DM

Sleep Analysis Through Electroencephalogram Cyclic Alternating Pattern A Phase Detection

MASTER DISSERTATION

Arturo José Morais Alves
MASTER IN INFORMATICS ENGINEERING

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

February | 2022

Sleep Analysis Through Electroencephalogram Cyclic Alternating Pattern A Phase Detection

MASTER DISSERTATION

Arturo José Morais Alves
MASTER IN INFORMATICS ENGINEERING

ORIENTATION
Fernando Manuel Rosmaninho Morgado Ferrão Dias

CO-ORIENTATION
Fábio Rúben Silva Mendonça

Sleep Analysis Through Electroencephalogram Cyclic Alternating Pattern A Phase Detection

Arturo José Morais Alves

Supervisor:
Prof. Dr. Fernando Manuel Rosmaninho Morgado Ferrão Dias

Co-Supervisor:
Prof. Dr. Fábio Rúben Silva Mendonça

A thesis presented to the University of Madeira in fulfillment
of the requirements for the degree of Master of Informatics

Funchal – Portugal
February 2022

To my Brother, my Mother and my Father.

Sleep Analysis Through Electroencephalogram Cyclic Alternating Pattern A Phase Detection

Abstract

Sleep deprivation leads to a raise in mortality and other unwanted consequences, as such, the diagnosis of sleep conditions is crucial for medicine. Sleep is a complex physiological process which is subdivided into different stages of deepness. A more detailed analysis of sleep reveals that events may occur that indicate an abrupt change in the brain electrical activity. Cyclic alternating pattern is a repetitive pattern associated with sleep instability that can be detected by analysing bodily signals. This can contribute to the detection of sleep disorders such as obstructive sleep apnea, using the cyclic alternating pattern as an indicator of such conditions. These cycles tend to be hard and slow to manually identify due to the fact that extensive analysis of long recordings of physiological signals by professionals is required. So, the automatic detection of such patterns is necessary. This can be achieved using Machine Learning algorithms such as neural networks that use patient data to train a model that can later recognize patterns within the sleep structure. In this work features extracted from the patient electroencephalographic signal are used by the classifier to learn how to identify the cycles. The classification results were improved by making use of methodologies such as feature selection and model tuning. The final solution presents very good results, obtaining an accuracy of 73%, a sensitivity of 77% and a specificity of 74%.

Keywords: Machine Learning; Neural Networks; Sleep Analysis; CAP cycle; EEG

Sleep Analysis Through Electroencephalogram Cyclic Alternating Pattern A Phase Detection

Resumo

A privação do sono leva a um aumento da mortalidade e as outras situações indesejadas, logo, o diagnóstico de condições médicas relacionadas com o sono é crucial para a medicina. O sono é um processo fisiológico complexo que é subdividido em diferentes fases de profundidade. Uma análise mais detalhada do sono revela que podem ocorrer eventos que indicam uma mudança repentina na atividade elétrica cerebral. O padrão alternante cíclico é um padrão repetitivo associado à instabilidade do sono e que pode ser detetado através da análise de sinais fisiológicos. Esta análise pode contribuir para a deteção das doenças do sono como a apneia obstrutiva, usando o padrão alternante cíclico como um indicador das mesmas. Estes ciclos tendem a ser difíceis e lentos de identificar manualmente devido ao fato de que é necessária uma análise de longos registos de sinais fisiológicos por parte de profissionais. Portanto, a deteção automática destes padrões é necessária. Isto pode ser feito utilizando algoritmos de aprendizagem automática como as redes neuronais, que, utilizando dados de pacientes para treinar um modelo, poderão reconhecer os padrões dentro da estrutura do sono. Neste trabalho, foram extraídas features do sinal eletroencefalográfico de cada paciente para serem utilizadas pelo classificador com o objetivo de identificar estes ciclos. Os resultados da classificação foram melhorados utilizado metodologias como a seleção de features e a otimização do modelo. A solução final apresentou muito bons resultados conseguindo obter uma exatidão de 73%, uma sensibilidade de 77% e uma especificidade de 74%.

Palavras-chave: Machine Learning; Redes Neuronais; Análise do sono; ciclo CAP; EEG

Acknowledgements

Finishing this work was only possible due to a lot of support and help from different people.

I want to thank professor Morgado Dias for all the instructions given and teachings throughout the elaboration of this project.

Also, a special thanks to Fábio Mendonça for always helping with any question I had during the practical implementation of the endeavour.

I wish to also thank my family for all the parental and emotional aid during my time in the course.

Finally, a thanks to all my friends who have always backed me up by giving me advice and encouraged me as well, even at times of distress.

Contents

List of Figures

List of Tables

Abbreviations

AHI	Apnea Hypopnea Index.
AI	Artificial Intelligence.
ANN	Artificial Neural Network.
AUC	Area Under the Curve.
CAP	Cyclic Alternating Pattern.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
DT	Decision Tree.
EEG	Electroencephalography.
EMG	Electromyography.
EOG	Electrooculography.
FFNN	Feed Forward Neural Network.
FN	False Negative.
FP	False Positive.
FPR	False Positive Rate.
GA	Genetic Algorithm.
GD	Gradient Descent.
GDM	Gradient Descent with Momentum.
GPU	Graphics Processing Unit.
GRU	Gated Recurrent Unit.
kMC	K-Means Clustering.
kNN	K-Nearest Neighbours.

LDA	Linear Discriminant Analysis.
LM	Levenberg Marquardt.
LOO	Leave One Out.
LR	Logistic Regression.
LSTM	Long Short-Term Memory.
ML	Machine Learning.
MLP	Multilayer Perceptron.
MMSD	Macro-Micro Structure Descriptor.
mRMR	Minimum Redundancy Maximum Relevance.
MSE	Mean Squared Error.
N-REM	Non-Rapid Eye Movement.
OSA	Obstructive Sleep Apnea.
PCA	Principal Component Analysis.
PSD	Power Spectral Density.
RAM	Random Access Memory.
ReLU	Rectified Linear Unit.
REM	Rapid Eye Movement.
RLS	Restless Legs Syndrome.
RMS	Root Mean Square.
ROC	Receiver Operating Curve.
Rprop	Resilient Backpropagation.
SBLE	Statistical Behaviour of Local Extrema.
SBS	Sequential Backward Selection.
SFS	Sequential Forward Selection.
SMOTE	Synthetic Minority Over-sampling Technique.
SVM	Support Vector Machine.
TEO	Teager Energy Operator.
TN	True Negative.
TNR	True Negative Rate.
TP	True Positive.

TPR	True Positive Rate.
VRAM	Video Random Access Memory.

Chapter 1

Introduction

This chapter contains the motivation for this study, the objectives to be accomplished with this work, and an introduction to the main topics that are going to be discussed throughout the work.

1.1 Objectives and Motivation

It is observable a tendency of increase in the use of Artificial Intelligence (AI) tools to solve problems in multiple areas such as medicine. As such, this work intends to support the detection of an event in the brain electrical activity related to sleep instability, continuing the work of the Bio-inspired Expert Systems and Applications Laboratory. The event is known as the Cyclic Alternating Pattern (CAP) and it can be detected by assembling an automatic detection algorithm that can be later applied to a real world scenario. This model would be beneficial for the detection of sleep related disorders as well as aiding the estimation of sleep related metrics and sleep quality. The system can be implemented by making use of Machine Learning (ML) algorithms such as Artificial Neural Networks (ANNs), where patient data is used to train a model that can learn to identify patterns and distinguish between different classes of samples. This project is also focused on testing and exploring the steps necessary for achieving CAP classification, proposing a solution for the automatic analysis. This includes not only the proposed features and how they are calculated, but also the model parameters and structure. To improve results, feature selection techniques and model tuning procedures were examined to provide the best model possible within the

constraints. Other processes such as data set balancing and scaling were assessed through the analysis of multiple methodologies to find which one is most fit for the task.

An inspection on the state-of-the-art work was also elaborated to analyse different approaches that have been already tested, checking each solution in regards to their methods, results, and other technicalities. This analysis is also important to find spots where improvements can be made.

Overall, the proposed solution is going to provide insights into the problem of automatic A-phase classification and provide new paths for future research on the area.

The following section is going to summarize the background knowledge on the area of study and functions as an introduction to main topics of the work.

1.2 The Role and Importance of Sleep

Sleep is important for the state of mind and body, comprising about one third of the human lifespan. It is commonly agreed that the quality of sleep of each individual is a crucial factor for an overall good quality of life [1]. It has been shown that sleep may have implications on the encoding and consolidation of memories and that sleep deprivation may cause people to have significantly lower performance in tests involving memory, especially those that involve the encoding of emotional content [2]. Sleep deprivation may also lead to long-term repercussions such as increased mortality as a consequence of an elevated probability of heart failure, obesity, stroke, and other related conditions [3]. Sleep quality seems to also mediate the relationship between socio-economic status and better physical health [4]. It is estimated that an increase in the amount of sleep hours on the population of the United States would cause an economic gain between 299.4$ billion and 433.8$ billion in 2020 [5]. Under the same circumstances, Japan, Germany, the UK and Canada would experience an economic gain of, respectively, $93.6 billion to $145.9 billion, $62.3 billion to $40.9 billion, $46.4 billion to $53.8 billion and $13.9 billion to $21.9 billion [5]. In the case of children and adolescents, there is also a correlation between school performance and sleep quality, insufficient sleep and sleepiness [6].

There are also a multitude of sleep related disorders and conditions such as insomnia, Obstructive Sleep Apnea (OSA), Restless Legs Syndrome (RLS),

somnambulism, bruxism, sleep terror, and others [3]. Insomnia is a condition where patients show difficulty initiating or maintaining sleep, and it has been demonstrated that it negatively affects the workspace productivity and public safety [7]. OSA is characterized most notably by choking episodes during sleep which can lead to sleep fragmentation and other complications [8]. Apnea Hypopnea Index (AHI) is the number of apneas and hyponeas that occur per hour of sleep and is used to classify the severity of the condition [9]. RLS is a movement related disorder and its symptoms include unpleasant sensations on the areas between the knees and ankles that cause patients to have an urge to move their lower limbs [10]. Patients that suffer from bruxism tend to grind their teeth during sleep which could lead to other dental related complications [11]. Most of the effects of poor sleep quality can be observed as changes in the rhythm of the brain activity and thus can be diagnosed through metrics that involve the capturing of these signals.

Electroencephalography (EEG) is a non-invasive option with a high temporal resolution for the evaluation of multiple neurological problems including sleep related disorders [12]. Electrodes are positioned in specific locations along the scalp according to the International 10-20 system to capture the electrical brain activity as shown in Figure 1.1. This system describes a set of positions where electrodes should be placed and serves as a standard for EEG analysis [13]. Neurological oscillations can be observed through EEG that are used to characterize sleep and give us a clear understanding of how sleep functions and how it is structured [14].

1.3 The Structure of Sleep

Methodologies that try to use the analysis of sleep through EEG can be split into two distinct fields, one concerning the Sleep Macrostructure and another the Sleep Microstructure.

The Sleep Macrostructure is composed of phases that are long in duration and occur in repeated sequences, each phase corresponding to a different level of sleep deepness. Wave frequency and amplitude are the main descriptors of the sleep stages. Therefore several standardized bands of waves have been described to analyse the sleep segmented structure: delta waves (0.5–4 Hz), theta waves (4–8 Hz), alpha waves (8–12 Hz), sigma waves (12–15 Hz), and beta waves (15–30 Hz). The sleep stages correspond to stages of wake,

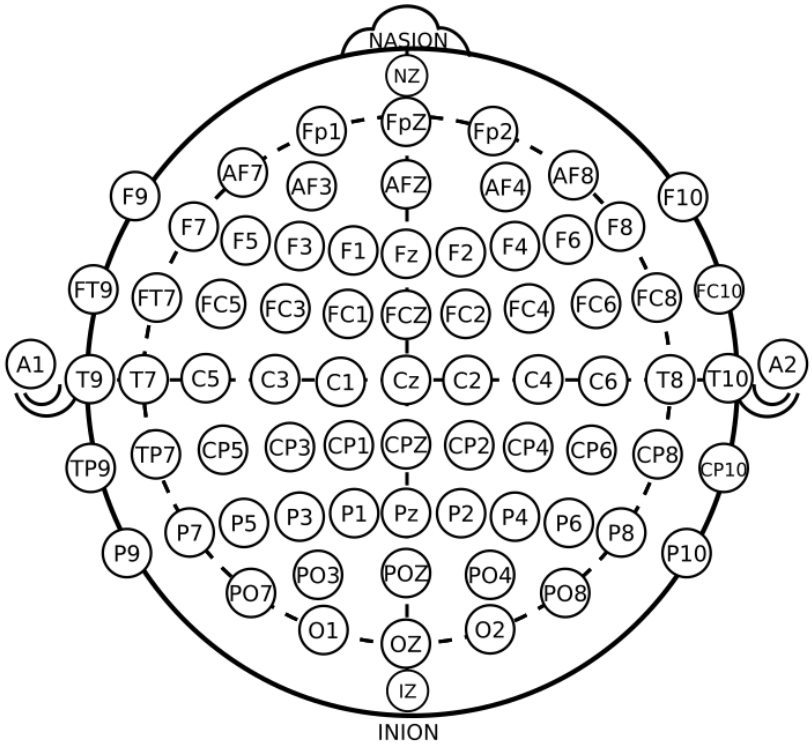

Figure 1.1: International 10-20 System for EEG. Retrieved from [15]

Rapid Eye Movement (REM) and Non-Rapid Eye Movement (N-REM), this last one being further subdivided into three different stages based on the frequency and type of signal present during the stage (N1,N2, and N3). REM sleep is characterized by desynchronised low amplitude brain activity and shows similarities to wakefulness [16], it gains its name due to the frequent eye movements detected with Electrooculography (EOG). EOG consists in measuring the electrical activity of the muscles close to the eyes therefore, measuring the eye movements. Dreaming typically takes place during REM sleep, although it can appear during N-REM sleep [17].

In N-REM sleep, N1 generally arises in the beginning of sleep and shows theta activity, then in the N2 stage consciousness starts to drop and sleep spindles plus k-complexes start to appear, finally N3 stage shows high voltage, slow waves [18]. In some studies, the N3 stage can be sometimes subdivided into N3 and N4 stages. An hypnogram is a graph that shows the sleep stages throughout a sleep period and can be retrieved from EEG, EOG

or Electromyography (EMG) through visual inspection [19]. EMG involves recording the electrical activity of the skeletal muscles. Figure 1.2 shows an example of an hypnogram.

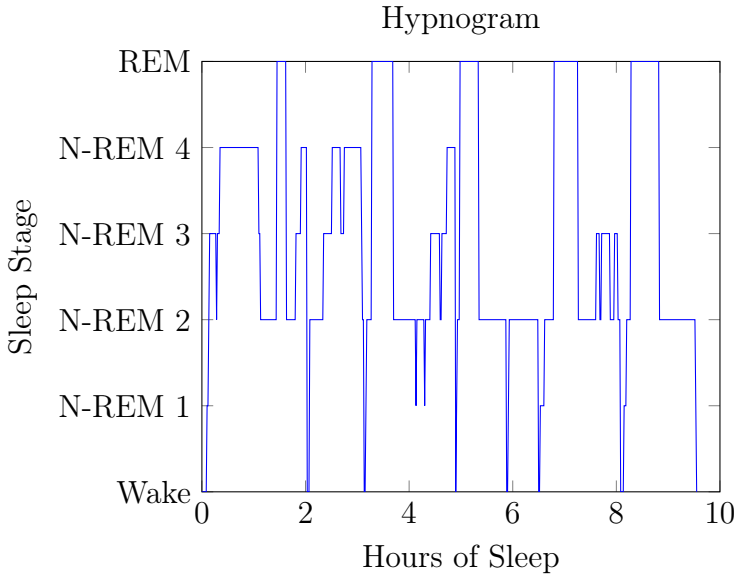

Figure 1.2: Hypnogram showing the sleep stages throughout sleep time. Individual data from [20] was used.

Sleep Microstructure is composed of shorter events with an average duration lower than 30 seconds, this includes vertex sharp transients, sleep spindles, k-complexes, k-alphas, Intermittent alpha, delta bursts, polyphasic bursts and other events that clearly indicate a change from the background brain electrical activity [20].

1.4 CAP Cycle

The CAP is a periodic event registered on the EEG signal that can frequently occur during N-REM sleep and more rarely during REM sleep, and it may be a manifestation of instability [21]. CAP Cycles are classified based on activation phases(A phases) where repetitive EEG patterns that stand out from the background activity occur, and B phases corresponding to the quiescent rhythm between two A phases.

There are three varieties of A phases, A1 where slower rhythms with high amplitude are most frequent, A3 are mostly composed of faster low amplitude waves while A2 contains a mix of slower and faster waves. The A and B phases together make up a cycle. Figure 1.3 shows an EEG sample containing an A phase and an adjacent B phase both constituting a CAP cycle. For a CAP to happen, an A phase and a subsequent B phase must have a duration between 2 and 60 seconds each. If two A phases are separated by a period lower than 2 seconds, the two A phases are joined together. A CAP sequence is an array of two or more consecutive CAP Cycles.A Non-CAP arises when an A-phase in not followed by another A-phase for more than 60 seconds, when this happens, the aforementioned A phase is not counted as part of the CAP sequence [20]. CAP rate is the ratio between the total CAP time and the total N-REM sleep time and higher values of CAP rate are associated with lower sleep quality [21].

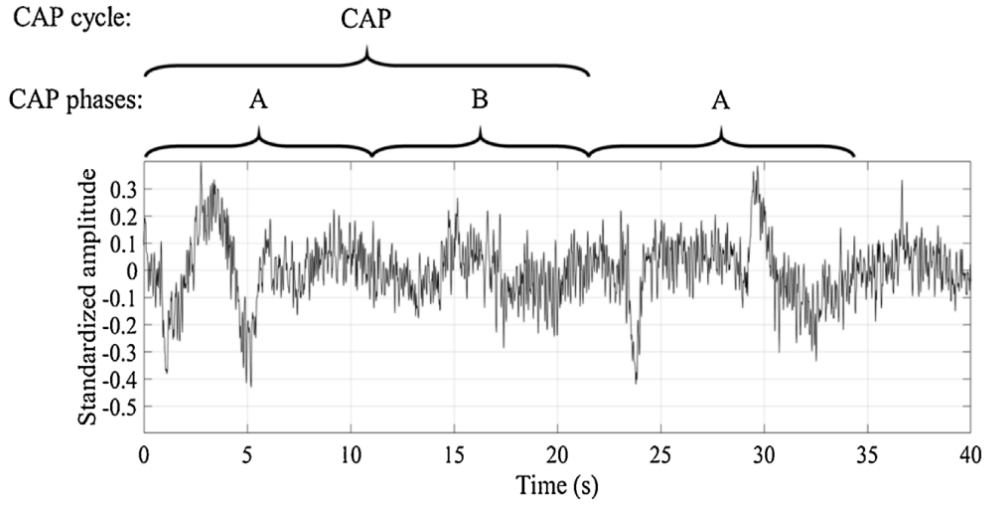

Figure 1.3: EEG sample containing a CAP cycle. As seen, abrupt changes in amplitude occur during the A phases. Retrieved from [22]

The visual detection of CAP cycles is a long and tiring process of looking for patterns on the patient's sleep EEG signal and this is generally done by trained physicians. Sleep sessions tend to be long in duration and, therefore, the analysis of the corresponding EEG recordings is slow and prone to errors. Another relevant aspect is the inter-scorer agreement for the same EEG

results which ranges between 69% and 77.5%. To address this limitations, automatic scoring algorithms are necessary [23].

1.5 Work Structure

This work is divided in five chapters:

1. **Introduction:** Presents the main topics discussed throughout the entire work and gives an introduction to the key components of sleep analysis. Also contains the proposed objectives and the work structure.

2. **Related Work:** Explores the state-of-the-work done on the area of study by viewing what kind of approaches where already tested and whether they succeeded or not. Also presents what contributions are going to be attained by this work.

3. **Methodology:** Constitutes a detailed description of the main steps necessary for the implementation of the proposed system.

4. **Results and Discussion:** Contains a summary and discussion of the results obtained by the proposed system .

5. **Conclusion and Future Work:** Encompasses a conclusion to the work and proposes some future work.

1.6 Key Remarks

As referred in the state of the art, sleep deprivation has a negative impact in not only the physical and psychological health of humans but also influences the economical status of individuals and the school performance of children.

These repercussions are caused by the advent of sleep disorders and so the diagnosis of such conditions is a crucial part of medicine. Techniques that capture physiological signals from the brain electrical activity, such as EEG, are used to diagnose the conditions. CAP is a pattern captured on the EEG which is connected to sleep instability. This cyclic event is scored by professionals, but the manual analysis is slow and prone to errors. Therefore, automatic methods are necessary, advocating the need for this work. The implementation of this work is publicly available on a github repository [24].

Chapter 2

State-of-the-Art

This chapter contains an analysis of the work done on the area of study. A table summarizing all the studies and their results is presented at the end of the chapter.

2.1 Prior Work

Several solutions have been proposed and implemented in the state of the art taking into account the analysis of sleep through CAP to help solving the problem of automatically diagnosing sleep disorders and poor sleep quality. Some approaches use ML algorithms to classify CAP, while others use more traditional methods that do not require the training of a model based on data.

A normalized mean amplitude measurement with thresholds for classification was presented by Barcaro et al. [25]. Eight normal patients with ages between 25 and 32 were examined. Mean amplitude was calculated for the different frequency ranges and compared to the background activity through a normalization process. Four EEG traces where used for the automatic analysis. The proposed descriptors provided good detection of sleep macrostructure and microstructure phenomena. A similar approach was also suggested by Navona et al. [26] where eight EEG traces were used to calculate five similar amplitude and frequency based descriptors. Sleep sessions were carried out with ten healthy individuals. The method achieved a accuracy of 77%, a specificity of 90%, and a sensitivity of 84%. These statistical metrics were calculated using Rosa et al. definition [27].

Karimzadeh et al. [28] tried to directly distinguish the CAP from the background activity using entropy and complexity based features with three classifiers, Linear Discriminant Analysis (LDA), Support Vector Machine (SVM), and K-Nearest Neighbours (kNN). Sleep recordings from four healthy subjects and four subjects with sleep-disordered breathing were used. Feature selection was employed using Sequential Forward Selection (SFS) and Kolgomorov, Shannon, and Sample entropy were selected. The SVM classifier provided the best results with an accuracy of 76%. It was concluded that entropy based features are suitable for CAP detection since they provide a measure of irregularity when transitions from non-CAP to CAP occur.

An initial study by Mariani et al. [29] tried to compute five band descriptors, Hjorth activity in low and high delta bands, and differential variance in the EEG signal to make thresholds for classification and evaluate the significance of these descriptors. The study used medical data from eight healthy people. All features were then remapped to principal components using Principal Component Analysis (PCA). The classifier obtained a mean accuracy of 73.98% using only the first principal component. A subsequent work by Mariani el at. [30] tested the same features on a linear discriminant classifier, a SVM, adaptive boosting, and a supervised ANN. The study was conducted with data from eight healthy individuals. The linear discriminant classifier provided the highest accuracy. A third study by Mariani et al. [31] first isolated the N-REM sleep part of the EEG signal through means of a neural network and then used a segmentation step preceding the feature calculation to maintain similar spectral characteristics within each calculation window. Five spectral band descriptors, Hjorth activity, and a variance descriptor were used for the CAP classification on a linear discriminant function. EEG data from 16 healthy subjects was used. Variable length windows outperformed the traditional fixed length windows probably due to the intrinsic properties of CAP A-phases. Machado et al. [32] focused on the usage of the Teager Energy Operator (TEO) which was calculated for each frequency band and compared with Macro-Micro Structure Descriptor (MMSD) a normalized frequency change descriptor. Data from 30 subjects with nocturnal frontal lobe epilepsy used to carry out the study. TEO is a measure of instantaneous energy. These features were then applied to a threshold computation model and TEO managed to obtain better the best results than MMSD. TEO calculated over the delta band achieved a sensitivity of 80.31% and a specificity of 82.93% on A1 classification.

Largo et al. [33] applied a discrete wavelet transform to the EEG signal

to determine the frequency band features as an alternative to the classical Fourier analysis. Data from six patients was employed. The detection algorithm used moving averages and thresholds to classify A-phases. A Genetic Algorithm (GA) was used to tune parameters necessary for the detection system. Wavelet based descriptors managed to achieve good discrimination.

Mendez et al. [34] evaluated a kNN approach with entropy, spectral, and other types of features to discriminate A phase types with the aim of assessing the feasibility of these methods for this type of classification. The method was tested on five healthy adult subjects. The authors noticed that the A phase classification algorithm achieved a separability similar to the inter-scorer agreement.

Ferri et al. [35] compared a human supervised computer-assisted detection of CAP against the visual detection approach. For that, 11 normal sleep recordings from four different raters was used. The system prompted the user to select threshold configurations until a satisfactory automatic detection was achieved. The study concluded that CAP rate could be estimated using the system with some supervision and correction.

Mendonça et al. [36] analysed a set of nine different classifiers using average power, standard deviation, Shannon entropy, log-energy entropy, autocovariance, TEO and five band descriptors as features. Recordings from 14 subjects were tested, four of which having sleep-disordered breathing and one with bruxism. A Feed Forward Neural Network (FFNN) obtained the best results with an average accuracy of 79%, and Logistic Regression (LR) attained similar results with a simpler implementation. PCA was also employed as an alternative to feature selection but attained weaker results. Power Spectral Density (PSD) on the beta band, Shannon entropy and TEO were selected as the most relevant features by means of SFS on the FFNN classifier. PSD is a measure of power present at different frequencies and can be used to measure frequency information. A second study by Mendonça et al. [37] directly fed a processed version of the EEG signal to three classifiers: a one dimensional Convolutional Neural Network (CNN), a Long Short-Term Memory (LSTM) and a Gated Recurrent Unit (GRU). These classifiers were trained using data from 19 individuals, four of them diagnosed with sleep breathing disorder. The LSTM achieved the best results with an Area Under the Curve (AUC) value of 0.752 for CAP phase estimation. This classifier was then utilized on a home monitoring device that employed EEG sensors to detect CAP.

Niknazar et al. [38] implemented a threshold using Root Mean Square (RMS) of the amplitude to identify the start of an A-phase, then the Statis-

tical Behaviour of Local Extrema (SBLE) method was used to confirm the A-phase and encounter its end. The method was employed on polysomnographic data from five healthy subjects. SBLE involves finding the local extremes of the signal and uncovering the trends presented by those data points. The proposed method managed to achieve good classification with an accuracy of 81%. It required a relatively low amount of computational resources.

Mostafa et al. [39] used a combination of a deep auto encoder and a shallow neural network to classify CAP cycles without the need to handcraft features. 14 subjects were tested, four having sleep-disordered breathing and one having bruxism. The deep auto encoder was used to label A-phases from the EEG data, while the shallow neural network had the task of finding CAP cycles from the A-phase labels. Without creating features, the system obtained decent results with an accuracy of 67%.

2.2 Key Remarks

All of the presented studies provide an insight into the problem at hand by testing approaches that can lead to a solution. As seen, numerical properties can be directly extracted from the EEG signal to help in the classification of CAP. All of the proposed features contribute to their respective detection algorithms. Entropy and complexity related features provide a measure of irregularity. Likewise, TEO contributes as a feature for providing a metric for energy. Finally, frequency and amplitude based features are essential for the problem since CAP phases are defined based on frequency and amplitude information.

Featureless methods provided relatively good results but did not manage to surpass or reach the feature based methods in terms of sensitivity, specificity, and accuracy. Also, these methods utilize more sophisticated and complex classifiers to compensate for the lack of features.

Five of the 15 presented works mentioned using data from patients with some sleep disorder. Since the classification of CAP is used to diagnose sleep conditions, it would be advisable to test the solution on real patients.

In summary, all the solutions provide good results, meaning that CAP classification is achievable using computerized methods. Table 2.1 encapsulates all of the essential information from each individual work, this includes the number of subjects, the method employed, and the accuracy, sensitivity, and specificity values.

Table 2.1: Studies analysed in the state of the art.

Work	Subjects	Method	Acc(%)	Sen(%)	Spe(%)
[25]	8	Frequency features with thresholds for classification	-	-	-
[28]	8	Entropy features fed in 3 classifiers	-	-	-
[32]	30	TEO with thresholds for classification	-	-	-
[33]	6	Wavelet based features with thresholds	-	-	-
[40]	-	Wavelet based features with thresholds	-	-	-
[35]	11	Computer assisted detection	-	-	-
[39]	14	EEG directly fed on a classifier	67	55	69
[29]	8	Features with thresholds for classification	72	52	76
[37]	19	EEG directly fed to three classifiers	76	75	77
[26]	10	Frequency features with thresholds for classification	77	84	90
[36]	14	Multiple features on nine classifiers	79	76	80
[38]	5	threshold RMS and SLBE	81	76	81
[34]	5	kNN with entropy and spectral features	82	87	74
[30]	8	Features fed in 3 classifiers	85	73	87
[31]	16	Linear discriminant function with segmentation step	86	67	90

Chapter 3

Materials

This chapter presents the data set used for the classification algorithm as well as the extracted features. Also, the data related processes that are necessary for the work are presented in this chapter.

3.1 Database

Data are necessary to train the classifier and for that a public database by Terzano et al. [20], considered as a reference for CAP analysis, containing samples of EEG from one single channel during sleep sections for multiple patients was used. For this work, data from 15 healthy individuals and four patients diagnosed with sleep disordered breathing were put to use for the development process. Each collection of samples for every patient has a different sampling frequency (100, 128, 200, 256 or 512 Hz) but the A-phase classification labels are organized such that one label is associated with each second of brain activity. A re-sampled version of the data set where all patients have the same sampling frequency (100 Hz) was also used. In total, the data set contains 562311 samples, of which 13.7% are A-phase samples. Sessions with patients have a duration ranging approximately between six and ten hours. Data from the different sleep stages was captured, therefore, hypnograms for each patient are also available. Table 3.1 roughly summarizes the contents of the database by providing the number of samples, percentage of A-phase samples, sampling frequency, and variance of the amplitude value applied over the whole signal for each patient. Figure 3.1 presents a histogram containing the number of each type of sample on the whole database.

Table 3.1: Table containing data related to each patient set of samples

Patient	Num Samples	% of A samples	Sampling Freq	Var. Amplitude
1	34410	11.9%	512 Hz	392.5
2	30000	10.4%	512 Hz	326.8
3	30000	7.8%	512 Hz	580.3
4	30330	7.4%	100 Hz	425.3
5	30240	13.1%	512 Hz	538.3
6	31200	13.7%	128 Hz	325.1
7	29520	9.0%	128 Hz	883.7
8	30000	10.9%	100 Hz	0.0
9	30840	6.8%	128 Hz	674.6
10	25800	10.6%	512 Hz	1707.2
11	31590	9.8%	512 Hz	743.0
12	29100	12.7%	200 Hz	702.5
13	29040	12.0%	200 Hz	436.4
14	29310	12.8%	200 Hz	480.7
15	30770	13.9%	100 Hz	253.9
16	23131	8.3%	256 Hz	13.0
17	30450	26.3%	256 Hz	18.2
18	23760	31.8%	512 Hz	22.3
19	32820	32.2%	256 Hz	20.8

3.1.1 Data set balancing

One of the problems that plagues learning as a whole is learning from imbalanced data. Most learning algorithms, including neural networks, assume that the number of samples between classes is balanced but on many occasions this is not the case [41]. The algorithms often tend to give focus to the majority class as it is more well defined. A study by Poolsawad et al. [42] showed that balancing techniques improved classification in terms of recall and precision in five different classifiers, including Multilayer Perceptron (MLP), SVM, and Decision Tree (DT). The current selected data set has two or four different classes depending on whether is intended to classify the different A-phase sub-types or to detect A-phases or not-A phase epochs.

The second option was followed in this work. It is important to bear in mind that the A-phase classes are a minority when compared to the majority not-A phase class. This is to be expected since the occurrence of an A-phase is a sporadic event. This issue can be solved by making usage of re-sampling methods or cost-sensitive learning.

Re-sampling methodologies tackle the problem by removing or adding samples to balance all classes. Under-sampling, for example, consists in reducing the number of samples of the majority classes. A simple implementation of this would be random under-sampling, which randomly deletes samples from the majority classes [43]. Removing samples that are close to the decision boundary is a technique that also provides good results. Tomek links are pairs of samples from two opposing classes that are closest to each other in terms of euclidean distance, removing the sample from the majority class would allow for a better classification [44].

There is also the option of over-sampling which tries to increase the size of the minority classes by creating synthetic samples or by repeating existing samples [43]. Synthetic Minority Over-sampling Technique (SMOTE), for instance, creates synthetic samples by doing linear interpolation between data points of the minority classes [45]. For this to happen, a random point from the minority class is picked and its nearest neighbour is found, then, a new synthetic point is generated by interpolating the two points. The nearest neighbour is found by finding the closest point in terms of euclidean distance, then, a random number between zero and one is generated to interpolate the two points. Figure 3.2 presents a visual demonstration of how SMOTE functions.

On the other hand, cost-sensitive learning does not try to change the data set itself but rather tries to change the way the algorithm learns by adding different cost values for each class during the learning process [46]. This is done by directly adding these cost values to the error function before updating the networks weights. This works not only as a balancing technique but also as a mean to increase the importance of a specific class if desired [46]. The cost values used for cost sensitive learning can be calculated using equation 3.1, where n_{total} is the total number of samples, n_c is the number of samples for class c and C is the number of classes.

$$weight_c = \frac{n_{total}}{n_c C} \tag{3.1}$$

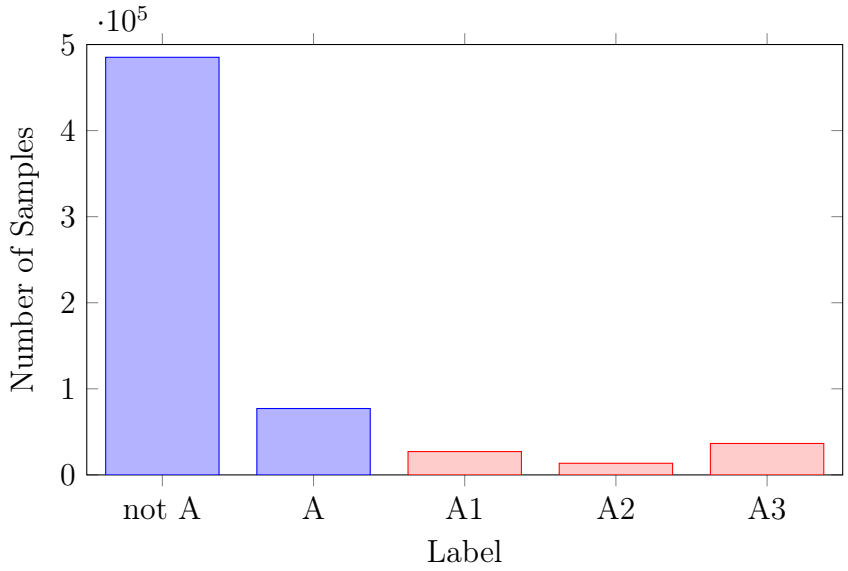

Figure 3.1: Histogram showing the number of samples in each of the CAP labels for the whole data set. The red bars indicate the amount of samples in three subtypes of A phases.

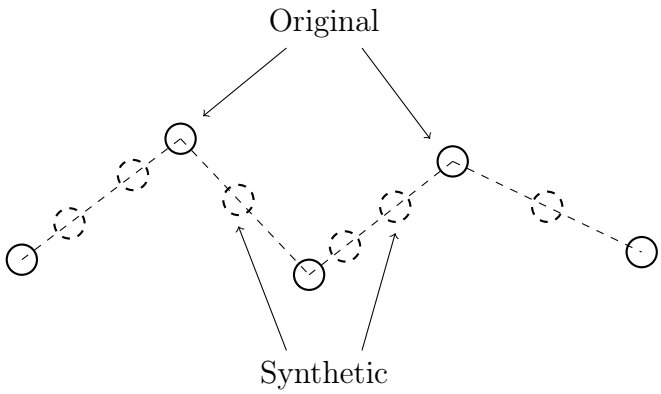

Figure 3.2: A visual representation of how SMOTE generates synthetic samples. In this example, from the five existing points in the feature space six new points were created.

If not carefully considered, some of these techniques might lead to a bad performing model. For instance, random oversampling has the chance of increasing the likelihood of overfitting happening due to the repetition of samples. Furthermore, random undersampling may discard potentially useful data from the majority classes [43]. For its simplicity, cost sensitive learning was the balancing technique employed in this work since it has the advantage of not requiring the alteration of the original distribution of the data.

3.2 Features

3.2.1 Feature Extraction and Calculation

Features or descriptors are numeric properties that have to be directly extracted from the available data in order to be used by the classification algorithm. Although the learning algorithm could learn directly from the EEG values, it would be advisable to use features that are somewhat more closely related to classification problem. So, based on the literature and problem knowledge, a set of features was extracted to be used for A-phase classification. These features are listed together with a description, a mathematical formula for its calculation, and the MATLAB code required to compute the feature:

- **Mean Amplitude Value:** Average value of the amplitude calculated over the EEG window. Also known as the first statistical momemtum [47]:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{3.2}$$

```
inputs_i(i,1) = mean(signal);
```

- **Variance of the Amplitude Value:** Variance of the amplitude value calculated over the EEG window. Can been interpreted as the average

square distance between each observation and the mean. Also known as the second statistical momentum [47]:

$$Var(X) = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 \tag{3.3}$$

```
inputs_i(i,2) = var(signal);
```

- **Skewness of the Amplitude Value:** Skewness value computed on the EEG window. It is a measure of asymmetry in the distribution of the amplitude values. Also known as the third statistical momentum [48]:

$$Skew(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^3}{(\sqrt{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2})^3} \tag{3.4}$$

```
inputs_i(i,3) = skewness(signal);
```

- **Kurtosis of the Amplitude Value:** Kurtosis value computed on the EEG window. It describes the flatness of the distribution curve. Also known as the fourth statistical momentum [48]:

$$Kurt(X) = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^4}{(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2)^2} \tag{3.5}$$

```
inputs_i(i,4) = kurtosis(signal);
```

- **Amplitude Variation:** Calculated for each epoch using the following equation where W_e is the window of EEG values for the specified epoch:

$$Av_e = max(W_e) + max(W_{e-2}) - max(W_{e-1}) \tag{3.6}$$

```
inputs_i(i,5) = max(signal)+(prev2EpochMax-prevEpochMax)
prev2EpochMax = prevEpochMax;
prevEpochMax = max(signal);
```

- **Standard Deviation of Amplitude Value:** Standard deviation value of the amplitude calculated over the EEG window. Same as the square root of the variance [47]:

$$\sigma_X = \sqrt{Var(X)} \tag{3.7}$$

```
inputs_i(i,6) = std(signal);
```

- **Amplitude Range Difference:** Maximum amplitude value subtracted by the minimum amplitude value on the EEG window [47]:

$$Ard(X) = max(X) - min(X) \tag{3.8}$$

```
inputs_i(i,7) = max(signal) - min(signal);
```

- **Shannon, Log Energy, Renyi and Tsallis Entropy:** Entropy is a measure of disorder or randomness in data. Entropy can be thus calculated in multiple forms. Shannon and log energy entropies are calculated as stated in equations 3.9 and 3.10 in MATLAB's implementation of the *wentropy* function [49]. Renyi and tsallis entropy are generalized entropy functions which can be calculated as stated in equations 3.11 and 3.12 [50, 51].

$$Shannon(X) = -\sum_{i=1}^{n} X_i^2 log(X_i^2) \tag{3.9}$$

$$Logenergy(X) = \sum_{i=1}^{n} log(X_i^2) \tag{3.10}$$

$$Renyi(X) = \frac{1}{1-\alpha} log(\sum_{i=1}^{n} p_i^{\alpha}) \tag{3.11}$$

$$Tsallis(X) = \frac{k}{q-1}(1 - \sum_{i=1}^{n} p_i^{q}) \tag{3.12}$$

```
inputs_i(i,8) = wentropy(signal,'shannon');
inputs_i(i,9) = wentropy(signal,'log energy');
inputs_i(i,10) = renyi_entropy_lps(signal');
inputs_i(i,11) = tsallis_entropy_lps(signal');
```

- **Autocovariance:** Covariance between the signal and multiple time shifted versions of the same signal. Equation 3.13 shows how to calculate covariance between X and Y. Since the autocovariance function returns various values for different time delays, the mean of the absolute values is used as feature [47, 52]:

$$Cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{N} \tag{3.13}$$

```
inputs_i(i,12) = mean(abs(xcov(signal)));
```

- **Autocorrelation:** Same as autocovariance but by calculating the correlation between the signal and time shifted versions of the same signal. Equation 3.14 shows how to calculate the correlation coefficient between Y and X. So the autocorrelation is calculated as follows [47, 52]:

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sigma_X \sigma_Y} \tag{3.14}$$

```
inputs_i(i,13) = mean(abs(xcorr(signal)));
```

- **TEO:** Instantaneous energy measure which is calculated as stated in equation 3.15. Since TEO returns a value for each time instant, the mean of these values is used as a feature [53]:

$$Teo(t) = (X_t)^2 + X_{t-1}X_{t+1} \tag{3.15}$$

```
teager = zeros(1,size(signal,1)-2);
for j = 1:1:(size(signal,1)-2)
    teager(j) = signal(j+1)*signal(j+1) + signal(j)*signal(j+2);
end
inputs_i(i,14) = mean(teager);
```

- **PSD on the Delta, Theta, Alpha, Sigma, Beta 1, and Beta 2 Bands:** A measure of power present in each of the frequency bands. PSD was calculated for the delta (0.5-4 Hz), theta (4-8 Hz), alpha (8-12 Hz), sigma (12-15 Hz), beta 1 (15-24 Hz), and beta 2 (24-30 Hz) bands. Both Welch's Overlapped Segment Averaging Spectral Estimation and wavelet decomposition were used to estimate PSD.

  Welch's method calculates the periodogram for multiple overlapping segments of the time series and averages the values to obtain an estimate for the PSD. Since the method returns the PSD for each frequency, the area between the two frequency boundaries of the band is used as the feature.

```
%calculating a few parameters
nfft = floor(signalSize/divideNfft);
noverlap = floor(nfft/2);
fs = samplingFreq;

%obtaining the PSD graph
[pxx,f] = pwelch(signal,hanning(nfft)
,noverlap,nfft,fs);
pxx2 = 10*log10(pxx*fs/nfft);

%obtaining band frequencies indexes on the graph
auxUnit = samplingFreq/(2*noverlap);
```

```
baseFrequencies = [0.5,4,8,12,15,24,30];
freqMatrix = floor(baseFrequencies/auxUnit);
freqMatrix = auxUnit*freqMatrix;

DeltaBaseFreq = find(f == freqMatrix(1));
ThetaBaseFreq = find(f == freqMatrix(2));
AlphaBaseFreq = find(f == freqMatrix(3));
SigmaBaseFreq = find(f == freqMatrix(4));
Beta1BaseFreq = find(f == freqMatrix(5));
Beta2BaseFreq = find(f == freqMatrix(6));
Beta2MaxFreq = find(f == freqMatrix(7));

%obtaining PSD for each frequency band
inputs_i(i,15) = trapz(pxx2(DeltaBaseFreq
:(ThetaBaseFreq-1)));
inputs_i(i,16) = trapz(pxx2(ThetaBaseFreq
:(AlphaBaseFreq-1)));
inputs_i(i,17) = trapz(pxx2(AlphaBaseFreq
:(SigmaBaseFreq-1)));
inputs_i(i,18) = trapz(pxx2(SigmaBaseFreq
:(Beta1BaseFreq-1)));
inputs_i(i,19) = trapz(pxx2(Beta1BaseFreq
:(Beta2BaseFreq-1)));
inputs_i(i,20) = trapz(pxx2(Beta2BaseFreq
:Beta2MaxFreq));
```

Wavelet decomposition decomposes the signal into approximation and details coefficient arrays using low-pass and high-pass filters. The approximation coefficients contain the low frequency information of the original signal, while the details coefficients hold the high frequency data. As seen on figure 3.3, the low amplitude high frequency component is filtered on the approximation coefficients while the high amplitude low amplitude component is filtered on the details coefficient. These coefficients can be further decomposed using the same procedure to create new frequency data with smaller ranges. The original signal can be reconstructed by employing the inverse wavelet transform on the coefficients. To estimate the PSD value for a specific frequency band, wavelet decomposition was conducted in multiple levels until

the desired frequency bands are achieved, then, the signal needs to be reconstructed using the coefficients than contain the frequency information for that band. With the reconstructed signal, power can be estimated using equation 3.16 where X is the reconstructed signal and n is the number of amplitude samples in that signal.

$$PSD(X) = \frac{\sum_{i=1}^{n}(X_i)^2}{n} \tag{3.16}$$

There were also included other 72 features which were obtained by directly applying the first, second, third, and fourth statistical momentum's (mean, variance, skewness, and kurtosis), and the Shannon, log energy, Tsallis and Renyi entropies to nine of the wavelet decomposition coefficient arrays. The frequency range captured by each set of coefficients is presented in table 3.2. These features were calculated using a resampled version of the same data set where all patients have a 100Hz sampling rate. This facilitated the calculations.

Table 3.2: Wavelet decomposition coefficients used for the calculation of the additional features.

Designation	Frequency Range
c1	0 to 50 Hz
c3	0 to 25 Hz
c4	25 to 50 Hz
c5	0 to 12.5 Hz
c6	12.5 to 25 Hz
c9	0 to 6.25 Hz
c10	6.25 to 12.5 Hz
c13	0 to 3.125 Hz
c14	3.125 to 6.25 Hz

All of the features were calculated directly from the EEG data by applying operations to each of the label's corresponding windows of values, in other words, one value per epoch (or second). The EEG values were standardized for each patient separately before running the feature calculation step due to the different amplitude characteristics of the different EEG signals. Overlapping of three time epochs was also employed to provide more contextual

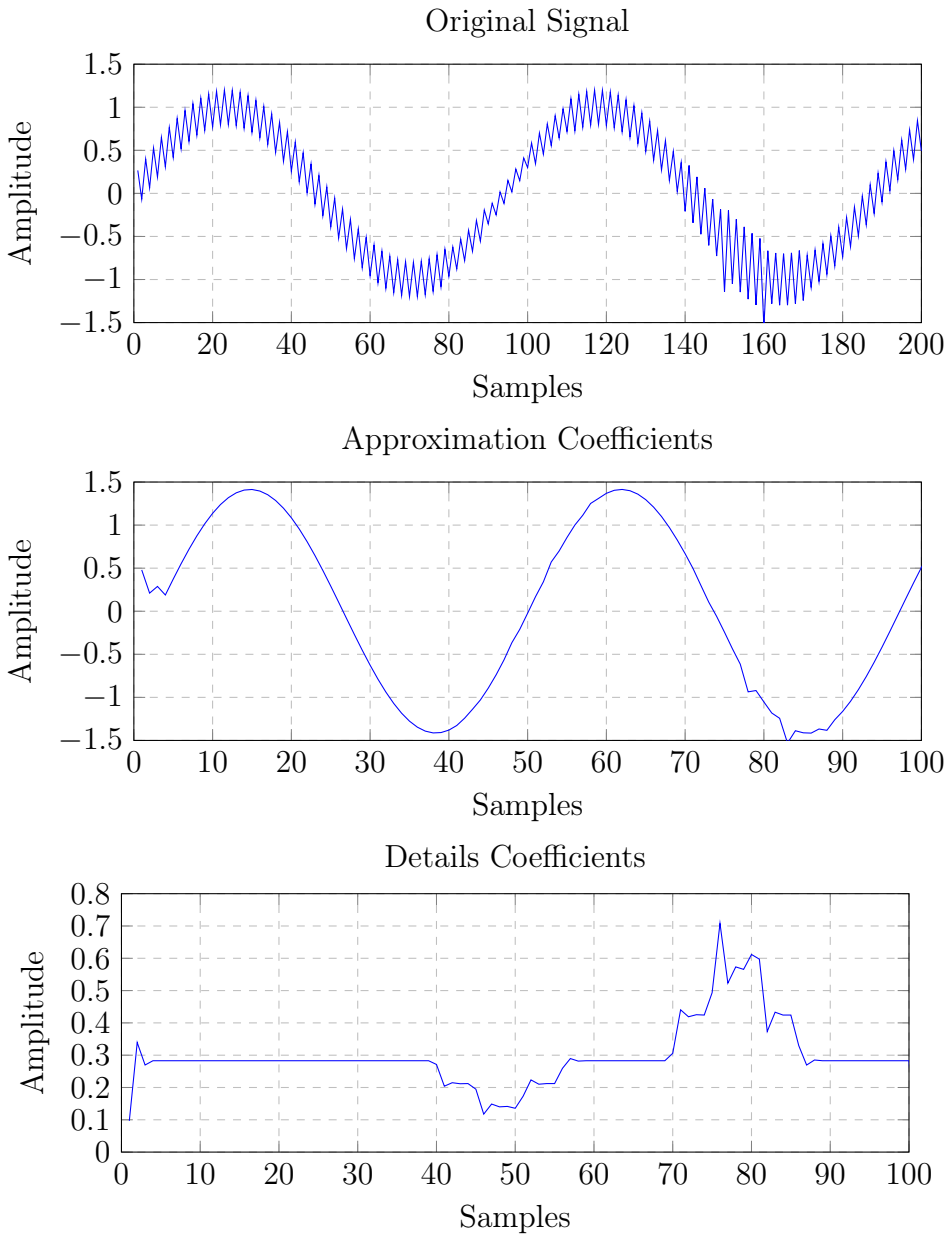

Figure 3.3: Decomposing a signal into the approximation and details coefficients.

information for the feature calculation. Figure 3.4 illustrates how calculation windows look. The middle area corresponds to the epoch's set of values and the areas to the sides are the overlapping values(3 time epochs behind and 3 time epochs forward). Feature values for time epoch number 6 would be calculated using time epochs 3 to 9. Each feature with the exception of the Amplitude Variation is calculated using this overlapping technique.

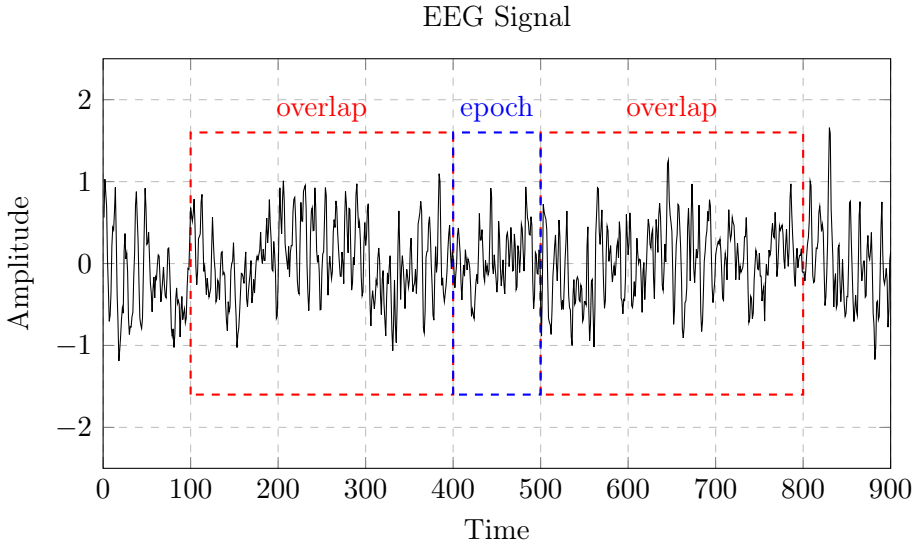

Figure 3.4: EEG signal for patient number 8 showing an example of the calculation window necessary to obtain the features for epoch number 5.

All features were computed using a MATLAB script that gathers each feature value for each window of EEG values. Most of the necessary functions were already available on MATLAB, but some had to be either implemented or calculated using custom packages.

```
...
%for each sample
for i = 1:1:sampleNum
    %obtain the EEG indexes for the sample
    upperIndexes = min([i+calculationRanges;zeros(1,15)+sampleNum]);
    lowerIndexes = max([i-calculationRanges-1;zeros(1,15)]);
    upperBounds = upperIndexes*sampleSize;
    lowerBounds = lowerIndexes*sampleSize + 1;
```

```
        %compute the feature
        signal = eegSensor(lowerBounds(1):upperBounds(1));
        inputs_i(i,1) = mean(signal);%mean
        ...
    end
    ...
```

3.2.2 Feature and Data Scaling

Since features have different distributions, they may also have different effects on the way the ML algorithm behaves. Also, different patients seem to show different EEG characteristics in terms of amplitude which can be observed in the variance of the amplitudes in Table 3.1. To prevent features or patients from having a bigger or lower influence than others on the learning process, scaling is necessary. Scaling has been shown to be beneficial on neural network training in terms of Mean Squared Error (MSE) [54]. There are two main approaches to this problem. Min-max normalization, which involves re-scaling the distribution so that it ends up ranging between two values, usually zero and one, and Z-score standardization, where data is scaled so that it ends up with a mean of zero and a standard deviation of one [55]. Equation 3.17 represents the calculations necessary to normalize the X variable according to the Min-max normalization process and Equation 3.18 shows how to do Z-score standardization, where σ_X is the standard deviation of the X variable and $\bar{X}$ is the average value for the same variable [55].

$$X_{scaled} = \frac{X - min(X)}{max(X) - min(X)} \tag{3.17}$$

$$X_{scaled} = \frac{X - \bar{X}}{\sigma_X} \tag{3.18}$$

In this work, Z-score standardization was chosen for feature scaling since it is less prone to problems arising from outliers [56]. It has also been demonstrated that a similar metric called statistical standardization provided bet-

ter results than Min-max normalization in neural networks [54]. With this in mind, each feature was individually scaled and feature data from each patient were separately standardized. Data scaling was also applied to the EEG amplitude values for each patient before running the feature calculation process. Figure 3.5 shows how scaling preserves the overall shape of the data while changing its range of values.

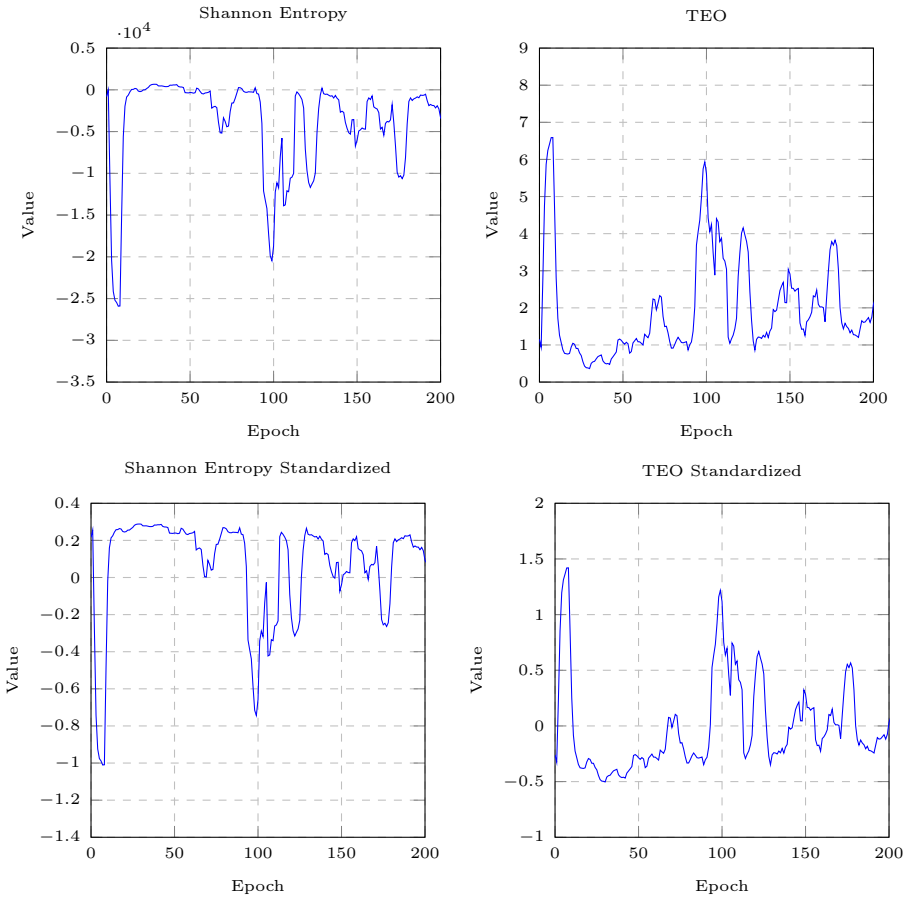

Figure 3.5: Shannon Entropy and TEO before and after applying feature scaling using Z-score standardization.

3.3 Key Remarks

A data set with medical information from 19 individuals was used for the classification algorithm. This data includes EEG samples from sleep sessions with an average duration between six and ten hours. Four of the 19 studied patients are diagnosed with a sleep breathing disorder.

From the EEG signals, a total of 98 features were extracted. Complexity features such as Shannon entropy were included, as well as statistical, amplitude, frequency and energy related features. These features were standardized before putting them to use in the learning algorithm due to the different EEG characteristics displayed by different patients and the magnitude of different features.

Chapter 4

Methods

This chapter describes the classification algorithm and explains a few of the processes necessary to improve its results. This includes model tuning and feature selection.

4.1 A-phase Classification Algorithm

The A-phase classification algorithm could be implemented using various methods. Traditional methods to implement the recognition algorithm that do not require the use of AI are hard to create due to the complex nature of sleep EEG analysis. ML is a subarea of AI where algorithms learn to do tasks by training on data or by training on an environment. As such, these algorithms are extremely versatile and can be used to solve a variety of problems. ML can be further subdivided into supervised and unsupervised learning. Supervised learning takes into account the real output values for the model and together with the predicted outputs calculates an error function which is used change to the model in order to become adapted to the task. Supervised learning includes algorithms such as ANNs, LR, SVMs and kNN [57, 58].

Unsupervised learning does not take any real output values into account and only uses the input data to improve the model performance. Unsupervised learning includes clustering algorithms such as K-Means Clustering (kMC) and dimensionality reduction techniques such as PCA [57, 58].

4.1.1 The Proposed Algorithm

A classification FFNN was proposed to implement the automatic A-phase detection algorithm. A FFNN is a structure containing layers of multiple interconnected neurons and behaves similarly to biological neural networks [59]. It is composed by an input layer, an output layer, and one or more hidden layers. The input neurons pass the feature values to the first hidden layer while the outputs correspond to the variables or classes for prediction. Figure 4.1 shows a simple representation of a FFNN.

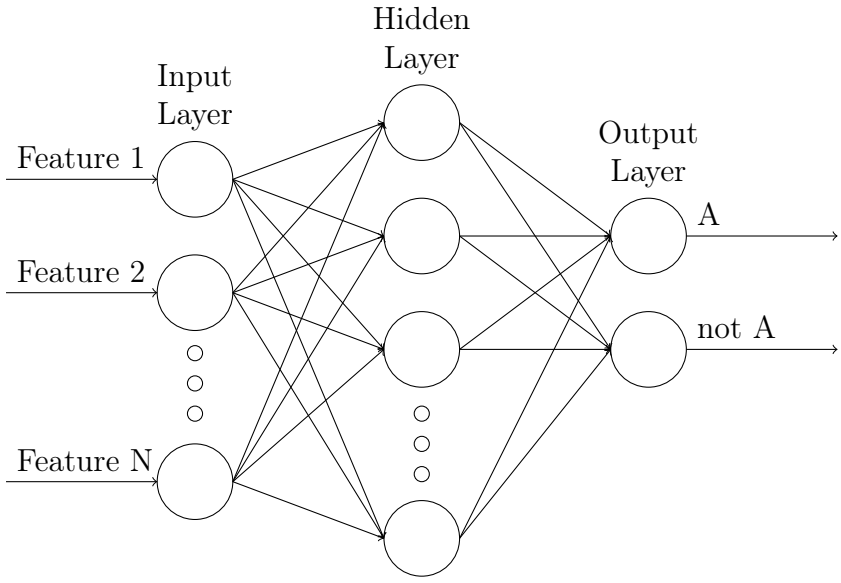

Figure 4.1: Representation of a FFNN with one hidden layer.

Each neuron does a weighted sum on the values outputted by the neurons on the previous layer, then passes the result through an activation function and finally transfers those values to the next layer [59]. Some of the most commonly used activation functions are the sigmoid, hyperbolic tangent, Rectified Linear Unit (ReLU), identity and softmax functions which are partly represented in figure 4.2. Equations 4.1, 4.2, 4.3 and 4.4 present the mathematical formulas for the sigmoid, hyperbolic tangent, ReLU and softmax functions respectively [57]. The sigmoid and hyperbolic tangent functions are frequently used as the activation function in the hidden layers neurons in networks with a low amount of hidden layers, otherwise, relu and its variants are preferred. This is due to the vanishing gradient problem. In regression,

the identity function is often utilized in the neurons on the output layer, while the softmax function is mostly used in classification problems since it outputs probability values for a sample belonging to a specific class [58].

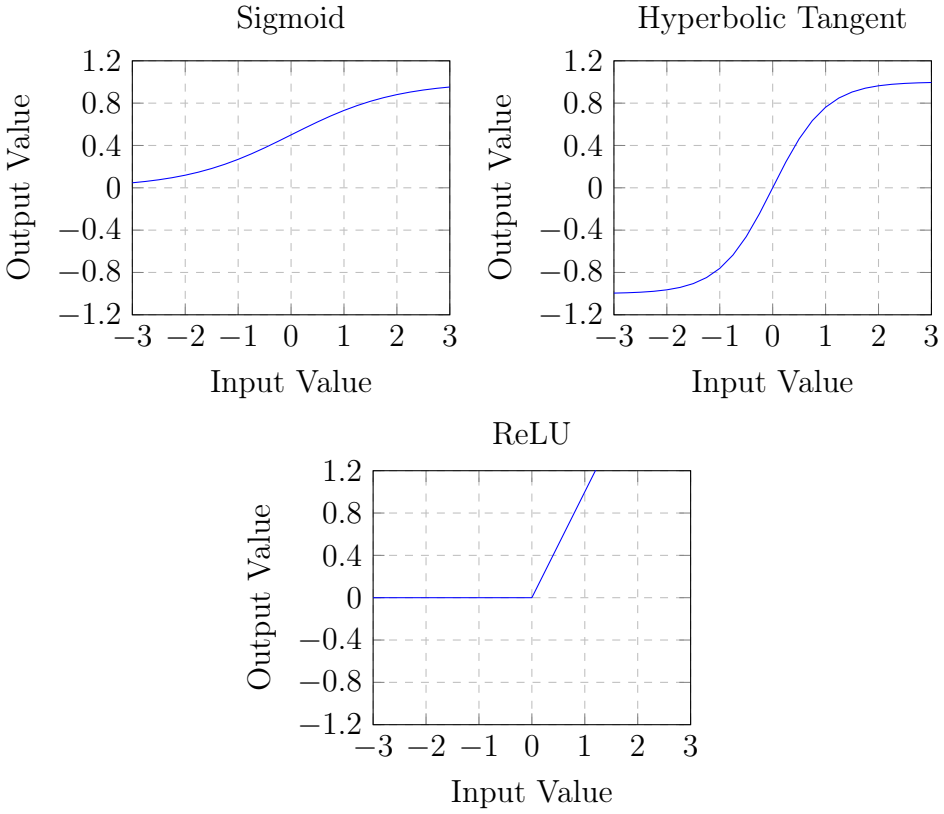

Figure 4.2: Sigmoid, Hyperbolic Tangent and ReLU Activation functions.

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{4.1}$$

$$\phi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.2}$$

$$\phi(x) = \begin{cases} 0, & if\ x < 0 \\ x, & if\ x >= 0 \end{cases} \tag{4.3}$$

´

$$\phi(x) = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{4.4}$$

Equation 4.5 shows the calculations made by each neuron where ϕ is the activation function, w_i is the weight for input i, b is the bias, x_i is the input i and n is the number of inputs [59].

$$y = \phi(b + \sum_{i=1}^{n} w_i x_i) \tag{4.5}$$

The weights and biases necessary for the weighted sums are changed during the training process in order to achieve the most accurate predictions on the output layer. However, neural networks can be trained using various algorithms. The one being used in this implementation is the backpropagation learning algorithm which uses a cost function also known as loss or performance, and tries to change the weights and biases so that the function is minimized [59]. A flow chart of the backpropagation algorithm is presents in figure 4.4. The cost function value represents the error exhibited by the model. A commonly used cost function is the MSE, its value is calculated as stated in equation 4.6, where t_i is the target value(label) for sample i, p_i is the prediction(output) for sample i and n is the total number of samples [57].

$$E = \frac{\sum_{i=1}^{n} (t_i - p_i)^2}{n} \tag{4.6}$$

Another possible cost function is the cross-entropy function which, for a single sample, can be calculated using equation 4.7 where t_c is the target value for class c, p_c is the prediction for sample class c and n is the total number of classes [58].

$$E = -\sum_{c=1}^{n} t_c.log(p_c) \tag{4.7}$$

The weights and biases are updated at each epoch taking into account gradient values which are backpropagated using the chain rule. Therefore, if the value of output o_i is directly dependent on the value of weight w_{ij}, then the gradient value can be obtained as stated in Equation 4.8. Multiple training algorithms use different formulas to update the weights, the simplest of which is the Gradient Descent (GD) algorithm.

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_i} \frac{\partial o_i}{\partial w_{ij}} \tag{4.8}$$

Neural networks with back propagation work as a supervised learning tool, meaning that they take into account the labels that are associated to each sample of input data for learning. Regression and classification problems can be both addressed using neural networks in tasks where ambiguity or complexity hinder the development of a precise algorithm using other conventional techniques [60]. Regression problems involve the prediction of continuous variables and classification involves the discrimination of samples according to a discrete set of classes [58]. Figure 4.3 shows a simple representation of a neural network being used in this work. W represents the weights and b represents the biases. The activation function on the hidden layer is the hyperbolic tangent and the softmax function in the output layer.

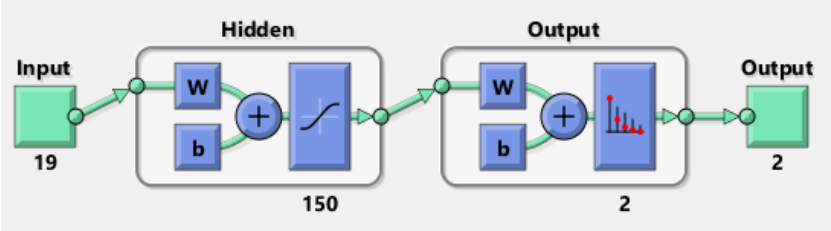

Figure 4.3: An example of a classification neural network with 19 inputs, two outputs and 150 hidden layer neurons. Made using MATLAB.

The training process of a neural network finishes when a certain stopping condition is met. This stopping condition could be a maximum number of training iterations or epochs, a minimum gradient value, a maximum training time or others.

4.1.2 Training Algorithms

Multiple training algorithms can be used to train the neural network. The ones evaluated in this work are described in the following text.

- **Gradient Descent Algorithms:** This algorithm updates the networks weights according to a value called the learning rate and to the partial derivative of the cost function in relation to each weight, also known as the gradient [61]. Equation 4.9 shows how weight w_{ij} is updated according to GD, where α is the learning rate and E is the cost function.

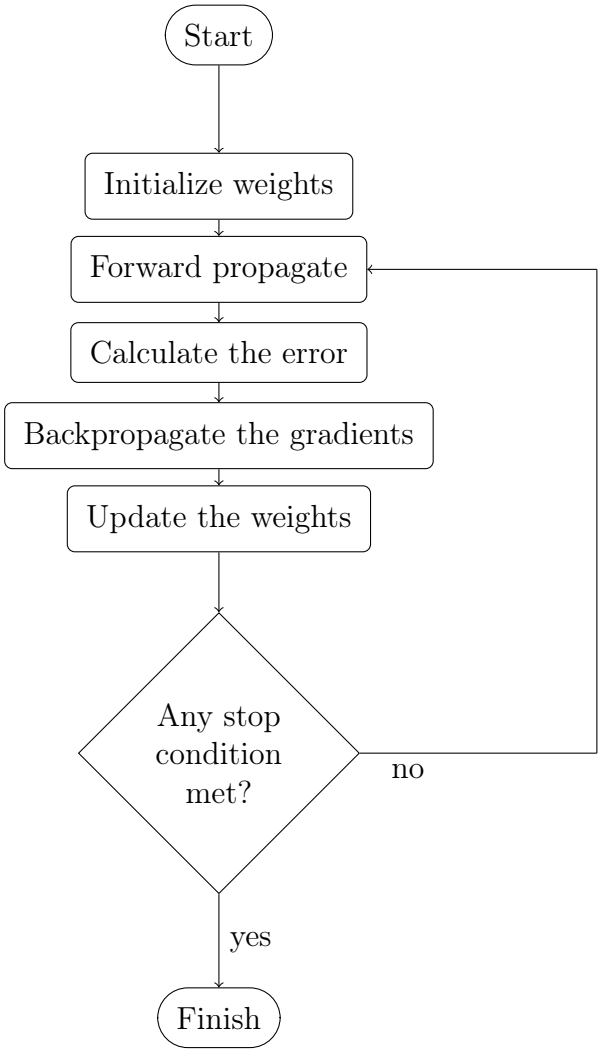

Figure 4.4: Flow chart of the backpropagation algorithm.

$$w_{ij}^{t+1} = w_{ij}^{t} - \alpha \frac{\partial E}{\partial w_{ij}}(t) \tag{4.9}$$

A variation of GD known as Gradient Descent with Momentum (GDM) adds a term related to momentum:

$$w_{ij}^{t+1} = w_{ij}^{t} - (\alpha \frac{\partial E}{\partial w_{ij}}(t) + m\Delta w_{ij}) \tag{4.10}$$

- **Resilient Backpropagation (Rprop):** This algorithm takes only into account the sign and its changes for each gradient value. If the gradient maintains its sign, the update-value(Δ_{ij}) increases by η^+, while if the gradient changes sign, Δ_{ij} decreases by η^-. Weight w_{ij} is updated by incrementing Δ_{ij} in the direction opposite to the gradient. Δ_{ij} is initialized at Δ_0 value and cannot be bigger than Δ_{max} or smaller than Δ_{min}. This algorithm is described by equations 4.11, 4.12 and 4.13. [62]

$$\Delta_{ij}^{t+1} = min(\eta^+ \Delta_{ij}^{t}, \Delta_{max}) \tag{4.11}$$

$$\Delta_{ij}^{t+1} = max(\eta^- \Delta_{ij}^{t}, \Delta_{min}) \tag{4.12}$$

$$w_{ij}^{t+1} = -sign(\frac{\partial E}{\partial w_{ij}}(t)).\Delta_{ij} \tag{4.13}$$

- **Quasi Newton Backpropagation:** This optimization algorithm is similar to Newton's method but is used when the matrix of first or second derivatives is not available or too expensive to calculate [63]. Newton's method is used to find the zeros or minima of a function using the first and second derivatives of that function. For a single variable function, the zeros and minima of the function can be found using an iterative process described by equations 4.14 and 4.15 respectively.

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)} \tag{4.14}$$

$$x_{n+1} = x_n - \frac{f'(x)}{f''(x)} \tag{4.15}$$

Quasi-newton methods replace the first or second derivative exact calculations with approximations that can be given by multiple update formulas.

- **Steepest Descent:** Similar to GD but instead of decrementing weights by $\alpha \frac{\partial E}{\partial w_{ij}}(t)$, the steepest descent algorithm tries to find the minima along the direction pointed by the gradient. This minima is settled as the new approximation for the solution and the process is iteratively repeated until the solution has been found [64].

- **Conjugate Gradient:** The conjugate gradient method is used to find the local minima of a function. It updates the network weights according to conjugate vectors. Vectors v and u are conjugate if:

$$v^T A u = 0 \tag{4.16}$$

  A is the matrix of second derivatives for the cost function. The method assigns the initial vector as the negative gradient, then it calculates an α value which corresponds to the step size in the vector direction. Weights are then updated and a new vector that is conjugate to this initial vector needs to be found by using β. The process is repeated by finding new conjugate vectors at each iteration by recalculating the α and the β values until the solution has been found [65].

All the described algorithms have their advantages and disadvantages. Depending on the problem at hand, one might be preferred over another.

4.1.3 MATLAB Implementation

The network was implemented using MATLAB's *patternet* function. This function receives the network structure as an argument. For example, if a network with two hidden layers of size 10 each was required, the following line of code would be used:

```
net = patternnet([10 10]);
```

This will initialize a neural network to be used for classification problems. Therefore, the activation function on the output layer is the softmax function. The network training algorithm, cost function, and activation functions can be then selected using:

```
net.trainFcn='trainrp';%training algorithm
net.performFcn='crossentropy';%cost function
net.layers{1}.transferFcn = 'tansig';%activation
net.layers{2}.transferFcn = 'tansig';%activation
```

Various training algorithms are already implemented on MATLAB. Table 4.1 presents the main implemented algorithms.

Table 4.1: Table to test captions and labels

	Description
trainlm	Levenberg-Marquardt
trainbfg	BFGS Quasi-Newton
trainrp	Resilient Backpropagation
trainscg	Scaled Conjugate Gradient
traincgb	Conjugate Gradient with Powell/Beale Restarts
traincgf	Fletcher-Powell Conjugate Gradient
traincgp	Polak-Ribiére Conjugate Gradient
trainoss	One Step Secant
traingdx	Gradient descent with momentum and adaptive learning rate

Afterwards, the training parameters of the network can be defined in the *trainParam* variable. So, the minimum gradient value, the optimization goal, the maximum number of training epochs, the maximum number of failed validation checks, and the maximum training time can be defined as follows:

```
net.trainParam.min_grad= 1.0000e-07;
net.trainParam.goal = 0;
net.trainParam.epochs = 5000;
net.trainParam.max_fail = 10;
net.trainParam.time = 8640;
```

After the network has been fully configured it can be trained using the *train* function. This function takes in the configured network, the network inputs, and the network targets as arguments. It returns the trained network and a variable called the training record(tr):

```
[net, tr] = train(net,inputs,targets);
```

To train the model using the Graphics Processing Unit (GPU), the inputs and targets must be first stored in the GPU's memory and the train function must be then run with the *useGPU* option set to yes:

```
inputsg = gpuArray(inputs);
targetsg = gpuArray(targets);
[net, tr] = train(net,inputsg,targetsg,'useGPU','yes');
```

The *inputs* matrix rows correspond to the features while the *targets* matrix rows correspond to the classes. The columns of both variables correspond to the samples and both matrices must have an equal amount of samples. The training record contains information about the training process such as the performance value at each epoch, the amount of time passed between each epoch and other related data.

The network predicted outputs can be obtained using the *sim* function:

```
outputs = sim(net,inputs);
```

These outputs can be then used to calculate the network performance and plot the confusion matrix and the Receiver Operating Curve (ROC) chart. The training record variable can be used to draw the performance plot and the train state plot:

```
performance = perform(net,targets,outputs);
plotconfusion(targets,outputs);
plotroc(targets,outputs);
plotperform(tr);
plottrainstate(tr);
```

The sensitivity, specificity, and accuracy of the model on the test set can be obtained using the confusion matrix. First, the number of True Positives (TPs), True Negatives (TNs), False Positives (FPs) and False Negatives

(FNs) is calculated for each class of samples and then each of the aforementioned parameters are determined by applying their respective formulas:

```
%obtain the confusion matrix
[~,cm,~,~] = confusion(targets(:,tr.testInd),
outputs(:,tr.testInd));

cm = cm';

%obtain the TPs for each class
for k = 1:1:size(cm,1)
    tp(k) = cm(k,k);
end
%obtain the TNs for each class
for k = 1:1:size(cm,1)
    aux = cm;
    aux(:,k) = [];
    aux(k,:) = [];
    tn(k) = sum(sum(aux));
end
%obtain the FPs for each class
for k = 1:1:size(cm,1)
    aux = cm(k,:);
    aux(k) = [];
    fp(k) = sum(aux);
end
%obtain the FNs for each class
for k = 1:1:size(cm,1)
    aux = cm(:,k);
    aux(k) = [];
    fn(k) = sum(aux);
end

%compute the measures
sensitivity = tp./(tp+fn);
specificity = tn./(tn+fp);
accuracy = (tp+tn)./(tp+tn+fp+fn);
```

The AUC of the model for the test set can be computed from the ROC chart by means of integration:

```
[tpr,fpr,~] = roc(targets(:,tr.testInd),
outputs(:,tr.testInd));

auc = trapz(fpr{1,2},tpr{1,2});
```

Hardware

The model training could be run in the Central Processing Unit (CPU) or the GPU. In general, the GPU is the best option for training a neural network as they are faster at making the calculations necessary for the process. The disadvantage of using a GPU is that the amount of Video Random Access Memory (VRAM) available to store the model data is often smaller than the Random Access Memory (RAM) available to the CPU. This means that models can be more complex on the CPU without it running out of RAM. Because of this, and due to the fact that one of the available computers does not have a dedicated GPU, the CPU was responsible for training the networks in this project.

4.1.4 Weight Initialization

The initialization of weights and biases for the neural network can influence the final results after training, so it can have negative consequences if done wrongly [66]. The first and most simple problem is that if weights are initialized at the same values, all neurons in the network would end up learning the same tasks and this is not desirable. Also depending on the activation function, if weights are set to very high values, it could cause the neuron to die altogether because of vanishing gradients. Vanishing gradients happen when the activation function of a specific neuron reaches a point in which the gradient is so small that progress becomes halted [67].

One method proposes that the weights and biases are initialized by generating random values between $\frac{-2.4}{N}$ and $\frac{2.4}{N}$ in an uniform distribution, where N is the number of neurons of the previous layer [68]. Xavier Initializa-

tion, first proposed by Glorot et al. [69], suggests that weights are initialized by taking random numbers from an uniform distribution with the following limits:

$$U[-\sqrt{\frac{1}{N}}, \sqrt{\frac{1}{N}}] \tag{4.17}$$

A version of the same procedure using a normally distributed numbers can be also achieved by setting the mean value of the distribution to zero and the standard deviation to $\sqrt{\frac{1}{N}}$:

$$N[0, \sqrt{\frac{1}{N}}] \tag{4.18}$$

He Initialization, proposed by He et al. [70], presents a very similar approach but the standard deviation of the normal distribution is changed to:

$$\sigma = \sqrt{\frac{2}{N}} \tag{4.19}$$

Using normal distributions to generate weights can also be generalized for any activation function ϕ differentiable at zero using equation 4.20 [71].

$$N[0, \sqrt{\frac{1}{N(\phi'(0))^2(1+\phi(0)^2)}}] \tag{4.20}$$

In this work, various methods were used to initialize the weights of the neural network. In the hidden layers, the formula presented on equation 4.20 was used to compute the weights, while in the output layer Xavier initialization was used. This is because the neurons in the hidden layers are going to contain either a hyperbolic tangent or a sigmoid activation function, therefore, the method allows for versatility when switching between these functions in the processes such as the tuning of the model. As for the output layer, Xavier initialization was chosen due to it performing better than others methods on tests made on the implemented system.

4.1.5 Training Process

Data Division Techniques

The presented data had to be divided into training, test, and validation sets before running the training process. The training set is used to train the model, the test set is used to measure how well the model behaves on data that has never seen before, and the validation set is used for tuning the learning parameters and early stopping.

The usual approach is to divide the data such that 70% of the samples are assigned for training, 15% for test, and 15% for validation. All patient data could be joined and then samples could be randomly picked to obtain sets that fit these percentages, but mixing all patient data would inhibit the ability to check on whether or not the model can generalize for patients it has not seen before, since different patients show different characteristics in the EEG signals. So, the alternative solution would be to use k-fold cross validation, where initially the data set is divided into k sets named folds and then each one of the folds is used to test the model with the remaining folds being used for training and validation purposes [72]. For example, if 5-fold cross-validation was employed, the data set would be divided into five folds, then each fold would be used as the test set with the rest assigned for training while also picking a few separate patients for validation. Since there are 19 patients in total, four of the folds would have four patients each and one would have three patients, this is so that patient samples are kept in separate folds.

As a specific case, LOO Cross Validation occurs when the number of folds equals the number of patients, this is useful to check how much the results are differing between patients and to verify which patients are harder to classify. As the number of folds is increased, the size of the training set increases as well and the higher the amount of tests required. So, usually, LOO is much slower to run than 2-Fold Cross Validation.

After all the separate tests are run, the results can be summed up by calculating the average. For visualization purposes, the data division process using k-fold cross validation is represented in figure 4.5.

Data Division Implementation

To define the training, validation, and test sets on MATLAB the variable *divideParam* is used. If the training set contains samples 1 to 700, the vali-

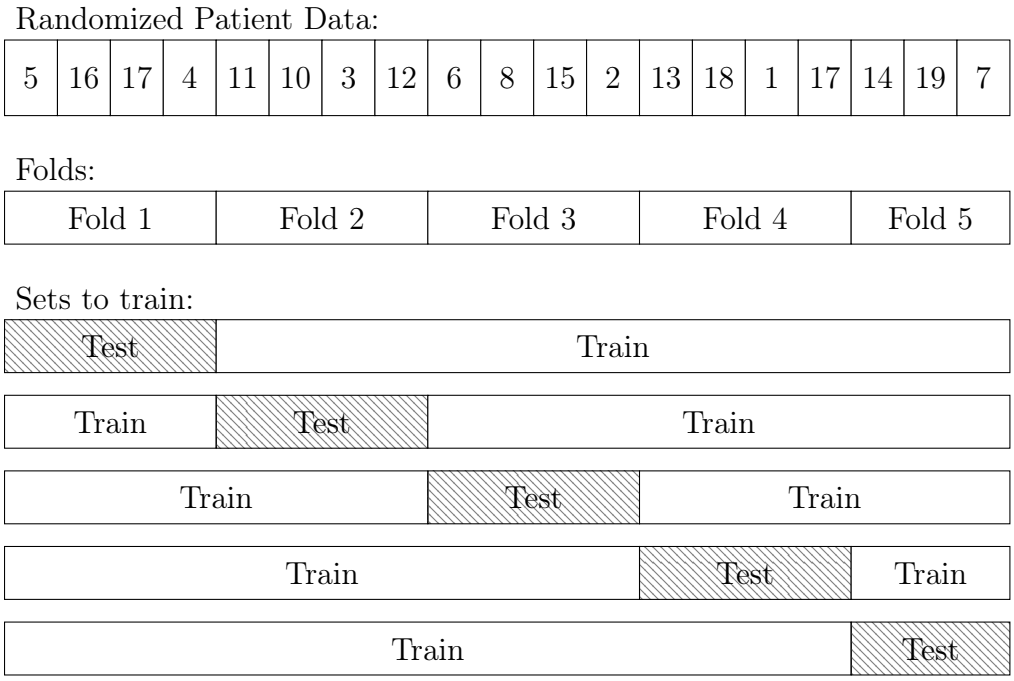

Figure 4.5: Diagram explaining how the data set is divided using 5-fold cross validation and used to train the model to retrieve results.

dation set contains samples 701 to 850 and the test set contains samples 851 to 1000, then the sample indexes for each set should be provided as follows:

```
net.divideFcn='divideind';
net.divideParam.trainInd=1:700;
net.divideParam.valInd=701:850;
net.divideParam.testInd=851:1000;
```

For the final model, each configuration is tested multiple times and the results are summarized by applying mean, standard deviation, minimum, and maximum operations over all the scores that are gathered. The mean score will provide an understanding of how well the model is behaving, while the standard deviation will demonstrate the amount of variation in the results. The maximum and minimum score are used to collect the best and worst models. Confusion matrix, ROC chart, train state plot, and performance plot were saved for the best and worst performing neural network along with other relevant data.

4.2 Tuning the Model

There are multiple parameters that need to be tuned or selected in order to obtain the best classification possible. This is generally done by running multiple instances of the same model with different values for each variable that needs tuning or by simply applying problem knowledge. These parameters could include:

- Training algorithm and its parameters.
- Number of neurons in each hidden layer.
- Number of hidden layers.
- Cost function.
- Activation functions.
- Regularization parameters.

- Maximum failed validation checks number.
- Minimum gradient value.
- Maximum number of training epochs.

4.2.1 Weak Models

If precautions are not taken during the training phase, the ML model can overfit or underfit to the training data. Overfitting happens when the model is too adapted to the training set and thus cannot generalize to classify new data. This occurs as a result of the model fitting the noise and errors in the training data. It also happens as a consequence of the model being more complex than necessary for the task at hand [73]. Underfitting, in contrast, happens when the model is not capable of fitting the data to an acceptable degree [74]. The compromise between overfitting and underfitting is often called as the bias-variance trade-off. A few methodologies can be used to prevent overfitting:

- **Early Stopping:** Technique that forces the model to stop training when a certain number of validation checks without improvements occur. Hence, the validation set is evaluated using the loss function at each epoch to check if the error increased or decreased, if the error does not decrease for a certain amount of sequential epochs the training process is stopped, thus avoiding overfitting [75]. This also reduces the amount of time necessary for training. The technique is regulated by the maximum failed validation checks number.
- **Regularization:** Method that changes the learning process to include a metric directly related to the magnitude of the network weights. This helps keeping the weights at low values which avoids model complexity and overfitting [76]. There are two main types of regularization techniques, L1 and L2 regularization. With regularization in place, the loss function is now calculated using equations 4.21 or 4.22 for L1 or L2 regularization respectively, where E is the original loss value, λ is the regularization rate and w_i is weight i.

$$loss_{L1} = E + \lambda \sum_{i=1}^{n} |w_i| \tag{4.21}$$

$$loss_{L2} = E + \lambda \sum_{i=1}^{n} w_i^2 \tag{4.22}$$

MATLAB uses an altered version of L2 regularization in which the loss function is calculated as stated in equation 4.23, where γ is the performance ratio [77].

$$loss = (1 - \gamma)E + \gamma \sum_{i=1}^{n} w_i^2 \tag{4.23}$$

The performance ratio can be set in MATLAB using:

```
net.performParam.regularization = 0.01;
```

Both early stopping and regularization were tested on the model. For the initial feature selection and other procedures, the performance ratio was initially set to zero, meaning that none of the cost function value is influenced by the sum of neural network weights.

In order to prevent underfitting the selected features need to be relevant to the problem, that is, they need to have predictive power. Also, the neural network needs to be capable enough to learn how to classify samples. Therefore, it needs to have a certain minimum amount of neurons in the hidden layers. So, both feature selection and model tuning can help preventing underfitting.

4.2.2 Model Tuning Planing

Before starting the tuning process, the cost function, the maximum failed validation checks number, the minimum gradient value, and the maximum number of training epochs were set to fixed values to reduce the number of tuning variables and thus cut the amount of parameter combinations which would in turn lower the amount of time necessary for tuning the model. The

cost function was set to cross-entropy since it is the most recommended for classification problems, the maximum failed validation checks number was set to 10, the minimum gradient value was set to its default value on MATLAB, and the maximum number of training epochs was set to 5000. Therefore, the rest of the model parameters need to be tuned and for that a set of possible values needs to be first settled for each parameter:

- Training algorithm:
  - Resilient backpropagation
  - Scaled conjugate gradient backpropagation
  - BFGS quasi-Newton backpropagation
  - Conjugate gradient backpropagation with Powell-Beale restarts
  - Conjugate gradient backpropagation with Fletcher-Reeves updates
  - Conjugate gradient backpropagation with Polak-Ribiére updates
  - Gradient descent with momentum and adaptive learning rate backpropagation
  - One-step secant backpropagation
- Number of neurons in the hidden layer: 10, 30, 50, 70, 90, 110, 130, 150, 170, 190, 210, 230, 250, 270, 290 or 310.
- Number of hidden layers: 1 or 2.
- Activation function:
  - Hyperbolic tangent
  - Sigmoid
- Performance ratio: 0, 0.01, 0.04 or 0.07.

The included training algorithms are the most common algorithms implemented on MATLAB. Exactly eight algorithms were selected to fully occupy three bits of data. The Levenberg Marquardt (LM) algorithm was not examined as it is a second order algorithm which requires the calculation of the Hessian matrix. This process is significantly computationally demanding

and time consuming, making the use of this algorithm not suitable for this work where a large number of simulations were performed. The number of neurons were selected based on steps of 20 from 10 to 310, this will result in 16 possible values which takes precisely four bits of data to represent. In terms of layers, the network can either have one or two layers since these are the most common amounts for shallow configurations. The same rationale was applied to the possible activation functions. The performance ratio was maintained at low values.

4.2.3 Model Tuning Methods

With these options there are a total of 2048 possible configurations, the tuning process would involve trying to find the combination that provides the best results. Two of most common approaches are grid search and random search. Grid search involves testing every single combination and selecting the one that obtained the highest score, while random search randomly picks a subset of combinations and finds the best solution within that subset. Grid search returns the optimal solution but at the cost of time, in contrast, random search provides a good solution in a lower amount of time but may not reach the optimal solution. Testing the 2048 possible combinations is not feasible as it would take too much time, while random requires a proper Monte Carlo sampling to converge to a good solution, which, again, will require a large amount of simulations. Therefore, a third possibility is to use an heuristic search, such as GA, to tune the learning parameters.

4.2.4 Model Tuning using the Genetic Algorithm

The GA takes its inspiration from the way species evolve to become more adapted to their environment through natural selection, sexual recombination, and mutation [78]. This algorithm is used to solve optimization problems where the minimization or maximization of a certain parameter is necessary to find the best solution [79]. Each solution or individual is identified by their unique genetic code which contains an encoded string of parameters related to the individual. The entire algorithm can be summarized in a set of steps:

- **Step 1:** Generate a random population by creating a set of individuals with different genetic codes. The genetic code (or chromosome) is

going to contain information associated with the values of each tuning parameter.

- **Step 2:** Evaluate each individual using a fitness function. This is done by running each individual neural network with the tuning parameters associated with it and collecting the resultant AUC value which is used as the fitness score.

- **Step 3:** Select individuals to use in the crossover step. Individuals with a better fitness score should have a higher chance of being picked.

- **Step 4:** Combine the selected individuals to create a new generation. Two parent chromosomes are mixed to generate a new chromosome, this is known as Crossover.

- **Step 5:** Apply random mutations on the generated individuals. This would involve random changes to the genetic code.

- **Step 6:** Keep a few of the best individuals from the previous generation on the next generation. This is known as Elitism.

- **Step 7:** Repeat steps 2 to 6 with each consecutive new generation until a certain stopping condition is met.

The information containing which tuning parameters to use for each individual was encoded in a string of 11 bits called the chromosome. The number of bits necessary to encode each parameter can be calculated by applying the logarithm base two to the number of possible values for each parameter. Four bits were used to represent the number of hidden layer neurons, three bits for the training algorithm, two bits for the regularization parameter, one bit for the activation function in the hidden layers, and one bit for the number of hidden layers.

Selection of individuals can be done using tournament selection where a certain number of individuals is randomly selected from the population and then the best individual is used as a parent for crossover. This selection method is analogous to the way individuals compete for mating in nature [78].

Having the selected parents, crossover can be done using two-point splitting where two random points on the string of bits are selected to split the genetic code in sections and then sections from the parents are mixed as

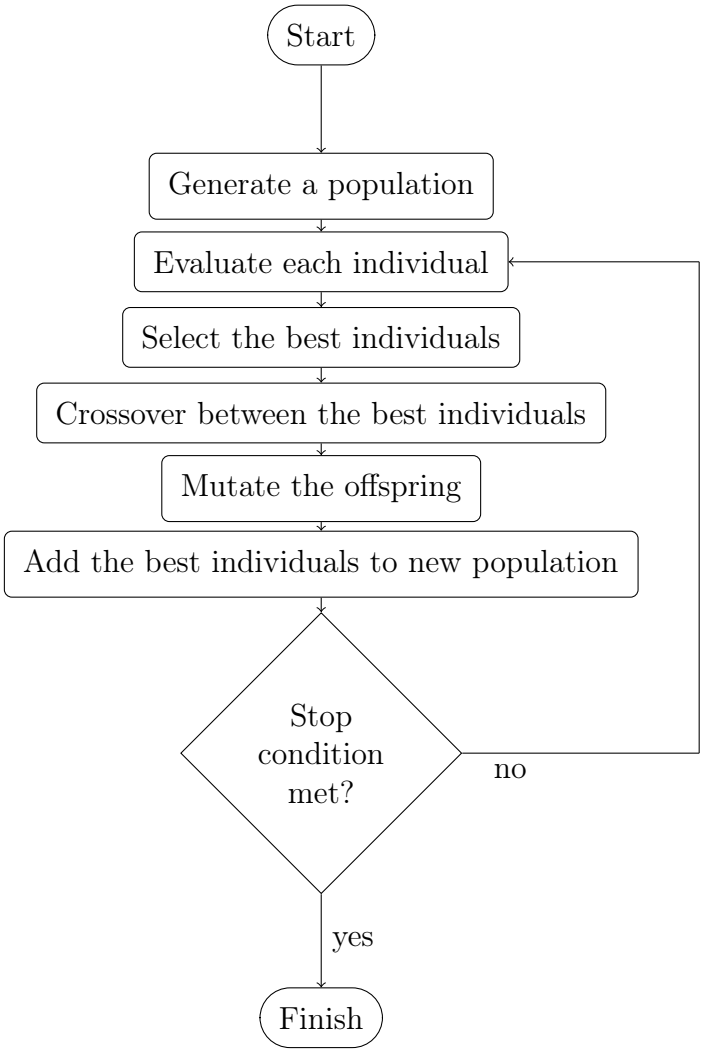

Figure 4.6: Flow chart of the genetic algorithm.

demonstrated in figure 4.7. Any amount of points could be used in the splitting of the chromosome and having more points results in a higher amount of possible combinations. Nevertheless, two splitting points provides with a good amount of variability. Crossover serves the propose of generating new offspring to be tested and carrying the genetic information from the best fitting parents to the next generation [78]. Also, to increase the amount of exploration done by the algorithm a certain percentage of the children are simply obtained by cloning one of the parents. These children are known as mutation children since they only differ from their parent because of mutations.

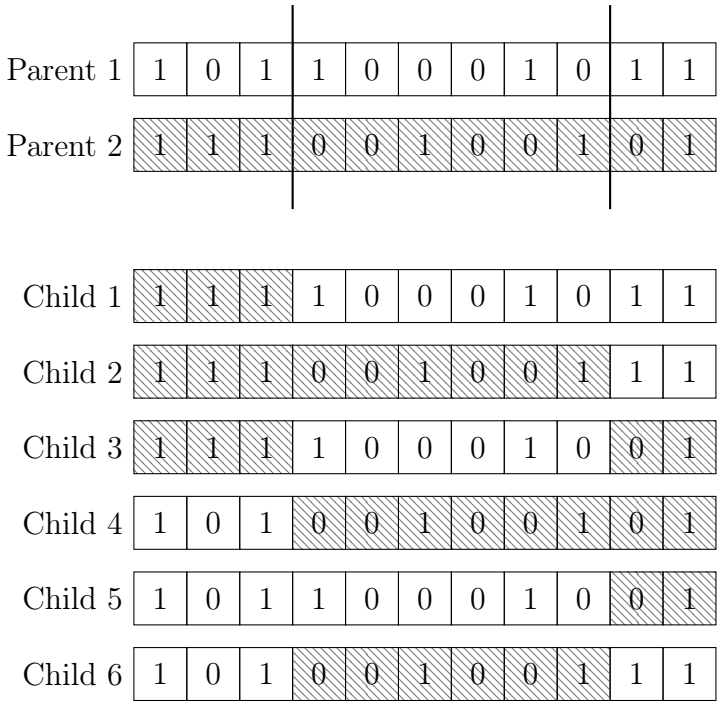

Figure 4.7: How two-point crossover can be done in the 11 bit long chromosome. Each couple of chromosomes can create up to six new combinations.

After this step, mutations can be applied by randomly selecting bits from the child chromosome using a mutation probability value and then changing those bits.

The first obtained score value for a given chromosome was maintained throughout the evolution. This means that if chromosome appears in multi-

ple generations, its score value will not be recalculated and instead its first calculated value is used. This reduces the amount of time necessary for tuning.

During the assessment of the algorithm, to test the amount of diversity in each population, a diversity score can be calculated as stated in equation 4.24, where p_{size} is the population size, c_{bits} is the number of bits per chromosome and $hdist(c_i, c_j)$ is the hamming distance between chromosome i and chromosome j. Hamming distance is the number of bits that differ between two strings of bits.

$$\frac{2}{c_{bits}(p_{size}^2 - p_{size})} \sum_{i=1}^{p_{size}} \sum_{j=i+1}^{p_{size}} hdist(c_i, c_j) \tag{4.24}$$

Wang [80] used an implementation of the genetic algorithm to calibrate a rainfall- runoff model and managed to achieve a good calibration with ten runs of 10000 fitness function evaluations each.

In this endeavor, a maximum number of generations of 50 was established as the stopping condition. The number of individuals in each generation was set to 20, the number of elites was set to two, the percentage of mutation children was set to 10% and the mutation probability was set to 5%. The selected fitness value was 1-AUC since the implemented algorithm does minimization instead of maximization.

4.3 Feature Selection

Dash et al. [81] defined feature selection as the process in which the features are picked according the their relevance to the target concept. Smaller groups of features are generated and evaluated until a certain stopping condition is reached. So, in order to improve the model performance and results, a subset of the initial features containing the most relevant features in this context has to be selected. This is due to models performing better with a lower amount features and also because some features that might be less relevant to the problem could cause the model to behave poorly [82]. Also, feature selection results in a simpler model that runs in a lower amount of time, with a better accuracy, and with a reduction in the amount of storage required [83]. Feature selection also provides a more comprehensible model that is easier to analyse and explain. There are multiple methods to implement feature selection,

each of them with their own conveniences. Brute forcing all combinations of features is not a feasible method since the number of possible combinations increases exponentially with the increase in the number of initial features. Taking into account the available 98 features, a brute-force method would test $2^{98} - 1 \approx 3.17 \times 10^{29}$ combinations, therefore alternative methods need to be used.

4.3.1 Selection Methods

SFS is one of the possible algorithms to do feature selection, it works by creating two sets of descriptors, the first initially containing all the possible descriptors and the second is empty. Then, all the features present on the first set are individually tested and the one that provided the best score is added to the second set and removed from the first set. The process is then repeated but each feature of the first set is tested together with the features present on the second set until the best set of features has been found. The algorithm finishes when all the features have been transitioned to the second set. The second set is going to contain all the features ordered from most relevant to least relevant. To finish the selection process, the final set of selected features is the set containing the first n features from the second set that provided the best score value.

Sequential Backward Selection (SBS) works in similar way but instead of sequentially adding features to a set of best features, all the features are tested together and sequentially transitioned to a set of worst features. Sequential search methods are easy to implement and provide a feature set that is suitable for the implemented model. The number of combinations tested by the sequential feature selection methods increases with the square of the number of features. The number of combinations in relation to the number of features n can be calculated as follows:

$$\sum_{i=1}^{n} i = \frac{n^2 + n}{2} \tag{4.25}$$

So, for the 98 available features, the model would be tested 4851 times. The drawback of SFS and SBS is that the model is trained at each iteration to test each set of combined features, thus it can be slow.

There is also the option of using the Minimum Redundancy Maximum Relevance (mRMR) selection algorithm. This algorithm scores each feature

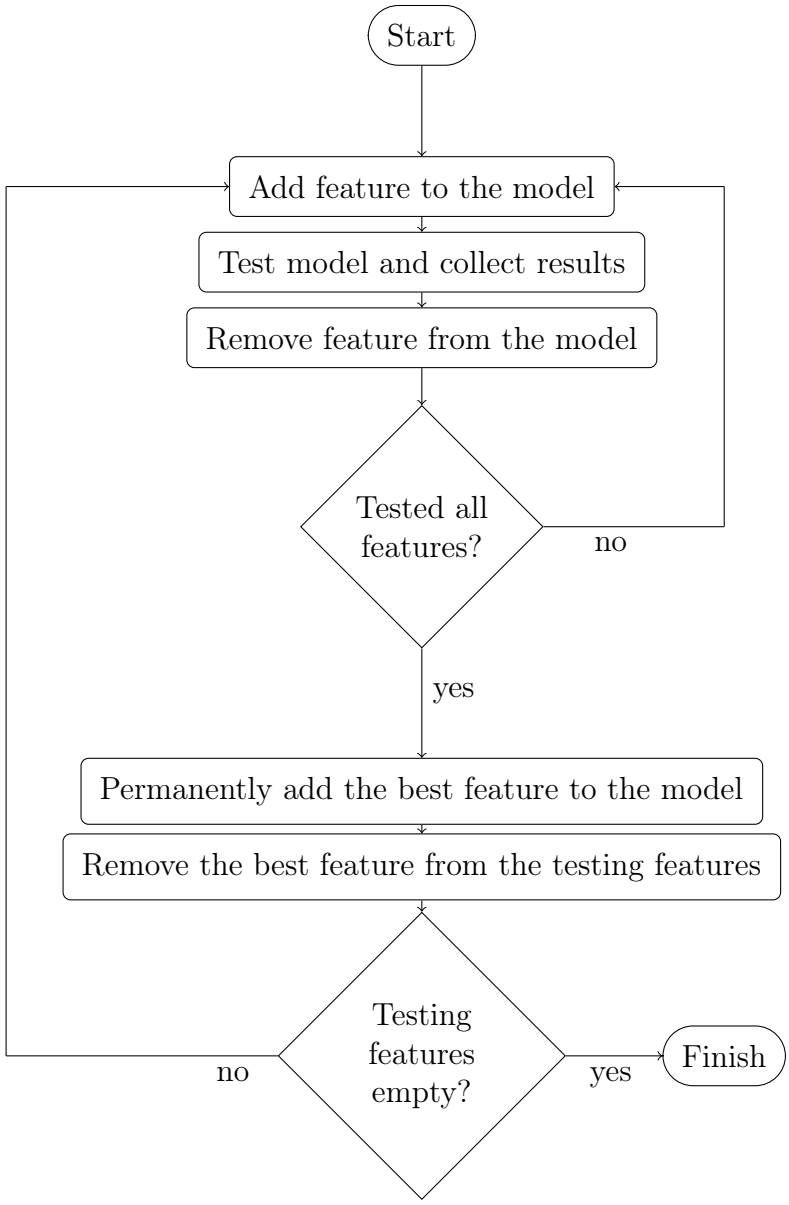

Figure 4.8: Flow chart of the SFS algorithm.

according to their redundancy and relevance to the problem, so, features that have high relevance and low redundancy earn a higher score [84]. This method does not directly use the learning model to estimate these parameters and is based on the mutual information between features. According to mRMR, a set of best features S needs to be found such that its relevance V_S is maximized and its redundancy W_S is minimized.

$$V_S = \frac{1}{|S|} \sum_{x \in S} I(x, y) \tag{4.26}$$

$$W_S = \frac{1}{|S|^2} \sum_{x,z \in S} I(x, z) \tag{4.27}$$

These parameters are calculated using the mutation information (I) between two sets of random variables. So the redundancy of a set of features is the average amount of mutual information between features, while the relevance is the average amount of mutual information between each feature and the target variable. The mutual information between A and B can be calculated as follows:

$$I(x, z) = \sum_{i,j} P(A = a_i, B = b_j) log \frac{P(A = a_i, B = b_j)}{P(A = a_i)P(B = b_i)} \tag{4.28}$$

To attain the ordered set of features use:

```
[idx, scores] = fscmrmr(inputs, targets);
```

Then, the features can be sequentially added (best scoring first) in the model to find which set of features provides the best results. mRMR is relatively fast at calculating the scores.

If we consider the 98 available features, the mRMR approach would test the model 98 times while SBS and SFS would test the model 4851 times.

As a third alternative, PCA could also be applied to the EEG to generate new features which would correspond to the resultant principal components. PCA allows for a change of basis on the data to find new axes that are more relevant in terms of variance. These axes are called principal components

and can be used as features. So, 98 principal components would be generated by the PCA algorithm from the original 98 features. Only a subset of those components needs to be selected, corresponding to the ones that have a higher influence in the spread present in the data distribution. These are the components that generate the highest variation in the data. These components are calculated by applying a linear combination on the original data variables [85]. PCA does not take into account the labels for classification as it is purely based on the distribution of the input data. A visual representation of the principal components is presented by figure 4.9.

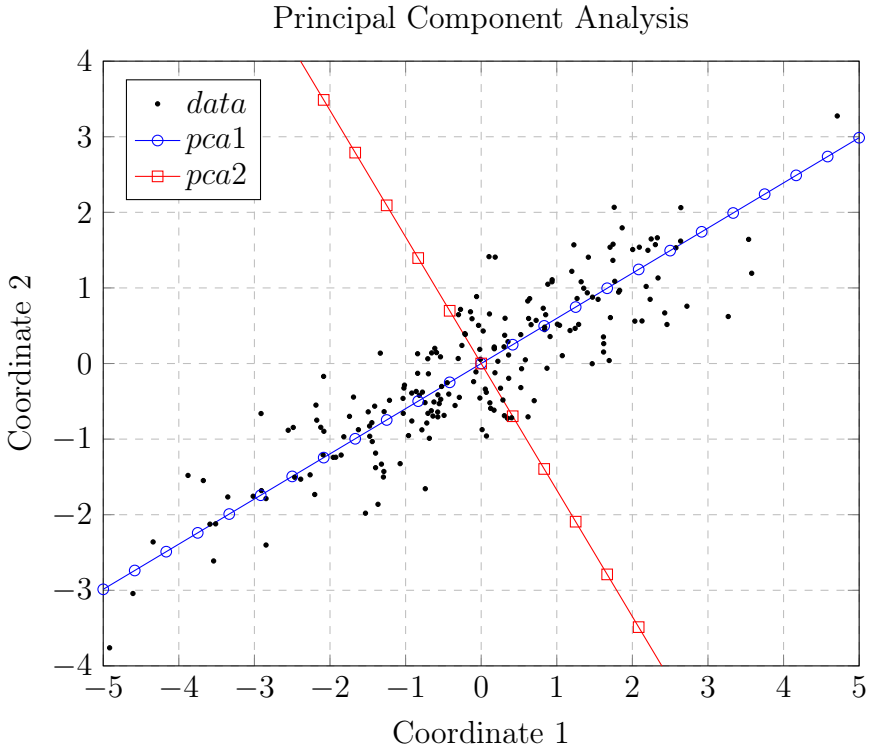

Figure 4.9: An example of PCA on two dimensions. The data coordinates are remapped to principal components one and two. Component one captures the highest amount of variation on the original data.

4.3.2 Solution

In this work, feature selection was done in three separate occasions. First, mRMR was used to obtain a set of features to be used on the model tuning process. Then, after the model was tuned, feature selection was executed to find out a best possible set of features, this time using SFS. Finally, PCA was applied to the previously selected features to further reduce the dimensionality.

4.4 System Summary

A summary of the steps needed for the implementation of the system is displayed in the diagram presented in Figure 4.10. The whole architecture can be described as the following set of procedures required for the conception of the entire system:

- **Feature Extraction and Calculation:** numerical attributes need to be extracted from the raw data set in order to be used as predictors by the classifier.
- **Feature and Data Scaling:** to improve classification, features and some of the EEG data needs to be scaled as they could exhibit different data distributions.
- **Data Set Balancing:** the number of samples needs to be balanced between different classes of samples.
- **A-phase Classification Algorithm:** ML algorithm that learns to classify A-phase samples using the extracted features.
- **Training Process:** data needs to be divided into training, test and validation sets before training.
- **Tuning the Model:** to improve results, the learning parameters associated with the algorithm need to be tuned.
- **Feature Selection:** to improve results, a smaller set of features needs to be selected.

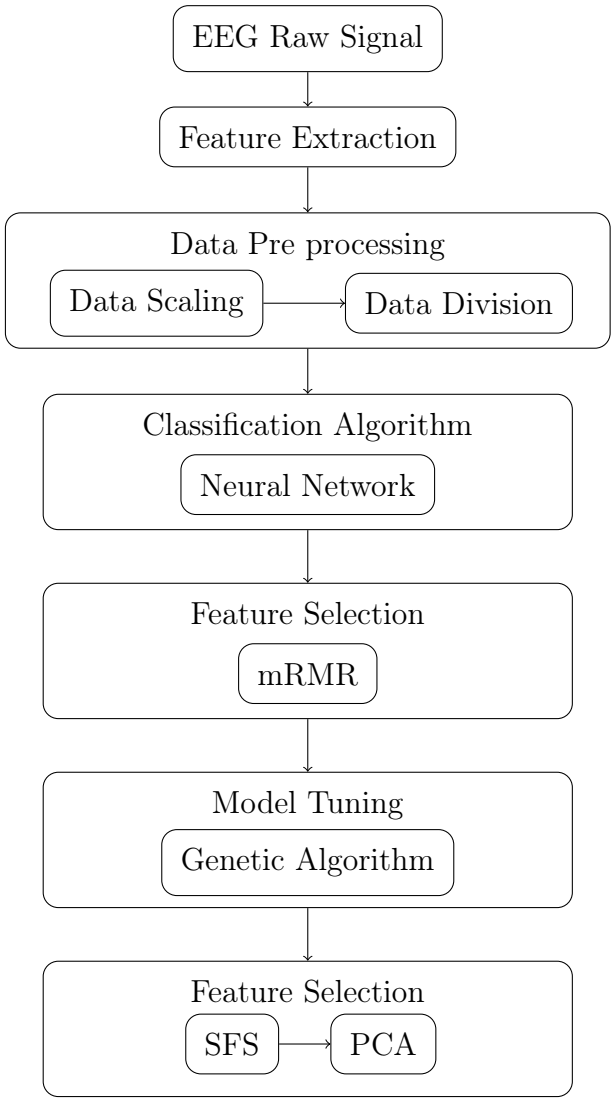

Figure 4.10: Diagram containing all the proposed steps required for the conception of the system.

4.5 Software and Hardware Resources

This work was developed under the MATLAB environment as it provides multiple tools that facilitate the implementation of neural networks and other procedures necessary for the detection system. Also, MATLAB provides simple to use materials that are well tested. Most of the procedures required for the conception of this system were implemented by directly using MATLAB functions.

In terms of hardware, the project was implemented and tested on two machines with the following specifications:

- CPU: Intel i7-9750HF, 2.6 GHz, 6 cores.
- GPU: NVIDIA Geforce GTX 1650, 4 GB V-RAM.
- RAM: 16 GB.

- CPU: Intel i5-4460, 3.2 GHz, 4 cores.
- GPU: No Graphics Card.
- RAM: 12 GB.

The first machine is a portable computer which was used to implement the system on MATLAB and to run short tests, while the second machine is a desktop used for the longer tests.

4.6 Evaluation Methodology

4.6.1 Metrics for Evaluation

To improve the proposed implementation, metrics for the evaluation of the proposed method need to be administered. These metrics help to decide which changes to make on the system and to also give a final rating to it with the perspective of choosing the best option. The correctness and performance of the whole system is directly dependent on the classification of the A-phase samples. Therefore, the proposed system can be evaluated using diverse metrics that are already used to score ML algorithms themselves.

Correctness Metrics

In terms of correctness, the model was evaluated using accuracy, sensitivity and specificity values which can be calculated using the following formulas:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.29}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.30}$$

$$Specificity = \frac{TN}{TN + FP} \tag{4.31}$$

TPs are positive samples that were correctly classified as positive, TNs are negative samples that were correctly classified as negative, FPs are negative samples that were incorrectly classified as positive and FNs are positive samples that were incorrectly classified as negatives samples [86]. In this work, A phase samples are positive samples and not-A phase samples are negative samples. Therefore, this work is going to focus on the binary classification between A phases and non-A phases.

Accuracy provides an overall idea of how well the model behaves on all tested samples, in short, it is the percentage of correctly classified samples(TPs and TNs). Sensitivity, also known as True Positive Rate (TPR), indicates how well the classifier behaves on positive data, meaning that a high sensitivity corresponds to less positive samples being classified as negative samples. Specificity, on the other hand, shows how the classifier performs on negative data, so a classifier with high specificity is less prone to classifying negative samples as positives ones [87]. Specificity is also known as the True Negative Rate (TNR).

Ideally, a good classifier would exhibit high values in terms of sensitivity, specificity, and accuracy together. For example, in binary classification, a classifier that shows a very high sensitivity (close to 100%) but a very low specificity (close to 0%) means that all samples are mostly being classified as positive samples, even if they are negative, which renders the classifier useless. Also, using the accuracy value alone to score the ML model can be misleading since the data may exhibit some imbalance between positive

and negative samples. So, for instance, if there are 90 negative samples and 10 positives samples, classifying all samples as negative would lead to an accuracy of 90%, a specificity of 100%, and a sensitivity of 0%.

As a fourth possible measure, AUC is calculated as the area below the ROC plot and could be also used to measure the model correctness. The ROC plot plots the TPR (same as sensitivity) for each False Positive Rate (FPR) for different classification thresholds [88]. Equation 4.32 shows how to calculate the FPR.

$$FPR = \frac{FP}{FP + TN} = 1 - Specificity \tag{4.32}$$

To plot the ROC curve, the classification threshold is first set to a value such that all samples are classified as negative samples, this is going to incur in a FPR of 0% and a TPR of 0%, then this threshold is gradually changed until all samples are classified as positives, leading to a FPR of 100% and a TPR of 100%. All the points in between are plotted on the graph, and points that are closer to upper left corner of the graph signal a good classification since they provide a good TPR for a low compromise in FPR. So, the ROC chart works as a way of assessing the trade-offs between TPs and FPs. ROC plots that are above the diagonal line signify a model that is better than random classification, and the closer the graphs are to the upper and left border, the better they are. Figure 4.11 shows different ROC plots.

Taking all of this into account, the AUC value can range between 0 and 1 or 0% and 100%, and an AUC above 0.5 or 50% signifies better than random classification.

Classification error on the test set is another metric for estimating model correctness, and its value directly depends on the selected cost function.

AUC would be desired as the selection metric for the tests since it has been shown to be a better metric than accuracy to evaluate the predictive ability of learning algorithms [89]. As an alternative, the arithmetic mean between the sensitivity, specificity and accuracy values can be used to select networks in a similar way to the AUC value.

Performance Metrics

Model performance is also important since it would be valuable to achieve the best solution in the least time possible. This includes measuring the time for training and the number of training epochs. The amount of variation in

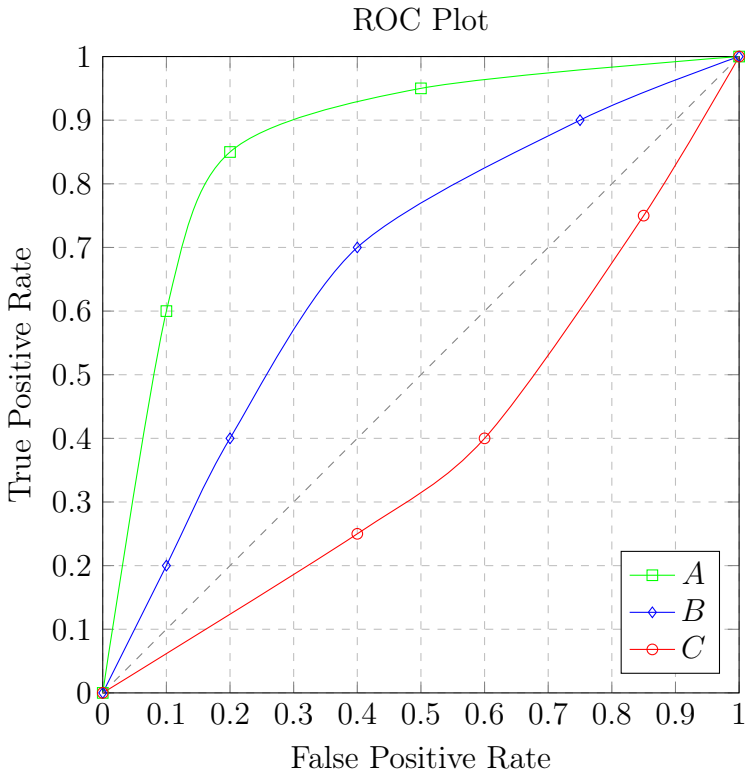

Figure 4.11: Three different ROC charts. Classification for graph A is better than for graph B, and graph C is worse than random classification.

terms of correctness in the trained networks is also an important measure as a too high variation may signify a few problems like the model having a high amount of dependence on the initial weights. The spread can be measured by directly applying measures such as the standard deviation on the correctness measures.

4.6.2 Evaluation of a single classifier

To evaluate the results in a more detailed manner, the results of a single trained classifier can be also summarized using a set of four charts:

- **Confusion Matrix:** organizes the number of samples in a table according to their original labels (targets) and the classification labels

(predictions). It can used to calculate the number of TP, TN, FP, and FN as well as the accuracy, sensitivity and specificity values for the classifier. The amount of misclassifications can be easily understood using this chart. Figure 4.12 presents a confusion matrix where class 1 represents not-A samples and class 2 represents A samples. In this confusion matrix the bottom three percentages from left to right are the specificity, the sensitivity and the accuracy values. As an example, 460 A-phase samples were classified as not-A samples by the classifier.

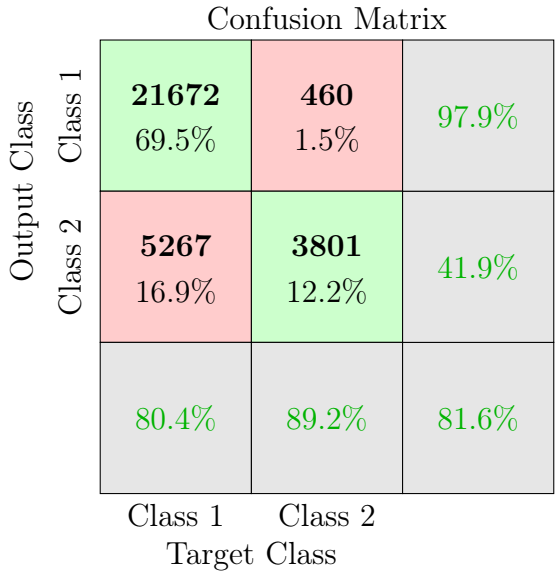

Figure 4.12: An example of a confusion matrix for a trained classifier.

- **ROC chart:** plots the TPR for each FPR at different classification threshold values for each class of samples.

- **Train state plot:** displays gradient magnitude and number of failed validation checks at each epoch. The gradient magnitude can be used to assess the learning speed at different epochs. The number of failed validation checks indicates if the model has problems such as overfitting. Figure 4.13 presents a train state plot where the first graph plots the gradient value for each training epoch and the second graph plots

the number of repeated failed validation checks for each training epoch.

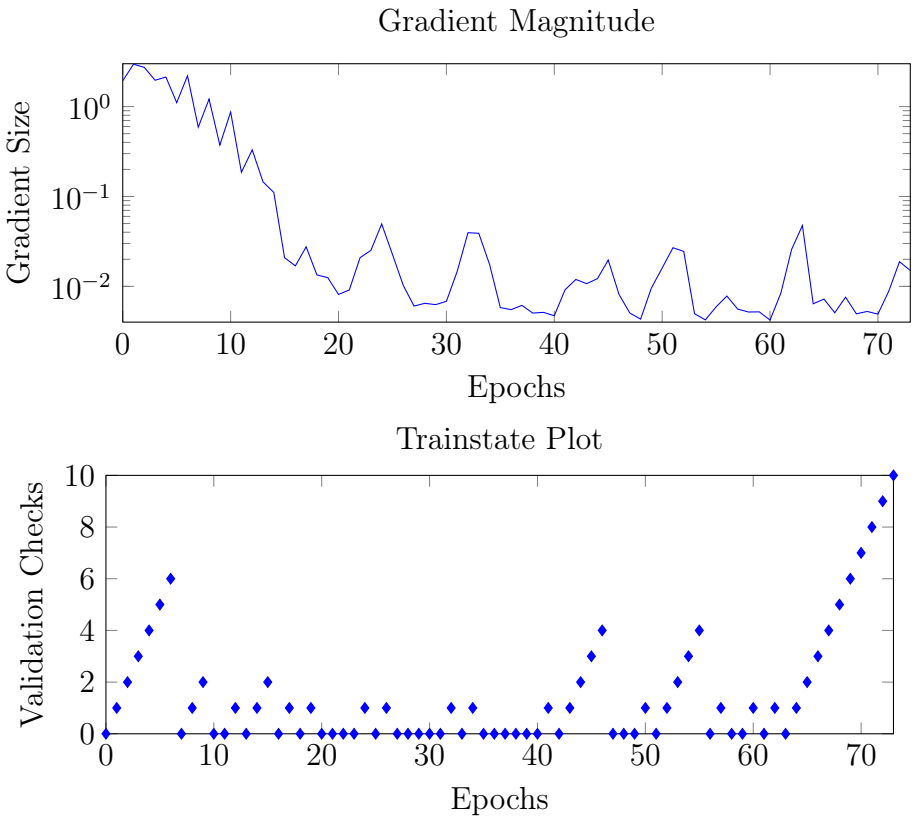

Figure 4.13: Train state plot for a trained classifier. The first graph plots the gradient value for each training epoch and the second graph plots the number of repeated failed validation checks for each training epoch.

- **Performance plot:** shows the cost function value for the training, validation and/or test sets at each training epoch. Problems such as underfitting, overfitting and others can be easily detected on the performance chart. Figure 4.14 presents an example of a performance plot with cross-entropy as the cost function.

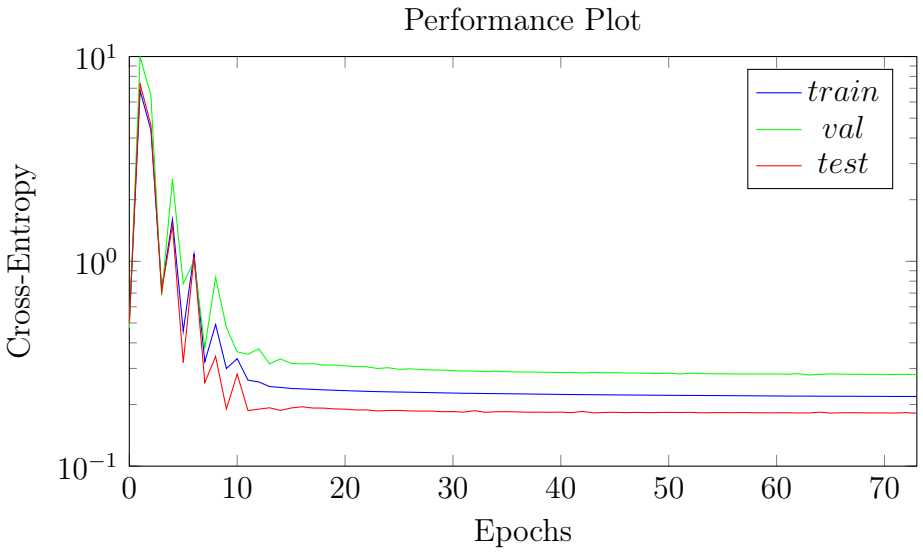

Figure 4.14: Performance plot for a trained classifier.

4.7 Key Remarks

A classification FFNN was utilized for the detection algorithm. The training algorithm is going to improve the results of the neural network by changing its weights and biases. Processes such as the division of data, the weight initialization, and the implementation of the network were thoroughly explained.

Multiple parameters needed to be selected for the creation of the neural network. To prevent problems such as overfitting and underfitting, model tuning was executed using a genetic algorithm. This algorithm selected the network's parameters that provided the best results in terms of AUC.

Finally, feature selection was applied on three different occasions. Initially, a feature selection method using the mRMR algorithm was used to selected features for the tuning process. Then, feature selection was applied using SFS to select features with the tuned model. Lastly, a final feature selection method was applied using PCA to further reduce the dimensionality of the input variables.

Chapter 5

Results

The main results of the proposed solution are going to be thoroughly presented in this chapter. A discussion is going to follow the presentation of the results.

5.1 Initial Results

A set of initial results had to be obtained to have a comparison measure for the improvements done in the subsequent sections. With the features and the model fully implemented, two fold cross validation was applied on an initial model with initial parameters and all of the available features as inputs. These parameters were arbitrarily selected taking into account the problem at hand. Each fold was trained 50 times to achieve statistical significance in the results, therefore 100 neural networks were trained in total. The results of these networks in terms of AUC, Accuracy, Sensitivity and Specificity are displayed on table 5.1. The mean, standard deviation, minimum and maximum values of each metric is presented on the table.

Table 5.1: Results using two fold cross validation with all the features.

	AUC(%)	Acc(%)	Sen(%)	Spe(%)
Mean	75.21	70.49	65.91	71.43
Std	8.5	7.76	9.71	8.91
Min	37.36	35.63	42.23	34.27
Max	83.13	78.75	84.69	84.11

5.2 mRMR Feature Selection

After testing the model with the initial conditions, mRMR was run to obtain scores associated with each feature. The model parameters from the previous test were maintained for this selection. As explained, the available features were then progressively added to the existing model and tested by order of score given by the mRMR algorithm. Each set of features was tested using LOO with 10 repetitions per fold. With this finished, a smaller set of 13 features was selected. On the trained conditions, the set provided an AUC value of 80.57%. According to the scoring, the best feature was Shannon Entropy and the selected set also contained Amplitude Variation and PSD on the Delta, Theta and Beta 1 ranges. The AUC values for each iteration of the mRMR algorithm can be seen on figure 5.1. The selected features are also presented on table 5.2. This table also presents the AUC values for the set containing the presented feature and the features above in terms of score. For the model tuning, the selected 13 features were used as inputs. This allowed for a faster model tuning as the amount of inputs is reduced.

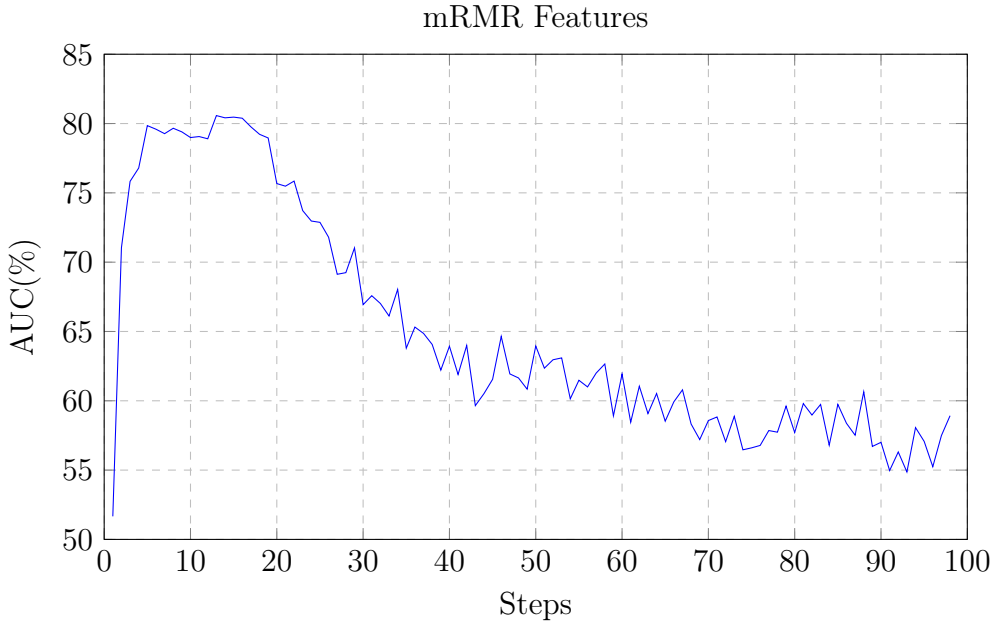

Figure 5.1: AUC Value for each set of features.

Table 5.2: Table containing the features selected by the mRMR.

Number	Designation	AUC(%)	Sen(%)	Spe(%)	Acc(%)
8	Shannon	51.66	41.42	61.06	57.88
45	Skewness (c4)	71.05	66.42	63.34	63.51
48	Log Energy (c4)	75.83	69.85	68.47	68.01
93	Skewness (c14)	76.79	70.96	67.62	67.30
16	Welch PSD (theta)	79.85	73.97	71.18	70.49
86	Kurtosis (c13)	79.59	74.12	70.57	70.06
5	Amplitude Variation	79.27	73.69	69.60	69.06
78	Kurtosis (c10)	79.66	74.03	70.56	69.99
17	Welch PSD (alpha)	79.40	74.26	69.36	68.95
89	Renyi (c13)	78.99	73.52	69.98	69.52
35	Mean (c3)	79.06	73.94	70.27	69.89
15	Welch PSD (delta)	78.90	73.68	70.03	69.42
19	Welch PSD (beta 1)	80.57	75.27	70.66	70.27

5.3 Model Tuning

Having completed the mRMR feature selection, model tuning was executed using the initially discussed configuration and the features selected by the previous selection step. Each individual configuration was tested five times using two fold cross validation. As explained, the GA was carried out with a population of 20 per generation, a mutation rate of 5%, 10% of mutation children, a maximum number of generations of 50 and the number of elites was set to two. The results were summarized using a set of graphs and tables. The diversity values for each generation of the genetic algorithm are displayed on figure 5.2. The mean and best fitness values for each generation are demonstrated on the graphs presented in figure 5.3. The algorithm achieved a minimum fitness value of 0.18979 at generation 15, which is equivalent to an AUC of 81.02%. The network configuration that managed to achieve these results was composed by two hidden layers of 190 neurons each, the activation function in these layers was the sigmoid, the regularization parameter was set to 0.07, and the training algorithm was the Rprop.

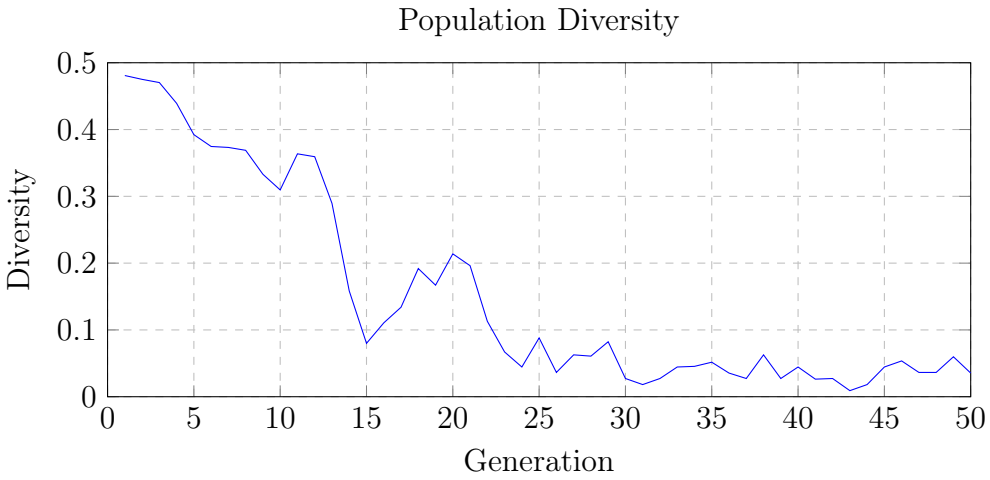

Figure 5.2: Population diversity values for each generation.

5.4 SFS Feature Selection

With the tuned model, feature selection using SFS was carried out. This selection step was carried out on all of the available features and not on the features selected by the mRMR algorithm. This is because the initial selection step had the objective of improving the tuning of the model and not selecting features for the final model. In each step of the SFS algorithm the model was tested using two fold cross validation with one repetition. The set of features that provided that highest mean AUC value contained 16 features. This set managed to achieve an AUC of 83.0%, a sensitivity of 74.0%, a specificity of 73.2% and an accuracy of 72.6% on two fold cross validation. The results for the multiple sets in terms of AUC, accuracy, sensitivity and specificity are presented on table 5.3. Each row of the table corresponds the results for the set containing the features from the first row to the row in question. According to the selection, PSD on the Delta Band using Welch's method is the best feature. For comparison purposes, the results of the model using the selected features on LOO are presented on table 5.4, together with the LOO results using the initial model and all features. The values on the left columns correspond to the results from the initial model, while the values on the right columns correspond to results provided by the tuned model with the features selected by SFS. Each row represents the results for one specific patient assigned as the test set, with the rest of the patients being used

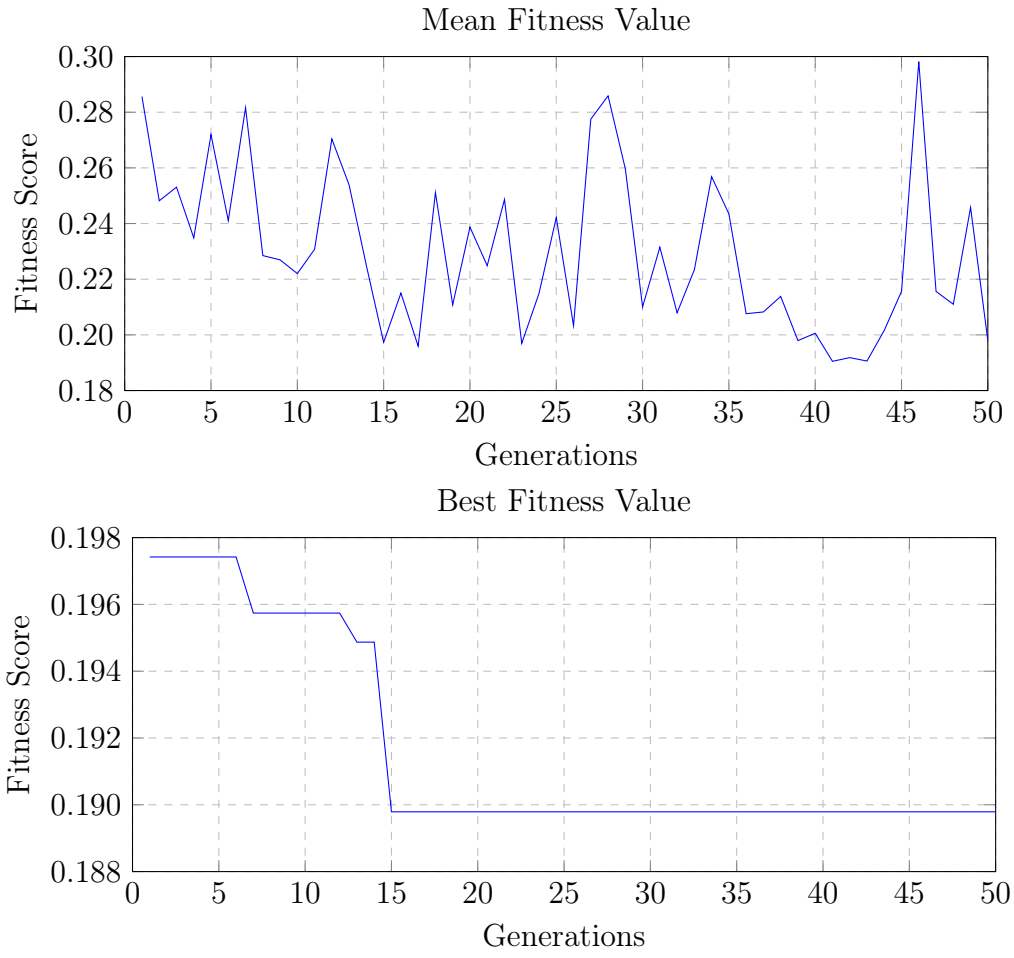

Figure 5.3: Fitness value for each generation.

for training and validation. The mean, standard deviation, minimum and maximum of the parameters are also presented on the bottom of the table.

5.5 PCA

With the 16 features selected by the SFS, PCA was employed to further reduce the number of features necessary. This reduction was directly applied on the 16 features. With a cumulative percentage of variance value of 81.98%, six principal components were extracted. The collective percentage

Table 5.3: Table containing the AUC, sensitivity, specificity and accuracy values for each step of the SFS.

Number	Designation	AUC(%)	Sen(%)	Spe(%)	Acc(%)
15	Welch PSD (delta)	72.5	76.3	39.9	44.2
60	Variance (c6)	75.8	66.3	69.0	68.4
50	Tsallis (c4)	77.5	63.8	76.6	74.8
19	Welch PSD (beta 1)	80.4	73.5	71.6	71.6
98	Tsallis (c14)	81.2	74.4	72.4	72.6
36	Variance (c3)	81.8	67.5	78.7	77.1
7	Amplitude Range	81.5	71.3	75.2	74.7
2	Variance	82.4	72.9	74.9	73.9
18	Welch PSD (sigma)	82.7	73.2	74.8	74.1
78	Kurtosis (c10)	82.3	71.5	74.3	73.3
71	Shannon (c9)	81.9	73.6	72.1	71.6
73	Renyi (c9)	81.3	74.4	73.3	73.4
46	Kurtosis (c4)	82.0	72.9	75.3	74.8
96	Log Energy (c14)	81.9	78.5	70.2	71.1
64	Log Energy (c6)	81.2	73.5	74.0	73.9
81	Renyi (c10)	83.0	74.0	73.2	72.6

of variance covered by the principal components is displayed on figure 5.5. The six extracted components were used as features using the model with the parameters selected by the GA and retrained with LOO. The results obtained from the trained networks are summarized on table 5.5.

5.6 Data Removal

To visualize the relationship between the amount of training data and the network performance, removal of the training data was applied in steps of 20% and at each step the final model was retrained using the reduced data. This process can work as an indicator of the necessity for additional data in the model. The results of this analysis in terms of AUC are presented on figure 5.6.

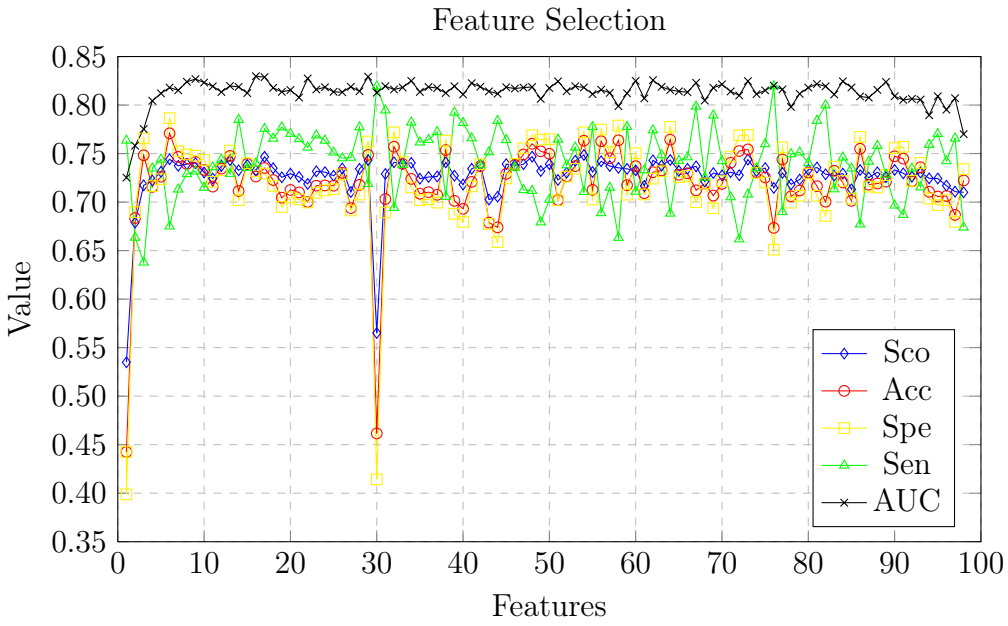

Figure 5.4: Mean AUC Value for each step of the SFS algorithm. The best mean AUC value is plotted for each iteration of tests.

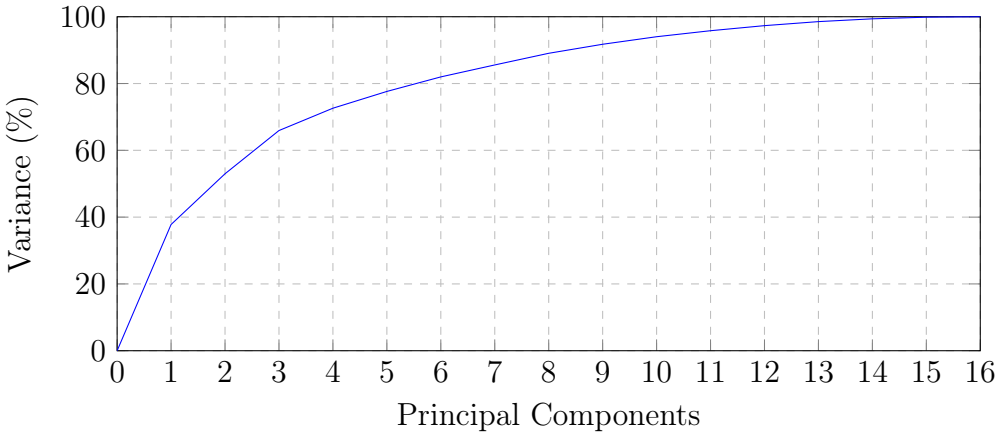

Figure 5.5: Cumulative percentage of variance covered by the principal components.

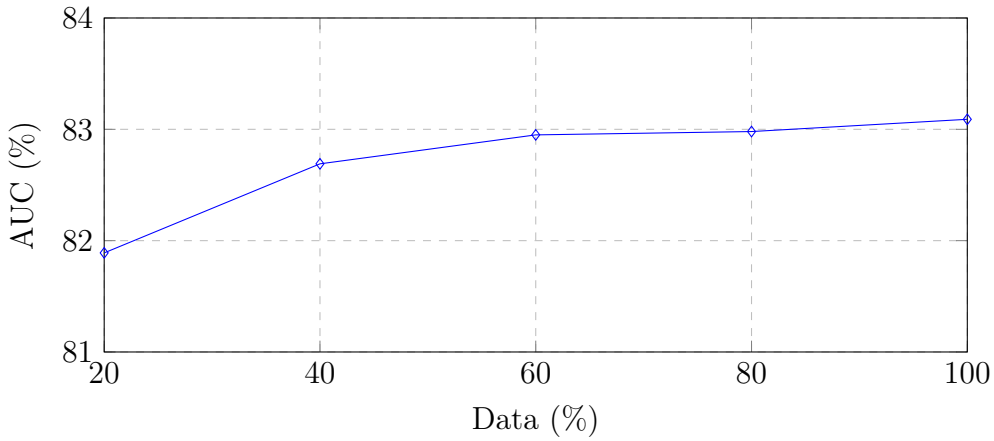

Figure 5.6: AUC value for different percentages of training data used on the final model.

5.7 Discussion

The initial networks using all the available features show good results with an average AUC value of 75.21% using two fold cross validation, and of 79.27% using LOO. Still, improvements were made to obtain higher correctness metrics.

With this in mind, the initial mRMR selection method managed to select a set of 13 features which provided an AUC of 80.57% on LOO. This indicates an increase of 1.3% from the initial model. In figure 5.1, it is observable that the AUC roughly maintains its value between the first 5 and 19 features. Any of the sets outside these boundaries would not be desirable for the selection. As seen, too few features might not provide enough information to the model for training, while too many features might weaken the model by giving it unnecessary information and complexity.

In the genetic algorithm, the best fitness value at the last epoch reaches a minimum value of 0.18979 at generation number 15, there is also a high drop in the population diversity in the generations that follow. This indicates that the algorithm has saturated to a specific set of individuals that are very similar and therefore the evolution is slowed down. After about generation 25, the diversity values are close to the mutation chance (5%), from here on the diversity does not drop significantly since mutations are keeping the diversity at these values. The average fitness score of the generations seems to

fluctuate, but a global down trend is observable during the initial generations.

Finally, the subsequent SFS managed to collect a total of 16 features with the tuned model. This selection criteria was based on the AUC value. Also, the highest point in terms of sensitivity occurred at step 30, and the peak specificity and accuracy values occurred at step 6. After about step 6 the AUC roughly maintained its value. This suggests that the first six features would be enough for the model to obtain decent results. At step 30 an instance of high sensitivity and low accuracy and specificity values is observed, most probably caused by a bad random combination of folds. Compared to the initial results in terms of LOO, the tuned model with the 16 selected features managed to increase its average AUC value by 3.71%. The average accuracy, sensitivity and specificity values increased by 1.79%, 4.21% and 1.42% respectively. These results show that both feature selection and model tuning are able to improve the results of a trained model to a significant degree by reducing the complexity of the model and eliminating unnecessary information.

The model performed worse on patients that had sleep disordered breathing shown by an average AUC of 71.19%, which is lower than the average AUC for healthy patients by 14.93% (86.12%). This is caused by the fact that these patients are harder to classify as well as the fact that there are a lower amount of sleep disordered breathing patients than healthy individuals on the data set. Therefore, the data provided by these patients cause the overall results to be worse than if only healthy individuals were used, but the employed approach illustrates a more realistic picture of how the model would perform under real circumstances.

From the data reduction process demonstrated in figure 5.6, it can be concluded that an increase in the amount of training data would not cause a significant difference on the final results in terms of AUC. This is indicated by the visible asymptote at a value close to 83%. Also, the difference in terms of AUC between using 80% of the original data and 100% is of only 0.11%.

The further reduction of dimensionality provided by PCA lowered the amount of input variables from 16 to six, while also reducing the mean AUC value by 1.47%. Taking into account the tuned structure of the network, the amount of weights and biases of the model is reduced by 4.76%. Also, the ratio between the AUC and the number of weights and biases is improved by 3.14%. Therefore, the cut in the number of inputs managed to reduce the complexity of the model while also slightly increasing the ratio between the performance and the complexity of the algorithm.

For comparison purposes, table 5.6 presents the results of some of the studies discussed in the state of art together with the results of this work. The works included in this analysis mentioned using sensitivity, specificity and accuracy to evaluate their models. Of these, the approaches presented by Navona et al. [26], Mariani et al. [29–31], Mendez et al. [34] and Niknazar et al. [38] used only data from healthy individuals. Five of these works managed to achieve a higher accuracy than this work with the third work by Mariani et al. [31] retaining the highest accuracy value. In terms of sensitivity, this work obtained a higher performance than four of the six works. As explained, the usage of data from patients with sleep conditions has the capability of reducing the accuracy of the model. As such, it is expected that the model trained in this work to achieve a lower performance. Therefore, the results of this work were obtained in a significantly more challenging population, making the direct comparison with other state-of-the-art works difficult. Nevertheless, the achieved results are likely to be more generalisable as both healthy and sleep disordered subjects were examined and employed to produce the models.

Of the three remaining works, which used data from individuals with some sleep condition, the work presented by Mostafa et al. [39] obtained lower performance metrics with an accuracy of 67%. The work employed the use of a featureless method with a combination of a deep auto encoder and a shallow neural network. The works presented by Mendonça et al. [36, 37] attained both superior results with accuracy values of 79% and 76% reached by means of a LSTM and a FFNN respectively. However, in terms of sensitivity, both works managed to obtain a weaker value. It is important to highlight that a high sensitivity is of the upmost importance for CAP A Phase analysis as this metric will determine the ability of the model to designate an individual epoch labeled as A phase in the data set as an A phase. Also, taking into consideration the strong data imbalance in the A phase distribution, having a high sensitivity is crucial to have a model useful for real world applications. Both of these works applied a post procedure to the data returned by the model which improved the results by reducing the amount of misclassifications on the boundaries between CAP events. Taking this into account, the presented work holds a performance similar to the performance of the featureless LSTM presented by Mendonça et al. [36]. Also, the second work by Mendonça et al. [37] extracted eleven features of which five where selected by the SFS. In contrast, this work extracted a total of 98 features of which 16 where selected for the final model.

5.8 Key Remarks

An initial model was trained with all the available features to make an introductory evaluation of the proposed model. Then, 13 features were selected using the mRMR selection algorithm which resulted in an increase of 1.3 % in the AUC value. This selection step was crucial to reduce the amount of features for the tuning process as all features would result in a slower training of the networks necessary for the GA. These features were then fed to the model and, as explained, tuning was employed using the GA to find the most adequate parameters for the networks. Afterwards, feature selection was executed again on the tuned model using SFS, where 16 features were selected. This resulted in an increase of 3.71% in terms of AUC from the initial model. The performance of the tuned model with the selected features managed to attain a performance of 82.98%, 73.47%, 77.26% and 73.63% in terms of AUC, accuracy, sensitivity and specificity respectively. Finally, PCA was applied over the selected features and resulted in a reduction from 16 to six features with the performance to complexity ratio increasing by 2.43 %.

Compared to the previous works done in the area, the results attained by this final model managed to achieved a similar performance when looking at the works that employed a comparable approach. This work focused on achieving a balanced results with the sensitivity and specificity having similar values. This allowed for more useful results for real case scenarios.

Table 5.4: LOO results on the initial model and the final model.

Patient	AUC(%)		Acc(%)		Sen(%)		Spe(%)	
	initial	final	initial	final	initial	final	initial	final
1	86.52	89.38	74.14	73.92	84.84	90.01	72.69	71.75
2	78.26	80.55	69.49	71.10	73.53	76.83	69.02	70.44
3	79.17	84.91	66.91	76.81	78.18	75.42	65.96	76.93
4	76.45	81.14	69.45	71.63	70.95	76.25	69.33	71.26
5	87.00	89.63	74.47	77.5	85.51	86.29	72.79	76.17
6	89.51	91.12	82.82	82.46	79.15	85.62	83.40	81.96
7	86.10	89.92	74.54	73.46	87.16	95.14	73.29	71.31
8	79.00	80.62	73.60	70.23	67.10	75.79	74.40	69.54
9	84.47	90.41	76.14	77.90	80.30	90.72	75.83	76.97
10	73.68	78.99	63.98	69.41	72.73	73.79	62.94	68.89
11	79.53	83.88	67.11	72.79	79.62	80.14	65.74	71.99
12	85.98	85.97	80.66	76.64	77.15	82.09	81.17	75.85
13	81.26	86.67	78.72	78.91	70.27	80.35	79.87	78.71
14	88.66	90.60	83.10	80.46	77.68	86.67	83.89	79.55
15	85.87	87.99	78.13	75.47	80.86	87.12	77.69	73.59
16	71.56	72.37	58.42	66.45	73.84	65.80	57.03	66.51
17	72.61	76.58	72.35	73.55	48.97	51.63	80.69	81.36
18	63.18	67.16	60.29	63.12	54.47	51.97	63.00	68.33
19	57.40	68.66	57.58	64.18	45.61	56.38	63.24	67.87
Mean	79.27	82.98	71.68	73.47	73.05	77.26	72.21	73.63
Std	10.49	7.62	9.27	6.61	12.85	14.55	9.52	7.52
Min	57.40	67.16	57.58	63.12	45.61	51.63	57.03	66.51
Max	89.51	90.41	83.10	80.46	87.16	95.14	83.89	81.96

Table 5.5: Results using LOO with the six first principal component.

	AUC(%)	Acc(%)	Sen(%)	Spe(%)
Mean	81.51	70.79	77.41	70.50
Std	7.80	6.82	14.10	7.74
Min	56.52	52.30	20.76	49.88
Max	91.21	88.37	96.41	89.99

Table 5.6: Comparison between of the state-of-the-art results and the presented solution.

Work	Subjects	Method	Acc(%)	Sen(%)	Spe(%)
[39]	14	EEG directly fed on a classifier	67	55	69
[29]	8	Features with thresholds for classification	72	52	76
[37]	19	EEG directly fed to three classifiers	76	75	77
[26]	10	Frequency features with thresholds for classification	77	84	90
[36]	14	Multiple features on nine classifiers	79	76	80
[38]	5	threshold RMS and SLBE	81	76	81
[34]	5	kNN with entropy and spectral features	82	87	74
[30]	8	Features fed in 3 classifiers	85	73	87
[31]	16	Linear discriminant function with segmentation step	86	67	90
this	19	Multiple features on FFNN	73	77	74

Chapter 6

Conclusion

6.1 Overview of the work

The advent of sleep related conditions lead to the necessity of advancements in their diagnosis. CAP is an important indicator of these conditions and works as an essential tool for the detection of patterns in the sleep structure that relate to instability in the electrical activity of the brain. Due to the slow nature of the manual classification of CAP phases, the necessity for the automatic detection of CAP is evident. For that, features were extracted from the EEG samples of 19 individuals to build an A-phase classification algorithm. The descriptors were collected based on the problem knowledge and previous approaches use to solve the problem at hand. This approach stands in contrast with featureless methods which rely on the model being able to classify samples based purely on the raw signal data.

The extracted features provided relevant information for the classification algorithm. This is proven by the good results obtained during the training of the model. The performance of the model was further improved by means of model tuning and feature selection which proved to be essential for the project. The employed GA managed to select the best parameters for the network using a versatile approach which can be applied in a variety of problems. Also, both the mRMR and SFS selection steps managed to improve the tuning of the model, reduce the complexity of the model and improve the overall results of the classification. Furthermore, the final dimensionality reduction step using PCA further decreased the complexity of the algorithm at a low cost in terms of performance. Finally, all of the additional processes

and techniques such as feature scaling, data balancing and others provided an understanding of the matter at hand and helped on the conception and consolidation of the entire system.

Regarding the literature, the results were in line with the state-of-the-art works in terms of accuracy, sensitivity and specificity. These metrics are themselves standards for the evaluation of these types of ML models. The usage of patients with sleep related disorders solidified the results by presenting a more realistic evaluation of the model performance on real scenarios.

With regards to software, the tools provided by MATLAB facilitated the implementation of the proposed model by allowing the utilization of simple to use functions to implement the neural network and the other processes necessary for the development of the AI system.

During the development of the solution it was observed that testing the method required some careful planning as the training of the algorithm itself required a long execution time. This is due to complexity of the network, the amount of training data and the hardware. In contrast, testing code in other areas of development does not often require waiting for code to run for such long periods of time. Thus, the development of the system can be slowed down.

In summary, the objectives of the work were met. The detection algorithm was implemented and achieved good results in terms of sensitivity, specificity, accuracy and AUC. Overall the project provided valuable contributions to the related areas of knowledge and can lead to future real world applications regarding automatic sleep analysis.

6.2 Limitations of the Work

The use of a data set with a low amount of individuals is not ideal for the evaluation of the method. It would be necessary to test this algorithm on a bigger data set with thousands of patients to more accurately assess its performance. Also, data from patients with different sleep disorders would be required to verify the capabilities of the model on different kinds of conditions. Finally, it was also not possible to test all existing ML algorithms and features. This would allow for a more comprehensible analysis of all the possible methodologies to solve the problem of sleep disorder diagnosis by means of ML.

6.3 Future Work

For the future, multiple class classification of the A phase sub types could be addressed by tuning the model and selecting features under those conditions to evaluate the model performance.

Class balancing techniques could also be more thoroughly tested by training multiple models under different algorithms to check which ones resulted in a better classification of CAP.

Also, feature selection and tuning could be tested together on the same process by using the genetic algorithm with features and parameters simultaneously added on the genetic sequence.

Finally, other ML frameworks such as TensorFlow could be used to implement the algorithm and compared to the solution provided by MATLAB in terms of ease of implementation and performance on the hardware.

Bibliography

[1] J. Zeitlhofer, A. Schmeiser-Rieder, G. Tribl, A. Rosenberger, J. Bolitschek, G. Kapfhammer, B. Saletu, H. Katschnig, B. Holzinger, R. Popovic, *et al.*, "Sleep and quality of life in the austrian population," *Acta Neurologica Scandinavica*, vol. 102, no. 4, pp. 249–257, 2000.

[2] M. P. Walker, "The role of sleep in cognition and emotion," *Annals of the New York Academy of Sciences*, vol. 1156, no. 1, pp. 168–197, 2009.

[3] S. Chokroverty *et al.*, "Overview of sleep & sleep disorders," *Indian J Med Res*, vol. 131, no. 2, pp. 126–140, 2010.

[4] P. J. Moore, N. E. Adler, D. R. Williams, and J. S. Jackson, "Socioeconomic status and health: the role of sleep," *Psychosomatic medicine*, vol. 64, no. 2, pp. 337–344, 2002.

[5] M. Hafner, M. Stepanek, J. Taylor, W. M. Troxel, and C. Van Stolk, "Why sleep matters—the economic costs of insufficient sleep: a cross-country comparative analysis," *Rand health quarterly*, vol. 6, no. 4, p. 11., 2017.

[6] J. F. Dewald, A. M. Meijer, F. J. Oort, G. A. Kerkhof, and S. M. Bögels, "The influence of sleep quality, sleep duration and sleepiness on school performance in children and adolescents: A meta-analytic review," *Sleep medicine reviews*, vol. 14, no. 3, pp. 179–189, 2010.

[7] M. R. Rosekind and K. B. Gregory, "Insomnia risks and costs: health, safety, and quality of life.," *The American journal of managed care*, vol. 16, no. 8, pp. 617–626, 2010.

[8] P. J. Strollo Jr and R. M. Rogers, "Obstructive sleep apnea," *New England Journal of Medicine*, vol. 334, no. 2, pp. 99–104, 1996.

[9] W. R. Ruehland, P. D. Rochford, F. J. O'Donoghue, R. J. Pierce, P. Singh, and A. T. Thornton, "The new aasm criteria for scoring hypopneas: impact on the apnea hypopnea index," *sleep*, vol. 32, no. 2, pp. 150–157, 2009.

[10] C. Trenkwalder, W. Paulus, and A. S. Walters, "The restless legs syndrome," *The lancet neurology*, vol. 4, no. 8, pp. 465–475, 2005.

[11] G. Lavigne, S. Khoury, S. Abe, T. Yamaguchi, and K. Raphael, "Bruxism physiology and pathology: an overview for clinicians," *Journal of oral rehabilitation*, vol. 35, no. 7, pp. 476–494, 2008.

[12] J. R. Hughes and E. R. John, "Conventional and quantitative electroencephalography in psychiatry," *The Journal of Neuropsychiatry and Clinical Neurosciences*, vol. 11, no. 2, pp. 190–208, 1999.

[13] H. H. Jasper, "The ten-twenty electrode system of the international federation," *Electroencephalogr. Clin. Neurophysiol.*, vol. 10, pp. 370–375, 1958.

[14] A. Rechtschaffen, "A manual for standardized terminology, techniques and scoring system for sleep stages in human subjects," *Brain information service*, 1968.

[15] O. B., "International 10-20 system for eeg electrode placement, showing modified combinatorial nomenclature.." [Online] `https://commons.wikimedia.org/wiki/File:International/_10-20/_system/_for/_EEG-MCN.svg`, accessed on 25 January 2021.

[16] L. Matarazzo, A. Foret, L. Mascetti, V. Muto, A. Shaffii, and P. Maquet, "A systems-level approach to human rem sleep," *Rapid Eye Movement Sleep: Regulation and Function*, vol. 8, p. 71, 2011.

[17] D. Markov and M. Goldman, "Normal sleep and circadian rhythms: neurobiologic mechanisms underlying sleep and wakefulness," *Psychiatric Clinics*, vol. 29, no. 4, pp. 841–853, 2006.

[18] J. E. Brinkman and S. Sharma, "Physiology, sleep," *StatPearls [Internet]*, 2019.

[19] R. Cabiddu, S. Cerutti, S. Werner, G. Viardot, and A. M. Bianchi, "Modulation of the sympatho-vagal balance during sleep: frequency domain study of heart rate variability and respiration," *Frontiers in physiology*, vol. 3, p. 45, 2012.

[20] M. G. Terzano, L. Parrino, A. Sherieri, R. Chervin, S. Chokroverty, C. Guilleminault, M. Hirshkowitz, M. Mahowald, H. Moldofsky, A. Rosa, *et al.*, "Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (cap) in human sleep.," *Sleep medicine*, vol. 2, no. 6, pp. 537–553, 2001.

[21] L. Parrino, R. Ferri, O. Bruni, and M. G. Terzano, "Cyclic alternating pattern (cap): the marker of sleep instability," *Sleep medicine reviews*, vol. 16, no. 1, pp. 27–45, 2012.

[22] F. Mendonça, S. S. Mostafa, F. Morgado-Dias, and A. G. Ravelo-García, "Cyclic alternating pattern estimation based on a probabilistic model over an eeg signal," *Biomedical Signal Processing and Control*, vol. 62, p. 102063, 2020.

[23] A. Rosa, G. R. Alves, M. Brito, M. C. Lopes, and S. Tufik, "Visual and automatic cyclic alternating pattern (cap) scoring," *Arquivos de neuro-psiquiatria*, vol. 64, no. 3a, pp. 578–581, 2006.

[24] A. A., "Cap phase detection implementation on matlab." [Online] `https://github.com/SvelaT/arturo_sleep`.

[25] U. Barcaro, C. Navona, S. Belloli, E. Bonanni, C. Gneri, and L. Murri, "A simple method for the quantitative description of sleep microstructure," *Electroencephalography and clinical neurophysiology*, vol. 106, no. 5, pp. 429–432, 1998.

[26] C. Navona, U. Barcaro, E. Bonanni, F. Di Martino, M. Maestri, and L. Murri, "An automatic method for the recognition and classification of the a-phases of the cyclic alternating pattern," *Clinical neurophysiology*, vol. 113, no. 11, pp. 1826–1831, 2002.

[27] A. Rosa, L. Parrino, and M. Terzano, "Automatic detection of cyclic alternating pattern (cap) sequences in sleep: preliminary results," *Clinical neurophysiology*, vol. 110, no. 4, pp. 585–592, 1999.

[28] F. Karimzadeh, E. Seraj, R. Boostani, and M. Torabi-Nami, "Presenting efficient features for automatic cap detection in sleep eeg signals," in *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, pp. 448–452, IEEE, 2015.

[29] S. Mariani, E. Manfredini, V. Rosso, M. O. Mendez, A. M. Bianchi, M. Matteucci, M. G. Terzano, S. Cerutti, and L. Parrino, "Characterization of a phases during the cyclic alternating pattern of sleep," *Clinical Neurophysiology*, vol. 122, no. 10, pp. 2016–2024, 2011.

[30] S. Mariani, E. Manfredini, V. Rosso, A. Grassi, M. O. Mendez, A. Alba, M. Matteucci, L. Parrino, M. G. Terzano, S. Cerutti, *et al.*, "Efficient automatic classifiers for the detection of a phases of the cyclic alternating pattern in sleep," *Medical & biological engineering & computing*, vol. 50, no. 4, pp. 359–372, 2012.

[31] S. Mariani, A. Grassi, M. O. Mendez, G. Milioli, L. Parrino, M. G. Terzano, and A. M. Bianchi, "Eeg segmentation for improving automatic cap detection," *Clinical Neurophysiology*, vol. 124, no. 9, pp. 1815–1823, 2013.

[32] F. Machado, F. Sales, C. Bento, A. Dourado, and C. Teixeira, "Automatic identification of cyclic alternating pattern (cap) sequences based on the teager energy operator," in *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 5420–5423, IEEE, 2015.

[33] R. Largo, C. Munteanu, and A. Rosa, "Wavelet based cap detector with ga tuning.," *WSEAS Transactions on Information Science and Applications*, vol. 2, no. 5, pp. 576–580, 2005.

[34] M. O. Mendez, A. Alba, I. Chouvarda, G. Milioli, A. Grassi, M. G. Terzano, and L. Parrino, "On separability of a-phases during the cyclic alternating pattern," in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pp. 2253–2256, IEEE, 2014.

[35] R. Ferri, O. Bruni, S. Miano, A. Smerieri, K. Spruyt, and M. G. Terzano, "Inter-rater reliability of sleep cyclic alternating pattern (cap) scoring

and validation of a new computer-assisted cap scoring method," *Clinical neurophysiology*, vol. 116, no. 3, pp. 696–707, 2005.

[36] F. Mendonça, A. Fred, S. S. Mostafa, F. Morgado-Dias, and A. G. Ravelo-García, "Automatic detection of cyclic alternating pattern," *Neural Computing and Applications*, pp. 1–11, 2018.

[37] F. Mendonça, S. S. Mostafa, F. Morgado-Dias, and A. G. Ravelo-García, "A portable wireless device for cyclic alternating pattern estimation from an eeg monopolar derivation," *Entropy*, vol. 21, no. 12, p. 1203, 2019.

[38] H. Niknazar, S. Seifpour, M. Mikaili, A. M. Nasrabadi, and A. K. Banaraki, "A novel method to detect the a phases of cyclic alternating pattern (cap) using similarity index," in *2015 23rd Iranian Conference on Electrical Engineering*, pp. 67–71, IEEE, 2015.

[39] S. S. Mostafa, F. Mendonça, A. Ravelo-García, and F. Morgado-Dias, "Combination of deep and shallow networks for cyclic alternating patterns detection," in *2018 13th APCA International Conference on Automatic Control and Soft Computing (CONTROLO)*, pp. 98–103, IEEE, 2018.

[40] R. Largo, C. Munteanu, and A. Rosa, "Cap event detection by wavelets and ga tuning," in *IEEE International Workshop on Intelligent Signal Processing, 2005.*, pp. 44–48, IEEE, 2005.

[41] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on knowledge and data engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[42] N. Poolsawad, C. Kambhampati, and J. Cleland, "Balancing class for performance of classification with a clinical dataset," in *proceedings of the World Congress on Engineering*, vol. 1, pp. 1–6, 2014.

[43] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[44] I. Tomek *et al.*, "Two modifications of cnn.," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-6, no. 11, pp. 769–772, 1976.

[45] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[46] M. Kukar, I. Kononenko, *et al.*, "Cost-sensitive learning with neural networks.," in *ECAI*, vol. 15, pp. 88–94, Citeseer, 1998.

[47] P. Dangeti, *Statistics for machine learning*. Packt Publishing Ltd, 2017.

[48] I. The MathWorks, "Statistics and machine learning toolbox™ user's guide(r2021b)." [Online] `https://www.mathworks.com/help/pdf_doc/stats/stats.pdf`, accessed on 13 December 2021.

[49] I. The MathWorks, "Wavelet toolbox™ user's guide(r2021b)." [Online] `https://www.mathworks.com/help/pdf_doc/wavelet/wavelet_ug.pdf`, accessed on 13 December 2021.

[50] A. Rényi, "On measures of entropy and information," in *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, vol. 4.

[51] C. Tsallis, "Possible generalization of boltzmann-gibbs statistics," *Journal of statistical physics*, vol. 52, no. 1, pp. 479–487, 1988.

[52] I. The MathWorks, "Matlab® function reference(r2021b)." [Online] `https://www.mathworks.com/help/pdf_doc/matlab/matlab_ref.pdf`, accessed on 13 December 2021.

[53] J. F. Kaiser, "On a simple algorithm to calculate the'energy'of a signal," in *International conference on acoustics, speech, and signal processing*, pp. 381–384, IEEE, 1990.

[54] M. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, no. 4, pp. 385–397, 1996.

[55] P. Bruce and A. Bruce, *Practical statistics for data scientists: 50 essential concepts*. " O'Reilly Media, Inc.", 2017.

[56] nishkarsh146, "Normalization vs standardization." [Online] `https://www.geeksforgeeks.org/normalization-vs-standardization/`, accessed on 26 October 2021.

[57] S. Raschka, *Python machine learning*. Packt publishing ltd, 2015.

[58] G. Bonaccorso, *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[59] S. I. Gallant and S. I. Gallant, *Neural network learning and expert systems*. MIT press, 1993.

[60] B. Widrow, D. E. Rumelhart, and M. A. Lehr, "Neural networks: applications in industry, business and science," *Communications of the ACM*, vol. 37, no. 3, pp. 93–106, 1994.

[61] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[62] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *IEEE international conference on neural networks*, pp. 586–591, IEEE, 1993.

[63] J. E. Dennis, Jr and J. J. Moré, "Quasi-newton methods, motivation and theory," *SIAM review*, vol. 19, no. 1, pp. 46–89, 1977.

[64] H. B. Curry, "The method of steepest descent for non-linear minimization problems," *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261, 1944.

[65] R. Fletcher, "Conjugate gradient methods for indefinite systems," in *Numerical analysis*, pp. 73–89, Springer, 1976.

[66] S. Koturwar and S. Merchant, "Weight initialization of deep neural networks (dnns) using data statistics," *arXiv preprint arXiv:1710.10570*, 2017.

[67] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *2019 7th international conference on smart computing & communications (ICSCC)*, pp. 1–4, IEEE, 2019.

[68] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and adaptive systems: fundamentals through simulations with CD-ROM*. John Wiley & Sons, Inc., 1999.

[69] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, JMLR Workshop and Conference Proceedings, 2010.

[70] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[71] S. K. Kumar, "On weight initialization in deep neural networks," *arXiv preprint arXiv:1704.08863*, 2017.

[72] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation.," *Encyclopedia of database systems*, vol. 5, pp. 532–538, 2009.

[73] D. M. Hawkins, "The problem of overfitting," *Journal of chemical information and computer sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[74] H. Jabbar and R. Z. Khan, "Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)," *Computer Science, Communication and Instrumentation Devices*, vol. 70, pp. 163–172, 2015.

[75] L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, 1998.

[76] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural network design*. PWS Publishing Co., 1997.

[77] I. The MathWorks, "Deep learning toolbox™ user's guide(r2021b)." [Online] `https://www.mathworks.com/help/pdf_doc/deeplearning/nnet_ug.pdf`, accessed on 13 December 2021.

[78] J. R. Koza and J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[79] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[80] Q. Wang, "Using genetic algorithms to optimise model parameters," *Environmental Modelling & Software*, vol. 12, no. 1, pp. 27–34, 1997.

[81] M. Dash and H. Liu, "Feature selection for classification," *Intelligent data analysis*, vol. 1, no. 1-4, pp. 131–156, 1997.

[82] V. Kumar and S. Minz, "Feature selection: a literature review," *SmartCR*, vol. 4, no. 3, pp. 211–229, 2014.

[83] S. Khalid, T. Khalil, and S. Nasreen, "A survey of feature selection and feature extraction techniques in machine learning," in *2014 science and information conference*, pp. 372–378, IEEE, 2014.

[84] C. Ding and H. Peng, "Minimum redundancy feature selection from microarray gene expression data," *Journal of bioinformatics and computational biology*, vol. 3, no. 02, pp. 185–205, 2005.

[85] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[86] A. Baratloo, M. Hosseini, A. Negida, and G. El Ashal, "Part 1: simple definition and calculation of accuracy, sensitivity and specificity," *Emergency*, vol. 3, no. 2, pp. 48–49, 2015.

[87] A. G. Lalkhen and A. McCluskey, "Clinical tests: sensitivity and specificity," *Continuing education in anaesthesia critical care & pain*, vol. 8, no. 6, pp. 221–223, 2008.

[88] J. N. Mandrekar, "Receiver operating characteristic curve in diagnostic test assessment," *Journal of Thoracic Oncology*, vol. 5, no. 9, pp. 1315–1316, 2010.

[89] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.