine Learning Model

The extracted parameter values from the use of the LM algorithm can be repeatedly used over many EEG spindles extracted from raw EEG data to build a dataset of parameter values that either correspond to sleep spindles or not. The parameter values will be used as features along with their associated labels (Spindle, S or Non-Spindle, NS) to train a machine learning (ML) model to predict whether a given set of QPS parameter values corresponds to an AASM defined sleep spindle as per the problem statement for the thesis. In this case, the machine learning model needs to be able to perform binary classification from the features.

The final machine learning model used and tested was an artificial neural network (ANN). A possible structure for the neural network model is shown in Figure 5.2 that comprises of an input layer which accepts an input vector $\mathbf{x} \in \mathbb{R}^n$ with elements corresponding to data for the n features in the dataset, hidden layers which comprise of neurons that perform non-linear transformations to the data and an output layer that effectively maps the data to an output space for generally binary or multiclass classification.

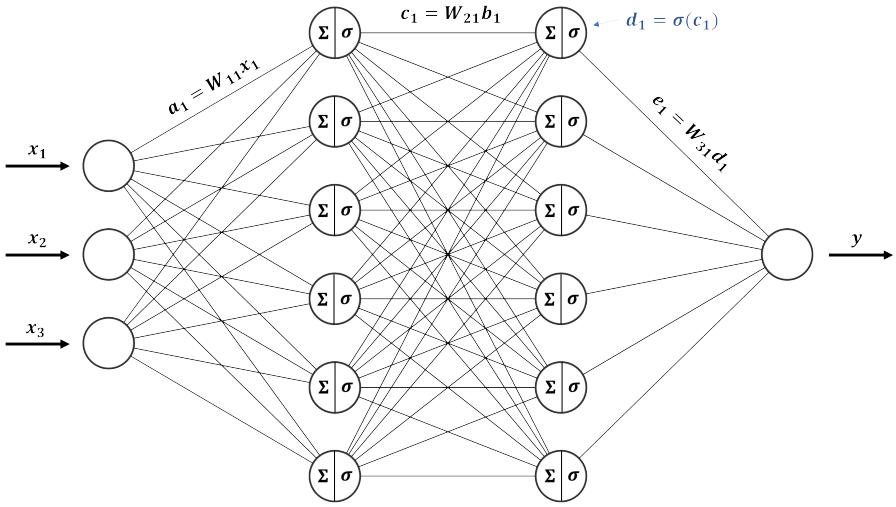


Figure 5.2: Proposed ANN architecture to perform binary classification

5.3.1 Feedforward Process

In the context of the problem statement, the input neurons represent the features or the QPS parameter values obtained from the NLLS optimisation process. Each input, which we shall call x_i, has an associated weight $w_{ij} \in \mathbb{R}$ which determines the strength of the connection between the j^{th} hidden layer neuron and the i^{th} input neuron. The input to a hidden layer neuron is the linear combination of the weighted inputs which can be expressed mathematically per Equation 5.17 as:

$$b_j = \sum_{i=1}^{N} x_i w_{ij} \tag{5.17}$$

An *activation function*, $\sigma : \mathbb{R}^n \to \mathbb{R}^m$ is then applied to the weighted sum as a non-linear transformation. The mapping can be expressed as:

$$c_j = \sigma(b_j) = \sigma\left(\sum_{i=1}^{N} x_i w_{ij}\right) \tag{5.18}$$

The non-linear transformation is performed to map the input-space to a higher dimension output space with the hopes that the mapping will reveal class separations between the data

that the neural network can easily distinguish. This mapping is shown in Figure 5.3 where the mapping allows a hyper-plane that best separates the classes.

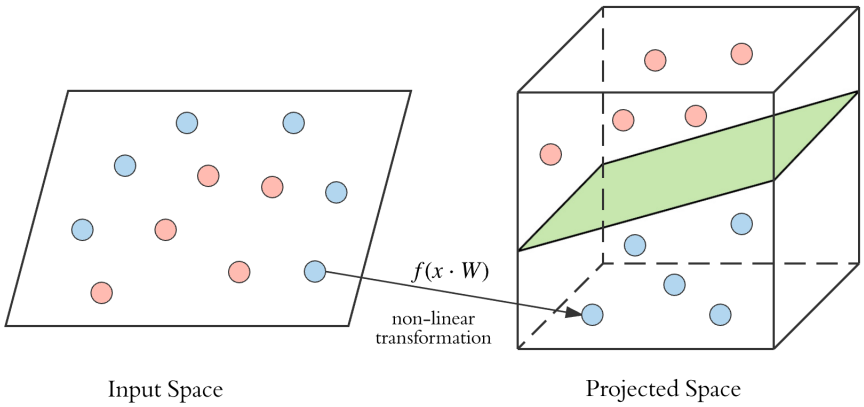


Figure 5.3: Non-linear activation functions in the hidden layers map the input (feature) space to higher dimensions where the classes can be separated (hopefully) by a hyper-plane [43].

The *Rectified Linear Unit* (ReLU) function is typically used to achieve this and is expressed via Equation 5.3.1 and its graph is shown in Figure 5.4. The function essentially compares the input value to zero and picks the maximum of those two values. Positive values are still fitted to the linear portion of the function while negative values are mapped to zero. The ReLU function is inherently non-linear since the input space is not entirely mapped from $\mathbb{R}^n \to \mathbb{R}^m$ from a linear function due to the portion where $\sigma(x) = 0$ for all $x < 0$.

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

The ReLU function is a popular choice for an activation function in the hidden layer as it performs the linear regression for all $x > 0$ which is computationally more efficient than using non-linear functions such as the sigmoid or the hyperbolic tan ($tanh$) which utilise an exponential term in the computation [44]. Furthermore, the ability for the ReLU function to map negative values allows it to output a true zero value unlike the sigmoid or $tanh$ function where a zero output value is not truly zero (due to their asymptotic and/or approximate behaviour). This is useful as this allows the MLP model to map many inputs to zero and

fewer to non-zero values making the learning process faster and simpler. This is other known as the *sparse representation* or *representational learning* [45].

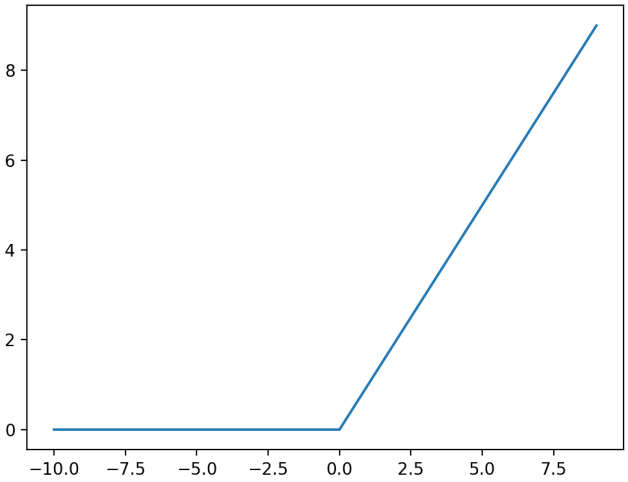


Figure 5.4: The ReLU Function

The output layer is responsible for producing values in the output space that represents the type of classification desired. In the case for this study, we want the neural network to distinguish between spindles and non-spindles which is a binary classification problem. That is, we want the output $y \in [0, 1]$. The sigmoid activation function defined in Equation 5.19 maps all input values in this desired range. Figure 5.5 shows the plot of the sigmoid function for all $x \in \mathbb{R}$.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.19}$$

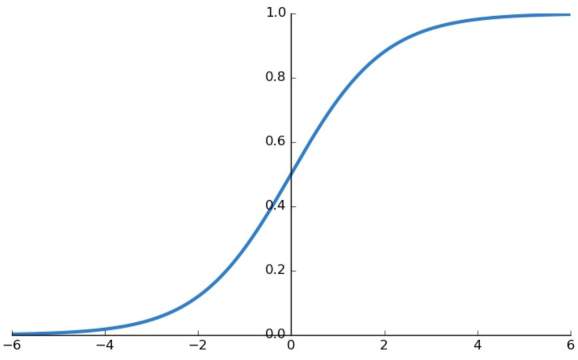


Figure 5.5: The sigmoid function

5.3.2 Training Process - Backpropagation

In the first iteration of the model, all predicted outputs will be far from the true output value from the dataset. This gives rise to an error or *residual*, r between the predicted value from the MLP, $\hat{y}$ and the true value to be predicted, y. That is:

$$r = y - \hat{y} \tag{5.20}$$

Minimisation of the residual is best performed by converting to it to a mean-squared error (MSE). Averaging the MSE based on the total number of training examples in our classification problem (which in our case is 2), we get the cost (objective) function denoted by J to be minimised for each parameter in the vector $\mathbf{p}$ for N training examples from the dataset.

$$J(\mathbf{p}) = \frac{1}{2N} \left(y - \hat{y}(\mathbf{p})\right)^2 \tag{5.21}$$

Where the $\frac{1}{2}$ is a convenient factor when the derivative is taken such as that the coefficient of the derivative is unity. The minimisation of the objective function is achieved by solving the following differential equation:

$$\frac{\partial J(\mathbf{p})}{\partial \mathbf{p}} = \mathbf{J}^T \left(y - \hat{y}(\mathbf{p})\right) \tag{5.22}$$

Where $\mathbf{J}$ is the Jacobian matrix comprising of the derivatives of the objective function with respect to each and every parameter (or feature) that served as the initial inputs to the MLP in the first place. The backpropagation algorithm works via the chain rule principle which states that the derivative of a function with respect to a variable can be related to a third variable. In general, this is typically written as:

$$\frac{dy}{dx} = \frac{dy}{dt} \times \frac{dt}{dx} \tag{5.23}$$

The cost function minimisation process involves finding the minimum of the cost derivative with respect to the *weights* used in the hidden layers. Suppose a single weight used between the output layer and the third hidden layer is w_{i3}. Let e be the summation of all the weighted inputs to the output neuron (based off of Figure 5.2 before the activation function is used to

produce the output y). Then, the link between the cost function and the weights in terms of their optimisation can be expressed mathematically via the chain rule:

$$\frac{\partial J}{\partial w_{i3}} = \frac{\partial J}{\partial y} \times \frac{\partial y}{\partial e} \times \frac{\partial e}{\partial w_{i3}} \tag{5.24}$$

In this case, we have:

$$e = \sum_{i=1}^{n} w_{i3} c_i \tag{5.25}$$

Where c_i is the output of the i^{th} neuron in the third hidden layer. On the k^{th} iteration of the training process, we use the calculated derivative to compute an update parameter, $\Delta_{k(i3)}$ that is used to change the value of the weight. Given some learning rate, $\eta \in \mathbb{R}$, the update parameter is computed as:

$$\Delta_k = \eta \cdot \frac{\partial J}{\partial w_{i3}} \tag{5.26}$$

If we let the $w_{i3(next)}$ and $w_{i3(current)}$ be the new and current weight values, then, the update parameter is added to the current weight value:

$$w_{i3(next)} = w_{i3(current)} + \Delta_{k(i3)} \tag{5.27}$$

This algorithm of iteratively computing an update parameter with respect to the minimisation of the cost function is effectively *gradient descent* as was described for the NLLS algorithm. In essence, gradient descent is used to iteratively compute the derivative of the cost function with respect to the parameter to be optimised until the local (or global) minima of the cost function is reached. This is shown diagrammatically in Figure 5.1.

When the minimum of the cost function has been reached, the model should be able to accurately predict most (if not all) the classes in the dataset. This optimisation occurs for all weights in the MLP in between all layers in general and will be left out for brevity.

Chapter 6

Methodology

6.1 Data Pre-Processing Block Diagram

The block diagram shown in Figure 6.1 summarises the process of taking the PSG recordings for the MASS C1-SS2 database for each test subject and extracting epochs. In this case, since spindles are predominantly found in Stage N2, we restricted our analysis and the provided expert scorer annotations to spindle samples found in this stage. The extraction of the epochs was achieved via the MNE Python library [46].

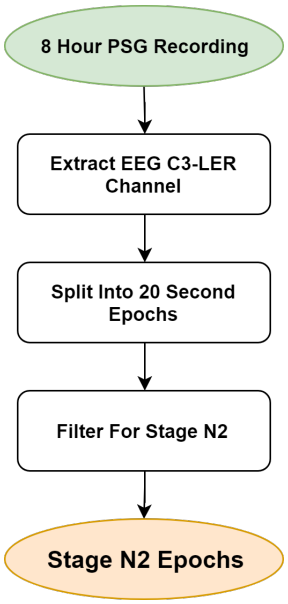


Figure 6.1: Block diagram for pre-processing the raw PSG data for relevant EEG channels

6.2 Feature Extraction Block Diagram

Figure 6.2 summarises the frame acquisition and feature extraction process. We followed Kulkarni's process of using a moving frame to acquire spindle and non-spindle samples [39]. In this case, we used a moving frame with size $T_w = 100$ ms with a 50 ms overlap (stride). The choice of the 100 ms frame size was such that the frame was within the 0.5-2.0 seconds duration of a spindle as defined by the AASM. The overlap was chosen to be smaller than the window size so as to ensure no spindle/non-spindle samples detected by the scorers would be skipped as well as generating more spindle samples as a result of the overlap.

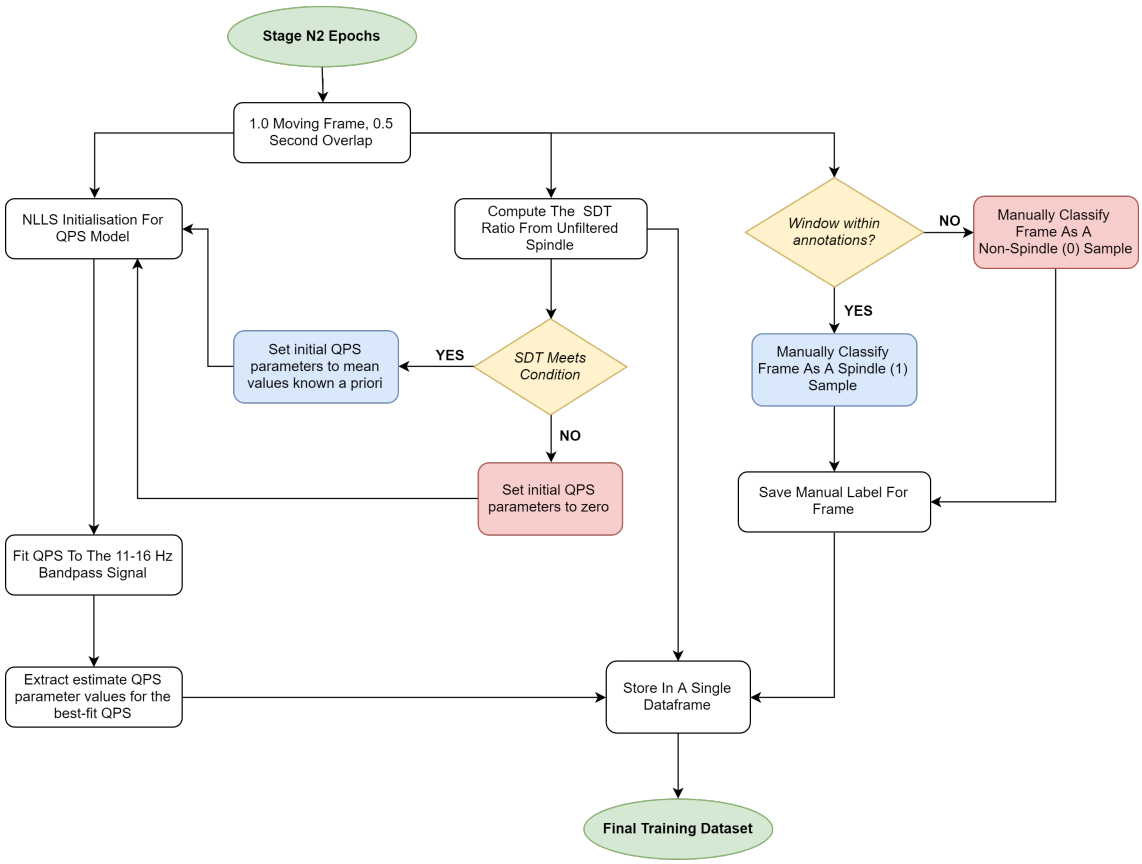


Figure 6.2: Frame acqusition and feature extraction process

A series of computations occur for a single frame that is either responsible for the NLLS initialisation, feature extraction or the manual classification as is shown in the flow chart. We break down each of the processes and formally declare the criterion used in the regression process.

6.2.1 Frame Acquisition & Manual Classification Process

To demonstrate movement of the frame along an EEG segment consider Figure 6.3. Let the yellow region represent the time-interval that the expert scorer has stipulated that a spindle has occured. The onset and offset are the left-most and right-most boundaries of this interval. Let the green region be the moving frame with size T_w.

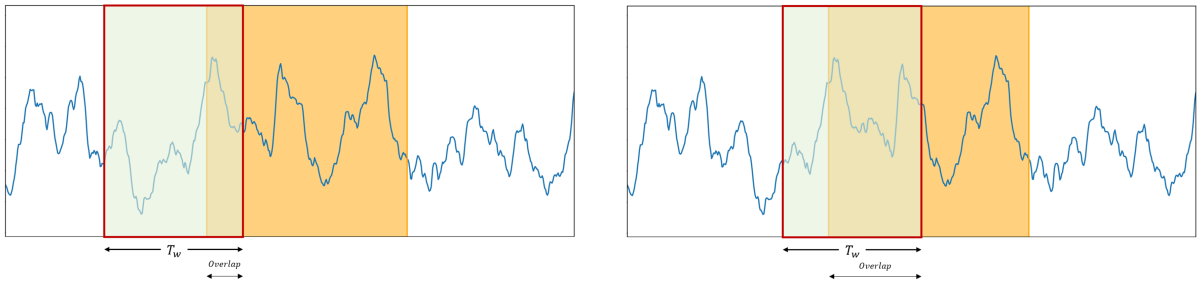


Figure 6.3: (LEFT) The overlap between the frame and the onset is too small to consider the acquired frame a spindle sample. (RIGHT) the overlap is sufficiently greater than 50% of T_w and so can be classified as a spindle sample

Consider the scenario where the frame approaches a scored spindle. In order to classify the captured portion of the EEG as a spindle, we require that overlap between the end of the moving frame and the scored spindle *must be* greater than 50% of the window size. This is shown in 6.3 on the right figure where the frame is well within the annotated spindle event time interval. Figure 6.4 shows a scenario where the moving frame is well within the annotated spindle event. The captured signal in this interval will be continuously classified as a spindle sample until the frame leaves the annotated event.

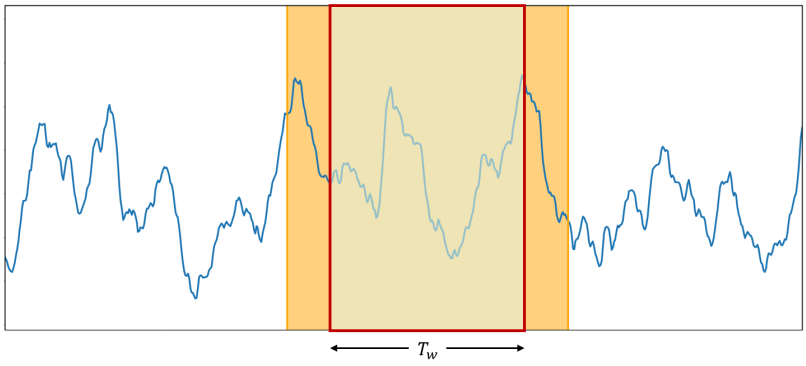


Figure 6.4: The moving frame is within the annotated spindle event. Hence, the captured signal is manually classified as a spindle sample

A similar criteria to Figure 6.3 is used when considering the departure of the moving frame from the spindle event. If the overlap between the start of the moving frame and the offset of the annotated spindle event is greater than 50% of T_w, the captured signal is classified as a spindle. Any less of an overlap and we consider the captured signal to be a non-spindle sample which marks the complete departure of the frame from the event. This is shown in Figure 6.5.

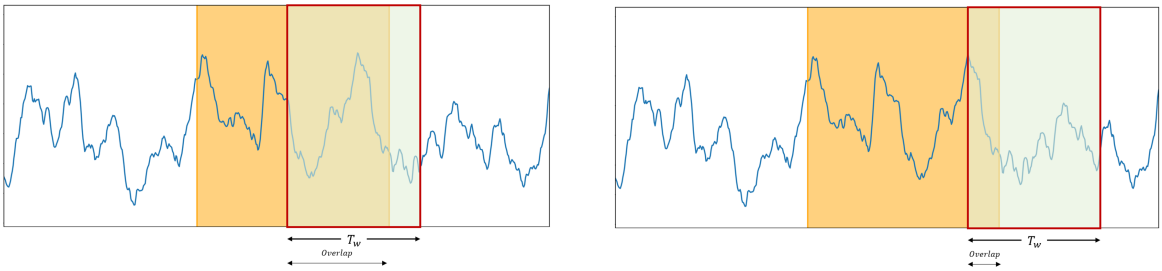


Figure 6.5: (LEFT) The overlap between the frame and the offset marker is sufficiently greater than 50% of T_w and so can be classified as a spindle sample. (RIGHT) The overlap is too small which means the captured signal must be classified as a non-spindle event.

6.2.2 Computing The SDT Ratio

For each acquired signal captured by the moving frame, the SDT ratio is computed as was defined via Equation 2.10 that Kulkarni et al used as a feature in their machine learning model. In this thesis, we use the ratio as a conditional criteria for the NLLS parameter initialisation. The relative spindle power in the 11-16 Hz range as well as the relative delta and theta was obtained by computing the *multitaper* PSD for each acquired frame. This PSD which we denote by S, is computed by using a series of windows or *tapers*, $a_k(t)$ (where k denotes the k^{th} taper) that effectively act as bandpass filters at different frequency bands [47]. Mathematically this is calculated as:

$$\hat{S}_k(f) = \left| \sum_{t=0}^{N-1} x(t) a_k(t) e^{-j\omega t} \right|^2 \tag{6.1}$$

These tapers reduce the spectral leakage that would be generated if an ordinary periodogram is obtained via a rectangular window [48]. Using multiple tapers generates multiple windowed PSDs which can then be averaged to produce the average PSD. We can then compute the

absolute power in a particular frequency range by evaluating the area under the average PSD curve. If we choose to divide the absolute power in a frequency range by the total power (total area under the PSD curve), we can calculate the *relative* (normalised) power of the band under consideration. This allows for the SDT ratio to be calculated which to reiterate is:

$$\text{SDT Ratio} = \frac{\text{Spindle Power}}{\text{Delta Power} + \text{Theta Power}} \tag{6.2}$$

Where the powers are relative to the total power under the periodogram for the entire frequency range in the captured frame.

SDT Ratio As A Criterion For NLLS Parameter Initialisation

We then used the EEG recording from Subject 1 of the MASS C1-SS2 database to get an idea of the range of values for the SDT ratio for spindles and non-spindles. This is further explored in the results in chapter 7. The aim is to use the SDT Ratio of the raw signal as the basis for the NLLS parameter initialisation as opposed to using the scorer annotations as performed by Palliyali et al [30]. This ensures minimal bias and assumes the context where no expert scorers are used to classify spindles whatsoever.

For comparison sake, the performance of this initialisation scheme will be compared to the prediction performance if the NLLS initialisation is based on the expert scorer annotations.

6.2.3 NLLS Regression Of The QPS To A Spindle

Once the QPS parameters have been initialised depending on the SDT Ratio, the regression is performed in order to determine the best estimate fit of the QPS to the raw spindle captured by the frame after bandpass filtering in the 11-16 Hz range. The resultant parameter values are then saved and stored into a dataframe along with the corresponding SDT ratio value and the manually classified labels for the subsequent machine learning stage.

6.3 Dataset Preparation & Constructing The Neural Network

The presence of outliers in the dataset, that could either be positive or negative, poses a small issue in the normalisation of the data prior to training the neural network. We use Z-score normalisation to ensure the resulting distribution of the dataset has a zero mean ($\mu = 0$) and a standard deviation of 1 ($\sigma = 1$). If X is the original data and X' is the normalised data, then the Z-score normalisation is given by Equation 6.3

$$X' = \frac{X - \mu}{\sigma} \tag{6.3}$$

Keras was used to build the artificial neural network for the machine learning process. Keras is a high-level API for Python that allows for the simple addition of hidden layers and neurons with the ability to alter the number of neurons in each layer and the activation functions used. As discussed in chapter 5, the architecture used is a simple feed-forward multi-layer perceptron with backpropagation via the *Adam* optimiser. For this study, we used a 6/20/10/1 architecture and monitored the validation loss and accuracy for each training iteration. The ability to monitor the validation loss and accuracy before and after feature selection was also another main reason in choosing to use a neural network as the machine learning model.

The final, normalised dataset is thensplit to form a training and test dataset in an 80:20 ratio after the order of the data is randomised. The training set is then further divided using an 80:20 split to also produce a validation set. The maximum number of epochs used for the training was 200 but an early stopping callback was used with a *patience* of 10 that would stop the training if the validation loss did not improve after 10 epochs at any point during the training process. This ensures data over-fitting is minimised allowing the ANN to generalise rather than memorise.

6.3.1 Feature Selection Using T-Test

Feature selection is then performed to remove certain QPS parameters that may prevent the validation loss from further reducing and to ensure a greater overall statistical difference between the spindle and non-spindle datasets. A two-sided T-test is used to compare the QPS parameters between spindles and non-spindles. We begin by stating the null (H_0) and alternative hypothesis (H_A). Let any of the QPS parameters be θ:

- **Null Hypothesis (H_0):** The difference between the mean parameter values of the spindles and non-spindles is zero. That is:

$$\mu_{\theta(spindle)} - \mu_{\theta(non-spindle)} = 0 \tag{6.4}$$

- **Alternative Hypothesis (H_A):** There is a non-zero difference between the mean parameter values for spindles and non-spindles. That is:

$$\mu_{\theta(spindle)} - \mu_{\theta(non-spindle)} \neq 0 \tag{6.5}$$

The null hypothesis cannot be rejected a significance level (α) of 1%. Alternatively, we accept the alternative hypothesis if the p-value is less than or equal to α. The end goal is to select the parameters fall under the alternative hypothesis. That is, select parameters that exhibit a significant statistical difference between spindles and non-spindles.

6.4 Metrics For Classification Performance

The performance of the neural network was evaluated using six different metrics which were the classificaiton accuracy, precision, recall, F1 score and the AUC score derived from the *Receiver Operator Characteristics* curve. Each of the metrics are dependent on the true and false positives (TP and FP respectively) which are classes correctly and wrongly classified as the positive class (logic 1) respectively. The metrics are dependent on the true negatives and false negatives (TN and FN respectively) which are classes correctly and incorrectly classified as negative (logic 0) classes by the machine learning model. We define some of the metrics before proceeding onto the results.

Classification Accuracy: The ratio of the number of correctly classified cases (TP and TN) divided by all cases classified $N = TP + FP + TN + FN$. That is:

$$Accuracy = \frac{TP + TN}{N} \tag{6.6}$$

Precision: Tells us the degree to which our model can correctly predict the positive classes (true positive, TP). A wrong classification of a positive class is called a false positive (FP).

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.7}$$

Recall: The recall is intuitively the ability of the classifier to find all the positive samples.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.8}$$

AUC-ROC Score This score computes the area under the curve (AUC) of the *Receiver Operator Characteristics* (ROC) curve. The ROC is a curve that graphs the rate of true positive classifications against the false positives. The AUC score can also be interpreted as the probability of detecting the true positive over the false positive class:

$$\text{AUC Score} = \mathbb{P}\left(TP > FP\right) \tag{6.9}$$

Chapter 7

Results

The results of this study compare the performance of our machine learning model between two scenarios which we will call *Scenario 1* and *Scenario 2* for brevity. For both scenarios, the classification system is the same and is based on the frame condition as was described in Subsection 6.2.1:

1. **Scenario 1**: Using the expert scorer annotations as the basis for the NLLS parameter initialisation and the manual classification

2. **Scenario 2:** Using the SDT ratio as the fair basis for the parameter initialisation and the scorer annotations purely for manual classification.

7.1 Scenario 1

In this scenario, the NLLS parameters were initialised depending on whether the frame was within a marked spindle marked by a scorer. We considered the two scenarios where:

(i) The NLLS was initialised to the full mean QPS parameter values if the frame was within a marked spindle and set to 0 if not.

(ii) The NLLS was initialised to the full mean QPS parameter values for both spindle/non-spindle samples captured by the moving frame.

a	b	c	d	e	f
0.82 (1.78)	1.05 (9.05)	-10 (3.87)	0 (4.69)	84.5 (3.86)	-0.9 (4.96)

Table 7.1: Values Used For The NLLS Initialisation

We test to see how effective the NLLS is distinguishing between spindles and non-spindles based on the parameter initialisation. Table 7.1 shows the mean values for each QPS parameters known a priori that were used for the NLLS initialisation for all cases where spindles were captured. The standard deviation for each parameter is also shown but is not included in the initialisation process.

7.1.1 Different Initialisation For Spindles and Non-Spindles

The frame acquisition process generated a total of 13533 spindles and 378865 non-spindles from 15 subjects in the MASS C1-SS2 database. To ensure an equal number of spindle and non-spindle samples, the non-spindle samples were first randomised and then 13533 non-spindles were pulled from the 378865 non-spindles. Table 7.2 shows the overall summary statistics for the QPS parameter values for both spindles and non-spindles.

	a	b	c	d	e	f
Mean	0.818	1.052	-13.791	0.146	82.405	-2.348
Std. Dev.	1.776	9.051	16.995	2.186	8.086	16.050
Min	-19.186	-72.416	-129.030	-13.573	18.162	-85.603
Max	3.393	86.470	10.504	16.864	137.854	78.280

Table 7.2: Values Used For The NLLS Initialisation

Table 7.3 shows the summary statistics for all non-spindles classified across all 15 patients. The results show disparate difference in the mean values for the QPS parameters. Furthermore, we should expect spindles to exhibit values of $a > 0$ in order for the envelope to be gaussian which the non-spindles largely do not show. The mean value for e for the non-spindles is not within the suitable range of $69.115 \leq e \leq 100.531$ which corresponds to

the AASM defined frequency band of 11-16 Hz. Figure 7.1 shows the respective box-and-whisker plot for the QPS parameters which shows the difference in the mean values for each parameter.

	a	b	c	d	e	f
Mean	-2.018	0.656	-3.172	2.514×10^{7}	-0.046	0.030
Std. Dev.	2.298	7.441	14.296	2.306×10^{7}	18.877	15.167
Min	-49.714	-86.041	-309.163	-7.802×10^{7}	-203.460	-214.649
Max	2.587	263.825	90.973	7.169×10^{7}	158.827	183.689

Table 7.3: Values Used For The NLLS Initialisation

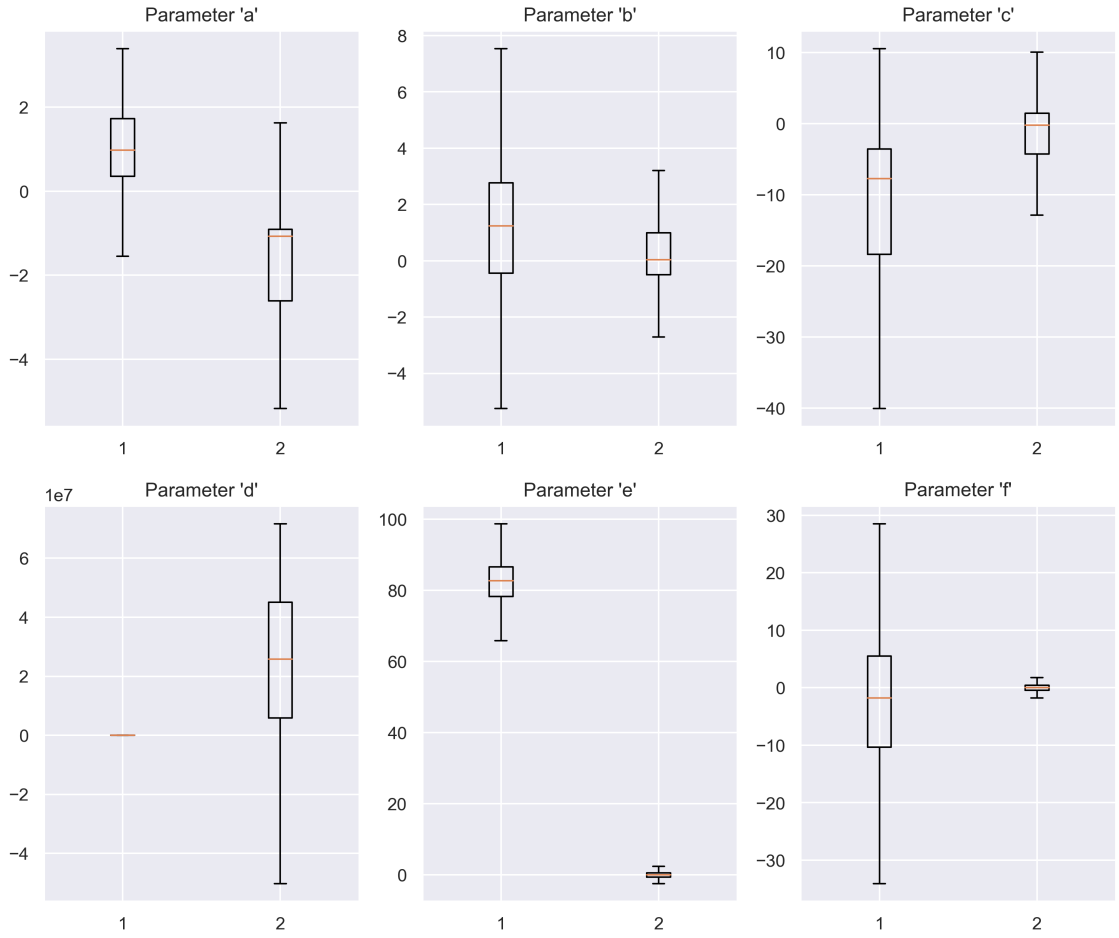


Figure 7.1: Distribution of QPS parameter values using the NLLS initialisation scheme based purely on the expert scorer annotations. Note that labels '1' and '2' correspond to spindles and non-spindles respectively.

Figure 7.1 shows the large difference in the mean values between the parameters, specifically parameters a, c, d and e. Note that outliers have been left out for clarity but Figure B.1 has been provided in Appendix B which shows that the QPS parameter values for non-spindles exhibit significantly more outliers than the spindle samples. This is due to the sensitivity of the NLLS regression process and the collapse of the QPS model when the parameters are initialised far from their optimal values. This is explained further in Chapter 8.

Table 7.4 shows the p-values for each QPS parameter after performing a two-sided independent T-test between the spindle and non-spindle samples and whether we accept the null hypothesis or not (i.e. $H_0 = 1$ means we accept, $H_0 = 0$ means we reject). The results show that parameters a, c, d, e and f exhibit the greatest statistical difference between their means for the spindles and non-spindles except for parameter b which had a p-value greater than 1%.

	a	b	c	d	e	f
p	0	0.306	0	0	0	0
H_0	0	1	0	0	0	0

Table 7.4: p-values for the T-test for all parameters between the spindle and non-spindle samples.

Figure 7.2 shows the pairwise scatter plots for each of the QPS parameters as a visual representation of the class separation that has formed between the spindle and non-spindle samples that have formed clusters for particular pairs of parameters such as b and e. However, the clusters are not perfectly separated and exhibit overlaps due to the large amount of outliers that have been generated for both spindles and non-spindles such as that shown for the scatter plot between parameters f and a.

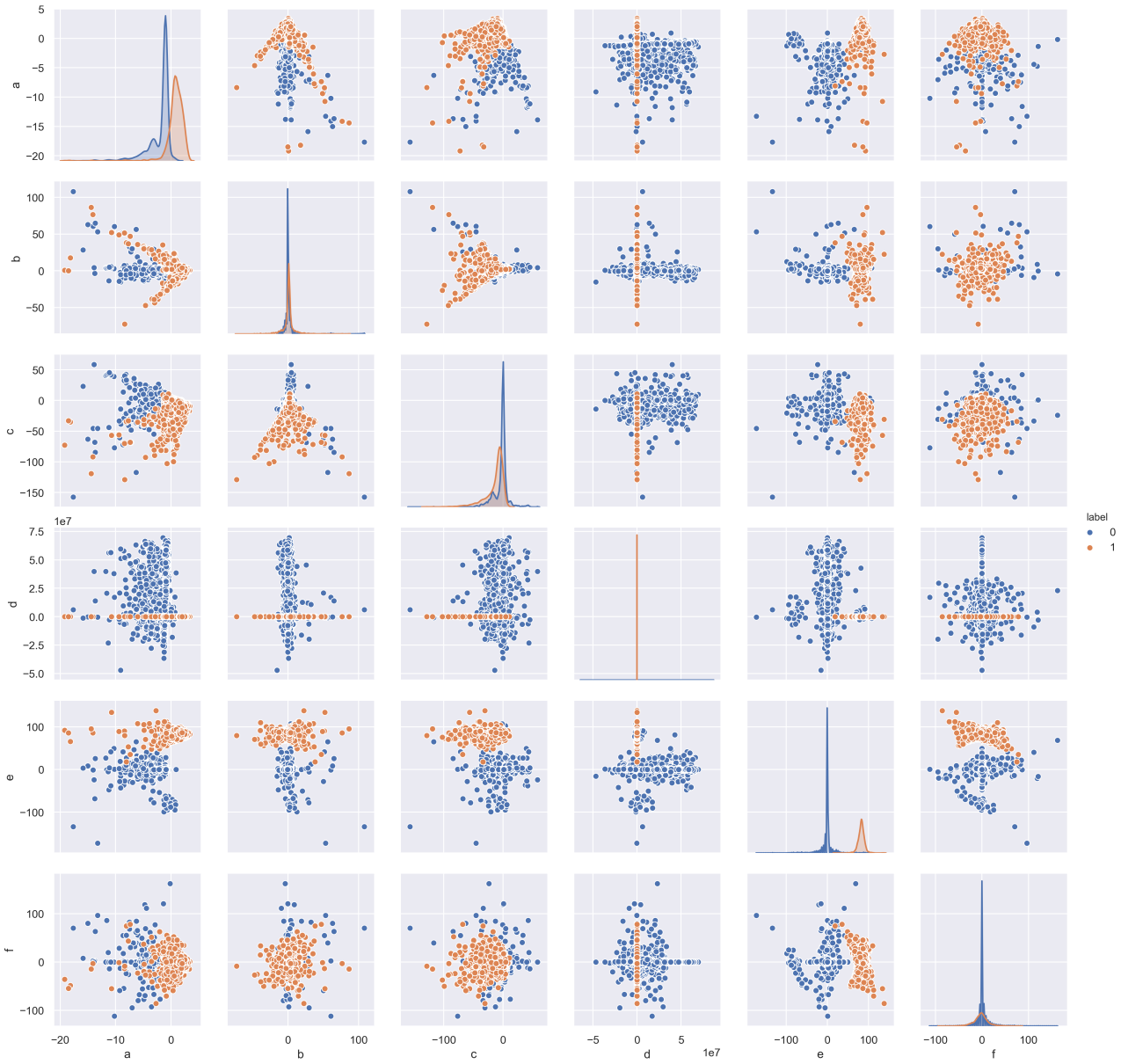


Figure 7.2: Pairwise scatter plot between the QPS parameters shows the presence of clusters that justify class separation between the spindle and non-spindle samples.

7.1.2 Machine Learning Performance

Using the 6/20/10/1 ANN architecture, we evaluated the performance of using all 6 QPS parameters as input feature vectors into the ANN after Z-score normalisation. The first step was to perform a 5-fold cross validation to get an average accuracy metric for the model with which we determined to be 98.7%. We also found the final training and validation loss

to be 3.97% and 5.97% respectively. Using the test subset of the overall dataset (20% split), the accuracy was found to be 99.4% with a loss of 3.5%. Table 7.5 summarises the metrics obtained from the model such as the precision, recall, F1 score and the AUC score. Figure 7.3 suggesting a high true positive detection rate.

Recall	Precision	F1 Score	AUC Score
98.4%	100.0%	99.1%	0.99%

Table 7.5: Evaluative metric for the neural network performance in spindle classification using all QPS parameters as features.

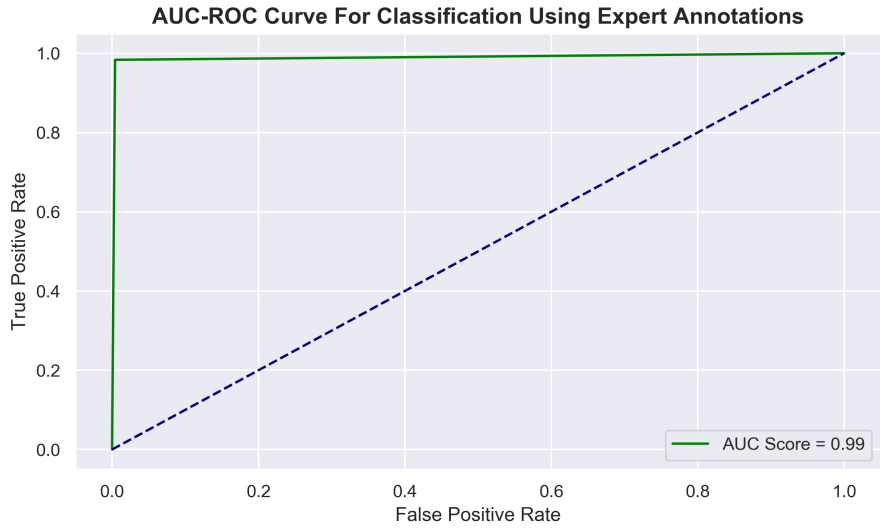


Figure 7.3: AUC-ROC curve using all QPS parameters as features.

Similar metrics were obtained after removing parameter b as part of the feature selection with the average accuracy after 5-fold cross validation being 98.9%. Table 7.6 shows the final evaluative metrics further reinforcing the strong performance of the neural network model in the classification process.

Recall	Precision	F1 Score	AUC Score
97.9%	100.0%	98.9%	0.99%

Table 7.6: Evaluative metric for the neural network performance in spindle classification using selected QPS parameters as features based on the two-sided T-test

7.1.3 Same Initialisation For Spindles & Non-Spindles

This time, we observed the classification performance of the neural network when the NLLS is initialised indifferently for spindle and non-spindle samples as they were being acquired by the frame acquisition process. The initial QPS parameter values used remained the same as that shown in Table 7.1. Table 7.7 shows the summary statistics for the QPS parameters obtained from spindle samples while Table 7.8 shows the same statistics for non-spindles.

	a	b	c	d	e	f
Mean	0.818	1.052	-13.791	0.146	82.405	-2.348
Std. Dev.	1.776	9.051	16.995	2.186	8.086	16.0340
Min	-19.186	-72.416	-129.030	-13.573	18.162	-85.603
Max	3.393	86.470	10.504	16.864	137.854	78.280

Table 7.7: Summary statistics QPS parameter values for spindle samples using a consistent NLLS initialisation

	a	b	c	d	e	f
Mean	0.691	1.215	-14.744	0.092	82.841	-2.115
Std. Dev.	1.721	10.072	19.068	2.260	8.428	17.002
Min	-50.000	-71.343	-246.959	-33.126	-124.380	-169.675
Max	4.591	232.195	141.366	56.761	197.180	134.944

Table 7.8: Summary statistics QPS parameter values for non-spindle samples using a consistent NLLS initialisation

The results on Table 7.7 and 7.8 show a great degree of similarity between the mean values and standard deviations for all 6 QPS parameters which is an opposite observation from what was seen in the case where spindles and non-spindles were initialised differently. Figure 7.4 is a visual representation of the distribution in the values for each QPS parameters which show the small difference in the mean for the QPS parameters.

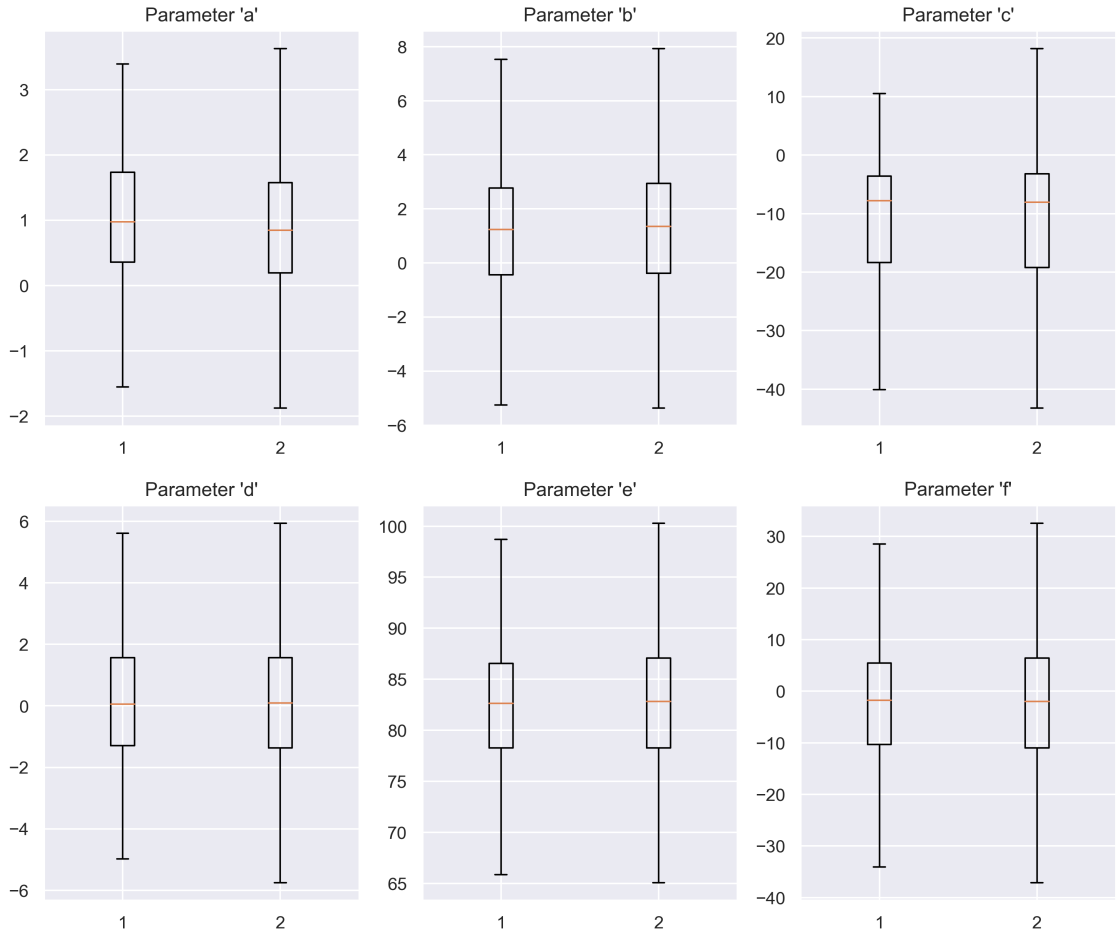


Figure 7.4: Distribution of QPS parameter values when the NLLS is initialised identically for spindles and non-spindles. Note that labels '1' and '2' correspond to spindles and non-spindles respectively.

The consequence of the indifference in the mean values for the parameters between spindles and non-spindles is shown in the pairwise scatter plot shown in Figure 7.5. While the spindles and non-spindles exhibit clustering, the clusters overlap each other completely with no hyperplane that can be used to separate the clusters. The histograms shown in the same figure show that for each individual QPS parameter, the spindle and non-spindle samples have the same distribution characteristics. As expected, the independent two-sided T-test showed no significant statistical difference in the mean values for a significance level of $\alpha = 1\%$ for each QPS parameter except for parameters a and d. The p-values for each parameter is summarised in Table 7.9 where $H_0 = 1$ means we accept the null hypothesis while $H_0 = 0$ means we do not.

	a	b	c	d	e	f
p	0.000	0.026	0.916	0.007	0.107	0.363
H_0	0	1	1	0	1	1

Table 7.9: p-values for the T-test for all parameters between the spindle and non-spindle samples.

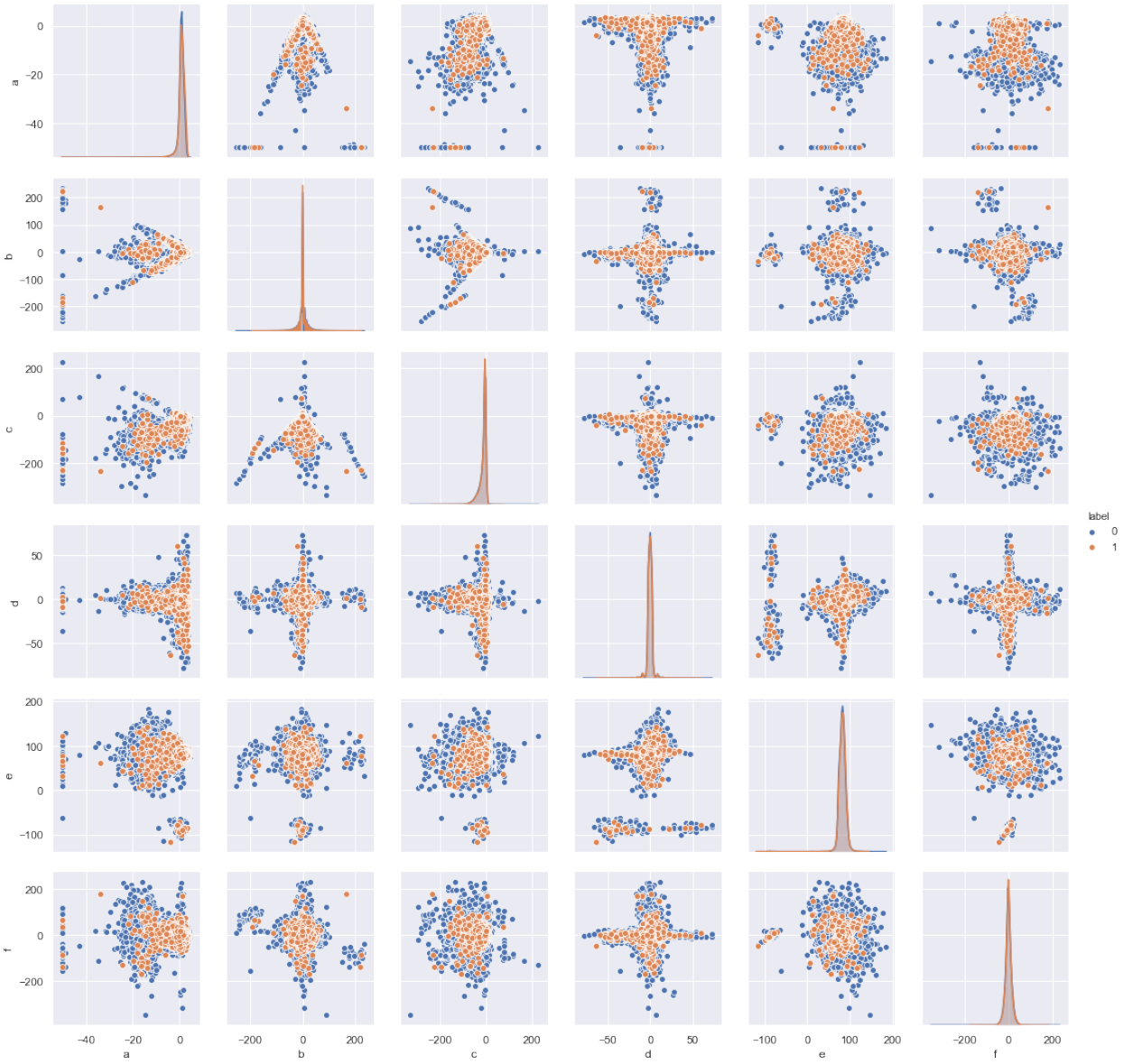


Figure 7.5: Pairwise scatter plot for the QPS parameters when the parameters were initialised in the same way for both spindles and non-spindles.

7.1.4 Machine Learning Performance

Using the same ANN architecture, we assessed the same metrics such as the accuracy, recall, precision, F1 score and the AUC score. The performance was comparatively worse and the machine learning model was unable to distinguish between the spindles and non-spindles when all QPS parameters were used as features. The 5-fold accuracy was determined to be 54.0%. Using the test subset of the overall dataset, the accuracy was found to be 53.7% but the loss was 69.6%, far beyond the loss in the case of using different parameter initialisation for spindles and non-spindles. The validation loss during the model training was also divergent as shown in Figure C.1 in Appendix C Table 7.10 summarises the other performance metrics for this case and Figure 7.3 shows the corresponding AUC-ROC curve showing a very small true positive rate.

Recall	Precision	F1 Score	AUC Score
54.1%	43.9%	48.4%	0.538%

Table 7.10: Evaluative metrics for the neural network performance in spindle classification using all QPS parameters as features.

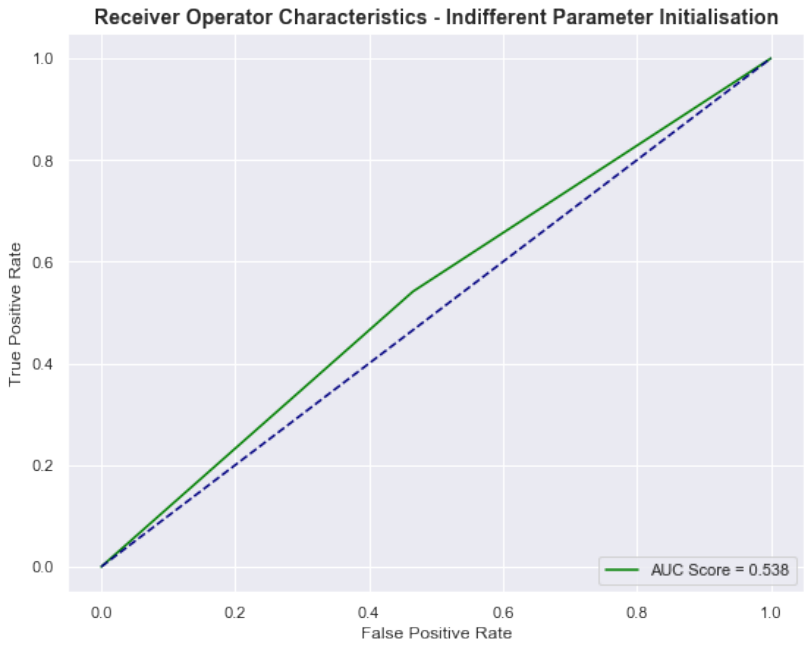


Figure 7.6: AUC-ROC curve when all features are used showing poor true positive rate detection performance

Isolating parameters a and d as per the T-test shows slight improvement in the AUC score to 0.54. However, the precision and F1 score both reduce to 0.31 and 0.399 respectively despite a and d exhibiting statistical differences in their mean values between spindles and non-spindle samples. This is clear from the scatter plot in Figure 7.5 where the univariate distributions for all parameters is nearly identical for spindles and non-spindle samples. Figure 7.7 shows the AUC-ROC curve after feature selection via the T-test.

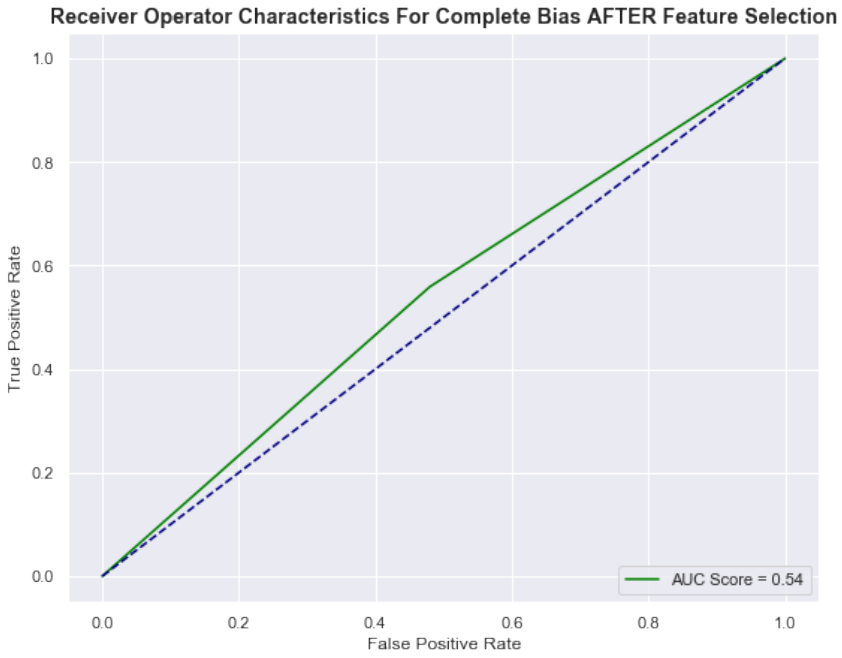


Figure 7.7: AUC-ROC curve after feature selection

Comments On Results

The large difference in the performance of the neural network in both of the scenarios is largely attributed to *how* the parameter initialisation is performed prior to the NLLS regression. A consistent parameter initialisation for spindle and non-spindle samples captured by the moving frame causes the QPS to best fit to all samples. Since the captured signal is bandpass filtered in the 11-16 Hz range prior to the regression, all the parameter values will be similar for spindle and non-spindle samples which was evident from Figures 7.4 and 7.5 leading to poor classification.

However, using different parameter initialisation for spindles and non-spindles allows for clear class separation. This is to be expected as the non-spindles were initialised as far from the optimal QPS parameter values as much as possible. The result is failure of the QPS model to best-fit to the non-spindles resulting in values far different from the distribution of QPS parameter values for spindle samples. The sensitivity of NLLS regression is further discussed in Chapter 8.

7.2 Scenario 2

In this case, we used the SDT ratio as the basis for the NLLS initialisation. In this scenario, we assume the context where sleep spindle annotations have not been provided. However, this requires that we know the SDT power ratio a priori. We used patient #1 to determine the distribution in the SDT ratio to be used in the initialisation which was determined to be a mean value of $\mu_{SDT} = 0.368227$ with a standard deviation of $\sigma_{SDT} = 0.263520$. Since using different parameter initialisation proved to be the best performing in separating spindles from non-spindles in Scenario 1, this initialisation scheme was also used in this scenario. Let $SDTR$ be the SDT ratio for a frame:

- If $\mu_{SDT} - \sigma_{SDT} \leq SDTR \leq \mu_{SDT} + \sigma_{SDT}$, initialise all QPS parameters to their full mean values as per Table 7.1.
- Else, initialise all QPS parameters to 0.

Tables 7.11 and 7.12 show the summary statistics for the spindle and non-spindle samples acquired with the SDT ratio as the initialisation criterion.

	a	b	c	d	e	f
Mean	0.216	-0.600	10.368	9.098×10^{6}	63.352	0.591
Std. Dev.	2.078	9.221	16.322	1.997112×10^{7}	36.407	14.435
Min	-50.000	-193.355	-235.545	-7.208346×10^{7}	-129.645	-162.043
Max	3.729	221.855	47.917	7.372272×10^{7}	138.052	179.838

Table 7.11: Summary statistics QPS parameter values for spindle samples using the SDT ratio as an initialisation criterion.

	a	b	c	d	e	f
Mean	0.159	-0.349	-10.699	7.210×10^{6}	64.886	0.703
Std. Dev.	8.911	9.221	16.285	1.763×10^{7}	35.281	14.591
Min	-50.000	-265.259	-311.263	-9.164×10^{7}	-200.680	-317.138
Max	4.668	231.193	119.249	7.791×10^{7}	231.493	226.420

Table 7.12: Summary statistics QPS parameter values for non-spindle samples using the SDT ratio as an initialisation criterion.

Figure 7.8 shows the resultant QPS parameter distribution for spindles and non-spindles. With no outliers included, it can be seen that there is a minute statistical difference in the means between the two classes for each parameter but not as large as that for case (i) of Scenario 1. Figure D.1 in Appendix D shows the same parameter distribution but with outliers included. The presence of a significant amount of outliers for each parameter distorts the symmetry of each of the distributions which affects the validity of the T-test (with which we assume a normal distribution). For this reason, we assess the performance of the machine learning model using all the QPS parameters without feature selection.

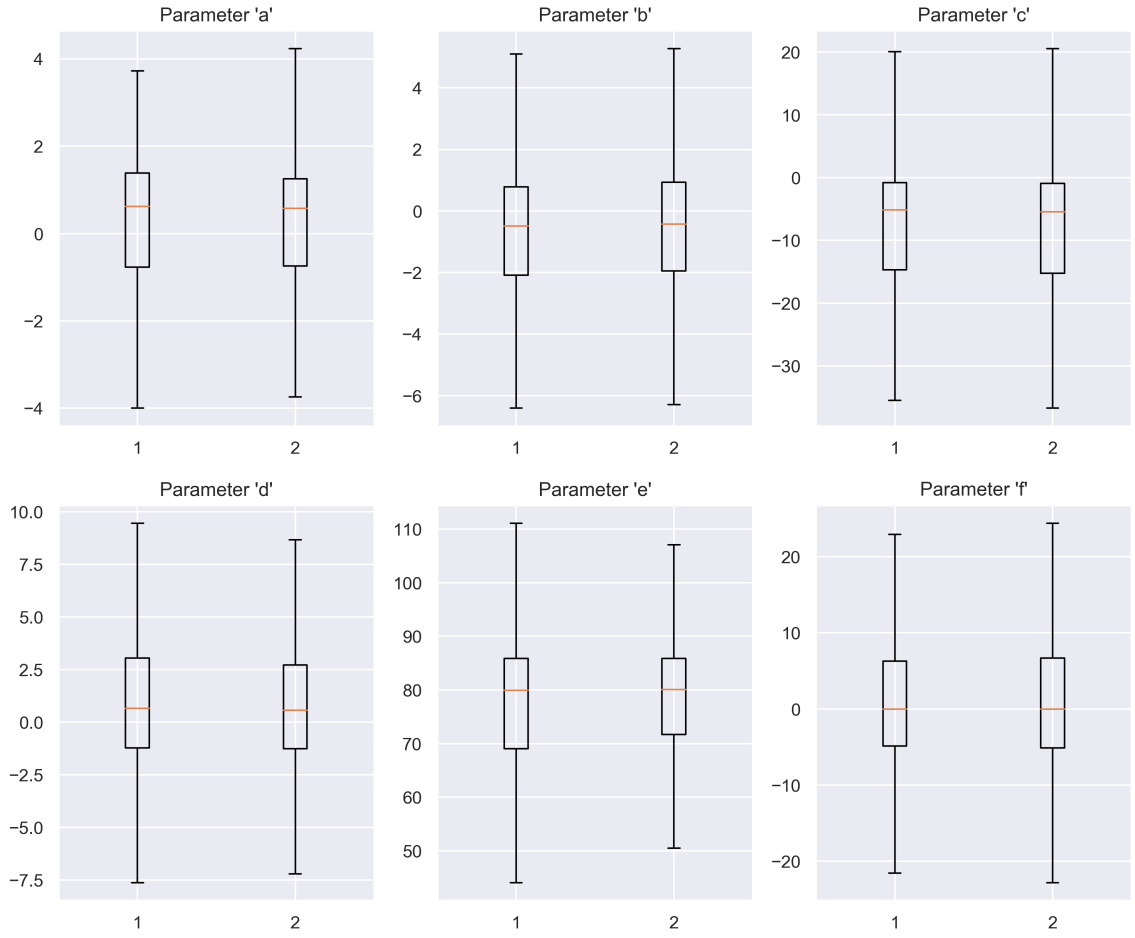


Figure 7.8: Distribution of QPS parameter values as a result of different initialisation for spindles and non-spindles based on the SDT ratio criterion

Figure 7.9 shows the pairwise scatter plot for the spindle and non-spindle samples generated. Like case (ii) for Scenario 1, there is clear overlap between the two classes with no distinguishing separation plane that can differentiate between the two classes.

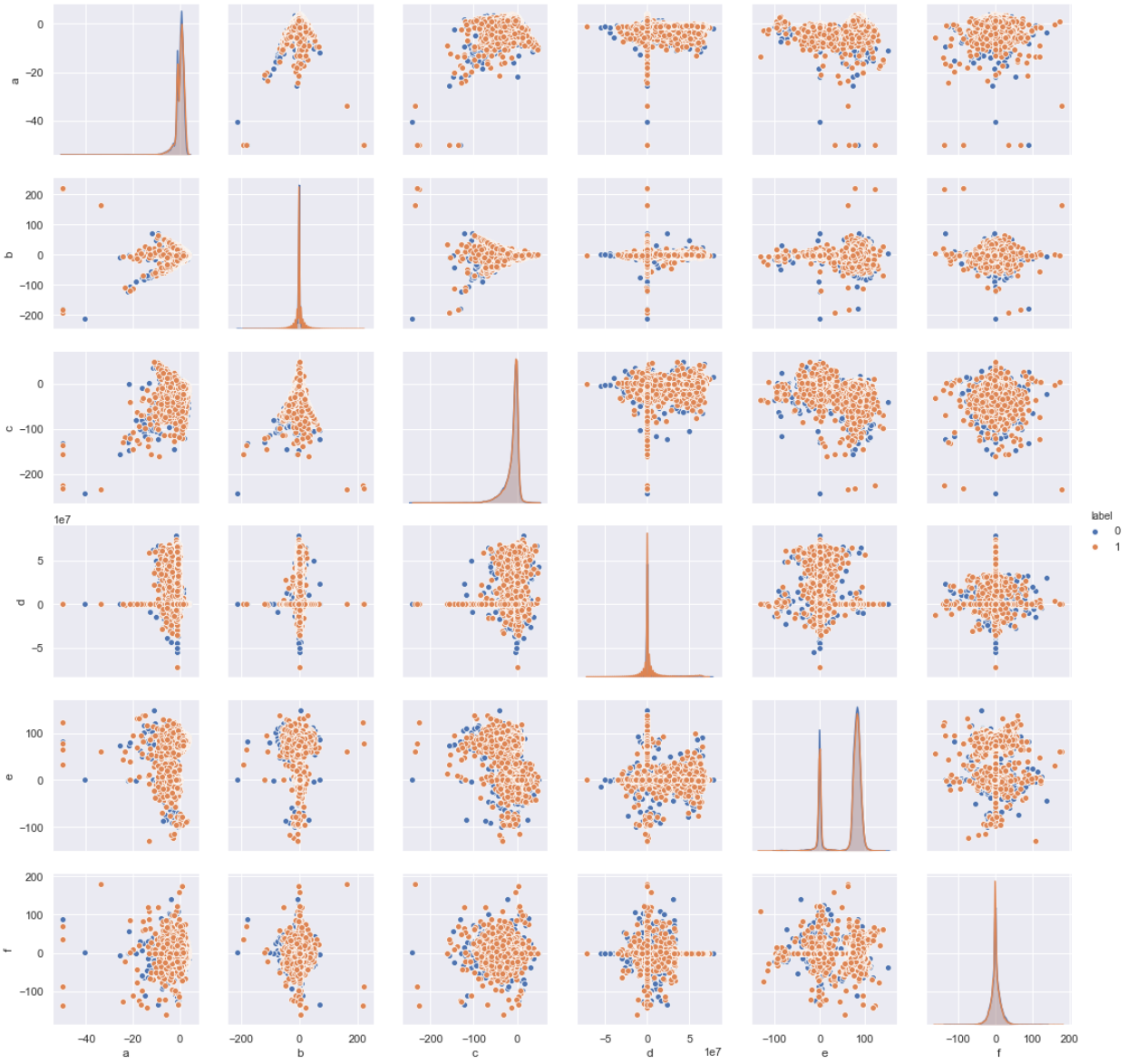


Figure 7.9: Pairwise scatter plot for the QPS parameters using SDT Ratio parameter initialisation.

As expected, the performance of the neural network in classifying spindles from non-spindles remains poor. The 5-fold accuracy was determined to be 53.5% where by the validation loss diverged from the training loss. Table 7.13 summarises the metrics obtained and is consistent with that from case (ii) of Scenario 1. Figure 7.10 shows the resultant AUC-ROC curve showing poor performance in the true positive rate.

Recall	Precision	F1 Score	AUC Score
0.574%	48.4%	52.5%	0.54%

Table 7.13: Evaluative metrics for the neural network performance in spindle classification using all QPS parameters as features.

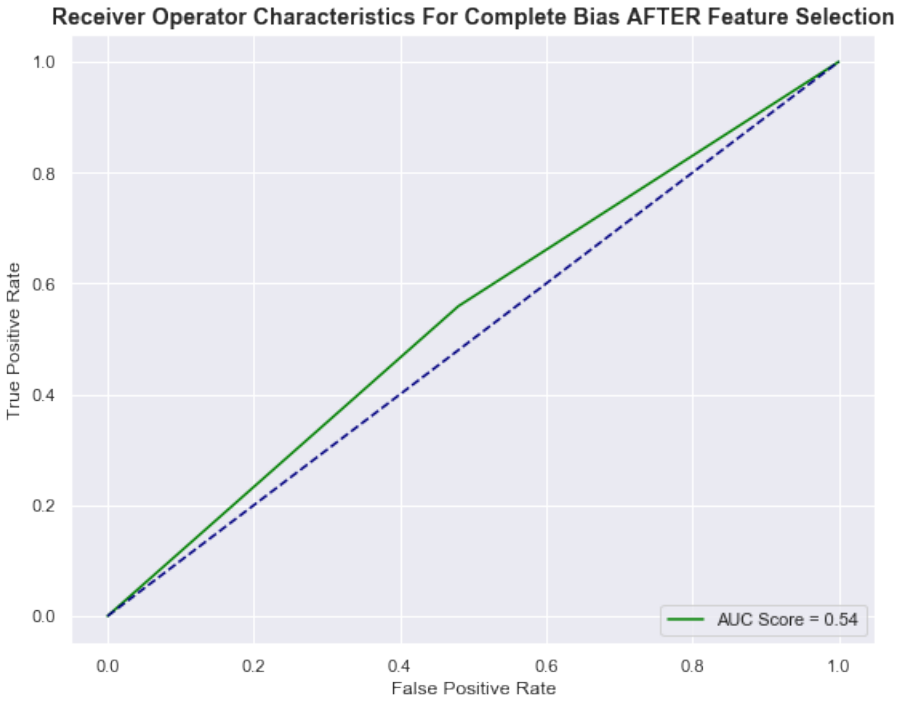


Figure 7.10: AUC-ROC curve using all QPS features

Chapter 8

Discussion

The results have primarily shown problematic issues with utilising the NLLS as a means of regression and feature extraction. These issues ultimately effect initialisation schemes involving the raw characteristics of the spindles and non-spindles captured. In this chapter, we explore some of the reasons for the disparate performance neural network when using the scorer annotations as the basis for the NLLS initialisation and using the SDT ratio as a fair criteria for initialisation.

8.1 Sensitivity of the NLLS

Like any regression algorithm, the NLLS is heavily reliant on how close the initial guess of the QPS parameter values are to the actual optimum which is found by minimising the cost function which in this case is the SSR between the captured signal and the QPS model to be fitted. Figure 8.1 shows the optimal fitting of the QPS model to the raw spindle using the same initial guess as per Table 7.1. This reinforces once more the ability of the QPS model to reconstruct spindles under the assumption that the initial guess to the parameter values is close to the local minima of the cost function. However, simply halving the initial guess to each parameter causes the QPS model to collapse and failing to fit to the raw spindle. This is shown in Figure 8.2 and shows how sensitive the NLLS algorithm is to small perturbations about the local minimum of the cost function.

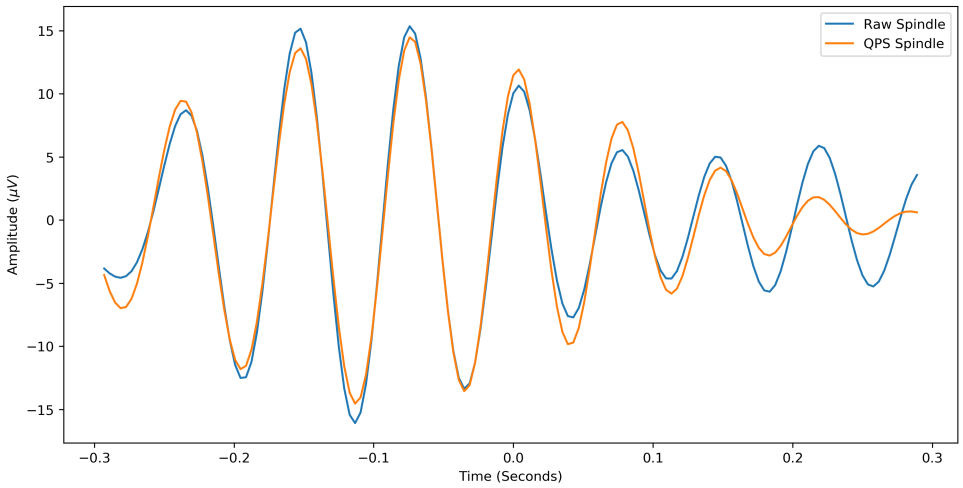


Figure 8.1: Close approximative fit of the QPS model to the raw spindle (after 11-16 Hz bandpass filtering)

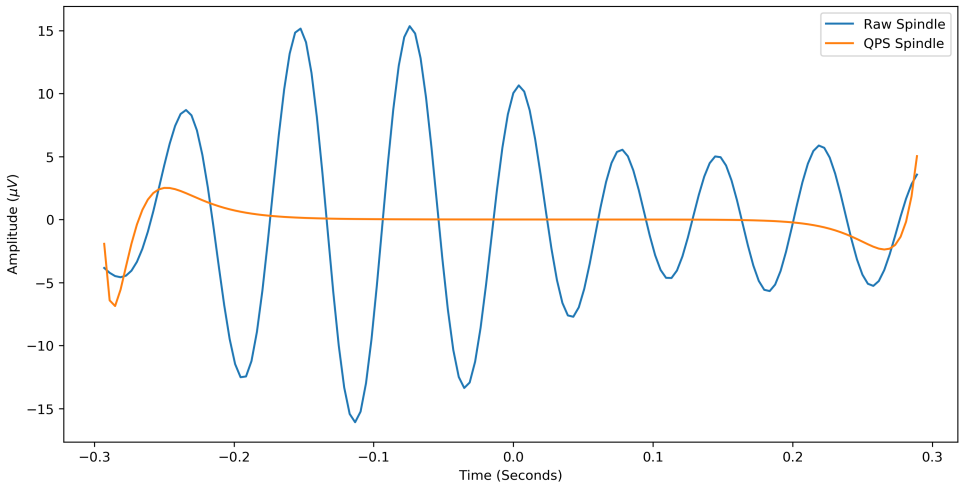


Figure 8.2: Collapse of the QPS model when the initial guess to the parameter values is not configured near the minima of the cost function.

The sensitivity of the NLLS can prove advantageous for classification and feature extraction if and only if it is known for certain that the acquired segment of the EEG signal (via the moving frame) is a spindle or not. Then, by initialising the NLLS with QPS parameter values near the optimum and non-spindles to any value far away from the optimum, we allow the QPS model to breakdown for the majority of non-spindles while fitting perfectly for spindle samples.

Consequently, this means the captured spindle samples are characterised with QPS parameter values that are consistent with values known a priori (e.g. from Kulkarni [39] and Palliyali et al [30]) and that defined by the AASM in terms of the frequency as a result of the successful regression whilst the QPS model produces non-sensical values as a result of a non-optimum NLLS initialisation. This leads to a clear class separation between spindle and non-spindle samples that allows classification to be clear and simple for the neural network.

8.1.1 The Achilles Heel Of The NLLS

However, the NLLS performed in this manner must be done cautiously. From case (i) of Scenario 1, it was seen that the classification is effective if and only if the microevent marked in the EEG is known for certain as a spindle. However, this becomes ineffective when if spindles and non-spindles are initialised with the same set of parameter values. Since band-pass filtering in the 11-16 Hz range is an imperative step prior to the regression, it should be expected that the filtered signal carries properties such as frequency, carrier phase and envelope amplitude that the QPS model can best-fit *regardless of whether the captured sample is a spindle or non-spindle*. This results in non-spindles sharing QPS parameter values close to the distribution of parameter values for spindle samples.

This was largely witnessed in case (ii) of Scenario 1 which led to both classes having similar parameter distributions resulting in the poor performance of the neural network. This is consistent with the findings by Palliyali et al [30] that showed that majority of the QPS parameters satisfied the null hypothesis for a two-sided T-test between spindles and non-spindles when subject to the same initial parameter values. This reinforces that the NLLS must be initialised differently for both classes in order for class separation to occur.

8.2 SDT Ratio As A Poor Initialisation Criteria

The outcome of using the SDT ratio as a mean of NLLS parameter initialisation proved to be poor performing despite the expectation of the SDT ratio of spindles concentrating at some interval (where the spindle power is relatively higher than any other time region of the EEG signal) as per the work by Devuyst [32] and Kulkarni et. al. [39]. This can be attributed to a flaw in the range used for the SDT ratio used in the initialisation in our implementation as well as inconsistencies in the spindle power across the entire signal.

In the methodology, we used Patient #1 to derive a suitable range with which we believed the SDT ratio for spindles would most likely within since this range needs to be known empirically as with the initial parameter values for the NLLS. However, as shown in Figure E.1 in Appendix E, the distribution of the SDT ratios for both spindles and non-spindles are roughly the same. Like the case of the NLLS, this results in an overlap between the spindle and non-spindle samples also leading to poor classification performance. Furthermore, the use of range purely based on one subject is a naive approach as not all subjects will share the same range of SDT ratio values.

8.2.1 Comparisons To Other Related Work

Devuyst et al [32] made similar observations when using the spindle power ratio as a means of spindle detection and classification. They found an increase in the number of false positive spindle samples when considering the intersection of both scorers as the ground truth with an alarming false positive rate of 239.62%. However, they found that the union of the scorers produced a false positive rate of of only 26.44 % for a sensitivity of 70.20%. Though an improvement, the false positive rate in both cases shows that such a power ratio is not the most reliable feature to be used when differentiating between spindles and non-spindles.

Kulkarni et al [39] relied on their convolutional and recurrent neural networks to extract features based on the overall signal and envelope of the baseline and the spindle as time-series. This suggests that spindle and non-spindle differences may be better derived from nuanced differences in the raw signal and envelope characteristics that the QPS model may otherwise not capture as a result of performing a regression. However, the consequence of using such a method is a clear gap in understanding how the neural network is learning features and more over *what features* the neural network is extracting and learning even though synthetic spindles derived from the QPS were used as training samples.

8.3 Further Work

The following subsections outline some of the aspects of the methodology that need to be largely improved on if any amendments to this study are to be made in the future. Much of the future improvements stated come from prior work investigated in the literature review that may not have been touched on in this study as a result of time-constraints and misdirection.

8.3.1 Sleep Spindle Detection Algorithms

The sensitivity of the NLLS, while an achille's heel in the overall process, works extremely effectively when it is known for certain that a detected event is a spindle. Using the different initialisation for spindles and non-spindles thereafter creates a clear class separation that allows for accurate spindle/non-spindle classification. However, the SDT ratio is clearly unreliable as a condition to be met in order for a particular set of parameter initialisation to occur.

Existing methods of spindle detection such as that developed by Lajnef et al [12] have an opportunity to be experimented with as a more robust alternative than using a power ratio. Proper threshold conditions could be applied to the coefficients extracted from the wavelet transform that could not only yield better correlation with the spindle annotations but could also prove to be a better way of performing the NLLS initialisation. However, computational

efficiency needs to be taken into account when performing time-frequency analysis coupled with machine learning.

8.3.2 Non-NLLS Based Feature Extraction

One limitation of the NLLS is its heavy reliance on initial conditions for parameters to be optimised. Such initial conditions require a priori parameter estimation which may be unreliable as not all spindles will share the same a priori estimates. An analytical method that does not depend on initial conditions is a future task that needs to be explored.

8.3.3 Other Machine Learning Models

While neural networks were used in this study for the sake of investigating the impact of each parameter on the learning process, better spindle detection algorithms and feature extraction methodologies may allow for clearer class separation between spindles and non-spindles that may open up opportunities for more current and comparatively efficient machine learning models such as support vector machines which find the global minima of the cost function as opposed to a local minima that a neural network generally does via gradient descent.

Chapter 9

Conclusion

In this thesis report, we proposed a solution and a streamline methodology that continues the work of Palliyali et al [30] in using the parameters in the QPS model as features that can be used in a machine learning model such as a simple feed-forward artificial neural network. The feature extraction process involves using a moving frame throughout Stage N2 sleep epochs in an EEG recording and acquiring segments of the EEG corresponding to spindles and non-spindles with which the QPS can best-fit to. The results however, show that while the sensitivity of the NLLS can be used to effectively separate spindles from non-spindles, it is a poor performing algorithm if a coupled automatic spindle detection algorithm is unreliable in distinguishing spindles from non-spindles. More work needs to be done to incorporate a more robust spindle detection algorithm that can be used as a basis for NLLS parameter initialisation or to develop an analytical method of parameter estimation that need not require parameter values known a posteriori.

Appendix A

QPS Parameter Changes

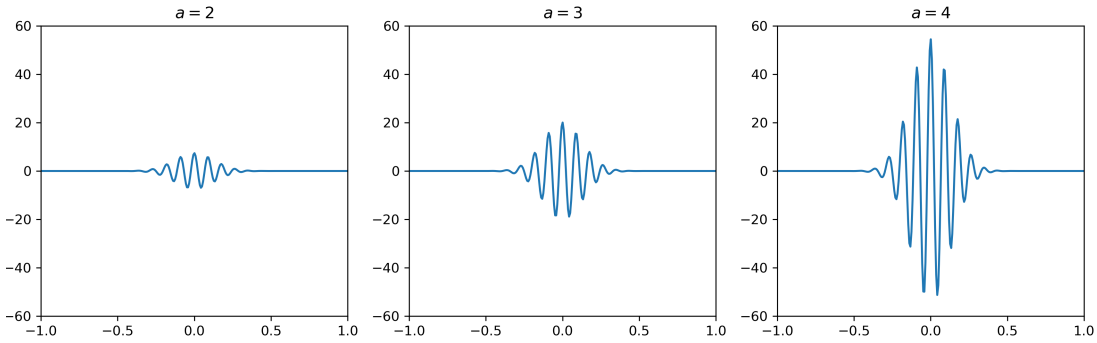


Figure A.1: The effect of varying a on the amplitude

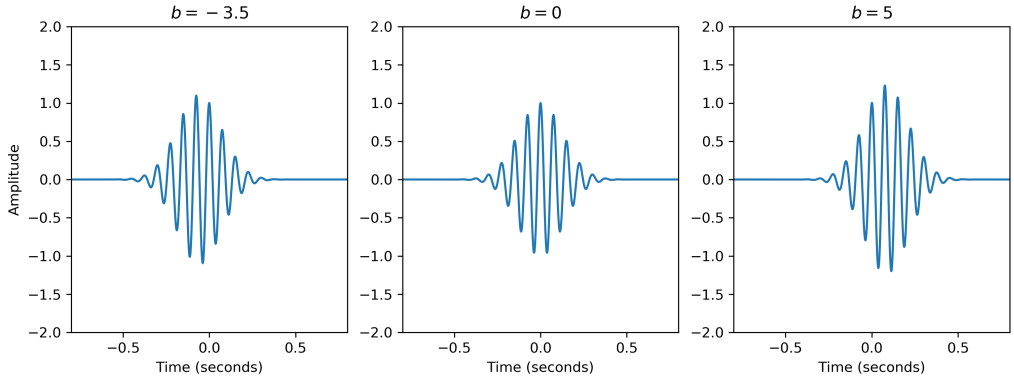


Figure A.2: Changes to b generate a shift to the spindle on the time-axis

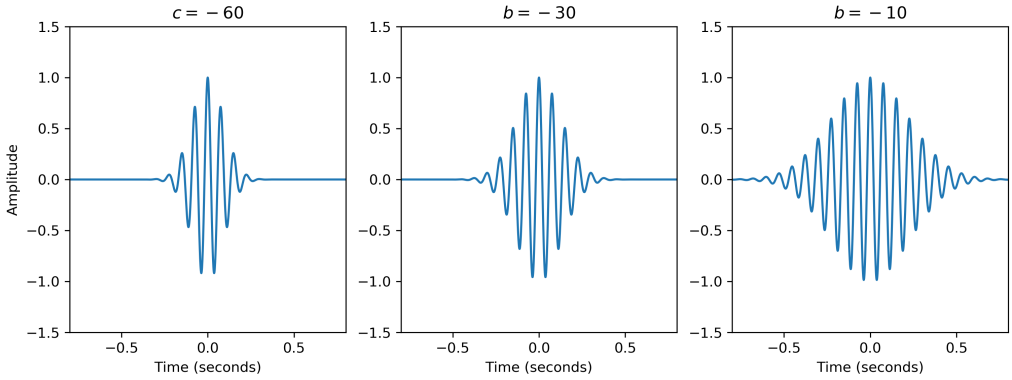


Figure A.3: Gaussian envelope decay rate changes with parameter c

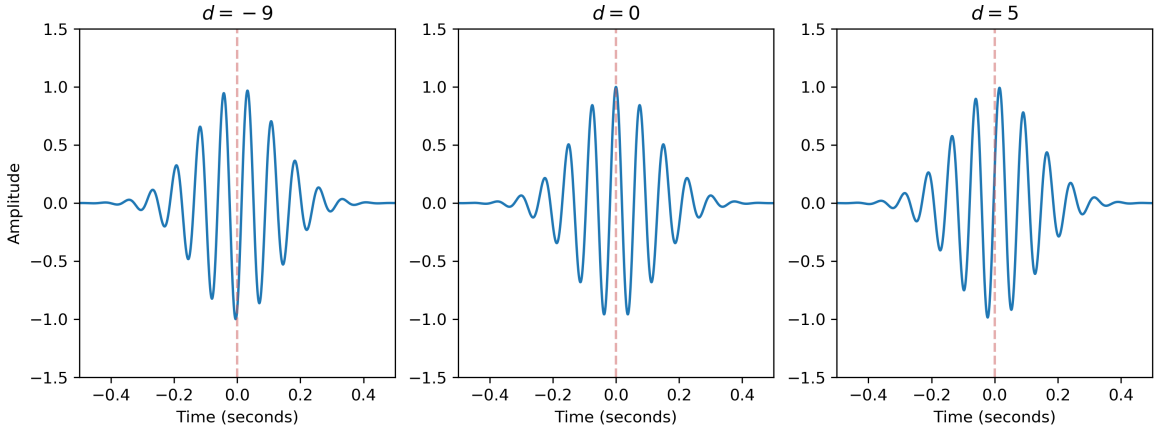


Figure A.4: Phase offset to the sinusoidal component of spindle due to changes in d

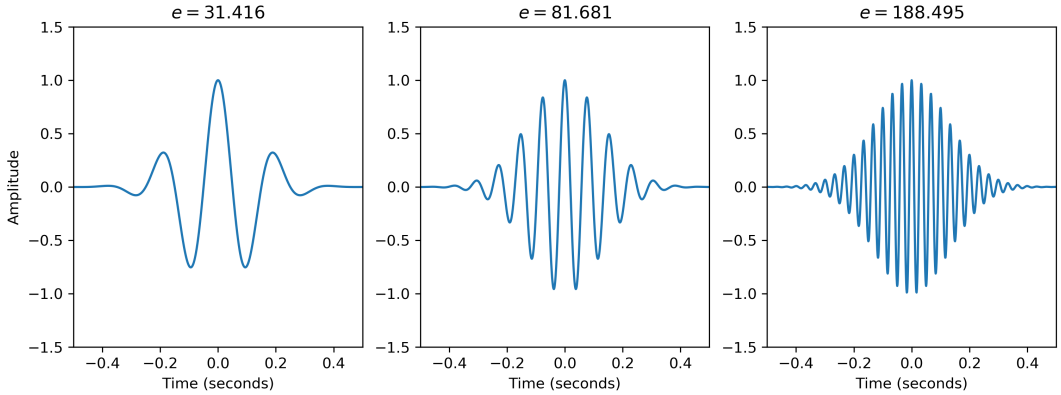


Figure A.5: Changes to the angular frequency via parameter e

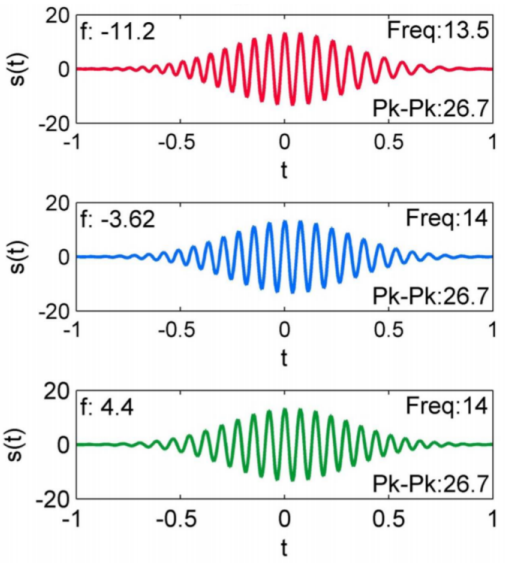


Figure A.6: Changes in f cause small changes about the centre frequency of the spindle. [30]

Appendix B

Scenario 1 - Split Initialisation WITH Outliers

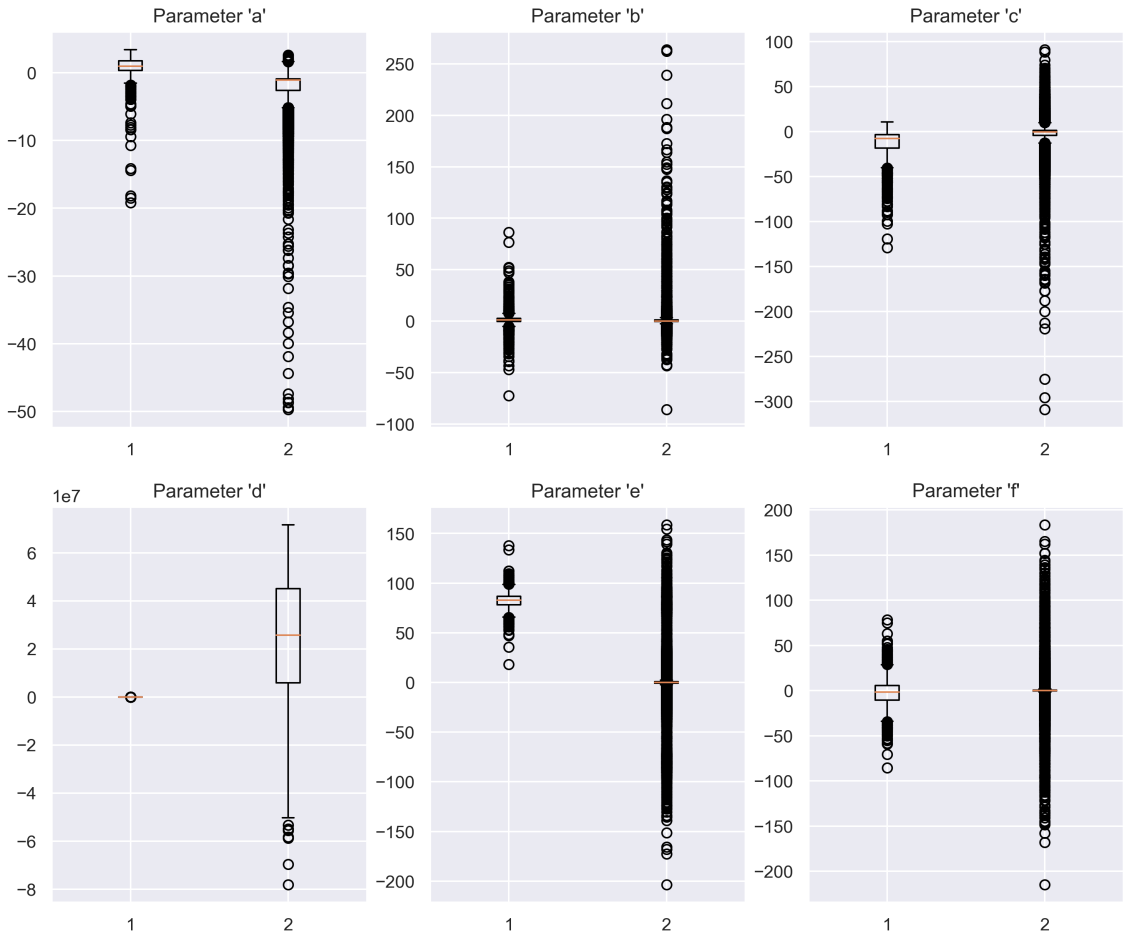


Figure B.1: Distribution of QPS parameter values using the NLLS initialisation scheme based purely on the expert scorer annotations with outliers included. The non-spindles exhibit many more outliers due to the collapse of the QPS model when the NLLS has not been initialised properly near the optimum.

Appendix C

Scenario 1 - Training & Validation Loss For Consistent Parameter Initialisation Between Spindles Non-Spindles (All Features)

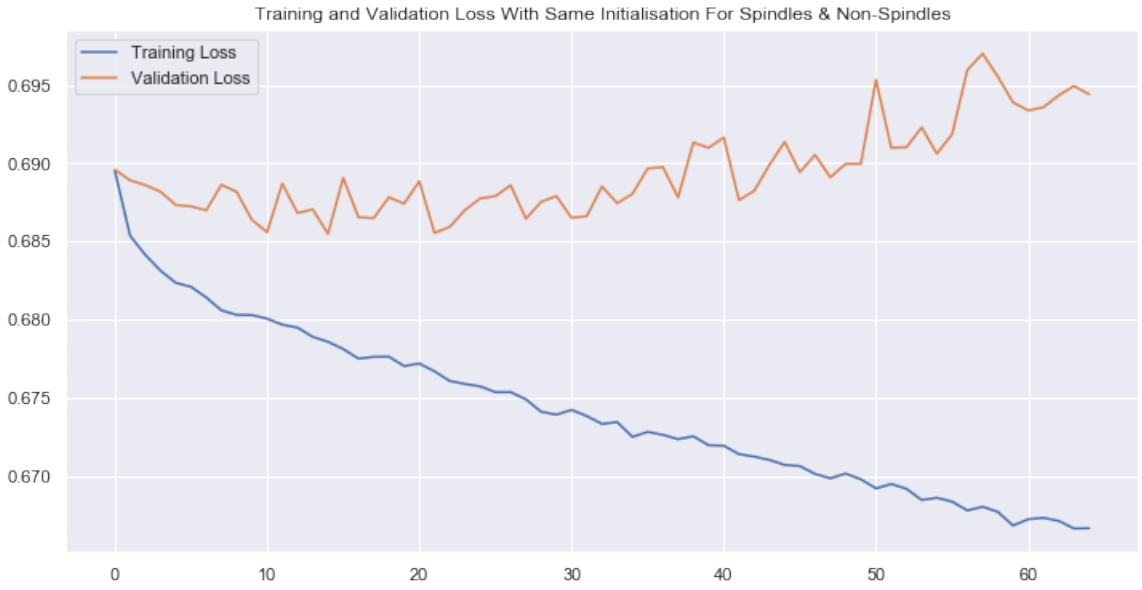


Figure C.1: Divergence between training and validation loss when all QPS features are used in the case where spindles and non-spindles were initialised with the same QPS parameter values prior to the NLLS regression for 150 training epochs with a patience of 50.

Appendix D

Scenario 2 - QPS Parameter Distribution - SDT Ratio Initialisation Criterion

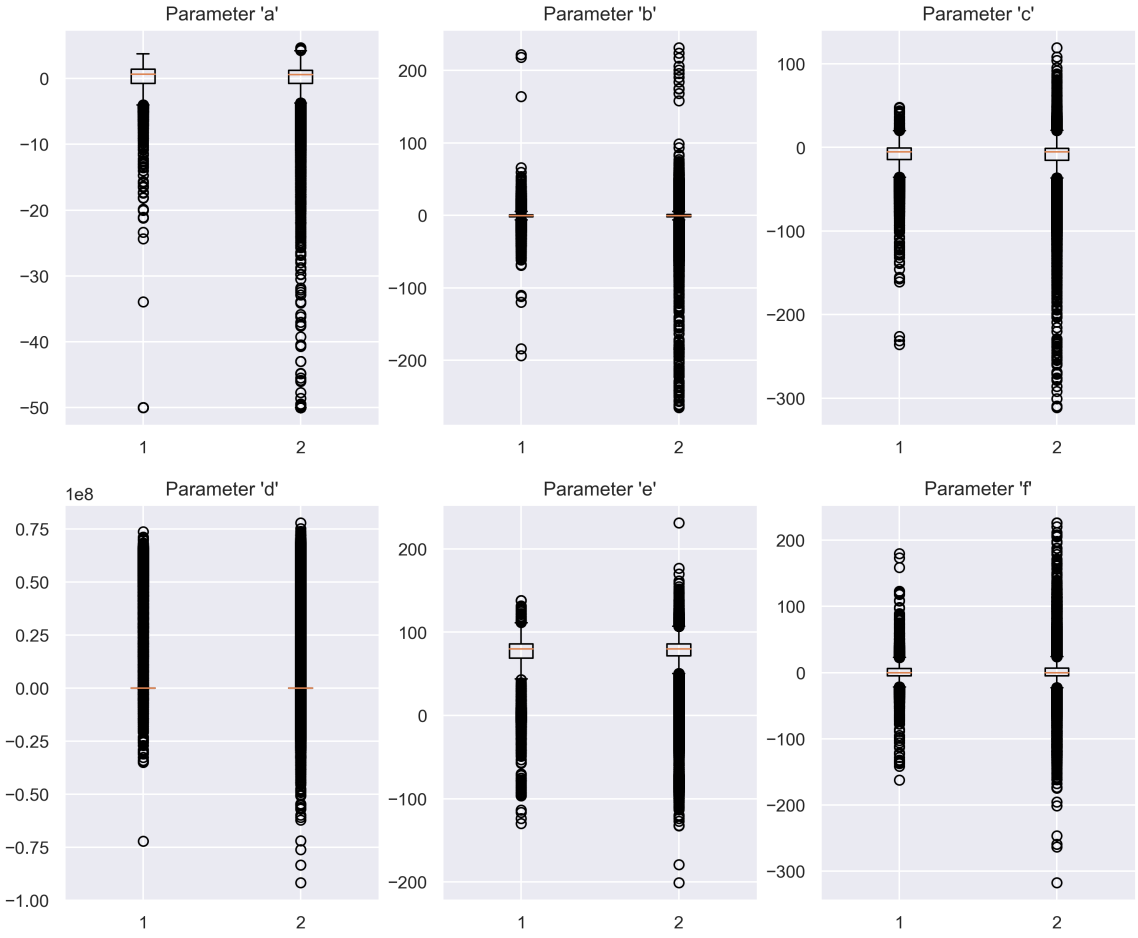


Figure D.1: Distribution of QPS parameter values using the NLLS initialisation scheme based on the SDT Ratio with outliers included..

Appendix E

Near Equivalent SDT ratio Distribution For Spindle and Non-Spindle Samples

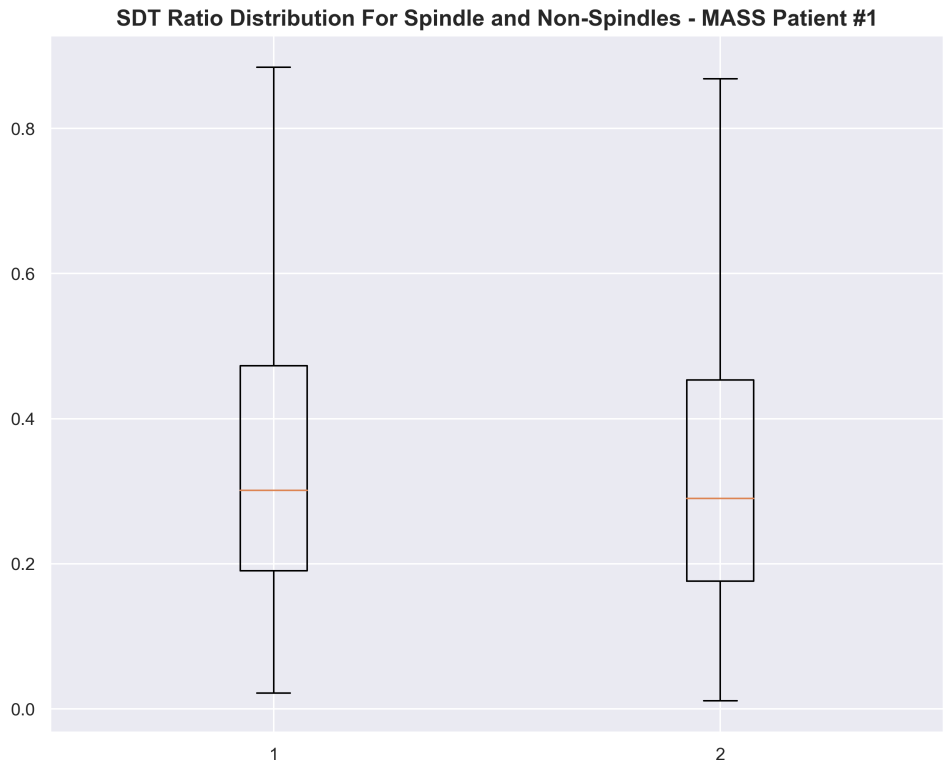


Figure E.1: Distribution of the SDT ratio values for MASS Subject 1 shows no statistical difference between the classes.

Appendix F

PSG Recording Data Pre-Processing - Code Excerpt

```
def read_eeg_and_annot(absolute_filepath, psg_rec, sleep_stage_annot,
   scorer_annot_1, scorer_annot_2 = None):

       """
       Function: 'read_eeg_and_stages'

       This function accepts the raw PSG recording (in EDF format) as well
           as its
       corresponding hypnogram (sleep stage) annotation file and the
          scorer annotation
       for spindles detected by expert scorers.

       The function uses the hypnogram annotations to create 20 second
          epochs for all channels
       in the form of a pandas dataframe. It should also return the raw
          PSG file itself (for future use),
       the intersection of the spindle annotations (if there are TWO
          scorers) and more (under 'returns').
```

```
    Parameters:
            - argv[0] = absolute_filepath: The filepath that contains
              the PSG and annotations (dtype = str)
            - argv[1] = psg_rec: The PSG recording (in EDF) in the
              absolute filepath (dtype = str)
            - argv[2] = sleep_stage_annot: An EDF file containing the
              sleep stages for the corresponding PSG (dtype = str)
            - argv[3] = scorer_annot_1: An EDF file containing the
              spindle annotations from scorer 1.
            - argv[4] = scorer_annot_2: An EDF file containing the
              spindle annotations from scorer 2. Not necessary if there
               is only ONE scorer.

    Returns:
            - raw, epochs_df, scorer_intersection, sampling_frequency
    """

    ###########################################################
    # Step 1: Need to load the raw PSG into the workspace #
    ###########################################################

    psg_file = absolute_filepath + "\\" + psg_rec
    stage_annot = absolute_filepath + "\\" + sleep_stage_annot

    # Load raw MNE file.
    raw = mne.io.read_raw_edf(psg_file, preload=False, verbose=False)
```

\newpage

```
##############################################################
    # Step 2: Load sleep stage annoations onto the raw file and create
        epochs
    ##############################################################

    # Load annotaions per the sleep stage EDF file into workspace. Then
        set annots onto raw
    stages = mne.read_annotations(stage_annot)
    raw.set_annotations(stages)

    # Do not filter the PSG recordings yet. Only do this when accessing
        each individual window.
    # Perform this in the 'compute_features' function.

    # Get the event and event_id from the annotations:
    event, event_id = mne.events_from_annotations(raw)

    # Create epochs from the event and event_id
    epochs = mne.Epochs(raw, event, event_id, tmin=0.0, tmax=20,
        baseline=None) # 20 second epochs (based off stage annotations)

    # Create dataframe from epochs
    epochs_df = epochs.to_data_frame()

    #
        ##########################################################################

    # Step 3: Get the intersection between both scorers (if there are
        two): #
```

```
#
   ##########################################################################


if (scorer_annot_2 == None):

        # This means there is only a single scorer (scorer 1). This
            annotation will serve as the intersection.
        scorer_1 = absolute_filepath + "\\" + scorer_annot_1
        s1_df = pd.read_csv(scorer_1)
        s1_df.columns = ['start', 'duration']
        s1_df['end'] = s1_df['start'] + s1_df['duration']

        scorer_intersection = s1_df

else:

        # In this case, we have TWO scorers. We need to perform an
            inner join to find the intersection
        # between the two scorers.

        # Begin with scorer 1:
        scorer_1 = absolute_filepath + "\\" + scorer_annot_1
        s1_df = pd.read_csv(scorer_1)
        s1_df.columns = ['start', 'duration']
        s1_df['end'] = s1_df['start'] + s1_df['duration']

        # Make a temporary column for the start times rounded to the
             nearest integer:
        s1_df['start_floor'] = s1_df['start'].astype('int32')
```

```
        # Then load scorer 2:
        scorer_2 = absolute_filepath + "\\" + scorer_annot_2
        s2_df = pd.read_csv(scorer_2)
        s2_df.columns = ['start', 'duration']
        s2_df['end'] = s2_df['start'] + s2_df['duration']

        # Make a temporary column for the start times rounded to the
            nearest integer:
        s2_df['start_floor'] = s2_df['start'].astype('int32')

        # Get the 'inner join' of the two scorers' dataframes in
           order to get the ground truth.
        scorer_inner = pd.merge(s1_df, s2_df, how='inner', on=['
           start_floor'])
        scorer_inner.head()

        # Take the mean of the start (onset) tie, duration of the
           spindle and the end time for each scorer:
        scorer_final = pd.DataFrame()
        scorer_final['start'] = (scorer_inner['start_x'] +
           scorer_inner['start_y']) / 2
        scorer_final['duration'] = (scorer_inner['duration_x'] +
           scorer_inner['duration_y']) / 2
        scorer_final['end'] = (scorer_inner['end_x'] + scorer_inner['
           end_y']) / 2

        scorer_intersection = scorer_final

##########################################################
```

```
# Step 4: Get the sampling frequency from the raw file #
#######################################################

sampling_frequency = float(raw.info['sfreq'])

###############
# Final Step #
###############

return (raw, epochs_df, scorer_intersection, sampling_frequency)
```

Appendix G

Epoch Selection From Sleep Stage Annotations - Code Excerpt

```
def epochs_isolate(epochs_df, sleep_stage, ch_name):

        """
        Function: "epochs_isolate"

        'epochs_isolate' accepts the epochs derived from the '
           read_eeg_and_annot' function
        and the desired sleep stage name and channel name to pull out from
           the dataframe.
        This function essentialy works solely with a pandas dataframe.

        Parameters:
                - epochs_df: Dataframe containing the sleep stages and epochs
                    for each PSG channel (EEG, EOG, EMG etc.) (dtype = pandas
                    dataframe)
                - sleep stage: The associated sleep stage desired from the
                   dataframe. Per MASS conventions, the stages are either
                             one of the following: 1, 2, 3, 4, ?, R and W (
                                dtype = str (from command line))
```

```
        - ch_name: The channel type and the electrode placement
            (10-20 system) (e.g. EEG C3-LER)

Returns:
        - eeg_epochs: Filtered dataframe as per the inputs into the
            function.
        - epoch_list: List of all the epochs relevant to the sleep
            stage desired from user:
"""

def_string = "Sleep stage "
stage_str = str(sleep_stage) # If need be:
stage_wanted = def_string + stage_str

#########################################################################

# Step 1: Reset the indices for epochs_df and get the epoch numbers
    #
#########################################################################


temp = epochs_df.reset_index(drop=False)
tmp = temp[temp['condition'] == stage_wanted]
epoch_list = tmp['epoch'].unique()
print("Number of Relevant Epochs = {}".format(len(epoch_list)))

#########################################################################

# Step 3: Pull out the relevant epochs and channel from the
    dataframe: #
```

```
##########################################################################


selected_channels = epochs_df[[ch_name]] # Pull out as a dataframe,
   not a series
eeg_epochs = selected_channels.loc[stage_wanted, :]

return (eeg_epochs, epoch_list)
```

Appendix H

Frame Acquistion - Code Excerpt

```
def manual_class_and_qps(eeg_epochs, epoch_list, scorer_intersection,
   ch_name, window_length, window_stride, sampling_rate):

       """
       Function: 'manual_class_and_qps'

       'manual_class_and_qps' performs manual classification for spindles
          that have been identified by expert scorers.
       The process begins by generating a lower and upper threshold for
          tentative spindles. These thresholds are then used to detect
       preliminary spindles by generating an onset-offset marker pair. Note
           that these annotations are NOT the expert scorer annotations
       and are ONLY used to generate labels for the sake of the
          initialisation of the QPS parameters during the NLLS regression
          stage.

       The expert annotations are used to generate the OFFICIAL label for
          each frame acquired by the frame-acquisition process. If the
       frame OFFICIALLY does NOT pass an expert onset/offset marker, we
          MUST classify it as a non-spindle. The opposite is true for
          spindles
```

```
detected!

        - Uses a 1.0 sec (100 ms) window with a 0.5 sec (50 ms)
          overlap/stride as it moves along the entire 20.0 sec epoch
        - The moving window checks if the expert-scored annotations
          lies in the moving window.
              - If the window has moved past the annotation more
                than 50% of the windows duration (i.e. 50ms), then
                classify as a spindle (1)
              - Else, classify the spindle as a non-spindle (0).
        - For the spindles/non-spindles captured by the frame,
          perform NLLS to compute a, b, c, d, e and f
        - Also compute the other features (residual, RSER, QSER, RMS
          etc.) that was performed in the previous iterations of the
           project
        - Collect all the data and compile all in a single .csv file
          for the single patient.

Parameters:
        - eeg_epochs: Desired epochs (and PSG channel) from the pre-
          processed epochs_df pandas DataFrame
        - epoch_list: Standalone list of epochs the epochs_df pandas
          DataFrame
        - scorer_intersection: DataFrame containing the onset,
          duration and end of each spindle annotated by one/two
          expert scorers.
        - ch_name: The desired PSG channel to be pulled from the
          eeg_epochs dataframe.
        - window_length: The length of the moving window to be used
          to sample (in seconds) a portion of the 20.0 second epoch.
```

```
        - window_stride: The movement of the window (in seconds)
          along the epoch. The stride should ideally be less than
          the window_length to ensure overlap occurs.
        - sampling_rate: The sampling_rate used for the recording of
          the raw PSG recording.

Returns:
        - final_df: The final dataframe containing the QPS parameter
          values and values for all other relevant features. To be
          saved after the function.

"""

# Create variables/parameters to be used for the windowing/framing:
Tw = 1.0        # 1.0 second window length
stride = 0.5      # 0.5 second stride movement of the window along
   the epoch.
frame_len = int(Tw * 1000 / 4)  # frame_len = Tw but in terms of
   indices. Multiply by 1000 to convert to ms. Divide by 4 to go
   from ms to samples.
shift = int(0.5 * 1000 / 4)   # shift = stride buy in terms of
   indices. Mult by 1000 -> convert to ms. Divide by 4 to go from ms
    to samples.
curr_shift_factor = 0    # Factor is used to multiply with the
   stride. This effectively move the window by the time defined by '
   stride' (or 'shift')

# Create a dataframe to collate ALL computed values and features
   from the process below.
final_df = pd.DataFrame()
```

```
# We also want another dataframe comprising ONLY of labels for each
   row of 'final_df':
label_df = pd.DataFrame()

# Completed epochs - For print out purposes on the console.
completed_epochs = 1;

# Create a for-loop to access each epoch in 'epoch_list'. This is
   the outermost loop:
for i in epoch_list:

        # Get the current epoch index being accessed from the list:
        epoch_idx = i
        curr_epoch = eeg_epochs.loc[i]
        signal = curr_epoch[ch_name]

        # Any 'signal' containing NaN values needs to be fixed before
            proceeding to the windowing stage.
        # The EEG signal can be treated as a time-series. Use pd.
           interpolate(method = 'time') to interpolate the missing
           sections in the signal:
        signal = signal.interpolate(method = 'time').fillna(0)

        print()
        print("Current Epoch Number = {}\n".format(epoch_idx))
        print("Epoch's Completed = {}/{}".format(completed_epochs,
           len(epoch_list)))

        completed_epochs = completed_epochs + 1
```

```
# Get the times of each sample in 'signal' captured in the
    curr_epoch. The time array will ONLY be used to check for
    the presence of annotations:
time = curr_epoch.index.values / 1000  # ms -> seconds
time = time + (epoch_idx * 20)   # The current time-interval
    can be accessed by multiplying the time by the epoch
    number * 20 seconds

# Generate the annotations via the automatic detection
    algorithm process
auto_intersection = auto_spindle_detect(time, signal,
    sampling_rate)

# The 'curr_shift_factor' has an upper bound that is
    restricted by the last index of the epoch. We can
    calculate this upper bound before proceeding.
csf_upper_bound = int(np.ceil((len(time) - frame_len) / shift
    ))

#####################################################################

# Feature Computation Using Annotations From Auto Detect
    Algorithm #
#####################################################################



print()
print("################################")
print("# Feature Computation #")
```

```
print("################################")
print()

# Flag to keep check if a frame has went PAST an onset but is
    YET to detect the OFFSET:
# We can assume that at the VERY start, no spindle has been
   detected. Initialise as 1.
annotation_finish = 1;

# Now, we want to use a window to access 2.0 second portions
   of the current epoch. This window will then be shifted
   along the epoch progressively.
for curr_shift_factor in range(csf_upper_bound):

        time_window = time[0 + (curr_shift_factor*shift):
           frame_len + (curr_shift_factor*shift)]   # JUST TO
           CHECK AGAINST ANNOTATION
        signal_window = signal[0 + (curr_shift_factor*shift):
           frame_len + (curr_shift_factor*shift)] # THIS IS
           WHAT WE CARE ABOUT.

        # Boolean condition to check if time_window contains
           an annotation from 'auto_intersection' DataFrame:
        on_condition = (scorer_intersection['start'] >= min(
           time_window)) & (scorer_intersection['start'] <=
           max(time_window))
        off_condition = (scorer_intersection['end'] >= min(
           time_window)) & (scorer_intersection['end'] <= max(
           time_window))
```

```
# If this is true, we should see a particular spindle
   extracted from the annotations.
fall_in_on = scorer_intersection[on_condition]
fall_in_off = scorer_intersection[off_condition]

# Get the lengths of the onset and offset dataset
   generated. Their lengths are the basis for the
   manual classification.
num_onset = len(fall_in_on)
num_offset = len(fall_in_off)

# Print out the current frame number:
print("Frame Number: {}".format(curr_shift_factor))

# A 1.0 second window can either contain ONE spindle
   or NONE at all. We can use a conditional block to
   check.
if (annotation_finish == 1):

        # This means we have YET to detect an onset for
            a new spindle
        # This technically satifies having completely
           past an annotated spindle.
        # Now check whether or not the frame is
           detected an onset or not:

        if (num_onset == 1):

                # Frame captures an onset for the FIRST
                   time OR after exiting a preceding
```

```
    microevent
# Compute the percentage with which the
    frame has PASSED the onset. Must be
    >= 50%!
elapsed_on = max(time_window) -
    fall_in_on['start']
elapsed_on_percentage = float(elapsed_on
     / Tw * 100)

# Need to check if the percentage
    elapsed is greater than or equal to
    50%:
if (elapsed_on_percentage >= 50.0):

        # If so, we should be good to
            classify the frame as having
            captured a spindle.
        # We need to check if the offset
             is in the frame or not and
            computed the percentage

        if (num_offset == 1):

                # Elapsed percentage ALSO
                     for the offset marker
                    .
                elapsed_off = max(
                    time_window) -
                    fall_in_off['end']
```

```
elapsed_off_percentage =
    float(elapsed_off / Tw
     * 100)

# Now, we need to check
    if the offset is <=
    50%. We want it as low
     as possible
# as this means the frame
     is well WITHIN the
    spindle and has not
    passed it considerably
if (
    elapsed_off_percentage
     <= 50.0):

        # THERE IS A
            SPINDLE PRESENT
             IN THE WINDOW.
             Print to user
            to confirm:
        print("A tentative
             spindle occurs
             in this window
            ! Spindle used
            to compute
            features:")

        # This means we
            are in the safe
```

```
     zone to
    classify the
    frame as having
# captured a
    spindle. Update
     the label flag
    :
s_or_ns = 1

print("Percentage
    Time Frame
    Elapsed After
    ONSET = {}".
    format(
    elapsed_on_percentage
    ))
print("Percentage
    Time Frame
    Elapsed After
    OFFSET = {}".
    format(
    elapsed_off_percentage
    ))
print("Label For
    Frame = {}".
    format(s_or_ns)
    )
print()
```

```
# Ready to compute
    QPS parameters
    and features:
feature_df =
   compute_features
   (time_window,
   signal_window,
   sampling_rate,
   s_or_ns)

# Append
   feature_df onto
    final_df:
if (len(final_df)
   == 0):
       # i.e. We
          starting
           fresh.
       for col in
          feature_df
          .columns
          :
              final_df
                 [
                 col
                 ]

                 =

                 feature_df
```

```
                [
                col
                ]


    else:
        # Need a
            temporary

            dataframe
            :
        tmp = pd.
            DataFrame
            ()
        for col in
            feature_df
            .columns
            :
                tmp[
                    col
                    ]

                    =

                    feature_df
                    [
                    col
                    ]
```

```
                final_df =
                    pd.
                    concat([
                    final_df
                    , tmp],
                    axis=0).
                    reset_index
                    (drop=
                    True)

            # We have not yet
                passed the
                offset (> 50%).
                 Hence, we have
                 yet to walk
            # past the
                annotations.
                Toggle the
                annotation_finish
                 flag to 0
            annotation_finish
                = 0

    else:

            # This means we
                most likely
                walked too far
                over the offset
                 (greater than
```

```
    50% of the
    frame length)
# If this is the
    case, then its
    most likely
    than the onset
    was NEVER in
    the frame in
    the first place
    .
# We should expect
     this condition
     to never be
    reached. Still,
     we classify as
     a non-spindle
    (0)
# and toggle the
    annotation_finish
     back to 1 (
    since we have
    technically
    walked way past
     the microevent
    )
s_or_ns = 0

print("PARADOX! (
    TYPE-1)")
```

```
print("Percentage
    Time Frame
    Elapsed After
    ONSET = {}".
    format(
    elapsed_on_percentage
    ))
print("Percentage
    Time Frame
    Elapsed After
    OFFSET = {}".
    format(
    elapsed_off_percentage
    ))
print("Label For
    Frame = {}".
    format(s_or_ns)
    )
print()
plt.show()

# Ready to compute
     QPS parameters
     and features:
feature_df =
    compute_features
    (time_window,
    signal_window,
    sampling_rate,
    s_or_ns)
```

```
# Append
    feature_df onto
     final_df:
if (len(final_df)
    == 0):
        # i.e. We
            starting
             fresh.
        for col in
            feature_df
            .columns
            :
                final_df
                    [
                    col
                    ]

                    =

                    feature_df
                    [
                    col
                    ]


else:
        # Need a
            temporary
```

```
    dataframe
    :
tmp = pd.
    DataFrame
    ()
for col in
    feature_df
    .columns
    :
        tmp[
            col
            ]

            =

            feature_df
            [
            col
            ]


final_df =
    pd.
    concat([
    final_df
    , tmp],
    axis=0).
    reset_index
    (drop=
    True)
```

```
                annotation_finish
                    = 1

        else:

                # Most likely means the
                   frame has NOT captured
                    the offset marker.
                # We are in the safe-zone
                    to classify the frame
                    has having captured a
                    spindle
                s_or_ns = 1;

                print("A tentative
                   spindle occurs in this
                    window! Spindle used
                   to compute features:")
                print("Percentage Time
                   Frame Elapsed After
                   ONSET = {}".format(
                   elapsed_on_percentage)
                   )
                print("NO OFFSET IN FRAME
                   .")
                print("Label For Frame =
                   {}".format(s_or_ns))
                print()
```

```
# # Make a plot of the
   spindle. Jupyter's "
   QTConsole" should be
   able to generate a
   figure
# # whilst performing the
    computations.
# fig = plt.figure(
   figsize=(10,5))
# plt.plot(time_window,
   signal_window)
# plt.axvline(float(
   fall_in_on['start']),
   color='red', label='
   Spindle Onset')
# plt.axvspan(float(
   fall_in_on['start']),
   max(time_window),
   color='red', alpha
   =0.15)
# plt.legend()
# plt.show()

# Ready to compute QPS
   parameters and
   features:
feature_df =
   compute_features(
   time_window,
   signal_window,
```

```
    sampling_rate, s_or_ns
    )

# Append feature_df onto
    final_df:
if (len(final_df) == 0):
        # i.e. We starting
            fresh.
        for col in
            feature_df.
            columns:
                final_df[
                    col] =
                    feature_df
                    [col]

else:
        # Need a temporary
            dataframe:
        tmp = pd.DataFrame
            ()
        for col in
            feature_df.
            columns:
                tmp[col] =
                    feature_df
                    [col]

        final_df = pd.
            concat([
```

```
                                final_df, tmp],
                                 axis=0).
                                reset_index(
                                drop=True)

                        # We need to toggle the '
                            annotation_finish'
                            flag to 0 since we
                            have YET to
                        # reach the END of the
                            microevent (signified
                            by the offset marker)
                        annotation_finish = 0;


        else:

                # Else, the frame has NOT
                    sufficiently passed the onset
                     marker. Classify as a non-
                    spindle.
                s_or_ns = 0;

                print("Percentage Time Frame
                    Elapsed After ONSET = {}".
                    format(elapsed_on_percentage)
                    )
                print("NO OFFSET IN FRAME")
                print("Label For Frame = {}".
                    format(s_or_ns))
```

```
print()

# Ready to compute QPS
   parameters and features:
feature_df = compute_features(
   time_window, signal_window,
   sampling_rate, s_or_ns)

# Append feature_df onto
   final_df:
if (len(final_df) == 0):
        # i.e. We starting fresh.
        for col in feature_df.
           columns:
               final_df[col] =
                  feature_df[col]

else:
        # Need a temporary
           dataframe:
        tmp = pd.DataFrame()
        for col in feature_df.
           columns:
               tmp[col] =
                  feature_df[col]

        final_df = pd.concat([
           final_df, tmp], axis
           =0).reset_index(drop=
           True)
```

```
                # Keep the annotation_finish
                    flag at 1 since this we haven
                    't 'officially' detected a
                    spindle yet
                # and, hence, haven't walked
                    PAST an onset marker.
                annotation_finish = 1;

    else:

            # This means that we ALSO haven't
                captured an onset. Automatically
                classify as a non-spindle
            s_or_ns = 0

            print("NO ONSET IN FRAME")
            print("NO OFFSET IN FRAME.")
            print("Label For Frame = {}".format(
                s_or_ns))
            print()

            # Ready to compute QPS parameters and
                features:
            feature_df = compute_features(
                time_window, signal_window,
                sampling_rate, s_or_ns)

            # Append feature_df onto final_df:
            if (len(final_df) == 0):
```

```
                        # i.e. We starting fresh.
                        for col in feature_df.columns:
                                final_df[col] =
                                    feature_df[col]

                else:
                        # Need a temporary dataframe:
                        tmp = pd.DataFrame()
                        for col in feature_df.columns:
                                tmp[col] = feature_df[col
                                    ]

                        final_df = pd.concat([final_df,
                            tmp], axis=0).reset_index(
                            drop=True)

                # Keep the annotation_finish flag at 1.
                    We still haven't detected an onset:
                annotation_finish = 1


elif (annotation_finish == 0):

        # This means we HAVE walked past an onset but
            have YET to pass an offset completely
        # In this situation, it essentially means we
            are in the MIDDLE of a spindle event
        # and the ONSET has been passed completely BUT
            the offset has NOT been walked past yet
```

```
# We can see this as a case of the frame size
   being considerably SMALLER than the duration
# of the spindle.

# The case where an onset is detected is
   UNLIKELY to be met in this case.
if (num_onset == 1):

        # If an onset so HAPPENS to still be in
           the frame, we need to check
           conditions once more:
        elapsed_on = max(time_window) -
           fall_in_on['start']
        elapsed_on_percentage = float(elapsed_on
            / Tw * 100)

        if (elapsed_on_percentage >= 50.0):

                # Now, we need to check the
                   offset marker:
                if (num_offset == 1):

                        # Elapsed percentage ALSO
                            for the offset marker
                           .
                        elapsed_off = max(
                           time_window) -
                           fall_in_off['end']
                        elapsed_off_percentage =
                           float(elapsed_off / Tw
```

```
    * 100)

# Now, we need to check
   if the offset is <=
   50%. We want it as low
    as possible
# as this means the frame
    is well WITHIN the
   spindle and has not
   passed it considerably
if (
   elapsed_off_percentage
    <= 50.0):

       # This means we
          are in the safe
           zone to
          classify the
          frame as having
       # captured a
          spindle. Update
           the label flag
          :
       s_or_ns = 1

       print("A tentative
            spindle occurs
            in this window
          ! Spindle used
          to compute
```

```
    features:")
print("Percentage
    Time Frame
    Elapsed After
    ONSET = {}".
    format(
    elapsed_on_percentage
    ))
print("Percentage
    Time Frame
    Elapsed After
    OFFSET = {}".
    format(
    elapsed_off_percentage
    ))
print("Label For
    Frame = {}".
    format(s_or_ns)
    )
print()

# Make a plot of
    the spindle.
    Jupyter's "
    QTConsole"
    should be able
    to generate a
    figure
# whilst
    performing the
```

```
    computations.
# fig = plt.figure
    (figsize=(10,5)
    )
# plt.plot(
    time_window,
    signal_window)
# plt.axvline(
    float(
    fall_in_on['
    start']), color
    ='red', label='
    Spindle Onset')
# plt.axvline(
    float(
    fall_in_off['
    end']), color='
    blue', label='
    Spindle End')
# plt.axvspan(
    float(
    fall_in_on['
    start']), float
    (fall_in_off['
    end']), color='
    red', alpha
    =0.15)
# plt.legend()
# plt.show()
```

```
# Ready to compute
    QPS parameters
    and features:
feature_df =
   compute_features
   (time_window,
   signal_window,
   sampling_rate,
   s_or_ns)

# Append
   feature_df onto
    final_df:
if (len(final_df)
   == 0):
       # i.e. We
          starting
           fresh.
       for col in
          feature_df
          .columns
          :
              final_df
                 [
                 col
                 ]

                 =

                 feature_df
```

```
                [
                col
                ]


else:
        # Need a
            temporary

            dataframe
            :
        tmp = pd.
            DataFrame
            ()
        for col in
            feature_df
            .columns
            :
                tmp[
                    col
                    ]

                    =

                    feature_df
                    [
                    col
                    ]
```

```
                final_df =
                    pd.
                    concat([
                    final_df
                    , tmp],
                    axis=0).
                    reset_index
                    (drop=
                    True)

            # We have not yet
                passed the
                offset (> 50%).
                 Hence, we have
                 yet to walk
            # past the
                annotations.
                Toggle the
                annotation_finish
                 flag to 0
            annotation_finish
                = 0

    else:

            # This means we
                most likely
                walked too far
                over the offset
                 (greater than
```

```
    50% of the
    frame length)
# If this is the
    case, then its
    most likely
    than the onset
    was NEVER in
    the frame in
    the first place
    .
# We should expect
     this condition
     to never be
    reached. Still,
     we classify as
     a non-spindle
    (0)
# and toggle the
    annotation_finish
     back to 1 (
    since we have
    technically
    walked way past
     the microevent
    )
s_or_ns = 0

print("Percentage
    Time Frame
    Elapsed After
```

```
    ONSET = {}".
    format(
    elapsed_on_percentage
    ))
print("Percentage
    Time Frame
    Elapsed After
    OFFSET = {}".
    format(
    elapsed_off_percentage
    ))
print("Label For
    Frame = {}".
    format(s_or_ns)
    )
print()

# Make a plot of
    the spindle.
    Jupyter's "
    QTConsole"
    should be able
    to generate a
    figure
# whilst
    performing the
    computations.
# fig = plt.figure
    (figsize=(10,5)
    )
```

```
# plt.plot(
    time_window,
    signal_window)
# plt.axvline(
    float(
    fall_in_on['
    start']), color
    ='red', label='
    Spindle Onset')
# plt.axvline(
    float(
    fall_in_off['
    end']), color='
    blue', label='
    Spindle End')
# plt.axvspan(
    float(
    fall_in_on['
    start']), float
    (fall_in_off['
    end']), color='
    red', alpha
    =0.15)
# plt.legend()
# plt.show()

# Ready to compute
     QPS parameters
     and features:
```

```
feature_df =
    compute_features
    (time_window,
    signal_window,
    sampling_rate,
    s_or_ns)

# Append
    feature_df onto
     final_df:
if (len(final_df)
    == 0):
        # i.e. We
            starting
             fresh.
        for col in
            feature_df
            .columns
            :
                final_df
                    [
                    col
                    ]

                    =

                    feature_df
                    [
                    col
                    ]
```

```
else:
        # Need a
           temporary

           dataframe
           :
        tmp = pd.
           DataFrame
           ()
        for col in
           feature_df
           .columns
           :
               tmp[
                  col
                  ]

                  =

                  feature_df
                  [
                  col
                  ]


        final_df =
           pd.
           concat([
```

```
                                final_df
                                , tmp],
                                axis=0).
                                reset_index
                                (drop=
                                True)

                        annotation_finish
                            = 1

            else:

                    # Perhaps we're still
                       trudging through the
                       event since the window
                        stride is SLOW
                    # Classify as a spindle.
                       We have yet to walk
                       sufficiently past the
                       offset.
                    s_or_ns = 1

                    print("A tentative
                       spindle occurs in this
                        window! Spindle used
                       to compute features:")
                    print("Percentage Time
                       Frame Elapsed After
                       ONSET = {}".format(
                       elapsed_on_percentage)
```

```
    )
print("NO OFFSET IN FRAME
     = {}".format(
    elapsed_off_percentage
    ))
print("Label For Frame =
    {}".format(s_or_ns))
print()

# # Make a plot of the
    spindle. Jupyter's "
    QTConsole" should be
    able to generate a
    figure
# # whilst performing the
     computations.
# fig = plt.figure(
    figsize=(10,5))
# plt.plot(time_window,
    signal_window)
# plt.axvline(float(
    fall_in_on['start']),
    color='red', label='
    Spindle Onset')
# plt.axvspan(float(
    fall_in_on['start']),
    max(time_window),
    color='red', alpha
    =0.15)
# plt.legend()
```

```
# plt.show()

# Ready to compute QPS
   parameters and
   features:
feature_df =
   compute_features(
   time_window,
   signal_window,
   sampling_rate, s_or_ns
   )

# Append feature_df onto
   final_df:
if (len(final_df) == 0):
       # i.e. We starting
           fresh.
       for col in
          feature_df.
          columns:
              final_df[
                 col] =
                 feature_df
                 [col]

else:
       # Need a temporary
           dataframe:
       tmp = pd.DataFrame
          ()
```

```
                        for col in
                            feature_df.
                            columns:
                                tmp[col] =
                                    feature_df
                                    [col]

                        final_df = pd.
                            concat([
                            final_df, tmp],
                             axis=0).
                            reset_index(
                            drop=True)

                annotation_finish = 0

        else:

                # This one is a bit of a paradox
                    . We cannot be waiting to
                    finish yet
                # barely having walked past the
                    onset. This conditional will
                    NEVER be executed
                s_or_ns = 0

                # Ready to compute QPS
                    parameters and features:
                feature_df = compute_features(
                    time_window, signal_window,
```

```
                    sampling_rate, s_or_ns)

                # Append feature_df onto
                    final_df:
                if (len(final_df) == 0):
                        # i.e. We starting fresh.
                        for col in feature_df.
                            columns:
                                final_df[col] =
                                    feature_df[col]

                else:
                        # Need a temporary
                            dataframe:
                        tmp = pd.DataFrame()
                        for col in feature_df.
                            columns:
                                tmp[col] =
                                    feature_df[col]

                        final_df = pd.concat([
                            final_df, tmp], axis
                            =0).reset_index(drop=
                            True)

                annotation_finish = 0;

        else:
```

```
# So NO onset. That's fine. But if we DO
    see an offset marker still in the
   frame...
if (num_offset == 1):

        # Need to check the offset
           marker:
        elapsed_off = max(time_window) -
            fall_in_off['end']
        elapsed_off_percentage = float(
           elapsed_off / Tw * 100)

        # Now, we need to check if the
           offset is <= 50%. We want it
           as low as possible
        # as this means the frame is
           well WITHIN the spindle and
           has not passed it
           considerably
        if (elapsed_off_percentage <=
           50.0):

            # This means we are in
               the safe zone to
               classify the frame as
               having
            # captured a spindle.
               Update the label flag:
            s_or_ns = 1
```

```
print("A tentative
   spindle occurs in this
    window! Spindle used
   to compute features:")
print("NO ONSET IN FRAME
   ")
print("Percentage Time
   Frame Elapsed After
   OFFSET = {}".format(
   elapsed_off_percentage
   ))
print("Label For Frame =
   {}".format(s_or_ns))
print()

# # Make a plot of the
   spindle. Jupyter's "
   QTConsole" should be
   able to generate a
   figure
# # whilst performing the
    computations.
# fig = plt.figure(
   figsize=(10,5))
# plt.plot(time_window,
   signal_window)
# plt.axvline(float(
   fall_in_off['end']),
   color='blue', label='
   Spindle End')
```

```
# plt.axvspan(min(
    time_window), float(
    fall_in_off['end']),
    color='red', alpha
    =0.15)
# plt.legend()
# plt.show()

# Ready to compute QPS
    parameters and
    features:
feature_df =
    compute_features(
    time_window,
    signal_window,
    sampling_rate, s_or_ns
    )

# Append feature_df onto
    final_df:
if (len(final_df) == 0):
        # i.e. We starting
            fresh.
        for col in
            feature_df.
            columns:
                final_df[
                    col] =
                    feature_df
                    [col]
```

```
else:
        # Need a temporary
            dataframe:
        tmp = pd.DataFrame
            ()
        for col in
            feature_df.
            columns:
                tmp[col] =
                    feature_df
                    [col]

        final_df = pd.
            concat([
            final_df, tmp],
             axis=0).
            reset_index(
            drop=True)

# We have not yet passed
    the offset (> 50%).
    Hence, we have yet to
    walk
# past the annotations.
    Toggle the
    annotation_finish flag
     to 0
annotation_finish = 0
```

```
                else:

                        # This means we most
                           likely walked too far
                           over the offset (
                           greater than 50% of
                           the frame length)
                        # Furthermore, there's no
                            onset makrer in the
                           frame. THIS makes
                           sense.
                        # We can classify the
                           captured frame
                           officially as being
                           non-spindle.
                        # And at this point, we
                           have walked
                           sufficiently past the
                           offset marker such
                           that
                        # we are NO longer in the
                            frame of a spindle.
                        s_or_ns = 0

                        print("NO ONSET IN FRAME
                           ")
                        print("Percentage Time
                           Frame Elapsed After
                           OFFSET = {}".format(
                           elapsed_off_percentage
```

```
    ))
print("Label For Frame =
    {}".format(s_or_ns))
print()

# Ready to compute QPS
    parameters and
    features:
feature_df =
    compute_features(
    time_window,
    signal_window,
    sampling_rate, s_or_ns
    )

# Append feature_df onto
    final_df:
if (len(final_df) == 0):
        # i.e. We starting
            fresh.
        for col in
            feature_df.
            columns:
                final_df[
                    col] =
                    feature_df
                    [col]

else:
```

```
                                # Need a temporary
                                    dataframe:
                                tmp = pd.DataFrame
                                    ()
                                for col in
                                    feature_df.
                                    columns:
                                        tmp[col] =
                                            feature_df
                                            [col]

                                final_df = pd.
                                    concat([
                                    final_df, tmp],
                                     axis=0).
                                    reset_index(
                                    drop=True)

                            annotation_finish = 1

                    else:

                            # This means NEITHER onset or
                                offset are in the frame YET
                                we still have yet
                            # to completely pass an event.
                                Classify IMMEDIATELY as a
                                spindle
                            s_or_ns = 1;
```

```
print("A tentative spindle
   occurs in this window!
   Spindle used to compute
   features:")
print("NO ONSET IN FRAME")
print("NO OFFSET IN FRAME.")
print("Label For Frame = {}".
   format(s_or_ns))
print()

# # Make a plot of the spindle.
   Jupyter's "QTConsole" should
   be able to generate a figure
# # whilst performing the
   computations.
# fig = plt.figure(figsize
   =(10,5))
# plt.plot(time_window,
   signal_window)
# plt.show()

# Ready to compute QPS
   parameters and features:
feature_df = compute_features(
   time_window, signal_window,
   sampling_rate, s_or_ns)

# Append feature_df onto
   final_df:
if (len(final_df) == 0):
```

```
                # i.e. We starting fresh.
                for col in feature_df.
                    columns:
                        final_df[col] =
                            feature_df[col]

            else:
                # Need a temporary
                    dataframe:
                tmp = pd.DataFrame()
                for col in feature_df.
                    columns:
                        tmp[col] =
                            feature_df[col]

                final_df = pd.concat([
                    final_df, tmp], axis
                    =0).reset_index(drop=
                    True)

            # We have still yet to walk
               completely past the annotated
                event.
            annotation_finish = 0


    return final_df
```

Appendix I

Bandpower - Code Excerpt

```
def bandpower(signal, sampling_rate, freq_range, frame_size, frame_stride,
   relative=False):

       # Get the lower and upper cutoff frequency as individual variables.
       band = np.asarray(freq_range)
       low, high = band

       # Variable to store the maximum band power within the particular
          band:
       max_bp = 0

       # Initialise parameters for a moving window. The window scans
          through the signal
       # computes the PSD in that frame and then moves along based on a
          stride.
       # The length of this frame is relative to the signal length. For 1.0
           second duration signal
       # We should have a frame size half of the size (e.g. 0.5 second).
          The frame moves along the signal
       # with overlap perhaps equal to the 1/4 of the frame size (e.g. 0.25
           s).
```

```
# Convert the frame size to the equivalent length in samples:
frame_size_samples = int(frame_size * sampling_rate)

# Convert the frame stride to the equivalent stride length in
   samples:
frame_stride_samples = int(frame_stride * sampling_rate)

# Let 'curr_frame' be the current frame number:
curr_frame = 0

# Compute the upper bound for the current frame. This is based on
   the length of the signal (which is going to be like 1.0 seconds)
curr_frame_limit = int( np.ceil((len(signal) - frame_size_samples) /
    frame_stride_samples) )

# Create a for-loop that extracts a frame, computes the PSD and
   stores the bandpower (bp) in max_bp is the current
# computed bandpower is greater than that of the last frame captured
   .
for curr_frame in range(curr_frame_limit):

       # Get the lower and upper limits of the frame being extracted
          :
       frame_min = 0 + (curr_frame * frame_stride_samples)
       frame_max = frame_size_samples + (curr_frame *
          frame_stride_samples)

       # Assign 'data' as the frame extracted from the signal
       data = signal[frame_min:frame_max]
```

```
        # Computing the PSD. Frequency-axis goes up to the Nyquist
            frequency which is Fs/2.
        psd, freqs = psd_array_multitaper(data, sampling_rate,
            adaptive=True, normalization='full', verbose=0)

        # The frequency resolution (df) necessary for integral
            calculation via simpson's rule
        df = freqs[1] - freqs[0]

        # Find index of band in frequency vector
        idx_band = np.logical_and(freqs >= low, freqs <= high)

        # Integral approximation of the spectrum using parabola (
            Simpson's rule)
        bp = simps(psd[idx_band], dx=df)

        if relative == True:
            bp = bp / simps(psd, dx=df)

        if bp >= max_bp:
               max_bp = bp;

# Return the final bandpower value (Relative value recommended)
return max_bp
```

Appendix J

Feature Computation - Code Excerpt

```
def compute_features(time_window, signal_window, sampling_rate, manual_label
    ):

        """
        Function: 'compute_features'

        Function accepts a frame from 'manual_class_and_qps' and computes
            the QPS parameters from the spindle via NLLS.
        The parameter values are initialised from values known a posteriori
            (from other papers + previous analysis of the MASS
        spindles). The function also computers other features such as the
            residual and QPS energy and more. These features are
        defined in more depth in the final report.

        Parameters:
                - time_window: The time-vector corresponding to the raw
                    signal.
                - signal_window: A raw portion of an epoch captured in the
                    manual_class_and_qps function
                - sampling_rate: Sampling rate used to record the EEG signal.
```

```
        - manual_label: The label associated with the annotations
            generated from the automatic detection algorithm.

Returns:
        - feature_df: Pandas dataframe containing the features
            computed from the raw signal passed in as input.

"""

# Set up the time-vector for the NLLS fitting:
tmp = len(time_window)
sampling_rate = sampling_rate
t = np.arange(-tmp/(2*sampling_rate), (tmp-1)/(2*sampling_rate), 1/
    sampling_rate)

# Set up handler function to be used to compute and minimise the
    residual via 'lmfit'
def residual(params, t, data):
        a, b, c = params['a'], params['b'], params['c']
        d, e, f = params['d'], params['e'], params['f']

        # The overall envelope MUST be positive. If a negative value
            is present, set to 0.
        envelope = np.exp(a + b*t + c*t**2)

        for i in range(len(envelope)):
                if envelope[i] < 0:
                        envelope[i] = 0

        carrier = np.cos(d + e*t + f*t**2)
```

```
        model = envelope * carrier
        chi = (data - model)
        return chi

# Set up initialised values for the QPS parameters if manual_label
   == 1
# a, b and c from previous analysis of MASS dataset
# d, e and f from Kulkarni et. al.

##############################################################################

# Here, we use the SDT ratio in order to perform the NLLS parameter
   initialisation. The greatest separation between the spindles
# and non-spindles is when we initialise spindles as the mean values
    (known a priori) and the non-spindles to 0. In this sense
# Analysis of MASS Patient #1 showed that the mean SDT ratio was
   found to be 0.368227 (0.263520) where the number in the brackets
# is the standard deviation of the SDT ratio. We use this range to
   perform the initialisation.
##############################################################################


##############################################################################

# Calculate the spindle-to-(delta+theta) Ratio (SDT Ratio) with
   Multitaper method #
##############################################################################
```

```
psd_frame_size = 0.5   # 0.5-second
psd_frame_stride = 0.25  # 0.25 of a second

spindle_bp = bandpower(signal_window, sampling_rate, [11,16],
   psd_frame_size, psd_frame_stride, relative=True)
delta_bp = bandpower(signal_window, sampling_rate, [0.5, 4],
   psd_frame_size, psd_frame_stride, relative=True)
theta_bp = bandpower(signal_window, sampling_rate, [4,7],
   psd_frame_size, psd_frame_stride, relative=True)

# Compute the SDT Ratio:
sdt_ratio = spindle_bp / (delta_bp + theta_bp)

# Use a conditional to initialise the parameters:
if ((sdt_ratio >= 0.368227 - 0.263520) & (sdt_ratio <= 0.368227 +
   0.263520)):

       a = 0.82
       b = 1.05
       c = -10
       d = 0
       e = 84.5
       f = -0.9

else:

       a = 0.0
       b = 0.0
       c = 0.0
       d = 0.0
```

```
        e = 0.0
        f = 0.0

# Plug in the initialised parameters into lmfit's 'Parameters'
    function:
params = Parameters()
params.add('a', value=a, min=-50.0, max=10)
params.add('b', value=b)
params.add('c', value=c)
params.add('d', value=d)
params.add('e', value=e)
params.add('f', value=f)

# Before minimisation, need to ensure all NaN values have been
    converted to some non-NaN value.
t = np.array(pd.Series(t).interpolate(method='linear').fillna(0))
signal_window = np.array(pd.Series(signal_window).interpolate(method
    ='linear').fillna(0))

# Perform a bandpass filtering operation on the signal_window:
signal_window_filtered = butter_bandpass_filter(signal_window, 11,
    16, sampling_rate, order=5)

# Perform the minimisation via NLLS.
out = minimize(residual, params, args=(t, signal_window_filtered))

# We now need to gather all the parameter values from the 'out'
    return from the minimize function
dict_params = {'a':0, 'b':0, 'c':0, 'd':0, 'e':0, 'f':0}
list_params = ['a','b','c','d','e','f']
```

```
for param in list_params:
    dict_params[param] = out.params[param].value

# Generate the QPS model from the particular signal captured by the
   window:
a, b, c = dict_params["a"], dict_params["b"], dict_params["c"]
d, e, f = dict_params["d"], dict_params["e"], dict_params["f"]
qps = np.exp(a + b*t + c*t**2) * np.cos(d + e*t + f*t**2)

#########################################################
# Compute the residual from the signal_window and qps #
#########################################################

raw = signal_window_filtered
residual = raw - qps

# Converting the dictionary of QPS parameter values into a pandas
   dataframe. The indices are not crucial at this point
param_df = pd.DataFrame(dict_params, index=[0,])

########################
# Energy Calculations #
########################

# Compute Energy of QPS
qps_energy = np.sum(np.square(qps))
param_df["qps_energy"] = qps_energy

# Compute Energy Of REAL Spindle
real_energy = np.sum(np.square(signal_window))
```

```
param_df["real_energy"] = real_energy

# Compute Energy Of Residual
residual_energy = np.sum(np.square(residual))
param_df["residual_energy"] = residual_energy

# QPS-Spindle Energy Ratio (QSER)
param_df['qser'] = qps_energy / real_energy

# Residual-to-Spindle Energy Ratio (RSER)
param_df['rser'] = residual_energy / real_energy

# QPS Spindle Energy Error Percentage (Can Be Used For Boxplot)
param_df['energy_error_percent'] = (np.abs(qps_energy - real_energy)
    / real_energy) * 100.0

##########################
# Frequency Calculations #
##########################

# Compute the peak frequency of the band-pass filtered signal:
hann_window = hann(len(signal_window_filtered))      # Creating a
   hanning window to multiply with the captured signal.
N = len(signal_window_filtered)          # Compute length of signal
freqx = fftfreq(N, 1/sampling_rate)         # Calculate frequency
   bins (x-axis)
freqy = (2/N) * abs(fft(signal_window_filtered * hann_window))  #
   Compute the magnitude spectrum of the real spindle. Need a window
    to ensure no spectral leakage
```

```
centre_freq_index = np.where(freqy == max(freqy))     # Determine
   the array index (in freqy) where the centre (max) frequency
   occurs
real_freq = abs(freqx[centre_freq_index[0][0]])      # Obtain the
   real frequency from the freqx array
param_df["real_freq_hz"] = real_freq        # Assign as another
   column of a dataframe

# Compute the peak frequency based off the generated QPS (parameter
   'e'):
qps_freq = e / (2*np.pi)
param_df['qps_freq_hz'] = qps_freq

# Compute the percentage frequency error between the actual spindle
   and the QPS spindle (Can Be Used For Boxplot)
param_df['freq_error_percent'] = (np.abs(qps_freq - real_freq) /
   real_freq) * 100.0

##########################################################
# Correlation Coefficient Between Raw and QPS Spindle #
##########################################################

param_df['raw_qps_corrcoeff'] = np.corrcoef(pd.Series(qps),pd.Series
   (raw))[0,1]
R = np.corrcoef(pd.Series(qps),pd.Series(raw))[0,1]

#########################################################################

# Add the SDT ratio as a feature in the dataframe. May or may not
   use #
```

```
#########################################################################


param_df['sdt_ratio'] = sdt_ratio

#############################################################
# Manually classify the frame based on the expert scorer. #
#############################################################

param_df['label'] = manual_label

#############################################################
# Return the dataframe containing all computed features: #
#############################################################

feature_df = param_df.copy()

return feature_df
```

Appendix K

Neural Network Construction - Code Excerpt

```
# Split dataset into spindle and non-spindles:
# Randomise the non-spindle set and truncate so equal in length to the
   spindle set:
spindles = df_important[df_important['label'] == 1]
non_spindles = df_important[df_important['label'] == 0].sample(frac=1).
   reset_index(drop = True)

# Truncate non-spindles:
non_spindles = non_spindles.iloc[0:len(spindles), :]

# Concatenate the spindle and non-spindle dataset then jumble once again
   before splitting into train and test set:
final_df = pd.concat([spindles, non_spindles], axis=0).reset_index(drop=True
   )

# Split into X and y (feature vectors and label vector):
X = final_df[['a', 'b', 'c', 'd', 'e', 'f']]
y = final_df['label']
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split the 'all_features' dataframe into training and test subsets. CV is
   performed using 'cross_val_score'
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Normalise training set first and map mean and variance of standardised
   training set onto the test set.
std_scale = StandardScaler().fit(X_train)
X_train = std_scale.transform(X_train)
X_test = std_scale.transform(X_test)

# Importing libraries for machine learning via Keras:
from keras.models import Sequential
from keras.layers import Dense, Activation

model_nn = Sequential()
model_nn.add(Dense(20, activation='relu', input_dim=6))
model_nn.add(Dense(10, activation='relu'))
model_nn.add(Dense(1, activation='sigmoid'))

model_nn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
   accuracy'])

# simple early stopping
from keras.callbacks import EarlyStopping
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience = 50)
```

```
history = model_nn.fit(X_train, y_train, validation_split=0.2, epochs=100,
    callbacks=[es])

# evaluate the model
_, train_acc = model_nn.evaluate(X_train, y_train, verbose=0)
_, test_acc = model_nn.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

# plot training history
fig = plt.figure(figsize=(12,6))
plt.plot(history.history['acc'], label='Training Accuracy')
plt.plot(history.history['val_acc'], label='Validation Accuracy')
plt.legend()
plt.show()

# plot training history
fig = plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.show()
```

Bibliography

[1] "Sleep Spindles and K-Complex Sleep," https://www.tuck.com/sleep-spindles/, May 2017, online; Accessed 22-February-2019.

[2] R. Yuan, X. Di, P. A. Taylor, S. Gohel, Y.-H. Tsai, and B. B. Biswal, "Functional topography of the thalamocortical system in human," *Brain Structure and Function*, vol. 221, no. 4, p. 1971–1984, Apr 2015, accessed 3-April-2019.

[3] J. Żygierewicz, "Analysis of sleep spindles and model of their generation," Ph.D. dissertation, 2000. [Online]. Available: https://www.fuw.edu.pl/~jarekz/

[4] D. Coppieters, P. Maquet, and C. Phillips, "Sleep spindles as an electrographic element: Description and automatic detection methods," *Neural Plasticity*, vol. 2016, pp. 1–19, 01 2016.

[5] C. Iber, S. Ancoli-Israel, A. Chesson, and S. Quan, "The AASM Manual for the Scoring of Sleep and Associated Events: Rules, Terminology and Technical Specifications," *Westchester, IL: American Academy of Sleep Medicine*, 01 2007.

[6] K. O'Keefe, "Measuring sleep: The squiggles explained! (part 1)," Mar 2011. [Online]. Available: https://sciblogs.co.nz/sleep-on-it/2011/03/23/measuring-sleep-the-squiggles-explained-part-1/

[7] A. Bahammam, D. Gacuan, S. George, K. L. Acosta, S. R. Pandi-Perumal, and R. Gupta, *POLYSOMNOGRAPHY I: PROCEDURE AND TECHNOLOGY*, 09 2016, pp. 443–456.

[8] "Eeg introduction," https://www.medicine.mcgill.ca/physio/vlab/biomed_signals/eeg_n.htm, Jan 2008, accessed 3-April-2019.

[9] D. Purves and S. M. Williams, *Neuroscience. 2nd edition.* Sinauer Associates, 2001. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK10996/

[10] P. Georgieva, F. Silva, M. Milanova, and N. Kasabov, *EEG Signal Processing for Brain–Computer Interfaces*, 01 2014, pp. 797–812.

[11] "The role of the spontaneous and evoked k-complex in good-sleeper controls and in individuals with insomnia," *Sleep*, vol. 34, no. 9, p. 1251–1260, Sep 2011. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3157667/

[12] T. Lajnef, S. Chaibi, J.-B. Eichenlaub, P. Ruby, P.-E. Aguera, M. Samet, A. Kachouri, and K. Jerbi, "Sleep spindle and k-complex detection using tunable q-factor wavelet transform and morphological component analysis," in *Front. Hum. Neurosci.*, 2015.

[13] "Stages of Sleep – American Sleep Association," https://www.sleepassociation.org/about-sleep/stages-of-sleep/, Jul 2018.

[14] "What is REM sleep?" https://www.nichd.nih.gov/health/topics/sleep/conditioninfo/rem-sleep, Jan 2018.

[15] "Testing — Sleep Apnea," http://healthysleep.med.harvard.edu/sleep-apnea/diagnosing-osa/testing, May 2011, Online; Accessed 8 April, 2019.

[16] M. Teplan, "Fundamental of eeg measurement," *MEASUREMENT SCIENCE REVIEW*, vol. 2, 01 2002.

[17] G. H. Klem, H. O. Lüders, H. H. Jasper, and C. Elger, "The ten-twenty electrode system of the international federation. the international federation of clinical neurophysiology." *Electroencephalography and clinical neurophysiology. Supplement*, vol. 52, pp. 3–6, 1999.

[18] V. Jurcak, D. Tsuzuki, and I. Dan, "10/20, 10/10, and 10/5 systems revisited: Their validity as relative head-surface-based positioning systems," *NeuroImage*, vol. 34, pp. 1600–11, 03 2007.

[19] *10/20 System Positioning*, Trans Cranial Technologies, 2410 Fortis Tower, 77-79 Gloucester Road, Wanchai, Hong Kong, 2012.

[20] A. Vignoli, G. Oggioni, G. De Maria, A. Peron, M. Nella Savini, E. Zambrelli, V. Chiesa, F. La Briola, K. Turner, and M. P. Canevini, "Lennox-gastaut syndrome in adulthood: Long-term clinical follow-up of 38 patients and analysis of their recorded seizures," *Epilepsy behavior : EB*, vol. 77, pp. 73–78, 11 2017.

[21] S. R. Pandi-Perumal, D. Spence, and A. Bahammam, *Polysomnography: An Overview*, 07 2014, pp. 29–42.

[22] Y. Urakami, A. A., and G. K., "Sleep Spindles – As a Biomarker of Brain Function and Plasticity," *Advances in Clinical Neurophysiology*, 2012.

[23] Z. Fang, L. B. Ray, A. M. Owen, and S. M. Fogel, "Brain activation time-locked to sleep spindles associated with human cognitive abilities," *Frontiers in Neuroscience*,

vol. 13, p. 46, 2019. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2019.00046

[24] Epilepsy Action Australia, "Seizure Types," https://www.epilepsy.org.au/about-epilepsy/understanding-epilepsy/seizure-types-and-classification/, Aug 2017.

[25] E. Wamsley, M. Tucker, A. Shinn, K. E Ono, S. K McKinley, A. V Ely, D. Goff, R. Stickgold, and D. Manoach, "Reduced sleep spindles and spindle coherence in schizophrenia: Mechanisms of impaired memory consolidation?" *Biological psychiatry*, vol. 71, pp. 154–61, 10 2011.

[26] B. C Clawson, J. Durkin, and S. Aton, "Form and function of sleep spindles across the lifespan," *Neural Plasticity*, vol. 6936381, 04 2016.

[27] S. M. Purcell, D. Manoach, C. Demanuele, B. E. Cade, S. Mariani, R. Cox, G. Panagiotaropoulou, R. Saxena, J. Pan, J. Smoller, S. Redline, and R. Stickgold, "Characterizing sleep spindles in 11,630 individuals from the national sleep research resource," *Nature Communications*, vol. 8, p. 15930, 06 2017.

[28] R. Zhao, J. Sun, X. Zhang, H. Wu, P. Liu, X. Yang, and W. Qin, "Sleep spindle detection based on non-experts: A validation study," *PLoS ONE*, vol. 12, 05 2017.

[29] C. O'Reilly and T. Nielsen, "Automatic sleep spindle detection: benchmarking with fine temporal resolution using open science tools," *Frontiers in Human Neuroscience*, vol. 9, p. 353, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnhum.2015.00353

[30] A. J. Palliyali, M. N. Ahmed, and B. Ahmed, "Using a quadratic parameter sinusoid model to characterize the structure of EEG sleep spindles," *Frontiers in Human Neuroscience*, vol. 9, May 2015.

[31] "Non-Stationary Nature of Speech Signal," vlab.amrita.edu/?sub=3&brch=164&sim=371&cnt=1104, 2011, Online; Accessed 26 April, 2019.

[32] S. Devuyst, T. Dutoit, P. Stenuit, and M. Kerkhofs, "Automatic sleep spindles detection — overview and development of a standard proposal assessment method," in *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Aug 2011, pp. 1713–1716.

[33] R. G. Osuna, "L6: Short-time fourier analysis and synthesis." [Online]. Available: http://research.cs.tamu.edu/prism/lectures/sp/l6.pdf

[34] D. Gorur, U. Halici, H. Aydin, G. Ongun, F. Ozgen, and K. Leblebicioglu, "Sleep spindles detection using short time fourier transform and neural networks," in *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, vol. 2, May 2002, pp. 1631–1636 vol.2.

[35] C. M Leavey, M. James, J. Summerscales, and R. Sutton, "An introduction to wavelet transforms: A tutorial approach," *Insight*, vol. 45, pp. 344–353, 05 2003.

[36] J. W. Baker, "Quantitative classification of near-fault ground motions using wavelet analysis," 2007.

[37] A. Graps, ""an introduction to wavelets"," *IEEE Comp. Sci. Engi.*, vol. 2, pp. 50–61, 02 1995.

[38] T. Lajnef, S. Chaibi, J.-B. Eichenlaub, P. Ruby, P.-E. Aguera, M. Samet, A. Kachouri, and K. Jerbi, "Sleep spindle and k-complex detection using tunable q-factor wavelet transform and morphological component analysis," *Frontiers in Human Neuroscience*, vol. 9, p. 414, 2015. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnhum.2015.00414

[39] P. M Kulkarni, Z. Xiao, E. J Robinson, A. Sagarwa Jami, J. Zhang, H. Zhou, S. E Henin, A. Liu, R. S Osorio, J. Wang, and Z. Sage Chen, "A deep learning approach for real-time detection of sleep spindles," *Journal of Neural Engineering*, vol. 16, 02 2019.

[40] H. P. Gavin, "The levenberg-marquardt method for nonlinear least squares curve-fitting problems c ©," 2013.

[41] "Gauss-newton algorithm for nonlinear models."

[42] "Minpack least squares minimization of vector functions."

[43] A. Dertat, "Applied deep learning - part 1: Artificial neural networks," Oct 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

[44] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *CoRR*, vol. abs/1811.03378, 2018. [Online]. Available: http://arxiv.org/abs/1811.03378

[45] J. Brownlee, "A gentle introduction to the rectified linear unit (relu)," Aug 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[46] A. Gramfort, M. Luessi, E. Larson, D. Engemann, D. Strohmeier, C. Brodbeck, R. Goj, M. Jas, T. Brooks, L. Parkkonen, and M. Hämäläinen, "Meg and eeg data analysis with mne-python," *Frontiers in Neuroscience*, vol. 7, p. 267, 2013. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2013.00267

[47] R. Vallat, "Raphael vallat." [Online]. Available: https://raphaelvallat.com/bandpower.html

[48] W. V. Drongelen, "Multitaper power spectrum estimation," 2016.