7-2023

Identification of Heart Disorders With Symbolic Aggregate Approximation

Moses K. Owusu
The University of Texas Rio Grande Valley

IDENTIFICATION OF HEART DISORDERS WITH SYMBOLIC AGGREGATE APPROXIMATION

A Thesis

by

MOSES K. OWUSU

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Major Subject: Applied Statistics and Data Science

The University of Texas Rio Grande Valley

July 2023

IDENTIFICATION OF HEART DISORDERS WITH SYMBOLIC AGGREGATE APPROXIMATION

A Thesis
by
MOSES K. OWUSU

COMMITTEE MEMBERS

Dr. Hansapani Rodrigo
Chair of Committee

Dr. Tamer Oraby
Committee Member

Dr. Michael Machiolatti
Committee Member

Dr. George Yanev
Committee Member

July 2023

ABSTRACT

Owusu, Moses K., Identification of Heart Disorders with Symbolic Aggregate Approximation. Master of Science (MS), July, 2023, 85 pp., 12 tables, 29 figures, references, 47 titles.

Motif identification in Electrocardiogram (ECG) time series is challenging due to the nature of the data as well as the computational complexity of the algorithms used to identify such patterns. This project utilizes the Symbolic Aggregate Approximation (SAX) on 1000 fragments of ECG signals from 45 patients (42% females aged between 23 and 89 years and 58% males aged 32 to 89 years) using data obtained from the MIT-BIH Arrhythmia database to recognize cardiac health disorders (Pławiak 2018a). Data include a Normal Sinus Rhythm, and ECG readings for 11 heart disorders, making 12 in total. The aim is to use SAX to identify heart disorders using the ECG signals, by first analyzing QRS-complexes, splitting the time series into smaller equally sized segments using the Piecewise Aggregate Approximation (PAA) approach. SAX is then used to discretize the normalized PAA series to a string of arbitrary length alphabets. Using a sliding window algorithm, we create a list of word bags that are the result of SAX on our data set – this gives the words as well as frequency counts. This will be done to the sample data set for each heart disorder and at the end, we are able to determine which exact patterns contributes the most (motifs) and the least to our class of words by comparing weights, thereby facilitating the identification of possible heart disorders. Long Short Term Memory (LSTM) neural network approach was also used to classify the ECG signals and results were in line with that obtained for SAX.

DEDICATION

To my parents, siblings (Adwoa, Ama and Kwabena), Mr.Roland Acheampong, Albert Arko Paintsil, Samuel Nti Frimpong, Nathaniel Addare Boateng, Asiamah Richard and the RGV Ghanaian community especially all PCFC members.

ACKNOWLEDGMENTS

I want to thank Dr. Rodrigo for her immense support and guidance throughout the period of developing this thesis. She was readily available even on weekends to proofread, run codes herself to debug errors or discuss the way forward despite being with child. To Dr. Oraby, Dr. Yanev and Dr. Machiorlatti for being a part of the committee and giving insight. I have been exposed to a lot of experiences throughout the course of writing this thesis; learning, re-learning and unlearning what I have been exposed to in this journey for an MS degree. My sincerest gratitude to Prof. & Mrs. Andoh-Baidoo who were my parents away from home. To Theophilus Gyedu-Baidoo, Nicholas Niako and David Banahene, my roommates, the humor during the stressful days, the sumptuous meals I came back to on tiring days and help with things I didn't understand is very much appreciated. I thank Vigne Krishnamoorthy for his insight and explanation of the algorithms to me. I appreciate all those who supported me in one way or the other.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Patterns Recognition In Time Series Data

Anomaly detection has gained much traction owing mainly to its vast applications with the aim of first representing the data in a way that reduces its dimension but keeps key information (Ren et al. 2018). Time series motifs are repeated segments across a time series (see Figure 1.1). Discords or anomalous patterns are seen as the outliers observed in time series data (He, Xu, and Deng 2003). One distinguishing factor of motifs is their similarity which cast doubts on their occurrence being random (Mueen 2014). Motifs carry important information about the underlying dynamics and conditions of the system from which the time series was recorded just as Deoxyribonucleic acid (DNA) carries genetic information. These repeated patterns appear with different frequencies, lengths, lags, disparities across an entire series ((Mueen 2014) , (Huang et al. 2015)).

Segmentation plays a key role in the symbolization process without which pattern recognizing algorithms cannot be implemented. It is done by either quantization (breakpoints are in amplitude domain) or temporal segmentation (breakpoints are based on temporal domain). The latter requires similar breaks be first clustered before they are symbolized (Sant'Anna and Wickström 2011). Literature on time series anomaly detection have mostly focused on implementing algorithms that are; highly representative of the original data, capture the slightest underlying characteristics as well as have reasonable computational cost. The nature of these anomalies being recognized and the random way they occur in time series data makes it difficult for one single method to satisfy optimally, all three objectives highlighted above.

1.2 Literature Review

Representation of time series to avoid misrepresentation or loss of valuable data has been the focus of many time series studies (Lin, Keogh, Wei, et al. 2007). Moreover, having found a way to represent these data, the need arises to develop very efficient methods to analyze and understand important characteristics. Interestingly, many studies have had to either tackle one problem at the expense of another or risk a balanced trade-off that makes the results of their analysis doubtful. A very representative approach is highly likely to result in huge computational cost and would require more space to save vital information as the algorithm runs.

A subsequence that is maximally different from all remaining subsequences of a time series data qualify as discords. (Huang et al. 2015) used J-distance discord (JDD) which imitates the basics of k-nearest neighbor (KNN) to tackle the challenge of repeated anomalous patterns in which these repetitions are not too distinguishable to be classified as separate discords. JDD measures similarity between an identified subsequence and the j^{th} subsequence it is most similar to just like the numerosity reduction SAX uses. (Woodbridge et al. 2015) also applied the brute force version of SAX as well as Heuristically Ordered Time series (HOT SAX) to 240 Hz waveform data obtained from 9,723 patients. Results support the time complexity and information loss that is observed in choosing optimal parameters for the SAX algorithm with the brute force having an execution time $O(n^2)$.

(Kha and Anh 2015) used a Cluster-based Discord approach (CBD) suggested by (He, Xu, and Deng 2003) and observed that the anomalies identified in the data by both HOTSAX and their method were perfectly matched. However, they claim their approach is more efficient than SAX despite using fixed parameters and not observing results from varying these values. SAX representation of time series has been applied to numerous data mining situations such as: partitional clustering, query by content, motif discovery, anomalous behaviour detection, decision tree classification, hierarchical clustering, nearest neighbor classification and its visualizations (Lin, Keogh, Wei, et al. 2007). SAX visualizations is unique as the method takes advantage of the reliability of text to capture varying sections of time series to make tree-like figures using position

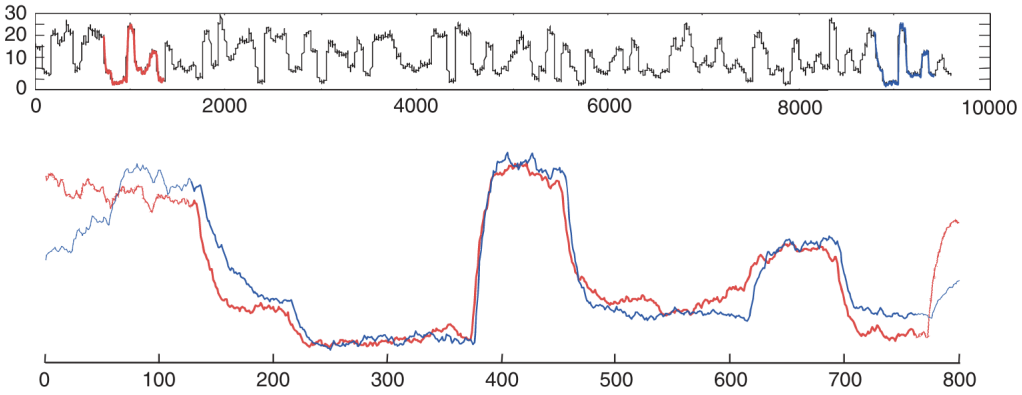

Figure 1.1: Steam flow time series with an overlay of the identified motifs
Source: (Mueen 2014)

of a symbol. This means that, for instance, all character strings starting with a particular alphabet can be observed and checked for repetitions or outlying numbers that signify anomalous behaviour (Krishnamoorthy 2018b).

Real-value representation based anomaly detection is convenient in numeric operation with the Piecewise Aggregate Approximation (PAA) as a typical example. Generally, the aim of motif discovery algorithms is to identify recurring patterns that can then be used to distinguish between observations efficiently (Mohammad and Nishida 2009). The challenge of these algorithms is the non-existence of prior knowledge about the precise positions of these patterns thereby increasing computational cost if an exhaustive search approach that loops through the entire sequence of observations to find pre-determined patterns is implemented. More so, using a small motif length size leads to identifying too many existing patterns that may not necessarily signal anomaly while a longer length can lead to non-identification of significant motifs that may or may not carry any important information.

Extraction of patterns from time series is applicable in disease diagnosis and detection, traffic flow analysis as well as stock price prediction (Ohsaki, Abe, and Yamaguchi 2007). (Sun et al. 2014) proposed trend distance (TDIST), a distance measure to tackle the problem of having different trend segments assigned to the same symbol because they have the same average. Their results showed better distance measures than that of SAX highlighting the inability to capture the

trend in a segment with few character representations.

1.3 Problem Statement

Time series symbolic representations are not new and have been in use for some time. Two major flaws have been identified: (1) dimensionality challenge and (2) distance measures used in these former methods have less in common with the actual distances that is existent in the data (Lin, Keogh, Wei, et al. 2007). It is challenging to find patterns (motif) in Electrocardiography (ECG) data unlike it is for others generated from physiological experiments due to the periodic nature of its observations (Mueen and Keogh 2010). Identification of motifs is difficult and many researchers have resorted to different methods. In this study, we utilize the SAX algorithm to identify motifs across patient ECG time series data. SAX helps us identify irregular patterns relating to the QRS complexes for each of the heart disorder due to the varying fluctuations observed in the ECG signal for each disorder class owing to the contraction, expansion and relaxation of heart muscles to generate this data. Hence, comparison of this patterns can be made to distinguish them from a normal human heart beat.

1.4 Research Questions

The following research questions were asked to serve as a guide to the research:

1. How can ECG signals be used to distinguish between heart disorders?
2. Can SAX detect abnormal ECG and identify their position?
3. Which ECG analysis approach performs better, LSTM or SAX?

1.5 Research Objectives

The aim of this study is to classify heart disorders by using different measures to distinguish patterns identified for 12 heart disorder classes. Graphical plots showing weighted patterns will be used to depict discrepancies between each class and the Normal Sinus Rhythm (NSR). Numerical measures of the Euclidean distance, Term Frequency Inverse Document Frequency (TFIDF), and cosine similarity will be compared to the results of other algorithms such as Long Short Term

Memory (LSTM). Classification will be done using LSTM and the results will be compared with that of SAX using various model diagnostic measures to check how well the algorithm works and to clarify doubts about the data being overly fitted to a particular method. The objectives for this study are as follows:

- Classify heart disorders using different data mining techniques, that is; LSTM and SAX
  ECG signals will be discretized (assigned to symbols). Irregular patterns will be identified for ECG signals of each class. Patterns will be compared and their differences used to distinguish between the varying heart conditions from which they were recorded.

- Use SAX to identify discords in ECG data
  SAX will be used to identify pattern(s) in the time series that are unique and maximally different (highest modified Euclidean distance) from the other pattern(s) observed.

- Compare results for LSTM and SAX to see which performs better.
  The performance measures (accuracy) obtained from the implementation of LSTM and SAX on the ECG signals will be compared to see which approach performs better as well as understand the rational behind the performance.

1.6 Research Design

This work will be in four chapters: the first, an introduction with details of the history behind the methods chosen, observations present in literature, the problem identified and what this research seeks to achieve. The second chapter will define the methods to be implemented, with emphasis on the basic ideas and formulation dynamics of these methods and how parameters are estimated. Chapter three will present the results obtained from the LSTM to make classification while the fourth chapter will highlight the SAX algorithm implementation results. Results will be discussed in the fifth chapter in relation to what is existent in literature and appropriate recommendations for future research will be provided.

CHAPTER II

METHODOLOGY

2.1 Symbolic Time Series Analysis (STA)

Time series consist of a collection of data points collected sequentially over a period of time. The main aim of time series analysis is to model how the observations are generated over time as well as make predictions ((Chatfield and Xing 2019),(Cryer 1986)). The possibility of breaking down time series data into varying components in its makeup such as trend and noise makes it an interesting type of data to be studied (Hamilton 2020). Analysis of time series helps with the detection of regularities in records of a variable to capture characteristics needed to be exploited for predicting future outcomes (Kirchgässner, Wolters, and Hassler 2012). Due to the way observations are collected over time, fluctuations in the phenomenon under study with respect to time are also captured (Fuller 2009).

Electrocardiogram (ECG) signal is analogous to the electrical activity of the heart that is used to record signals from the heart to: examine heartbeat rhythm, measure heart rate, diagnose abnormalities of the heart etc. ((Berkaya et al. 2018), (Bonow et al. 2011)). ECG can be used to express cardiac attributes distinctive from person to person (Israel et al. 2005). These signals are not fixed because they are as a result of relaxation and contraction of heart muscles which vary depending on whether an individual is at rest or engaging in a physical activity. ECG can be divided based on the re-polarization and depolarization of heart muscles as shown in Figure (2.1) (Biel et al. 2001).

Output data of several experiments in various fields have been subjected to Symbolic Time Series (STA) methods to observe patterns (Daw, Finney, and Tracy 2003). Despite being able to

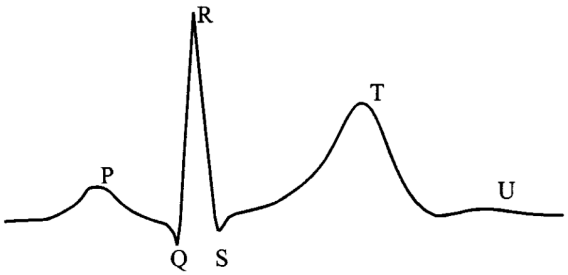

Figure 2.1: ECG complex elements
Source: (Biel et al. 2001)

identify these changes in the data with the use of traditional approaches such as Fourier Transforms, discretization methods increase computational efficiency, decreases sensitivity to noise and improves analysis of complex processes. Many traditional quantization methods used to analyze time series data such as Symbolic Aggregate Approximation (SAX) differ from others due to the way the splitting is done that is, the amplitude approach (Sant'Anna and Wickström 2011).
Moreover, defining two of the main parameters in the SAX algorithm that is; alphabet size and window size brings about a trade-off between computational complexity and information loss ((Sant'Anna and Wickström 2011), (Mörchen and Ultsch 2005)). It is imperative to obtain parameters that minimize the challenge posed by this trade-off. Preprocessing steps such as; PAA, string assignment and filtering to remove the periodicity by eliminating repetitive representative segments anywhere it occurs can also be applied. In doing so, subsequent analysis is expected not to reveal repetitions, as such, any identified set of similar motifs spark interest and would be further observed (Mueen 2014).

Piecewise Aggregate Approximation (PAA) will be applied to the ECG fragments to divide them into smaller segments (divide a series of length say m into smaller equally sized segments of size n) that still capture any underlying characteristics relevant to the observations (Kulahcioglu, Ozdemir, and Kumova 2008). SAX will then be used to discretize the series, that is, convert the equally spaced numeric series into a string of symbols (alphabets) (see Figure 2.7). Wavelet filtering is applied to our ECG data to remove irrelevant features and noise that does not represent underlying patterns. Extracted features from the wavelet analysis will then be used as the input data for LSTM

implementation that is explained in the next section.

2.2 Long Short-Term Memory

Long Short Term Memory (LSTM) models are special deep learning approaches which are characterized with an 'internal memory' and have been successfully used for speech recognition, sentiment analysis and Natural Language Processing (NLP) ((Y. Yu et al. 2019), (Smagulova and James 2019)). Traditional Neural Networks (NN) are characterized by two main components in their architecture: an input, which is processed through a defined architecture to produce an output. In scenarios where prediction based on immediate or a sequence of preceding elements is the target, there is the need for our NN to have some form of memory of what has just immediately occured (Smagulova and James 2019). LSTM serves this purpose as it forms the needed memory required as an input for the next stage of the NN implementation thereby helping with predicting future outcomes based on knowledge of previous stages.

These NN architectures are characterized by a middle layer where output from the first step is fed as input to a middle (hidden) layer (Staudemeyer and Morris 2019). The hidden layer(s) is,are the most important feature since it serves as the memory (input) upon which the output layer depends (see Figure 2.2). Several studies have implemented variants of LSTM including but not limited to; unidirectional, BiLSTM (two LSTM) cells, stacked LSTM and many others whose structures are mostly dictated by the requirements/specifications of the problem at hand. (Greff et al. 2016) in their study however note that LSTM variants do not improve the standard structure significantly while highlighting the output activation function as the most important aspect of its architecture.

Despite LSTM being a very good predictive model, it is unable to help us find the exact position of motifs in our data and so, SAX (presented in the next section) is used for this purpose. Gradient based algorithms are used to update weights in Recurrent Neural Networks (RNN) usually result in the vanishing gradient problem. LSTM has been observed to overcome this challenge posed efficiently (Smagulova and James 2019). In Recurrent Neural Networks (RNN), outputs from nodes affect later inputs since the connection between nodes is cyclic. In other words, this

NN architecture saves some information of previous inputs as activations (short-term memories) which is used subsequently to update the state of both present and subsequent inputs ((Chen and Soo 1996), (Elman 1991)). This interesting attribute of RNNs have been successfully used to tackle some problems ((Karpathy, Johnson, and Fei-Fei 2015), (Li et al. 2018)). In scenarios with long order of inputs, short-term memory becomes inadequate and RNNs are unable to learn from these sequential entries.

(Hochreiter and Schmidhuber 1997) came up with the LSTM approach which is a modification of RNNs with a significant longer short-term memory to tackle the challenge RNNs faced in analyzing long-term dependencies. Interestingly too, this new architecture faced the drawback of being unable to eliminate unessential states of preceeding inputs. This hindrance in simple terms meant the algorithm will continue to increase out of bounds until it finally crashes. (Gers, Schmidhuber, and Cummins 2000) came up with the "forget gate" (see Figure (2.3) that helps to eliminate unwanted redundancies.

2.2.1 LSTM architecture

The standard LSTM structure (Figure 2.2) does not have a forget gate. It simply consists of an output and input gate. The input gate processes and decides what should be added to the cell at any time while the output gate determines the output to be produced based on the state of the cell. The process is repeated for each time the cell states are modified. $i_t = \sigma\left(W_{ih}h_{t-1} + W_{ix}x_t + b_i\right)$

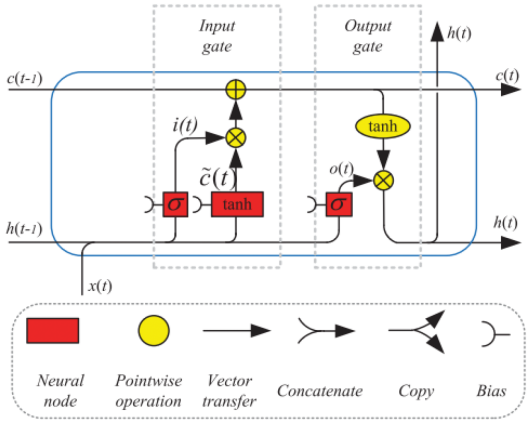

Figure 2.2: Standard LSTM Architecture
Source: (Y. Yu et al. 2019)

$$\tilde{c}_t = \tanh\left(W_{\tilde{c}h} h_{t-1} + W_{\tilde{c}x} x_t + b_{\tilde{\tau}}\right)$$

$$c_t = c_{t-1} + i_t * \tilde{c}_t$$

$$o_t = \sigma\left(W_{oh} h_{t-1} + W_{ox} x_t + b_o\right)$$

$$h_t = o_t * \tanh(c_t)$$

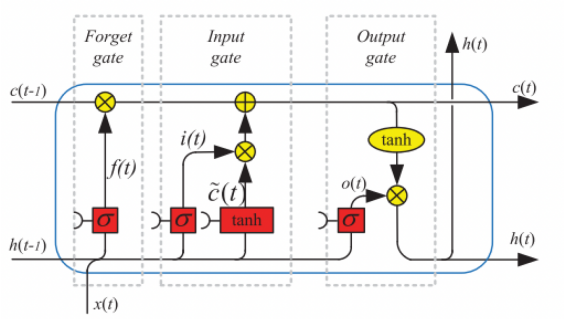

Figure 2.3: LSTM Architecture with Forget Gate Function
Source: (Y. Yu et al. 2019)

where:

x_t are the inputs

h_t recurrent information at time t

b_i is the bias

c_t is cell state

W_i, $W_{\tilde{c}}$ and W_0 are the weights

$*$ an operator for the multiplication of vectors

The equations above are the mathematical expressions of the pictorial connections as seen in the standard LSTM structure. The equations below are slight modifications that capture the introduction of the "forget gate".

$$f_t = \sigma\left(W_{fh} h_{t-1} + W_{fx} x_t + b_f\right)$$

$$i_t = \sigma\left(W_{ih} h_{t-1} + W_{ix} x_t + b_i\right)$$

$$\tilde{c}_t = \tanh\left(W_{\tau h} h_{t-1} + W\ x_t + b_{\tilde{\tau}}\right)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

$o_t = \sigma\left(W_{oh}h_{t-1} + W_{ox}x_t + b_o\right)$

$h_t = o_t * \tanh(c_t)$

2.3 Symbolic Aggregate Approximation

SAX was proposed by Eamonn Keogh and Jessica Lin in 2002 (Lin, Keogh, Lonardi, et al. 2003). It makes use of the PAA divided series to transform it into a string of characters, maintaining the low computational complexity the reduced series offers. A series of arbitrary length, n is reduced to a string of arbitrary length w such that $w < n$ and the alphabets (symbols) used should be > 2 (Krishnamoorthy 2018b). The main advantages of SAX as outlined by (Sant'Anna and Wickström 2011) in comparison to other symbolization techniques is that SAX yields the lower bound of of the true distance measure used to evaluate two time series. Also, SAX is an efficient algorithm that reduces the size of a series under consideration through PAA implementation before the actual discretization is done.

2.3.1 Piecewise Aggregate Approximation

Piecewise Aggregate Approximation (PAA) is an algorithm used to reduce the dimension of a time series by first dividing the entire series into m equally sized portions which are as a result of averaging the observations that fall within each segment ((Keogh et al. 2001), (Krishnamoorthy 2018a)). The two main components of PAA are z-normalization and dimensionality reduction.
Z-normalization: Before the algorithm computes means of the sub-intervals, the entire series has to be first z-normalized. That is, the observations are normalized to have a mean zero (0) and standard deviation of one (1). According to (Senin and Malinchik 2013), significantly different time series can be modified to look similar when the observations are z-normalized (Senin and Malinchik 2013). Hence, it is easier to compare two time series that have been z-normalized (see Figure 2.4).

$$x_i^{'} = \frac{x_i - \mu}{\sigma}, i \in \mathbb{N} \tag{2.1}$$

where: x_i is the ith observation

μ is the mean of the series

σ is the standard deviation of the series

x_i' is the normalized output of observation x_i

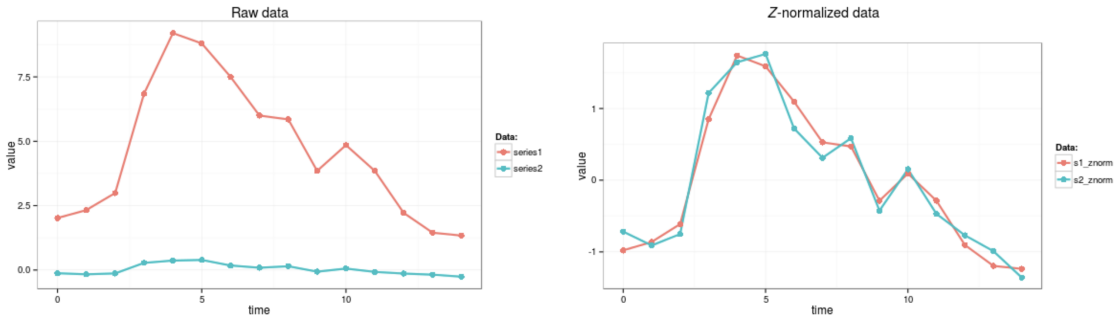

Figure 2.4: Two raw time series (left) and their similar looking z-normalization plot (right)
Source: (Senin and Malinchik 2013)

Dimensionality Reduction: Suppose we have a time series $X = X_1, X_2, \ldots, X_n$ of length (n) that is to be divided into a series $\bar{X} = \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_M$ where $M \leq n$. An equation to describe the reduced series can be represented by:

$$\bar{x}_i = \frac{M}{n} . \sum_{j=\frac{n}{M}(i-1)+1}^{(n/M).i} x_j \tag{2.2}$$

(Senin and Malinchik 2013)

where:

M = length of divided series

n = length of original series

j = 1, $1 + \frac{n}{M}$, $1 + 2\frac{n}{M}$ …, n

i = 1,2, …, M

$\bar{x}_i$ = mean of observations in each equi-sized segment

x_j = observation j in each of the i equi-sized regions

For example, for $n = 30;$ and $M = 10,$ there will be M(10) segments that contain 3 observations each $j = 1, 4(1+3), 7(1+2*3), \ldots, 30;$ i = $1, 2, \ldots, 10$. The $\bar{x}_i$ would be as follows:

$$\bar{x_1} = \frac{1}{3}\sum_{j=1}^{3} x_j, \quad \bar{x_2} = \frac{1}{3}\sum_{j=4}^{6} x_j \quad ,\ldots, \quad \bar{x_{10}} = \frac{1}{3}\sum_{j=28}^{30} x_j,$$

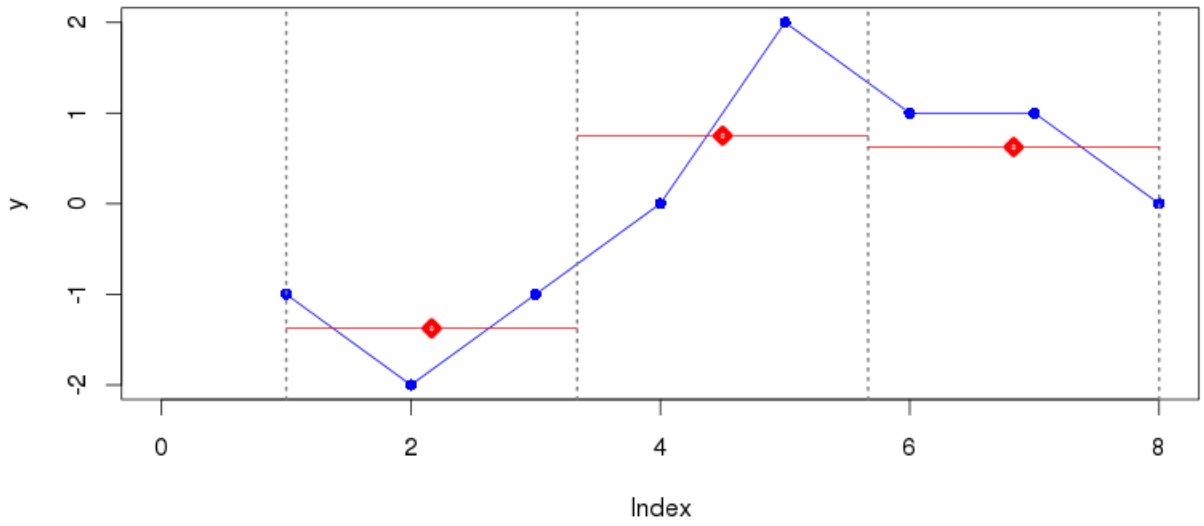

Figure 2.5: 8-points time series and its PAA transform into 3 points
Source: (Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018)

In Figure 2.5 above, a PAA transform of 8 points (blue) into 3 (red lines) is shown. It can be seen that the observations are first split into 3 equal intervals and the mean of values in each sub-interval is computed and used to represent that interval. Special cases of this division resulting from Equation 2.2 also exist when $M = n$ the split series is an exact replication of the original, and $M = 1$ the split series is the average of the original ((Krishnamoorthy 2018a), (Lin, Keogh, Lonardi, et al. 2003)). By applying PAA to time series, the computational cost can be reduced (Keogh et al. 2001). Discretization, explained in the following section is the next step after dimensionality reduction.

2.3.2 Look-up tables

In Figure (2.6), the breakpoints that define which observations from the PAA would be assigned to alphabets *a,b* and *c* are obtained from Table (2.1) below. The breakpoints are such that $\alpha = \alpha_1, \ldots, \alpha_{\beta-1}$ and for α_i to α_{i+1}, the area under the curve is $\frac{1}{\beta}$. For instance, for a word string *abcde*, *a* would be assigned to all observations from the PAA that fall between α_1 and α_2 with area

$\frac{1}{5} = 0.2$, where β = 5 (see Figure 2.6 below).

Table 2.1: Breakpoints Lookup Table

	β							
α_i	3	4	5	6	7	8	9	10
α_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
α_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
α_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
α_4			0.84	0.43	0.18	0	-0.14	-0.25
α_5				0.97	0.57	0.32	0.14	0
α_6					1.07	0.67	0.43	0.25
α_7						1.15	0.76	0.52
α_8							1.22	0.84
α_9								1.28

Source: (Lin, Keogh, Wei, et al. 2007)

2.3.3 Discretization

The normalized series have a Gaussian distribution, and so splitting with equiprobability becomes easy. If a string of length say m is chosen, the area under the normal curve is split into m equal parts, each serving as breakpoints that define the boundaries of each character in our chosen string (Krishnamoorthy 2018b).

Discretization maps the first character in the chosen m-length string to all PAA coefficients that fall below the first breakpoint, the next character is assigned to all coeficients that fall in the region greater than the first breakpoint but less than the second breakpoint until the entire equally split Gaussian curve is assigned a character from our chosen m-length letter string.

A *word* is a sequence of symbols that is generated by a SAX algorithm. For example, *aacbd*, could be a 5-letter SAX output for a series. Each word is a sub-sequence M with length n such that if β_j is the j-th element of the word, the PAA output is mapped to a word $\hat{C}$ using $\hat{C} = \beta_j$ if $\alpha_{i-1} \leq \bar{C}_j < \alpha_i$ ((Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018), (Krishnamoorthy 2018b)).

where; $\hat{C}$ is any alphabet/symbol in our list of symbols

$\bar{C}$ is the mean of observations in that range (PAA approximation)

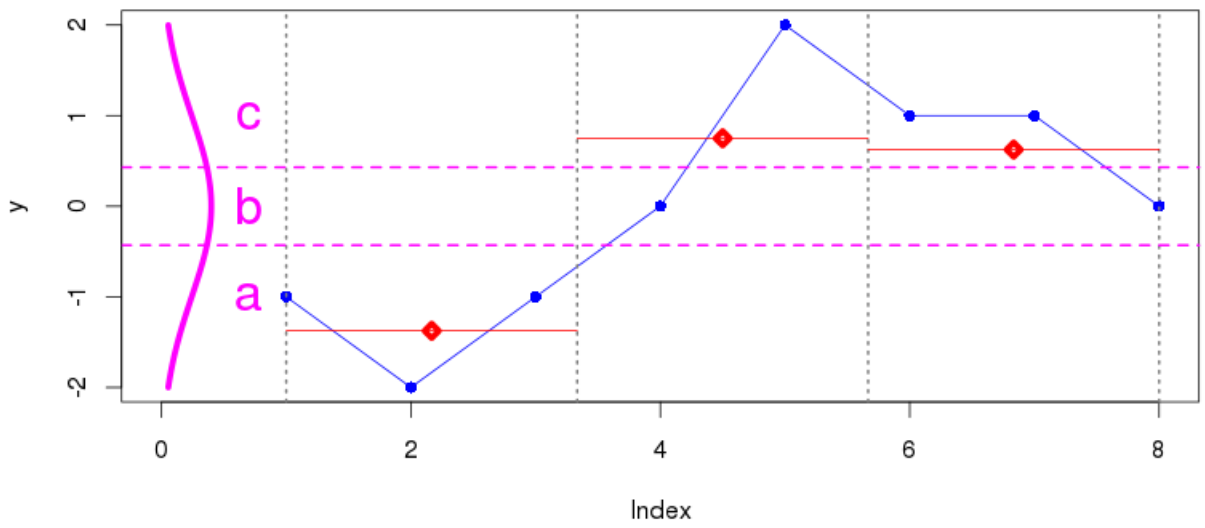

Figure 2.6: Normalized series assigned characters
Source: (Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018)

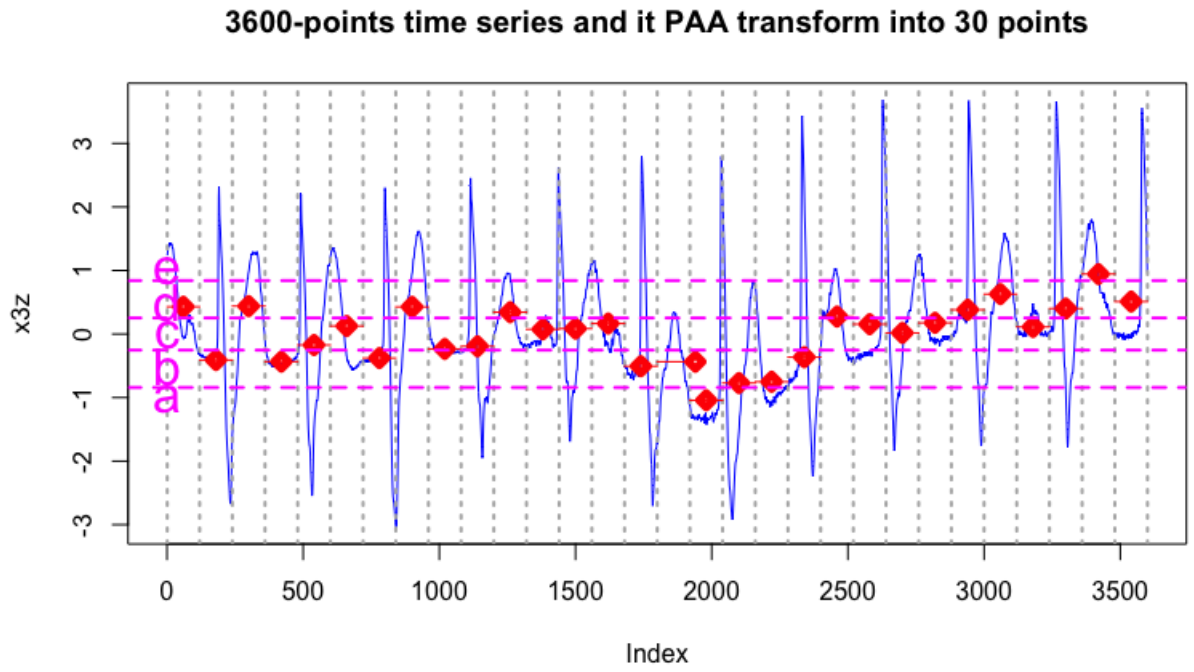

Figure 2.7: Discretization of 3600-points series to 5 alphabets

2.3.4 Distance Measures

A major part of motif identifying algorithms is distance measures. Dynamic Time Warping (DTW), correlation coefficient and Euclidean distance are examples in use (Mueen 2014). Given two time series C and Q of the length n, their Euclidean distance is defined as follows:

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \tag{2.3}$$

for q_i, c_i, in Q and C respectively (Lin, Keogh, Wei, et al. 2007).

When the series is transformed using PAA (see 2.3.1), the Euclidean distance of the new series Q_i and C_i is as follows:

$$D(\bar{Q},\bar{C}) \equiv \sqrt{\frac{n}{w}}\sqrt{\sum_{i=1}^{w}(\bar{q}_i - \bar{c}_i)^2} \tag{2.4}$$

Equation (2.4) above gives a lower bound approximation of the original series. After symbolization, we can define a function MINDIST that computes the minimum distance between the string (word) output of two time series.

$$MINDIST(\hat{Q},\hat{C}) \equiv \sqrt{\frac{n}{w}}\sqrt{\sum_{i=1}^{w}(dist(\hat{q}_i,\hat{c}_i))^2} \tag{2.5}$$

The main difference between Equation (2.4) and (2.5) is the $dist()$ function that uses a lookup table (Table (2.2)). For an word of size four (4), the distance two alphabets can be checked by tracing across the row and column in which the two characters are found. For instance, *dist*(**b,c**) = 0 and *dist* (**b,d**) = 0.67. Table (2.2) only needs to be calculated once for any given alphabet size a. The output is stored and subsequent iterations just lookup distances without having to compute them again, making the algorithm faster than other methods.

Table 2.2: Lookup Table for String of Length 4

	a	**b**	**c**	**d**
a	0	0	0.67	1.34
b	0	0	0	0.67
c	0.67	0	0	0
d	1.34	0.67	0	0

Source: (Lin, Keogh, Wei, et al. 2007)

$$cell_{r,c} = \begin{cases} 0, & \text{if} |r-c| \leq 1 \\ \beta_{\text{max(r,c)-1}} - \beta_{\text{min(r,c)}}, & \text{otherwise} \end{cases} \tag{2.6}$$

where r & c represent the row and column respectively

2.3.5 Discretization - Sliding Window

For a time series $T = t_1, \ldots, t_m$, a subsequence C is a sample $t_p, \ldots, t_{p+n+1}$ of points with length $n \ll m$ with p a random position such that $1 \leq p \leq m-n+1$. Subsequences are pulled out using a sliding window. All possible subsequences of length n (defined by user) of an m length times series T can be obtained by sliding an n-size window across the series (Senin and Malinchik 2013). This process yields an output of SAX words with each, corresponding to the leftmost sliding window point. For example, the sequence Y_1 where each word is a subsequence obtained from the main series using a sliding window and then discretization with SAX is performed. Subscripts denote the start position of the subsequence:

$$Y_1 = \text{eeg}_1\text{eeg}_2\text{efg}_3\text{eff}_4\text{egh}_5\text{eeg}_6\text{eeg}_7\text{eeg}_8\text{efg}_9 \ldots$$

2.3.6 Numerosity Reduction

For the sequence Y_1 in Section 2.3.5, we could see the output of the sliding window for neighbouring subsequences look similar which results in a huge number of consecutive identical SAX words. Subsequently, this leads to the identification of trivial matches that affects the perfor-

mance of SAX algorithm. To avoid this, a numerosity strategy is put in place to mitigate the effects of identified similar neighbouring patterns. Numerostiy reduction takes note of and records only the first appearance of a SAX word if there occurs consecutive similar subsequences (Senin and Malinchik 2013). It ensures the output string contains only one of such similar patterns but still maintains records of position and takes that into account when reporting different subsequent words in the same series. When numerostiy reduction is applied to the sequence Y_1 in the previous section, it yields:

$$Y_2 = \text{eeg}_1\text{efg}_3\text{eff}_4\text{egh}_5\text{eeg}_6\text{efg}_9\ldots$$

By reducing space requirements, numerosity reduction speeds up the algorithm implementation as well as makes it possible to find motifs/anomalies with variable length.

2.3.7 Term Frequency - Inverse Document Frequency (TFIDF) Statistics

After the SAX algorithm has implemented the processes outlined in section 2.3.5 and 2.3.6, a bag of words representing the time series from the sliding window is produced. Bags are produced for each series in the training set after which these bags are combined into a single set called term frequency matrix where term refers to - one SAX word (Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018). Each column of this matrix represents a class from the training while the rows is the set of all SAX output words identified in all the groups (classes). An element a_{ij} in this matrix corresponds to the observed frequency of a term in a group and because some words observed in one class of a time series may not be identified in other classes, this matrix of wordbag sets is mostly sparse (Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018).

SAX-VSM then uses the output matrix to compute $tf * idf$ weights as outlined in equation 2.9 below. Each frequency is transformed into a weight. the weight for each term t is computed as the product of inverse term frequency (**idf**) and term frequency (**tf**). The log of the first term is taken to scale it.

$$\mathrm{tf}_{t,d} = \begin{cases} log(1+f_{t,d}), & \text{if} f_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.7}$$

t is the term

d is the set of bags (word bags)

$\mathrm{f}_{t,d}$ is the frequency associated with a term in a bag.

idf is calculated as follows:

$$\mathrm{idf}_{t,D} = \log \frac{|D|}{|d \in D : t \in d|} = \log \frac{N}{\mathrm{df}_t} \tag{2.8}$$

where; N is the set of bags cardinality

D : total number of classes

df_t : number of bags in which term t occurs.

Finally, the $tf * idf$ for t in the bag d of D set of bags is given as:

$$tf * idf(t,d,D) = tf_{t,d} \times idf_{t,D} = \log(1+f_{t,d}) \times \log \frac{N}{df_t} \tag{2.9}$$

$\forall$ instances where $f_{t,d} > 0$ and $\mathrm{df}_t > 0$, or zero otherwise.

After SAX-Vector Space Model (VSM) computes weights, creating a term weight matrix from the term frequency matrix, classification of the classes is computed using cosine similarity (see 2.10).

2.3.8 Cosine Similarity

For vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, cosine similarity is computed based on the definition of inner product and is given as follows:

$$\text{similarity}(\mathbf{a},\mathbf{b}) = cos(\theta) = \frac{a \cdot b}{\|a\|\|b\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{2.10}$$

Similarity values vary from 1 to -1 (exactly the same and opposite respectively). A value of 0 indicates orthogonality and since we use TFIDF weights (see Section (2.3)), the cosine similarity between word bags generated from our input series will range from 0 to 1 (Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018).

2.4 Filtering

In order to implement the LSTM, the data is first filtered to remove noise and observations that are not contributing much to the characteristic pattern of a disorder. Three methods were applied: low-pass, median and wavelet filtering. ECG fragments were first normalized between -1 and 1 before filtering.

2.4.1 Median Filtering

Median filtering uses a defined kernel size (odd scalar that defines the size of the median filter window) (Tyan 2006). The default kernel size is three (3) and so for a series X of length n, it is calculated as follows:

$$y_n = \text{median}(x_{n-k}, \ldots, x_n, \ldots x_{n+k}) \tag{2.11}$$

$\text{median}(x_{-k} \leqq \cdots \leqq x_0 \leqq \cdots \leqq x_k) = x_0$ for k odd, else, median = $\frac{x_0+x_1}{2}$

The algorithm picks observations of size k at a time from the ECG signals, obtains the median for each selection and creates a new data set of medians.

2.4.2 Low-pass Filtering

Low-pass filtering involves setting up thresholds (low and high). Signals are then passed through with only those below its cutoff allowed through while the signals outside the cutofff boundaries are weakened (Roger D. 2020). The steps involved are as follows:

1. define boundaries (lowcut and highcut)

2. define a threshold greater than the highest frequency observed

3. divide both lowcut and highcut by the frequency threshold

4. use the "butter" function to generate filter coefficients

5. filter the given signal using the pre-defined parameters in step 3

2.4.3 Wavelet Filtering

The wavelet transform is defined as follows:

$$\psi_{s,\tau} = \frac{1}{\sqrt{|s|}} \psi \left(\frac{t-\tau}{s} \right) \tag{2.12}$$

(Poungponsri and X.-H. Yu 2013)

It allows for signals to be represented in both frequency and time. The parameters are; dilation (scale) factor (s), the shift (translation) factor (τ), a basic wavelet $\psi(t)$ also reffered to as the mother wavelet and time (t). A signal $x(t)$ has a wavelet transform given by:

$$T(s,\tau) = \int_{-\infty}^{\infty} x(t) \psi^* \left(\frac{t-\tau}{s} \right) dt \tag{2.13}$$

In simple terms, a wavelet transform can be seen as the "cross-correlation" of a signal with wavelets of different "widths". The output of these filtering methods and how the best was chosen using the least Mean Squared Error (MSE) value is explained in Chapter (III).

CHAPTER III

DATA DESCRIPTION AND LSTM RESULTS

3.1 Data Description

Data used in this study was obtained from the Massachusetts Institute of Technology Beth Israel Hospital (MIT-BIH) arrhythmia database and consists of 1000, 10-second (3600 non overlapping samples) ECG fragments of 45 patients recorded at a frequency of 360 Hz and 200[adu/mV] gain. The data includes observations for 17 different classes; 15 heart disorders, a Normal Sinus Rhythm (NSR) and pacemaker rhythm (see Table 3.1) each of which had at least 10 fragments collected from lead one (MLII) (Pławiak 2018b).

3.1.1 Disorder Description

To ensure each disorder type was adequately represented in both the training and sample, classes with less than 3 patients were excluded and so, the final data used for the analysis did not include pre-excitation, Idioventricular rhythm, Ventricular flutter, second-degree heart block and the pacemaker rhythm. Data division was also done to ensure no patient in the training set was included in the test set. A 70-30 split using number of patients in each disorder class was first done. Observations were assigned to the training or test set using patient IDs to select fragments that belonged to each particular patient. In total, there were 876 observations (610 training and 266 test). Patient IDs as used in the original data were considered in the split to ensure no observations from the same patient appeared in both the training and test.

Table 3.1: Description of disorders and samples

No.	Class	Number of Fragments	Number of Patients
1	Normal Sinus Rhythm	283	23
2	Atrial Premature Beat	66	9
3	Atrial Flutter	20	3
4	Atrial Fibrillation	135	6
5	Supra-ventricular Tachyarrhythmia	13	4
6	Pre-excitation (WPW)	21	1
7	Premature Ventricular Contraction	133	14
8	Ventricular Bigeminy	55	7
9	Ventricular Trigeminy	13	4
10	Ventricular Tachycardia	10	3
11	Idioventricular Rhythm	10	1
12	Ventricular Flutter	10	1
13	Fusion of Ventricular and normal beat	11	3
14	Left Bundle Branch Block Beat	103	3
15	Right Bundle Branch Block Beat	62	3
16	Second-degree heart block	10	1
17	pacemaker rhythm	45	2
	Total	**1000**	**45**

3.1.2 Disorder Plots

The figure below (see Figure 3.1) is the plots for 3 of the disorders used in this study (that is; Normal Sinus Rhythm, APB and IVR). The QRS complexes peaks vary slightly from one disorder to the other as can be seen from the plot. For the normal case, the peaks can be seen around the mid-point of the distance between the highest and lowest point. For the second disorder, this same pattern can be observed but the QRS peaks are more widely spaced and range around the middle than that observed in the normal case. In the first disorder group however, the QRS complexes are closer to the endpoints of the plot and are more tightly squeezed together than the other two classes.

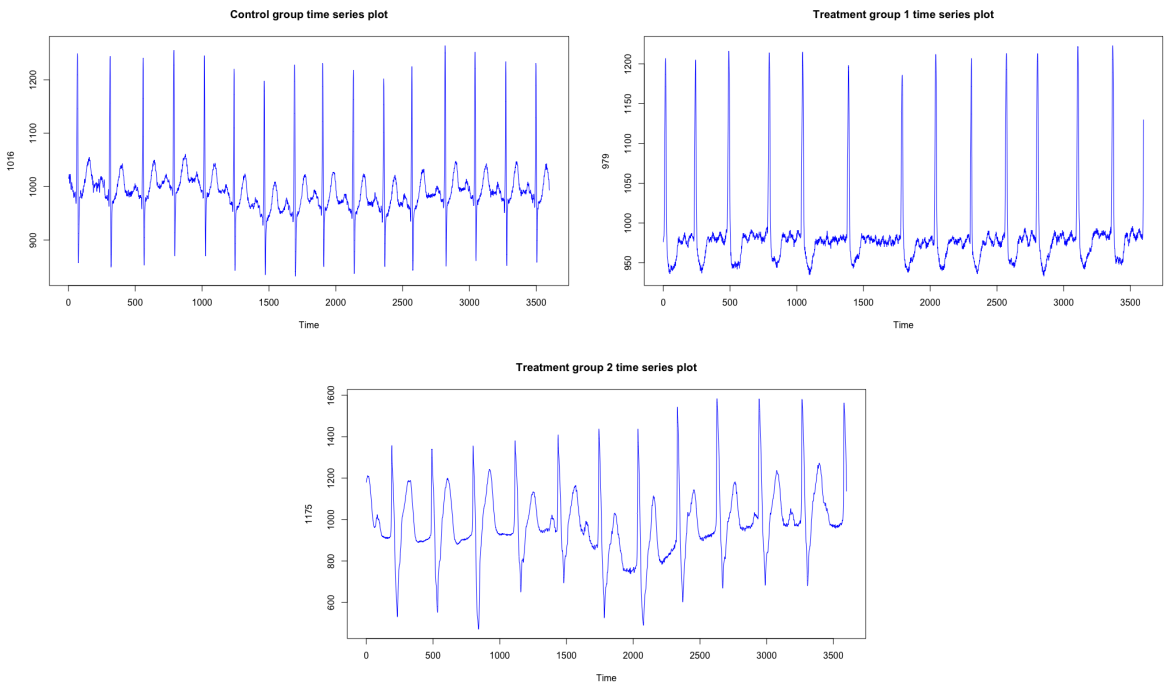

Figure 3.1: Plots for NSR, APB and IVR

3.2 LSTM Summary

In this section, LSTM is applied to the ECG signals to make classification for two classes at a time. The normal case (NSR) and each other disorder class. Three filtering methods were applied to remove noise: low-pass filtering, median filtering and wavelet filtering. The steps below outline how the LSTM was implemented.

- Data extraction: Load data
- Exploratory Data Analysis
- Pre-processing: split data into features and labels, normalize the data, filter using different algorithms and choose the best using the least observed Mean Squared Error (MSE).
- Training and test set split
- Feature extraction: extract relevant features from the data to train our model
- Train the model and set necessary parameter

- Model testing: calculate model predictions
- Evaluation of Model: assess model based on diagnostic measures (accuracy, AUC and their plots)

Table (3.2) below gives the MSE values obtained for the three filtering methods applied to the ECG data. It can be seen that the wavelet filtering gives the least MSE for all disorders. Further investigation to check this by making plots of the output data from each filtering method reveals the wavelet output almost always overlaps that of the original data (see Figure (3.3 and 3.6). The remaining plots for the filtered data for all disorder classes in the Appendix is also similar.

Table 3.2: Mean Squared Error For Different Filters

	Filtering method		
Class	Median	Low-pass	Wavelet
APB	0.0648	0.1531	0.0003
AFL	0.0698	0.1552	0.0004
AFIB	0.0527	0.1217	0.0003
SVTA	0.0694	0.1562	0.0004
PVC	0.0539	0.1229	0.0003
Bigenimy	0.0539	0.1230	0.0003
Trigenimy	0.0686	0.1550	0.0004
VT	0.0700	0.1555	0.0004
Fusion	0.0696	0.1628	0.0004
LBBBB	0.0588	0.1319	0.0003
RBBBB	0.0650	0.1099	0.0003

3.2.1 NSR and APB results

In Table (3.3), the best AUC value observed is 0.67 ($\pm$0.07) which corresponds with an accuracy of 63.7795 ($\pm$1.64). This value is consistent with the SAX results that gave a classification error of 0.3543 (0.6457 accuracy) as can be seen in Table (4.5). In the confusion matrix below, the difference between the actual positive predicted negative (FN) and actual positive predicted positive (TP) is not too large. However, the TN is much different from the FP.

Figure (3.2 and 3.3) are plots of a sample of 3 fragments (3 from the NSR and 3 from the APB class) and the filtered data plot obtained for the 3 filtering methods. It can be seen from

Table 3.3: LSTM AUC and Accuracy values

Class	AUC	Average AUC	Accuracy	Average Accuracy
APB	0.67		63.78	
	0.56	0.62 (±0.07)	61.42	60. 04 (±1.64)
	0.68		63.25	
AFL	0.53		86.32	
	0.85	0.72 (±0.17)	74.74	74.39 (±12.11)
	0.78		62.11	
AFIB	0.65		79.46	
	0.63	0.64 (±0.03)	72.32	74.49 (±4.40)
	0.59		71.43	
SVTA	0.86		73.86	
	0.78	0.74 (±0.11)	59.09	79.55 (±17.10)
	0.65		93.18	
PVC	0.74		67.44	
	0.72	0.72 (±0.03)	67.44	68.60 (±2.24)
	0.77		71.32	
Bigenimy	0.61		40.01	
	0.51	0.52 (±0.09)	65.91	58.56 (±14.77)
	0.68		39.77	
Trigenimy	0.49		94.19	
	0.38	0.45 (±0.06)	72.09	86.43 (±12.44)
	0.49		93.02	
VT	0.70		39.76	
	0.73	0.72 (±0.02)	45.78	44.18 (±3.87)
	0.73		46.99	
FUSION	0.79		58.33	
	0.78	0.80(±0.02)	57.14	60.32 (±4.51)
	0.82		65.48	
LBBBB	0.73		73.32	
	0.80	0.74(±0.05)	70.10	73.58 (±3.62)
	0.70		77.32	
RBBBB	0.61		43.29	
	0.59	0.59(±0.02)	40.21	43.30 (±3.09)
	0.57		46.39	

Figure (3.3) that the wavelet filtering with the least MSE as reported in Table (3.2) overlaps with the original ECG data as indicated on the plot (purple color). The error plots for the validation error does not look too convincing as it increases across the epochs.

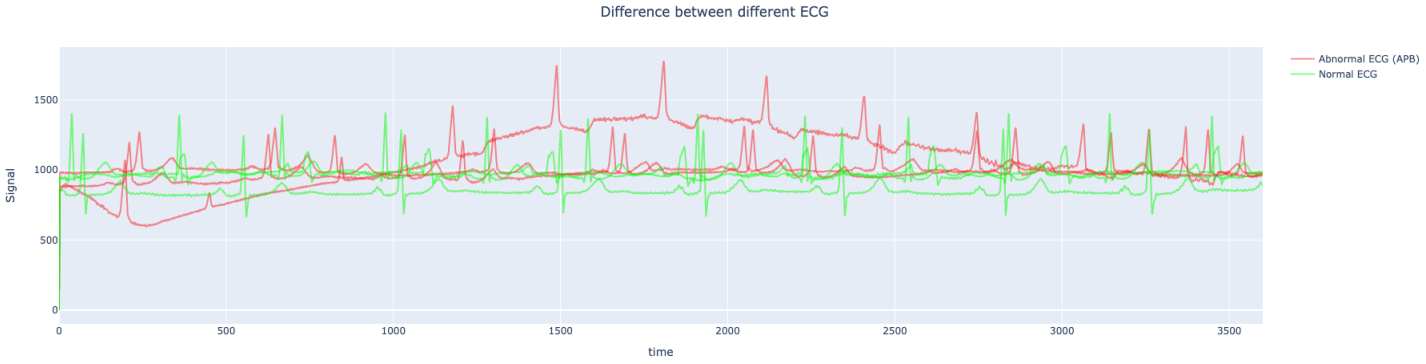

Figure 3.2: A Plot 3 Normal Fragments and 3 disorders (NSR and APB)

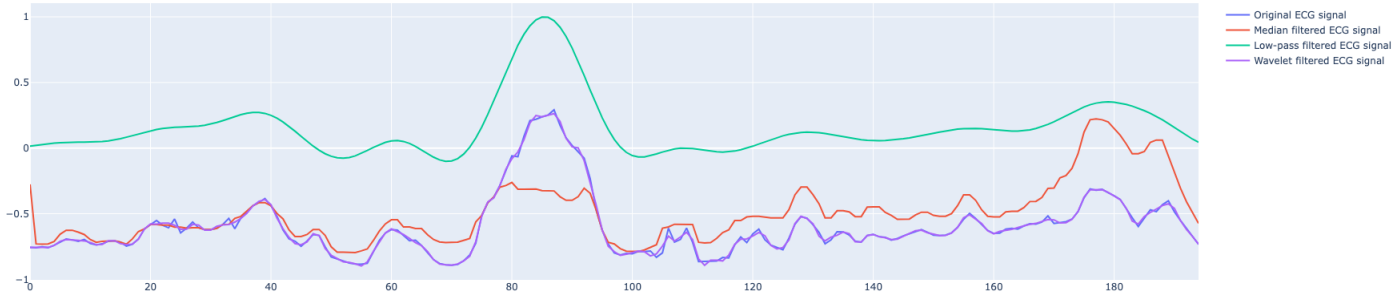

Figure 3.3: Filtered data output for the 3 methods(NSR and APB)

3.2.2 NSR and AFL results

From table (3.3), the best AUC is 0.85 ($\pm$0.17) with an accuracy of 74.7368 ($\pm$12.11). Figures (3.5 and 3.6) are the plots for a sample of 3 fragments as well as the plots for the original data and the output for the three filtering methods. As was observed in Figure (3.3) above, the data points after wavelet transform overlap with the original observations.

The performance of the LSTM on the NSR and AFL classes is also consistent with that obtained from the SAX. SAX reported an accuracy of 86.32 while LSTM gave 74.7368. Manipulating cut-off values for the predicted labels resulted in a closer value as seen in Table (3.3) however, the AUC reported for this was 0.53 which casts some doubts on our classification. Interestingly too, the validation error plot did not depict a decrease across epochs.

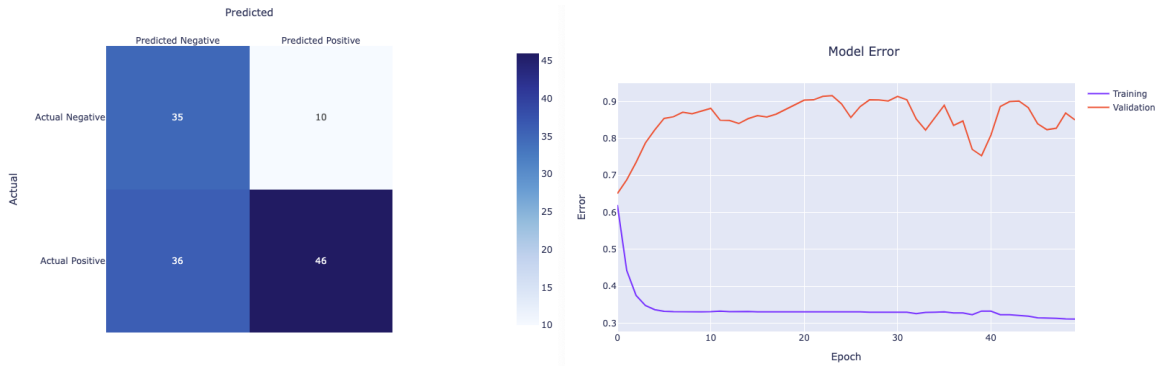

Figure 3.4: Confusion matrix and error plot for NSR and APB

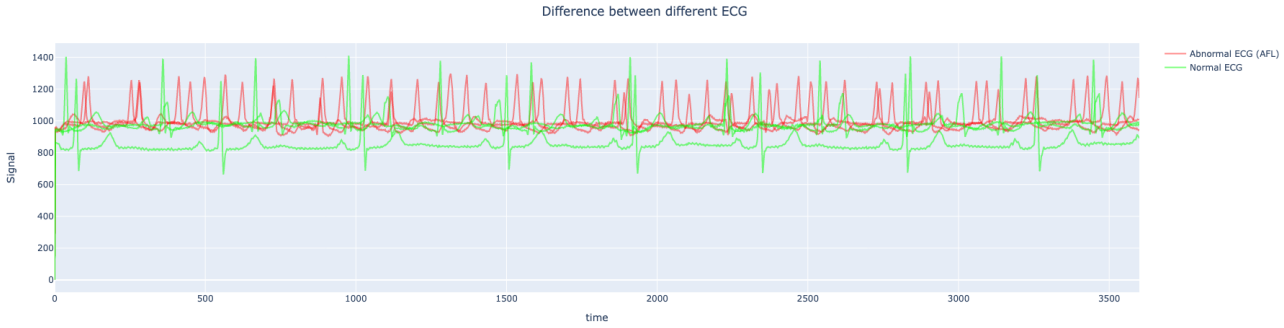

Figure 3.5: A Plot 3 Normal Fragments and 3 disorders(NSR and AFL)

3.2.3 NSR and AFIB results

The highest accuracy recorded for this run is 79.46 ($\pm$4.40) with an AUC of 0.65 ($\pm$0.031). The SAX table gives an error of 0.2679 as compared to 0.2054 from the LSTM which gives a hint of just a little variation in values but similar conclusions.

The filtering output for the ECG signals in Figure (3.9) is similar to that observed in the previous classes. The median filter in this case yielded values close to the original signals but did not perform as well as the wavelet transform. Hence, we can observe some gaps in the plot between the original signal and median filtered data as well as direct overlaps. The FP value as shown in the confusion matrix (see Figure (3.10)). On the other hand, the TP is vastly different from the FN. Unlike the plots for APB and AFL, Figure (3.9) shows a similar pattern observed for AFIb. Median filtering shows a good overall fit to the original signals but exhibit some gaps and has a

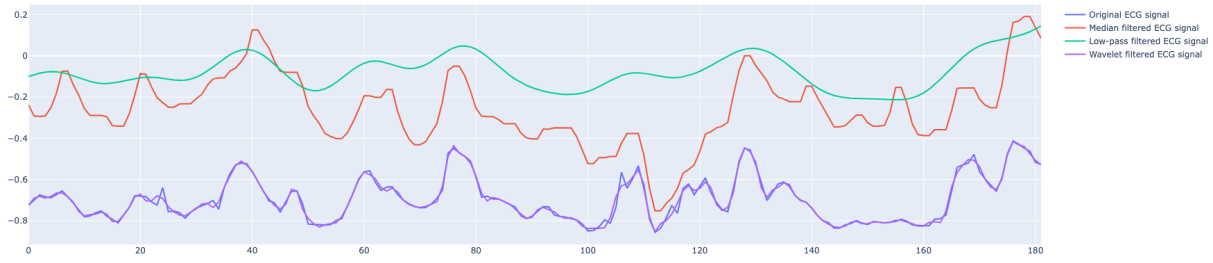

Figure 3.6: Filtered data output for the 3 methods (NSR and AFL)

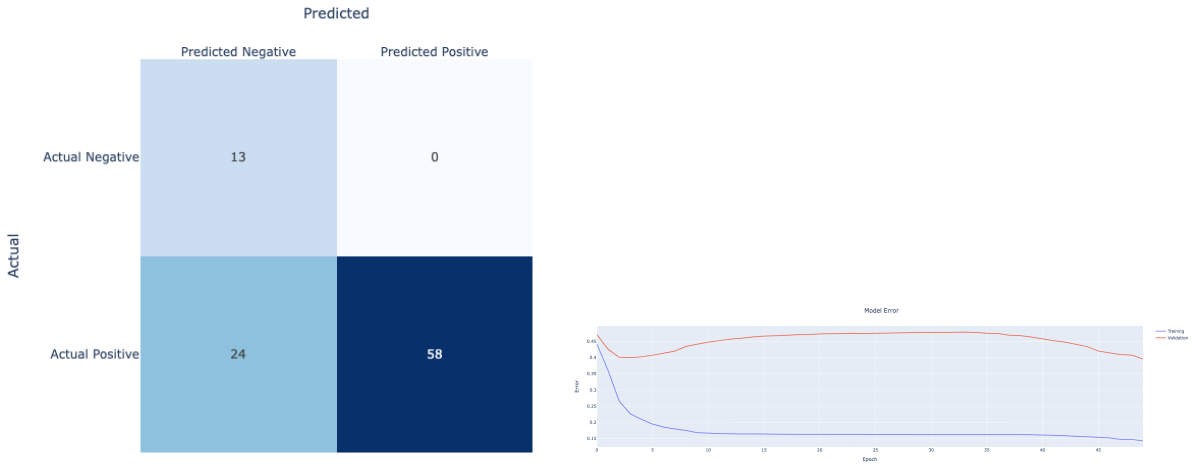

Figure 3.7: Confusion matrix and error plot for NSR and AFL

bigger MSE than wavelet filtering which was also applied to the data before the LSTM for this class was implemented.

3.2.4 NSR and SVTA results

Overall, the highest AUC and accuracy were observed for the SVTA class. The highest AUC observed was 0.86 ($\pm$0.11) while the highest accuracy was 93.1818 ($\pm$17.10). There was however, a vast difference in accuracy scores between SAX and LSTM but from table (3.3), an observed accuracy of 73.86 matches that observed for SAX in table (4.5).

Both the TP and TN as compared to the FP and FN in the confusion matrix (Figure 3.11) show significant differences that attest to the good performance of our model. The validation and training error also show a descent across the epochs as preferred.

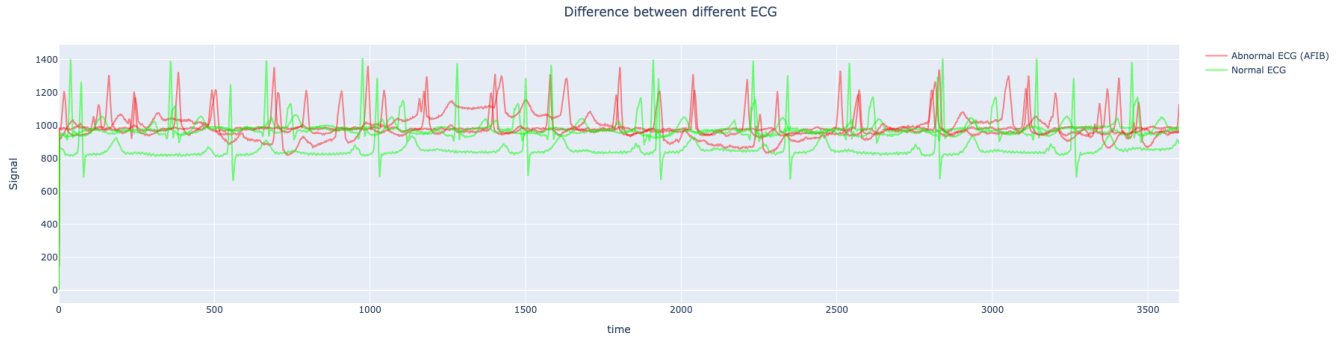

Figure 3.8: A Plot 3 Normal Fragments and 3 disorders (NSR and AFIB)

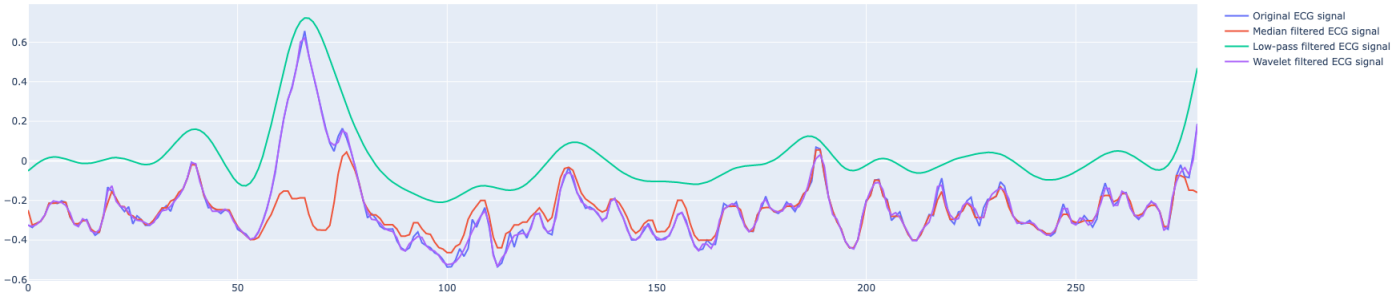

Figure 3.9: Filtered data output for the 3 methods (NSR and AFIB)

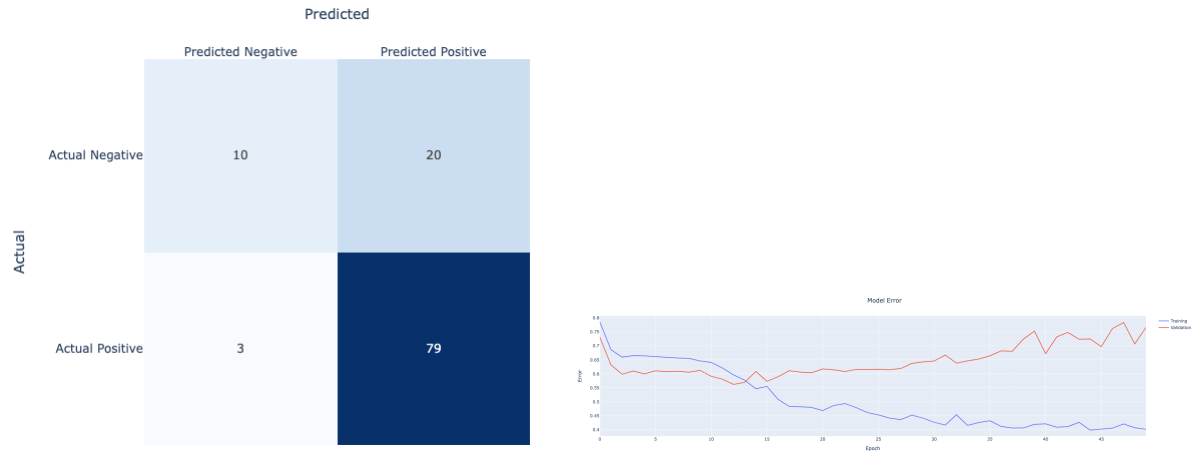

Figure 3.10: Confusion matrix and error plot for NSR and AFIB

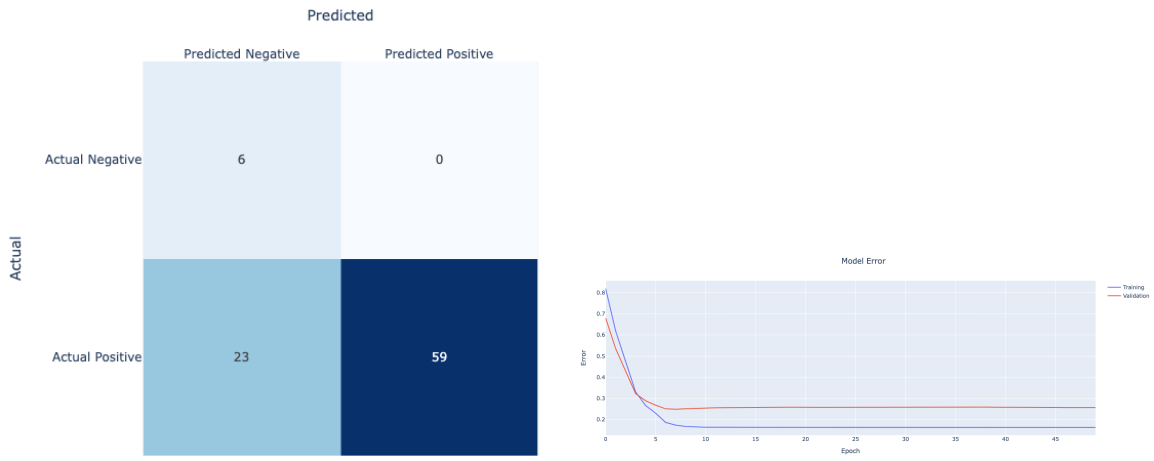

Figure 3.11: Confusion matrix and error plot for NSR and AFL

3.2.5 NSR and PVC results

For the PVC class, the best AUC was 0.77 ($\pm$0.03), observed with an accuracy of 71.32 ($\pm$2.24). The AUCs observed for different cutoffs were more consistent than other classes. The confusion matrix (Figure C.1) shows a very good prediction for the negative classes while the FN and TP do not show vast differences and could be claimed to be balanced. The error plot looks convincing too as a continuous decrease across the epochs is seen.

3.2.6 NSR and BIGENIMY results

The best AUC observed for Bigenimy was 0.68 ($\pm$0.09) with an accuracy of 39.77 ($\pm$14.77)). These values are low compared to the previous classes but this could be attributed to the number of fragments present in this class. Disorder classes that had fewer observation in the training and test had lesser metric scores. The confusion matrix does not show a very good prediction (Figure C.2) even though the validation error plot shows a continuous decrease.

3.2.7 NSR and TRIGENIMY results

Figure (C.4) shows the confusion matrix for this class. A significant TP rate is identified but our model fails to predict any TN. This is also evident in Table (3.3) where we can observe high accuracies ($\pm$12.43) and low AUC ($\pm$0.06) values.

3.2.8 NSR and VT results

AUC values for this class were >0.70 ($\pm$0.017) but accuracies recorded were < 50%. This class had just nine(9) fragments in the training and test as compared to 257 for the normal ECG signal (see Table 4.4). This makes results doubtful due to the huge discrepancy in sample sizes for the two classes being compared. The error plot Figure C.3 shows a downward trend but the confusion matrix shows a very poor prediction.

3.2.9 NSR and FUSION results

FUSION had one of the least observations (11 fragments) but the metric scores were relatively good. Highest AUC was 0.82 ($\pm$0.02) with the highest accuracy as 65.48 ($\pm$4.51). HOT-SAX gave an accuracy of 97.62 which supports the results with a bigger margin.

3.2.10 NSR and LBBBB results

LBBBB had the 3rd highest number of ECG signals in the dataset (103) and so good metric scores were achieved Highest AUC for this class is 0.80 ($\pm$0.05) with the accuracy ranging from >70 ($\pm$3.62).

3.2.11 NSR and RBBBB results

Despite having 62 fragments, classification for RBBBB was not as convincing as other classes that had fewer ECG signals.

CHAPTER IV

SAX AND MOTIF IDENTIFICATION

This chapter highlights the results of the SAX implementation. It includes, parameter estimation, weighted pattern plots, motif identification for brute force and HOT-SAX and comparisons between methods.

4.1 Optimal SAX Parameters

Table 4.1 below gives a summary of the optimal parameter outcomes for the window size, PAA divisions and alphabet size for a given number of iterations and boundary specifications with different cross validations. After several executions with different thresholds, the persisting values of the parameters observed (optimal parameter values) were obtained and subsequently used in the SAX implementation. These are: window size = 62, PAA divisions = 31 and alphabet size (number of alphabets) = 6. These runs were done for a selection of fragments as well as the full dataset and for each execution, similar results were obtained.

Table 4.1: Optimal SAX parameters

Nfolds	Lower & Upper Boundaries	Iterations	Optimal w,p,a	Error
15	c(10,2,2) - c(120,60,12)	10	(28,50,7)	0.5000
15	c(3,2,2) - c(120,60,15)	10	(62,31,8)	0.5000
15	c(3,2,5) - c(120,60,10)	10	(62,31,6)	0.4167
15	c(3,2,5) - c(120,60,10)	20	(62,31,6)	0.4167
10	c(3,2,5) - c(120,60,10)	20	(62,31,6)	0.4167
10	c(3,2,2) - c(120,60,10)	10	(62,31,6)	0.4167
10	c(3,2,2) - c(120,60,12)	10	(62,31,6)	0.8872

SAX is an amplitude-based quantization method as shown in Figure (2.6 & 2.7). As such, too many alphabets/characters reduces efficiency of the algorithm leading to too many

representations that create redundancies and make ideal pattern discovery problematic.

4.2 Discord Discovery algorithms

4.2.1 Brute Force

Brute force algorithms do not approach problems in the would be preferred precise, efficient and deliberately planned method. Brute force, relies on the application of force, sequentially going through the entire length to find all possible matches, compute errors and replace better outcomes with the best one until the end of the sequence. For brute force discord discovery, using specified parameters, each likely sub-sequence is identified by the algorithm through the entire time series. Distances are computed as outlined in Chapter II and the output is presented in Table (C.1) (Woodbridge et al. 2015). The brute force is an alternative to HOT-SAX although the latter has been identified to perform better. Brute force uses the distance measure to compute distances between closest non-matching discords in the entire length of the series for all identified matches.

Table 4.2: Brute Force and HOT-SAX Discovered Discords

		Brute Force		**HOT-SAX**	
Class	No of discords	nn_distance	Position	nn_distance	Position
APB	1	197.91	686	152.23	2618
AFL	1	110.21	1045	90.87	401
AFIB	1	120.43	1187	161.31	2293
SVTA	1	226.39	3399	226.39	3399
PVC	1	991.60	3361	1336.18	1941
BIGENIMY	1	536.49	2414	604.85	1882
TRIGENIMY	1	285.38	3535	168.39	2547
VT	1	579.46	2700	579.46	2700
FUSION	1	677.33	3325	677.33	3325
LBBBB	1	153.00	475	1079.87	2428
RBBBB	1	169.55	3415	152.26	0

4.2.2 HOT-SAX

Table C.1 and Table C.2 summarized in Table 4.2 gives the best discords identified across the series for each class as well as its position and distance. The *nn_distance* gives a measure of the distance between an identified discord and the next similar pattern. It is calculated using

Equation (2.9) in Chapter 2. The *position* in the table, is the location in our time series where the discord is identified. Comparing Table (C.2) and (C.1), we can see that out of the 11 classes, SVTA, VT and FUSION have the same *nn_distance* for both algorithms while AFIB, PVC and LBBBB (*nn_distance* < HOT-SAX) are the exceptions to what exists in literature that SAX gives the minimum distance lower bound among discord discovery algorithms.

Interestingly, classes with the same *nn_distance* for both brute force and HOT-SAX also had the same position at which these discords were located whereas the other classes all had different positions and *nn_distances*. According to (Senin, Lin, Wang, Oates, Gandhi, Arnold P Boedihardjo, et al. 2014), the HOT-SAX approach is faster to execute than the brute force. Figure (4.1), depicts a plot of each class observations (blue) and an indication of the discord (red). It is evident from the plots that the red portions are somewhat different from the other peaks on the same plot. Interestingly, these anomalies occur somewhat around the middle of the plot with none appearing close to the end. Only one plot from the VT class has an identified discord close to the start.

4.2.3 HOT-SAX Multi-discord

Table (4.3) summarizes the three of the most important discords identified for all classes together with their position, the distances and the distance_calls. It can be seen that these discords occur at different positions and the distances to the nearest non-self match also show differences between them.

(Table 4.4) summarizes the segment distribution across the disorders as well as the average number of distinct fragments used in each class. It can be seen from the table that the NSR has the highest number of ECG data fragments followed by AFIB with 105 and then LBBBB with 88. Number of distinct fragments in each class is not proportional to the size of observations. There are several different measurements for the same fragment giving our training and test set multiple observations for the same class.

Table 4.3: HOT-SAX Multi-Discord Discovery

Class	No of discords	Position	nn_distance
APB	3	187.5020	2954
		155.5056	1484
		143.3946	412
AFIB	3	161.3103	2293
		68.6659	12
		62.2254	208
AFL	3	442.5890	230
		425.7628	2889
		408.3173	3068
SVTA	3	264.6658	2018
		227.5390	1883
		181.6398	2971
PVC	3	449.6321	1946
		103.3586	3350
		102.3572	1758
BIGENIMY	3	133.3192	572
		128.2459	3045
		119.6411	2596
TRIGENIMY	3	148.3408	1453
		114.9522	213
		114.9522	2685
VT	3	158.3888	39
		151.8058	2420
		148.1857	209
FUSION	3	704.2088	1899
		522.2193	1524
		443.1365	3461
LBBBB	3	1079.8694	2428
		525.5568	2536
		177.0678	2975
RBBBB	3	1605.0210	3183
		1316.3570	3247
		123.9960	3309

4.2.4 HOT-SAX Misclassification Errors

Table (4.5) summarizes the classification errors for the SAX algorithm taking the NSR and each of the other eleven classes at a time. Computing the proportion of the number of observations in each class (excluding NSR) in relation to total observations (NSR + observations for particular

Table 4.4: Classes and Segments

Class	Total number of fragments	Number of patients	Number of distinct fragments Train	Test	Average
NSR	201	23	16	7	12.56
APB	20	9	6	3	3.33
AFL	7	3	2	1	3.50
AFIB	105	6	4	2	26.25
SVTA	7	4	3	1	2.33
PVC	86	14	10	4	7.81
BIGENIMY	49	7	5	2	9.80
TRIGENIMY	9	4	3	1	3.00
VT	8	3	2	1	4.00
FUSION	9	3	2	1	4.50
LBBBB	88	3	2	1	44.00
RBBBB	47	3	2	1	23.50
Total	**636**	**58**			

Table 4.5: Classification Errors Between NSR and Other Classes

Class	Fragments in class	Misclassified observations	HOT-SAX Misclassification Error	Weighted Error
NSR	82			
APB	45	45	0.3543	0.0079
AFL	13	13	0.1368	0.0105
AFIB	30	30	0.2679	0.0089
SVTA	5	34	0.3864	0.0773
PVC	47	51	0.3953	0.0084
BIGENIMY	6	13	0.1477	0.0246
TRIGENIMY	4	4	0.0465	0.0116
VT	1	1	0.0120	0.0120
FUSION	2	2	0.0238	0.0119
LBBBB	15	15	0.1546	0.0103
RBBBB	15	15	0.1546	0.0103

class) shows that SAX does a very good job classifying our data as the errors are consistent with the proportions using number of observations present. Despite runs of the LSTM (III) giving slightly different error values with different specified cutoff values for the predicted, it can be observed that there is at least one accuracy consistent with the SAX output.

4.2.5 Discord Plots

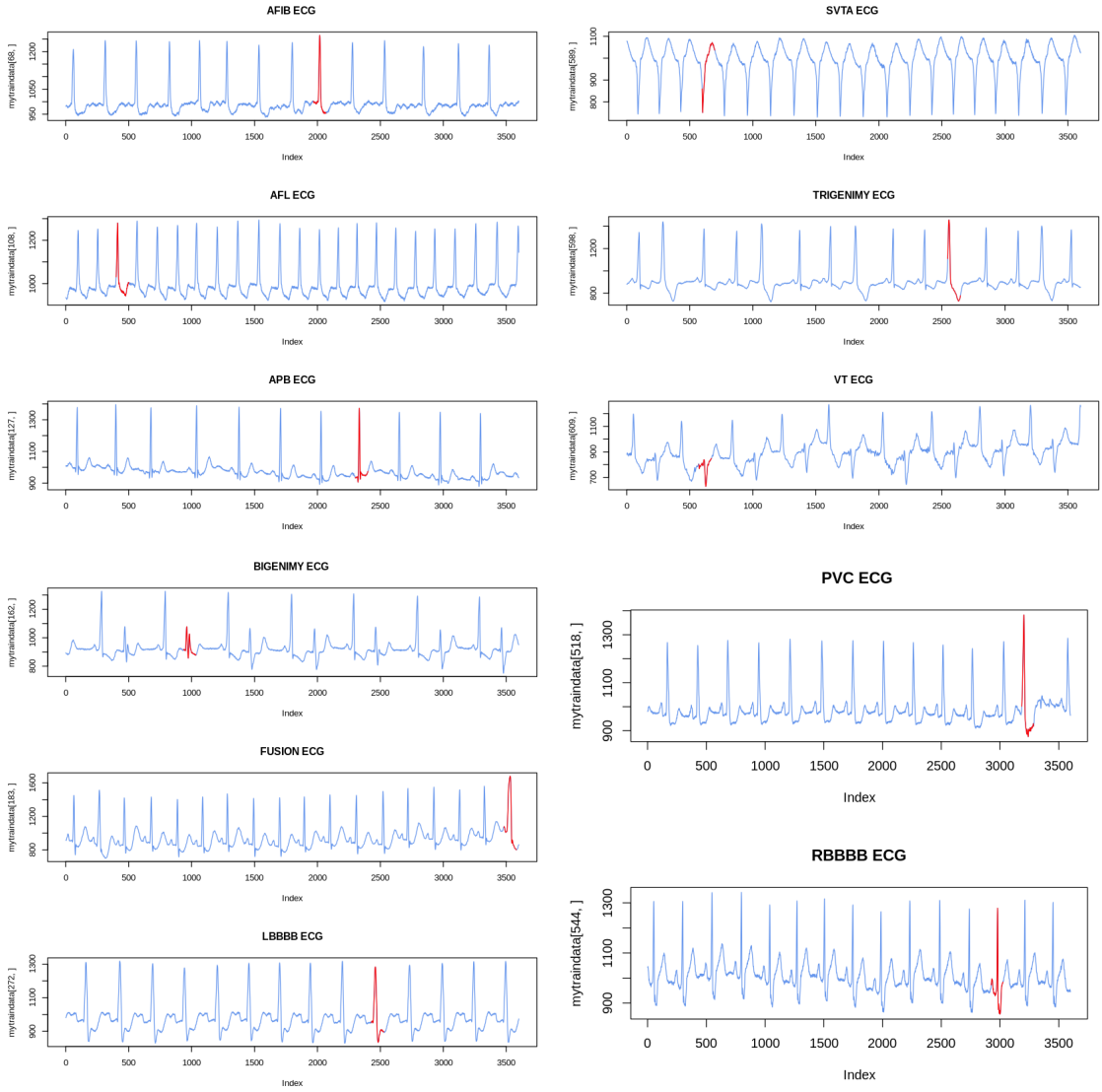

Figure 4.1: HOT-SAX Discord Plot For Disorders

Figure (4.1) displays the most important (in red) identified with the execution of the HOT-SAX algorithm summarized in Table (C.2). It is obvious that the shape/peaks look slightly different from the remaining QRS complexes across the entire plot further confirming the results of the identification made by SAX and the location it provided.

4.2.6 Multi-discord Plots

Figure (4.2) is a plot for 3 of the best discords identified for all classes as summarized in Table (4.3). It includes the best discord as first identified in Table (C.2) and two subsequent best discords shown in red, black and green respectively. SVTA has all 3 discords almost overlapping as well as FUSION and RBBBB. NSR has the highest distance between the observed discords while PVC, TRIGENIMY and VT have only two discords overlapping.

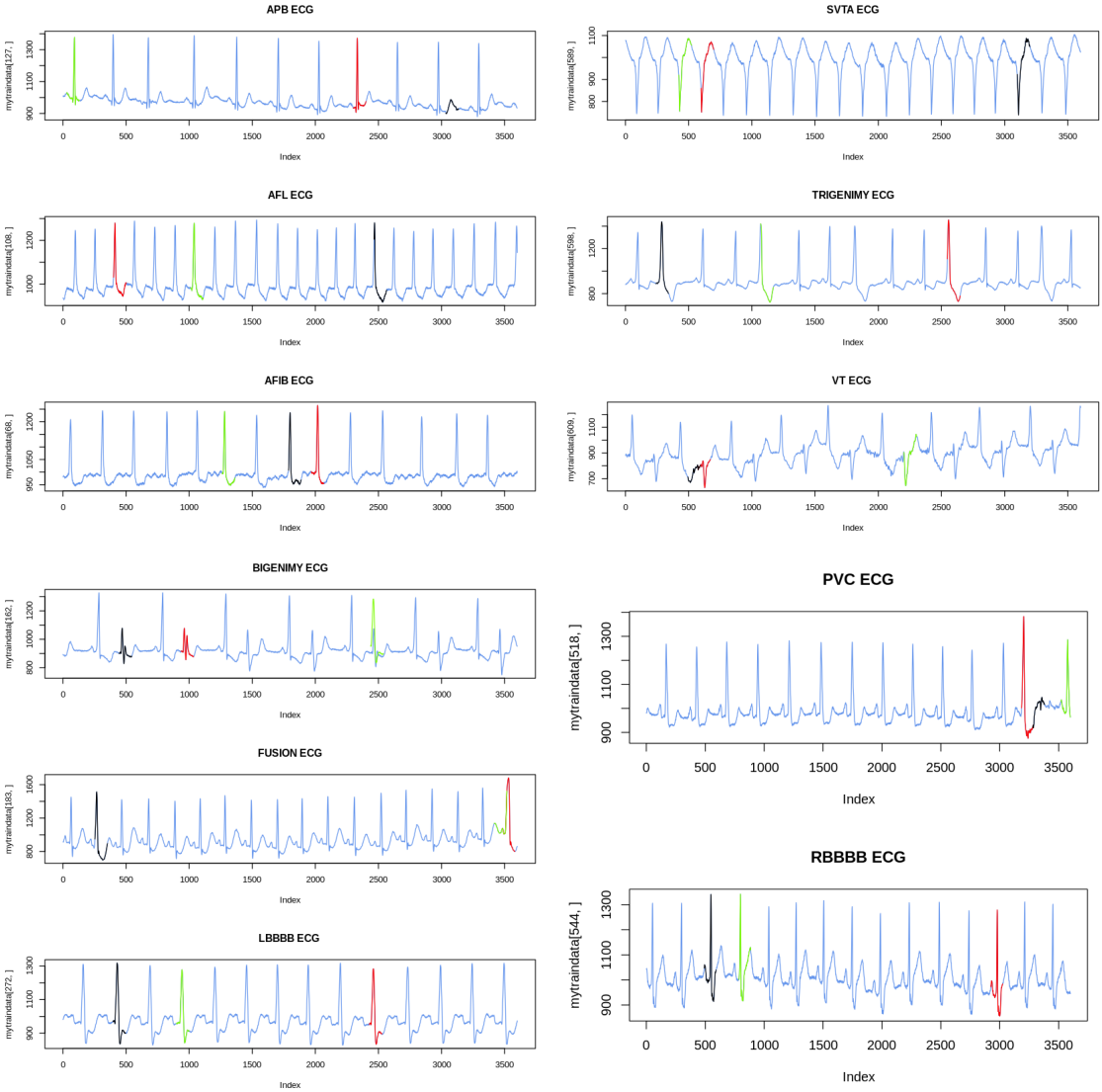

Figure 4.2: Multi-discord plot for disorders

4.3 Weighted Pattern Plots

Figures (4.3, 4.4, 4.5) give the weighted pattern plots of each disorder class against the NSR (that is;APB, AFL, AFIB, SVTA, PVC, Bigenimy, Trigenimy, VT, Fusion, LBBBB and RBBBB). The color discrepancies depict the differences between each disorder and the normal ECG readings. These differences are color coded (negative, neutral and high) to show how high or low the differences are. For each word in the wordbag, neutral means the weight of that word for both the disorder under consideration and the NSR are equal which may imply no disturbing patterns in those positions. High means that the weight for a word in the disorder wordbag is more than that observed for the same word in the normal case. In the implementation of SAX, a bag of words with corresponding counts for each word is generated. These bag of words for each class are combined to form the *tfidf* weights as described in Equation (2.9). The *tfidf* output contains the most relevant words and their weights across each disorder.

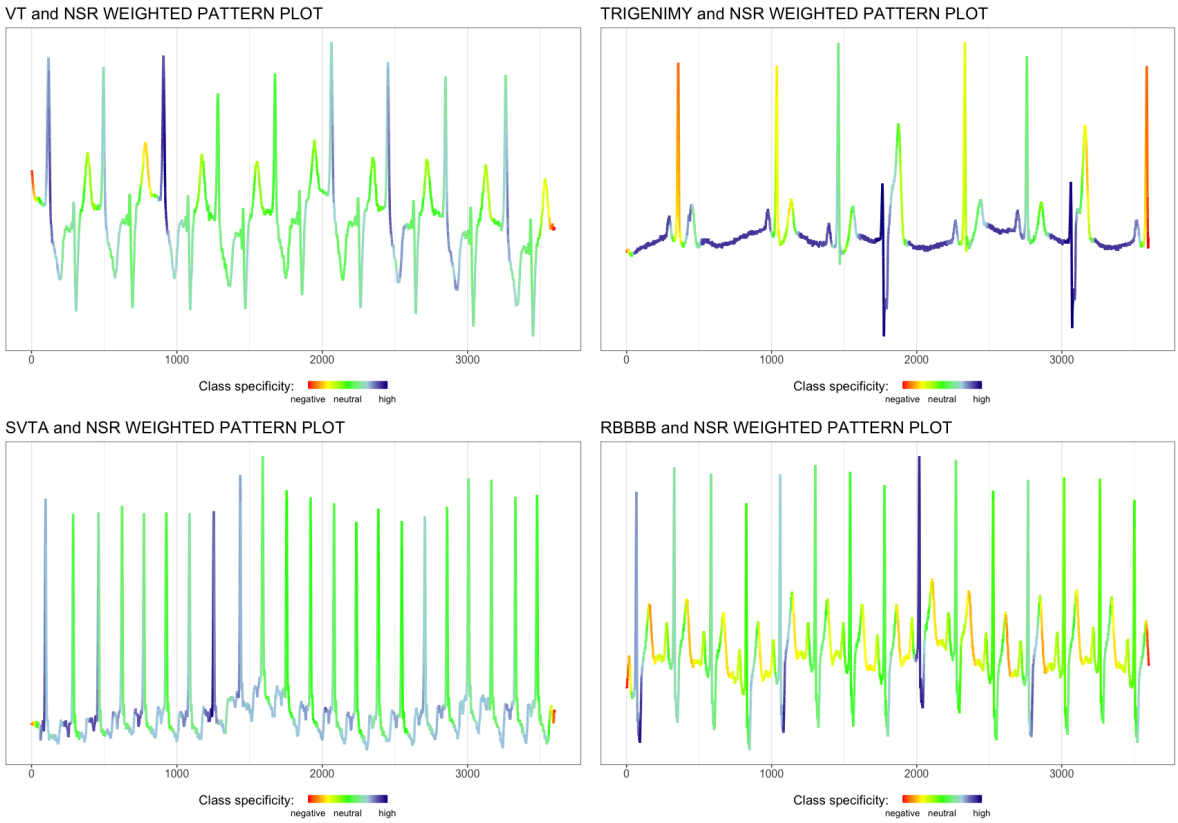

Figure 4.3: Weighted-pattern plots for VT, Trigenimy, SVTA and RBBBB

For the plots above and below, a negative (orange-red) indication on the plot would mean a negative value between the weights observed for that particular disorder and the normal case. In the same way, neutral would mean balanced weights while high (violet) would mean a particular word occurred more in the disorder than in the NSR. It is interesting to note that all plots have representations of neutral weights (green) which is an indication of equal weights observed in the *tfidf* statistics. This is not very surprising as all the ECG fragments for the various disorders all follow the same underlying pattern of peaks and falls (QRS complexes) and not the normal trends observed in many time series data.

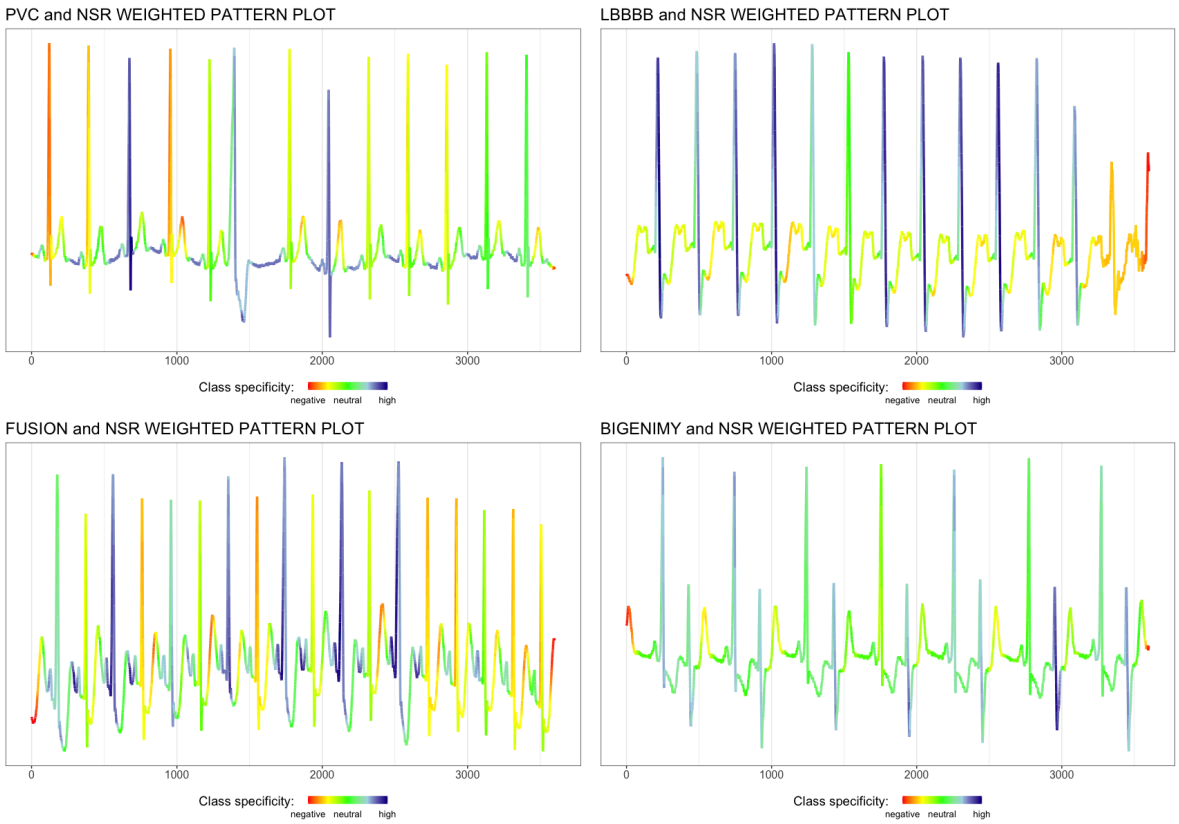

Figure 4.4: Weighted-pattern plots for PVC, LBBBB, Fusion and Bigeminy

Comparing the first plot in Figure (4.3) VT and the fourth plot in Figure (4.4) (Bigenimy), we can see these weighted pattern plots show more neutral than the rest. Even without checking numerical measures, these plots are very similar to that observed in the first plot in Figure (3.1).

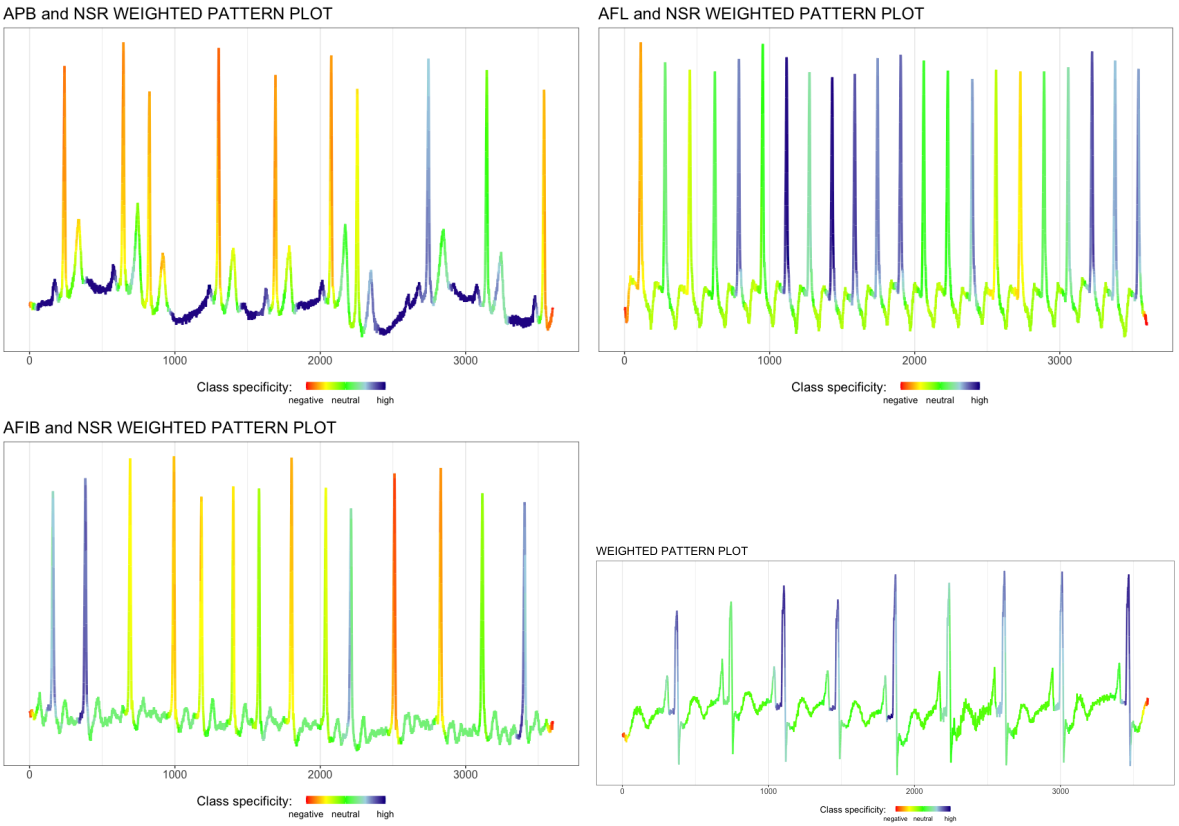

Figure 4.5: Weighted-pattern plots for APB, AFL, AFIB and overall

CHAPTER V

DISCUSSION AND FUTURE RESEARCH

5.1 Discussion

This project aimed at identifying heart disorders by implementing the SAX algorithm to discretize the numeric data and then apply the SAX-VSM approach to identify motifs and observe abnormalities in the ECG data. The study also implemented the LSTM neural network to classify the ECG signals after pre-processing to eliminate unwanted noisy features. The results of the SAX classification errors shows relatively better performance with low errors as presented in Table (4.5).

Further analysis was done to identify the most important discords for each class as summarised in Table (C.2) and pictorially shown in Figure(4.1). The distance measure and position of the best discords identified by HOT-SAX depict huge discrepancies between each of the disorder classes and normal ECG readings. This highlights differences in the ECG signals and suggests the presence of varying heart conditions as we initially sought to find out.

Analyzing further, we identified the three most important discords as summarized in Table (4.3) for each class. These discords are shown in Figure (4.2) as the red, green and black highlights. As was identified for single discords in Figure (4.1), the positioning of these irregular patterns suggest the presence of vital differences between the ECG signals. Similar location of best discords (Table 3.1), equal distance measures (Table C.2) and tightly overlapping raw ECG signal plot (Figure 3.1, and 3.8) would mean all signals collected are the same and conform to same heart condition but this was not the case observed.

Classes with significantly large number of ECG fragments in the dataset showed better metric scores (see Table 3.3) that were consistent with the results obtained from SAX (Table 4.5).

For classes that had fewer ECG signals in the training set, AUC and accuracy values were low and confusion matrices showed poor prediction that favors higher FP and FN values. Some of the validation and training error plots for the disorders based on the LSTM showed either a slight decrease or none at all. This could be attributed to the LSTM being unable to identify the exact location of discords, check for repetitions using distances before computing errors as SAX does. LSTM focused more on the numerical values of the ECG signals and not the position of anomalies present.

5.2 Study Limitation

One challenge with the implementation of SAX is the choice of optimal parameters. That is; window size, number of PAA divisions and appropriate number of alphabets to represent the equi-probable regions. A smaller PAA value means the string representations would not capture the real fluctuations (spikes and dips) in the original data set. Similarly, more divisions may lead to too many alphabet representations that results in many repeated sequences which may not necessarily represent the underlying behaviour that should be observed in the data. Furthermore, symbols are assigned to the mean of observations in each defined segment which could result in varying segments with equal averages being assigned the same symbols even though they may represent a different variation in the series. Hence, trends and underlying characteristics may not be captured ((Sun et al. 2014), (Ren et al. 2018)).

To tackle this, different validation values for different iterations were used to estimate errors for the SAX implementation to our ECG data by setting thresholds (minimum and maximum) for all three parameters (see Table 4.1). After several runs, the window size, alphabet size and number of PAA divisions that gave the minimum error was obtained.

5.3 Recommendation for Future Research

This study uses SAX, a quantization technique that does the characterization based on the amplitude domain it is recommended that future studies make use of temporal segmentation methods such as persist that make PAA divisions based on time domain, group similar observations

before discretizing to make comparison of results (see Figure 1 in (Sant'Anna and Wickström 2011)). It is also recommended that a more representative dataset be used as our data had more ECG fragments for the normal heart beats as compared to the remaining disorders. For future research, the researcher plans to simulate ECG signals that can be analysed and results compared with the signals obtained from the MIT-BIH database. ECG signals and their behaviour would be better understood by a cardiologist. As such, it is recommended that results be presented to a medical expert to evaluate our algorithm output with that of medical validation techniques being used.

REFERENCES

Berkaya, Selcan Kaplan et al. (2018). "A survey on ECG analysis". In: *Biomedical Signal Processing and Control* 43, pp. 216–235.

Biel, Lena et al. (2001). "ECG analysis: a new approach in human identification". In: *IEEE transactions on instrumentation and measurement* 50.3, pp. 808–812.

Bonow, Robert O et al. (2011). *Braunwald's heart disease e-book: A textbook of cardiovascular medicine*. Elsevier Health Sciences.

Chatfield, Chris and Haipeng Xing (2019). *The analysis of time series: an introduction with R*. CRC press.

Chen, Tung-Bo and Von-Wun Soo (1996). "A comparative study of recurrent neural network architectures on learning temporal sequences". In: *Proceedings of International Conference on Neural Networks (ICNN'96)*. Vol. 4. IEEE, pp. 1945–1950.

Cryer, Jonathan D (1986). *Time series analysis*. Vol. 286. Duxbury Press Boston.

Daw, C Stuart, Charles Edward Andrew Finney, and Eugene R Tracy (2003). "A review of symbolic analysis of experimental data". In: *Review of Scientific instruments* 74.2, pp. 915–930.

Elman, Jeffrey L (1991). "Distributed representations, simple recurrent networks, and grammatical structure". In: *Machine learning* 7, pp. 195–225.

Fuller, Wayne A (2009). *Introduction to statistical time series*. John Wiley & Sons.

Gers, Felix A, Jürgen Schmidhuber, and Fred Cummins (2000). "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10, pp. 2451–2471.

Greff, Klaus et al. (2016). "LSTM: A search space odyssey". In: *IEEE transactions on neural networks and learning systems* 28.10, pp. 2222–2232.

Hamilton, James Douglas (2020). *Time series analysis*. Princeton university press.

He, Zengyou, Xiaofei Xu, and Shengchun Deng (2003). "Discovering cluster-based local outliers". In: *Pattern recognition letters* 24.9-10, pp. 1641–1650.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Huang, Tian et al. (2015). "J-distance discord: an improved time series discord definition and discovery method". In: *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE, pp. 303–310.

Israel, Steven A et al. (2005). "ECG to identify individuals". In: *Pattern recognition* 38.1, pp. 133–142.

Karpathy, Andrej, Justin Johnson, and Li Fei-Fei (2015). "Visualizing and understanding recurrent networks". In: *arXiv preprint arXiv:1506.02078*.

Keogh, Eamonn et al. (2001). "Dimensionality reduction for fast similarity search in large time series databases". In: *Knowledge and information Systems* 3, pp. 263–286.

Kha, Nguyen Huy and Duong Tuan Anh (2015). "From cluster-based outlier detection to time series discord discovery". In: *Trends and Applications in Knowledge Discovery and Data Mining*. Springer, pp. 16–28.

Kirchgässner, Gebhard, Jürgen Wolters, and Uwe Hassler (2012). *Introduction to modern time series analysis*. Springer Science & Business Media.

Krishnamoorthy, Vignesh (2018a). *Piecewise Aggregate Approximation*. Ed. by Vigne Krishnamoorthy. URL: `https://vigne.sh/posts/piecewise-aggregate-approx/`.

— (2018b). *Symbolic Aggregate Approximation*. Ed. by Vigne Krishnamoorthy. URL: `https://vigne.sh/posts/symbolic-aggregate-approximation/`.

Kulahcioglu, Burcu, Serhan Ozdemir, and Bora Kumova (2008). "Application of symbolic Piecewise Aggregate Approximation (PAA) analysis to ECG signals". In: *17th IASTED International conference on applied simulation and modelling*. Citeseer.

Li, Shuai et al. (2018). "Independently recurrent neural network (indrnn): Building a longer and deeper rnn". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5457–5466.

Lin, Jessica, Eamonn Keogh, Stefano Lonardi, et al. (2003). "A symbolic representation of time series, with implications for streaming algorithms". In: *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pp. 2–11.

Lin, Jessica, Eamonn Keogh, Li Wei, et al. (2007). "Experiencing SAX: a novel symbolic representation of time series". In: *Data Mining and knowledge discovery* 15, pp. 107–144.

Mohammad, Yasser and Toyoaki Nishida (2009). "Constrained motif discovery in time series". In: *New Generation Computing* 27.4, pp. 319–346.

Mörchen, Fabian and Alfred Ultsch (2005). "Optimizing time series discretization for knowledge discovery". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 660–665.

Mueen, Abdullah (2014). "Time series motif discovery: dimensions and applications". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 4.2, pp. 152–159.

Mueen, Abdullah and Eamonn Keogh (2010). "Online discovery and maintenance of time series motifs". In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1089–1098.

Ohsaki, Miho, Hidenao Abe, and Takahira Yamaguchi (2007). "Numerical time-series pattern extraction based on irregular piecewise aggregate approximation and gradient specification". In: *New Generation Computing* 25.3, pp. 213–222.

Pławiak, Paweł (2018a). "Novel methodology of cardiac health recognition based on ECG signals and evolutionary-neural system". In: *Expert Systems with Applications* 92, pp. 334–349.

— (2018b). "Novel methodology of cardiac health recognition based on ECG signals and evolutionary-neural system". In: *Expert Systems with Applications* 92, pp. 334–349. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.09.022. URL: https://www.sciencedirect.com/science/article/pii/S0957417417306292.

Poungponsri, Suranai and Xiao-Hua Yu (2013). "An adaptive filtering approach for electrocardiogram (ECG) signal noise reduction using neural networks". In: *Neurocomputing* 117, pp. 206–213.

Ren, Huorong et al. (2018). "Anomaly detection using piecewise aggregate approximation in the amplitude domain". In: *Applied Intelligence* 48.5, pp. 1097–1110.

Roger D., Peng (2020). *A Very Short Course on Time Series Analysis*. Ed. by Peng Roger. URL: https://bookdown.org/rdpeng/timeseriesbook/.

Sant'Anna, Anita and Nicholas Wickström (2011). "Symbolization of time-series: An evaluation of sax, persist, and aca". In: *2011 4th international congress on image and signal processing*. Vol. 4. IEEE, pp. 2223–2228.

Senin, Pavel, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P Boedihardjo, et al. (2014). "Grammarviz 2.0: a tool for grammar-based pattern discovery in time series". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part III 14*. Springer, pp. 468–472.

Senin, Pavel, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, et al. (Feb. 2018). "GrammarViz 3.0: Interactive Discovery of Variable-Length Time Series

Patterns". In: *ACM Trans. Knowl. Discov. Data* 12.1, 10:1–10:28. ISSN: 1556-4681. DOI: 10.1145/3051126. URL: http://doi.acm.org/10.1145/3051126.

Senin, Pavel and Sergey Malinchik (2013). "SAX-VSM: Interpretable time series classification using sax and vector space model". In: *2013 IEEE 13th international conference on data mining*. IEEE, pp. 1175–1180.

Smagulova, Kamilya and Alex Pappachen James (2019). "A survey on LSTM memristive neural network architectures and applications". In: *The European Physical Journal Special Topics* 228.10, pp. 2313–2324.

Staudemeyer, Ralf C and Eric Rothstein Morris (2019). "Understanding LSTM–a tutorial into long short-term memory recurrent neural networks". In: *arXiv preprint arXiv:1909.09586*.

Sun, Youqiang et al. (2014). "An improvement of symbolic aggregate approximation distance measure for time series". In: *Neurocomputing* 138, pp. 189–198.

Syed Huma, Shah (2023). *Anomaly Detection in ECG Signals: Identifying Abnormal Heart Patterns Using Deep Learning*. Ed. by Shah Syed Huma. URL: https://www.analyticsvidhya.com/blog/2023/02/anomaly-detection-in-ecg-signals-identifying-abnormal-heart-patterns-using-deep-learning/.

Tyan, SG (2006). "Median filtering: Deterministic properties". In: *Two-Dimensional Digital Signal Prcessing II: Transforms and Median Filters*, pp. 197–217.

Woodbridge, Diane Myung-kyung et al. (2015). "Time series discord detection in medical data using a parallel relational database". In: *2015 IEEE international conference on bioinformatics and biomedicine (BIBM)*. IEEE, pp. 1420–1426.

Yu, Yong et al. (2019). "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7, pp. 1235–1270.

APPENDIX A

APPENDIX A

SAX CODE

Code was obtained and modified from (Senin, Lin, Wang, Oates, Gandhi, Arnold P. Boedihardjo, et al. 2018). Available at https://github.com/jMotif/jmotif-R

```
setwd("C:/Users/18133/OneDrive - The University of Texas-Rio
   Grande Valley (2)/Fall 2022/Moses")
#install_github('jMotif/jmotif-R')
#install.packages("readxl")
#install.packages('readMat')
library(devtools)
library(jmotif)
library(readxl)
library(R.matlab)
library(plyr)
library(cvTools)
library(nloptr)


# PICK one observation to see how the PAA works
C0 <-read_excel("C0.xlsx")
x1<-ts(C0) #make file time series
x1z<-znorm(x1) # z-nomralize the data points
plot(x1, type="l", col="blue", main="Control group ", xlab = "
   observation", ylab = "frequency")  #visual plot
```

```
#define parameters for PAA
a = 5 #alphabet size
p = 30 ## paa size
b = linspace(1,p,p) ##number of paa divisions
seq = linspace(0,3600,30)  ## sequence of breakpoints. my data
   has 3600 points
plot(x1z, type="l", col="blue", main="3600-points time series and
    it PAA transform into 30 points")
abline(v=c(0,linspace(0,3600,p)), lty=3, lwd=2, col="gray70")  #
   draw vertical lines to split the series into equal sizes
y_paa30 = paa(x1z, p)  ### does the piecewise aggregate
   approximation
##to show segments and their midpoints
for (i in b) {
  for (j in seq) {
   segments(seq[i],y_paa30[i],seq[i+1],y_paa30[i],lwd=1,col="red"
   )
    points(x = seq[i] + 120/2,y=y_paa30[i],col="red",pch=23,lwd
   =5)
  }
}
### showing the alphabet cuts on the plot
y1<-x1 #y1 assigned to original time series
lines(x1z,y1, type="l", lwd=5, col="magenta")
abline(h = alphabet_to_cuts(5)[2:5], lty=2, lwd=2, col="magenta")
text(0.4,-1,"a",cex=2,col="magenta")
text(0.4,-0.5 ,"b",cex=2,col="magenta")
```

```
text(0.4, 0,"c",cex=2,col="magenta")
text(0.4, 0.5,"d",cex=2,col="magenta")
text(0.4, 1,"e",cex=2,col="magenta")
CO_sts<-series_to_string(y_paa30, 5)   # series to 5 letter
   string
CO_stc<-series_to_chars(y_paa30,5)
#if any of the PAA divisions does not fall in the range of a
   given alphabet,
#the series to string and series to character won't contain it

trainData<-read.csv("TRAIN.csv",header = FALSE)
head(trainData,10)
trainData[1:5,1:5]

mytraindata<-as.matrix(trainData[,-c(1,2)])## Important: Please
   convert these training data to a matrix.
#In my first two columns I had saved ClassName (NSR,APB,AFL,IVR)
   and ClassLabels (NSR==1,APB==2,AFL==3,IVR==4) of training data
   .
dim(mytraindata) # 610 by 3600

testData<-read.csv("TEST.csv",header = FALSE)
head(testData,10)
testData[1:5,1:5]

mytestdata<-as.matrix(testData[,-c(1,2)]) # Important: Please
   convert these training data to a matrix.
```

```
##In my first two columns I had saved ClassName (NSR,APB,AFL,IVR)
    and ClassLabels (NSR==1,APB==2,AFL==3,IVR==4) of test data.
dim(mytestdata)# 266 by 3600

my_list <- list(trainData[,2],mytraindata,testData[,2],mytestdata
    )
names(my_list) <- c("labels_trainM", "data_trainM","labels_testM"
    ,"data_testM")
str(my_list)

attributes(my_list)
# set the discretization parameters
w <-62 #60 # the sliding window size
p <- 31 #6  # the PAA size
a <- 6 #6  # the SAX alphabet size

# convert the train classes to wordbags (the dataset has three
    labels: 1, 2, 3)
nsr <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
    labels_trainM"]] == 1,], w, p, a, "exact", 0.01) #8024 by 2
apb <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
    labels_trainM"]] == 2,], w, p, a, "exact", 0.01) #4822 by 2
afl <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
    labels_trainM"]] == 3,], w, p, a, "exact", 0.01)
afib <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
    labels_trainM"]] == 4,], w, p, a, "exact", 0.01)
svta <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
```

```
   labels_trainM"]] == 5,], w, p, a, "exact", 0.01)
pvc <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
   labels_trainM"]] == 6,], w, p, a, "exact", 0.01)
bigenimy <- manyseries_to_wordbag(my_list[["data_trainM"]][
   my_list[["labels_trainM"]] == 7,], w, p, a, "exact", 0.01)
trigenimy <- manyseries_to_wordbag(my_list[["data_trainM"]][
   my_list[["labels_trainM"]] == 8,], w, p, a, "exact", 0.01)
vt <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[["
   labels_trainM"]] == 9,], w, p, a, "exact", 0.01)
fusion <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list
   [["labels_trainM"]] == 10,], w, p, a, "exact", 0.01)
lbbbb <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[[
   "labels_trainM"]] == 11,], w, p, a, "exact", 0.01)
rbbbb <- manyseries_to_wordbag(my_list[["data_trainM"]][my_list[[
   "labels_trainM"]] == 12,], w, p, a, "exact", 0.01)
head(nsr)

# compute tf*idf weights for three bags
tfidf_M = bags_to_tfidf( list("nsr" = nsr, "apb" = apb, "afl" =
   afl,"afib" = afib, "svta" = svta, "pvc" = pvc,
                                "bigenimy" = bigenimy, "trigenimy"
   = trigenimy, "vt" = vt,
                                "fusion" = fusion, "lbbbb" = lbbbb,
   "rbbbb" = rbbbb) )
tail(tfidf_M) #this yields a data frame of five variables: the
   words which are "important" in TF*IDF terms
#(i.e. not presented at least in one of the bags) and their class
```

```
   -corresponding weights:
dim(tfidf_M)


library(dplyr)
head(arrange(tfidf_M, desc(nsr))) #arrange weights by disorder
head(arrange(tfidf_M, desc(apb)))


#SAX-VSM classification
# classify the test data
labels_predicted_M = rep(-1, length(my_list[["labels_testM"]]))
labels_test_M = my_list[["labels_testM"]]
data_test_M = my_list[["data_testM"]]
for (i in c(1:length(data_test_M[,1]))) {
  series_M = data_test_M[i,]
  bag_M = series_to_wordbag(series_M, w, p, a, "exact", 0.01)
  cosines_M = cosine_sim(list("bag"=bag_M, "tfidf" = tfidf_M))
  labels_predicted_M[i] = which(cosines_M$cosines == max(
   cosines_M$cosines))
}
labels_predicted_M


# compute the classification error
error_M = length(which((labels_test_M != labels_predicted_M))) /
   length(labels_test_M)
error_M


#findout which time series were misclassified
```

```
which((labels_test_M != labels_predicted_M))


###RUN THE CODE CHUNK BELOW TO OBTAIN THE OPTIMAL PARAMETERS FOR
   W,P & A
###SAX PARAMETER OPTIMIZATION FOR MY DATA###
x = c(3,30,5)  #c(w,p,a)
cverror_M <- function(x) {

  # the vector x suppose to contain reational values for the
  # discretization parameters
  w = round(x[1], digits = 0)
  p = round(x[2], digits = 0)
  a = round(x[3], digits = 0)

  # define the data for CV
  train_dataM <- my_list$data_trainM
  train_labelsM <- my_list$labels_trainM     #labels_trainM
  nfolds = 10
  # few local vars to simplify the process
  m<- length(train_labelsM)
  c <- length(unique(train_labelsM))
  folds <- cvFolds(m, K = nfolds, type = "random")

  # saving the error for each folds in this array
  errors_M <- list()

  # cross-valiadtion business
```

```
for (i in c(1:nfolds)) {
  # define data sets
  set_testM <- which(folds$which == i)
  set_trainM <- setdiff(1:m, set_testM)

  # compute the TF-IDF vectors
  bags_M <- alply(unique(train_labelsM),1,function(x){x})
  for (j in 1:c) {
    ll_M <- which(train_labelsM[set_trainM] == unique(
train_labelsM)[j])
    bags_M[[unique(train_labelsM)[j]]] <-
      manyseries_to_wordbag( (train_dataM[set_trainM , ])[ll_M
,], w, p, a, "exact", 0.01)
  }
  tfidf_M = bags_to_tfidf(bags_M)

  # compute the eror
  labels_predicted_M <- rep(-1, length(set_testM))
  labels_test_M <- train_labelsM[set_testM]
  data_testM <- train_dataM[set_testM,]

  for (j in c(1:length(labels_predicted_M))) {
    bag_M=NA
    if (length(labels_predicted_M)>1) {
      bag_M = series_to_wordbag(data_testM[j,], w, p, a, "exact
", 0.01)
    } else {
```

```
        bag_M = series_to_wordbag(data_testM, w, p, a, "exact",
    0.01)
      }
      cosines_M = cosine_sim(list("bag" = bag_M, "tfidf" =
    tfidf_M))
      if (!any(is.na(cosines_M$cosines))) {
        labels_predicted_M[j] = which(cosines_M$cosines == max(
    cosines_M$cosines))
      }
    }

    # the actual error value
    error_M = length(which((labels_test_M != labels_predicted_M))
    ) / length(labels_test_M)
    errors_M[i] <- error_M
  }

  # output the mean cross-validation error as the result
  err_M = mean(laply(errors_M,function(x){x}))
  print(paste(w,p,a, " -> ", err_M))
  err_M
}

# PERFORM THE PARAMETER OPTIMIZATION BY SPECIFYING THRESHOLD
#S <- directL(cverror_M, c(10,2,2), c(68,60,12),
# nl.info = TRUE, control = list(xtol_rel = 1e-8, maxeval = 10))
S <- directL(cverror_M, c(3,2,2), c(68,60,10),
```

```
            nl.info = TRUE, control = list(xtol_rel = 1e-8,
   maxeval = 10))

####use optimal sax parameters for classification
w = round(S$par[1], digits = 0)
p = round(S$par[2], digits = 0)
a = round(S$par[3], digits = 0)
para<-c(w,p,a)   #list of optimal parameters
para
#compute the TF-IDF vectors
bags_M <- alply(unique(train_labelsM),1,function(x){x})
for (j in 1:length(unique(train_labelsM))) {
  ll_M <- which(train_labelsM == unique(train_labelsM)[j])
  bags_M[[unique(train_labelsM)[j]]] <-
    manyseries_to_wordbag( train_dataM[ll_M,], w, p, a, "exact",
   0.01)
}
tfidf_M = bags_to_tfidf(bags_M)

# classify the test data
labels_predicted_M = rep(-1, length(my_list[["labels_testM"]]))
labels_test_M = my_list[["labels_testM"]]
data_test_M = my_list[["data_testM"]]
for (i in c(1:length(data_test_M[,1]))) {
  #print(paste(i))
  series_M = data_test_M[i,]
  bag_M = series_to_wordbag(series_M, w, p, a, "exact", 0.01)
```

```
  cosines_M = cosine_sim(list("bag"=bag_M, "tfidf" = tfidf_M))
  if (!any(is.na(cosines_M$cosines))) {
    labels_predicted_M[i] = which(cosines_M$cosines == max(
   cosines_M$cosines))
  }
}

# compute the classification error
error_M = length(which((labels_test_M != labels_predicted_M))) /
   length(labels_test_M)
error_M

# findout which time series were misclassified
which((labels_test_M != labels_predicted_M))
par(mfrow=c(3,1))
plot(my_list[["data_testM"]][28,], type="l")
plot(my_list[["data_testM"]][118,], type="l")
plot(my_list[["data_testM"]][253,], type="l")

###CODE FOR DISCORD DISCOVERY
################ HOT-SAX FOR DISCORD DISCOVERY ################
?find_discords_brute_force
#arguments(ts, w_size, discords_num) discords_num is the number
   of discords to report
#install.packages("lineprof")
#library("profvis")
find_discords_brute_force(mytraindata[87,],62,2) #87 is a row in
```

```
   my data corresponding to one observation


?find_discords_hotsax
# arguments (ts, w_size, paa_size, a_size, n_threshold,
   discords_num)
find_discords_hotsax(mytraindata[2,], 62, 31, 6, 0.01,1)


### FINDING MULTIPLE DISCORDS ####
discords = find_discords_hotsax(mytraindata[289,], 62, 31, 6,
   0.01, 3)  #
discords


## SELECTING THE BEST DISCORD AND PLOT ##### the one with largest
    nn_distance
discords = find_discords_hotsax(mytraindata[289,], 62, 31, 6,
   0.01, 3)
plot(mytraindata[289,], type = "l", col = "cornflowerblue", main
   = "ECG NSR ")
lines(x=c(discords[1,2]:(discords[1,2]+100)),
      y=mytraindata[289,][discords[1,2]:(discords[1,2]+100)], col
   ="red")
discords = find_discords_hotsax(mytraindata[289,], 62, 31, 6,
   0.01, 3)
plot(mytraindata[289,], type = "l", col = "cornflowerblue", main
   = "ECG NSR")
lines(x=c(discords[2,2]:(discords[2,2]+100)),
      y=mytraindata[289,][discords[2,2]:(discords[2,2]+100)], col
```

```
   ="red")
discords = find_discords_hotsax(mytraindata[289,], 62, 31, 6,
   0.01, 3)
plot(mytraindata[289,], type = "l", col = "cornflowerblue", main
   = "ECG NSR ")
lines(x=c(discords[3,2]:(discords[3,2]+50)),
      y=mytraindata[289,][discords[3,2]:(discords[3,2]+100)], col
   ="red")
###################### mutli discord with plot#####
discords = find_discords_hotsax(mytraindata[544,], 62, 31, 6,
   0.01, 3)
plot(mytraindata[544,], type = "l", col = "cornflowerblue", main
   = "ECG RBBBB - PATIENT 1 ")
lines(x=c(discords[1,2]:(discords[1,2]+100)),
      y= mytraindata[544,][discords[1,2]:(discords[1,2]+100)],
   col="red")
lines(x=c(discords[2,2]:(discords[2,2]+100)),
      y= mytraindata[544,][discords[2,2]:(discords[2,2]+100)],
   col="black")
lines(x=c(discords[3,2]:(discords[3,2]+100)),
      y= mytraindata[544,][discords[3,2]:(discords[3,2]+100)],
   col="lawngreen")
discords = find_discords_hotsax(mytraindata[583,], 62, 31, 6,
   0.01, 3)
plot(mytraindata[583,], type = "l", col = "cornflowerblue", main
   = "ECG RBBBB - PATIENT 2")
lines(x=c(discords[1,2]:(discords[1,2]+100)),
```

```
      y= mytraindata[583,][discords[1,2]:(discords[1,2]+100)],
   col="red")
lines(x=c(discords[2,2]:(discords[2,2]+100)),
      y= mytraindata[583,][discords[2,2]:(discords[2,2]+100)],
   col="black")
lines(x=c(discords[3,2]:(discords[3,2]+100)),
      y= mytraindata[583,][discords[3,2]:(discords[3,2]+100)],
   col="lawngreen")
discords = find_discords_hotsax(mytraindata[601,], 62, 31, 6,
   0.01, 3)
plot(mytraindata[601,], type = "l", col = "cornflowerblue", main
   = "ECG VT - PATIENT 3 ")
lines(x=c(discords[1,2]:(discords[1,2]+100)),
      y= mytraindata[601,][discords[1,2]:(discords[1,2]+100)],
   col="red")
lines(x=c(discords[2,2]:(discords[2,2]+100)),
      y= mytraindata[601,][discords[2,2]:(discords[2,2]+100)],
   col="black")
lines(x=c(discords[3,2]:(discords[3,2]+100)),
      y= mytraindata[601,][discords[3,2]:(discords[3,2]+100)],
   col="lawngreen")
```

APPENDIX B

APPENDIX B

LSTM CODE

Code was obtained from (Syed Huma 2023) available at https://www.analyticsvidhya.com/blog/2023/02/anomaly-detection-in-ecg-signals-identifying-abnormal-heart-patterns-using-deep-learning/

```
#Import all the libraries
import pandas as pd
import numpy as np
import random
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
import plotly.graph_objs as go
import plotly.express as px
from scipy.signal import medfilt, butter, filtfilt
import pywt
from sklearn.model_selection import train_test_split
import scipy.signal
from keras.models import Sequential
from keras.layers import LSTM, Dense, Reshape
from sklearn.metrics import confusion_matrix, accuracy_score,
    classification_report
from sklearn.metrics import roc_auc_score
from IPython import get_ipython
```

```
#load the dataset
df_train = pd.read_csv('/content/drive/MyDrive/UTRGV_thesis_data/
    mytrain.csv', header = None)
df_test = pd.read_csv('/content/drive/MyDrive/UTRGV_thesis_data/
    mytest.csv', header = None)
df_train[0].unique() #display the unique features
###specify just two classes of disorder
df_train = df_train[(df_train.loc[:,0] ==1)|(df_train.loc
    [:,0]==4)]
#df_train = df_train[df_train.loc[:,0].isin([1,2])]
df_test = df_test[(df_test.loc[:,0] ==1)|(df_test.loc[:,0]==4)]

df_train[0].value_counts() ## display the # of examples for each
    label
df_train.dtypes #display the datatypes for each column
df_train.isna().sum() #check for missing values in each column
df_train.isna().sum().sum() #check missing values for entire
    dataset
#plot graphs of normal and abnormal ECG to visualise the trends
abnormal = df_train[df_train.loc[:,0] ==9][:3]
normal = df_train[df_train.loc[:,0] ==1][:3]
# Create the figure
fig = go.Figure()
#create a list to display only a single legend
leg  = [False] * abnormal.shape[0]
leg[0] = True
```

```
# Plot training and validation error
for i in range(abnormal.shape[0]):
    fig.add_trace(go.Scatter( x=np.arange(abnormal.shape[1]),y=
   abnormal.iloc[i,:],name='Abnormal ECG (RBBBB)', mode='lines',
    marker_color='rgba(255, 0, 0, 0.9)', showlegend= leg[i]))
for j in range(normal.shape[0]):
    fig.add_trace(go.Scatter( x=np.arange(normal.shape[1]),y=
   normal.iloc[j,:],name='Normal ECG',  mode='lines',
   marker_color='rgba(0, 255, 0, 1)', showlegend= leg[j]))
fig.update_layout(xaxis_title="time", yaxis_title="Signal", title
   = {'text': 'Difference between different ECG', 'xanchor': '
   center', 'yanchor': 'top', 'x':0.5} , bargap=0,)
fig.update_traces(opacity=0.5)
fig.show()
#data preprocessing
#labels and features split
ecg_data_train = df_train.iloc[:,1:] #from col 1 to last column
ecg_data_test = df_test.iloc[:,1:]
labels_train = df_train.iloc[:,0] #1st column only
labels_test = df_test.iloc[:,0]

###change labels to zeros
labels_train[labels_train ==4] = 0 ##change the value befor] to
   particular disorder
labels_test[labels_test ==4] = 0

#normalize the dataset btn -1 and 1
```

```
scaler = MinMaxScaler(feature_range = (-1,1))
ecg_data_train = scaler.fit_transform(ecg_data_train)
ecg_data_test = scaler.fit_transform(ecg_data_test)

#filter the data
#filtering the raw signals
# Median filtering for train data
ecg_medfilt = medfilt(ecg_data_train, kernel_size=3)
# Low-pass filtering
lowcut = 0.05
highcut = 20.0
nyquist = 0.5 * 360.0
low = lowcut / nyquist
high = highcut / nyquist
b, a = butter(4, [low, high], btype='band')
ecg_lowpass = filtfilt(b, a, ecg_data_train)
# Wavelet filtering
coeffs = pywt.wavedec(ecg_data_train, 'db4', level=1)
threshold = np.std(coeffs[-1]) * np.sqrt(2*np.log(len(
    ecg_data_train)))
coeffs[1:] = (pywt.threshold(i, value=threshold, mode='soft') for
     i in coeffs[1:])
ecg_wavelet = pywt.waverec(coeffs, 'db4')

# Median filtering for test data
ecg_medfilt_ = medfilt(ecg_data_test, kernel_size=3)
# Low-pass filtering
```

```
lowcut = 0.05
highcut = 20.0
nyquist = 0.5 * 360.0
low = lowcut / nyquist
high = highcut / nyquist
b, a = butter(4, [low, high], btype='band')
ecg_lowpass_ = filtfilt(b, a, ecg_data_test)
# Wavelet filtering
coeffs_ = pywt.wavedec(ecg_data_test, 'db4', level=1)
threshold_ = np.std(coeffs_[-1]) * np.sqrt(2*np.log(len(
    ecg_data_test)))
coeffs_[1:] = (pywt.threshold(i, value=threshold, mode='soft')
    for i in coeffs_[1:])
ecg_wavelet_ = pywt.waverec(coeffs_, 'db4')

# Plot original ECG signal
fig = go.Figure()
fig.add_trace(go.Scatter(x=np.arange(ecg_data_train.shape[0]), y=
    ecg_data_train[30], mode='lines', name='Original ECG signal'))
# Plot filtered ECG signals
fig.add_trace(go.Scatter(x=np.arange(ecg_medfilt.shape[0]), y=
    ecg_medfilt[30], mode='lines', name='Median filtered ECG
    signal'))
fig.add_trace(go.Scatter(x=np.arange(ecg_lowpass.shape[0]), y=
    ecg_lowpass[30], mode='lines', name='Low-pass filtered ECG
    signal'))
fig.add_trace(go.Scatter(x=np.arange(ecg_wavelet.shape[0]), y=
```

```
    ecg_wavelet[30], mode='lines', name='Wavelet filtered ECG
    signal'))
fig.show()
#choosing the best filtering technique
#pad the signal with zeroes
def pad_data(original_data,filtered_data):
  # Calculate the difference in length between the original data
   and filtered data
  diff = original_data.shape[1] - filtered_data.shape[1]
    # pad the shorter array with zeroes
  if diff > 0:
        # Create an array of zeros with the same shape as the
   original data
      padding = np.zeros((filtered_data.shape[0], original_data.
   shape[1]))
      # Concatenate the filtered data with the padding
      padded_data = np.concatenate((filtered_data, padding))
  elif diff < 0:
      padded_data = filtered_data[:,:-abs(diff)]
  elif diff == 0:
      padded_data = filtered_data
  return padded_data

def mse(original_data, filtered_data):
    filter_data = pad_data(original_data,filtered_data)
    return np.mean((original_data - filter_data) ** 2)
# Calculate MSE
```

```
mse_value_m = mse(ecg_data_train, ecg_medfilt)
mse_value_l = mse(ecg_data_train, ecg_lowpass)
mse_value_w = mse(ecg_data_train, ecg_wavelet)
print("MSE value of Median Filtering:", mse_value_m)
print("MSE value of Low-pass Filtering:", mse_value_l)
print("MSE value of Wavelet Filtering:", mse_value_w)

#Step 5: Splitting Data into Train & Test Set
#The dataset is divided into 80% for training and 20% for testing
    and validation purposes.

# Splitting the data into train and test sets
#X_train, X_test, y_train, y_test = train_test_split(ecg_wavelet,
    labels, test_size=0.2, random_state=42)
##data splitting 610 x 3600
#488 x 19
X_train = ecg_wavelet
X_test = ecg_wavelet_
y_train = labels_train
y_test = labels_test

#feature extraction for training
# Initializing an empty list to store the features
features = []
# Extracting features for each sample
for i in range(X_train.shape[0]):
    #Finding the R-peaks
```

```
 r_peaks = scipy.signal.find_peaks(X_train[i])[0]
 #Initialize lists to hold R-peak and T-peak amplitudes
 r_amplitudes = []
 t_amplitudes = []
 # Iterate through R-peak locations to find corresponding T-
peak amplitudes
 for r_peak in r_peaks:
     # Find the index of the T-peak (minimum value) in the
interval from R-peak to R-peak + 200 samples
     t_peak = np.argmin(X_train[i][r_peak:r_peak+200]) +
r_peak
     #Append the R-peak amplitude and T-peak amplitude to the
lists
     r_amplitudes.append(X_train[i][r_peak])
     t_amplitudes.append(X_train[i][t_peak])
 # extracting singular value metrics from the r_amplitudes
 std_r_amp = np.std(r_amplitudes)
 mean_r_amp = np.mean(r_amplitudes)
 median_r_amp = np.median(r_amplitudes)
 sum_r_amp = np.sum(r_amplitudes)
 # extracting singular value metrics from the t_amplitudes
 std_t_amp = np.std(t_amplitudes)
 mean_t_amp = np.mean(t_amplitudes)
 median_t_amp = np.median(t_amplitudes)
 sum_t_amp = np.sum(t_amplitudes)
 # Find the time between consecutive R-peaks
 rr_intervals = np.diff(r_peaks)
```

```
    # Calculate the time duration of the data collection
    time_duration = (len(X_train[i]) - 1) / 1000 # assuming data
is in ms
    # Calculate the sampling rate
    sampling_rate = len(X_train[i]) / time_duration
    # Calculate heart rate
    duration = len(X_train[i]) / sampling_rate
    heart_rate = (len(r_peaks) / duration) * 60
    # QRS duration
    qrs_duration = []
    for j in range(len(r_peaks)):
      qrs_duration.append(r_peaks[j]-r_peaks[j-1])
    # extracting singular value metrics from the qrs_durations
    std_qrs = np.std(qrs_duration)
    mean_qrs = np.mean(qrs_duration)
    median_qrs = np.median(qrs_duration)
    sum_qrs = np.sum(qrs_duration)
    # Extracting the singular value metrics from the RR-interval
    std_rr = np.std(rr_intervals)
    mean_rr = np.mean(rr_intervals)
    median_rr = np.median(rr_intervals)
    sum_rr = np.sum(rr_intervals)
    # Extracting the overall standard deviation
    std = np.std(X_train[i])
    # Extracting the overall mean
    mean = np.mean(X_train[i])
    # Appending the features to the list
```

```
    features.append([mean, std, std_qrs, mean_qrs,median_qrs,
   sum_qrs, std_r_amp, mean_r_amp, median_r_amp, sum_r_amp,
   std_t_amp, mean_t_amp, median_t_amp, sum_t_amp, sum_rr, std_rr
   , mean_rr,median_rr, heart_rate])
# Converting the list to a numpy array
features = np.array(features)

#feature extraction for test set vb
# Initializing an empty list to store the features
X_test_fe = []
# Extracting features for each sample
for i in range(X_test.shape[0]):
    # Finding the R-peaks
    r_peaks = scipy.signal.find_peaks(X_test[i])[0]
    # Initialize lists to hold R-peak and T-peak amplitudes
    r_amplitudes = []
    t_amplitudes = []
    # Iterate through R-peak locations to find corresponding T-
   peak amplitudes
    for r_peak in r_peaks:
        # Find the index of the T-peak (minimum value) in the
   interval from R-peak to R-peak + 200 samples
        t_peak = np.argmin(X_test[i][r_peak:r_peak+200]) + r_peak
        # Append the R-peak amplitude and T-peak amplitude to the
    lists
        r_amplitudes.append(X_test[i][r_peak])
        t_amplitudes.append(X_test[i][t_peak])
```

```
    #extracting singular value metrics from the r_amplitudes
    std_r_amp = np.std(r_amplitudes)
    mean_r_amp = np.mean(r_amplitudes)
    median_r_amp = np.median(r_amplitudes)
    sum_r_amp = np.sum(r_amplitudes)
    #extracting singular value metrics from the t_amplitudes
    std_t_amp = np.std(t_amplitudes)
    mean_t_amp = np.mean(t_amplitudes)
    median_t_amp = np.median(t_amplitudes)
    sum_t_amp = np.sum(t_amplitudes)
    # Find the time between consecutive R-peaks
    rr_intervals = np.diff(r_peaks)
    # Calculate the time duration of the data collection
    time_duration = (len(X_test[i]) - 1) / 1000 # assuming data
is in ms
    # Calculate the sampling rate
    sampling_rate = len(X_test[i]) / time_duration
    # Calculate heart rate
    duration = len(X_test[i]) / sampling_rate
    heart_rate = (len(r_peaks) / duration) * 60
    # QRS duration
    qrs_duration = []
    for j in range(len(r_peaks)):
        qrs_duration.append(r_peaks[j]-r_peaks[j-1])
    #extracting singular value metrics from the qrs_duartions
    std_qrs = np.std(qrs_duration)
    mean_qrs = np.mean(qrs_duration)
```

```
    median_qrs = np.median(qrs_duration)
    sum_qrs = np.sum(qrs_duration)
    # Extracting the standard deviation of the RR-interval
    std_rr = np.std(rr_intervals)
    mean_rr = np.mean(rr_intervals)
    median_rr = np.median(rr_intervals)
    sum_rr = np.sum(rr_intervals)
      # Extracting the standard deviation of the RR-interval
    std = np.std(X_test[i])
    # Extracting the mean of the RR-interval
    mean = np.mean(X_test[i])
    # Appending the features to the list
    X_test_fe.append([mean, std,  std_qrs, mean_qrs,median_qrs,
   sum_qrs, std_r_amp, mean_r_amp, median_r_amp, sum_r_amp,
   std_t_amp, mean_t_amp, median_t_amp, sum_t_amp, sum_rr, std_rr
   , mean_rr,median_rr,heart_rate])
# Converting the list to a numpy array
X_test_fe = np.array(X_test_fe)

##model building and training
# Define the number of features in the train dataframe
num_features = features.shape[1]
# Reshape the features data to be in the right shape for LSTM
   input
features = np.asarray(features).astype('float32')
features = features.reshape(features.shape[0], features.shape[1],
    1)
```

```
X_test_fe = X_test_fe.reshape(X_test_fe.shape[0], X_test_fe.shape
    [1], 1)
# Define the model architecture
model = Sequential()
model.add(LSTM(64, input_shape=(features.shape[1], 1)))
model.add(Dense(1, activation='sigmoid'))
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
    metrics=['accuracy'])
# Train the model
history = model.fit(features, y_train, validation_data=(X_test_fe
    , y_test), epochs=50, batch_size=32)
# Make predictions on the validation set
y_pred = model.predict(X_test_fe)
# Convert the predicted values to binary labels
#y_pred = [1 if p>0.5 else 0 for p in y_pred]
#y_pred = np.argmax(y_pred, axis = 1)
#X_test_fe = np.asarray(X_test_fe).astype('float32')

# Convert the predicted values to binary labels
y_pred_new = [1 if p>0.91 else 0 for p in y_pred]
#y_pred = np.argmax(y_pred, axis = 1)
X_test_fe = np.asarray(X_test_fe).astype('float32')

##model evaluation
#calculating the metrics
# calculate the accuracy
```

```
acc = accuracy_score(y_test, y_pred_new)
#calculate the AUC score
#roc_auc_score(y_score=np_pred, y_true=np_label, multi_class="ovr
    ",average= weighted)
auc = round(roc_auc_score(y_test, y_pred_new),2)
#classification report provides all metrics e.g. precision,
    recall, etc.
all_met = classification_report(y_test, y_pred_new)

##displaying the metrics
# Print the accuracy
print("Accuracy: ", acc*100, "%")
print(" n")
print("AUC:", auc)
print(" n")
print("Classification Report: n", all_met)
print(" n")

###calculating and displaying confusion matrix
# Calculate the confusion matrix
conf_mat = confusion_matrix(y_test, y_pred_new)
conf_mat_df = pd.DataFrame(conf_mat, columns=['Predicted Negative
    ', 'Predicted Positive'], index=['Actual Negative', 'Actual
    Positive'])
fig = px.imshow(conf_mat_df, text_auto= True,
    color_continuous_scale='Blues')
fig.update_xaxes(side='top', title_text='Predicted')
```

```
fig.update_yaxes(title_text='Actual')
fig.show()


##plotting the training and validation error
# Plot training and validation error
fig = go.Figure()
fig.add_trace(go.Scatter( y=history.history['loss'], mode='lines'
    , name='Training'))
fig.add_trace(go.Scatter( y=history.history['val_loss'], mode='
    lines', name='Validation'))
fig.update_layout(xaxis_title="Epoch", yaxis_title="Error", title
    = {'text': 'Model Error', 'xanchor': 'center', 'yanchor': 'top
    ', 'x':0.5} , bargap=0)
fig.show()
```

APPENDIX C

APPENDIX C

EXTRA FIGURES

Table C.1: Brute Force Discord Discovery

Class	No of discords	nn_distance	Position
APB	1	197.91	686
AFL	1	110.21	1045
AFIB	1	120.43	1187
SVTA	1	226.39	3399
PVC	1	991.60	3361
BIGENIMY	1	536.49	2414
TRIGENIMY	1	285.38	3535
VT	1	579.46	2700
FUSION	1	677.33	3325
LBBBB	1	153.00	475
RBBBB	1	169.55	3415

Table C.2: HOT-SAX Discord Discovery

Class	No of discords	nn_distance	Position
APB	1	152.23	2618
AFL	1	90.87	401
AFIB	1	161.31	2293
SVTA	1	226.39	3399
PVC	1	1336.18	1941
BIGENIMY	1	604.85	1882
TRIGENIMY	1	168.39	2547
VT	1	579.46	2700
FUSION	1	677.33	3325
LBBBB	1	1079.87	2428
RBBBB	1	152.26	0

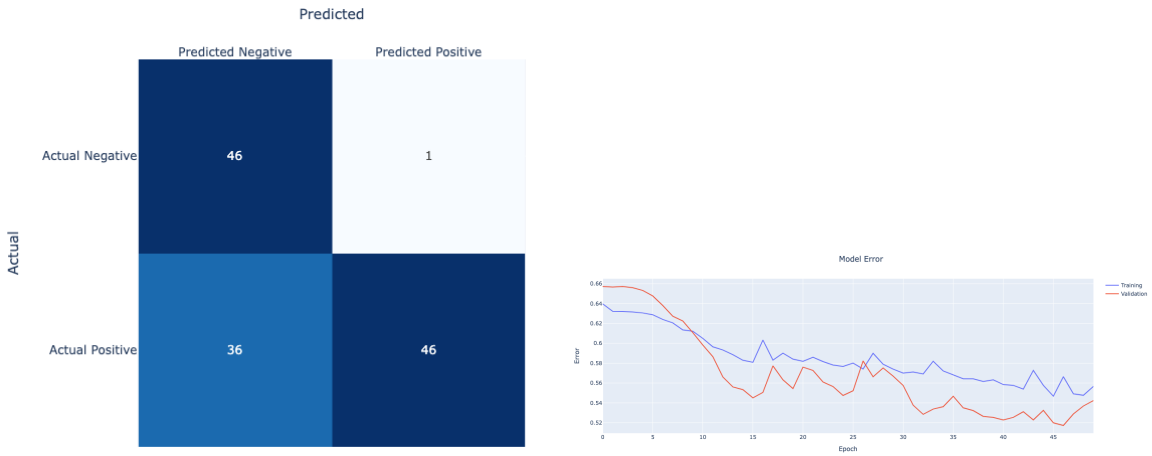

Figure C.1: Confusion matrix and error plot for NSR and PVC

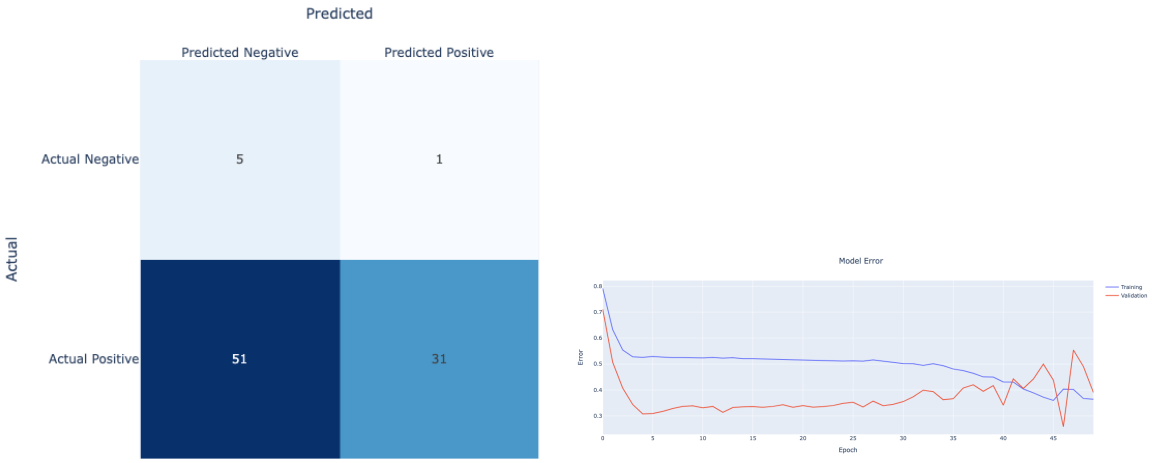

Figure C.2: Confusion matrix and error plot for NSR and BIGENIMY

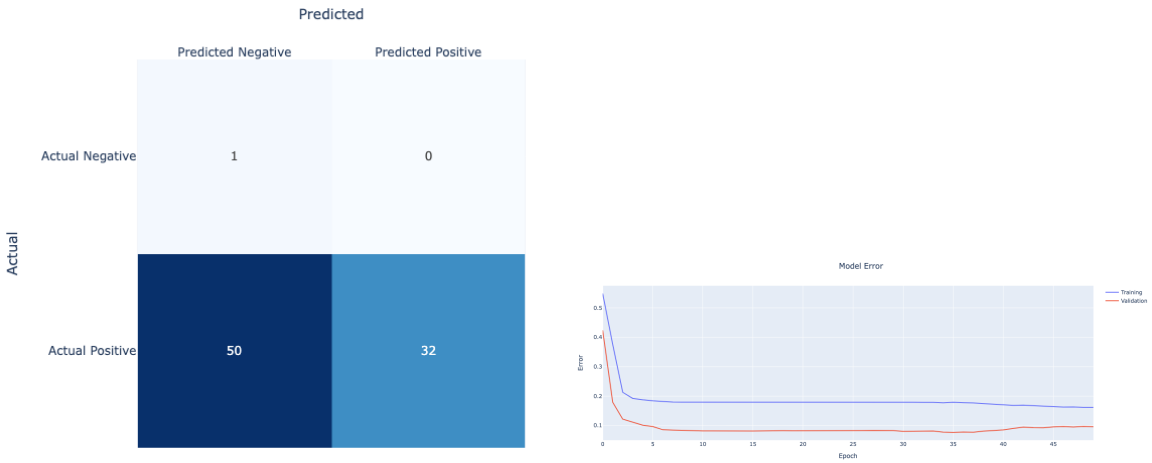

Figure C.3: Confusion matrix and error plot for NSR and VT

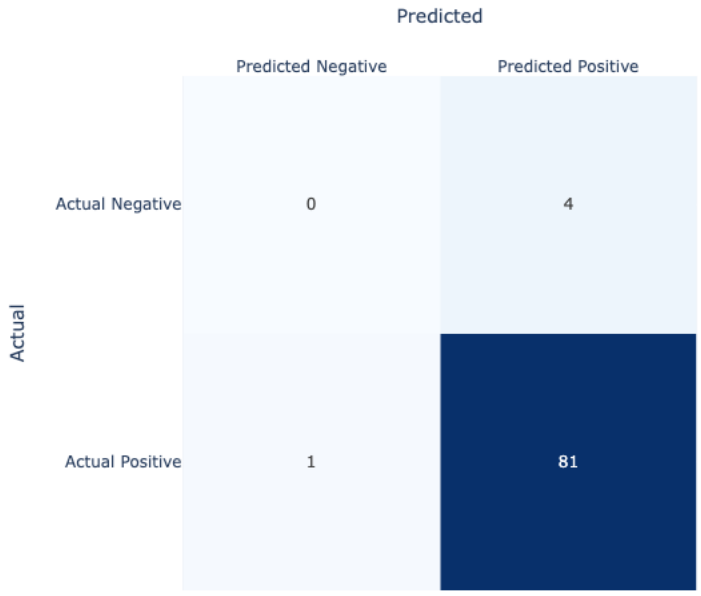

Figure C.4: Confusion matrix for NSR and TRIGENIMY

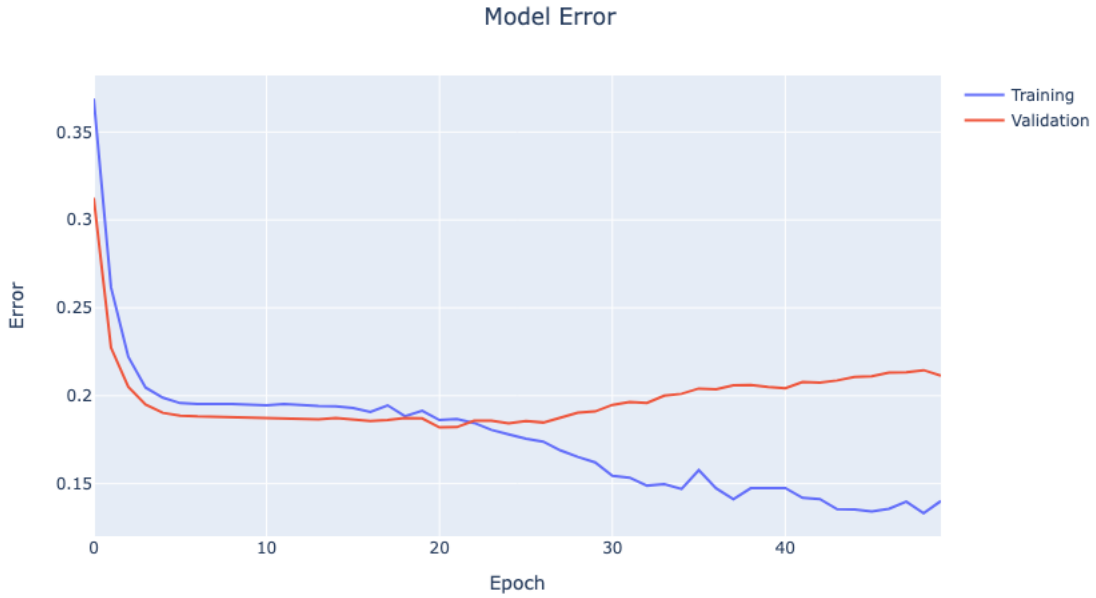

Figure C.5: Error plot for NSR and TRIGENIMY

BIOGRAPHICAL SKETCH

Moses Owusu completed his Bachelor of Science in Statistics at the Kwame Nkrumah University of Science and Technology (KNUST), Ghana in 2018 and a Master's Degree in Applied Statistics and Data Science in 2023 from the University of Texas Rio Grande Valley. After his BSc, he served as a teaching and research assistant at the department of Mathematics, KNUST where taught several undergraduate courses and assisted students with their research in areas such as Time Series and Survival Analysis. His research interests include Finance, Disease modelling, Operations Research, and Data Science. In summer 2022, he worked as a computational science graduate intern in the Theoretical Biology and Biophysics Group (T-6) at the Los Alamos National Laboratory (LANL), New Mexico. There, he worked on projects including mosquito-borne disease modelling specifically, West Nile Virus (WNV) and dengue fever. Contributing through spatial sampling of ebird data as well as preparing geological data from Brazil for analysis that involved multi-host vectors, human demographics, and climate data. He is currently working under the supervision of Dr. Hansapani P. Rodrigo to use Electrocardiogram (ECG) readings to identify and classify different heart disorders with the use of Symbolic Aggregate Approximation (SAX). After his MS degree, he plans to further with a PhD in Biostatistics and contribute through collaborative and pioneering research in Health, impart knowledge in statistics as a multi-lingual professor, while travelling the world.

To contact Moses, feel free to email him at: mkowusu25@gmail.com.