Department of Electrical and Electronic Engineering

Module EE4050 – Project

Final Report

Grading HIE Injuries in Neonates using Heart Rate Variability

Author's Name:	Andrew Cooke
Student Number:	117379906
Project Partner:	Daragh Crowley
Student Number:	117318203
Supervisor:	Dr. Gordon Lightbody
Date of Submission:	22/04/21

Declaration

"This report was written entirely by the author, except where stated otherwise. The source of any material not created by the author has been clearly referenced. The work described in this report was conducted by the author, except where stated otherwise."

Signed: Andrew Cooke

Date: 22/04/21

Summary

Hypoxic Ischemic Encephalopathy (HIE) is a brain injury in newborns that arises from oxygen deprivation and inadequate blood flow to the brain. The severity of HIE determines the long-term neurodevelopmental outcome of the neonate, with survivors of severe cases often developing cerebral palsy or epilepsy, as well as a host of other neurological and cognitive impairments. HIE is treated using therapeutic hypothermia, which can reduce the risk of the injury deteriorating further and prevent further disability from occurring. However, it is only effective in the first 6-hours following birth. Thus, it is critical that an accurate assessment of the severity of HIE is readily available in the Neonatal Intensive Care Unit (NICU). Continuous electroencephalogram (cEEG) is broadly regarded as the gold standard in assessing the severity of HIE, however the equipment and expertise required to interpret the results is not always available in the NICU.

The electrocardiogram (ECG) is routine recorded in the NICU as it is a non-invasive method of assessing the newborn's heart rate. Previous studies have demonstrated a statistically significant correlation between the severity of HIE injury and the heart rate variability (HRV) of the newborn. This project aimed to develop a subject-independent system that can automatically grade the severity of HIE using the heart rate variability of the neonate. To this end, a number of quantitative measurements of HRV were extracted from the ECG recordings of a cohort of 120 neonates. These features were used in conjunction with classic, feature-based machine learning classifiers that were trained to distinguish between severity grades. The performance of these classifiers was assessed using a subject-independent 'Leave Three Subjects Out' assessment routine, which evaluated their ability to distinguish between severity grades over both 5-minute and 1-hour long windows of data. The best performing feature-based classifiers achieved an epoch-level AUC of 0.77 and a subject-level AUC of 0.78 for the 5-minute and 1-hour windows respectively. Further work in the project investigated the performance of deep learning models to combine the feature extraction and classification stages into one model, improving the subject-level AUC to 0.80. This project used a larger dataset than any previous studies to date and to the author's knowledge is the only attempt at using deep learning for this application.

This report starts out by discussing previous studies in this area as well as their limitations and relevance to this project. The methods used to process the raw ECG data and extract the quantitative HRV features is discussed, and the discriminative power of these features is then demonstrated using a univariate analysis. A background is then given on the machine learning classifiers used in this project, before the results of these models using the 'Leave Three Subjects Out' assessment routine are presented. The theory and results of the deep learning models used in this project is then outlined, before a final discussion of the results and limitations of this project, as well as suggestions for future work in this area, is offered.

Acknowledgements

Firstly, I would like to thank my supervisor Dr. Gordon Lightbody for providing us with the opportunity to carry out this challenging and interesting project. I can say with certainty that both Daragh and I benefitted hugely from our weekly meetings which helped steer us in the right direction and give us confidence in our work.

I would also like to express my gratitude to the members of INFANT, in particular Dr John O' Toole and Dr Andreea Pavel, who provided us with the dataset and ultimately made this project possible. They answered any questions we had about the data and provided us with the code necessary to get started.

I would like to thank my project partner Daragh. I couldn't ask for a more cooperative or hard-working partner to do this project with. I can safely say I won't be forgetting the many hours spent on Teams calls combing through MATLAB scripts or staring at Jupyter notebooks anytime soon.

Thanks to all the staff in the department of Electrical and Electronic Engineering for teaching and guiding me through this degree.

Finally, I'd like to thank my parents, who supported me, fed me and kept a roof over my head for the last four years. I certainly wouldn't have been able to finish this degree without their constant support and motivation.

Table of Contents

List of Figures

List of Tables

List of Acronyms

aEEG	Amplitude-integrated Electroencephalogram
AUC	Area Under Receiver Operating Characteristic Curve
CART	Classification and Regression Trees
cEEG	Continuous Electroencephalogram
CNN	Convolutional Neural Network
ECG	Electrocardiogram
FN	False Negatives
FP	False Positives
HF	High Frequency
HIE	Hypoxic Ischemic Encephalopathy
HRV	Heart Rate Variability
LF	Low Frequency
LTSO	Leave Three Subjects Out
MLP	Multilayer Perceptron
NaN	Not a Number
NICU	Neonatal Intensive Care Unit
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives

Chapter 1: Introduction

Hypoxic Ischemic Encephalopathy (HIE) is a brain injury in newborns that arises from oxygen deprivation and inadequate blood flow to the brain. It is estimated to occur in 1.5 per 1000 live births [1] and is attributed to 23% of newborn deaths around the world [2]. The severity of HIE determines the long-term neurodevelopmental outcome of the neonate, with survivors of severe cases often developing cerebral palsy or epilepsy, as well as a host of other neurological and cognitive impairments.

HIE is treated using therapeutic hypothermia, which minimizes the risk of the injury deteriorating further and can prevent permanent disability from occurring. Therapeutic hypothermia involves whole-body cooling of the newborn to a temperature of 33.5 °C for 72 hours, before slow rewarming is carried out [3]. However, this treatment is only effective in the first 6 hours following birth [4]. Thus, it is imperative that an accurate method of assessment of the severity of HIE is readily available in the Neonatal Intensive Care Unit (NICU) to maximize the efficacy of the treatment.

HIE can be diagnosed using a number of methods. Clinical assessments such as the Apgar score and the Sarnat Scoring System rely on the physical symptoms of the neonate to assess the severity of HIE [5], however studies have shown that these markers are unable to accurately discriminate between severity grades [6]. Magnetic Resonance Imaging (MRI) can be used to determine the severity of HIE; however, it is difficult to obtain in the hours immediately following birth. As well as this, MRI only offers a single snapshot of the brain's activity – it is unable to monitor the evolution of the injury over time, as the severity of HIE can improve or worsen. Continuous electroencephalogram (cEEG) is broadly regarded as the best method for monitoring the severity and evolution of the injury [7]. Up to four different grades of HIE severity can be classified using cEEG [7]. Unfortunately, the expertise to interpret the cEEG is not always available, and amplitude-integrated EEG (aEEG) is frequently used as a substitute in NICUs. The use of aEEG poses its own problems, as it is reported that formal training is often lacking, which could result in a misinterpretation of results [8].

The electrocardiograph (ECG) is routinely recorded in NICUs as it is a non-invasive method of assessing the newborn's heart rate. It has long been known that changes in the inter-heartbeat interval are associated with periods of distress in the neonate [9]. A number of studies have been carried out to investigate the link between heart rate variability (HRV, the variation in the interval between successive heartbeats) and the severity of HIE.

Aliefendioğlu et al investigated the impact of HIE on both the low and high frequency activity of HRV [10]. The severity of HIE was assessed using clinical markers. The low frequency component of the HRV is thought to correspond to the sympathetic component of the nervous system, while the high frequency component is thought to correspond to the parasympathetic component of the nervous system. This study showed that compared to the control group (healthy neonates), the low frequency

(LF) component of the HRV was significantly lower in neonates with HIE while the high frequency (HF) component was higher. The study also showed that there was a larger decrease in the LF component for severe grades of HIE compared to moderate grades. Similarly, there was a greater increase in the HF component for severe grades of HIE compared to moderate grades. However, while this study offered encouraging results, it was carried out 7 days postpartum, and does not offer much insight into the effect of HIE on HRV in the narrow window after birth in which treatment is expected to be maximally effective.

R.M. Goulding et al measured the correlation between HRV and the severity of HIE [11]. The severity of HIE was assessed using EEG recorded 12 – 48 hours postpartum, and each neonate was assigned one of four grades ranging from normal (healthy) to severe. Several HRV features were extracted from the ECG which was recorded in the same timeframe. The features included the low and high frequency components of the HRV, as studied by Aliefendioğlu et al [10] , as well as some features based on the time domain, such as the mean heartbeat interval. The features were calculated using 5-minute epochs of the ECG and were then averaged across the entire recording. A statistically significant negative correlation was discovered between all the HRV features used and the EEG grade of HIE, meaning that HRV decreased as the severity of HIE increased. The study also identified a negative correlation between HRV and the 2-year neurodevelopmental outcome of the neonates with HIE.

The association between HRV and severity of HIE can be visualized by plotting against time the RR interval of both a healthy newborn and a newborn with a severe grade of HIE, as shown in Figure 1.

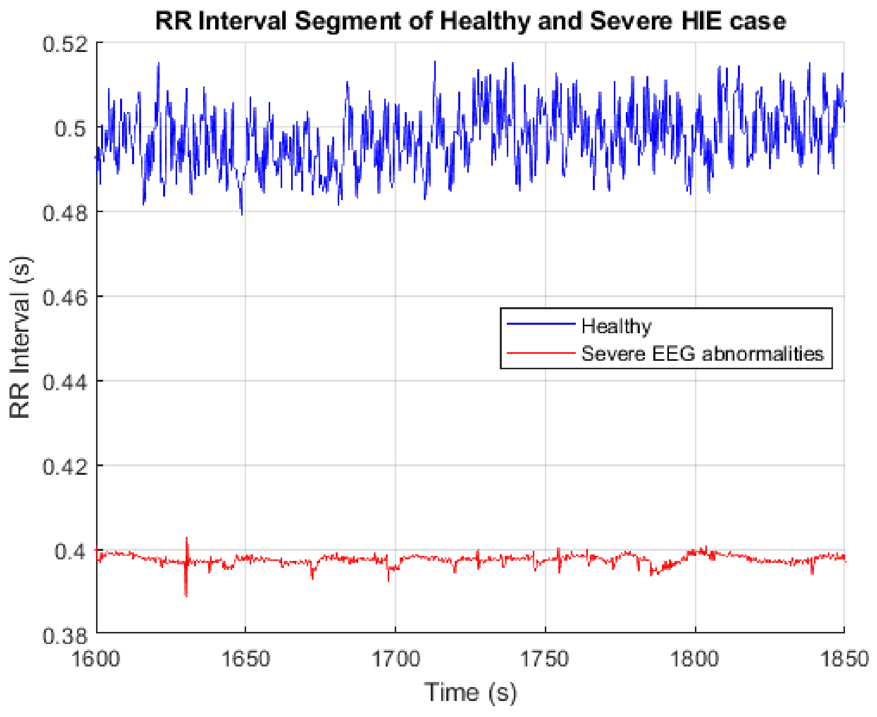


Figure 1: RR intervals of a healthy neonate and a neonate with severe EEG HIE grade.

The interval between successive heartbeats is often referred to as the RR interval, arising from the fact that the individual heartbeats on an ECG are termed the R peaks. It is obvious that the RR interval of the healthy neonate is more erratic and has greater fluctuations than the more suppressed profile of the neonate with HIE, demonstrating the correlation between the severity of HIE and a depressed heart rate variability. The difference between healthy and mild or moderate HIE severity grades isn’t always as visually striking, however.

To this end, a number of attempts have been made to develop a system that can accurately classify the severity of HIE using the heart rate variability of the neonate. Such a system would allow treatment to be applied rapidly in situations where state of the art equipment (such as cEEG or aEEG), or the expertise required to use them, are not available.

Matić et al reinforced the results produced by R.M. Goulding et al, finding a significant decrease in HRV in newborns with a severe grade of HIE [12]. The dataset used in this study consisted of 36 2-hour ECG segments from 19 neonates, recorded 18 – 48 hours postpartum. The severity of HIE was assessed using the Sarnat Scoring System and the background cEEG. The newborns were assigned one of three grades ranging from mild to severe. There was a large overlap in the HRV features extracted from the ECG with those employed by R.M. Goulding et al, however some unique features were used in addition. A comprehensive search was carried out to determine the best subset of the features to use as inputs to a Linear Discriminant Analysis (LDA) classifier. The best performing LDA classifier achieved an accuracy of 80%. While the results are encouraging, the study was somewhat limited in its approach. ECG recordings from a larger group of newborns would give a more accurate indication of the performance of this system. As well as this, the ECG recordings were not taken within 6 hours of birth, when treatment via therapeutic hypothermia is expected to have the greatest impact. Finally, the LDA classifier used is a somewhat simplistic approach. A more complex classifier could potentially yield a greater performance.

R. Ahmed took a sophisticated approach to classifying the severity of HIE using HRV [13]. The dataset was composed of 48 1-hour ECG recordings. The severity of HIE for each neonate was assessed using cEEG that was recorded in the same timeframe as the ECG recording. Each neonate was assigned one of five grades varying between normal (healthy) and severe. The approach taken in this study sought to explore the evolution over time of the HRV of the neonate and to utilize this information in a classifier system. Time domain, frequency domain and geometric HRV features were extracted from segmented epochs of the heart rate signal and were used to generate a statistical model, with significant overlap in the features used by R.M. Goulding et al and Matić et al [11, 12] . A supervector, representing sequences of HRV features extracted from sequential segments of the heart rate signal, was used as an input to a Support Vector Machine classifier. The parameters of the statistical model were varied along with the epoch segment length and the subset of features used. The best performing combination of parameters achieved an accuracy of 81.6%, slightly more than that achieved by Matić et al. Despite the more sophisticated approach in the classification system used, this study still had some limitations. It was noted by the author that the ECG recordings used in the dataset were of poor quality, with 6 recordings from the original dataset being discarded due to the presence of artefacts. These artefacts corrupt the signal, making it impossible to accurately estimate the inter-heartbeat intervals. Better quality ECG recordings from a larger cohort of neonates has the potential to improve the results demonstrated in this study. Furthermore, it was noted by the author that a coarse approach to feature selection was taken to determine the subset of features that would yield the best performance. A more robust feature selection method could further improve the results of the study.

It is the aim of this project to develop a subject independent system to automatically classify the severity of HIE using the heart rate variability of the neonate. A dataset consisting of ECG recordings of 120 neonates has been collected for this purpose. The ECG for each neonate was recorded within the 6-hour window after birth in which the efficacy of treatment using therapeutic hypothermia is maximized. The severity of HIE was assessed using the background cEEG recorded in the same timeframe. These grades correspond to the EEG grading system used in previous studies [13], with five grades ranging from normal (healthy) to severe. It is hoped that this large dataset with high quality ECG recordings will overcome some of the limitations of previous studies in this area. It is also hoped that the use of sophisticated classification models and a robust approach to model selection will yield a performance increase and overcome limitations in the previously mentioned studies. In this report, a background in the biomedical signal processing techniques used to process the raw ECG data is given in Chapter 2, before details of the quantitative HRV features that characterise the heart rate variability of the neonate are outlined. A background in the classic feature-based machine learning techniques is given at the start of Chapter 3, before the results of these models are presented. Chapter 4 offers an explanation of the deep learning methods employed in this project, as well as details of the best performing architectures and the results that were achieved with them. The report is concluded in Chapter 5, which offers a discussion of the results and limitations of the project, as well as suggestions for future work in this area.

Chapter 2: Biomedical Signal Processing

In order to develop a system that can automatically classify the severity of HIE using the heart rate variability of the neonate, it is necessary to extract quantitative features that characterise the fluctuations in the intervals between successive heartbeats. This chapter briefly outlines some information on the dataset collected for this project before detailing the methods used to process the raw ECG recordings from each new-born, as well as presenting an analysis of the quantitative features that measure the babies' heart-rate variability.

2.1. Dataset

As mentioned in Chapter 1, the dataset used in this project consists of approximately 1-hour ECG recordings from 120 neonates collected in a 6-hour window after birth. Continuous EEG (cEEG) was recorded within the same window. The cEEG recording for each newborn was used to assess the severity of HIE. A single severity grade was assigned to each neonate for the entire EEG and ECG recordings, corresponding to the grades defined by Murray et al [7]. The EEG grade descriptions are summarized in Table 1. In total, there were 13 neonates with normal EEG grade, 41 with mild EEG grade, 23 with moderate EEG grade, 22 with major EEG grade and 21 with severe/inactive EEG grade.

Grade	Description
0	Normal/Healthy
1	Mild abnormalities
2	Moderate abnormalities
3	Major abnormalities
4	Severe abnormalities/Inactive

Table 1: EEG grades denoting severity of HIE injury.

For this project, the problem was simplified to a binary classification problem: the 0 and 1 severity grades were grouped into their own class 'Normal/Mild' and relabeled as -1. The severity grades 2, 3 and 4 were also grouped into their own class 'Moderate/Major/Severe' and relabeled as 1. This particular threshold is used as treatment via therapeutic hypothermia is usually administered when the HIE injury evolves to moderate severity [7].

The following section outlines the details of the Pan-Tompkins algorithm which was used to process the ECG data collected for this study.

2.2. Pan-Tompkins Algorithm for QRS Detection

A typical electrocardiograph (ECG) signal can be broken down into three sections: The P wave, corresponding to the depolarization of the atria of the heart, the QRS complex, corresponding to the rapid depolarization of the ventricles of the heart, and the T wave, corresponding to repolarization of the ventricles. This is illustrated in Figure 2. Note that the amplitude of the U wave is usually so small that it is ignored.

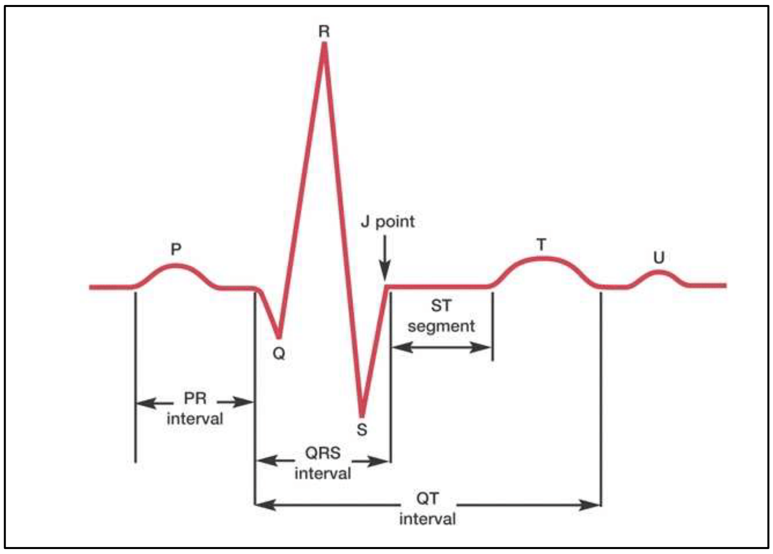


Figure 2: Typical ECG of a heart with a normal cardiac rhythm [14].

The Pan-Tompkins algorithm is designed to automatically detect the temporal locations of the QRS complexes of the ECG signal [15]. In brief, the algorithm works as follows. The ECG is first filtered by a bandpass filter consisting of a cascaded low pass and high pass filter to remove any noise that may have been picked up from the environment (e.g. muscle noise). The passband was adjusted to 4 – 30 Hz for this project to reflect the difference in the typical heart rate of a new-born compared to that of an adult. The filtered signal is then differentiated to obtain information about the slopes, before being squared. A moving average window is applied to the output of the squaring filter. An adaptive threshold is then applied to the resulting signal to detect the R peaks. The adaptive thresholds are continuously monitored and adjusted based on the peak signal and noise values. The use of two adaptive thresholds enables the algorithm to find any (initially) missed R peaks, by using the lower threshold to detect potential candidates for the R peak when one has not been found by the upper threshold after a period corresponding to 1.66 times the average interval between R peaks. Figure 3 shows an example of the Pan-Tompkins algorithm applied to an ECG signal, with the output of each stage shown.

The temporal locations of the QRS complexes from the Pan-Tompkins algorithm allow the interval between successive R-peaks to be calculated, termed the RR interval. This corresponds to the interval between successive heartbeats, as stated previously.

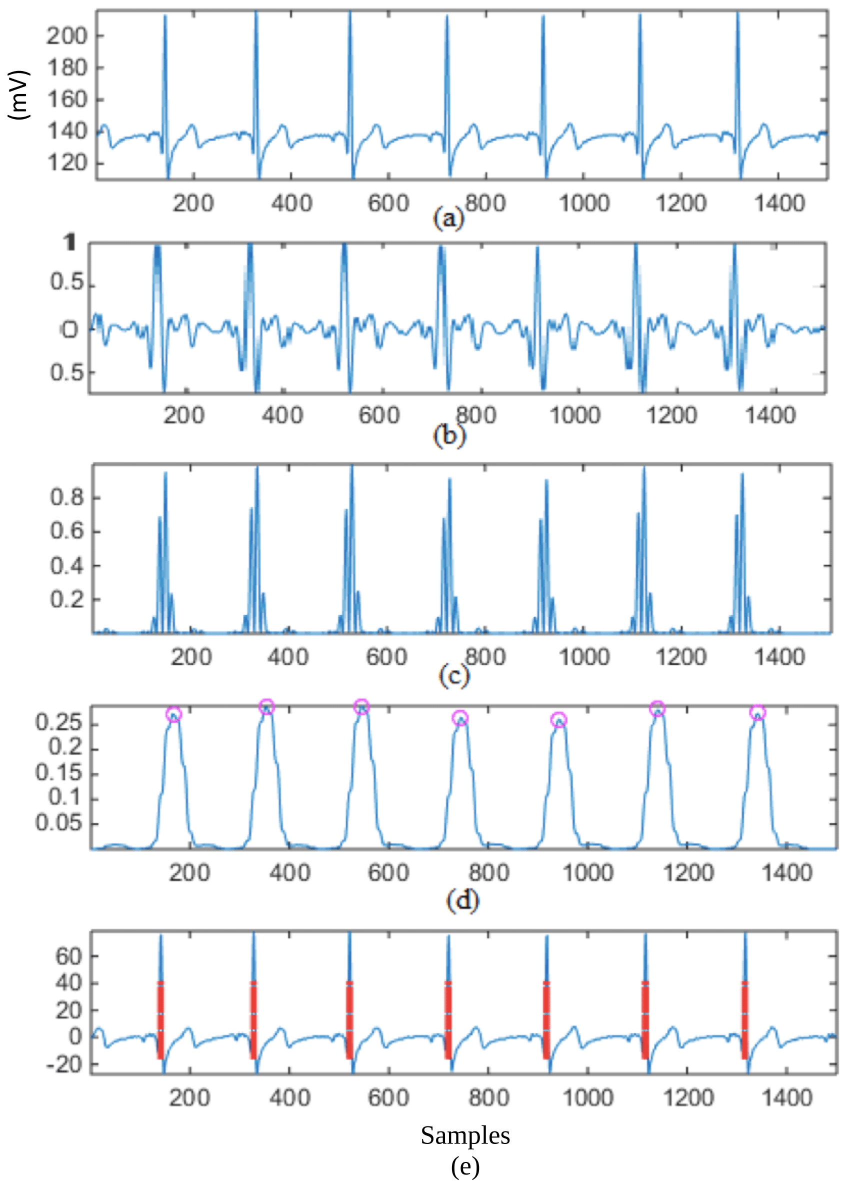


Figure 3: Example of Pan-Tompkins Algorithm applied to the ECG signal from an adult [16], where (a) shows the raw ECG signal, (b) shows the signal after the band-pass filter and differentiation, (c) shows the signal after squaring, (d) shows the output of the moving average filter and the following adaptive thresholding and (e) shows the timestamped locations of the QRS complex in the original ECG signal.

The raw ECG recordings from the NICU were inspected by the INFANT team and any artefacts were removed. Artefacts in the ECG can occur for several reasons, such as the electrodes becoming disconnected. The corrected ECG signal was then processed using the Pan-Tompkins algorithm as described above, and any missing segments in the resulting RR interval data (due to the previously mentioned artefacts) were annotated with NaNs (Not A Number). The irregularly sampled RR interval data had to be uniformly resampled to 256 Hz before certain features could be extracted (e.g. the features in the frequency domain). The resampled RR intervals are termed the NN intervals. As shown in the next section, both the RR interval and the NN interval data can be used to extract a number of features in the time and frequency domain that characterize the neonate's heart rate variability.

2.3. Heart Rate Variability Features

A total of fourteen features were extracted from the RR interval data. The extracted features quantify the heart rate variability of the neonate and include the seven features used by R.M. Goulding et al and the twelve used by R. Ahmed [11, 13]. The RR intervals must first be segmented into the desired epoch lengths, so that the features can be calculated on a per-epoch basis. This is consistent with the approach taken by other studies in this area [11, 13]. The segmented RR intervals are then uniformly resampled to produce the NN intervals. The time domain features correspond directly to the variability of the heart rate, while the frequency domain features measure the impact of the injury on the autonomic nervous system.

2.3.1. Time Domain Features

The time domain features include various statistics of the NN intervals:

1. Mean NN interval (mean_NN), the average NN interval for a given epoch.
2. Standard deviation of the NN interval (SD_NN), which quantifies the variability in the NN intervals for a given epoch.
3. Skewness of the NN interval distribution, which measures how asymmetric the NN interval distribution is.
4. Kurtosis of the NN interval distribution, which is a measure of how Gaussian the NN interval distribution is.
5. Approximate Entropy (ApEn), which characterises the randomness of the NN intervals, where a greater approximate entropy corresponds to greater randomness.
6. Root Mean Squared Successive Difference (RMSSD) of the RR intervals, which represents the short-term or high frequency variability in the HRV signal.
7. Ratio of SD_NN to RMSSD (SD_NN_RMSSD), which measures the balance between the total variability in an epoch and the short-term or high frequency variability.

Various geometric time domain features are also used to describe the HRV:

8. Triangular Interpolation of the NN interval histogram (TINN) measures the width of the base of the triangle which best approximates the histogram of NN intervals for a given epoch [17]. This is illustrated in Figure 4. It can be seen that for a severe grade of HIE, the histogram of NN intervals would be much narrower than for a healthy baby, and so TINN would be smaller.

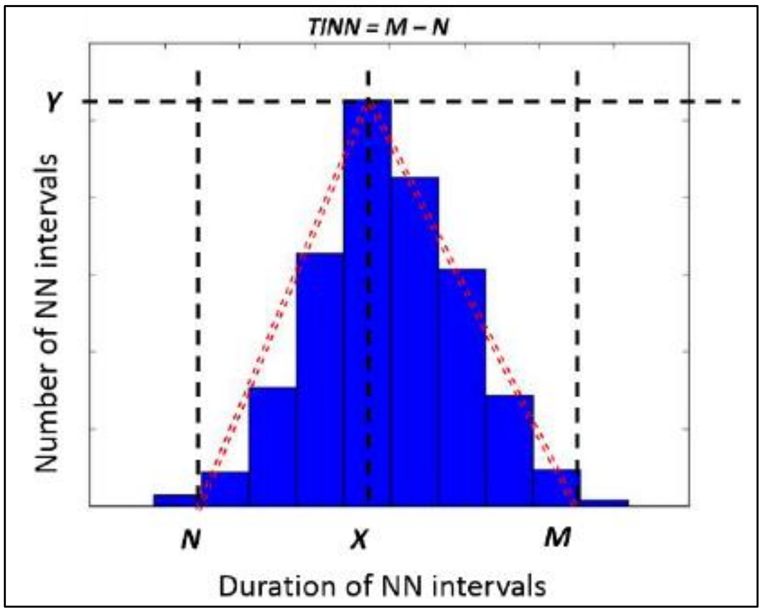


Figure 4: Illustration of the TINN feature. X is the bin with largest number of NN intervals, while N and M correspond to the base of the interpolating triangle [18].

9. First and second derivatives of the standard deviation (SD1_Poincare and SD2_Poincare) are generated from the Poincaré plot of the NN intervals, which plots the NN interval against the successive NN interval value. The Poincaré plot for a healthy baby and a baby with severe HIE injury is shown in Figure 5.

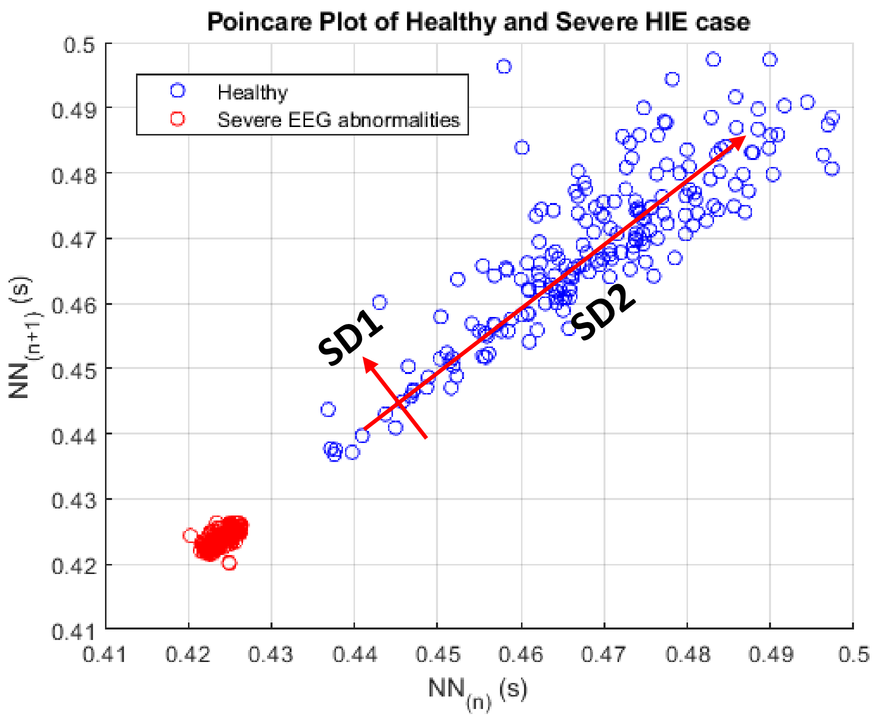


Figure 5: Poincaré plot for a healthy and encephalopathic neonate, showing the first and second derivatives of the standard deviation for the healthy baby.

2.3.2 Frequency Domain Features

As mentioned previously, the irregularly sampled RR intervals had to be uniformly resampled to 256 Hz using piecewise cubic hermit splines before the Fast Fourier Transform (FFT) could be computed. The mean power in various frequency bands is then extracted from the FFT. The features extracted from the frequency domain characterise the response of the autonomic nervous system of the neonate:

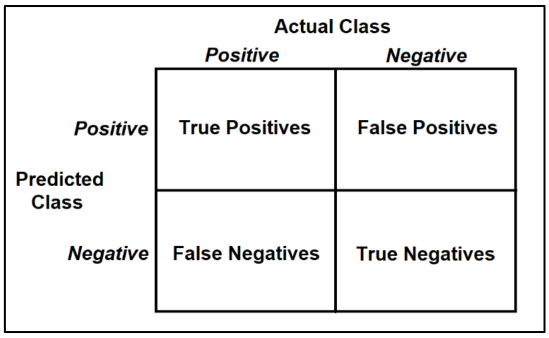
10. Power in the Very Low Frequency band (VLF_power), which measures the power spectral density in the 0.008 – 0.04 Hz band and is thought to correspond to variations in the thermoregulation of the body [19].
11. Power in the Low Frequency band (LF_power), which measures the power spectral density in the 0.04 – 0.2 Hz band and corresponds to the sympathetic component of the autonomic nervous system [20].
12. Power in the High Frequency band (HF_power), which measures the power spectral density in the 0.2 – 2 Hz band and corresponds to the parasympathetic component of the autonomic nervous system [21].
13. The ratio of the LF and HF power (LF_HF_ratio), which measures the balance between the sympathetic and parasympathetic components of the autonomic nervous system.

All fourteen of the HRV features showed promising results in previous studies carried out [11-13]. The remainder of this chapter details an analysis of the discriminative power of the individual features to assess their ability in classifying the severity of HIE in newborns.

2.4. Receiver Operating Characteristic and Area Under Curve

The Area Under the Receiver Operating Characteristic (ROC) Curve is used in this chapter to quantify the discriminative power of the fourteen individual HRV features – i.e. to measure how well a single feature can distinguish between the two classes of HIE severity grades (‘Healthy/Mild’ and ‘Moderate/Major/Severe’). The ROC curve and the corresponding Area Under the Curve (AUC) are widely used metrics for assessing the performance of machine learning classifiers and are also used in Chapters 3 and 4 [22]. This section will discuss the use of the ROC and AUC with respect to machine learning classifiers while simultaneously giving context to the results of the univariate feature analysis in Section 2.5.

The ROC curve is derived from the confusion matrix of a binary classifier, which is a matrix of predictions made by a classifier on a given dataset [22]. An example of a confusion matrix is shown in Figure 6. The true positives (TP) are the number of positive instances correctly identified by the classifier, the false positives (FP) are the number of negative instances incorrectly classified as positive, the false negatives (FN) are the number of positive instances incorrectly classified as negative and the true negatives (TN) are the number of negative instances correctly identified by the classifier.



		Actual Class	
		Positive	*Negative*
Predicted Class	*Positive*	True Positives	False Positives
	Negative	False Negatives	True Negatives

Figure 6: Confusion matrix for a binary classifier.

The ROC curve plots the true positive rate (TPR, the ratio of correctly identified positive classes) of the classifier on the y-axis against the false positive rate (FPR, the ratio of incorrectly classified negative classes) on the x-axis. A classifier generally assigns a score or probability to each instance that it makes a prediction on. This instance is assigned to the positive class if its score or probability is above a certain threshold, or the negative class otherwise. A sequence of points for a given classifier can be generated by varying the classifier's decision threshold and calculating the TPR and FPR for each value of the threshold. Plotting these points yields the ROC curve. An example of an ROC curve and the corresponding distribution of probability scores for some classifier is shown in Figure 7.

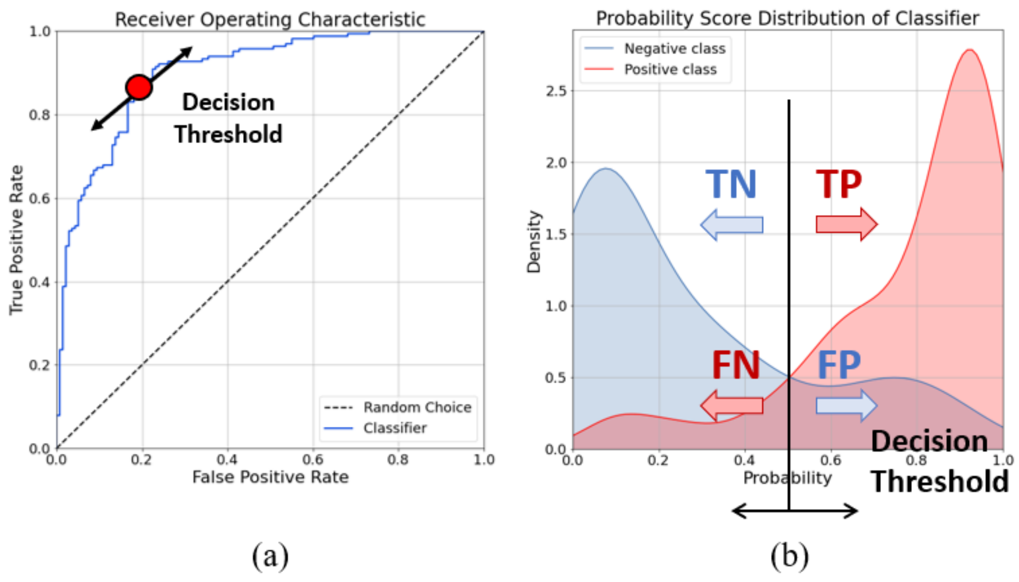


Figure 7: ROC curve and probability score distribution for a given classifier on some dataset, where (a) shows the ROC curve and (b) shows the distribution of scores - TP is the true positives, FP is the false positives, TN is the true negatives and FN is the false negatives. The classifier shown here has an AUC of 0.90.

The diagonal on the ROC represents the classifier that randomly selects the class for a given instance – that is, it does not use any information from the data to make a prediction. A classifier that passes through the point (0,1) represents a perfect classifier that makes no mistakes. The distribution of probability scores illustrates how the number of true positives and false positives change as the decision threshold is varied.

The Area Under the ROC curve (AUC) is a performance metric that concisely summarizes the information in the ROC curve. The AUC measures the probability that a randomly selected positive instance from the dataset will have a higher score or probability assigned by the classifier than a randomly selected negative instance [22] . It essentially measures the degree of separability between the positive and negative class score distributions of the classifier, such as that shown in Figure 7 (b). The perfect classifier that passes through the point (0,1) will have an AUC of 1.0. A perfectly random classifier will have an AUC of 0.5, corresponding to the area under the diagonal of Figure 7 (a). A classifier that always classifies the positive class as negative and the negative class as positive has an AUC of 0.0.

The ROC curve can also be plotted for a feature by varying a decision threshold along the sequence of values for that feature, thus generating a sequence of points for the TPR and FPR. The AUC can then be calculated from the ROC. An AUC of 1.0 means the positive and negative class are totally separable using that feature. An AUC of 0.5 means the classes are totally overlapped and using this feature to distinguish between the classes is no better than random guessing. It generally doesn't matter if the AUC for a particular feature is less than 0.5 when carrying out univariate feature analysis over the whole dataset, as it's the amount of the separability that matters. As a result, any feature with an AUC less than 0.5 is subtracted from 1.0 – thus, all feature AUCs are on the same scale from 0.5 (feature is as good as random guessing) to 1.0 (feature can be used to perfectly distinguish between the positive and negative class).

2.5. Feature Extraction and Univariate Feature Analysis

In our initial approach to the problem, only seven HRV features were extracted from the data: mean_NN, SD_NN, TINN, VLF_power, LF_power, HF_power and LF_HF_ratio. This was done using a MATLAB script provided by a member of the INFANT team. As mentioned previously, the RR interval data for each baby had to be segmented into epochs before the HRV features could be extracted. A 5-minute feature extraction window with 0% overlap was selected, as this corresponds to the window length and overlap used by R.M. Goulding et al [11]. The MATLAB script was designed to handle any NaNs in the RR interval data that may be present due to artefacts in the original ECG recording, and an epoch was discarded if more than 30% of the data consisted of NaNs. The AUCs of the initial seven HRV features are shown in Table 2, which indicate how well the features are able to discriminate between the two classes of HIE severity grades, 'Normal/Mild' and 'Moderate/Major/Severe'. As mentioned in Section 2.4, features that had an AUC less than 0.5 were subtracted from 1.0 to ensure the feature AUCs remained in the range from 0.5 to 1.0.

Feature	AUC
mean_NN	0.67
SD_NN	0.51
VLF_power	0.54
LF_power	0.52
HF_power	0.57
LF_HF_ratio	0.67
TINN	0.60

Table 2: AUCs of original seven HRV features.

As discussed later in Chapter 3, these seven HRV features alone were not discriminative enough to accurately distinguish between HIE severity grades, and so it was necessary to expand the feature set to the full fourteen HRV features outlined earlier. The code for extracting the additional seven HRV features was adapted from a MATLAB script written by a former PhD student of the INFANT centre. The adapted PhD student's code was incorporated into the original MATLAB script used to calculate the initial seven HRV features. An important change was also made to the feature extraction code: the NN intervals were normalised with respect to the mean_NN for that epoch when extracting the other thirteen HRV features. This was included after a study asserted that heart rate variability is significantly coupled with the average heart rate (whether the HRV is being calculated from the heart rate measured in bpm or the RR intervals) [23]. This coupling is caused by both physiological and mathematical factors and can be removed by normalising with respect to the mean RR interval (or mean heart rate in bpm if that method is being used). Indeed, the discriminative power of the original HRV features improved after using this method, which is discussed shortly. A 5-minute epoch with 0% overlap was used again when extracting the fourteen HRV features.

Once the fourteen HRV features were generated, an analysis of the discriminative power of the features was carried out. The kernel density estimates of the fourteen features were plotted (the kernel density estimate is an approximation to the probability density function of a random variable, or in this case, a feature). The KDE plots allow the separation between the classes to be visualized, giving a quick indication of the discriminative ability of the individual features. The resulting KDE plots are shown in Figure 8. It's clear that there is significant overlap present for the majority of the features, with the mean_NN showing the greatest separation between the classes. It can be seen in the KDEs that the positive class ('Moderate/Major/Severe') was lower than the negative class ('Normal/Mild') for many of the HRV features, which makes sense, as it was outlined in Chapter 1 that a depressed HRV is correlated with severe grades of HIE.

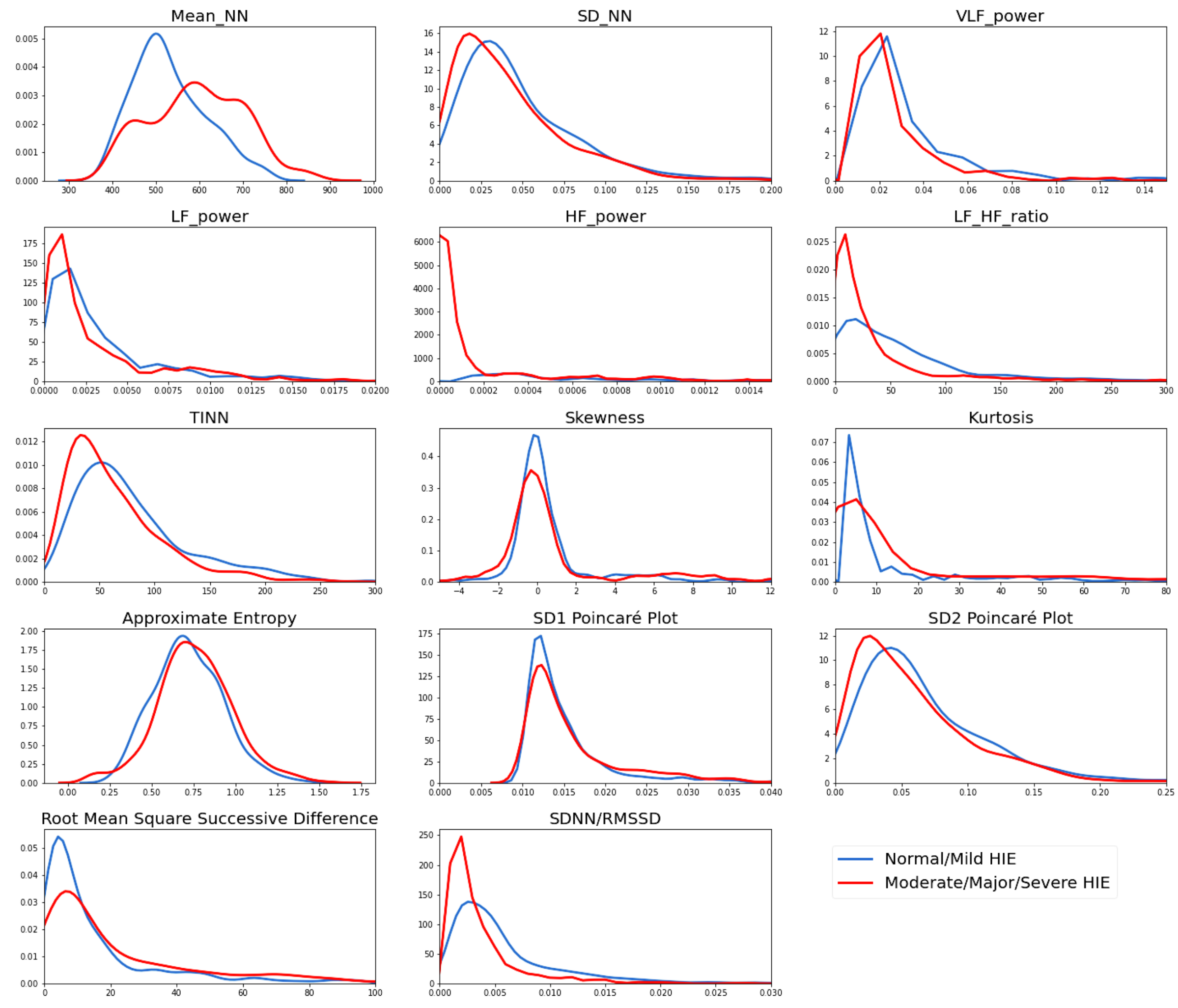


Figure 8: Kernel Density Estimate plots of fourteen HRV features using a 5-minute epoch and 0% overlap. Note that the x-axis was limited to remove the long tails of the distributions.

The AUC was calculated for each feature to quantify this separability or discriminative power, the results of which are shown in Table 3. It's clear that some of these features are better than originally thought based on the KDEs. Skewness (0.53), SD1_Poincare (0.54) and HF_power (0.54) were the worst performing features, with a discriminative power only slightly better than random guessing. The best performing features were SD_NN_RMSSD (0.70), mean_NN (0.67), LF_HF_ratio (0.67) and TINN (0.60). The benefit of normalising the NN interval data with respect to the mean_NN for each epoch is easily inferred by comparing these results with the AUCs of the original seven HRV features that didn't use normalisation in Table 2. The AUCs for the SD_NN, VLF_power and LF_power were all much closer to 0.5 (i.e. random guessing) before normalisation was introduced.

Feature	AUC	Feature	AUC
mean_NN	0.67	Skewness	0.53
SD_NN	0.57	Kurtosis	0.58
VLF_power	0.59	ApEn	0.56
LF_power	0.56	SD1_Poincare	0.54
HF_power	0.54	SD2_Poincare	0.57
LF_HF_ratio	0.67	RMSSD	0.59
TINN	0.60	SD_NN_RMSSD	0.70

Table 3: AUCs of expanded fourteen HRV features.

The results from the AUCs for each feature closely aligned with the univariate feature analysis carried out by Daragh using the ANOVA test.

The F-test or ANOVA ('Analysis of Variance') test can be used to compare the means of two class distributions, essentially giving a measure of how separable the class distributions are for each particular feature [24]. This is extremely similar to the AUC of each feature, which also measured the separability of each class. Unlike the AUC, the F-test can be used for the multiclass case (for example, if it was decided to detect one of the five HIE severity grades rather than lumping them together for a binary classification problem). The F-value for each feature x_i can be calculated using Equation (1).

$$F(x_i) = \frac{\frac{1}{K-1}\sum_{C=1}^{K} N_C\left(\overline{x_i}^C - \overline{x_i}\right)^2}{\frac{1}{N-K}\sum_{C=1}^{K}\sum_{k=1}^{N_C}\left(x_i^C(k) - \overline{x_i}^C\right)^2} \tag{1}$$

Where K is the number of classes, N is the total number of samples for the feature x_i, and N_C is the number of samples of feature x_i corresponding to class C.

The resulting F-values for the fourteen HRV features are shown in Table 4. The four best features according to the F-value are the mean_NN (156.42), SD_NN_RMSSD (57.94), LF_HF_ratio (39.44) and TINN (32.17). These correspond to the four best features as measured by the AUC.

Feature	F-value	Feature	F-value
mean_NN	156.42	Skewness	1.73
SD_NN	11.30	Kurtosis	29.58
VLF_power	2.44	ApEn	13.17
LF_power	1.51	SD1_Poincare	7.53
HF_power	2.72	SD2_Poincare	12.17
LF_HF_ratio	39.44	RMSSD	21.94
TINN	32.17	SD_NN_RMSSD	57.94

Table 4: F-values between HRV features and target labels.

2.6. Conclusion

This chapter briefly introduced the dataset before outlining the Pan-Tompkins algorithm which was used to extract the temporal locations of the QRS complexes from the ECG recordings, from which the RR intervals could be calculated. A background was also given on the fourteen HRV features that were extracted from the resulting RR interval data. A univariate analysis was carried out on these features, and it showed that they had limited discriminative power in distinguishing between HIE grades, with the best feature (SD_NN_RMSSD) having an AUC of 0.70. These results were corroborated by Daragh's univariate feature analysis using the ANOVA test.

The univariate feature analysis confirmed that there was no single feature good enough to perfectly distinguish between the severity grades (i.e. no feature had an AUC close to 1.0). This necessitates the use of machine learning, which can combine the predictive power of all the HRV features together to identify complex relationships in the data and hence distinguish between HIE severity grades better than any of the features individually.

Chapter 3: Machine Learning

As shown in the last chapter, the quantitative features that characterise the heart rate variability of the neonate offered only limited discriminative power when considered individually. This warrants the use of a machine learning classifier. This chapter starts out by introducing some of the fundamental concepts of machine learning before covering the background theory behind the classification models used in this project. The initial work carried out on the project using only seven HRV features is then outlined, followed by a description of the robust performance evaluation framework developed and the results of various classification models that were tested for this application.

3.1. Introduction to Machine Learning

Machine learning has been defined as the field of study that enables computers to learn without being explicitly programmed [25]. It can broadly be broken down into two categories: supervised and unsupervised learning. In supervised learning, an algorithm is given example input data and the corresponding outputs, or labels, for this example data. The supervised learning algorithm attempts to 'learn' the relationship that maps the example input data to their corresponding output. In unsupervised learning, an algorithm is given example input data and no outputs or labels, and the goal is to find any structure or relationships that may exist in the data (e.g. clustering algorithms).

Supervised learning tasks can be broken down further into two categories: regression and classification. Regression involves predicting some continuous numerical value based on some input data (e.g. predicting stock prices), whereas classification involves assigning the input data to one of a limited number of discrete classes (e.g. classifying pictures of animals as either 'dog' or 'cat'). Classification problems involving only two classes are known as binary classification problems, and problems with more than two classes are known as multi-class classification problems.

Within classification, there are two main types of classifiers. Generative classifiers attempt to model the joint probability $p(x, y)$ of the inputs x and output y, making predictions using Bayes Theorem by calculating $p(y|x)$ and selecting the output with the highest probability (e.g. Naïve Bayes classifier) [26]. Discriminative classifiers attempt to learn $p(y|x)$ directly and can be thought of as learning the decision boundary that best separates the classes in the feature space (e.g. Support Vector Machine classifier) [26].

For most supervised classification problems, the dataset must be segmented into a 'training set', a 'validation set', and a 'test set'. The training set is used by the machine learning model to adjust its parameters and learn the relationships between the input data and the output data. For generative classifiers, training can be thought of as constructing the model $p(x, y)$ from the training set. For discriminative classifiers, training usually involves minimising some cost or objective function which measures how well the model ‘fits’ or separates the data. It is possible for the model to ‘overfit’ the training data, meaning the model is essentially memorising the training data and minimising the objective function, but will generalise very poorly to unseen data. The validation set is used to monitor the trade-off between how well the model fits the training set and how well it generalises to unseen data during training. The validation set is also used to adjust the model hyperparameters – these are user-adjustable parameters that are used to control the learning process and are distinct from the model’s parameters which are learned automatically during training. The hyperparameters are vital for tuning the model performance and regularising the model to prevent overfitting. The test set is used as a final evaluation of the model which had the greatest performance on the validation set.

3.2. Support Vector Machines

The Support Vector Machine (SVM) is a popular machine learning model that can be applied to a wide range of problems. The model can be used for both supervised regression and classification tasks, as well as unsupervised learning tasks such as clustering [27]. Both linear and non-linear SVM classifiers were evaluated in this study to assess their performance in grading the severity of HIE in neonates.

The primary goal in training the supervised SVM classifier is to map the input training data to a high-dimensional feature space and separate the binary classes by fitting the linear hyperplane with the largest margin between the two classes [28]. The width of this margin is determined by the ‘support vectors’, the data instances in each binary class that are ‘closest’ to instances of the other class. In practice, it is not always possible to ensure that all data instances are kept on or outside the margin boundary, thus it is necessary to define a slack variable which measures the amount by which each data instance can violate the margin boundary. This goal of maximizing the margin of the decision function hyperplane while minimizing the number of margin violations can be expressed mathematically as shown in Equations (2) and (3).

$$minimize_{(\boldsymbol{w},b,\zeta)} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{m}\zeta^{(\mathrm{i})} \tag{2}$$

$$subject\ to\ y^{(i)}\left(\boldsymbol{w}^T\boldsymbol{x}^{(\boldsymbol{i})} + b\right) \geq 1 - \zeta^{(\mathrm{i})}\ and\ \zeta^{(\mathrm{i})} \geq 0\ for\ i = 1,2,\dots,m \tag{3}$$

Where $\boldsymbol{w}$ is the vector of weights of the hyperplane decision function, C is the degree of regularization which controls the amount of margin violations, $y^{(i)} \in \{-1,1\}$ is the label for the i^{th} training instance, $\boldsymbol{x}^{(i)}$ is the i^{th} training instance, b is the hyperplane decision function bias term and $\zeta^{(\mathrm{i})}$ is the slack variable for the i^{th} training instance.

For this convex constrained optimization problem, it is possible to express the dual form of the problem using the Lagrange Multipliers method, which allows the linear SVM objective to be minimized by computing the optimal Lagrange multipliers $\widehat{\alpha}^{(\mathrm{i})}\ for\ i = 1,\dots,m$. The dual form of the linear SVM objective is shown in Equation (4).

$$minimize_{(\alpha)} \ \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(j)} - \sum_{i=1}^{m}\alpha^{(i)} \tag{4}$$

$$subject\ to\ \alpha^{(j)} \geq 0\ for\ i = 1,2,\ldots,m$$

As it turns out, $\hat{\alpha}^{(i)} = 0$ for all training instances besides the support vectors which determine the width of the margin of the decision function hyperplane. The optimal Lagrange multipliers $\hat{\alpha}^{(i)}$ can then be used to determine the optimal weights $\boldsymbol{w}$ of the hyperplane decision function and the bias term b. Finally, the decision function of the linear SVM classifier can be expressed as shown in Equation (5).

$$h\left(\boldsymbol{x}^{(n)}\right) = sign\left(\sum_{i=1}^{m}\hat{\alpha}^{(i)}y^{(i)}\boldsymbol{x}^{(i)^T}\boldsymbol{x}^{(n)} + b\right) = sign(\boldsymbol{w}^T\boldsymbol{x}^{(n)} + b) \tag{5}$$

Where $\boldsymbol{x}^{(n)}$ is the data instance to make a prediction on. As $\hat{\alpha}^{(i)} = 0$ for all training instances besides the support vectors, making a prediction only requires the support vectors.

The benefit of using the Lagrange multipliers to express the dual form of the problem is that it allows the ‘kernel trick’ to be carried out. The kernel trick allows the SVM to construct a non-linear decision boundary in the original feature space by projecting the data to a higher dimension without explicitly having to carry out this computationally intensive mapping [28]. If we define the mapping function as Φ, apply this higher dimensional mapping to the input vectors $\boldsymbol{x}$, and fit a linear SVM classifier to the result, Equation (5) becomes:

$$h\left(\boldsymbol{x}^{(n)}\right) = sign\left(\sum_{i=1}^{m}\hat{\alpha}^{(i)}y^{(i)}\Phi\left(\boldsymbol{x}^{(i)}\right)^T\Phi(\boldsymbol{x}^{(n)}) + b\right) \tag{6}$$

The only difference between Equation (5) and Equation (6) is that the dot product of the transformed input vectors replaced the dot product of the original input vectors. In machine learning, a kernel is any function that allows the dot product of $\Phi(\mathbf{a})^{\mathrm{T}}\Phi(\mathbf{b})$ to be computed, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are two vectors and Φ is some mapping to a higher dimension [28] . Shown in Equation (7) is the Gaussian Radial Basis Function (RBF) which is a popular kernel for performing non-linear classification with an SVM.

$$K(\boldsymbol{a},\boldsymbol{b}) = \Phi(\mathbf{a})^{\mathrm{T}}\Phi(\mathbf{b}) = e^{-\gamma\|\boldsymbol{a}-\boldsymbol{b}\|^2} \tag{7}$$

Where γ is a scalar hyperparameter. This kernel function can be used to replace the dot product $\Phi\left(\boldsymbol{x}^{(i)}\right)^T\Phi(\boldsymbol{x}^{(n)})$ in Equation (6). That is, the kernel function can be used to efficiently obtain the exact same result as if you had gone through the computationally intensive process of mapping the input vectors to a higher dimensional space and then fit a linear SVM classifier to the result. Thus, the decision function for the RBF kernel SVM classifier can be expressed as shown in Equation (8).

$$h\left(\boldsymbol{x}^{(n)}\right) = sign\left(\sum_{i=1}^{m}\hat{\alpha}^{(i)}y^{(i)}K\left(\boldsymbol{x}^{(i)},\boldsymbol{x}^{(n)}\right) + b\right) \tag{8}$$

Again, as $\hat{\alpha}^{(i)} = 0$ for all training instances besides the support vectors, making a prediction with the RBF kernel SVM still only requires the support vectors.

3.3. Boosted Decision Trees

In machine learning (in the context of classification), ensemble methods refer to learning algorithms that are composed of multiple classifiers, with the output of the ensemble usually determined by a weighted vote from all the classifiers in the ensemble [29]. Boosting is an ensemble method which involves training a number of classifiers in series, with each classifier correcting the error of its predecessor. It has been shown that a boosted ensemble of weak learners (only slightly better than random guessing) can actually produce a strong, highly accurate learner [30]. Extreme Gradient Boosted Decision Trees (XGBoost) is a boosting algorithm composed of sequential decision trees and was tested for this application.

3.3.1. Decision Trees

Decision trees are one of the most intuitive machine learning classifiers available and have a typical binary if/else structure. An example of a decision tree is shown in Figure 9.

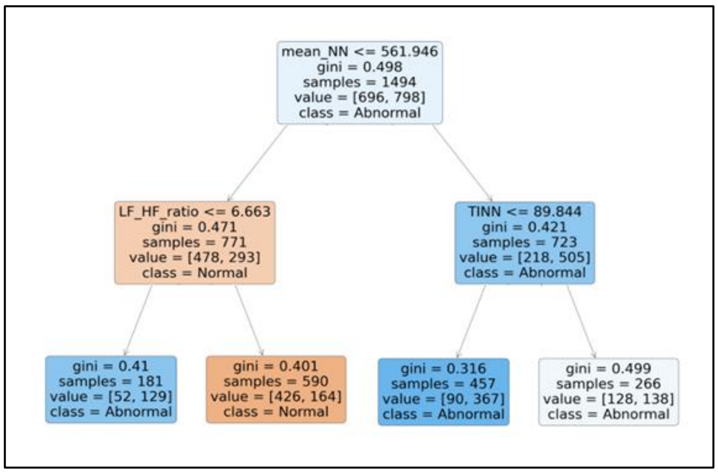


Figure 9: Decision tree of depth 2 with 4 leaf nodes trained using the CART algorithm.

The top of the decision tree is termed the root node, while the outputs of the decision tree (at the bottom) are termed leaf nodes. Binary decision trees are often trained using the CART algorithm, which seeks to minimise the 'Gini impurity' at the output leaf nodes [31]. A node is considered pure if all the training instances it applies to belong to the same class. During training, starting at the root node, the decision tree searches over the feature set to find the feature x_k and the threshold t_k that minimises the impurity at the two child nodes (i.e. it finds the feature and threshold that best splits the two classes in the training set). The tree continues to grow, recursively splitting the training set at each node to further separate the classes. This process continues until the impurity of the leaf (output) nodes is 0 or until the tree reaches some user-defined stopping criterion. When making a prediction on a sample, the decision tree shown in Figure 9 checks if the sample's mean_NN feature is less than or equal to 561.946. If this is true, then the sample moves down the left-hand side of the tree. This process continues until the sample reaches the leaf node, where it is assigned either the class 'Abnormal' or 'Normal'.

An in-depth explanation of the CART algorithm is not offered here, as XGBoost uses a different algorithm when training the trees in the ensemble. Instead, it is hoped that this explanation gives a high-level understanding of a decision tree classifier's structure that will aid understanding of the XGBoost algorithm.

3.3.2. Extreme Gradient Boosted Decision Trees (XGBoost)

Gradient boosting is a powerful boosting method that improves the performance of the overall ensemble by training sequential classifiers on the residual errors of the previous classifiers [32]. XGBoost is an extremely fast and scalable gradient boosting algorithm which uses decision trees as its base classifiers [33] .

Intuitively, the XGBoost algorithm for binary classification can be explained as follows: XGBoost makes an initial prediction on all the training instances in the training set. The predictions made by XGBoost correspond to the probability of the positive class. It then trains a decision tree using the residual errors (i.e. the difference) between the ground truth labels and the initial predicted probability as the target labels. The output of this decision tree for each training instance is added to the initial prediction. The residuals between the new predicted probabilities and the ground truth labels for each instance are computed again, and another decision tree is trained using those results as the target labels. This process is repeated until the residual error is minimised across the training set, or more commonly, until early stopping halts training (early stopping uses an unseen validation data set to estimate the generalisation error of the classifier during training – if this starts increasing, the model is overfitting and training is stopped). A prediction for a single instance is made by summing the outputs of all the decision trees in the ensemble.

Mathematically, the prediction $\hat{y}^{(i)}$ of an XGBoost model composed of K classifiers on an instance $\boldsymbol{x}^{(i)}$ can be expressed as shown in Equation (9), where f_k corresponds to a decision tree q with leaf weights (output values) w.

$$\hat{y}^{(i)} = \sum_{k=1}^{K} f_k\left(\boldsymbol{x}^{(i)}\right) \tag{9}$$

During training, the XGBoost algorithm trains one tree at a time. For the t^{th} decision tree, it seeks to minimise the objective function shown in Equation (10), where n is the number of training instances, l is some differentiable convex loss function (usually the cross-entropy loss function is used for binary classification, which is shown in Appendix A), $\hat{y}_{t-1}^{(i)}$ is the summed predictions of the previous $t-1$ decision trees for the i^{th} instance, $y^{(i)} \in \{0,1\}$ is the ground truth label for the i^{th} instance, $f_t(\boldsymbol{x}^{(i)})$ is the prediction of the current t^{th} decision tree and $\Omega(f_t)$ is a regularisation term which penalises the model to prevent overfitting.

$$L_t = \sum_{i=1}^{n} l\left(y^{(i)}, \hat{y}_{t-1}^{(i)} + f_t\left(\boldsymbol{x}^{(i)}\right)\right) + \Omega(f_t) \tag{10}$$

A second order Taylor series expansion is applied to the loss function l and hence g_i refers to the gradient or first derivative of the loss function l for the i^{th}training instance and h_i refers to the Hessian or second derivative. Dropping the constant term yields a greatly simplified objective function, as shown in Equation (11).

$$\widetilde{L_t} = \sum_{i=1}^{n} \left[g_i f_t\left(\boldsymbol{x}^{(i)}\right) + \frac{1}{2} h_i f_t^2\left(\boldsymbol{x}^{(i)}\right)\right] + \Omega(f_t) \tag{11}$$

Even with this simplification, it is not computationally feasible to evaluate all possible decision tree structures to minimise the objective function for the t^{th} tree. As a result, starting from a single leaf (i.e. the root node), XGBoost searches over the set of features to find the best possible split to separate the two classes – but this does not guarantee the optimal decision tree will eventually be produced, hence it is a greedy algorithm. A function that measures how well a given feature can split the data at a node can be derived from the simplified objective function shown in Equation (11). This measure is called the gain, and is shown in Equation (12), where λ is a regularisation term and g_i and h_i are the first and second derivatives of the loss function for the i^{th} training instance as mentioned previously, I_L is the subset of training instances in the left child node for this split, I_R is the subset of training instances in the right child node for this split, I is the subset of training instances in the parent node for this split, and γ is another regularisation term.

$$L_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i+\lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i+\lambda} - \frac{\left(\sum_{i\in I} g_i\right)^2}{\sum_{i\in I} h_i+\lambda}\right] - \gamma \tag{12}$$

XGBoost recursively partitions the training data at each node, evaluating how well each split separates the two classes in the training set using the gain function shown in Equation (12). It stops training the t^{th} tree once the user specified tree depth parameter is reached.

As well as the presence of various regularisation terms in the gain and objective functions, XGBoost also uses other regularisation techniques. Shrinkage scales the output of each decision tree in the ensemble, reducing the influence of individual trees and allowing new trees to be added to the model [34]. This is called the learning rate hyperparameter 'eta' of the XGBoost model. XGBoost also offers the possibility of column sampling (only allowing the model to evaluate a subset of the features at each node) and row subsampling (allowing each decision tree to train on only a portion of the data) as tuneable hyperparameters. Both of these techniques have shown to improve predictions and reduce the variance or overfitting of machine learning models [35, 36] .

The remainder of this chapter details the performance of these classifiers in distinguishing between the two lumped classes of HIE severity – 'Normal/Mild' and 'Moderate/Major/Severe'.

3.4. Initial Approach to the Problem

As mentioned in Chapter 2, only seven HRV features were extracted from the RR interval data at first. The features were extracted using a 5-minute epoch with 0% overlap. The dataset of 5-minute epochs of HRV features from the 120 newborns was split into a training, validation and test set. The data was randomly shuffled, and 96 neonates were kept for the training set, with 12 neonates assigned to both the validation and test sets. The data was split in such a way as to ensure that none of the epochs of a particular neonate that was in the training set would end up in the validation or test set, or vice versa. This was done to avoid any kind of bias in the data or models.

There was a significant skew present in the distributions of some features. As well as this, the features had totally different scales. The Support Vector Machine (SVM) can struggle with features that have very different scales, with the largest scale features effectively dominating the location of the decision boundary. Thus, the data was (natural) log transformed and standardised to squash the features down to the same scale, approximately -1 to 1. Standardisation involves subtracting the mean from each feature value and dividing by the standard deviation, with the mean and standard deviations calculated over the training set.

A linear SVM classifier using all seven HRV features was evaluated first. The models used in this initial approach were implemented using the scikit-learn package [37] . Cross validation with 5 folds was performed on the training set to determine the best value for the regularisation hyperparameter C. Cross validation involves splitting the training data into 5 folds, training on 4 of the folds and testing on the remaining left-out fold. This process is repeated until each of the 5 folds have been used as a test fold and the mean performance across all folds is then reported. The performance metric evaluated on the test fold at each iteration was the AUC. The following values were tested for the regularisation hyperparameter C: [1, 5, 10, 20, 50, 75, 100]. The value of C that had the greatest mean performance across the 5 folds was C = 1.0, with a mean AUC of 0.755. The linear SVM was then retrained on the whole training set with its hyperparameter C = 1.0.

Wrapper-based feature selection was then carried out using a variation of Recursive Feature Elimination (RFE) with a linear SVM (with hyperparameter C = 1.0, the default value in scikit-learn) in the wrapper. RFE is a feature selection method that uses the weights of each feature from a trained model to determine the rank (or importance) of each feature, iteratively removing the feature with the lowest rank, retraining on the smaller subset of remaining features, and repeating this process until the desired number of features remain [38]. The optimal number of features to keep was not known prior to carrying out feature selection, therefore this variation of RFE used cross validation with 5 folds to determine the optimal number of features to retain. The cross validation seeks only to find the best number of features to retain but doesn't actually identify which features these are, returning the number of features that achieved the highest mean performance across the 5 folds. Normal RFE is then carried out on the entire training set, retaining the number of features that maximised the performance in the cross validation. This run of RFE is used to determine the actual features that are retained. In this case, the cross validation determined that two features maximise the performance of the linear SVM, with a mean AUC of 0.757 across the 5 folds. The following RFE then determined the mean_NN and LF_power to be the best features.

The linear SVM using all seven HRV features and the linear SVM using the two HRV features selected by RFE were then evaluated on the validation and test sets. Both the validation and test set were log transformed and standardised using the means and standard deviations of the features in the training set. The performance of both models is shown in Table 5.

Model	**Linear SVM**		**Linear SVM with RFE**	
Dataset	**Validation Set**	**Test Set**	**Validation Set**	**Test Set**
AUC	0.75	0.68	0.60	0.49
Accuracy	62%	59%	44%	50%

Table 5: Performance of linear SVM models on validation and test sets.

The ROCs of both SVM models computed on both the validation and test set are shown in Figure 10. The ROC curve allows the trade-off between the true positive rate and false positive rate for various decision thresholds to be visualised – this is particularly important in biomedical applications, where you want to ensure that treatment is applied to as many sick patients as possible (true positive rate) while minimising the number of false detections (false positive rate), as too many false positives may cause the system to be ignored by medical staff.

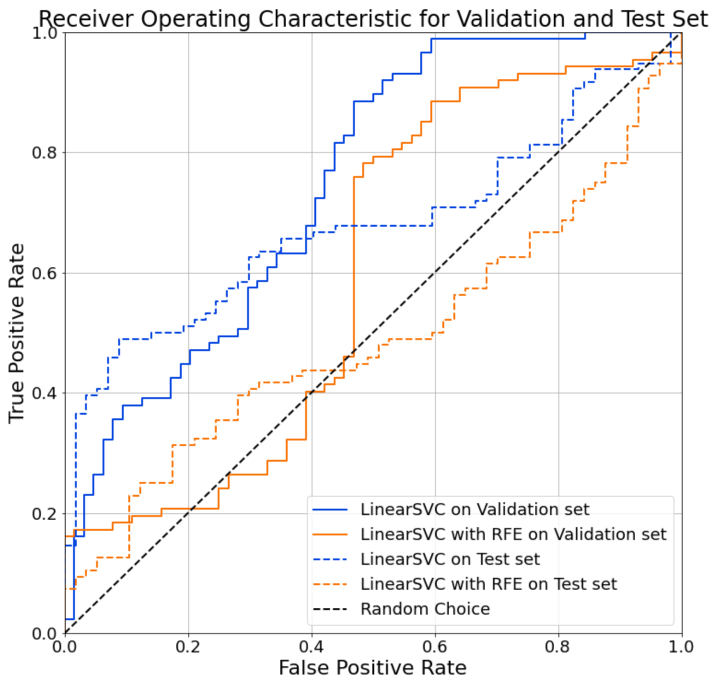


Figure 10: ROCs of both SVM models on validation and test sets.

While the AUC of the SVM with all seven HRV features remained consistent on the validation set, the performance of the SVM using the two features selected by RFE was much worse, with a drop in AUC to 0.6 (from 0.757 on the training set).

The performance of both models dropped when evaluated on the test set, with the SVM using RFE achieving an AUC that's no better than randomly selecting the class of each test instance. The base rate of the test set was 62.7%, meaning the accuracy of the best performing SVM was still worse than simply selecting the positive class for every test instance. After investigating this drop in the performance in the test set, it was determined that the imbalance in classes in the test caused this degradation - in the test set, there were 96 positive data instances compared to 57 negative instances. After inspecting the confusion matrix of the SVM using seven HRV features on the validation set, it was clear that it struggled when presented with a positive class, only guessing correctly 55% of the time. On the other hand, it had an accuracy of 70% when presented with the negative class. This explains the performance drop seen on the test set – since the test set had significantly more positive instances than negative ones (96 vs 57), it's obvious that the performance will decrease when the classifier struggles to correctly classify a positive instance. A similar result was concluded for the SVM using the two features selected by the RFE feature selection routine.

There were a number of key takeaways from this initial approach. It was apparent that the seven HRV features alone were not discriminative enough, and it was essential to expand the feature set to include more discriminative features, hopefully improving the performance of the overall classifiers. This was addressed by expanding the feature set to the full fourteen HRV features outlined in Chapter 2, where it was shown that some of the extra features indeed had better discriminative power (e.g. SD_NN_RMSSD had an AUC of 0.70, higher than any of the original seven HRV features).
It was also observed that the SVM proved to be robust to the presence of noisy or even useless features – the SVM using all seven HRV features had a better performance in both the validation and test set. This was corroborated by the findings of O. Doyle et al, who used a similar approach to seizure detection using HRV parameters. The study stated that the SVM model is particularly robust to the inclusion of irrelevant features, and that the consequences of unknowingly removing good, discriminative features outweighs the potential benefit of removing useless features [39-41]. Thus, it was decided that feature selection is not necessary in this application given the relatively small number of features being used.
There was also considerable variation in results depending on the test set being used – changing the random seed used to shuffle and split the data into the training, validation and test set had a huge impact on results. For example, for one particular split of the data, both SVMs achieved an AUC of 0.82 on the validation set, while the RFE selected six features to retain during cross validation on the training set. This warranted the move to a more robust testing framework rather than a simple 80/20 train/test split, that would test the models across the entire dataset and give a better estimate of their true performance.

Finally, more complex, non-linear classifiers such as RBF kernel SVMs needed to be evaluated for this application, as the simple linear classifiers were not capable of achieving good performance.

3.5. Leave Three Subjects Out

As mentioned in the last section, there was considerable variation in the model’s results depending on the test set used. To overcome this challenge, a new performance evaluation framework was set up: ‘Leave Three Subjects Out’. The code for this performance assessment routine was developed using the NumPy and pandas libraries [42, 43], and is shown in Appendix B.

Leave Three Subjects Out (LTSO) is a subject independent test routine used to estimate the generalisation error of the classifier on unseen data. The dataset of 120 babies is segmented into 40 folds. All the epochs of HRV features for the babies in a particular fold are contained within that fold (i.e. the epochs from the babies do not get mixed into the other 39 folds). At each iteration of LTSO, 117 babies, or 39 folds, are taken and used as the training set. The remaining 3 babies are used as the test set. A nested 5-fold cross validation is carried out on the training set to determine the model hyperparameters that maximise the classifier’s performance across the training set. As outlined earlier, the cross validation segments the training set into 5 folds and trains a model with a given set of hyperparameters on 4 of the folds, testing on the 5th fold. This process is repeated until each of the 5 folds are used as a test fold. The next set of hyperparameters is then tested using the same process, until every combination of hyperparameters has been tested. The set of hyperparameters that had the greatest mean performance across the 5 folds is then selected and used to train the classifier on the full training set of 117 babies. The trained classifier is then tested on the 3 held-out babies. An important distinction to the earlier approach is that the probability of the positive class, or ‘Moderate/Major/Severe’ grade of HIE, is generated at an epoch and subject level. The epoch-level probability is the classifier’s predicted probability of abnormal EEG grade for each 5-minute window. The subject-level probability uses the geometric mean of the predictions for all the 5-minute windows for a particular baby to generate an overall probability of abnormal EEG grade for the entire 1-hour ECG recording. The epoch-level AUC is then generated for the 3 babies in the test set using the epoch-level probabilities.
This process is repeated for each of the 40 folds, such that each fold of 3 babies is eventually used as a test fold. At the end of the evaluation, the mean, standard deviation, median and interquartile range of the epoch-level AUCs across the 40 test folds is reported, as well as an overall subject-level AUC using the subject-level probabilities at each test fold. The most common model hyperparameters selected in the nested 5-fold cross validation across the 40 iterations are also reported. The LTSO performance evaluation routine is illustrated in Figure 11.

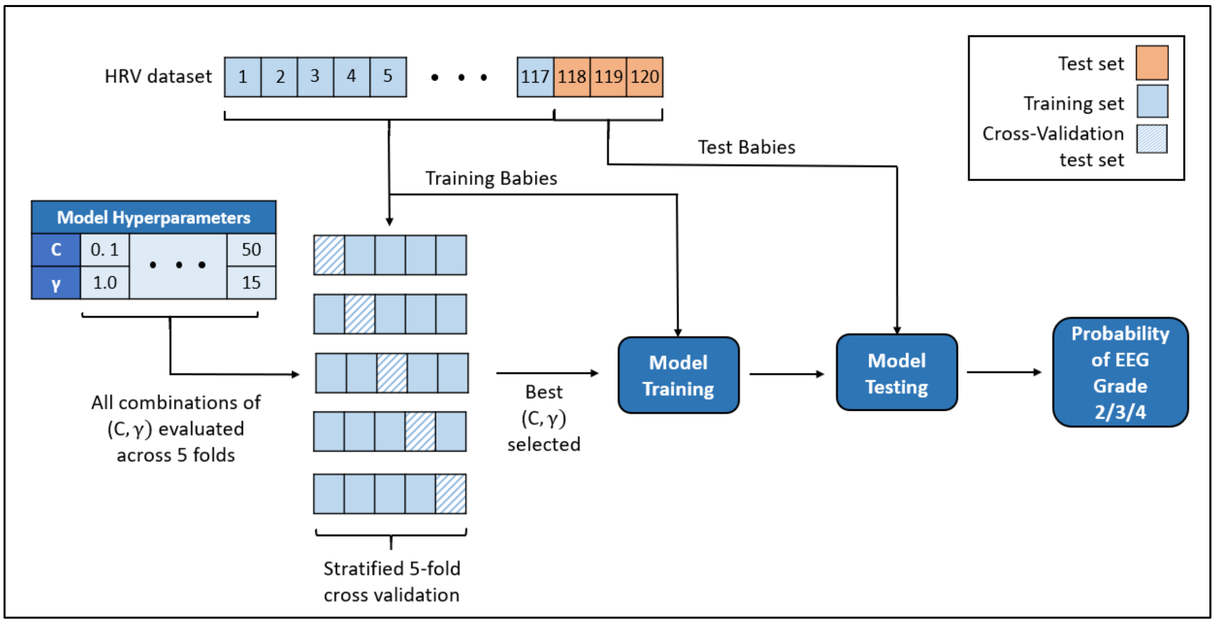


Figure 11: Leave Three Subjects Out performance assessment routine.

Traditionally, a Leave One Subject Out (LOSO) test routine is used as opposed to Leave Three Subjects Out. LOSO has been shown to be an almost unbiased estimate of the classifier's true generalisation error [44]. However, this project uses a large dataset of one-hour ECG recordings from 120 babies, which is much larger than any previous studies carried out [12, 13] . As a result, the decision was made to hold out the data from three subjects at each iteration in order to speed up the testing process.

The LTSO performance assessment framework was used to evaluate a range of classifiers, starting with the simple linear SVM classifier to establish a benchmark performance, before moving to more advanced models such as XGBoost. The following sections detail the results of these models for the LTSO performance evaluation.

3.6. Leave Three Subjects Out Model Evaluation Results

3.6.1. Linear SVM

The linear SVM classifier was evaluated first to establish a benchmark performance with a simple linear model. The SVM classifier from the scikit-learn package was used in the evaluation [37]. The following values were tested for the regularisation hyperparameter C: [0.001, 0.01, 0.1, 1, 10]. The output of the SVM was converted to a probability of the positive class ('Moderate/Major/Severe') using Platt scaling [45].

The linear SVM achieved a mean epoch-level AUC of 0.714, with a standard deviation of 0.232. It achieved a median epoch-level AUC of 0.747 with an interquartile range of 0.296. The overall subject-level AUC was 0.746. The most common hyperparameter value selected during the evaluation was C = 0.001. The ROC curve for the linear SVM is shown in Figure 12.

Such a small value for the regularisation hyperparameter C indicates that the SVM didn't even come close to fitting a hard margin, and it was forced to allow a large number of margin violations in order to fit the optimal decision boundary. The results for the linear SVM were disappointing, achieving a mean epoch-level AUC that was only just better than using our best feature to make predictions (SD_NN_RMSSD had an AUC of 0.70). It's apparent that the classifier became more accurate when considering the entire hour-long recording for each neonate, achieving a subject level AUC of 0.75.

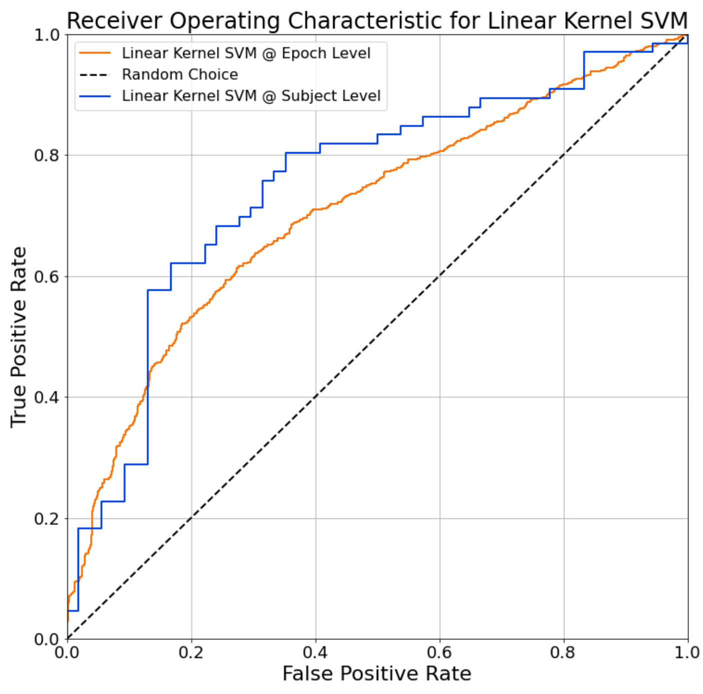


Figure 12: ROC curve for LTSO evaluation of linear SVM.

3.6.2. RBF Kernel SVM

The RBF kernel SVM was evaluated next. As outlined earlier, the RBF kernel SVM is a powerful non-linear classifier that is capable of constructing non-linear decision boundaries that are not possible with the linear SVM. The RBF kernel SVM was implemented using the scikit-learn package [37]. The following values were tested for the regularisation hyperparameter C: [0.001, 0.01, 0.1, 1, 10, 100]. For the hyperparameter gamma (γ, used in the RBF kernel), the following values were tested: [0.001, 0.01, 0.1, 1, 10, 100, 1000].

The RBF kernel SVM achieved a mean epoch-level AUC of 0.738, with a standard deviation of 0.226. It achieved a median epoch-level AUC of 0.762 with an interquartile range of 0.309. The overall subject-level AUC was 0.768. The most common hyperparameter values selected during the evaluation were C = 0.1 and γ = 0.01. The ROC curve for the RBF kernel SVM is shown in Figure 13.

The hyperparameter gamma plays a vital role in constructing the non-linear decision boundary – a very large value of gamma indicates a highly complex boundary that is most likely overfitting the data, whereas a very small value indicates that the classifier is not able to capture the complexity of the data and the boundary is more linear. The values that were selected most frequently for C and gamma indicate that the model struggled to fit a more complex boundary that outperformed a simple linear boundary, allowing a large number of margin violations. That said, both the mean epoch-level AUC and subject-level AUC improved while the standard deviation and interquartile range in epoch-level AUC was maintained.

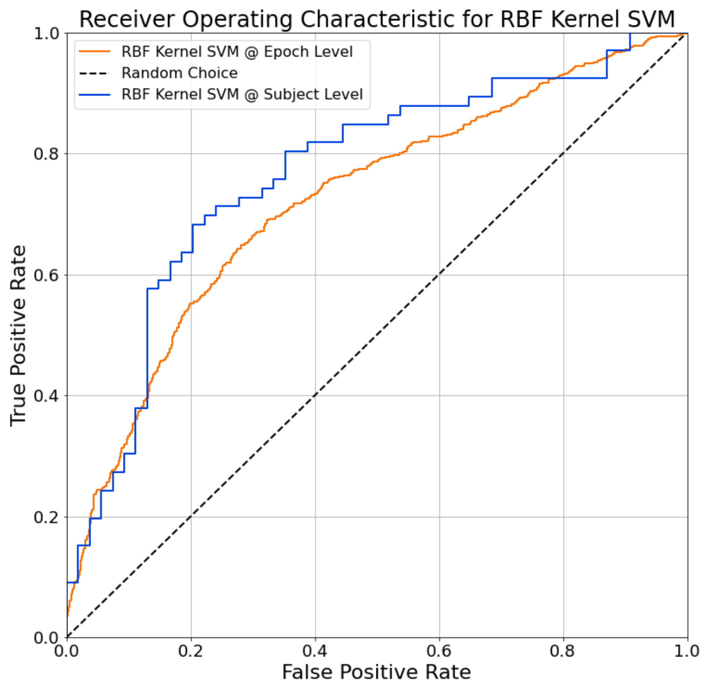


Figure 13: ROC curve for LTSO evaluation of RBF kernel SVM.

3.6.3. XGBoost

XGBoost is a state-of-the-art classification model that has been used in many competitive data science competitions such as the Netflix Prize competition [33, 46], and thus was evaluated for use in this project. The XGBoost classifier was implemented using the open source XGBoost library [33]. The following hyperparameter values were tested: max_depth: [2, 3, 4, 5, 6, 7, 8], subsample: [0.6, 0.8, 1.0], colsample_bytree: [0.2, 0.3, 0.4, 0.5], eta (η): [0.1, 0.2, 0.3, 0.4, 0.5]. ‘max_depth’ corresponds to the maximum allowed number of nodes in each tree, ‘subsample’ specifies the proportion of the training data to use to train each tree, ‘colsample_bytree’ specifies the maximum proportion of features that can be evaluated at each tree node for splitting and ‘eta’ corresponds to the learning rate, which scales the output of each leaf node in each tree.

The XGBoost classifier achieved a mean epoch-level AUC of 0.769, with a standard deviation of 0.23. It achieved a median epoch-level AUC of 0.842, with an interquartile range of 0.349. The overall subject-level AUC was 0.782. The most common hyperparameter values selected during the evaluation were max_depth = 3 and subsample = 1.0, colsample_bytree = 0.5 and eta = 0.3. The ROC curve for the XGBoost model is shown in Figure 14.

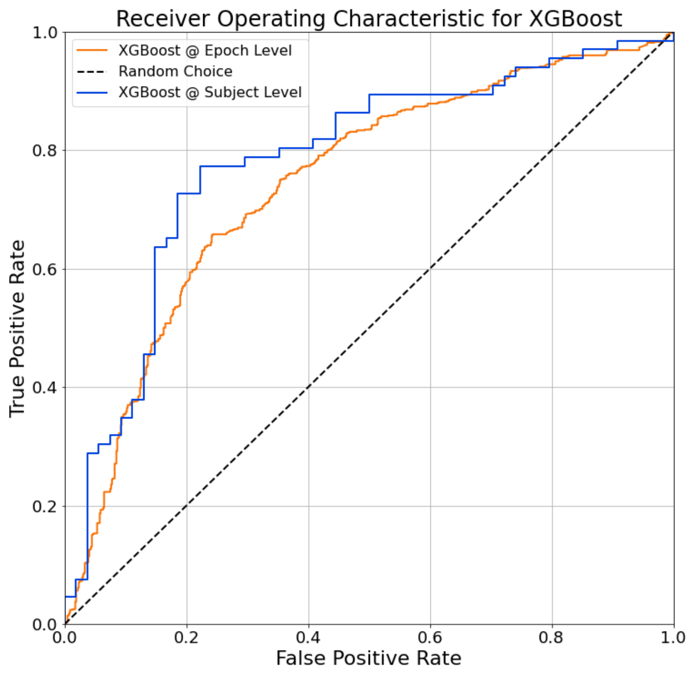


Figure 14: ROC curve for LTSO evaluation of XGBoost.

The XGBoost showed improvements over both SVMs tested previously, achieving the highest mean epoch-level AUC and subject-level AUC. It's worth noting that the interquartile range increased to 0.349, meaning the XGBoost was slightly less consistent in its predictions.

3.6.4. Soft Voting Ensemble (XGBoost and RBF kernel SVM)

A soft voting ensemble uses multiple classifiers in parallel, using the average of their scores or probabilities as the prediction when classifying an instance. Soft voting ensembles work best when the classifier's predictions aren't correlated, such that the classifiers make a diverse set of mistakes. As the RBF kernel SVM and XGBoost model are completely different classification algorithms and appeared to be making different mistakes across the 40 folds during the LTSO evaluation, it was decided to implement these in a soft voting ensemble. The code for the LTSO evaluation was changed, such that two separate cross validations were carried out at each iteration, one for the RBF kernel SVM and one for XGBoost. Both of these models then made a prediction on the 3 left-out babies, and the probability of 'Moderate/Major/Severe' grade of HIE was averaged between them for each 5-minute epoch. The following hyperparameters were evaluated for XGBoost: max_depth: [2, 3, 4, 5, 6, 7, 8], subsample: [0.6, 0.8, 1.0], colsample_bytree: [0.2, 0.3, 0.4, 0.5], eta (η): [0.1, 0.2, 0.3, 0.4, 0.5]. These hyperparameters were evaluated for the SVM: C: [0.001, 0.01, 0.1, 1, 10, 100], gamma (γ): [0.001, 0.01, 0.1, 1, 10, 100, 1000].

The soft voting ensemble achieved a mean epoch-level AUC of 0.769, with a standard deviation of 0.226. It achieved a median epoch-level AUC of 0.858, with an interquartile range of 0.362. The overall subject-level AUC was 0.783. The most common hyperparameter values selected for XGBoost during the evaluation were max_depth = 3 and subsample = 1.0, colsample_bytree = 0.5 and eta = 0.3. The most common hyperparameter values selected for the RBF kernel SVM during the evaluation were C = 0.1 and γ = 0.01. The ROC curve for the ensemble is shown in Figure 15.

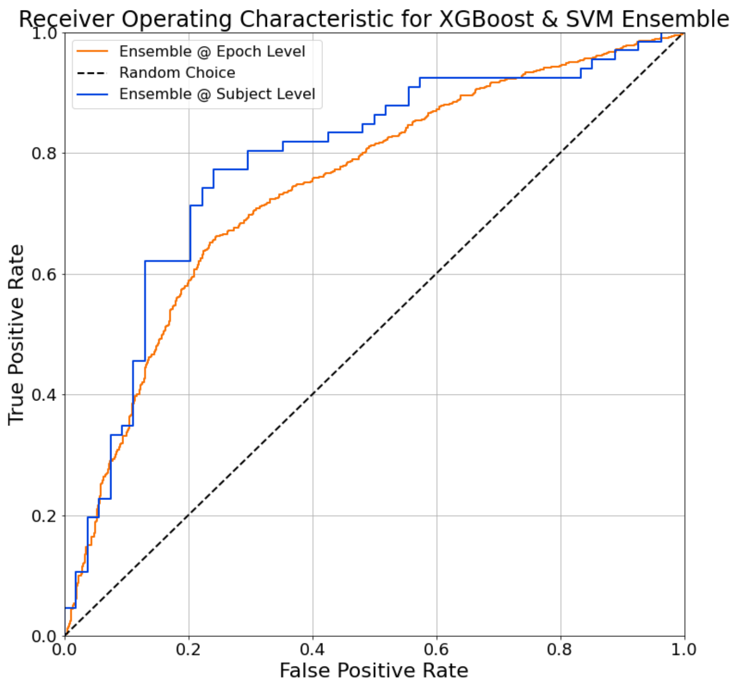


Figure 15: ROC curve for LTSO evaluation of XGBoost & SVM ensemble.

The soft voting ensemble did not offer much (if any) improvement over XGBoost. The median epoch level AUC was the only metric that improved, while the interquartile range of the epoch-level AUC increased, meaning the ensemble was less consistent than XGBoost on its own. It's clear that the SVM and XGBoost were actually making similar mistakes, contrary to what was originally thought. Rather than each classifier complementing the other, they seemed instead to make the same mistakes and largely make the same predictions, which can explain the lack of improvement in performance of the ensemble compared to XGBoost on its own.

3.6.5. Comparison of Model Results

A side-by-side comparison of each model is offered in this section, including the classifiers tested by Daragh. Daragh evaluated a Decision Tree classifier, a Random Forest classifier and an Adaptive Boosting (AdaBoost) classifier. In brief, a Random Forest classifier is an ensemble method composed of multiple parallel decision trees, with each tree trained on a subset of the data to produce a diverse ensemble. The output of the Random Forest is determined by a vote between all the decision trees in the forest [35]. AdaBoost is similar to XGBoost, using a number of decision trees in series, each correcting the error of the previous tree. However, rather than correcting the residual error of the previous tree, AdaBoost increases the weights of instances misclassified by previous trees, and trains the next tree using these newly weighted instances. AdaBoost makes predictions with a weighted vote from all the trees in the ensemble [47]. These classifiers were all implemented using the scikit-learn library [37]. A comparison of all models is shown in Table 6.

	Linear SVM	**RBF SVM**	**Decision Tree**	**Random Forest**	**XGBoost**	**Soft Vote Ensemble**	**Adaboost**
Mean epoch AUC	0.714	0.738	0.724	0.741	0.769	0.770	0.728
Std epoch AUC	0.232	0.226	0.226	0.250	0.230	0.226	0.256
Median epoch AUC	0.747	0.762	0.790	0.833	0.842	0.859	0.800
IQR epoch AUC	0.296	0.309	0.305	0.306	0.349	0.361	0.446
Subject-level AUC	0.749	0.766	0.725	0.754	0.782	0.782	0.764

Table 6: Comparison of LTSO results from all classifiers evaluated. Std corresponds to the standard deviation and IQR corresponds to the interquartile range of the epoch-level AUC across all 40 test folds.

The best model was a toss-up between XGBoost and the soft voting ensemble of the RBF kernel SVM and XGBoost. They both had the same mean epoch-level AUC and subject-level AUC, however XGBoost had a slightly lower IQR, while the ensemble had a higher median. Overall, there wasn’t much difference in results: most of the classifiers had a subject level AUC in the range 0.75 – 0.78. The lowest subject level AUC was 0.725 with a simple decision tree classifier, while the highest subject level AUC was 0.782 with the complex XGBoost model.

3.7. Conclusion

This chapter gave a very brief introduction to some key concepts in machine learning, before covering some of the important background theory behind the classification models used in this project. The initial approach to the project and the resulting conclusions drawn was then outlined, followed by a description of the performance evaluation routine that was developed to assess the classifiers. Finally, the results of all the classifiers were presented and discussed.

It was clear from the results that a performance ceiling was reached that couldn’t be broken with even the most powerful classification models, such as XGBoost. In machine learning, the models are only as good as the data given to them. Maximum performance is being squeezed from the fourteen HRV features by the classifiers, but obviously the features themselves are not discriminative enough to yield a greater performance. Moving to deep learning would allow these features to be discarded. Deep learning is a subfield of machine learning which studies deep artificial neural networks. State of the art neural network architectures such as convolutional neural networks can train directly on the raw RR interval data and extract their own custom features that help the model classify each instance. This has the potential to smash the performance ceiling that seems to be limiting the classical machine learning classifiers that are forced to rely on the fourteen HRV features and offer greater discriminative power between the ‘Normal/Mild’ and ‘Moderate/Major/Severe’ HIE severity grades.

Chapter 4: Deep Learning

In the previous chapter, it was shown that a performance ceiling had been reached with the classic machine learning classifiers. It was concluded that the models were limited by the discriminative power of the fourteen HRV features. This issue can be mitigated by moving to deep learning, where complex neural network architectures can eliminate the need for hand-crafted features. This chapter starts by outlining the theory behind multilayer perceptrons (MLPs) and convolutional neural networks (CNNs), before discussing the architectures that were evaluated and the results that were achieved.

4.1. Multilayer Perceptrons

Multilayer perceptrons (MLPs) are composed of stacked layers of neurons, with each MLP having an input layer, at least one hidden layer and an output layer [48]. When all the neurons in a given layer are connected to every neuron in the previous layer, it is termed a ‘fully connected’ or ‘dense’ layer. Every layer besides the output layer includes an extra bias neuron that always outputs 1. The neurons in the input layer simply pass through whatever data is fed to the network. The neurons in the hidden layer(s) and output layer are slightly more complex: each neuron computes a weighted sum of its inputs and the bias term. The output of this sum is fed to an activation function whose purpose is to introduce non-linearity to the network (without the activation function, the network would just be a chain of linear transformations). The choice of activation function depends on the layer: for hidden layers, the ReLU activation function is normally used [49]. This is shown in Equation (13).

$$\text{ReLU}(0, z) = \max(0, z) \tag{13}$$

The activation function for the output layer depends on the task. A sigmoid activation function is usually used for binary classification problems as the output can be interpreted as the probability of the positive class. An MLP with one hidden layer, and a neuron with a ReLU activation function is shown in Figure 16. The connections between neurons in the MLP architecture represent the weights of the network.

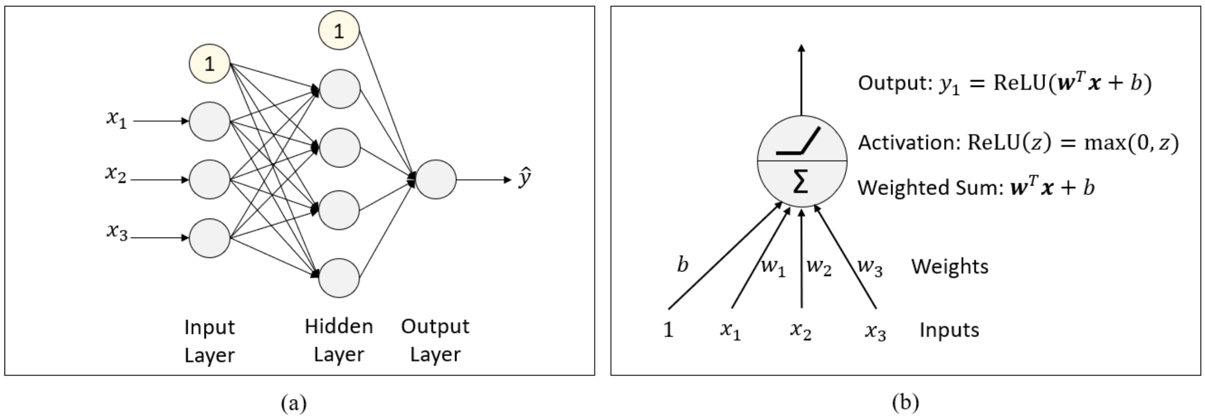

1
1
x_1
x_2
x_3
$\hat{y}$
Input Layer
Hidden Layer
Output Layer
(a)
Output: $y_1 = \text{ReLU}(\boldsymbol{w}^T\boldsymbol{x} + b)$
Activation: $\text{ReLU}(z) = \max(0, z)$
Weighted Sum: $\boldsymbol{w}^T\boldsymbol{x} + b$
Σ
b
w_1
w_2
w_3
Weights
1
x_1
x_2
x_3
Inputs
(b)

Figure 16: Illustration of an MLP, where (a) shows an MLP with one hidden layer with three input features and a single output, and (b) shows the operation of a neuron with a ReLU activation function.

MLPs are trained using the backpropagation algorithm [50]. Backpropagation trains the MLP using mini batches of the training data (typically 32 instances) and works as follows. All of the weights in the network are randomly initialised. The MLP makes a prediction on the batch, called the forward pass, and the outputs of each neuron are saved. The algorithm measures the output error of the MLP using a loss function (e.g. cross-entropy for binary classification), and then works backward through the network, applying the chain rule to measure the contribution of each connection weight to the error (i.e. the error gradient of each weight). This is called the reverse pass. Backpropagation then uses the error gradients in a gradient descent optimisation step to tweak the weights and reduce the loss. This process is repeated for many different batches, passing through the entire training set multiple times. Each iteration through the training set is called an epoch. Training stops after a specified number of epochs.

4.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special type of artificial neural network that are popular for computer vision problems. They do not rely on engineered features and instead can generate their own features from the raw data (e.g. an image or 1-D signal) [51]. The fundamental building block of a CNN is the convolutional layer. Each convolutional layer contains a number of filters made up of neurons. The convolutional layers shown in Figure 17 have 5 filters each. Unlike the neurons in an MLP, the neurons in a convolutional layer only span a limited number of samples from the previous layer, called the ‘kernel size’ or ‘receptive field’ of the neuron. The neurons in the first convolutional layer in Figure 17 have a kernel size of 4 samples. Similar to the neuron in Figure 16 (b), each neuron computes a weighted sum of the samples in its receptive field, plus a bias term. The activation function is then applied to the result. The next neuron in that filter shifts its receptive field by an amount known as the ‘stride’. The neurons in the first convolutional layer in Figure 17 have a stride of 2 samples. It then computes a weighted sum of its own unique receptive field and applies the activation function. Every neuron in a certain filter has the same weights and bias terms (but obviously, different filters have different weights and biases).

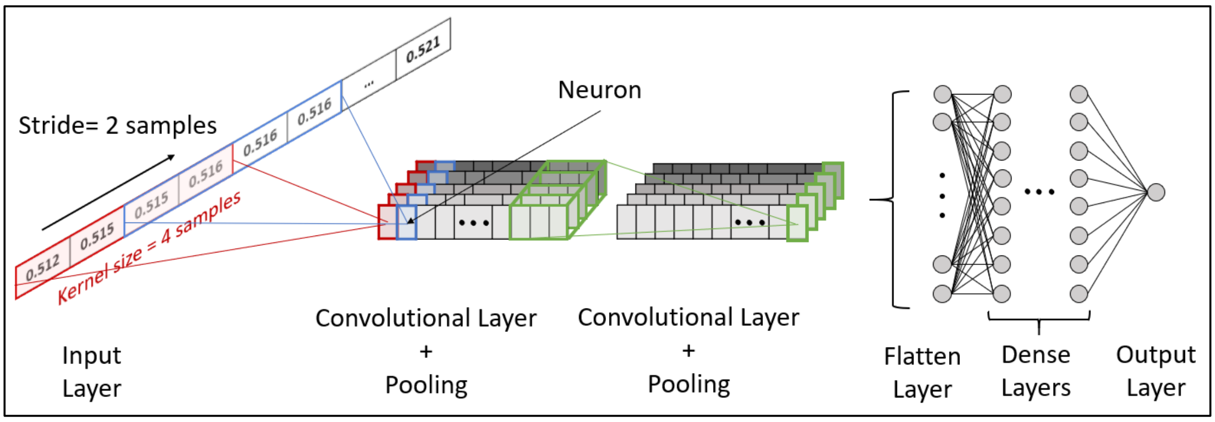


Figure 17: Illustration of a CNN for a binary classification problem, showing the filters, neurons, kernel size, stride and dense network at the output.

All of the filters in a certain convolutional layer have the same kernel size and stride, as well as the same activation function for each neuron. The padding of the convolutional layer defines whether to pad the input to the layer with zeros such that for a stride of 1, the size of the output will be the same as the input. A pooling layer usually follows a convolutional layer to down sample the output of each filter and reduce the dimensionality of the data [51]. For simplicity, the pooling layers are not shown in Figure 17. Neurons in a pooling layer are similar to neurons in a convolutional layer: each neuron has its own receptive field, but rather than using a weighted sum and activation function, it simply takes the maximum value in its receptive field (Max Pooling) or it takes the average of the samples (Average Pooling). The kernel size, stride, and padding all have to be defined for the pooling layer. For convolutional layers that take the output of another convolutional layer or pooling layer as its input (i.e. the second convolutional layer in Figure 17), the receptive field of the neurons spans across all the filters in the previous layer. Therefore, each filter in that layer has a 2-D set of weights, where the dimensions correspond to the filter's receptive field and the number of filters in the previous layer. The filter still has only 1 bias term, however. Stacking convolutional layers in this manner allows 'low level' features to be learned from the raw input data which are progressively refined by succeeding convolutional layers to create 'high level' features.

The output of the convolutional layers is fed to a regular fully connected network such as the MLP. The function of the MLP is to learn how to classify the features extracted by the lower convolutional layers. A 'flatten' layer must be used to concatenate the outputs of the filters in the last convolutional layer to ensure the input to the dense network is 1-D. As discussed in the last section, the hidden layers of the dense network typically use ReLU activation functions, while the activation function of the output layer depends on the task (typically sigmoid for binary classification). The weights of this upper dense network and the filter weights and biases in each convolutional layer are learned during training using the backpropagation algorithm [50, 51].

CNNs are an attractive tool as they do not require engineered features but also require less parameters than a regular fully connected network: the fact that each neuron has a limited receptive field means that less weights need to be learned during training.

4.3. Multilayer Perceptron Evaluation and Results

Various MLP architectures were implemented and tested first. While the reason for moving to deep learning was to eliminate the need for the fourteen HRV features, MLPs do require the use of features. The purpose of evaluating MLPs first was to gain experience in Keras and Tensorflow, which are popular open-source deep learning libraries [52, 53] . The Leave Three Subjects Out performance assessment script had to be modified to remove the nested 5-fold cross validation. It's not computationally feasible to evaluate many different architectures with a cross validation as there are simply too many possible hyperparameters configurations (e.g. the number of hidden layers, the number of neurons in each layer, choice of activation function). Thus, a single architecture was tested across the full 40 folds and any subsequent changes to the architecture were tested across the entire 40 folds again. Support for Tensorboard was also added to the LTSO evaluation, which is an industry standard tool used to monitor the progress of deep learning models during training. A screenshot of the Tensorboard dashboard is shown in Appendix C.

The best performing architecture is shown in Figure 18. This architecture used the cross-entropy loss function, ReLU activation for the neurons in the hidden layers, sigmoid activation for the output layer and the Nesterov accelerated Adaptive Momentum optimiser for the gradient descent performed during backpropagation [54]. The training limit was set to 300 epochs and the batch size was set to 32 instances. The training set at each iteration of the LTSO evaluation was scaled such that the feature values ranged from 0 to 1. The same transformation was applied to the test babies at each iteration using the maximum and minimum feature values from the training set. Early stopping was used with a limit of 10 epochs to prevent overfitting – a validation set was created from a simple 80/20 split of the epochs in the training data.

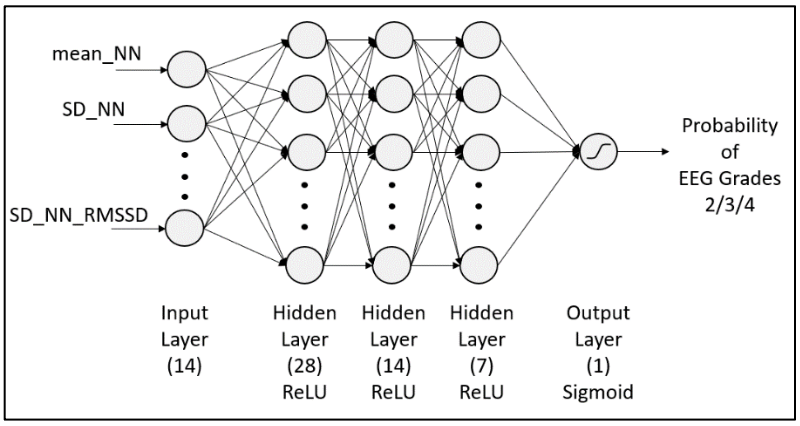


Figure 18: Best performing MLP architecture, where (N) is the number of neurons in that layer. The bias neurons are implicit.

This architecture achieved a mean epoch-level AUC of 0.737, with a standard deviation of 0.217. It achieved a median epoch-level AUC of 0.769 with an interquartile range of 0.304. The overall subject-level AUC was 0.770. This was a very similar performance to the RBF kernel SVM.

4.4. Convolutional Neural Network Evaluation and Results

Modifications had to be made to the original feature extraction MATLAB script, which was carried out by Daragh. The raw RR interval data was segmented into 5-minute epochs and resampled to 4 Hz - no features were extracted. As described earlier, this resampled RR interval data can be fed directly into the CNN. Data augmentation was also carried out to increase the size of the overall dataset – the overlap in epochs was increased to 90%. This increased the size of the dataset from approximately 1,500 epochs to just over 14,000.

The LTSO evaluation script was set up in Google Colab to avail of their free GPUs (NVIDIA Tesla K80 GPU with 12 GB VRAM) to speed up the training of the CNN [55]. Similar to the MLP, the nested 5-fold cross validation was removed and support for Tensorboard was added. All of the CNN architectures were implemented using Keras and Tensorflow [52, 53] .

The first CNN architectures that were implemented were variations of an architecture used for predicting ventricular tachyarrhythmia in adults using heart rate variability [56]. The architecture in that study used 4 convolutional layers, the first with 3 filters and a kernel size of 102, the second with 10 filters and a kernel size of 24, the third with 10 filters and a kernel size of 11 and finally the fourth with 10 filters and a kernel size of 9. ReLU activation functions were used for the convolutional layers. Batch normalisation layers followed each convolutional layer. Batch normalisation is a regularisation technique that standardises and then scales and shifts the data [57]. During training, it tracks a moving average of the mean and standard deviation across each mini batch, and also learns the two parameters which scale and shift the standardised data. It has been shown to reduce the sensitivity of networks to weight initialisation and speed up training [57] . Max pooling layers with a kernel size of 2 then follow the batch normalisation layers which reduces the dimensionality of the data. Finally, a dense network consisting of two hidden layers with 22 neurons in each (and ReLU activation functions) was used for classification. Daragh implemented this architecture with a dense network consisting of two hidden layers with 8 neurons in each, as it was overfitting the data when 22 neurons were used. This architecture ultimately achieved a mean epoch-level AUC of 0.673, with a standard deviation of 0.216. It achieved a median epoch-level AUC of 0.694 with an interquartile range of 0.353. The overall subject-level AUC was 0.761.

Experiments were carried out by building a CNN up from 1 convolutional layer and 1 dense layer, varying the kernel size and number of filters, comparing average pooling and max pooling, and iteratively adding convolutional and dense layers to see how it affects performance. The training and validation loss curves were monitored closely using Tensorboard. Experiments with batch normalisation layers and dropout layers were also performed. Dropout is another regularisation technique for dense networks which randomly sets the output of a proportion of neurons in a dense layer to 0 for each mini batch during training [58]. This has the effect of forcing the neurons to learn features independently and not have classification rely on only a few ‘strong’ neurons in the dense network. Dropout is typically only used in the upper dense layers of the network (i.e. the fully-connected network close to the output that performs the classification).

The best performing CNN is shown in Figure 19. The architecture has 3 convolutional layers with 24 filters in each. The first and second layers have kernel sizes of 8 samples and a stride of 4, while the third layer has a kernel size of 4 samples with a stride of 1. No padding was used for any of the convolutional layers, which was beneficial in reducing the dimensionality of the data. Batch normalisation and average pooling is used after each convolutional layer, the latter having a kernel size and stride of 2. Finally, the dense network had 1 hidden layer with 8 neurons. Batch normalisation at the input reduced the volatility in the validation loss while also eliminating the need to manually standardise the training and test set at each iteration of LTSO. Adding more layers or neurons to the dense network only increased the amount of overfitting. The use of dropout after the hidden dense layer greatly improved performance (a dropout rate of 50% is standard in CNNs [58]). Max pooling performed much worse than average pooling (in general, max pooling destroys more information than average pooling). The number of convolutional layers, the number of filters, the kernel size and other hyperparameters of the convolutional layers were decided upon through experimentation.

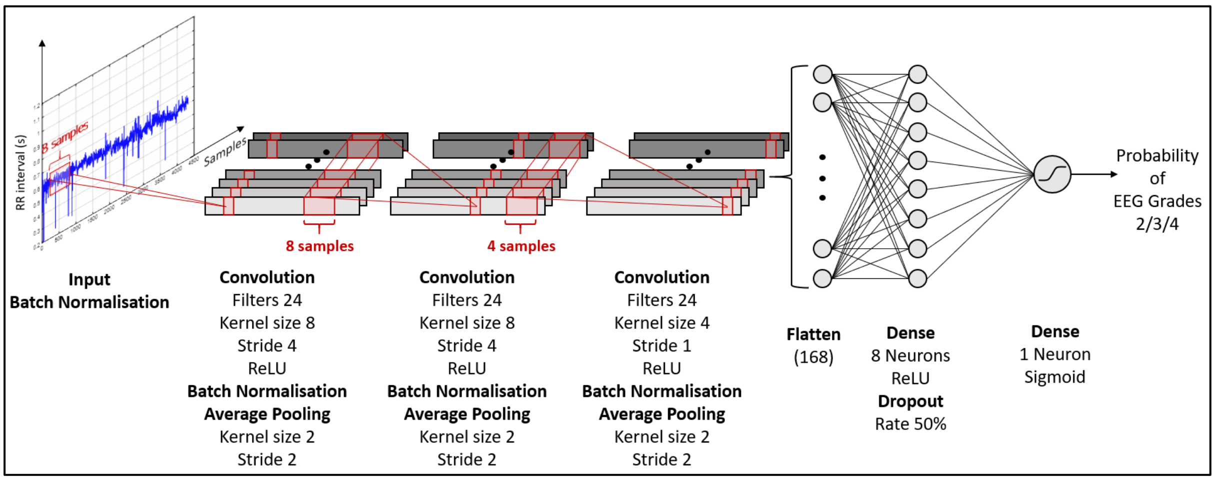


Figure 19:Illustration of best performing CNN.

For training, the architecture used the cross-entropy loss function and the Nesterov accelerated Adaptive Momentum optimiser for the gradient descent performed during backpropagation [54]. The training limit was set to 80 epochs and the batch size was set to 32 instances. Early stopping was used with a limit of 20 epochs to prevent overfitting – a validation set was created from a simple 80/20 split of the epochs in the training data. This architecture achieved a mean epoch-level AUC of 0.772, with a standard deviation of 0.207. It achieved a median epoch-level AUC of 0.829 with an interquartile range of 0.298. The overall subject-level AUC was 0.804. This was the greatest performance achieved by any classifier in this project, demonstrating an excellent performance in distinguishing between the 'Normal/Mild' and 'Moderate/Major/Severe' classes of HIE severity grades. The ROC curve for this CNN architecture is shown in Figure 20.

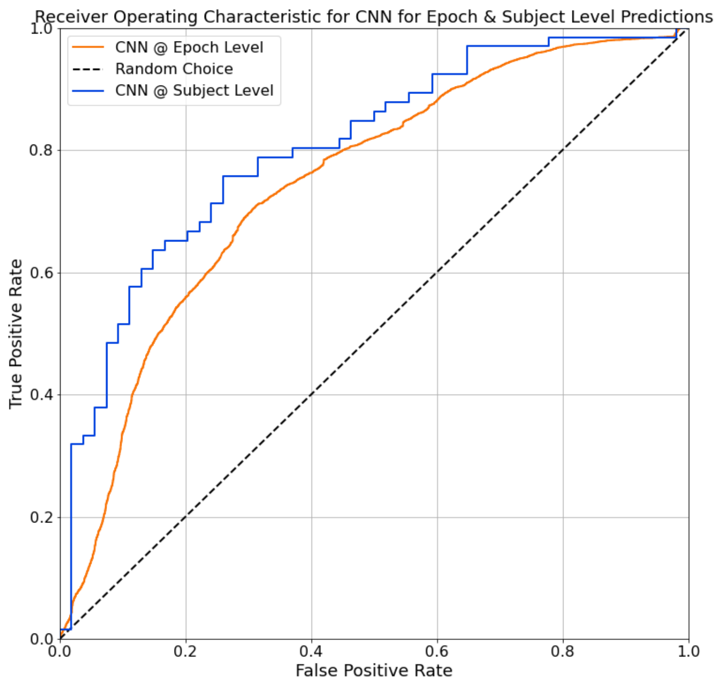


Figure 20: ROC curve for the best performing CNN architecture.

4.5. Conclusion

In this chapter, the theory behind MLPs and CNNs was outlined, showing how CNNs can perform both feature extraction and classification using the raw data. The architecture of the best performing MLP was also detailed, before the experiments with CNNs were discussed. Finally, the architecture and results of the best performing CNN was presented. The CNN saw a reasonable improvement in its subject level AUC (0.80), beating all the previous classifiers tested, but it also saw a reduced variance in the epoch level AUCs across the 40 test folds – meaning it's more consistent in its predictions. The results of this CNN architecture show great promise for the potential use of deep learning for this application.

Chapter 5: Discussion and Conclusion

This chapter offers a final discussion and evaluation of the results achieved with the HRV features and classifiers used in this project to grade the severity of HIE injuries in neonates. Limitations of the project are also discussed, as well as suggestions for future work in this area before a final conclusion to the project is presented.

5.1. Discussion of Results

As shown in Chapters 2 and 3, the original seven HRV features extracted from the RR interval data showed poor discriminative performance in distinguishing between the lumped classes of HIE severity grades ('Normal/Mild' and 'Moderate/Major/Severe'), with three of the features (SD_NN, VLF_power and LF_power) showing performance that's little better than random guessing. The discriminative power of the overall feature set was improved by normalising the NN intervals with respect to the mean_NN and including seven additional HRV features. The four most discriminative features ranked in order of their AUC were SD_NN_RMSSD (0.70), mean_NN (0.67), LF_HF_ratio (0.67) and TINN (0.60). The results of this univariate feature analysis confirmed that there is indeed some correlation between the HRV and severity of HIE. The discriminative power of the fourteen HRV features differed slightly to the findings of the study carried out by R. Ahmed who used a subset of the HRV features used in this project to grade the severity of HIE in neonates [13]. In that study, the four best features ranked in order of their AUC were Entropy (0.70), SD_NN (0.69), SD2_Poincare (0.69) and LF_power (0.67). It's worth noting that this project used a larger dataset of 120 neonates than the dataset of 48 neonates used by R. Ahmed. A direct comparison with the studies by R.M. Goulding et al and Aliefendioğlu et al is difficult, as these studies were carried out on data collected 12 – 48 hours postpartum and 7-days postpartum respectively, and sought only to analyse the correlation between HRV and severity of HIE (i.e. the AUC was not used as a performance metric for the HRV features) [10, 11]. Furthermore, Aliefendioğlu et al assessed the severity of HIE using the Sarnat scoring system rather than the cEEG grade. Both studies however confirmed the correlation between the HRV features used and the severity of HIE.

An exhaustive list of machine learning classification models were tested to combine the limited discriminative power of these fourteen HRV features and ultimately realise the goal of an automated HIE grading system using HRV. These models ranged from simple linear models such as the linear SVM to state-of-the-art models such as XGBoost. Further work was carried out by testing CNNs which performed feature extraction and classification using the raw RR interval data as input. The 'Leave Three Subjects Out' performance assessment routine offered a reliable estimate of the generalisation error of these models on unseen patient data. The results of the performance evaluation for each model are summarised in Table 7. The results of the MLP are not shown as it was only implemented to gain familiarity with Keras and Tensorflow, as mentioned previously.

	Linear SVM	**RBF SVM**	**Decision Tree**	**Random Forest**	**XGBoost**	**Soft Vote Ensemble**	**AdaBoost**	**CNN**
Mean epoch AUC	0.714	0.738	0.724	0.741	0.769	0.770	0.728	0.772
Std epoch AUC	0.232	0.226	0.226	0.250	0.230	0.226	0.256	0.207
Median epoch AUC	0.747	0.762	0.790	0.833	0.842	0.859	0.800	0.829
IQR epoch AUC	0.296	0.309	0.305	0.306	0.349	0.361	0.446	0.298
Subject-level AUC	0.749	0.766	0.725	0.754	0.782	0.782	0.764	0.804

Table 7: Performance summary of all classifiers tested in this project.

The epoch-level AUCs of the LTSO assessment measure the ability of the classifiers to distinguish between the ‘Normal/Mild’ and ‘Moderate/Major/Severe’ grades for each 5-minute window of unseen test subject data. It can be seen in Table 7 that the best performing models in terms of the mean epoch-level AUC were XGBoost (0.77), the Soft Voting Ensemble (0.77) and the CNN (0.77), with the Ensemble having the highest median (0.86). All the classifiers saw an improvement in performance when considering the subject-level AUC, which accumulated the probabilistic predictions of ‘Moderate/Major/Severe’ severity grade for each 5-minute window to give an overall prediction for each 1-hour recording. The CNN achieved a subject level AUC of 0.80, whereas XGBoost and the Soft Voting Ensemble achieved a subject-level AUC of 0.78.

An important consideration is the consistency in the predictions of the models. The distribution of epoch-level AUCs for each classifier during the LTSO evaluation are presented in the box plots shown in Figure 21. The interquartile range (IQR) and the whiskers corresponding to the 5th and 95th percentile of the epoch-level AUCs for the CNN do not stretch as far those for XGBoost and the Soft Voting Ensemble. The whiskers corresponding to the 5th percentile for XGBoost and the Ensemble go beyond the 0.4 epoch-level AUC gridline. It’s worth noting that the epoch-level AUCs less than 0.5 cannot be subtracted from 1.0 as was the case for the univariate feature analysis. The LTSO evaluation is a ‘subject independent’ assessment routine, meaning it’s not possible to predict which of the 40 test folds to apply this adjustment to without knowing the ground truth labels beforehand. The CNN achieved an IQR of 0.30, compared to the IQR of 0.35 and 0.361 of XGBoost and the ensemble respectively.

It’s worth noting that particular test folds performed badly across all classifiers, which is indicated by the outliers below the 5th percentile in Figure 21. Some of the babies in these folds had minor artefacts in the RR interval data, more concerning however was the presence of a completely depressed HRV for presumably healthy babies. The cause of these outliers is speculated about in Section 5.2.

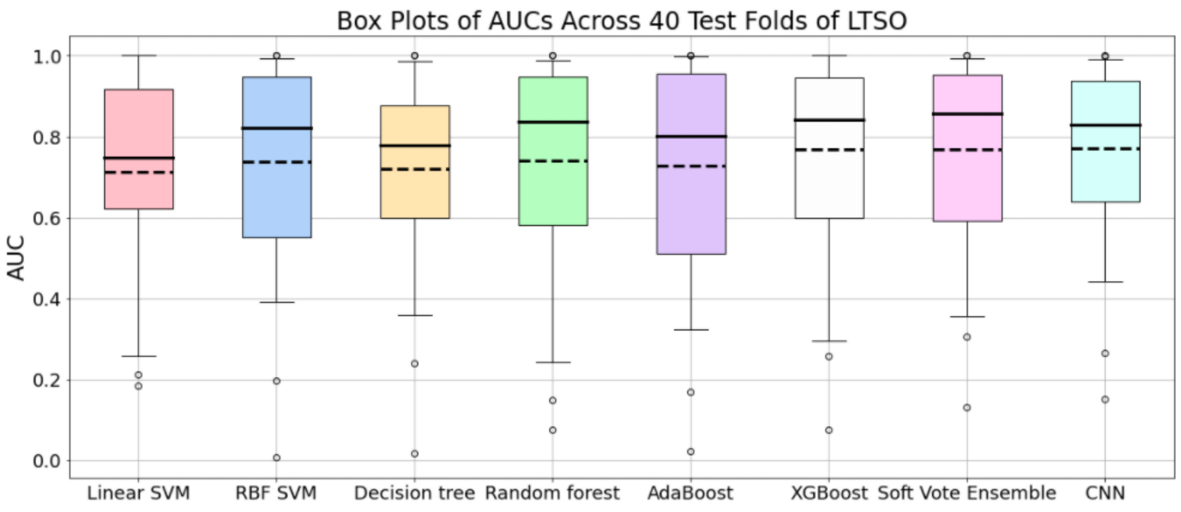


Figure 21: Box plot of epoch-level AUCs across 40 test folds for each classifier during the 'Leave Three Subjects Out' assessment. The solid horizontal line represents the median epoch-level AUC while the dashed line corresponds to the mean epoch-level AUC. The whiskers extend to the 5th and 95th percentile while the box extends from the first quartile to the third quartile (i.e. the IQR of epoch-level AUC).

The CNN ranked as the best classifier across all metrics besides the median epoch-level AUC. The CNN architecture that achieved these results used only 1 hidden dense layer consisting of 8 neurons for classification, which indicates that its superior performance arose from the features extracted by the convolutional layers rather than a sophisticated classification scheme. This suggests that the other feature-based classifiers may benefit from a more diverse, sophisticated set of features.

The results of these classifiers for automatic grading of HIE closely align with similar studies carried out. A 'Leave One Subject Out' performance assessment routine was used by R. Ahmed on the dataset mentioned previously to evaluate an SVM which relied on a 'supervector' composed of sequences of HRV features to classify the HIE severity grade assessed using cEEG [13]. An epoch-level AUC of 0.81 was achieved in that study, although it's not made clear how this value was calculated over all subjects. Matić et al similarly used the 'Leave One Subject Out' assessment to evaluate a Linear Discriminant Analysis (LDA) classifier on a dataset consisting of 2-hour ECG recordings from 19 neonates [12]. HRV features were calculated over the full 2-hour recording. An accuracy of 80% was achieved with this method. A direct comparison with this study is slightly more difficult as the dataset was much smaller and the recordings were taken within 18 – 48 hours postpartum, outside the window in which treatment using therapeutic hypothermia is maximally effective. To the author's knowledge, no previous studies have used deep learning to combine the feature extraction and classification stages for HIE grading using HRV and so no specific comparisons of deep learning models are available.

The results of this project represent a positive step towards realising a system that can automatically grade the severity of HIE using the HRV of the neonate. The dataset used in this project was much larger than any previous studies carried out and the results offer some insight into the limitations of the data collected for this project, as discussed in the next section. To the author’s knowledge, this is also the only attempt at combining the feature extraction and classification stages using deep learning for this application, showing superior performance over the traditional feature-based classifiers that were evaluated.

5.2. Limitations

The main limitation of this project was the potential confounding effects of clinical variables and biomarkers. Information regarding the administration of drugs or therapeutic hypothermia to any of the babies in the dataset was unavailable for this study and so it is only possible to speculate about its effect on the discriminate ability of HRV with respect to HIE severity.

R.M. Goulding et al adjusted for the effects of morphine and Phenobarbitone (a drug used for treating seizures, a common symptom of HIE), finding that the discriminative power of the seven HRV features remained significant after accounting for morphine, and five of the seven features remained significant after adjusting for Phenobarbitone (TINN and LF_power showed no significant differences between severity grades) [11] .

Various studies have investigated the effect of temperature on HRV. A study by Barbeau et al found that no statistically significant differences between LF_power, HF_power and LF_HF_ratio in grading HIE severity remained during normothermia [59]. The HRV features in that study were extracted from 5-minute epochs, with the ECG recorded 6 – 72 hours postpartum. Massaro et al investigated the effect of therapeutic hypothermia on HRV using a dataset of 44 neonates [60]. ECG recordings were taken 2 hours prior to rewarming lasting until 2 hours after rewarming was completed. It was found that HRV decreased as the neonate’s body temperature increased. This is a potential confounding effect that makes it difficult to determine whether a depressed HRV arose from rewarming or the evolution of the HIE severity.

Finally, it has been shown by a number of studies that factors such as gestational age, the presence of sepsis, and the birth weight of the neonate can result in a depressed HRV [61-63].

All of the factors discussed here represent potential confounding effects for the dataset used in this study, but without this information it’s impossible to quantitatively measure their impact on the results presented for this project – for example, to determine whether there is some correlation between these clinical factors and the data for babies which scored poorly across all the classifiers tested, as shown by the outliers below the 5th percentiles in Figure 21.

5.3. Future Work

As discussed in the previous section, it would be beneficial to obtain clinical information for the babies in the dataset, which could potentially be incorporated as features (e.g. the gestational age) with the other quantitative HRV features used in this study, or perhaps using this information to weight instances from certain babies in order to put more emphasis on certain data during training [64] . It’s worth noting that using the clinical information as a feature would be a barrier to implementing this as a truly automatic subject independent system. Furthermore, there is difficulty in obtaining this clinical information, due to ethical issues concerning data privacy but also due to the fact that the dataset may have been compiled from multiple different sources, making it difficult to keep the clinical information consistent across the entire dataset.

The results of this project showed that accumulating the probabilistic predictions of ‘Moderate/Major/Severe’ severity grade for each 5-minute window to give an overall prediction for each 1-hour recording improved the results of all the classifiers tested, ultimately achieving a subject-level AUC of 0.80 with the CNN. Future work could exploit the evolution of the HRV over the full 1-hour ECG recording which may yield an even greater performance.

Finally, further work could focus on the use of deep learning for this application. The initial results presented in this project are promising, but further work is required to evaluate other types of neural network architectures (such as Recurrent Neural Networks). The interpretability of these ‘black-box’ deep learning models is also worthy of an investigation to determine why these models are making certain predictions, and perhaps to determine a relationship between the features extracted by the models and known physiological processes.

5.4. Conclusion

This project aimed to develop a system that can automatically grade the severity of HIE injury in neonates using heart rate variability (HRV). A total of fourteen quantitative measurements of HRV were extracted from the RR intervals of the ECG signal of each neonate in the dataset, which individually showed limited discriminative power between HIE severity grades. These were combined using machine learning classifiers ranging from linear Support Vector Machines (SVMs) to XGBoost, before moving to more advanced deep learning models such as Convolutional Neural Networks (CNNs). The CNN ultimately achieved a good, consistent performance on unseen test subjects over both 5-minute windows and over the entire 1-hour recordings, with a mean epoch-level AUC of 0.77 and a subject-level AUC of 0.80 respectively. This shows great promise for the future development of a subject independent automatic HIE grading system that can ultimately aid clinicians in making decisions regarding treatment of HIE in the NICU.

Ethics

The data used in this project was collected in the Cork University Hospital (CUH) as part of the Delphi project and approval from the Clinical Ethics Committee of the Cork Teaching Hospitals was obtained for the use of the data for research purposes. Consent was also sought from the parents of the neonates in the study. The data was anonymized after collection. It is vitally important that medical data collected for research purposes is safely stored and transmitted. To this end, the data was distributed as a password protected zip file, with the password distributed over a second, separate medium. The data was also stored securely on the cloud and on local hard drives.

References

[1] J. J. Kurinczuk, M. White-Koning, and N. Badawi, "Epidemiology of neonatal encephalopathy and hypoxic–ischaemic encephalopathy," *Early human development,* vol. 86, no. 6, pp. 329-338, 2010.

[2] "Newborn death and illness," 2011,

[3] S. Shankaran *et al.*, "Whole-body hypothermia for neonates with hypoxic–ischemic encephalopathy," *New England Journal of Medicine,* vol. 353, no. 15, pp. 1574-1584, 2005.

[4] D. J. Eicher *et al.*, "Moderate hypothermia in neonatal encephalopathy: efficacy outcomes," *Pediatric neurology,* vol. 32, no. 1, pp. 11-17, 2005.

[5] H. B. Sarnat and M. S. Sarnat, "Neonatal encephalopathy following fetal distress: a clinical and electroencephalographic study," *Archives of neurology,* vol. 33, no. 10, pp. 696-705, 1976.

[6] D. M. Murray, C. A. Ryan, G. B. Boylan, A. P. Fitzgerald, and S. Connolly, "Prediction of seizures in asphyxiated neonates: correlation with continuous video-electroencephalographic monitoring," *Pediatrics,* vol. 118, no. 1, pp. 41-46, 2006.

[7] D. M. Murray, G. B. Boylan, C. A. Ryan, and S. Connolly, "Early EEG findings in hypoxic-ischemic encephalopathy predict outcomes at 2 years," *Pediatrics,* vol. 124, no. 3, pp. e459-e467, 2009.

[8] G. Boylan, L. Burgoyne, C. Moore, B. O'Flaherty, and J. Rennie, "An international survey of EEG use in the neonatal intensive care unit," *Acta paediatrica,* vol. 99, no. 8, pp. 1150-1155, 2010.

[9] E. H. Hon, "Electronic evaluations of the fetal heart rate patterns preceding fetal death, further observations," *Am J Obstet Gynecol,* vol. 87, pp. 814-826, 1965.

[10] D. Aliefendioğlu, T. Doğru, M. Albayrak, E. DibekMısırlıoğlu, and C. Şanlı, "Heart rate variability in neonates with hypoxic ischemic encephalopathy," *The Indian Journal of Pediatrics,* vol. 79, no. 11, pp. 1468-1472, 2012.

[11] R. M. Goulding, N. J. Stevenson, D. M. Murray, V. Livingstone, P. M. Filan, and G. B. Boylan, "Heart rate variability in hypoxic ischemic encephalopathy: correlation with EEG grade and 2-y neurodevelopmental outcome," *Pediatric research,* vol. 77, no. 5, pp. 681-687, 2015.

[12] V. Matić *et al.*, "Heart rate variability in newborns with hypoxic brain injury," in *Oxygen Transport to Tissue XXXV*: Springer, 2013, pp. 43-48.

[13] R. Ahmed, "Dynamic Classifiers for Neonatal Brain Monitoring," PhD, Department of Electrical and Electronic Engineering, National University of Ireland, Cork, Cork, 2016.

[14] E. A. Ashley and J. Niebauer, *Cardiology explained*. Remedica, 2004.

[15] J. Pan and W. J. Tompkins, "A real-time QRS detection algorithm," *IEEE transactions on biomedical engineering,* no. 3, pp. 230-236, 1985.

[16] H. Dubey *et al.*, "Fog computing in medical internet-of-things: architecture, implementation, and applications," in *Handbook of Large-Scale Distributed Computing in Smart Healthcare*: Springer, 2017, pp. 281-321.

[17] A. J. Camm *et al.*, "Heart rate variability: standards of measurement, physiological interpretation and clinical use. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology," 1996.

[18] O. Semenova, "Intelligent Monitoring and Interpretation of Preterm Physiological Signals Using Machine Learning," PhD, Department of Electrical and Electronic Engineering, National University of Ireland, Cork, Cork, 2020.

[19] E. Stephan-Blanchard, K. Chardon, A. Leke, S. Delanaud, V. Bach, and F. Telliez, "Heart rate variability in sleeping preterm neonates exposed to cool and warm thermal conditions," *PloS one,* vol. 8, no. 7, p. e68211, 2013.

[20] A. Malliani, M. Pagani, F. Lombardi, and S. Cerutti, "Cardiovascular neural regulation explored in the frequency domain," *Circulation,* vol. 84, no. 2, pp. 482-492, 1991.

[21] B. Pomeranz *et al.*, "Assessment of autonomic function in humans by heart rate spectral analysis," *American Journal of Physiology-Heart and Circulatory Physiology,* vol. 248, no. 1, pp. H151-H153, 1985.

[22] T. Fawcett, "An introduction to ROC analysis," *Pattern recognition letters,* vol. 27, no. 8, pp. 861-874, 2006.

[23] J. Sacha, "Why should one normalize heart rate variability with respect to average heart rate," *Frontiers in physiology,* vol. 4, p. 306, 2013.

[24] E. Ostertagová and O. Ostertag, "Methodology and application of oneway ANOVA," *American Journal of Mechanical Engineering,* vol. 1, no. 7, pp. 256-261, 2013.

[25] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development,* vol. 3, no. 3, pp. 210-229, 1959.

[26] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes," in *Advances in neural information processing systems*, 2002, pp. 841-848.

[27] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, "Support vector clustering," *Journal of machine learning research,* vol. 2, no. Dec, pp. 125-137, 2001.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning,* vol. 20, no. 3, pp. 273-297, 1995.

[29] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, 2000: Springer, pp. 1-15.

[30] R. E. Schapire, "The strength of weak learnability," *Machine learning,* vol. 5, no. 2, pp. 197-227, 1990.

[31] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[32] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics,* pp. 1189-1232, 2001.

[33] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System In Proceedings of the 22nd ACM SIGKDD," in *International Conference on Knowledge Discovery and Data Mining (KDD2016) 785 San Francisco, USA*, 2016.

[34] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis,* vol. 38, no. 4, pp. 367-378, 2002.

[35] L. Breiman, "Random forests," *Machine learning,* vol. 45, no. 1, pp. 5-32, 2001.

[36] L. Breiman, "Bagging predictors," *Machine learning,* vol. 24, no. 2, pp. 123-140, 1996.

[37] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *the Journal of machine Learning research,* vol. 12, pp. 2825-2830, 2011.

[38] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning,* vol. 46, no. 1-3, pp. 389-422, 2002.

[39] O. Doyle, A. Temko, W. Marnane, G. Lightbody, and G. Boylan, "Heart rate based automatic seizure detection in the newborn," *Medical engineering & physics,* vol. 32, no. 8, pp. 829-839, 2010.

[40] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik, "Feature selection for SVMs," *Advances in neural information processing systems,* vol. 13, pp. 668-674, 2000.

[41] I. Guyon *et al.*, "Design and analysis of the causation and prediction challenge," in *Causation and Prediction Challenge*, 2008, pp. 1-33.

[42] C. R. Harris *et al.*, "Array programming with NumPy," *Nature,* vol. 585, no. 7825, pp. 357-362, 2020.

[43] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, 2010, vol. 445: Austin, TX, pp. 51-56.

[44] V. Vapnik, *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[45] J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers,* vol. 10, no. 3, pp. 61-74, 1999.

[46] J. Bennett and S. Lanning, "The netflix prize," in *Proceedings of KDD cup and workshop*, 2007, vol. 2007: Citeseer, p. 35.

[47] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences,* vol. 55, no. 1, pp. 119-139, 1997.

[48] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436-444, 2015.

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097-1105, 2012.

[50] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[51] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE international symposium on circuits and systems*, 2010: IEEE, pp. 253-256.

[52] F. a. o. Chollet. "Keras." GitHub. (accessed.

[53] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[54] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[55] E. Bisong, "Google colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*: Springer, 2019, pp. 59-64.

[56] G. T. Taye, H.-J. Hwang, and K. M. Lim, "Application of a convolutional neural network for predicting the occurrence of ventricular tachyarrhythmia using heart rate variability features," *Scientific reports*, vol. 10, no. 1, pp. 1-7, 2020.

[57] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, 2015: PMLR, pp. 448-456.

[58] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929-1958, 2014.

[59] D. Yasova Barbeau *et al.*, "Heart rate variability and inflammatory markers in neonates with hypoxic-ischemic encephalopathy," *Physiological reports*, vol. 7, no. 15, p. e14110, 2019.

[60] A. N. Massaro *et al.*, "Effect of temperature on heart rate variability in neonatal intensive care unit patients with hypoxic ischemic encephalopathy," *Pediatric critical care medicine: a journal of the Society of Critical Care Medicine and the World Federation of Pediatric Intensive and Critical Care Societies*, vol. 18, no. 4, p. 349, 2017.

[61] F. A. Selig, E. R. Tonolli, E. Silva, and M. D. Godoy, "Heart rate variability in preterm and term neonates," *Arq Bras Cardiol*, vol. 96, no. 6, pp. 443-9, 2011.

[62] F. J. Bohanon *et al.*, "Heart rate variability analysis is more sensitive at identifying neonatal sepsis than conventional vital signs," *The American Journal of Surgery*, vol. 210, no. 4, pp. 661-667, 2015.

[63] A. Rakow, M. Katz-Salamon, M. Ericson, A. Edner, and M. Vanpée, "Decreased heart rate variability in children born with low birth weight," *Pediatric research*, vol. 74, no. 3, pp. 339-343, 2013.

[64] M. Hashemi and H. Karimi, "Weighted machine learning," *Statistics, Optimization and Information Computing*, vol. 6, no. 4, pp. 497-525, 2018.

Appendix A: Binary Cross Entropy Function

The binary cross entropy loss function is shown in Equation A.1, where $y^{(i)} \in \{0,1\}$ is the ground truth label for the i^{th} instance and $p^{(i)}$ is the predicted probability of the positive class for the i^{th} instance.

$$L\left(y^{(i)}, p^{(i)}\right) = -y^{(i)} log\left(p^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - p^{(i)}\right) \tag{A.1}$$

The plot for this function is shown in Figure A.1. It can be seen that for $y = 1$, the loss is greatest when p is closest to 0, and vice versa when $y = 0$. This heavily penalises models during training when there is a large error present in its predictions.

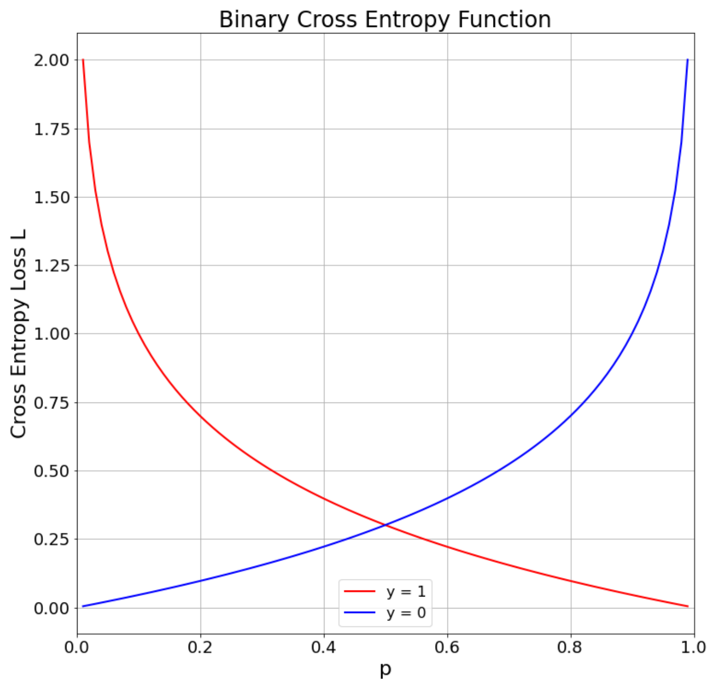


Figure A.1: Plot of binary cross entropy loss function.

Appendix B: Leave Three Subjects Out Script for RBF kernel SVM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from Create_Baby_IDs import Create_Baby_IDs
from prepare_hrv_feats_df import prepare_hrv_feats_df
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

raw_df =
pd.read_csv('../../Data_Semester_2/extended_features_5min_epoch_0_overlap_0
5-02-2021 15-54.csv')
raw_df.drop(columns=['Allan_Factor'],inplace=True)
data = prepare_hrv_feats_df(raw_df)
Baby_IDs = np.arange(1,121)
Baby_ID_labels = pd.Series(data=np.zeros(120),index=Baby_IDs,
                        name='Baby ID EEG Grades')
for ID in Baby_IDs:
    Baby_ID_labels[ID] = data.loc[ID,'EEG_grade'].mode()

%%time
n_LOO_folds = 40 # Number of folds to create in the outer loop - i.e.
number of independent test sets to create.
n_cv_folds = 5 # Number of folds in the internal hyperparameter cross
validation loop.

std_scaler = StandardScaler()

hrv_feats = data.copy() # Copy data passed to function.
hrv_feats_y = hrv_feats.EEG_grade # Series of labels.
hrv_feats_X = hrv_feats.drop(columns='EEG_grade') # Dataframe of feature
data.
std_scaler.fit(hrv_feats_X)
param_grid =
{'gamma':[0.001,0.01,0.1,1,10,100,1000],'C':[0.001,0.01,0.1,1,10,100]} #
Coarse parameter grid search.

num_gamma_params = len(param_grid['gamma']) # Number of gamma parameters
being tested.
num_C_params = len(param_grid['C']) # Number of C parameters being tested.

LOO = StratifiedKFold(n_splits = n_LOO_folds, shuffle=True, random_state=0)
# Initialise the Stratified/Shuffle/Split for the 40 folds
LOO_counter = 0 # Tracks the current test fold number.

best_CV_hyperparams = [] # Stores the optimal set of hyperparameters based
on the internal cross validation.
LOO_AUCs = [] # Stores AUCs from each of the 40 test folds.
```

```
subject_level_test_labels = [] # Stores the target labels of the test
babies in the order in which they were tested (used for overall subject
AUC)
epoch_level_test_labels = [] # Stores the target labels of the test baby
epochs in the order in which they were tested.
epoch_level_scores = [] # Stores the scores corresponding to each test
epoch in the order in which they appear.
subject_probabilities = [] # Stores G.M. of each test baby's probability of
abnormality.
geometric_mean = lambda X: np.power(np.prod(X),1/len(X)) # Lambda function
to calculate the geometric mean of a list/array X.

for train_idx,test_idx in LOO.split(Baby_IDs,Baby_ID_labels):
    # ================= Model Evaluation Loop =================
    # In this loop, n_LOO_folds - 1 folds are used as training data, while
the remaining
    # fold is used as test data. No epochs from babies in the training data
appear in the test data
    # and vice versa. If this occurs, an Exception is raised and execution
is interrupted.
    # A stratified shuffle split is done, such that the class distribution
in the test set is preserved
    # when possible.
    # Baby_IDs is an array of integers from 1 to 120. Baby_ID_labels a is
pandas Series, where the index is a
    # baby ID from 1 to 120 and the label is the EEG grade for that baby (-
1 or 1).
    # The stratified split is done on Baby_ID_labels.
    # The RBF kernel SVM is trained using the best parameters from the
internal CV loop,
    # and the epoch level AUC is computed along with the geometric mean
probability of abnormality for each
    # test subject.
    # ========================================================

    training_baby_IDs = Baby_IDs[train_idx] # List of Baby IDs (1 - 120)
for training.
    test_baby_IDs = Baby_IDs[test_idx] # List of Baby IDs (1 - 120) for
testing.
    test_subjects = [] # Used to store numpy arrays of each test subject's
feature data.


    for ID in test_baby_IDs:
        if ID in training_baby_IDs:
            # Exception is raised if a Baby ID in the test set is found in
the training set.
            raise Exception('Error: Overlap in outer loop training and test
sets.\nBaby {} was detected in both training and test sets.'.format(ID))

    X_cv = training_baby_IDs # Baby IDs used in cross validation.
    y_cv = Baby_ID_labels.loc[training_baby_IDs].values # Labels for the
babies used in cross validation.

    SKF_CV = StratifiedKFold(n_splits = n_cv_folds) # Stratified split to
ensure the class distribution is preserved in the internal cross
validation.

    # AUC_hyperparam_matrix will store the mean AUC for each cross-
validation iteration
```

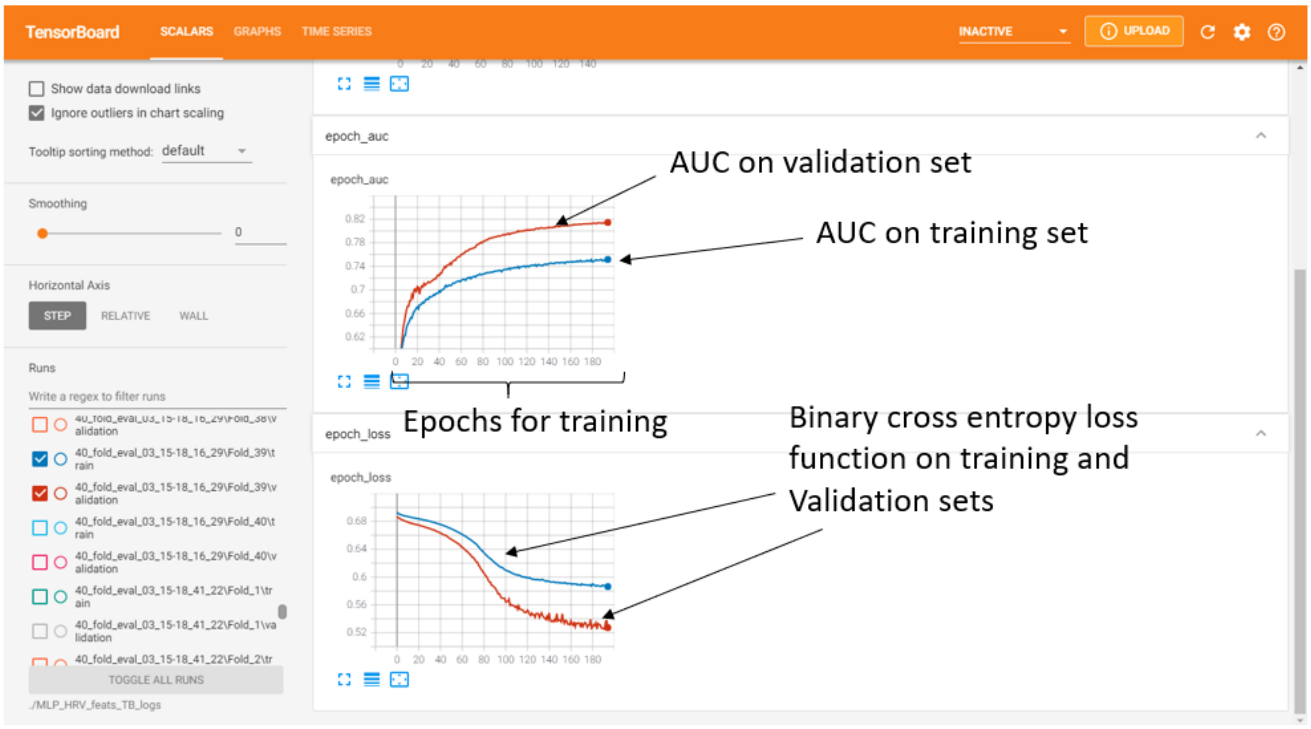
```
    # for a given pair of hyperparameters. It will have (#gamma,#C)
dimensions, i.e.
    # number of rows = number of gamma params being grid searched,
    # number of columns = number of C params being grid searched.
    # After each CV iteration, the corresponding element in the matrix will
be updated with the mean AUC across
    # the 5 folds.

    AUC_hyperparam_matrix = np.empty((num_gamma_params,num_C_params)) #
Initialise the matrix.
    AUC_hyperparam_matrix[:,:] = np.nan # NaNs will indicate if something
has gone wrong.

    AUC_per_fold = np.empty(n_cv_folds) # Stores the AUC for each of the 5
test folds for a given gamma, C.

    for gamma_idx, svm_gamma in enumerate(param_grid['gamma']):
        for C_idx, svm_C in enumerate(param_grid['C']):
            # ================= Internal Cross Validation Loop
=================
            # In this loop, every pair of hyperparameter values (as
specified by param_grid)
            # are trained on 4 (n_cv_folds - 1) of the 5 folds
(n_cv_folds). The resulting model
            # is then tested on the remaining fold. This process is
repeated with each fold
            # used as a test fold. The AUC on the test fold is stored in
AUC_per_fold. The mean AUC
            # across the 5 folds is stored in AUC_hyperparam_matrix, where
the row number = index
            # of the gamma value in param_grid, and the column number =
index of C value in param_grid.
            #
==================================================================

            AUC_per_fold[:] = np.nan # Initialise the AUCs of each test
folds as NaN so we'll know if something is wrong.
            cv_counter = 0 # Tracks the current CV test fold.

            for cv_train_index, cv_test_index in SKF_CV.split(X_cv,y_cv):
                cv_train_IDs = X_cv[cv_train_index] # Baby IDs in CV
training folds.
                cv_test_IDs = X_cv[cv_test_index] # Baby IDs in CV test
fold.

                X_train = hrv_feats_X.loc[cv_train_IDs,:] # Extract feature
data for training folds using Baby IDS.
                X_train = std_scaler.transform(X_train) # Standardise
training data.
                y_train = hrv_feats_y.loc[cv_train_IDs].values # Extract
labels for training fold using Baby IDS.

                X_test = hrv_feats_X.loc[cv_test_IDs,:] # Extract feature
data for test fold using Baby IDS.
                X_test = std_scaler.transform(X_test) # Standardise test
data.
                y_test = hrv_feats_y.loc[cv_test_IDs].values # Extract
labels for test fold using Baby IDS

                svm = SVC(kernel='rbf',gamma=svm_gamma,C=svm_C) #
Initialise new non-linear SVM.
```

```
                svm.fit(X_train,y_train) # Fit to training folds.
                y_test_scores = svm.decision_function(X_test) # Returns
scores on the test set to compute the AUC.
                AUC_per_fold[cv_counter] =
roc_auc_score(y_test,y_test_scores) # Store the AUC on this test fold.

                cv_counter += 1
            # Store the mean AUC across the 5 folds for this pair of
hyperparameters:
            AUC_hyperparam_matrix[gamma_idx,C_idx] = np.mean(AUC_per_fold)

    best_gamma_idx,best_C_idx = np.where(AUC_hyperparam_matrix ==
np.max(AUC_hyperparam_matrix)) # row,col = best gamma, best C
    # best_gamma_idx, best_C_idx are the indices of the best gamma & C
value as determined by the 5 fold CV.

    best_gamma = param_grid['gamma'][np.max(best_gamma_idx)] # Assigns the
best value of gamma to best_gamma
    best_C = param_grid['C'][np.max(best_C_idx)] # Assigns the best value
of C to best_C

    print('\n===== {} of {} evaluations
=====' .format(LOO_counter+1,n_LOO_folds))
    print("Best AUC from internal CV = {} with gamma = {} and C =
{}".format(np.max(AUC_hyperparam_matrix),best_gamma,best_C))

    best_CV_hyperparams.append((best_gamma,best_C)) # Stores the best
performing pair of hyperparameter values from CV.

    best_SVM = SVC(kernel='rbf',gamma=best_gamma,C=best_C,probability=True)
# Intialise a new SVM using the best model params.

    # Extract training data and labels using the training Baby IDs:
    X_train_LOO = hrv_feats_X.loc[training_baby_IDs,:]
    X_train_LOO = std_scaler.transform(X_train_LOO)
    y_train_LOO = hrv_feats_y.loc[training_baby_IDs].values

    best_SVM.fit(X_train_LOO,y_train_LOO)

    # Extract test data and labels using the test Baby IDs:
    X_test_LOO = hrv_feats_X.loc[test_baby_IDs,:]
    X_test_LOO = std_scaler.transform(X_test_LOO)
    y_test_LOO = hrv_feats_y.loc[test_baby_IDs].values

    y_test_LOO_scores = best_SVM.decision_function(X_test_LOO) # Return
scores for each epoch of test babies.
    epoch_level_test_labels = np.append(epoch_level_test_labels,
y_test_LOO)
    epoch_level_scores = np.append(epoch_level_scores, y_test_LOO_scores)

    AUC = roc_auc_score(y_test_LOO,y_test_LOO_scores) # Return epoch level
AUCs for this set of test babies.
    LOO_AUCs.append(AUC) # Save the AUC for this test set.

    print('Test Set Baby IDs: ',test_baby_IDs)
    print('Test Set EEG Grades: ',Baby_ID_labels.loc[test_baby_IDs].values)
    print('Epoch-level AUC across test set using decision scores =
{}'.format(AUC))

    for test_ID in test_baby_IDs:
        # This loop stores the array of each individual test baby's data.
        # i.e. for n_LOO_folds = 40, it is a list of 3 elements, with
        # each element being a numpy matrix of feature data for each baby.
        subject_data = hrv_feats_X.loc[test_ID,:]
        subject_data = std_scaler.transform(subject_data)
        test_subjects.append(subject_data)

        # Lines below store the labels of the test babies in
subject_level_test_labels.
        # These need to be in the correct order so that the overall subject
        # level AUC across all babies can be computed.
        subject_label = Baby_ID_labels.loc[test_ID]
        subject_level_test_labels.append(subject_label)

    for idx,subject in enumerate(test_subjects):
        probability_abnormal = [] # List which stores the probability of
abnormality for each epoch for each test baby.
        # subject_proba is an array of arrays, where the number of arrays =
number of epochs. Each epoch array
        # has 2 elements, index 0 = probability of normal (-1 grade), index
1 = probability of abnormal (+1 grade).
        subject_proba = best_SVM.predict_proba(subject)
        for epoch in subject_proba:
            # Iterate over the epoch probabilities and append the
probability of abnormal (+1 grade) to probability_abnormal
            # list.

probability_abnormal.append(epoch[np.where(best_SVM.classes_==1)])
        # Computes geometric mean of probability_abnormal list for this
baby - used as the subject level prediction probability:

        subject_probabilities.append(geometric_mean(probability_abnormal))
        print('Geometric Mean Probability of Baby {} being Abnormal:
{}'.format(test_baby_IDs[idx],geometric_mean(probability_abnormal)))

    LOO_counter += 1

mode_gamma, mode_C = max(set(best_CV_hyperparams),
key=best_CV_hyperparams.count)

print('\n===== Final Performance Across All Subjects: =====')
print('Mean AUC across all test folds: {}'.format(np.mean(LOO_AUCs)))
print('Standard Deviation of AUC across all test folds:
{}'.format(np.std(LOO_AUCs)))
print('Median of AUC across all test folds:
{}'.format(np.median(LOO_AUCs)))
print('Interquartile Range of AUC across all test folds:
{}'.format(np.subtract(*np.percentile(LOO_AUCs, [75, 25]))))
print('Final Subject level AUC across all subjects:
{}'.format(roc_auc_score(subject_level_test_labels,subject_probabilities)))
print('The most common hyperparameters selected were gamma = {} and C =
{}'.format(mode_gamma,mode_C))

fpr_epoch, tpr_epoch, thresholds_epoch =
roc_curve(epoch_level_test_labels,epoch_level_scores)
fpr_subject, tpr_subject, thresholds_subject =
roc_curve(subject_level_test_labels,subject_probabilities)
plt.figure(figsize=(12,12))
plt.plot(fpr_epoch,tpr_epoch, linewidth=2,label='RBF Kernel SVM @ Epoch
Level',color='xkcd:orange')
plt.plot([0,1],[0,1],linewidth=2,label='Random
Choice',color='black',linestyle='dashed')
```

```
plt.plot(fpr_subject,tpr_subject, linewidth=2,label='RBF Kernel SVM @
Subject Level',color='xkcd:blue')
plt.legend(fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=18)
plt.xlabel('False Positive Rate',fontsize=22)
plt.ylabel('True Positive Rate',fontsize=22)
plt.xlim([0,1])
plt.ylim([0,1])
plt.title('Receiver Operating Characteristic for RBF Kernel
SVM',fontsize=24)
plt.grid()
plt.savefig('rbf_svm_ROC.png')

plt.figure(figsize=(12,12))
sns.set_palette(sns.color_palette("tab10"))
plt.grid()
plt.title('Distribution of AUCs across 40 test folds for RBF Kernel
SVM',fontsize=24)
plt.xlabel('AUC values across test fold',fontsize=24)
sns.distplot(a=LOO_AUCs,kde=False,bins=20)
```

Appendix C: Tensorboard

Shown below in Figure C.1 is the Tensorboard application monitoring the training of a particular MLP architecture.



☐ Show data download links
☑ Ignore outliers in chart scaling

Tooltip sorting method: default

Smoothing 0

Horizontal Axis [✓ STEP] RELATIVE WALL

Runs

Write a regex to filter runs

☐ 40_fold_eval_03_15-18_16_29\Fold_38\validation
☑ 40_fold_eval_03_15-18_16_29\Fold_39\train
☑ 40_fold_eval_03_15-18_16_29\Fold_39\validation
☐ 40_fold_eval_03_15-18_16_29\Fold_40\train
☐ 40_fold_eval_03_15-18_16_29\Fold_40\validation
☐ 40_fold_eval_03_15-18_41_22\Fold_1\train
☐ 40_fold_eval_03_15-18_41_22\Fold_1\validation
☐ 40_fold_eval_03_15-18_41_22\Fold_2\tr

./MLP_HRV_feats_TB_logs

Figure C.1: Tensorboard for neural network training.

Appendix D: Logbook

12th October – 18th October

Objectives:

- ☑ Read Chapter 3 (Biomedical Signal Processing and ML) from Oksana's thesis.
- ☑ Read Oksana's published paper *"Prediction of short-term health outcomes in preterm neonates from heart-rate variability and blood pressure using boosted decision trees"*.
- ☑ Read Chapters 1, 2, 3 and 6 from Rehan's thesis.

Summary:

- Read the thesis chapter a number of times and took key notes down for the lab notebook.
    - Paid particular attention to analysis of 13 HRV features used in the thesis – these include time domain (e.g. MeanRR, SDNN) and frequency domain (e.g. Power spectral density in VLF, LF and HF frequency bands) features.
    - Gained insight into initial processing of raw ECG data – Pan-Tompkins method used for R-peak detection (and the reason this particular algorithm was chosen over others).
    - Background in various ML models (e.g. neural networks, SVMs) and classifier performance metrics (e.g. ROC, AUC).
    - Noted down a number of references related to the work in my project – ML papers, links between HIE and HRV, etc.
- Read through the published paper a number of times, noting down explanations for terms I didn't understand.
    - Helped understand the thesis chapter better – served as a good introduction to the thesis topic.
    - Noted down any relevant references in lab notebook.
- Read Chapters 1 and 2 from Rehan's thesis, taking down key points in the lab notebook.
    - Chapter 1 gave an introduction to the thesis topic, including a literature review of previous work carried out in the area of automated seizure detection and HIE grading. It also set out the topics/content of the thesis.
    - Chapter 2 detailed the medical background around HIE and seizures in neonates, including the methods currently used to assess the severity of the seizure or injury in the neonate.
- Installed anaconda on PC (package that contains relevant python libraries + Jupyter notebooks).
- Joined private Github repository set up by Daragh.

Goals for next week:

- Read Chapters 3 and 6 from Rehan's thesis.
- Read research papers relating to HIE and HRV referenced by Oksana and Rehan (recorded in lab notebook).

19th October – 25th October

Objectives:

- ☑ Rehan's Thesis (Chapters 3, 6)
- ☑ Continue with other related papers – e.g. Pan Tompkins method, and background on HIE & HRV link.

Summary:

- Read and took notes from Chapter 3 "From Static to Dynamic Classification" of Rehan's thesis.
    - The chapter outlined Rehan's approach to training a classifier to grade HIE in neonates using HRV features.

  - It was argued that it was vital to capture the long term, contextual information across multiple epochs in order to obtain the best performance from the classifier, as the HRV has subtle variations over time as a result of the HIE injury.
  - A number of dynamic classifier models were presented that could exploit this contextual information, as a simple static classifier could not. These models were sequential kernels for an SVM, e.g. supervector and Fisher kernels.
  - As seen later in Chapter 6, the following system was ultimately decided on: the HRV features were extracted from the Pan-Tompkins detection algorithm, these features were input to a Gaussian Mixture Model (GMM), from which the supervectors could be generated. These supervectors were then input to an SVM. A sigmoid function was then used to determine the probabilities of each class from the SVM output.
- Completed and submitted the EE4050 Engineering Ethics assignment.
- Read and took notes from Chapter 6 "Classification of HIE using heart rate variability" of Rehan's thesis.
  - This chapter outlined the features used in the dynamic classifier.
  - The features used were similar to Oksana's with a couple of differences: MeanNN, SDNN, TINN, skewness, kurtosis, entropy, 1st derivative of SDNN, 2nd derivative of SDNN, power in VLF, LF and HF bands and ratio LF/HF.
  - The normalised histograms of each feature for both classes (none/mild and moderate-severe HIE) were plotted to show the discriminatory power of the features.
  - The system using the GMM and supervector was tested with a number of parameters and features, with a subset of the features + the mean NN interval giving the best performance.
  - It was difficult to find some information in the chapter such as the best feature extraction window size.
  - From the results of the classifier performance, it seems the results could've been improved with better quality ECG data. This study was mainly focused on the use of EEG data, and so the quality of the ECG data was not consistent.
- Read and took notes on *R.M. Goulding, N.J. Stevenson, D.M. Murray, V. Livingstone, P.M. Filan, G.B. Boylan, Heart rate variability in hypoxic ischemic encephalopathy: correlation with EEG grade and 2-y neurodevelopmental outcome, Pediatr. Res. 77 (2015) 681–687, doi:10.1038/pr.2015.28.*
  - This paper establishes the link between HIE and HRV.
  - A subset of HRV features were examined: the mean NN interval, SDNN, TINN, power in VLF, LF and HF bands and ratio LF/HF.
  - A statistically significant negative correlation between the features and grade of HIE was discovered.
  - The effects of some treatments such as morphine and Phenobarbitone were taken into account in the study, finding that morphine did not have an effect on the correlation between the HRV features and the grade of HIE, while Phenobarbitone eliminated the correlation for the TINN and LF features.
  - For the calculation of the features, each ECG file was broken into 5 minute epochs, and the corresponding HRV features were extracted from the NN intervals in each epoch. The mean of each HRV feature was calculated across all available epochs for that particular HIE grade (I.e. if the grade was mild for the first 10 epochs then the mild grade features would be calculated across these 10 epochs). This mean was then used in the statistical analysis carried out.
  - The study also looked at the predictive power of the features in determining the 2 year neurodevelopmental outcome of the neonate.
- Read and took notes on *Aliefendioglu, D., Dogru, T., Albayrak, M., Dibekmısırlıoglu, E., & Sanli, C. (2012). Heart rate variability in neonates with hypoxic ischemic encephalopathy. Indian Journal of Pediatrics, 79(11):1468–1472.*
  - This paper again examined the link between a subset of HRV features and grade of HIE. In this study, the LF and HF power was examined, as well as the ratio LF/HF.

  - The results of this paper partially contradict that of R.M. Goulding et al – while the LF power decreased with increasing grade of HIE, it was found that the HF power increased with grade of HIE (I.e. positive correlation).
  - However, the method of collecting the data differed. The ECG data was collected **1 week after birth**, with some neonates having received treatment (e.g. a ventilator) during that week. This is not useful for our investigation, as It's necessary to look at the HRV in the neonate in the 6 hours after birth so that treatment can be administered if required. It was further noted in the paper that HF activity increases over the first 7 days of life in newborns, which could account for the previously mentioned positive correlation.

Objectives for next week:

- Read *Pan, J. & Tompkins, W. J. (1985).* ***A real-time QRS detection algorithm****. IEEE Transactions on Biomedical Engineering, 32(3):230–236.*
- Read *V. Matic, ´ P.J. Cherian, D. Widjaja, K. Jansen, G. Naulaers, S. Van Huffel, M. De Vos,* ***Heart rate variability in newborns with hypoxic brain injury****, Adv. Exp. Med. Biol. 789 (2013) 43–48, doi:10.1007/978-1-4614-7411-1_7.*
- Read *Doyle, O., Temko, A., Marnane, W., Lightbody, G., & Boylan, G. (2010).* ***Heart rate based automatic seizure detection in the newborn.*** *Medical Engineering & Physics, 32(8):829–839.* - Look at some of the HRV features in greater detail.
- Read *A. Temko, O. Doyle, D. Murray, G. Lightbody, G. Boylan, W. Marnane,* ***Multimodal predictor of neurodevelopmental outcome in newborns with hypoxicischaemic encephalopathy****, Comput. Biol. Med. 63 (2015) 169–177, doi:10.1016/ j.compbiomed.2015.05.017.*
- Start playing around with HRV feature extraction tool on MATLAB.

26th October – 1st November

Objectives:

☑ Read *Pan, J. & Tompkins, W. J. (1985).* ***A real-time QRS detection algorithm****. IEEE Transactions on Biomedical Engineering, 32(3):230–236.*
☑ Read *V. Matic, ´ P.J. Cherian, D. Widjaja, K. Jansen, G. Naulaers, S. Van Huffel, M. De Vos,* ***Heart rate variability in newborns with hypoxic brain injury****, Adv. Exp. Med. Biol. 789 (2013) 43–48, doi:10.1007/978-1-4614-7411-1_7.*
☑ Read *Doyle, O., Temko, A., Marnane, W., Lightbody, G., & Boylan, G. (2010).* ***Heart rate based automatic seizure detection in the newborn.*** *Medical Engineering & Physics, 32(8):829–839.* - Look at some of the HRV features in greater detail.
☑ Read *A. Temko, O. Doyle, D. Murray, G. Lightbody, G. Boylan, W. Marnane,* ***Multimodal predictor of neurodevelopmental outcome in newborns with hypoxicischaemic encephalopathy****, Comput. Biol. Med. 63 (2015) 169–177, doi:10.1016/ j.compbiomed.2015.05.017.*
☑ Start playing around with HRV feature extraction tool on MATLAB.

Summary:

- Read and took notes from *Pan, J. & Tompkins, W. J. (1985).* ***A real-time QRS detection algorithm****. IEEE Transactions on Biomedical Engineering, 32(3):230–236.*
  - This paper outlines the details and implementation of the Pan Tompkins QRS detection algorithm.
  - The algorithm processes the raw ECG as follows: a bandpass filter (consisting of a cascaded LPF and HPF) is applied to the raw ECG to attenuate the noise present. The filtered signal is then differentiated and squared. A moving average window is applied to the differentiated and squared signal. Finally, an adaptive threshold is applied to detect the QRS complexes.
  - On start-up, the algorithm actually has 2 learning phases, the first to initialise the adaptive thresholds and the second to record values for the mean RR interval and the

    maximum/minimum limits on the RR interval range. The detection phase starts after this.
  - The algorithm uses dual thresholding, which allows (initially) missed R-peaks to be detected. When an R-peak has not been detected for a time interval corresponding to 166% of the current mean RR interval, the largest peak in that time that's greater than the lower threshold is taken as a candidate for the QRS complex.
  - These thresholds are continuously adapted based on the signal and noise peaks present on the signal.
  - After each QRS complex there is a 200 ms refractory period where another QRS complex cannot physically occur. This eliminates the risk of the same QRS complex being detected multiple times.
  - When a QRS complex is detected after the 200 ms refractory period but within 360 ms of the previously detected QRS complex, it must be determined whether this is a T wave or a QRS complex. In this case, the detection with the greatest slope is taken as the QRS complex.
  - The mean RR interval is updated every heartbeat.
  - The remainder of the paper dealt with the actual implementation of the algorithm and its test results, which I read but didn't take any notes on.

- Read and took notes from *V. Matic, ´ P.J. Cherian, D. Widjaja, K. Jansen, G. Naulaers, S. Van Huffel, M. De Vos,* ***Heart rate variability in newborns with hypoxic brain injury****, Adv. Exp. Med. Biol. 789 (2013) 43–48, doi:10.1007/978-1-4614-7411-1_7.*
  - This paper was similar to the R.M. Goulding paper, where the predictive power of various HRV features in grading HIE was assessed.
  - The dataset consisted of 36 2-hour ECG segments recorded within 18 to 48 hours after birth.
  - The dataset was split into 2 labels: mild grade of HIE, and moderate/severe grade of HIE. The grade of HIE for each ECG segment was determined using Sarnat scores and assessment of the background EEG for that particular time interval.
  - ECG signal was upsampled from 256 to 1000 Hz before the Pan-Tompkins algorithm was applied. The results of the Pan-Tompkins algorithm were visually inspected to ensure no R-peaks were missed (or that there were no false positives). Various HRV features were then extracted from the signal.
  - Time domain HRV features were:
    - Mean NN interval.
    - Standard Deviation of NN interval (SDNN) - represents the total power of the HRV spectra. SDNN increases with the duration of ECG segments, therefore only the SDNN between similar duration ECG segments could be used for comparisons.
    - SDANN is the standard deviation of NN intervals calculated over 5 minute ECG epochs. This parameter was actually used to calculate SDNN – mean of SDANNs over 2 hours taken as SDNN.
    - RMSSD is the Root Mean Squared Difference between successive NN intervals.
    - pNN25 represents the number of successive NN intervals with a duration differing by >= 25 ms, divided by the total number of NN intervals and multiplied by 100.
    - SDSD is the standard deviation of differences between successive or adjacent NN intervals.
  - Geometric features were:
    - TINN
    - SD1 and SD2 of Poincaré plots.
  - A uniformly sampled signal was obtained by interpolating the NN intervals with a 4 Hz sampling frequency. Frequency domain features were:
    - VLF spectral power (0.01 - 0.04 Hz)

        - LF spectral power (0.04 - 0.2 Hz)
        - HF spectral power (0.2 - 2 Hz)
    - These features were taken from: *Task Force of the European Society of Cardiology and the North American Society of Pacing and Electrophysiology (1996) Heart rate variability: standards of measurement, physiological interpretation and clinical use. Circulation 93(5):1043–1065 .*
    - These features were used as inputs to an LDA classifier.
    - Leave One Out method was used to assess the performance of the classifier.
    - The best subset of features achieved an accuracy of 80%.
    - Noted that SDNN, LF power and SD2 of Poincaré plot were the best performing features.

- Completed and submitted Health and Safety risk assessment.

Objectives for next week:

- Read *Doyle, O., Temko, A., Marnane, W., Lightbody, G., & Boylan, G. (2010).* ***Heart rate based automatic seizure detection in the newborn.*** *Medical Engineering & Physics, 32(8):829–839. -* Look at some of the HRV features in greater detail.
- Read *A. Temko, O. Doyle, D. Murray, G. Lightbody, G. Boylan, W. Marnane,* ***Multimodal predictor of neurodevelopmental outcome in newborns with hypoxicischaemic encephalopathy****, Comput. Biol. Med. 63 (2015) 169–177, doi:10.1016/ j.compbiomed.2015.05.017.*
- Start playing around with HRV feature extraction tool on MATLAB.

2nd November – 8th November

Objectives:

☑ Read *Doyle, O., Temko, A., Marnane, W., Lightbody, G., & Boylan, G. (2010).* ***Heart rate based automatic seizure detection in the newborn.*** *Medical Engineering & Physics, 32(8):829–839. -* Look at some of the HRV features in greater detail.

☑ Read *A. Temko, O. Doyle, D. Murray, G. Lightbody, G. Boylan, W. Marnane,* ***Multimodal predictor of neurodevelopmental outcome in newborns with hypoxicischaemic encephalopathy****, Comput. Biol. Med. 63 (2015) 169–177, doi:10.1016/ j.compbiomed.2015.05.017.*

☐ Start playing around with HRV feature extraction tool on MATLAB.

☑ Look at distribution of features in hrv_feat_set_v1.csv.

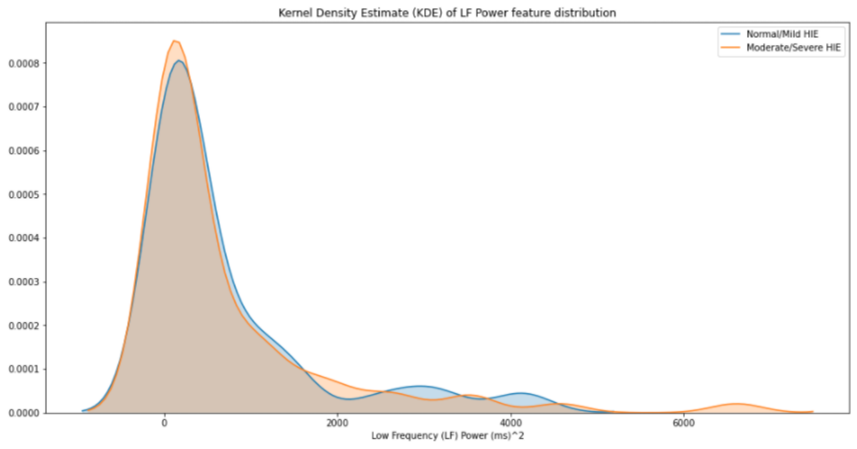
Summary:

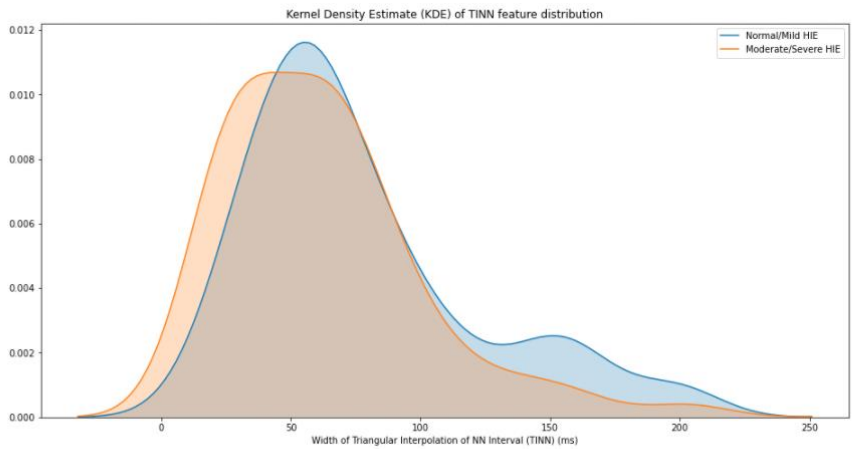
- Read and took notes on *Doyle, O., Temko, A., Marnane, W., Lightbody, G., & Boylan, G. (2010).* ***Heart rate based automatic seizure detection in the newborn.*** *Medical Engineering & Physics, 32(8):829–839.*
    - This paper attempted to develop an automated seizure detection system using heart rate variability. The dataset consisted of 208 hrs of ECG recordings from 14 neonates at a sampling rate of 256 Hz. R-peak detection was applied and the ECG recordings were segmented into 60 second epochs. 62 time and frequency domain heart rate variability features were extracted using a feature extraction window length of 60 seconds – feature set contained standard features like MeanNN, SDNN and power in frequency sub bands, but also contained more advanced features such as Shannon entropy and Allan factor. The SVM hyperparameters were tuned using 5 fold cross-validation and LOO (unbiased estimate of general error) was used to assess the performance of the classifier. Feature selection was performed by selectively removing the features with the smallest weights (I.e smallest influence on prediction). This actually degraded the performance. The AUC of the system in the end was 0.59 across all patients. - there was high inter-patient variability in the data used.

- Read and took notes on *A. Temko, O. Doyle, D. Murray, G. Lightbody, G. Boylan, W. Marnane,* ***Multimodal predictor of neurodevelopmental outcome in newborns with hypoxicischaemic encephalopathy****, Comput. Biol. Med. 63 (2015) 169–177, doi:10.1016/ j.compbiomed.2015.05.017.*
    - This paper developed a multimodal system to predict the neurodevelopmental outcome of neonates diagnosed with HIE. The dataset consisted of 1 hr ECG and EEG recordings extracted from 38 different neonates – the neonates were not treated with therapeutic hypothermia. R-peak detection was performed on the ECG and the recording was segmented into 60 second epochs. A 60 second feature extraction window was used on the ECG signal. In total, 120 features were extracted (60 from ECG and 60 from EEG), with a number of clinical metrics used as features in addition. The 60 ECG features again contained standard HRV parameters (e.g. MeanNN and SDNN), as well as more complex ones (e.g. Shannon Entropy). LOO was used to assess the performance of the SVM used, with recursive feature elimination being used at each LOO iteration – RFE seeks to eliminate the features which have the smallest impact on the cost function. The most common features across the top N performing features in each iteration were selected for the final evaluation of the system. Only 2 ECG features were selected by RFE, the Shannon Entropy and the power in the HF band. The remaining features were from the EEG signal (RFE was not applied to clinical assessment features as these were treated as categorical labels). The final AUC was approximately 86%. It's noted in the paper that the LOO error does not estimate the error that may arise from using a different ECG or EEG recording device, and also notes that a different system would need to be developed for neonates treated with therapeutic hypothermia, as this would have a significant impact on EEG and HR.

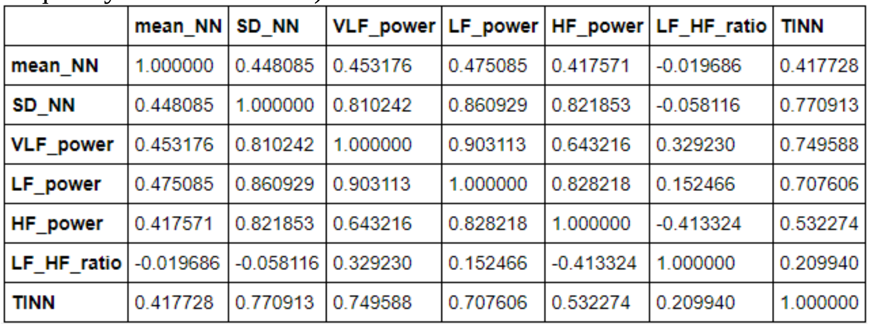
- Completed Kaggle courses on pandas and data visualization (seaborn and matplotlib) as a refresher.

- Looked at the distributions of data in hrv_feat_set_v1.csv.
    - Created a Jupyter notebook and imported the data – plotted histograms and KDE for each feature, highlighting the differences between the 2 classes. For example, shown below are the KDE plots for the LF power and the TINN feature, which have been shown to be discriminative features in previous papers:





- There is significant overlap between the 2 features shown here, and this overlap was repeated for pretty much all the features. The rest of the plots can be seen in the notebook Mean Features V1 Analysis.
- Finally, the correlation matrix for the features was computed (using the log-transformed frequency domain features):



	mean_NN	SD_NN	VLF_power	LF_power	HF_power	LF_HF_ratio	TINN
mean_NN	1.000000	0.448085	0.453176	0.475085	0.417571	-0.019686	0.417728
SD_NN	0.448085	1.000000	0.810242	0.860929	0.821853	-0.058116	0.770913
VLF_power	0.453176	0.810242	1.000000	0.903113	0.643216	0.329230	0.749588
LF_power	0.475085	0.860929	0.903113	1.000000	0.828218	0.152466	0.707606
HF_power	0.417571	0.821853	0.643216	0.828218	1.000000	-0.413324	0.532274
LF_HF_ratio	-0.019686	-0.058116	0.329230	0.152466	-0.413324	1.000000	0.209940
TINN	0.417728	0.770913	0.749588	0.707606	0.532274	0.209940	1.000000

- There is a noticeable correlation present between some of the features. All frequency domain features have some correlation, which is to be expected. In particular, SD_NN, the standard deviation of the NN interval has a significant correlation with the frequency domain features.

Objectives for next week:

- Fit a simple classifier (e.g. linear SVM) to the data in hrv_feat_set_v1.csv and test the performance of various combinations of features – good performance is not expected (due to the poor discrimination between classes shown by the distributions) but it could be useful to start working with scikit learn and create some generic code/functions that could be used for the real data.
- Continue reading papers – focused more on machine learning.

9th November – 15th November

Objectives:

☑ Continue reading papers – focused more on machine learning.
☑ Inspect real data.
☑ Run Oksana's feature extraction script.
☐ Fit a simple classifier (e.g. linear SVM) to the data in hrv_feat_set_v1.csv and test the performance of various combinations of features

Summary:

- Had a meeting with Daragh on Sunday evening.

- Set out the plan for the week – extract the 7 features included in John O'Toole's script from the data, analyse the correlation with the labels/their importance, then fit a linear SVM using the selected features.
- Manually inspected the data – there appears to be a number of NaNs in the NN interval data. We don't really understand where these came from as there are no NaN's in the corresponding R-peak data.
- Inspected and tried out Oksana's code for the feature extraction. A number of NaNs are generated in the feature vector, even after the removal of the NaNs in the data – we think it's related to the fs_ecg and window length used to generate the features.

Objectives for next week:

- Extract 7 simple features from the data using John O'Toole's script with various window lengths and overlaps.
- Look at the correlation of the features with the labels and the discrimination between the classes using the features.
- Gridsearch for the best hyperparameters for a linear SVM.
- Fit and test the linear SVM using LOO.
- Continue reading about feature selection techniques.

16th November – 22nd November

Objectives:

☑ Extract 7 simple features from the data using John O'Toole's script with various window lengths and overlaps.
☑ Look at the correlation of the features with the labels and the discrimination between the classes using the features. **I carried out RFE, Daragh carried out this – filter based feature selection.**
☑ Gridsearch for the best hyperparameters for a linear SVM.
☑ Fit and test the linear SVM using LOO. **Ignoring LOO for now.**
☐ Continue reading about feature selection techniques.

Summary:

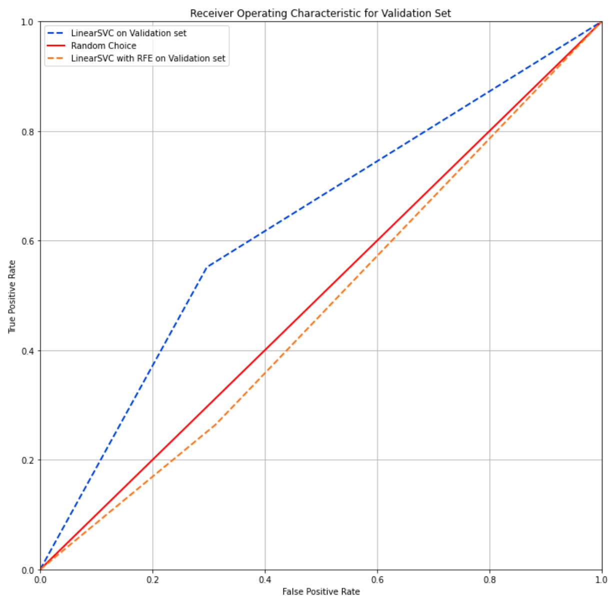
- Worked on a Python function to generate a list of Baby_IDs to be used as an index for each baby.
    - 7 features generated per epoch per baby by John O'Toole's script does not contain an index identifying which baby the epoch comes from.
    - The function iterates over the column 'start_time_secs' and assigns a baby_ID to a baby until it reaches a value of 0 s, which indicates the start of the recording for a new baby. The function then increments the baby_ID by 1, and assigns the new ID to each epoch until it reaches another value of 0 s.
    - This results in an index from 1 to 120 being assigned to each baby.
    - Uploaded the function to Github for Daragh to use.

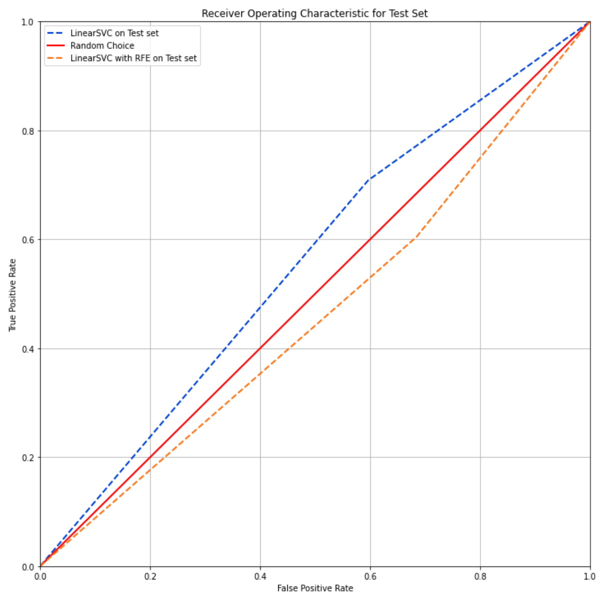
- Prepared the dataset for use in RFE
    - Changed column heading to EEG_grade (rather than baby_ID, as generated by John's script).
    - Appended the array of Baby_IDs to the data – this allows the manual creation of a test and training set.

- Worked on RFE with a linear SVM classifier, and a benchmark linear SVM classifier using all 7 features to compare the efficacy of RFE.
    - Split the 120 babies into a training set (96 babies), validation set (12 babies) and test set (12 babies). This was done manually by randomly shuffling the indices from 1 to 120, to ensure no epochs from a baby in the test or validation set ended up in the training set (or vice versa).
    - Decided on a (natural) log transform followed by standardization to force each feature down to a similar scale and with an approximately normal distribution - features with very different

scales can negatively impact the performance of the SVM - difficulties fitting the margin due to the different scales.

- Performed cross validation on the training set with 5 folds (cv = 5) for a Linear SVM classifier with all 7 features included. Best hyperparameter value for C = 1.0, which achieved an AUC of 0.755 during cross validation. Testing this on the validation set yielded an AUC of 0.63 and an accuracy of 62% (base rate of validation set = 58%).
- Used scikit-learn's RFECV class to perform recursive feature elimination and automatically select the best number of features using the training set with a Linear SVM classifier (C=1.0). The best performance (only by a slight margin) was achieved with 2 features - meanNN and LF_power. It assigned the following ranks to all 7 features:
  - mean_NN rank = 1
  - SD_NN rank = 2
  - VLF_power rank = 6
  - LF_power rank = 1
  - HF_power rank = 5
  - LF_HF_ratio rank = 4
  - TINN rank = 3
- The best performance of RFECV during cross validation on the training set was 0.757 (AUC). Transforming the validation set with the RFECV transformer and testing the linear SVC yielded an AUC of 0.48 and an accuracy of 44%. Finally, on the test set the LinearSVC with 7 features achieved an AUC of 0.56 with an accuracy of 59%. The base rate of the test set was 62%, so this actually performed worse than just selecting 1 for every instance. The LinearSVC with the 2 features selected by RFE achieved an accuracy of 50% on the test set with an AUC of 0.46 - very close to random.





- If I were to guess the reason why the performance dropped for both classifiers on the test set, I'd think it could be attributed to the imbalance of classes in the test set vs the validation and training set. The test set had almost double the number of positive classes (96) vs negative classes (57), whereas the validation set had 87 instances of the positive class and 64 of the negative. These distributions can be seen in the histogram at the start of the notebook.

Objectives for next week:

- Get Oksana's feature extraction script working.
- Read some more ML papers (e.g. feature selection) – falling behind on this.
- Continue reading Hands On Machine Learning book – good practical information in this.

23rd November – 29th November

To do:

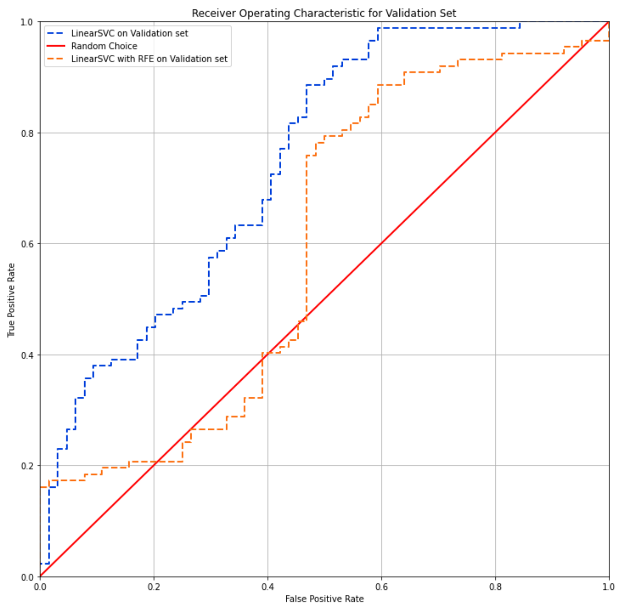
☐ Get Oksana's feature extraction script up and running.
☑ Correct the AUC calculations and ROC curves from last week.
☑ Plot the ROC and obtain AUC for each of the 7 features.
☑ Read Chapter 3 Hands On ML.
☐ Read paper on RFE (for report).

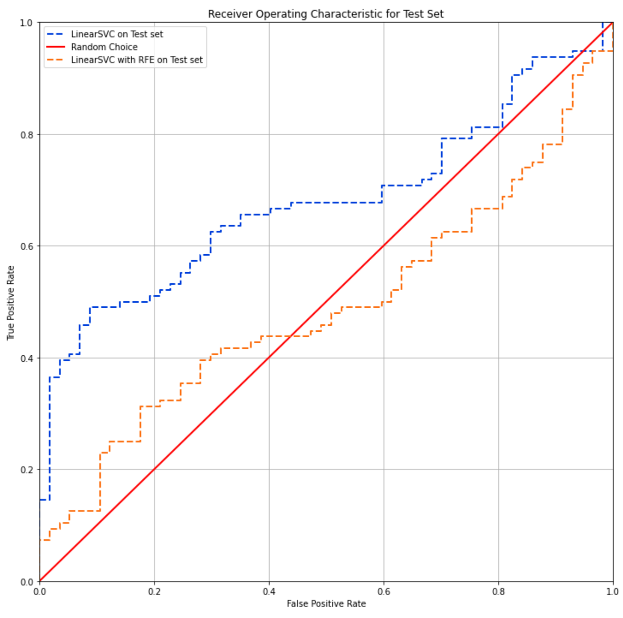
Summary:

- **Corrected notebook from last week for the SVM with 7 features and the SVM with RFE – AUC and ROC curves.**
  - Previously I was using the predictions from the SVM with 7 features and the SVM with RFE to calculate the AUC and create the ROC using scikit-learn. After reviewing the documentation, I realised that it's actually the scores from the decision function of the classifiers that are needed.
  - The new results are summarised in the table below. Note that the accuracy did not change as this function was used correctly.

	Performance Metric	**Training Set (cross-validation)**	**Validation Set**	**Test Set**
SVM (7 features)	**AUC**	0.755	0.75	0.68
	Accuracy	-	62%	59%
SVM RFE (2 features)	**AUC**	0.757	0.6	0.49
	Accuracy	-	44%	50%

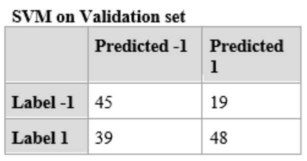
  - The correct ROC for both classifiers on the Validation and Test set are shown here:





  - Note that the features returned by RFE did not change, as AUC was automatically calculated correctly by the RFECV function.
  - There was still a very noticeable drop in performance on the test set, which I originally attributed to the imbalance in classes – nearly double the amount of positive classes vs negative classes in the test set. When correcting the notebook, I investigated this by

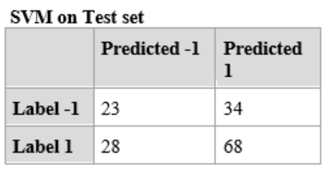
calculating the confusion matrix for the validation and test sets. The confusion matrix for the SVM with 7 features on the validation set is shown here:



SVM on Validation set

	Predicted -1	Predicted 1
Label -1	45	19
Label 1	39	48

It's clear that the classifier struggles when presented with the positive class, only guessing correctly 55% of the time. On the other hand, it's a lot surer when presented with a negative class, guessing correctly 70% of the time. The confusion matrix on the test set is:

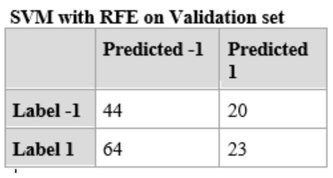


SVM on Test set

	Predicted -1	Predicted 1
Label -1	23	34
Label 1	28	68

While the % of correctly detected positive classes increases to 71%, with a drop in correctly detected negative classes to 40%, I think this can be attributed to the imbalance in classes – with nearly double the amount of positive instances, it makes sense that the number of positive predictions will increase. Conversely, the number of negative predictions will go down due to the smaller number of negative instances.

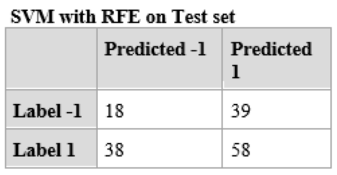
For the SVM with RFE, the confusion matrix for the validation set is shown here:



SVM with RFE on Validation set

	Predicted -1	Predicted 1
Label -1	44	20
Label 1	64	23

It only guesses the positive class correctly 26% of the time, and the negative class 69% of the time. On the test set, the same pattern as with the base SVM using 7 features is seen again:

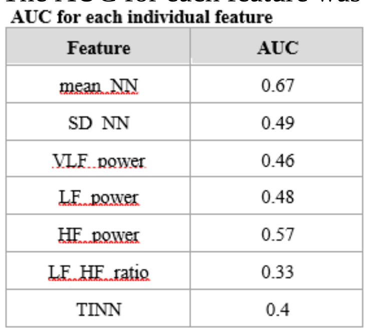


SVM with RFE on Test set

	Predicted -1	Predicted 1
Label -1	18	39
Label 1	38	58

It's clear that both classifiers seriously struggle to detect the positive class, which indicates that we need more discriminative features.

- **Calculated the AUC for each feature.**
  - The AUC for each feature was also calculated, shown here:



AUC for each individual feature

Feature	AUC
mean_NN	0.67
SD_NN	0.49
VLF_power	0.46
LF_power	0.48
HF_power	0.57
LF_HF_ratio	0.33
TINN	0.4

- **Worked on Oksana's script.**
  - Set up a MATLAB script to extract a feature vector for all 120 babies.
  - **It extracts the mean of the features over the entire recording.**
  - Plotted some histograms for the features but it's hard to get a sense of whether they're correct with so few instances.
  - The meanNN is still the only feature that corresponds to John's feature extraction script. By changing the calculation for SD_NN to use the rr interval instead of NN nterval (which John appears to do), I can get an answer similar to John's for SD_NN (23 from Oksana's script, 18 from John's).

Objectives for next week:

- Get Oksana's feature extraction script working. **Priority**
  - **Need to get some analysis done with these features –** whether that's some univariate filter based analysis, looking at the distributions, performing RFE or just using all the features to compare the AUC of a linear SVC classifier vs the classifier using 7 features.
- Start going over notes on papers from Lab notebook for literature review.
- Start writing literature review for report.

30th November – 6th December

To do:

☑ Get Oksana's feature extraction script working. **Priority**
Need to get some analysis done with these features – whether that's some univariate filter based analysis, looking at the distributions, performing RFE or just using all the features to compare the AUC of a linear SVC classifier vs the classifier using 7 features.
☑ Start going over notes on papers from Lab notebook for literature review.
☑ Start writing literature review for report.

Summary:

- Worked with Daragh to get the feature extraction script working.
  - Developed a MATLAB script to extract the extended HRV features from the data.
  - Extracted a dataset using 5 minute epochs and 0% overlap.
  - Numbers in the new dataset look a bit different for the frequency domain features – e.g. HF_power has a greater magnitude on average than in the dataset extracted using John's script.
  - Didn't do any further work with the new features but the script is at least up and running for next semester.

- Worked on preliminary report.
  - Read over lab notebook for notes on previous papers.
  - Read through a select few papers e.g. Chapters 1, 2 and 3 from Rehan's thesis, R.M. Goulding's paper.
  - Completed just under 6 pages of the report, covering the introduction and background. Background not finished and covers Pan-Tompkins, the 7 HRV features used so far and the theory behind the SVM. Planning on including the ROC curve and the AUC metric.

Objectives for next week:

- Finish preliminary report.

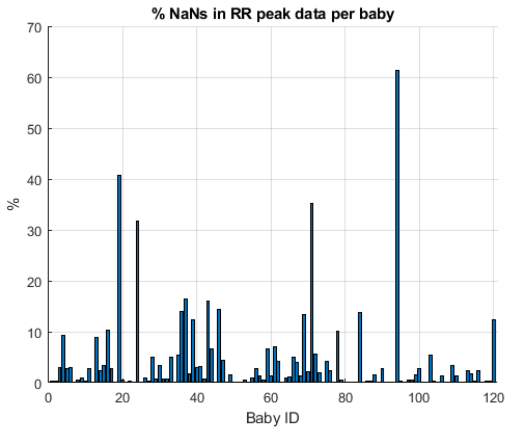
25th January – 31st January

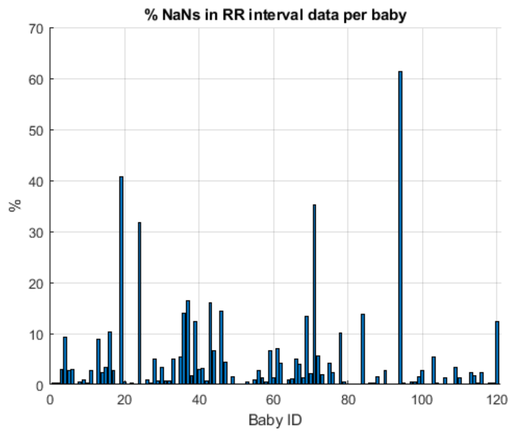
Summary:

- Read up on Power Spectral Density:
    - **Short-time Fourier Transform (STFFT):** Looks at short windows of the signal rather than the entire signal at once. The segmented epoch of the signal is attenuated/tapered at the edges, and the FFT is taken. The power/amplitude spectrum is calculated (y axis = power, x axis = frequency). Power spectrum is plot in the time-frequency domain, where the y axis = frequency, x axis = time and the colour of the bar indicates the power of the signal at each frequency.
      STFFT is similar to Welch PSD but plots each window in the time frequency domain rather than averaging each window over the entire signal.
      Width of window & amount of overlap are 2 parameters – too short a window, and there'll be few datapoints/frequencies between 0 and Nyquist. Temporal precision decreases with increasing window size. Can use a method that varies the window size depending on the frequencies present at that time point.
      Preserves the temporal dynamics of entire signal.

    - **Welch's PSD:** Gives a smoother result than Fourier Transform (easier to interpret) and more robust to signal non-stationarities.
      The full FFT is one FT over the entire signal – gives no information about where frequency 'bursts' occur in time – did it occur in 1 short window or did it happen over the entire signal duration? I.e. No information about temporal dynamics.
      Welch's method performs an FFT on a small window/epoch of the signal – size of epoch or window is a parameter of Welch's method. The edges of the window are tapered. The power spectrum of that window is then computed. This process is repeated as the window 'slides' along the signal – another parameter of Welch's method is the amount of overlap in successive windows.
      The result is that Welch's method smooths out the power spectrum.
      Different taper profiles available – e.g. Hann, Hamming, Gaussian taper. The edges of the window cause artefacts in the FT – strong non-stationarities that require a lot of energy/frequencies for the FT to represent. The taper minimises the amount of edge effects contaminating the power spectrum. This attenuates valid data at the edges of the window – therefore, some overlap in windows is important.
      Full FFT is sensitive to noise and non-stationarities. Offers maximal spectral precision – as for a fixed sampling frequency, the number of timepoints determines the frequency resolution -> FT over entire signal means more timepoints, meaning greater spectral precision. No temporal dynamics present.
      Welch's method smooths over non-systematic noise, is robust to non-stationarities, but offers reduced spectral precision compared to FT (shorter windows = less timepoints = decreased frequency resolution). No temporal dynamics present.

- Besides the fact that the values for the 5min epochs look very different to the same features generated using John's script, Oksana's script crashes when the epoch length is made smaller, due to issues with NaNs for certain babies.
    - While investigating this I looked into the % of NaNs in the data for each baby, and generated the following histograms:





Objectives for next week:

- Examine + try fix our feature extraction script for the 15 features.

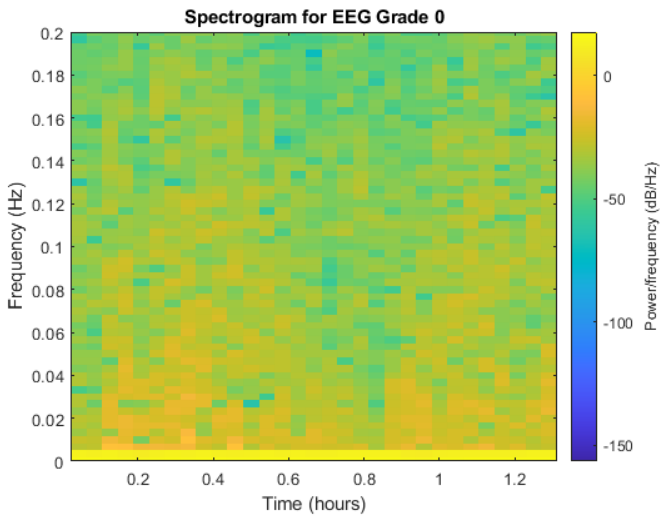
1st February – 7th February

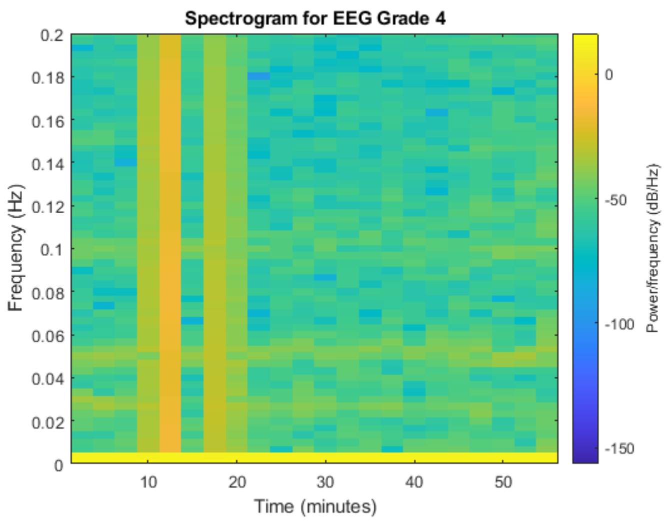
To do:

☑ Examine + try and fix our feature extraction script for the 15 features.

Summary:

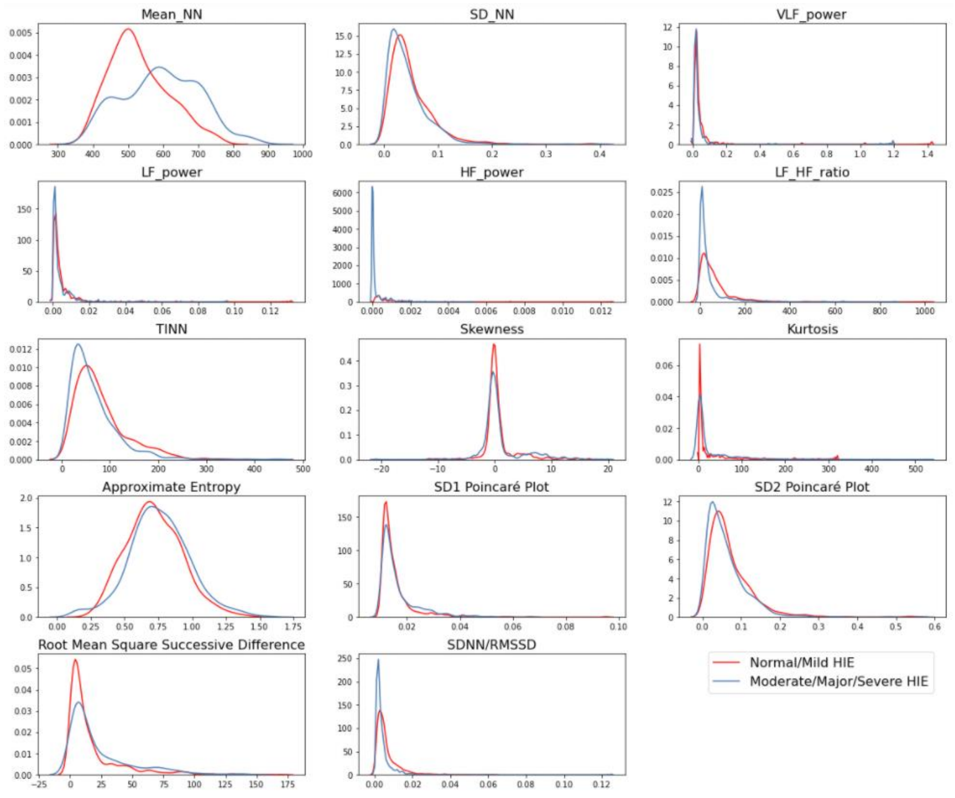
- Produced the spectrogram for 2 babies of EEG grade 0 and EEG grade 4. As expected, there seems to be a greater distribution of the power over a large number of frequencies for grade 0, whereas the power for the grade 4 baby is constrained to a few narrow frequency bands. The spectrograms are shown below. Note that the solid bar in the spectrogram of the grade 4 baby occurs due to an outlier in the RR intervals which appears as a large pulse in the plot of RR intervals vs time (outlier could be from a missed R peak as it's approximately twice the value of the other RR intervals).





- Worked with Daragh to get the extended feature extraction script working correctly:

- We stitched Oksana's feature extraction code and John O'Toole's code together, which allows us to get the robust NaN handling of John's code while also generating the extra 7 features from Oksana's script.
- We no longer need to discard the data for entire babies – instead, we edited John's script to discard any epoch with greater than 30% NaNs. These epochs will be analogous to 'U' or 'Undecided'.
- We edited John's script to use the NN intervals for the corresponding features (e.g. meanNN, SDNN) - it was using the raw RR intervals previous to this. Rehan and Oksana defined the NN intervals in their theses as the resampled (@256 Hz) RR intervals.
- We also finally figured out why Oksana's script was dividing the NN intervals by the meanNN interval for that epoch when generating certain features. In Oksana's paper, she referenced another paper (https://www.frontiersin.org/articles/10.3389/fphys.2013.00306/full) which argues that you should divide by the mean RR interval when calculating HRV features, as they will be heavily influenced by the meanRR. We divided the NN intervals by the meanNN for the SDNN, frequency domain, Skewness, Kurtosis and Approximate Entropy features. This (for the most part) slightly increased their AUCs, while also reduced the correlation of SDNN & frequency domain features with the meanNN (compared to before Christmas).
- We could not get the Allan Factor to generate sensible results – according to Oksana's thesis, the AF should generate a number between 0 and 1, with 0 indicating periodic, predictable signals, and 1 indicating random signals. This would imply higher EEG grades have a lower AF, and vice versa for lower EEG grades. However, this was not the case – the values seem to be totally random. We triple checked the function and it does what it's supposed to. It also generates NaNs for epochs where no other feature does. In the end we decided to drop the feature, as removing all the epochs with NaNs in the AF would delete too much good data.
- The new feature distributions are shown here:



- Finally, the new individual feature AUCs are:

```
Area Under Curve of mean_NN = 0.6740072163166537
Area Under Curve of SD_NN = 0.5652331259182439
Area Under Curve of VLF_power = 0.5880577881485323
Area Under Curve of LF_power = 0.5636505055742805
Area Under Curve of HF_power = 0.5428639846743295
Area Under Curve of LF_HF_ratio = 0.6739748077089275
Area Under Curve of TINN = 0.6005639097744361
Area Under Curve of Skewness = 0.5332080200501252
Area Under Curve of Kurtosis = 0.5836412151067323
Area Under Curve of ApEn = 0.5580330135684037
Area Under Curve of SD1_Poincare = 0.5376209921355112
Area Under Curve of SD2_Poincare = 0.568002261400628
Area Under Curve of RMSSD = 0.5884700976579379
Area Under Curve of SD_NN_RMSSD = 0.6998512805001008
```

- As a quick trial run of the new features, we trained a linear SVM and an LDA on the 14 features.
  - Did an 80/20 split for the training and test set.
  - Ensured there was approximately a 50/50 balance between positive and negative classes in both training and test set.
  - Achieved an accuracy of 68% and an AUC of 0.76 with the SVM (C = 0.001) on the test set.
  - Achieved an accuracy of 67% and an AUC of 0.76 with the LDA on the test set.
  - This was an improvement on what we saw before Christmas.
  - However, we used a different random seed for the train/test split – before Christmas the test set was skewed towards 1 class, which we wanted to avoid this time around.
  - Out of interest, I ran my notebook from before Christmas with the new random seed – and the results improved substantially (approximately the same as what we have with the 14 features) and generated very sensible results for Recursive Feature Elimination. This indicates to me that we got a lucky split now and before Christmas we were unlucky. At this point it seems crucial to set up a framework for LeaveOneOut so that we can get unbiased results from our models. I think at some point we'll need to train our linear models using LeaveOneOut so that we can get better estimates of their performance for comparison in the final report.

Objectives for next week:

- Set up a script/framework for LeaveOneOut – at this stage it's crucial to get this working so that we can generate unbiased results.
- Start working on a non-linear model using LeaveOneOut – this depends on how quickly we get LeaveOneOut up and running.

8th February – 14th February

To do:

- ☑ Set up LeaveOneOut framework for training and GridSearch-ing optimal hyperparameters.
- ☑ Develop a non-linear model using LeaveOneOut framework.

Summary:

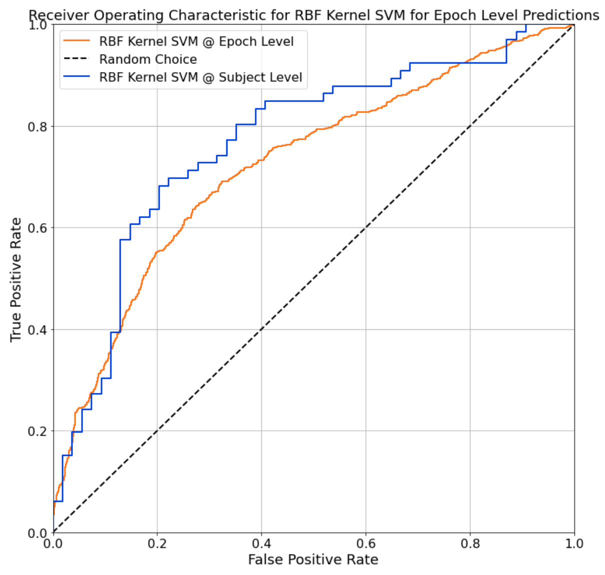
- Created a ‘Leave Three Out’ (or 40-fold evaluation) gridsearch and performance evaluation script.
  - Used Oksana’s thesis Chapter 5 as a reference when writing the script – as a result, it behaves almost identically (the main difference is Oksana’s script was Leave One Out).

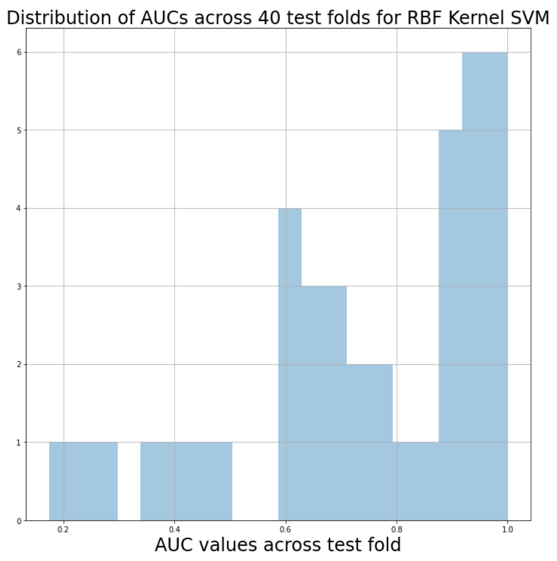
- The script has an outer 'Performance Evaluation' loop and an inner cross-validation loop.
- In the outer loop, 3 babies are randomly selected and kept as a hold-out test set.
- The 117 babies are taken and split into 5 folds.
- Cross validation is performed on the 5 folds (of 117 babies) - I.e. every pair of hyperparameters is tested on each of the 5 folds.
- The mean AUC across the 5 test folds is stored for each pair of hyperparameters.
- The hyperparameters corresponding to the largest mean AUC are then selected, and a model is trained on the 117 babies.
- The epoch level AUC across the 3 test babies is then calculated, and the mean AUC, the std, IQR and median are printed once the evaluation of all 40 folds is complete.
- The geometric mean probability of abnormality of all 120 babies is used to calculate the subject level AUC of the classifier.

- Tested a Gaussian kernel SVM with the gridsearch script. The results were:

```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7376790164556131
Standard Deviation of AUC across all test folds: 0.22588254686119744
Median of AUC across all test folds: 0.762410519763461
Interquartile Range of AUC across all test folds: 0.3088699494949495
Final Subject level AUC across all subjects: 0.7687991021324355
The most common hyperparameters selected were gamma = 0.01 and C = 0.1
Wall time: 14min 40s
```

The ROC of the SVM & distribution of AUCs was also plotted:





Attempted feature selection for the RBF kernel SVM – unable to use RFE for the RBF kernel due to the kernel trick (weights of the model can't be accessed/computed).
Tried univariate feature selection based on the AUCs of the individual features. The following features were selected:

```
Area Under Curve of mean_NN = 0.6740072163166537
Area Under Curve of VLF_power = 0.5880577881485323
Area Under Curve of LF_HF_ratio = 0.6739748077089275
Area Under Curve of TINN = 0.6005639097744361
Area Under Curve of Kurtosis = 0.5836412151067323
Area Under Curve of RMSSD = 0.5884700976579379
Area Under Curve of SD_NN_RMSSD = 0.6998512805001008
```

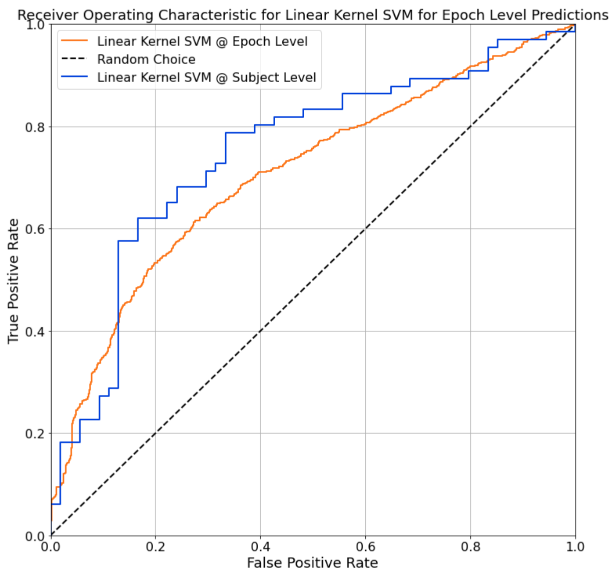
The SVM improved its performance on some test folds by a considerable amount, however, overall its performance declined:

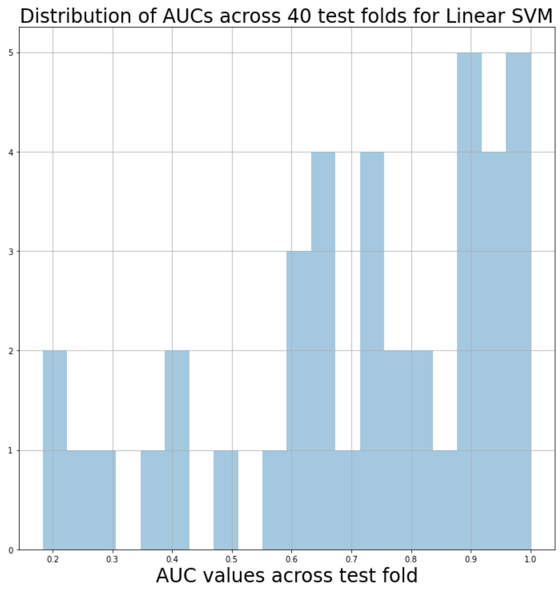
```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7197318953862286
Standard Deviation of AUC across all test folds: 0.23650192373448137
Median of AUC across all test folds: 0.7436974789915967
Interquartile Range of AUC across all test folds: 0.31710526315789467
Final Subject level AUC across all subjects: 0.75
The most common hyperparameters selected were gamma = 0.01 and C = 1
Wall time: 9min 29s
```

- Tested a linear kernel SVM to obtain a proper benchmark for the performance of the linear models. The results were:

```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7139864471282962
Standard Deviation of AUC across all test folds: 0.23189918456089442
Median of AUC across all test folds: 0.7470627139744788
Interquartile Range of AUC across all test folds: 0.2958829365079365
Final Subject level AUC across all subjects: 0.7474747474747474
The most common hyperparameter selected was C = 0.001
Wall time: 5min 41s
```

The ROC curve and the distribution of the AUCs across the test folds were:





- Ran a Gridsearch for a RandomForest classifier that sampled all 14 features at each node (Daragh's used random subsets of features):

```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7540993377153815
Standard  Deviation  of  AUC  across  all  test  folds:  0.26508087506208194
Interquartile Range of AUC across all test folds: 0.4188978292239163
Final Subject level AUC across all subjects: 0.7668350168350169
The most common hyperparameters selected were num trees = 50 and max depth
= 4
Wall time: 2h 4min 30s
```

Mean AUC across test folds improved but it had a greater spread. I hadn't added the ROC curve and AUC histograms to the script at the time I ran this gridsearch, and I didn't run it again since it had a 2-hour run time.

Objectives for next week:

- Set up a gridsearch and performance evaluation with XGBoost – we're hoping that the sequential error correcting predictors in XGBoost will help with the performance on test folds where the RBF SVM is only scoring 0.4 – 0.6, bringing the overall mean AUC up and reducing the spread.
- Set up and test a soft voting ensemble of classifiers.

15th February – 21st February

To do:

☑ Set up a gridsearch and performance evaluation with XGBoost.
☐ Set up and test a soft voting ensemble of classifiers.

Summary:

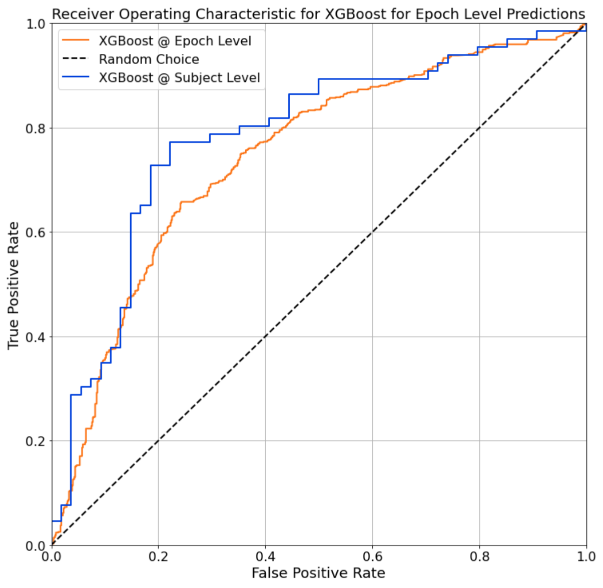
- Started a rough draft of the presentation slides with Daragh, mainly to plan out what will go in each slide.
- Calculated the AUC of SD2_Poincare/SD_NN, the result was 0.49.
- Set up a gridsearch/performance evaluation with XGBoost.
  - Worked with the XGBoost Python library (not scikit-learn), as this offers an optimised, efficient version of XGBoost.
  - Edited the gridsearch to search over 4 hyperparameters: the depth of the trees, the proportion of training instances sampled, the proportion of features sampled and the learning rate eta.
  - Used XGBoost's optimised data structures – the DMatrix. This allowed us to do approximately 84,040 fits in 27 minutes.
  - The gridsearch was done over these parameters:

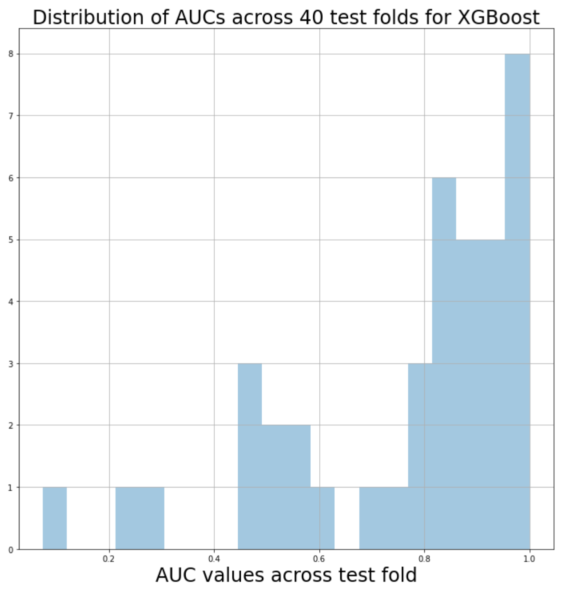
    param_grid = {'max_depth':[2,3,4,5,6,7,8],'subsample':[0.6,0.8,1.0],

    'colsample_bytree':[0.2,0.3,0.4,0.5],'eta':[0.1,0.2,0.3,0.4,0.5]}
  - The XGBoost classifier outperformed the RBF kernel SVM with the following results:

```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7694345940219428
Standard Deviation of AUC across all test folds: 0.2297173153133054
Median of AUC across all test folds: 0.8422619047619048
Interquartile Range of AUC across all test folds: 0.3488726551226552
Final Subject level AUC across all subjects: 0.781986531986532
The most common hyperparameters selected were max_depth = 3,
subsample = 1.0, colsample = 0.5 and eta = 0.3
Wall time: 27min 32s
```

  - The XGBoost ROC curve and he distribution of AUCs across the test folds:





Objectives for next week:

- Finish slides for presentation + background reading/notes for any questions in the seminar.
- Soft voting ensemble (Hopefully finishing this evening) - going to try a combination of RBF kernel SVM and XGBoost.
- Start reading up on neural networks.

1st March – 7th March

To do:

☑ Finish slides for presentation + background reading/notes for any questions in the seminar.
☑ Soft voting ensemble (Hopefully finishing this evening) - going to try a combination of RBF kernel SVM and XGBoost.
☑ Start reading up on neural networks.

Summary:

- FYP Seminar on Tuesday – questioned on Pan Tompkins algorithm, resampling before FFT, variance preserved with PCA and why Leave Three Out rather than Leave One Out.

- Finished soft voting ensemble using RBF SVM and XGBoost:
  - No real improvement over XGBoost.
  - Final results over all 40 test folds:

```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7693936831146033
Standard Deviation of AUC across all test folds: 0.22642971933198786
Median of AUC across all test folds: 0.8577556022408964
Interquartile Range of AUC across all test folds: 0.36163814484126999
Final Subject level AUC across all subjects: 0.7828282828282829
The most common XGBoost hyperparameters selected were max_depth = 3,
subsample = 1.0, colsample = 0.5 and eta = 0.3
The most common SVM hyperparameters selected were gamma = 0.01 and C
= 0.1
```

- Started reading about neural networks/deep learning:
  - Read background on artificial neural networks and multilayer perceptrons.
  - Implemented MLP for classification on a toy dataset (MNIST fashion) using the Keras library.
  - Started reading some documentation for Tensorflow.

Objectives for next week:

- Continue reading about neural networks to build up to CNNs and gain more exposure to Keras/Tensorflow.
- Set up script to extract raw RR interval data segmented into epochs so that it can be used for CNNs.

8th March – 14th March

To do:

- ☑ Continue reading about neural networks to build up to CNNs and gain more exposure to Keras/Tensorflow.
- ☐ Set up script to extract raw RR interval data segmented into epochs so that it can be used for CNNs.

Summary:

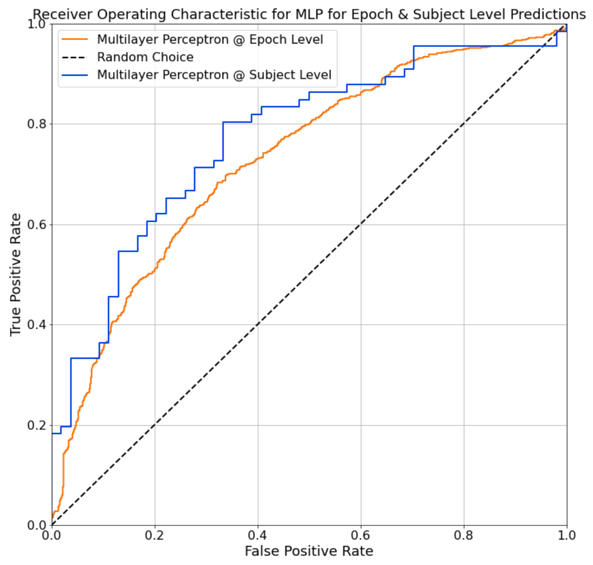
- Read about the basics of neural networks and multilayer perceptrons – backpropagation, autodiff, activation functions etc, as well as the Python libraries used to implement them – Keras and Tensorflow.

- Modified the Leave Three Out script so that it can be used for Tensorflow models.
  - Removed the gridsearch part of the script – with the time it's taking to simply train the MLPs, and the large number of hyperparameters that need to be configured for each network, we decided that for now we'll simply do 'manual' gridsearches - tweak some hyperparameter and run a 40 fold evaluation using that tweaked hyperparameter. We might implement a randomised search rather than a gridsearch which would test random combinations of hyperparameters – this would allow us to limit the number of searches done to reduce the time taken at each iteration.
  - The new Leave Three Out script performs a simple 80/20 split on the training epochs to create a validation set which we use for early stopping – this does mix subject data in the validation and training set, but all the data for the test subjects is still unseen.
  - Implemented support for Tensorboard – this allows us to monitor the model training in real time. It shows the training and validation loss, accuracy and auc in real time. It's set up in such a way that the data for each of the 40 models in the 40 fold evaluation can be monitored.
  - The script now automatically dumps all the results into a .txt file as well as saving the ROCs and an auto-generated image of the model structure – showing how many layers, neurons, types of layers, etc.

- Tested various MLP configurations using the 14 HRV features with the Leave Three Out script.
  - Experimented with various numbers of layers and neurons, as well as using dropout and batch normalisation.
  - The best result was achieved with 3 hidden layers, with 28, 14 and 7 neurons in each of the 3 layers:

```
===== Final Performance Across All Subjects: =====
Mean AUC across all test folds: 0.7368380136995667
Standard Deviation of AUC across all test folds:
0.2173325128869238
Median AUC across all test folds: 0.768939393939394
Interquartile Range of AUC across all test folds:
0.30417715964590963
Final Subject level AUC across all subjects: 0.7702020202020202
```

  - The ROC curve is:



- Completed Kaggle's microcourse 'Intro to Deep Learning' and 'Computervision', which covered the basics of CNNs.

Objectives for next week:

- Continue learning about CNNs.
- Test various CNN configurations with the Leave Three Out script using the raw RR interval data.

15th March – 21st March

To do:

☑ Continue learning about CNNs.
☑ Test various CNN configurations with the Leave Three Out script using the raw RR interval data.

Summary:

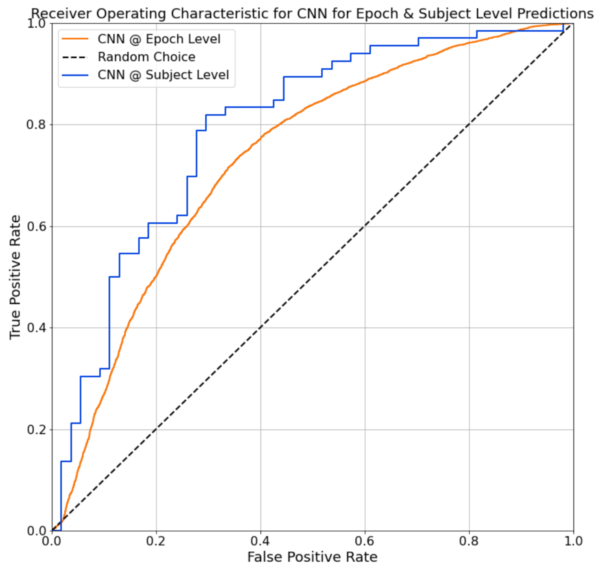
- Read a lot of papers and articles to try and gain some intuition behind choosing number of convolutional layers, number of filters, kernel sizes etc.

- Set up the Leave Three Out script with tensorboard on Google Colab to make use of their free GPUs for training.
    - Tested a lot of different 1D CNN architectures, starting with https://www.nature.com/articles/s41598-020-63566-8 which used a CNN for detecting ventricular tachyarrhythmia using HRV.
    - Model was overfitting quite quickly (after approximately 5 epochs). Early stopping meant the model was getting about 25 epochs of training before rolling back to the weights used at the fifth epoch.
    - There was considerable volatility in the validation loss curves, which was reduced by adding a batch normalisation layer to the input.
    - Changing the number of filters and kernel sizes did not help the overfitting and the model results were very poor, even with dropout and less neurons in the dense layers.
    - Tried a fully convolutional network, using the architecture described by Alison O'Shea in https://arxiv.org/ftp/arxiv/papers/1709/1709.05849.pdf
    - Again, this overfit the training set, with the validation loss skyrocketing after a few training epochs.

- Started from scratch with a single convolutional layer and a single dense layer with a few neurons to see how the performance depended on each hyperparameter, and built it up to a network with 3 convolutional layers and a single hidden dense layer:

  The architecture uses an input batch normalisation layer, followed by a 1D conv layer (6 filters, kernel size = 8, strides = 4), batch normalisation and average pooling (kernel size = 2, strides = 2), a 1D conv layer (6 filters, kernel size = 8, strides = 4), batch normalisation and average pooling (kernel size = 2, strides = 2), a 1D conv layer (6 filters, kernel size = 8, strides = 4), batch normalisation and average pooling (kernel size = 2, strides = 2), a 1D conv layer (3 filters, kernel size = 4, strides = 1), batch normalisation and average pooling (kernel size = 2, strides = 2), a dense layer with 8 neurons followed by the output dense layer with 1 neuron (sigmoid activation).

- The problem of overfitting still remained for certain folds – I allowed the model to run for the full 40 folds, achieving a mean epoch level AUC of 0.66 and a subject level AUC of 0.72. The validation set up to this point consisted of 20 babies, with all the data from these babies withheld during training. I replaced this validation set with a simple 80/20 splits on all epochs of the training data (117 babies) meaning epochs from the same baby could appear in both the training and validation set.
- This drastically improved the performance of the validation set (as expected), but more importantly it allowed the model to actually train for more than 5 epochs – in most cases getting the full 80 epochs of training. The same model mentioned in the previous bulletpoint was evaluated this way, and the overall results improved to 0.785 subject level AUC with a mean epoch level AUC of 0.73.
- I doubled the number of filters in each convolutional layer, which improved the performance further, with a subject level AUC of 0.79, the highest we've seen so far. The standard deviation of epoch level AUCs, the median epoch level AUC and the interquartile range were approximately equal to XGBoost, our best model so far. However, the mean epoch level AUC was a bit lower (0.74 vs 0.77). The ROC is:



- Daragh ran 40-fold evaluations to see if adding more dense layers/more neurons in the existing dense layers could improve performance but it generally degraded the performance
- Added a 4th convolutional layer but the performance wasn't any different to the architecture with 3 convolutional layers.

Objectives for next week:

- More or less need to finish practical work on the project by the end of the week.
- Experiment with increasing the number of filters in the 3 convolutional layers, as well as the strides, kernel size, dropout and max pooling vs average pooling.
- Create poster and video for open day.

22nd March – 28th March

To do:

☑ Experiment with increasing the number of filters in the 3 convolutional layers, as well as the strides, kernel size, dropout and max pooling vs average pooling.
☑ Create poster and video for open day.

Summary:

- Experimented with various CNN configurations:
  - Changed kernel size, strides, number of filters, max pooling, reduced the number of convolutional layers to 2.
  - Read into dropout more and found that conventionally it's only supposed to be placed **after** dense layers, whereas previously I had it after both the flatten and the dense hidden layer. As a result I tested Dropout(0.5) only after the dense layers for the 2 best performing CNNs: 3 Convolutional layers, with 18, 18 and 9 filters, followed by a dense hidden layer, and 3 convolutional layers, with 24, 24 and 24 filters, followed by a dense hidden layer. The architecture using 3 convolutional layers achieved the best results so far:

    ```
    ===== Final Performance Across All Subjects: =====
    Mean AUC across all test folds: 0.7715654159114773
    Standard Deviation of AUC across all test folds: 0.20728241822682947
    Median AUC across all test folds: 0.8287279913762127
    Interquartile Range of AUC across all test folds: 0.2984082568139117
    Final Subject level AUC across all subjects: 0.8035914702581369
    ```

  - The mean AUC matched that achieved by XGBoost, with a lower standard deviation and interquartile range. The final subject level improved to 0.8.
  - The data from all the different experiments are saved on Google Drive so that we can check them as we're writing our reports.

- Worked on the poster and video for the open day with Daragh.

Objectives for next week:

- Start writing the final report.