Summer 2020

Artificial Neural Network Models for Pattern Discovery from ECG Time Series

Mehakpreet Kaur

ARTIFICIAL NEURAL NETWORK MODELS FOR PATTERN DISCOVERY FROM ECG TIME SERIES

by

MEHAKPREET KAUR

(Under the Direction of Ionut Emil Iacob)

ABSTRACT

Artificial Neural Network (ANN) models have recently become de facto models for deep learning with a wide range of applications spanning from scientific fields such as computer vision, physics, biology, medicine to social life (suggesting preferred movies, shopping lists, etc.). Due to advancements in computer technology and the increased practice of Artificial Intelligence (AI) in medicine and biological research, ANNs have been extensively applied not only to provide quick information about diseases, but also to make diagnostics accurate and cost-effective. We propose an ANN-based model to analyze a patient's electrocardiogram (ECG) data and produce accurate diagnostics regarding possible heart diseases (arrhythmia, myocardial infarct, etc.). Our model is mainly characterized by its simplicity, as it does not require significant computational power to produce the results. We create and test our model using the MIT-BIH and PTB diagnostics datasets, which are real ECG time series datasets from thousands of patients.

INDEX WORDS: Artificial neural networks, Artificial intelligence, Machine learning, Electrocardiogram (ECG)

2009 Mathematics Subject Classification: 68T10, 62H30

ARTIFICIAL NEURAL NETWORK MODELS FOR PATTERN DISCOVERY FROM ECG TIME SERIES

by

MEHAKPREET KAUR

B.Sc., Guru Nanak Dev University, India, 2015

M.Sc., Khalsa College (Guru Nanak Dev University), India, 2017

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial

Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ARTIFICIAL NEURAL NETWORK MODELS FOR PATTERN DISCOVERY FROM ECG TIME SERIES

by

MEHAKPREET KAUR

Major Professor:	Ionut Emil Iacob
Committee:	Goran Lesaja
	Marcel Ilie

Electronic Version Approved:
July 2020

DEDICATION

This thesis is dedicated to my Guru (God) for his guidance, protection and grace upon me throughout my graduate program. To my parents, my father Dr. Maninder Singh, for his mentorship, guidance, unending moral, and emotional support, my mother Mrs. Rajminder Kaur, for her love, care and prayers. To my sibling Livjyot Singh, for being my strength and my uncle Dr. Kamaldeep Sidhu, without whose support, I would not have made it this far and my friends for always having my back.

ACKNOWLEDGMENTS

I owe my deepest gratitude to my thesis advisor Dr. Ionut Emil Iacob for his vision, enormous support, and guidance during the completion of my thesis. I am thankful to my committee members Dr. Goran Lesaja and Dr. Marcel Ilie for taking out time to read my thesis and appreciating it. I would like to appreciate and be thankful of all my professors at GSU. I am grateful to my family, relatives and friends for their support and encouragement during my graduate program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

$\mathbb{R}$	Real Numbers
$\mathcal{N}$	Normal ECG dataset
$\mathcal{A}$	Abnormal ECG dataset
$\mathcal{D}$	Binary or Multiclass dataset
$\hat{\beta}()$	Binary classification model
$\hat{\mu}()$	Multiclass classification model
$accL$	Accuracy Lower
$accU$	Accuracy Upper

CHAPTER 1

INTRODUCTION

1.1 PRELIMINARIES

This study focuses on the challenges involved in predicting results from the ECG time series data. This research aims to propose a mathematical framework for the analysis of ECG time series data. The structure is based on deep learning concepts, which constitute hidden layers in an Artificial Neural Network (ANN). ANN possesses a considerable learning capacity as the human brain's behavior inspires it, and it predicts unknown parameters through acquired information.

1.1.1 ELECTROCARDIOGRAM (ECG)

The heart is a central part of the human body and is critical for its wellbeing. Not only is it a blood-pumping organ, but also, it involves expressing our emotions and desires. Also, the heart has a very close relationship to our health. There has been enough evidence to suggest that there is a close association between the cardiovascular system and health. Heartbeats have a more intense story to tell than we know. Over the past few years, there have been many technological advancements in studies of pulsations to detect the heart's functioning, with the most advanced techniques used for this purpose being Electrocardiogram (ECG) devices, popularly known as EKG devices. ECG is used as a tool by researchers, investigators, and medical practitioners to monitor the cardiac health of patients by monitoring the electrical and muscular activities of the heart.

Cardiologists use ECG signals (Figure 1.1) to measure the rate, rhythm, and regularity of heartbeats. ECG signals also estimate the presence of any damage to the heart. While it is a relatively simple test to perform, the interpretation of ECG tracing requires significant training. ECG Signals, which are in the form of time series data, are used to

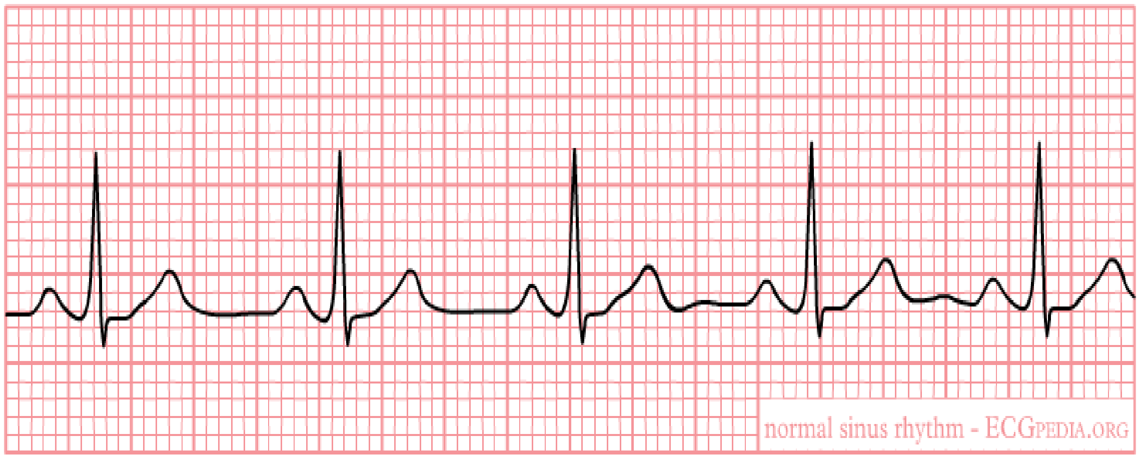


Figure 1.1: An ECG signal [21]

detect the anomalies in heart waves. The interpretation of data is complex to analyze, time-consuming, prone to errors. It requires a lot of human effort in the detection and categorization of different waveforms in the signals. Considering the global rise in cardiovascular diseases (CVDs), the diagnosis should be accurate and of low-cost, since CVDs are the highest cause of mortality globally. Millions of people around the globe are vulnerable to irregular heartbeats, which is a severe health concern. It can prove to be noxious if not taken care of within time. For addressing the issue of diagnostic accuracy of cardiovascular diseases, many studies include using machine learning techniques to solve the problems that arise from the manual analysis of ECG signals. In the medical field, more and more applications are created to assist medical practitioners, not only to provide quick, factual information about diseases but also to make diagnostics that would help doctors avoid possible fatal mistakes. With the advancement of technology, it is possible to detect patients' medical conditions in a timely and cost-effective manner. The objective of this research is to identify if a person is prone to develop a possible cardiovascular disease in the future by analyzing the time series data in the form of ECG signals of different patients.

Since the data from the research involves time series, it makes it even more complex to interpret the ECG signals. All recent studies involving the analysis of ECG data used complex models to train their network from a large amount of labeled data, and transferring the knowledge learned to the task using ECG signals. Our study aims to produce a less complicated, though accurate, model for the classification of different arrhythmias following the AAMI EC57 [8] standard. This study runs parallel to the paper [8], where the same heartbeat classification is performed, but using a different architecture of Neural Networks, namely Convolutional Neural Networks.

In the subsequent section, we introduce the idea of Artificial Neural Networks, which is based on our research and the foundation of our model.

1.1.2 Neural Network

Neural Networks: the name is self-evident; it is a network of neurons. Neurons are the fundamental and computational units of the human brain. A neuron is a node that has many inputs and a single output. Therefore, a network is comprised of many neurons, which are interconnected to one another. In technological terms, a neural network is a device that accepts the data from external sources to process it, and then it provides the desired response. The learning stage of a neural network consists of incoming and outgoing signals. The knowledge stage comprises the accumulated learned knowledge received from the input signals, which is then processed to generate valuable output signals. The discovery of neural networks was an attempt to exploit the human brain. The human brain is made up of about 100 billion nerve cells (neurons) that communicate with each other to perform tasks that have empirical results. The original idea was to train a machine to act like a human being by using known information to generate knowledge. A generic Artificial Neural Network (ANN) (Figure1.2) is a combination of simulated neurons, which are connected through links. These links have weights representing their relative importance, depending upon the strength of one node's influence over the other at a connection. This point of contact is where a propagation function transports the values of these neurons across a certain threshold having neural network layers. These layers perform transformations on the input data to generate an output. Usually, these input-output signals transverse through these layers multiple times before they can produce desired outcomes.

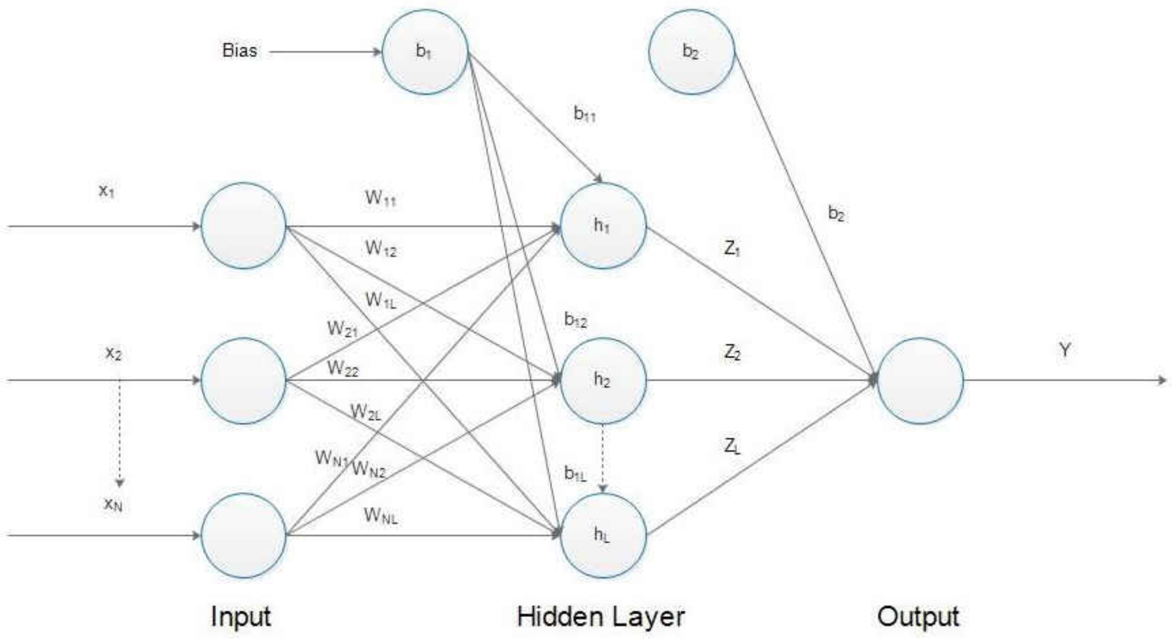


Figure 1.2: Artificial Neural Network with N inputs, one hidden layer (with N neurons, one bias), and an output (and one bias)

The goal of neural networks was primarily to exhibit properties of the brain. However, these days one can find popular applications of neural networks in email spam, fraud detection, image recognition, identification of numbers in a handwritten zip code, self-driven cars, speaker impersonation, machine translation, social network filtering and many more.

1.1.3 Model Representation of Neural Networks

The working of the neural network can be explained as follows:

1. **Obtaining Data**: To collect training data, that may be in the form of images for pattern recognition or sentences for generating text.

2. **Encoding the Data**: For encoding the data, numerical representation is required, because neural networks can only work with numbers.

3. **Building the architecture**: After that, comes the architecture of the model, where each neural network consists of three sets of layers: input, hidden, and output. If we are dealing with a deep neural network, it might have multiple hidden layers.

4. **training the model**: Following the architecture stage comes the training.

5. **Evaluating the model**: Later comes to the evaluation phase, where a prediction can be formulated using the model discovered.

1.1.4 The Classification Problem

The mathematical representation of a neural network can be defined in the form of a function f, which takes an observation as its input and produces decisions as to its output. The function $f : A \to B$, can either be a distribution only on A or on both A and B.

Our goal is to formulate a mathematical model that can make predictions relating to implementations. Input is a real number, and output is processed as a non-linear function of the sum of its inputs. We denote the set of all predictor variables to be $\mathbb{R}$, and the set of all response vectors they represent to be $\mathbb{R}$, as well. In our research, as our input data includes 187 predictor variables, we focus on finding what class they belong. There are five classes in general about which we will discuss in later sections. The function f of the above problem can be written as:

$$f : \mathbb{R}^{187} \to \mathbb{R}^5$$

The classification problem identifies the unknown category or class from a given database of categories or classes. Such classification can either be binary or multi-class. We aim to predict the class (namely five in our case) from the 187 predictor variables. After evaluating and processing the data in our model, our classification will be considered a multi-class classification.

A brief introduction to binary classification and multi-class classification will be helpful to get a clear image of the two problems.

Binary Classification in Euclidean Space

Here, we are looking for classification, which is either binary or linear. The difference between binary and linear classification is that binary aims at distinguishing between two categories (it takes two possible values), whereas linear classifies data using a linear function.

If **N** represents the network which has a connection where $x^{(i)}$ is the input and $y^{(i)}$ is the binary-valued response, then when we talk about binary, we possibly look at values which are positive and negative examples. The response vector can only accept values in {0,1}; 0 corresponds to the "negatives" and 1 corresponds to the "positives." We require a model that would determine how to compute by using the input dimensions or features. This model will be called a binary linear classifier.

Definition 1. [Binary Classification in Euclidean Space] Let M and N be finite and disjoint subsets of $\mathbb{R}^n$ also written as $M, N \subseteq \mathbb{R}^n$. Let f be a function from $f : \mathbb{R}^n \to \{-1, 1\}$ such that

$$f(x) = \begin{cases} 1, & \text{if } x \in M \\ -1, & \text{if } x \in N \end{cases}$$

the input variables $x^{(i)}$ can be written as the linear function of the form

$$w_1x_1 + w_2x_2 + w_3x_3 + ... + w_sx_s + b = w^T x + b$$

where w corresponds to the 'weight vectors' and b is a 'scalar bias'. The predictive response of the above problem can be computed as

$$z = w^T x + b$$

such that

$$w^T m + b > 0, \forall m \in M$$

$$w^T n + b < 0, \forall n \in N$$

A solution to such a binary classification problem can be seen in Figure 1.3, where we are given two categories depicted in red and blue colors. Given any arbitrary point in the region, we have to predict if it belongs to the red class or blue class.

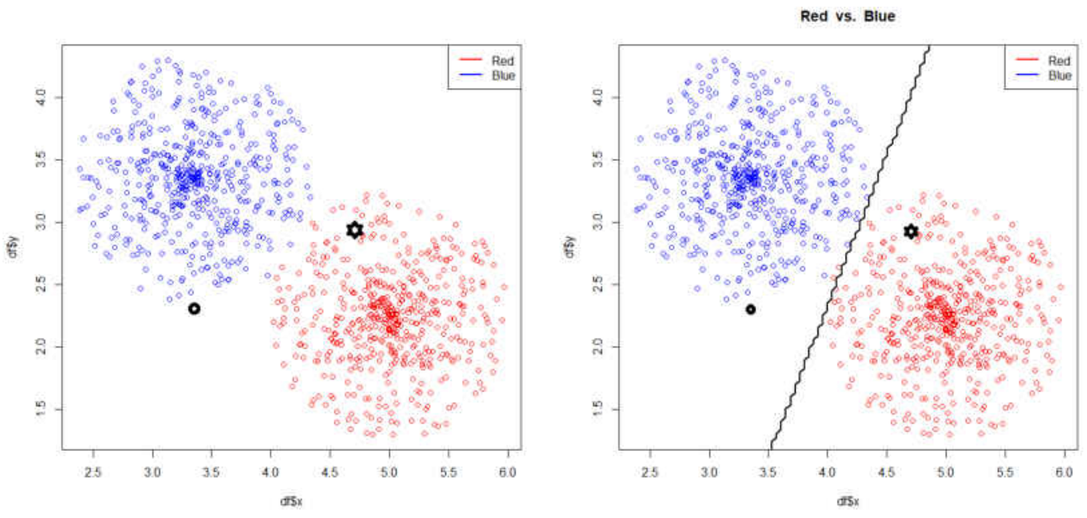


Figure 1.3: Binary Classification

In Figure 1.3-left, we are given two sets of points in $\mathbb{R}^2$ (R-Red and B-Blue) and the Figure 1.3-right depicts a hyperplane (the line) that separates these two sets. Suppose we are given two new points, which are un-colored points in $\mathbb{R}^2$. We have to predict the colors of the points using their relative positions to the hyperplane. Figure 1.3-left has two such points. Figure 1.3-right predicts the color of the points based on the fact that the point closer to class B is blue in color, whereas the one closer to class R is red in color.

Multi-class Classification in Euclidean Space

Multi-class classification is a generalization of binary classification. It aims at constructing a function that can correctly predict one class from N different classes to which a new subject belongs.

Definition 2. [Multi-class Classification in Euclidean Space] Let $M_1, M_2, ..., M_G$ be finite and disjoint subsets of $\mathbb{R}^n$, written as $M_1, M_2, ..., M_G \subseteq \mathbb{R}^n$. Let f be a function from $f_G : \mathbb{R}^n \to \{1, 2, 3, ..., G\}$ such that

$$f(x) = \begin{cases} 1, & \text{if } x \in M_1 \\ 2, & \text{if } x \in M_2 \\ ... & \\ ... & \\ G, & \text{if } x \in M_G \end{cases}$$

We want to find a solution to the above problem. This solution is a function that is capable of identifying the class or category to which the input variables $x^{(i)}$ belong, to predict the response vectors, making use of machine learning techniques to train the model without any prior knowledge.

The left side of Figure 1.4 represents different classes (or categories), and the right side of Figure 1.4 depicts a hyperplane (the line) that separates these classes, with one class in the middle. In contrast, the others have some degree of overlapping. In this problem, we are looking forward to creating a function capable of identifying the category (or class) and separating the class with a simple line without any prior information about the data. We will build a model based on the data for which knowledge to the preceding class exists, and then subsequently use the model to predict unknowns.

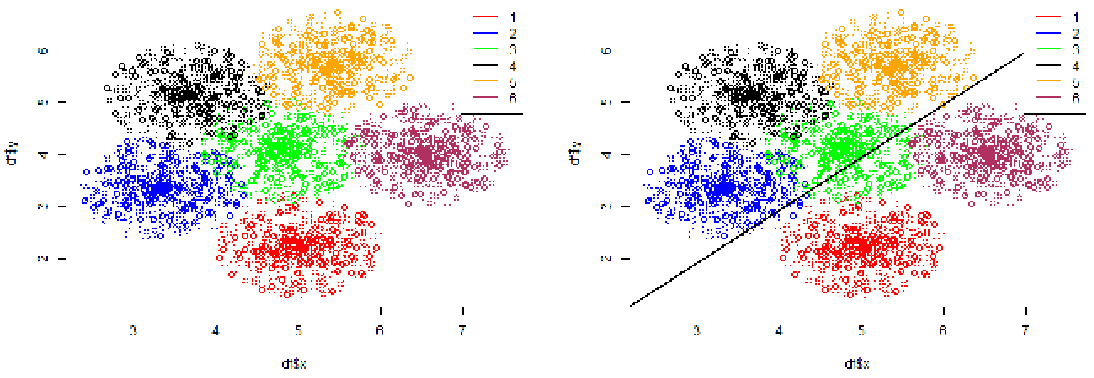


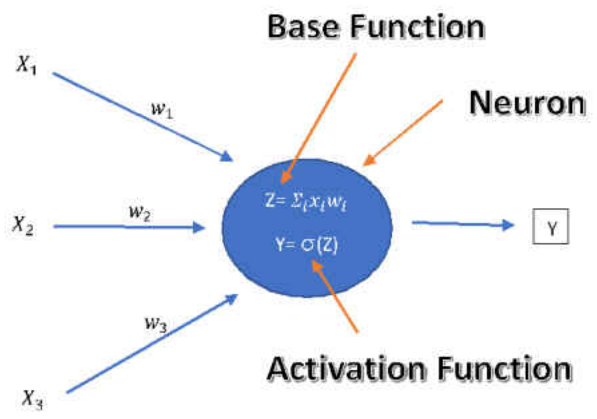
Figure 1.4: Multi-Class Classification

1.1.5 The Mathematics behind Neural Networks

A simple neural network is easy to solve, compute, and understand. However, if we have thousands of neurons, it may take a while for their computations, especially if the calculations involve dealing with matrices. In such circumstances, we use the machine learning computational power to make predictions or decisions on training data without being explicitly programmed to perform a specific task using instructions. We will focus on the structure, notation, initialization, and how the model makes classifications. A perceptron is a simplified model of how a brain works: a perceptron takes multiple inputs, which are strong enough to activate it, and outputs a signal.

The first layer is the input layer, and it represents the inputs to the model. The last layer is the output layer, which has one neuron for each of the K classes of the categorical response. Any layers in between are the hidden layers, (there can be any number of these hidden layers). There will be weight for each pair of nodes in the adjacent layers and bias for each node after the input layer.

A generic ANN model is composed of a collection of neurons, each with its own weights, organized into layers. The output of each layer is used as the input for the next



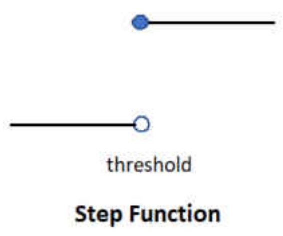


Figure 1.5: Left: Artificial Neural Network with 3 inputs (3 weight vectors), a base function, an activation function, and an output. Right: A Step function

layer, thus providing the name feed-forward. The output can be expressed as:

$$Output = \begin{cases} 0, & \text{if } \sum w_1 x_1 < threshold \\ 1, & \text{if } \sum w_1 x_1 \geq threshold \end{cases}$$

We set the threshold as b = -threshold and replace the step function with a smooth function of the same general shape. Whatever is used is called the activation function, with the logistic function being the most common choice. Therefore, the output can be represented as:

$$Output = logistic\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{1.1}$$

where the logistic is denoted by 'σ' and its value is

$$\sigma = \frac{e^x}{e^x + 1}$$

For an ANN model with N inputs, one hidden layer with L nodes, and a single output y, the mathematical model is described by:

$$y : \mathbb{R}^N \to (0, 1)$$

Our objective is to build a model: $y_i = \hat{f}(\boldsymbol{x}_i)$, using the back-propagation algorithm where $\sigma()$ is the activation function, known as sigmoid (logistic) function.

$$\sigma : \mathbb{R} \to \mathbb{R}, \qquad \sigma(x) = \frac{1}{1 + e^{-x}}$$

Therefore, equation 1.1 can be generalized as follows:

$$y(\mathbf{x}; \mathbf{w}, \mathbf{b}) = \sigma\left(\sum_{j=1}^{L} z_j \sigma\left(\sum_{i=1}^{N} w_{ij} x_i + b_{1j}\right) + b_2\right) \tag{1.2}$$

The optimal solution of the unconstrained optimization problem 1.2 can be defined as:

$$\min \sum_{i=1}^{S} (-Y_i \log y_i - (1 - Y_i) \log(1 - y_i))$$

We can notice that the ANN model represented by the output of the function can be estimated as a continuous value in the interval $(0, 1)$. In the case of binary classification, a threshold b is introduced. If the output $y \geq b$, the prediction is 0, otherwise the prediction is 1.

1.1.6 Organization of the thesis

The framework of the rest of this thesis is as follows. Chapter 2 gives a detailed description of the data-sets and provides an understanding of the electrocardiogram used in this research. Chapter 3 presents a detailed summary of the history of Artificial Neural Networks and its architecture. The detection of ECG data patterns by an ANN model can be found in Chapter 4. The implementation and experimental results are presented in Chapter 5, and the conclusion and future work follow in Chapter 6.

CHAPTER 2
DATA SETS DESCRIPTION

Classification is widely used in health care and bioinformatics, especially for the detection of arrhythmias. Arrhythmias are the irregularities in the heart rate, which may occur sporadically in a person's daily life. For capturing these infrequent rhythms, an ECG device is used to capture the potential electrical change of the heart. Therefore, the automatic detection of abnormal heartbeats from a large amount of ECG data is an important and essential task. In the last two decades, many methods have been proposed to address the problem of ECG beat classification. Simultaneously, deep learning has advanced rapidly and now demonstrates a state-of-the-art performance in various fields. To create our model, we propose an in-depth learning approach for ECG beat classification. We have conducted the experiments on the well-known PhysioNet MIT–BIH Arrhythmia Database [15], and PTB Diagnostics datasets [16], which are real ECG time-series datasets from thousands of patients.

2.1 MIT-BIH Arrhythmia dataset

The MIT-BIH Arrhythmia Database [15] consists of 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects, studied by the BIH Arrhythmia Laboratory between 1975 and 1979. Twenty-three recordings were chosen at random from a set of 4,000 24-hour ambulatory ECG recordings, collected from a mixed population of inpatients (60 percent) and outpatients (40 percent) at Boston Beth Israel Hospital. And the remaining 25 records were selected from the same set, to include less common, but clinically significant arrhythmias, that would not be well represented in a small random sample. These listings were digitized at 360 samples per second per channel with an 11-bit resolution over a 10 mV range. Two or more cardiologists independently annotated each record; disagreements were resolved to obtain the computer-readable reference annotations for each beat (approximately 110,000 annotations in all) included with the database.

Time series are zero-padded, 187-element vectors containing the ECG lead II signal for one heartbeat. We used 80 percent of the data from the entire dataset to train the model, further used the knowledge acquired for testing the rest of the 20 percent of the data and compared our results with the scientific literature. Final results show that our model is more efficient than state of the art in terms of accuracy and competitiveness in terms of sensitivity and specificity.

In the MIT-BIH Arrhythmia dataset, we have a total of 87,544 samples for training data and 21,892 samples for testing data. Each sample consists of a time series of 187 variables, and the last one i.e., 188^{th} variable, represents the class type. Labels [0, ..., 4] represent normal heartbeats (type 0) and four classes of arrhythmia (types 1, 2, 3, and 4), accounting for five different beat categories. The class distribution is highly skewed. Table 2.1 depicts the data classes for MIT-BIH with several testing and training data samples. In total we have N = [‘90,589’, ‘2,779’, ‘7,236’, ‘803’, ‘8,039’], which contains both the training and testing data for the categories [‘N’, ‘S’, ‘V’, ‘F’, ‘Q’] respectively. Table 2.2 gives a detailed description of mappings between beat annotations and AAMI EC57 [8] categories. We have a beat for each of the five different beat categories, as shown in Figure 2.1.

“N” stands for ‘Normal’ and hence the category. Categories ‘S’, ‘V,’ ‘F,’ ‘Q’ account for the sick people with irregular heartbeat rhythms. As it is evident from the table that category N has the most number of patients, and there are always more people healthier than sick. Our priority in this research is to find the number of people who are sick. As long as they are not healthy, any other category would hardly make any difference.

We did not perform any pre-processing on the data. The data was arranged in the

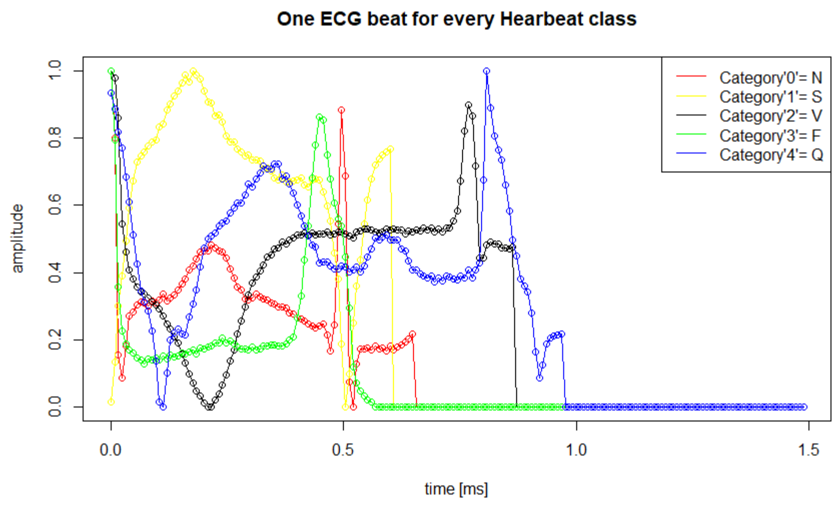


Figure 2.1: ECG waves

CLASS	CATEGORY	No of Samples(Test data)	No of Samples(Train data)
0	N	18,118	72,471
1	S	556	2,223
2	V	1,448	5,788
3	F	162	641
4	Q	1,608	6,431

Table 2.1: Data classes for MIT-BIH Dataset

sequence of the categories, i.e. [0, ..., 4]. We had to shuffle the data to get better results both in the training phase and the testing phase in the case of MIT-BIH Data sets.

2.2 PTB Diagnostic database

In PTB Diagnostic [16] the database consists of 549 ECG records from 290 subjects (aged 17 to 87, mean 57.2; 209 men, mean age 55.5, and 81 women, mean age of 61.6; ages were not recorded for 1 female and 14 male subjects): 148 diagnosed as MI, 52 healthy control, and the rest are diagnosed with 7 different diseases. We have a detailed clinical

CLASS	CATEGORY	ANNOTATION
0	N	Normal
		Left/Right bundle branch block
		Atrial Escape
		Nodal Escape
1	S	Atrial Pressure
		Aberrant Atrial Premature
		Nodal Premature
		Supra-Ventricular Premature
2	V	Premature ventricular contraction
		Ventricular escape
3	F	Fusion of ventricular and normal
4	Q	Paced
		Fusion of paced and normal
		Unclassificable

Table 2.2: Summary of mappings between beat annotations and AAMI EC57 [8]

summary, including age, gender, diagnosis, and where applicable, data on medical history, medication and interventions, coronary artery pathology, ventriculography, and hemodynamics. The clinical summary is not available for 22 subjects. ECG signals are from 12 leads, and in our study, we have only used ECG lead II and worked with myocardial infarction (MI) and healthy control categories having 148 and 52 number of subjects respectively for our analysis.

The PTB classification is a binary classification consisting of only two categories 0 and 1. Either the subjects are healthy or unhealthy, which points to the heartbeat's normal rate or the abnormalities in the heart rate, respectively. Table 2.3 depicts the data classes

CLASS	CATEGORY	No of Samples(Test data)	No of Samples(Train data)
0	N	810	3,236
1	AB	2,102	8,404

Table 2.3: Data classes for PTB Diagnostic

for PTB with a number of samples of testing and training data. We have a beat for each of the two categories, as shown in Figure 2.2.

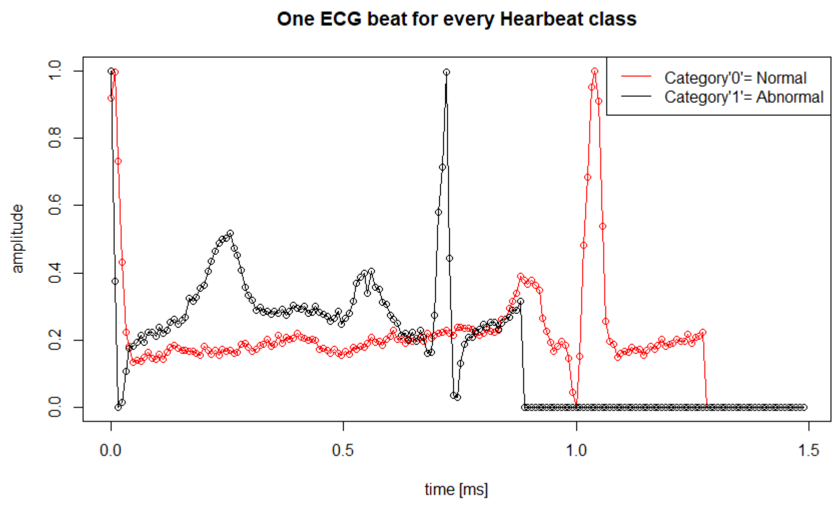


Figure 2.2: ECG waves for PTB Diagnostic Dataset

2.3 Electrocardiogram (ECG)

The Electrocardiogram (ECG) measures the electrical activity of the heart by a tool. This tool is external to the body in a non-invasive way. The resulting ECG signals' interpretation helps to detect a wide range of heart diseases such as abnormal heartbeat rhythm (arrhythmia), heart attack, or heart failure. However, the complexity of the abnormal ECG signals makes it very difficult to detect its characteristics. The electrical signals trigger the heart muscles to contract and pump blood through the arteries. As the human body is a

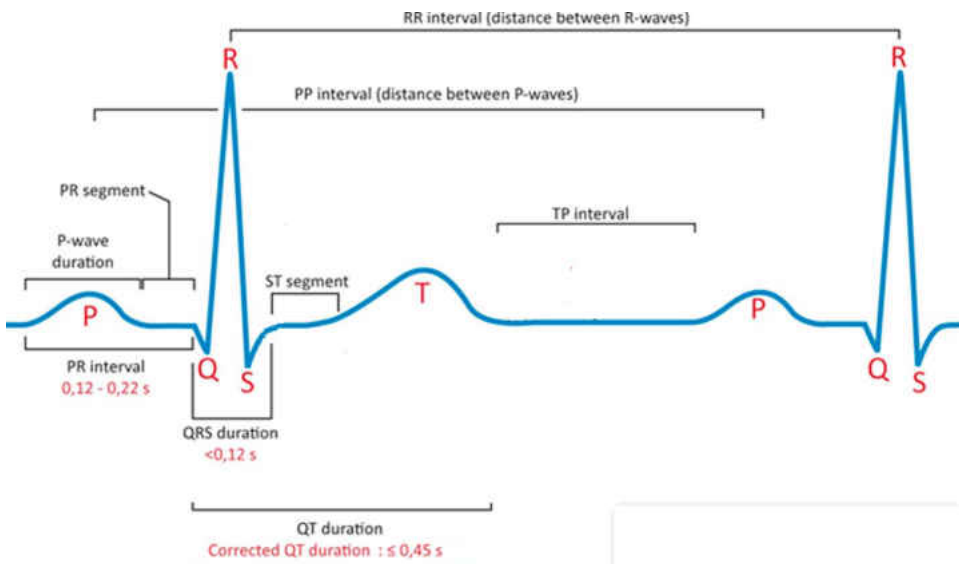


Figure 2.3: Most common waveform of the classical ECG curve [13]

great conductor due to the massive amounts of fluids it contains, the electrical signals can propagate through the body rapidly. The heart's electrical activity can be recorded through small metal electrode patches attached to the skin of a person's chest, arms, and legs. An ECG signal records the pace at which the heart is beating, its rhythm (steady or irregular), strength, and timing as these signals pass through the heart to investigate further symptoms related to heart problems. The focus is to detect the electrical impulses coming from the different directions within the heart and categorizing whether these impulses are normal patterns or abnormal patterns. The normal ECG signal can be divided into three sections.

An ECG is recorded using 10 electrodes, which capture 12 leads (signals) to get a complete picture of the heart. Each lead captures the electrical activity of the heart from a different angle. 12 leads are required for accurate diagnostic purposes; however, one lead can offer valuable information for the quick and initial assessment of the patient.

Each ECG cycle consists of five waves: P, Q, R, S, and T corresponding to different phases of the heart activities. The P-wave is the first positive deflection on the ECG, the QRS wave represents the simultaneous activation of the right and left ventricles, although most of the QRS waveform is derived from the larger left ventricular musculature and the T-wave should be concordant with the QRS complex. The P-R interval is measured from the beginning of the P-wave to the first deflection of the QRS complex. The Q-T interval is measured from the first deflection of the QRS complex to the end of the T-wave at the isoelectric line.

- P Wave: P-waves are the smaller spikes, i.e., a slight increase in the voltage right before the larger peak (the R wave). The P-waves represent the atria's depolarization during atrial systole that represents atrial depolarization (irregularly regular). They should be at the approximately same duration, direction, and shape through the entire ECG. P wave might be a small rise, or bump, on the graph. It will not be as sharp or as high as the QRS complex.

- QRS Wave: QRS wave complex is a small drop in voltage (Q) followed by voltage peak (R) and another small drop in voltage (S). The Q wave is the downward or negative dip right before the large spike. The R wave is right after that and is usually the largest spike on the readout. Following that is the S wave, which is the dip down below the baseline again. The QRS complex represents the depolarization of the ventricles during ventricular systole. The atria repolarize on the same period of the QRS complex. However, it does not show any effect on the ECG plot due to the size of the atria compared to the ventricles is known as the QRS complex, with each letter corresponding to a different part of the heart's action. It is important to note that the "R" of the complex is the area from which the values for analysis are taken. When we have several heartbeats next to each other, then the distance (in milliseconds) between each "R" is defined as the "RR interval" (or sometimes the "NN interval" to emphasize that the heartbeats are normal).

- ST segment: The ST interval is the time between the end of the QRS complex and

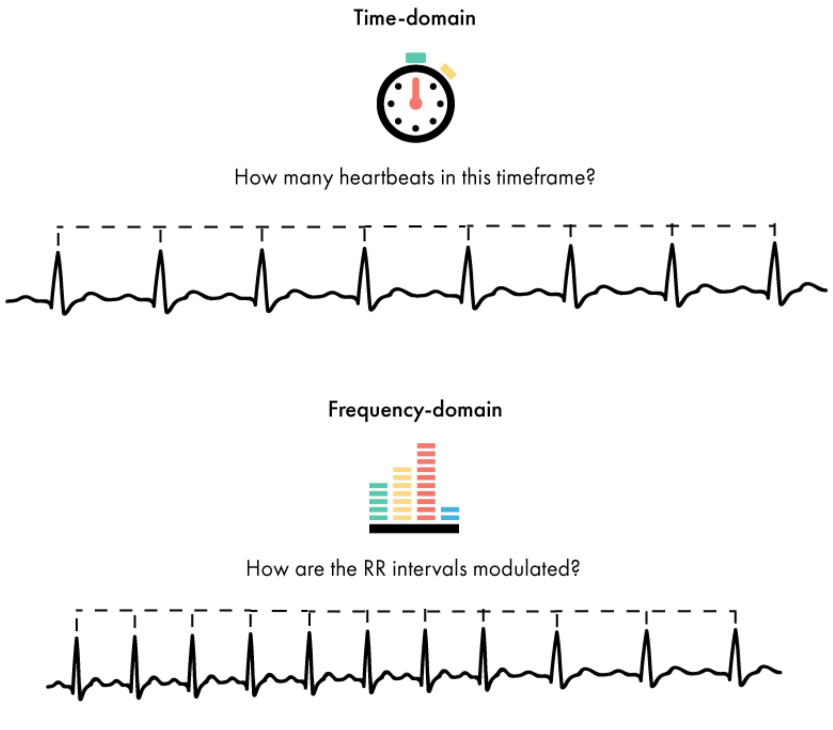

Time-domain
How many heartbeats in this timeframe?
Frequency-domain
How are the RR intervals modulated?

Figure 2.4: Time-vs-frequency-domain [7]

the start of the T wave. It reflects the period of zero potential between ventricular depolarization and repolarization.

- T Wave: T wave is a small peak (slightly bigger than P wave, which follows the QRS complex). It represents the repolarization of the ventricles during the relaxation of the cardiac cycle.

- RR Interval: One may have a regular or irregular heart rhythm. If your rhythm is irregular, it may be regularly or irregularly. It is crucial to see if the intervals are all the same lengths. A regularly irregular rhythm has a pattern of the irregular rhythm. An irregularly irregular rhythm has no pattern at all and is all over the place.

The crucial and most informative metric is not only dependent on the heart rate, but also on the rate at which it varies. In the detection of ECG lies a counter-intuitive capability about the accurate diagnosis of Heart Rate Variability (HRV), which can be associated with good health, or lower, related to ill health. Usually, the calculation of HRV can be tricky and can be done by using either time or frequency methods [7]. In this context, time-domain methods mean that the beat-to-beat amount within an amount of time is used, while frequency-domain methods count the amount of low and high-frequency beats that occur is explained further in the Figure 2.4. But this methodology does not concentrate on the measure of data classification, though it would be good to see it as future work.

2.3.1 Interpretion of the ECG

To read the ECGs accurately and confidently, a lot of practice is required. Although it is essential to be methodical, every ECG reading should start with an assessment of the rate, rhythm, and axis, yet the approach always reveals something about the ECG regardless of how unusual it is. According to medical practitioners, in a normal ECG, heartbeats in regular sinus rhythm are between 60 and 100 beats per minute (bpm). There can be different characteristics of normal and abnormal waves and intervals in terms of duration and amplitude. Such a proposal is presented in paper [13]. In addition, a good description of heart diseases can be found in [5].

CHAPTER 3

ARTIFICIAL NEURAL NETWORKS

3.1 History

How does a brain work? Humanity has always been captivated by the functioning of the human brain. The advancements in science and technology gave birth to the idea of making machines that can replicate the working of the human brain using artificial intelligence and make decisions on its own. Can such a device be made? Will it be able to exploit the biological functions of the brain effectively? Can a machine be conscious of its own decisions? Can a neural network adequately simulate the human brain? With the augmentation of Artificial Neural Networks, the dream came true. In 1943, an American neurophysiologist and cybernetician Warren McCulloch and a logician Walter Pitts came up with the idea of creating a computational model for neural networks. In the late 1940s, D.O.Hebb modulated a learning hypothesis on the mechanism of neural plasticity popularly known as Hebbian Learning, and he is also the author of the book 'Organisation of Behaviour.' In 1954 Farley and Wesley A. Clark simulated a Hebbian network making use of computational machines. Soon after, Rosenblatt created a perceptron (an algorithm for supervised learning of binary classifiers) in 1958. It was first of its kind to discover patterns. In 1965, Ivankhnenko and Lapa published a functional network with multiple layers as the 'Group Method of Data Handling.' Due to very few advances, von Neumann's architecture was gaining popularity. The basics of backpropagation in the context of control theory were derived by Kellen and Bryson in the early 1960s. But in the early 1970s, Marvin Minky and Seymour Papert discovered multiple problems with the perceptron model, which were later fixed by Paul Werbos in 1975 with the usage of Back Propagation. It was the first multi-layered network of its kind developed in 1975 as an unsupervised network. Using Linnainmmaa's AD method of discrete connected networks, which she published in 1972, Werbos applied it to neural networks in a way that became popular. For many years after that, research stagnated until 1992, when max-pooling was introduced to aid in 3D object recognition as it helped with the least shift-invariance and tolerance to deformation. That same year, Schmidhuber adopted a multi-layer hierarchy of networks to pre-train a level at a time by unsupervised learning, which was fine-tuned by backpropagation. In modern times, neural networks have several applications, one of which is the basis of the study. The primary thought behind neural networks is to work in computers after it is successfully implemented in nature. Down the road, advancements in the technological aspect and the hardware of neural networks can prove substantial.

Development in the field is stagnant due to the slow learning capacity of neural networks due to a limited number of processors. The focus of today's companies is to create a "silicon compiler" which can generate a particular type of integrated circuit optimized for the operation of neural networks. Amongst the various kinds of chips being developed, analog signals might be considered a thing of the past as they vary between a minimum and maximum values, whereas digital signals focus on two distinct states (on or off, 1 or 0). Nonetheless, brain signals are likely to be analog signals than digital signals; it may be a while before optical signals can be used in commercial applications.

3.2 Architecture of Neural Networks

The architecture of neural networks is inspired by the biological functioning of the human brain to solve problems. To better understand the neural networks, we should look at the structure and working of the biological neurons in the brain. The brain consists of a circuit of neurons with a cell body, an axon, dendrites, and axon terminal (Figure 3.1). Each neuron receives an electro-chemical input from the other neurons via a connection at the dendrites, which may be extensive. Connections, called synapses, are formed if the sum of these electrical inputs is sufficiently powerful to activate the neuron from the axons to dendrites through dendrodendritic synapses. Dendrites are responsible for the collection of information and passing it on to the cell bodies where the information is processed and then passed to axons through the synapses. The axons are inter-connected to the nerve cells through dendrites. In a neural network, dendrites are named as input channels, cell bodies as nodes, synapses as the activation function and axons as the output channel. Our brain is composed of these interconnected electro-chemical transmitting neurons where each neuron is responsible for performing a weighted sum of its inputs and then firing a binary signal if the total input exceeds a certain level. From a wide range of elementary processing units, the brain manages to perform extremely complex tasks.

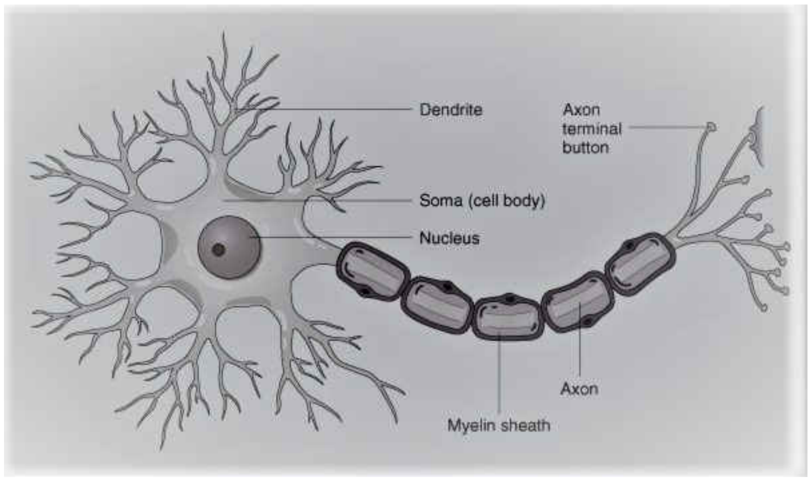


Figure 3.1: Structure of a nerve cell in the human brain

Thus, this is the model on which artificial neural networks are based. So far, they haven't even come close to modeling the brain's complexity, though they have proved to be good at problems that are easy for humans but complicated for a traditional computer, such as image recognition and predictions based on prior knowledge.

Neural networks replicate the functioning of the brain as it consists of millions of artificial neurons called units arranged in a series of layers, each of which are interconnected. Input units are designed to receive various forms of information from the outside world, which the network will tend to learn about, or process. Whereas the output units are responsible for how the network and signal will respond to the information it has learned during the process. In between the input units and output units, there may be one or more layers of hidden units that are interconnected to every unit in the layers on either side. The connections between each unit are represented by a number called weight, which can either be positive or negative, depending on the value assigned to it. Apart from these three layers, there is a constant vector called bias associated with each layer, which adjusts the error effect. In addition, an activation function, whose purpose is to introduce non-linearity, is present at each node and decides whether a neuron should be activated or not by calculating the weighted sum and then adding bias to it. The learning phase of the neural network consists of a statistical model that uses input units to predict the desired output, which is then associated with new inputs. Whereas the other phase includes the manual operations where the whole concept is based on similarity learning. Similarity learning measures the relationship between two objects by detecting patterns of information which are fed into the network via the input channels. Input channels trigger the layers of hidden units, and these, in turn, arrive at the output channels. This is a typical example of a Feed-forward network. After receiving the data as inputs from the left side, the inputs are multiplied by the weights. Information in the feed-forward neural network is fed forward from one layer to the next in the forward direction only. There are no feedback loops.

Components of Neural Networks

Neurons: Neural networks are composed of artificial neurons that retain the biological concept of neurons, making use of external data, such as images, words, or documents, combining those with their activation state to accomplish the output, such as recognizing characters in a word document.

Weights: Weight is a value that is initialized when a neural network is trained to produce

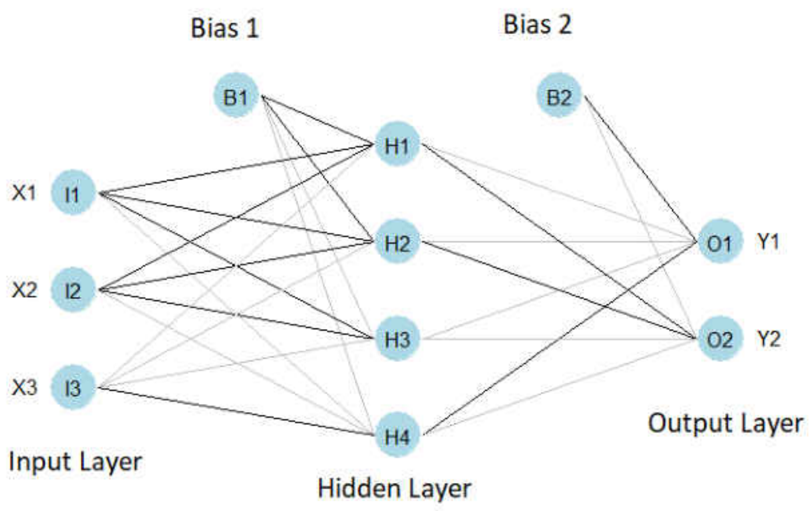


Figure 3.2: Artificial Neural Network with 3 inputs, one hidden layer (with 4 neurons, one bias), and 2 outputs (and one bias)

a decision. It is a connection between neurons, and it acts as the vehicle through which neurons participate in the activation function to acquire related outputs. Usually, in math and programming, we view weights in the form of matrices representing the strength of the value to which it is applied. The value of weights can either be initialized randomly or with a distribution such as normal or gaussian. Weights are modified to predict accurate results.

The Figure 3.2 has an input layer with 3 neurons, a hidden layer with 4, and a weight matrix depicting 3 rows and 4 columns into which we can insert the values of each weight in the matrix accordingly. We can name this matrix as W1. In the case, if we have more layers, we would have more weight matrices. More generally, if a layer L has N neurons, and the next layer L+1 has M neurons, the weight matrix is an N-by-M matrix (N rows and M columns) [22].

Bias: Bias can also be treated as a weight. It is simply a constant vector added to the product of inputs and weights and is utilized to offset the result by adding a change to the output. The bias is responsible for shifting the result of activation function towards a positive or a negative side. Just as weights can be expressed in the form of a matrix, bias

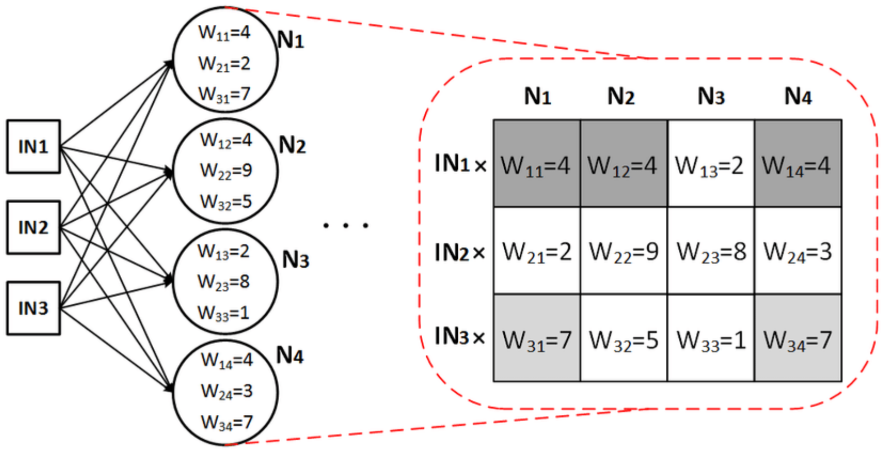


Figure 3.3: The mapping of first hidden layer onto a 3×4 Weight Matrix [22]

can also be written as a matrix with 1 column. In the absence of bias, a neural network will just perform matrix multiplication on inputs and weights. The bias is the causative factor to activate the activation function, which reduces the variance, increasing the flexibility and the generalization without the fear of over-fitting the data set.

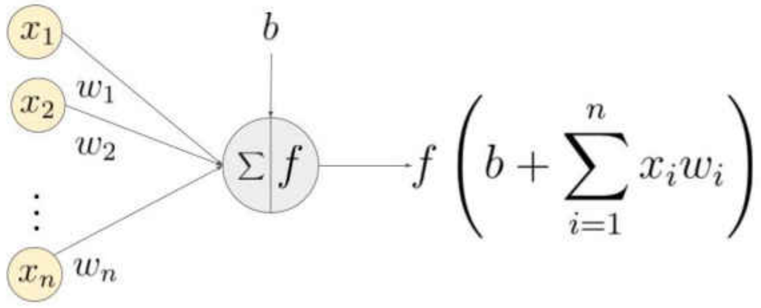


Figure 3.4: A NN with inputs, their corresponding weights, bias and activation function f applied to weighted sum of inputs [11]

Activation Function: An activation function (Figure 3.4) is a mathematical function used to normalize the data by determining the final value of the current memory state and output. The activation function maps the output of several equations to the desired range. There are certain activation functions, and we can choose one of them to suit our needs.

- Linear Activation Function: It is a simple function of the form $f(x) = x$. The input, in this case, is not modified to pass to the output. A linear function has the equation similar to as of a straight line i.e., y = ax. No matter how many layers we have, if all are linear, the last layer's final activation function is nothing but just a linear function of the input of the first layer. The range lies between -inf to +inf. The linear activation function is used at just one place i.e., the output layer. If we differentiate linear function to bring non-linearity, the result will no more depend on input "x" and the function will become constant, it won't introduce any ground-breaking behavior to our algorithm.

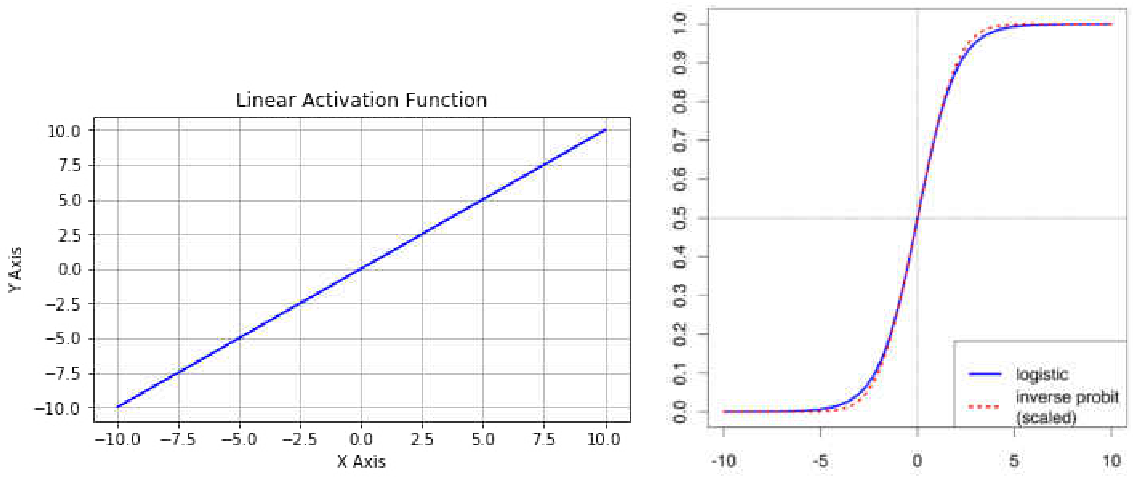


Figure 3.5: Linear and Non-Linear Activation Functions [11]

- Non-Linear Activation Functions: Such activation functions separate the data that is not linearly separable, and these are the ones mostly used. A non-linear equation maps inputs to outputs. There are a few examples of non-linear activation functions such as Sigmoid, Tanh, ReLu, and many more. We will discuss some commonly used functions in detail.

  1. Rectified Linear Units (ReLU) — ReLU is used to ensure that our output does not go below a certain threshold i.e., zero (or negative). Therefore if z (every-

thing coming into the neuron) is greater than zero, our output remains z, or if z is negative, our output is zero. The formula is f(z) = max(0, z). We select the maximum between 0 and z.

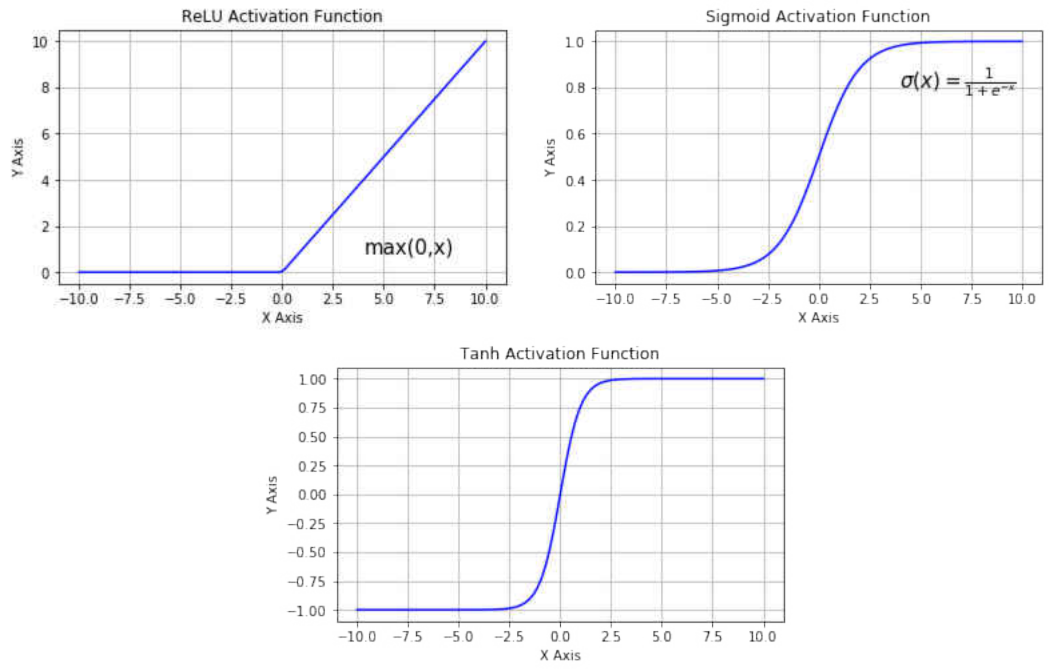


Figure 3.6: ReLu, Sigmoid, Tanh Activation Functions [11]

2. Tanh — The activation function Tanh is used to find the hyperbolic tangent of z and return it. It can be written as $f(z) = tanh(z)$. The calculations can be performed using a scientific calculator.

3. Sigmoid — To activate the sigmoid function we use the following formula:

$$f(x) = 1/(1 + e^{(-1\times z)})$$

4. Softmax Function — The softmax function is also a type of sigmoid function but is handy when we are trying to handle classification problems. It is non-linear. Usually, it is used when trying to handle multiple classes. The softmax function would narrow down the outputs for each class between 0 and 1 and

would also divide by the sum of the outputs. The softmax function is ideally used in the classifier's output layer, where we are trying to attain the probabilities to define the class of each input.

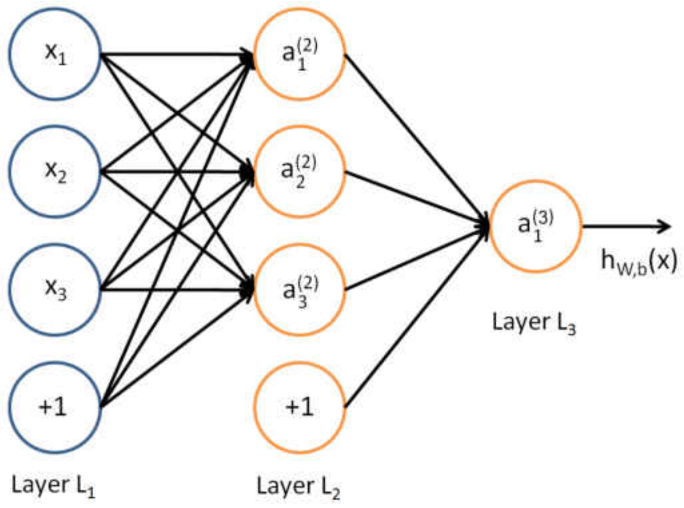


Figure 3.7: Structure of a simple neural network

Organisation: The mathematics behind the model can be explained in simpler terms. Consider a model having 4 input nodes (3+1"bias"), one hidden layer with 4 nodes (3+1"bias") and an output node in Figure 3.7. Name the "bias node"s as x_0 and y_0, while place the input nodes in the form of a vector X and the nodes from the hidden layer in vector A. The weights between the input layer and the hidden layer represents a 3×4 matrix and can be denoted as Θ. In general, a network having 'a' units in a layer 'j' and 'b' units in the layer 'j+1', then Θ_j is of the dimension $b \times (a+1)$.

$$X = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix}, A = \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix}, \Theta^{(1)} \begin{pmatrix} \Theta_{10} & \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{20} & \Theta_{21} & \Theta_{22} & \Theta_{23} \\ \Theta_{30} & \Theta_{31} & \Theta_{32} & \Theta_{33} \end{pmatrix}$$

To calculate the activation function for the hidden layers, we need to multiply the input vector X with the weights Θ^1 for the first layer i.e., $X * \Theta^1$ and after the multiplying the input vectors with the weights we get

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

The computed output node value of the hypothesis function is the multiplication of the hidden layer vectors with the weights Θ for the second layer $(A * \Theta)$ in case of one hidden layer and 4 nodes is as follows:

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(1)}a_1^{(2)} + \Theta_{12}^{(1)}a_2^{(2)} + \Theta_{13}^{(1)}a_3^{(2)})$$

Generalizing the above equation we get

$$a_n^{(L)} = [\sigma(\Sigma_m \Theta_{nm}^L[...[\sigma(\Sigma_j \theta_{kj}^2[\sigma(\Sigma_{ji}^1 x_i + b_j^1)] + b_k^2)]...]_m + b_n^L)]_n$$

where L layers have n nodes and L-1 layers have m nodes.

Next comes the activation function σ, which decides whether to activate the nodes based on the weighted sum. Denote the weighted sum value as z. For this purpose different activation functions can be used depending on what best fits the need.

To create a successful learning model, we need to find the network's biases and weights and then adjust the 'bias' node to get a better prediction for the data. The equation for the cost function can be expressed as follows:

$$J(\theta) = 1/m\left(\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})\right)$$

Such a type of classification problem can only take discrete values 0,1. Hence the solution y can only be in one type of class. In order to satify our hypothesis

$$0 \leq h_\theta(x) \leq 1$$

we can define it as

$$h_\theta(x) = g(\theta^T x)$$

The goal is to optimize the cost function and adjusting weights and biases in an attempt to minimize

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0 \end{cases}$$

Since y is always 0 or 1, we can rewrite the cost function as :

$$Cost(h_\theta(x^{(i)}), y^{(i)}) = -y\log(h_\theta(x)) - (1-y)\log(1 - h_\theta(x))$$

Then, comes the stage of training a neural network via the backpropagation algorithm (a technique to adjust the weights to compensate for each error found during learning). Usually, a general technique used to find parameters that minimize a cost function is gradient descent, an algorithm used for searching the minimum of a differentiable function by calculating derivatives. An estimate of the parameters is partly controlled by the magnitude of derivative, but influenced by its learning rate (steps a model takes to adjust for errors in each propagation). A high learning rate shortens the training time, but with lower ultimate accuracy, while a lower learning rate takes longer, but with the potential for greater accuracy.

CHAPTER 4

ANN MODEL FOR DETECTING ECG DATA PATTERNS

This chapter introduces some formal notations for the ECG data that we process. We formally describe the specific Artificial Neural Network (ANN) model we created for detecting patterns (standard and different varieties of abnormal cases) from ECG data.

4.1 ECG DATASETS FOR THE MODEL

For a formal description of the model, we will begin by describing the ECG data. Let $\mathcal{D} = \mathcal{N} \cup \mathcal{A}$ be an ECG dataset, where $\mathcal{N}$ and $\mathcal{A}$ are the normal and abnormal ECG disjoint datasets, respectively. We will use $size(\mathcal{N})$ and $size(\mathcal{A})$ to describe the number of ECG samples in the normal and abnormal datasets, respectively.

Definition 3. [ECG signal or time series] A normalized ECG signal (time series) is a finite sequence $\{s_k\}_{1 \leq k \leq 187} \in [0,1]^{187}$. We write $s_i = s(i),\ i = 1...187$.

Essentially, a normalized ECG signal is the time-continuous record of a person's heartbeat produced by electrocardiography sensors, which is subsequently sampled, converted into a sequence of numbers, and normalized. We can eventually describe the normal ($\mathcal{N}$) and ($\mathcal{A}$) datasets as collections of ECG signals as follows:

$$\begin{aligned} \mathcal{N} &= \left\{ \{n_k^{(i)}\}_{1 \leq k \leq 187} \mid n^{(i)} \text{ an ECG signal}, i = 1...size(\mathcal{N}) \right\} \\ \mathcal{A} &= \left\{ \{a_k^{(i)}\}_{1 \leq k \leq 187} \mid a^{(i)} \text{ an ECG signal}, i = 1...size(\mathcal{A}) \right\} \end{aligned} \tag{4.1}$$

The abnormal dataset $\mathcal{A}$ may be further divided into a number of disjoint datasets, where each such dataset corresponds to a type of abnormal ECG signal.

$$\begin{aligned} \mathcal{A} &= \bigcup_{j=1}^{4} \mathcal{A}_j,\ j = 1, ..., 4 \\ \mathcal{A}_k \cap \mathcal{A}_l &= \emptyset,\ \forall k, l \in \{1,2,3,4\},\ k \neq l \end{aligned} \tag{4.2}$$

We note that some ECG datasets are not further divided into subsets, as described by (4.2). For instance, the PTB Diagnostic dataset distinguishes between normal and abnormal ECGs, but not between different types of irregular heartbeats. We call these datasets *binary datasets*. However, the MIT-BIH Arrhythmia does disseminate between different types of heart conditions and provides the information described by (4.2). We call these datasets *multiclass datasets*.

For any given ECG signal, we define the signal type by the ECG data subset the signal belongs to, as follows.

Definition 4. [Binary signal type] The binary signal type for an ECG signal s is a function $\beta : \mathcal{D} \to \{0,1\}$ (where $\mathcal{D}$ may be a binary or multiclass dataset) defined as

$$\beta(s) = \begin{cases} 0, & \text{if } s \in \mathcal{N} \\ 1, & \text{if } s \in \mathcal{A} \end{cases}$$

Definition 5. [Multiclass signal type] The multiclass signal type for an ECG signal s is a function $\mu : \mathcal{D} \to \{0,1,2,3,4\}$ (where $\mathcal{D}$ must be a multiclass dataset) defined as

$$\mu(s) = \begin{cases} 0, & \text{if } s \in \mathcal{N} \\ j, & \text{if } s \in \mathcal{A}_j,\ j = 1,2,3,4 \end{cases}$$

With the terminology introduced in this section, we next proceed to describe our classification models.

4.2 The binary classification model

Definition 6. [Binary classification model] Let $\mathcal{D}$ be a binary or multiclass dataset. A mapping

$$\hat{\beta} : \mathcal{D} \to \{0,1\}$$

is called a binary classification model.

Using the *not equal* operator definition:

$$\neq(x,y) = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x \neq y \end{cases}$$

we define next the accuracy of a binary classification model on any subset of a binary dataset.

Definition 7. [Accuracy of a binary classification model] Let $\hat{\beta} : \mathcal{D} \to \{0,1\}$ be a binary classification model and let $\mathcal{D}_{tst} \subset \mathcal{D}$. The accuracy of $\hat{\beta}$ on $\mathcal{D}_{tst}$ is defined as:

$$acc(\hat{\beta}) = \frac{1}{size(\mathcal{D}_{tst})} \sum_{s \in \mathcal{D}_{tst}} \neq (\beta(s), \hat{\beta}(s))$$

4.3 The multiclass classification model

Definition 8. [Multiclass classification model] Let $\mathcal{D}$ be a multiclass dataset. A mapping

$$\hat{\mu} : \mathcal{D} \to \{0,1,2,3,4\}$$

is called a multiclass classification model.

Similar to the binary case, we define the accuracy of the multiclass classification model as follows.

Definition 9. [Accuracy of a multiclass classification model] Let $\hat{\mu} : \mathcal{D} \to \{0,1,2,3,4\}$ be a multiclass classification model and let $\mathcal{D}_{tst} \subset \mathcal{D}$. The accuracy of $\hat{\mu}$ on $\mathcal{D}_{tst}$ is defined as:

$$acc(\hat{\mu}) = \frac{1}{size(\mathcal{D}_{tst})} \sum_{s \in \mathcal{D}_{tst}} \neq (\mu(s), \hat{\mu}(s))$$

4.4 The Artificial Neural Network model implementation

We have implemented our binary and multiclass classification models (Definitions 6, 8) using the high-level neural network framework Keras for R [2]. Figure 4.1 shows the

```
#create the model
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
      input_shape = c(187)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = 'softmax')
```

Figure 4.1: Implementation of the binary classification model $\hat{\beta}()$

implementation of the binary classification model $\hat{\beta}()$ from Definition 6. In the binary model, with 187 predictor variables, we used this representation as input to a two layer fully-connected network with ‘relu’ activation function having 256 units and a learning rate of 0.4 for the first layer and 128 units and learning rate of 0.3 for the second layer, to predict for the layers and a softmax layer with 2 units to predict output class probabilities. The multiclass classification model $\hat{\mu}()$ from Definition 8 is shown in Figure 4.2. Similar to the binary model, in a multiclass model, we used the same representation as input to a two layer fully-connected network with ‘relu’ activation function having 256 units and a learning rate of 0.4 for the first laye,r and 128 units and a learning rate of 0.3 for the second layer. To predict for the layers, the only difference is a softmax layer with 5 units to predict output class probabilities.

```
#create the model
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
  input_shape = c(187)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 5, activation = 'softmax')
```

Figure 4.2: Implementation of the multiclass classification model $\hat{\mu}()$

4.5 Estimating the performance of the classification models

We performed multiple experimental tests using both binary and multiclass classification models that we implemented. We also measured accuracies (Definitions 7, 9) of both models for different case scenarios as described in Chapter 5. For each experiment, we estimated the confidence intervals of accuracy being at 95% of accuracy significance levels and the corresponding p-value from the McNemar's test [12].

More specifically, we computed the binomial proportion confidence interval (first given in [3]) for each model's accuracy on $\mathcal{D}_{tst}$ as follows:

$$\begin{aligned} accL &= \frac{n!}{(s-1)! \cdot (n-s)!} x^{s-1}(1-x)^{n-s} \\ accU &= \frac{n!}{(s)! \cdot (n-s-1)!} (1-x)^{s} x^{n-s-1} \end{aligned} \tag{4.3}$$

where:

$accL$, $accU$ are the lower and upper bounds for accuracy, respectively

$n = size(\mathcal{D}_{tst})$ is the number of samples in the test data

$s = \sum_{s \in \mathcal{D}_{tst}} \neq (\beta(s), \hat{\beta}(s))$ or $s = \sum_{s \in \mathcal{D}_{tst}} \neq (\mu(s), \hat{\mu}(s))$ and represents the success rate for each model, respectively,

$x = \frac{1 - 0.95}{2}$ represents the radius of the accuracy significance being considered.

The experimental results of our model evaluations are reported in Chapter 5.

CHAPTER 5
EXPERIMENTAL RESULTS

We organized our experiments in two categories: (i) binary classification using the PTB Diagnostic ECG Database [1, 16], and (ii) multiclass classification using the MIT-BIH Arrhythmia Database [15]. We have performed an initial analysis of the data source for all experiments, collecting and comparing information from both "normal" and "abnormal" data samples. A typical classification experiment, for both binary and multiclass classification was performed by randomly selecting 80% of the data for training the model and the remaining 20% of the data for testing the model. However, each model training phase was performed in parallel with testing the model on a small fraction of the training data. Consequently, we could record and visualize the progress of the model training process (in terms of precision gain and loss function evolution) during searching for the optimal model parameters.

All results of the experiments we conducted are reported in the subsequent sections of this chapter.

5.1 Detecting heart disease using binary classification

The PTB Diagnostic ECG Database [1, 16] contains records from 290 patients. There are multiple records from the same patient, and each record includes 15 simultaneously measured signals, sampled at 1,000 samples per second. The data is provided as a collection of time series divided into two parts: the normal (4,046 observations) and the abnormal (10,506 observations) datasets. Each observation consists of 188 parameters, with 187 signal samples and one value indicating the patient's health condition (0 for normal and 1 for abnormal). All signal samples were normalized in the interval [0,1].

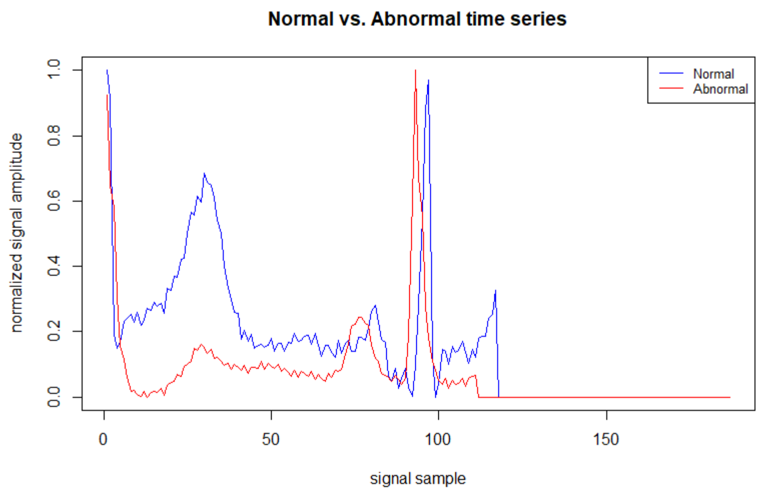


Figure 5.1: Normal vs. Abnormal ECG time series samples

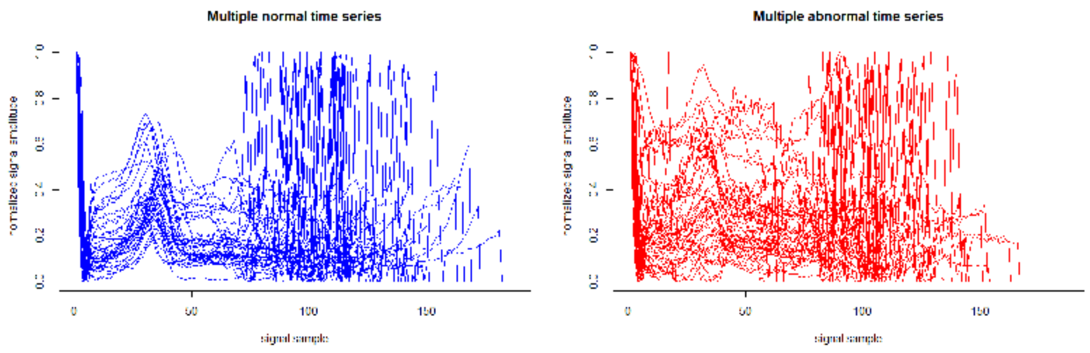


Figure 5.2: Multiple Normal vs. Abnormal ECG time series samples

5.1.1 Normal and abnormal binary ECG time series analysis

Figure 5.1 shows plots of normal and abnormal ECG samples. For a non-expert, either the figure does not reveal much difference between the two signals, or it can lead to incorrect conclusions (for instance, different means, standard deviations, or how wide the peaks of the two signals might be). For better understanding of the data, we have plotted multiple normal signals and abnormal signals, with the results shown in Figure 5.2. Moreover, to better understand the variation of normal and abnormal signals, we plotted multiple

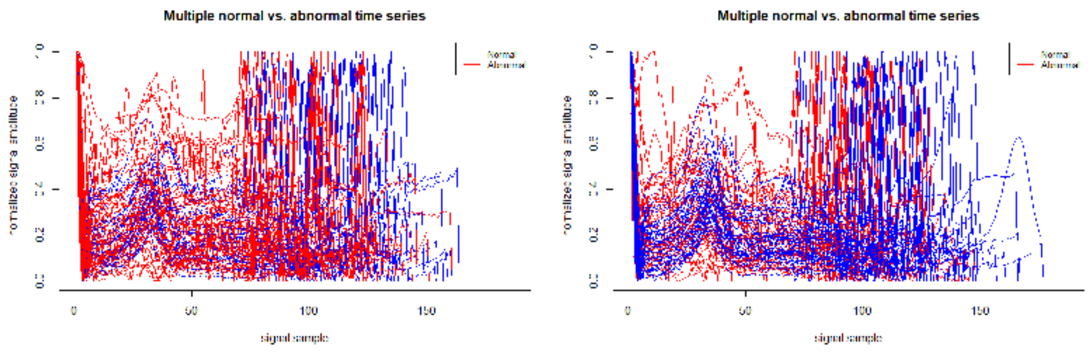


Figure 5.3: Multiple Normal vs. Abnormal ECG time series samples plotted against each-other

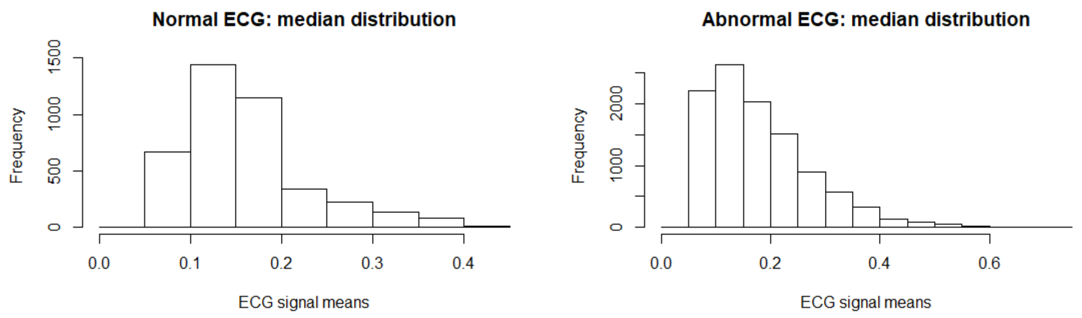


Figure 5.4: Normal vs. Abnormal ECG time series means

samples of each against each-other in Figure 5.3, with abnormal on top (left) and normal on top (right). All these direct comparisons show some differences between the two types of ECGs, but also many similarities. Clearly, on a direct visual inspection, a non-expert would be easily misled and even experts may incorrectly interpret them without additional measurements.

We continued our analysis by computing means, standard deviations, and sums for each signal in the normal and abnormal datasets. With the notations (4.1) established in Chapter 4, we computed the following measures:

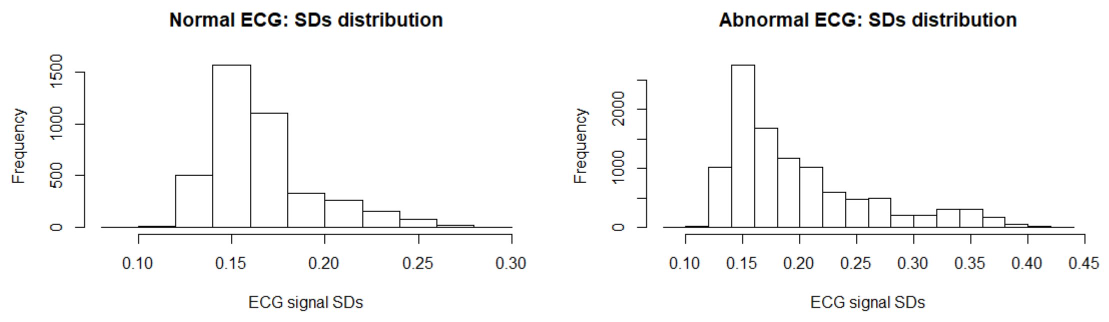


Figure 5.5: Multiple Normal vs. Abnormal ECG time series standard deviations

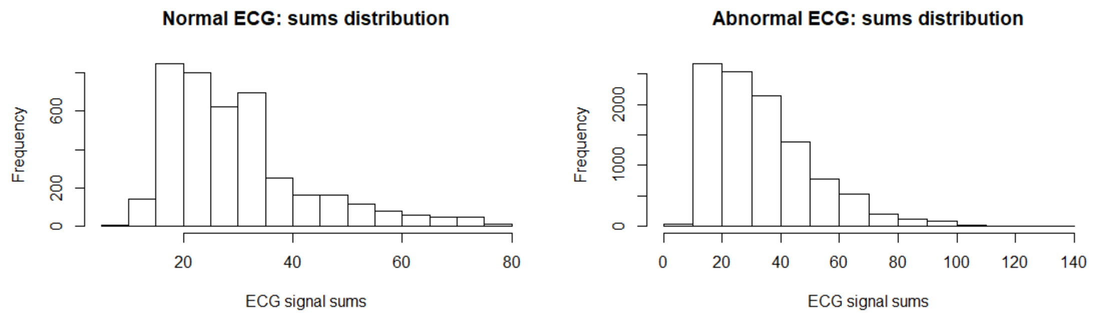


Figure 5.6: Multiple Normal vs. Abnormal ECG time series sums

1. Means:

$$\left\{\frac{1}{187}\sum_{k=1}^{187} n_k^{(i)} \mid n^{(i)} \in \mathcal{N}, i = 1...N, N = size(\mathcal{N})\right\}$$

$$\left\{\frac{1}{187}\sum_{k=1}^{187} a_k^{(i)} \mid a^{(i)} \in \mathcal{A}, i = 1...A, A = size(\mathcal{A})\right\}$$

2. Standard deviations:

$$\left\{SD(n_k^{(i)}) \mid n^{(i)} \in \mathcal{N}, i = 1...N, N = size(\mathcal{N})\right\}$$

$$\left\{SD(a_k^{(i)}) \mid a^{(i)} \in \mathcal{A}, i = 1...A, A = size(\mathcal{A})\right\}$$

3. Sums:

$$\left\{\sum_{k=1}^{187} n_k^{(i)} \mid n^{(i)} \in \mathcal{N}, i = 1...N, N = size(\mathcal{N})\right\}$$

$$\left\{\sum_{k=1}^{187} a_k^{(i)} \mid a^{(i)} \in \mathcal{A}, i = 1...A, A = size(\mathcal{A})\right\}$$

The *means* distribution Figure 5.4 would potentially show different elevations between normal and abnormal ECGs. While the results in Figure 5.4 do show a wider distribution (higher means) for some abnormal data, significant overlapping would not make mean a major disseminator between the two types of signals.

Figure 5.5 shows the *standard deviation* distribution among all signals in $\mathcal{N}$ and $\mathcal{A}$ respectively. The results confirm a wider variation for abnormal signals (as suggested by Figure 5.3). However, there is some significant overlapping (around value 0.15), which would make the standard deviation alone a non-reliable parameter to distinguish between the two types of signals.

Finally, the distributions of *sums* for each signal (in both $\mathcal{N}$ and $\mathcal{A}$) are shown side-by-side in Figure 5.6. A signal *sum* measures the total energy carried by the signal, which would emphasize signals with both higher mean and wider spikes (like the abnormal signals tend to display). The results in Figure 5.6 do exhibit such tendency (wider distribution,

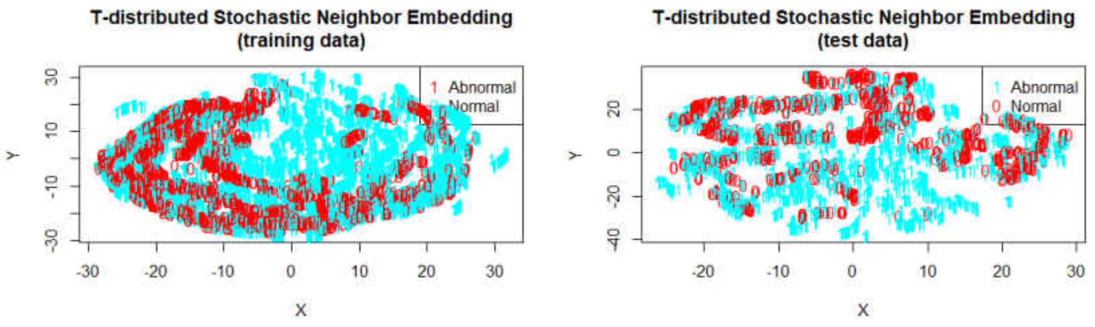


Figure 5.7: T-distributed Stochastic Neighbor Embedding visualization (for training and test PTB data)

larger sums). However, as for the other parameters we measured, it also shows significant overlapping. Consequently, we did not find this measure alone relevant to the classification task we overtake in this study.

An analysis of the ECG signals based on simultaneously considering all measurements we introduced in this section was not found in this study but constituted the subject of possible future work.

5.1.2 Unsupervised analysis of the binary ECG data

In addition to time series signal analysis we used the *t-distributed Stochastic Neighbor Embedding* (t-SNE) technique [19, 18, 9] for the high-dimensional ECG samples in two dimensions. The t-SNE method performs dimensionality reduction (to two or three dimensions) of high dimensional data based on computing feature similarities in the data. The Principal Component Analysis (PCA), a popular dimensionality reduction technique, performs dimensionality reduction by computing co-variance between all data features and subsequently selecting features in decreasing order of variance.

t-SNE computes the probability distribution of similarities between features in the original high dimensional space and then produces a similar two-dimensional (or three-dimensional) distribution of points (minimizes the divergence between the two distributions), which can be subsequently visualized in the respective low-dimensional space. The main advantage of t-SNE over PCA is preserving the local and global structure of data.

We have performed a two-dimensional t-SNE analysis on both training and test datasets, and the results are shown in Figure 5.7. Despite some substantial overlapping (which comes as no surprise given the significant dimensionality reduction), the plots display very distinct clusters of normal and abnormal data. While these results do not directly produce data classification (the main subject of this study), we found them very encouraging. As future work, we believe that unsupervised learning techniques can be employed to explore feature similarities in the original high-dimensional space (or, possibly, dimensionality reduced space) and subsequently separate normal and abnormal data samples. Furthermore, these results clearly show that the ECG dataset is separable into normal and abnormal categories. Motivated by these results, we proceed next to performing binary classification using an ANN model.

5.1.3 ANN binary classification of the binary ECG data

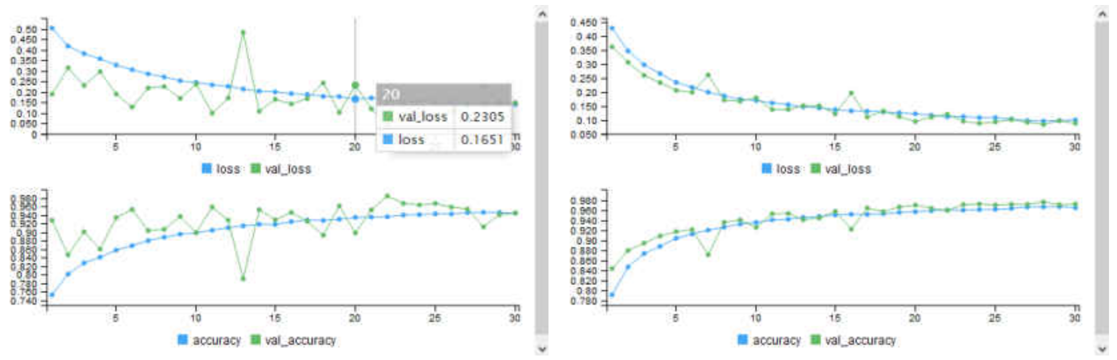


Figure 5.8: The model training (sorted and shuffled training data)

We have performed binary classification using an ANN model as described in Chapter 4 by combining the original PTB normal (4,046 samples of class label 0) and ab-

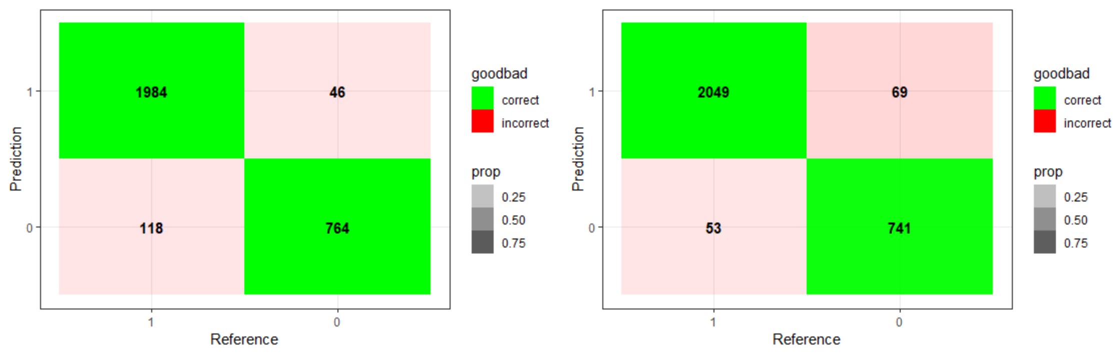


Figure 5.9: Confusion matrix (sorted and shuffled training data)

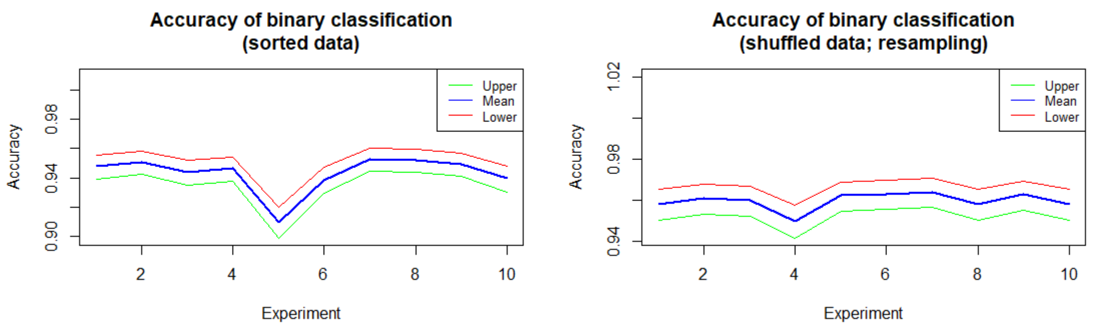


Figure 5.10: Accuracies from 10 trials (sorted and shuffled training data)

normal (10,506 samples of class label 1) datasets and selecting 80% from each category for training and the rest of 20% for testing. Our experiments were grouped into two categories. Firstly, we combined and randomly selected the training data *in the same order as in the original datasets*. The training data was completely sorted in normal (class 0) and abnormal (class 1). We call this category *sorted data*. Secondly, we combined and randomly selected the training data, then shuffled and used the bootstrapping technique (re-sampled the data). We call this category *shuffled data*. In each category of experiments, we ran 10 trials, each time selecting different samples of training data.

Figure 5.8 shows side by side the learning progress for the ANN model. Clearly, not only the shuffled data experiment exhibits faster and smoother learning (see the loss curve), but it also shows a higher accuracy of the model. In Figure 5.9, we present the model testing

Trial	Accuracy	AccuracyLower	AccuracyUpper	AccuracyPValue
1	0.9478022	0.9390909	0.9555990	3.935256e-218
2	0.9508929	0.9424076	0.9584565	7.487406e-226
3	0.9436813	0.9346805	0.9517769	3.255991e-208
4	0.9464286	0.9376193	0.9543265	8.859713e-215
5	0.9093407	0.8983248	0.9195256	3.323265e-139
6	0.9385302	0.9291851	0.9469815	2.218367e-196
7	0.9526099	0.9442539	0.9600404	2.982877e-230
8	0.9519231	0.9435151	0.9594072	1.754821e-228
9	0.9495192	0.9409325	0.9571876	2.171055e-222
10	0.9395604	0.9302827	0.9479421	1.064807e-198

Table 5.1: Accuracy results for sorted data trials

results for the first model in the sequence of 10 trials for each experiment category. Indeed, the shuffled (coupled with the bootstrapping technique) outperforms the sorted data trained model. Moreover, as Figure 5.10 shows, the shuffled data trained models to exhibit a more stable behavior overall 10 trials (in terms of lower and upper bounds for accuracy, and the mean accuracy).

The complete results for lower and upper bound accuracy, mean accuracy, and the corresponding trial p-value are given in Tables 5.1 and 5.2 for the sorted and shuffled data experiments, respectively. The accuracy means and bounds, as well as the p-values of trials in each category of experiments, show high confidence for the model accuracy. It is worth noting that our results were obtained with a relatively simple model (two hidden layers) and using only 30 epochs (steps) for training the model.

Overall, the means and standard deviations for accuracy for 10 trials in each category of experiments are as follows:

Trial	Accuracy	AccuracyLower	AccuracyUpper	AccuracyPValue
1	0.9581044	0.9501814	0.9650891	6.726155e-245
2	0.9608516	0.9531575	0.9676010	1.431672e-252
3	0.9598214	0.9520404	0.9666601	1.152242e-249
4	0.9498626	0.9413011	0.9575050	2.990798e-223
5	0.9622253	0.9546489	0.9688535	1.683261e-256
6	0.9629121	0.9553956	0.9694788	1.725589e-258
7	0.9639423	0.9565167	0.9704156	1.666655e-261
8	0.9581044	0.9501814	0.9650891	6.726155e-245
9	0.9625687	0.9550222	0.9691662	1.712397e-257
10	0.9581044	0.9501814	0.9650891	6.726155e-245

Table 5.2: Accuracy results for shuffled data trials

1. Sorted data: mean accuracy = 0.9430288, standard deviation = 0.01280973

2. Shuffled data: mean accuracy = 0.9596497, standard deviation = 0.004064048

The shuffled data technique produces a more accurate and stable model while preserving the model simplicity (same model structure of two hidden layers). Consequently, our model is more practical. It is also worth mentioning that, on the same ECG data set, our results outperform the ones presented in [8] (an average accuracy of 0.959 vs. 0.934), while using a simpler ANN model versus a more complex Convolutional Neural Network (CNN) model.

5.2 Detecting heart diseases using multiclass classification

The MIT-BIH dataset consists of 87,554 samples of ECG signals in a training dataset and 21,892 samples in a test dataset. We have used the training and test data partitioning as provided in the original dataset. This dataset, however, contains 5 types of signals: normal (label 0), and 4 categories of heart diseases (as described in Chapter 2, labeled as 1, 2, 3, and 4). Consequently, we have created a multiclass classification model (for 5 classes), which we trained using the provided training dataset and subsequently tested using the provided testing dataset.

As for the binary dataset scenario, we have organized our experiments as follows: analysis of multiclass ECG data samples, unsupervised learning, and ANN model multiclass classification. The results of our experiments are presented in the subsequent subsections.

5.2.1 Normal and abnormal multiclass ECG time series analysis

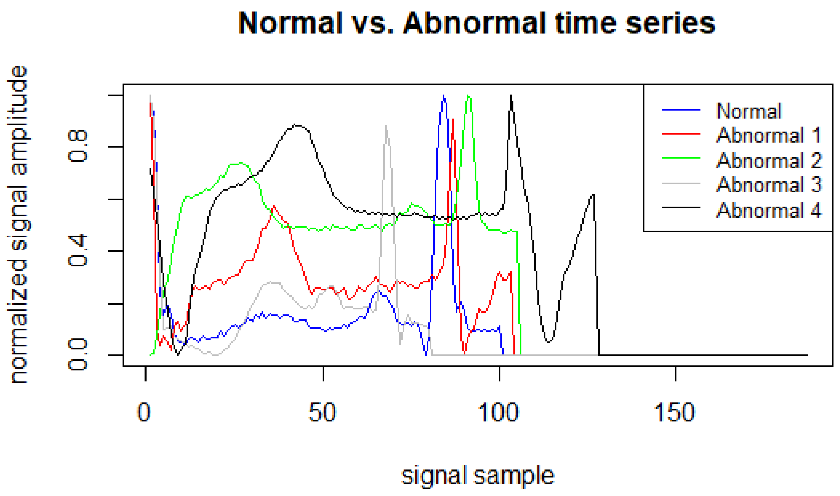


Figure 5.11: Normal vs. Abnormal multiclass ECG signals

As in the case of the binary PTB data we started our MIT-BIH ECG signal analysis by showing all signal types (normal of type 0 and four abnormal of types 1, 2, 3, and 4, respectively) as illustrated in Figure 5.11. These sample signals show clear differences between the ECG normal signal and most of the abnormal ECG types. At the first glance, the normal type display lower mean and variance than its abnormal counterparts (abnormal types have wider spikes).

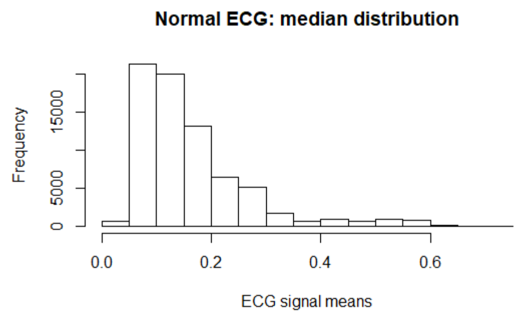


Figure 5.12: Multiclass ECG means (normal type 0)

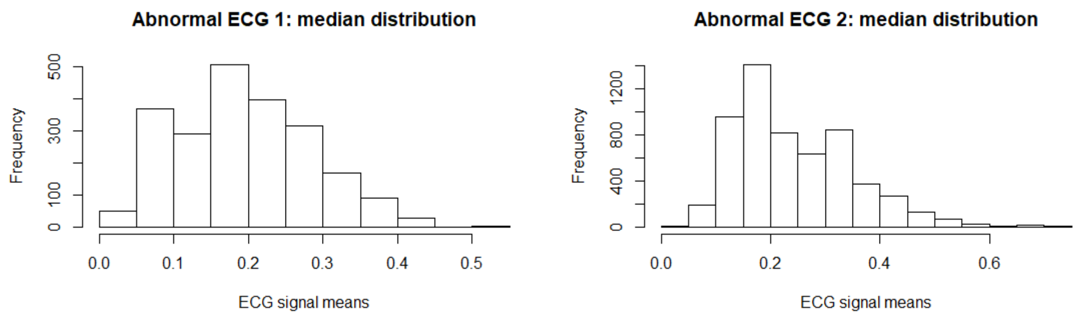


Figure 5.13: Multiclass ECG means (abnormal types 1 and 2)

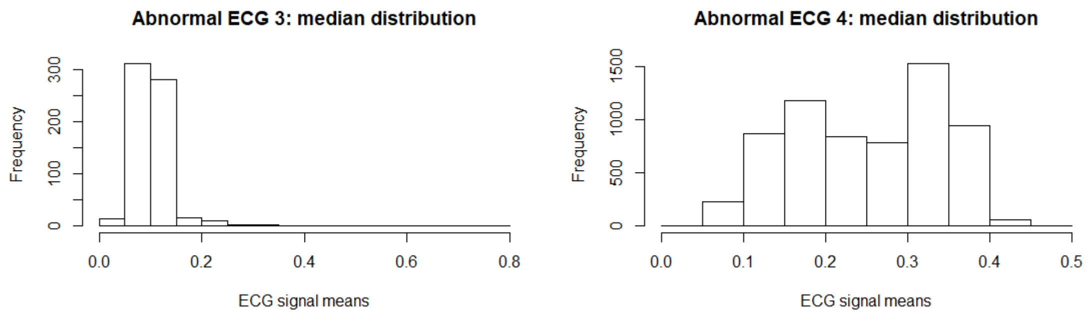


Figure 5.14: Multiclass ECG means (abnormal types 3 and 4)

As before, we computed the following measures (with the notations (4.2) established in Chapter 4):

1. Means:

$$\left\{\frac{1}{187}\sum_{k=1}^{187} n_k^{(i)} \mid n^{(i)} \in \mathcal{N}, i = 1...N, N = size(\mathcal{N})\right\}$$

$$\left\{\frac{1}{187}\sum_{k=1}^{187} a_k^{(i)} \mid a^{(i)} \in \mathcal{A}_j, i = 1...A_j, A_j = size(\mathcal{A}_j)\right\}, j = 1,2,3,4$$

2. Standard deviations:

$$\left\{SD(n_k^{(i)}) \mid n^{(i)} \in \mathcal{N}, i = 1...N, N = size(\mathcal{N})\right\}$$

$$\left\{SD(a_k^{(i)}) \mid a^{(i)} \in \mathcal{A}_j, i = 1...A_j, A_j = size(\mathcal{A}_j)\right\}, j = 1,2,3,4$$

3. Sums:

$$\left\{\sum_{k=1}^{187} n_k^{(i)} \mid n^{(i)} \in \mathcal{N}, i = 1...N, N = size(\mathcal{N})\right\}$$

$$\left\{\sum_{k=1}^{187} a_k^{(i)} \mid a^{(i)} \in \mathcal{A}_j, i = 1...A_j, A_j = size(\mathcal{A}_|)\right\}, j = 1,2,3,4$$

The corresponding means are shown in Figures 5.12, 5.13, and 5.14. Indeed, the normal type 0 shows lower means than all abnormal types but type 3. As in the binary data case scenario, means, by itself is not a sole criteria for disseminating the data types but can be an important feature to be considered (together with other features) for constructing a more simple multiclass classification model. We leave exploring this idea for future work.

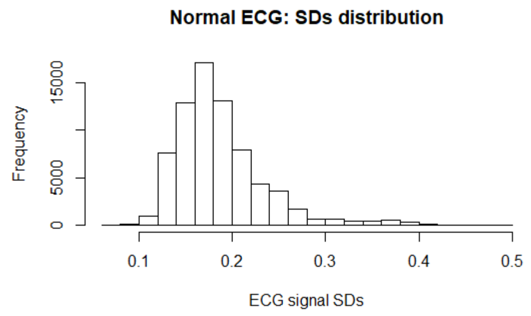


Figure 5.15: Multiclass ECG standard deviations (normal type 0)

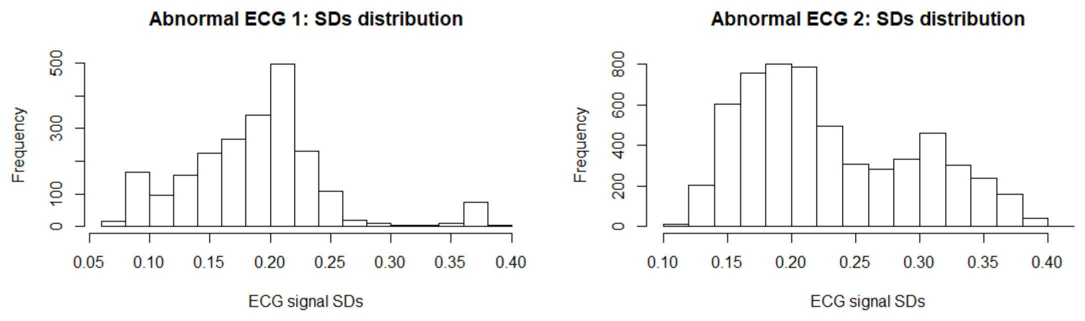


Figure 5.16: Multiclass ECG standard deviations (abnormal types 1 and 2)

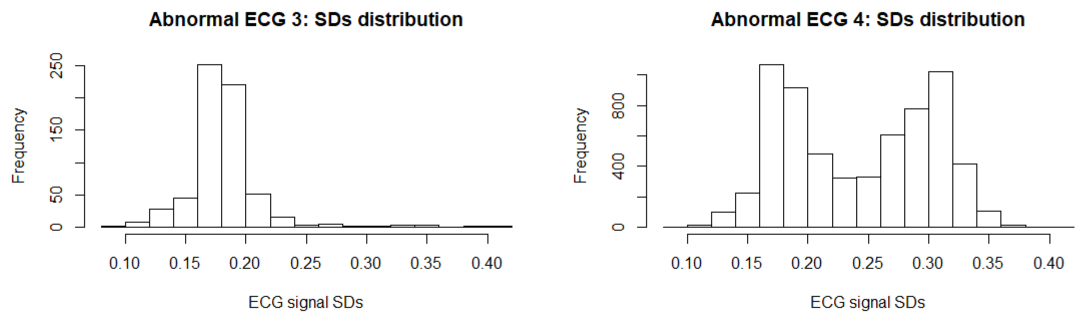


Figure 5.17: Multiclass ECG standard deviations (abnormal types 3 and 4)

The standard deviations for the normal ECG signal types and abnormal ones are shown in Figures 5.15, 5.16, and 5.17, respectively. They confirm the simple conclusion from Figure 5.11 that the normal type signal have smaller variations compared to all abnormal types but type 3.

Finally, Figures 5.18, 5.19, and 5.20 show the ECG signals sums for normal type and abnormal types, respectively. The results are consistent with the previous measurements: the normal type signals exhibit less variation for sums than all abnormal types but the abnormal type 3.

As these simple direct observations on ECG signal types from the multiclass MIT-BIH dataset consistently show significant differences between the normal type 0 and abnormal types 1, 2, and 4, but not much difference from abnormal type 3, it would make sense to

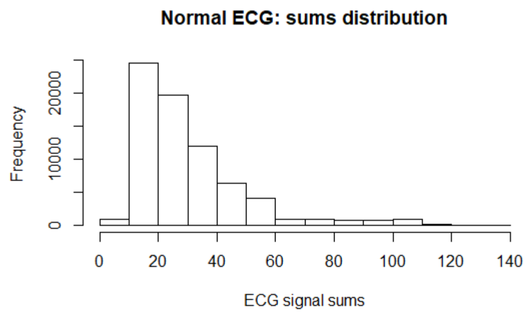


Figure 5.18: Multiclass ECG sums (normal type 0)

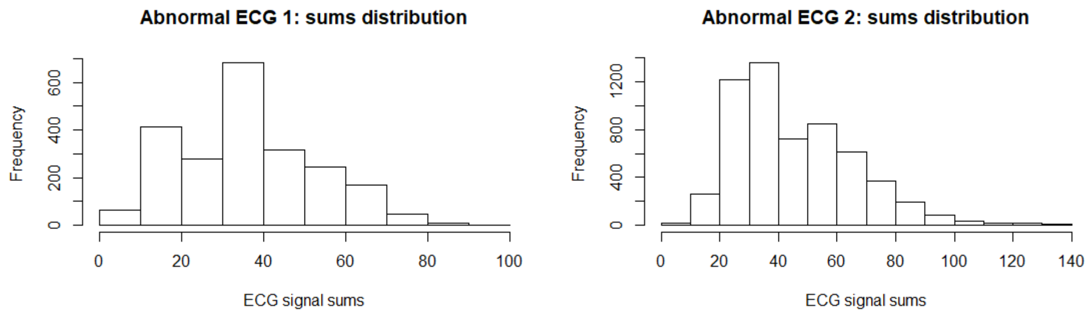


Figure 5.19: Multiclass ECG sums (abnormal types 1 and 2)

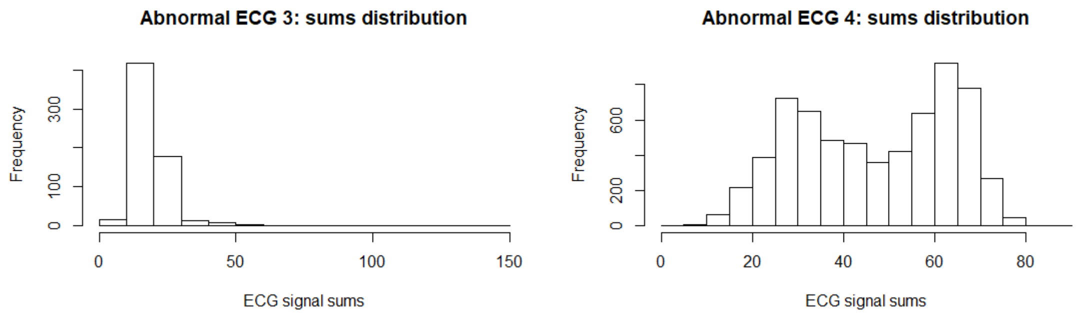


Figure 5.20: Multiclass ECG sums (abnormal types 3 and 4)

analyze our ANN multiclass classifier performance on identifying abnormal type 3.

The signal type analysis results presented in this section are consistent with the analysis of the binary ECG data presented before. It shows that different ECG signal types possess distinguishable features that can possibly be used for creating simplified multiclass classification models.

5.2.2 Unsupervised analysis of the multiclass ECG data

Like in the binary case scenario (for PTB data), we have used the t-SNE technique to visualize the high-dimensional MIT-BIH ECG data in two dimensions. The results are presented in Figure 5.21: for the test data (left) and training data (right).

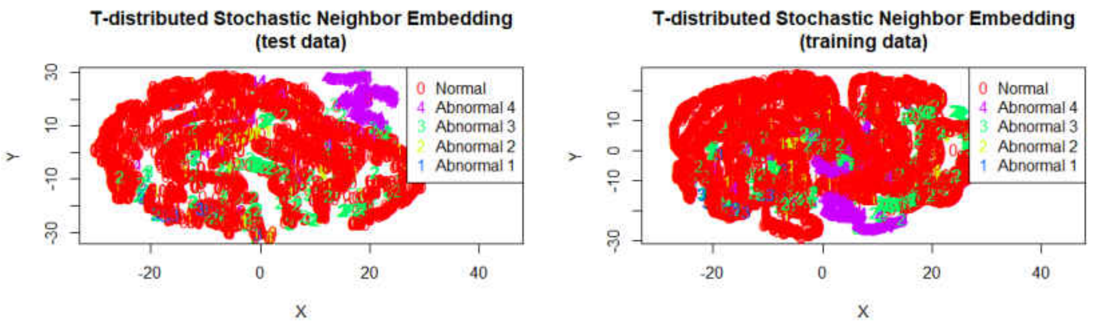


Figure 5.21: T-distributed Stochastic Neighbor Embedding (t-SNE) visualization (for the test and training MIT-BIH data)

We estimate that the results are very encouraging for performing data separation in classes. They show clusters of data of various types (with the majority of data being of type 0, normal), which is a good indicator that data of the same type have similar features. As before, the method performs data classification directly, but it would be the right direction to investigate future work.

With these results, we proceed next to performing multiclass classification on MIT-BIH data using an ANN model.

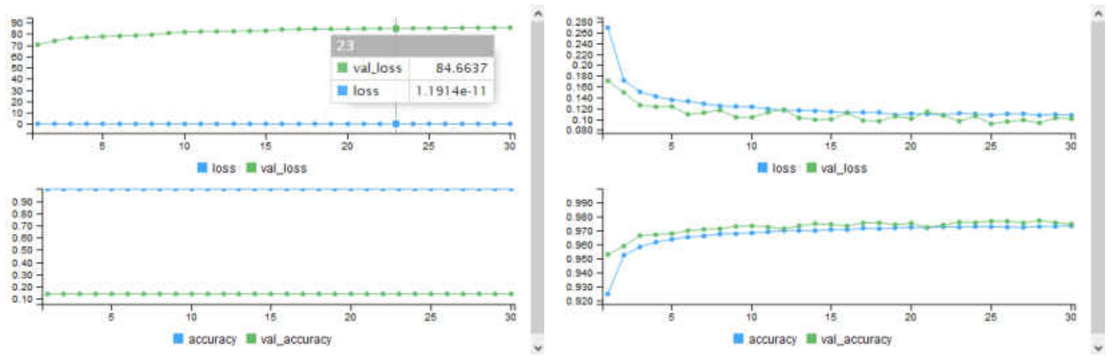


Figure 5.22: Multiclass ECG classification model training (sorted and shuffled data)

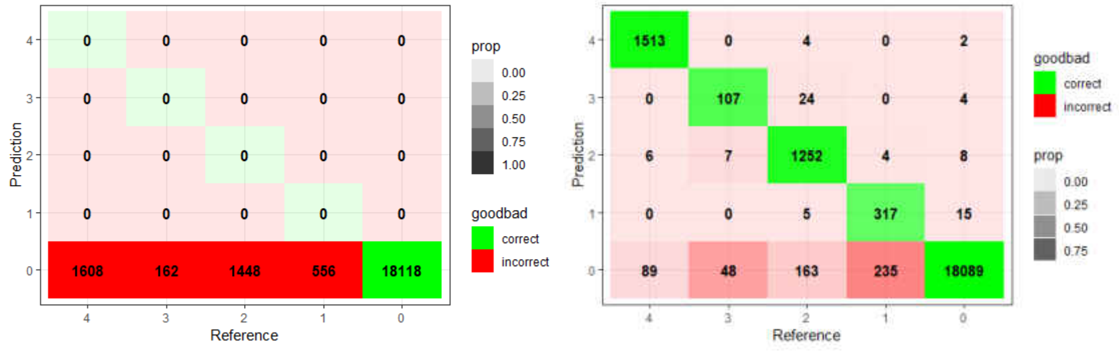


Figure 5.23: Multiclass ECG classification confusion matrix (sorted and shuffled data)

5.2.3 ANN multiclass classification of the ECG data

As for the binary classification experiment, we proceeded on the multiclass classification of MIT-BIH data by training an ANN model on the provided training data as given. It turned out that the 87,554 observations in the training data set were completed sorted as 72,471 samples of normal type 0, 2,223 samples of abnormal type 1, 5,788 samples of abnormal type 2, 641 samples of abnormal type 3, and 6,431 data samples of abnormal type 4. On such sorted, highly unbalanced data training, the model was a total failure: the training data was quickly overfitted (Figure 5.22, left).

While the performance on the training data might be misleading (with accuracy quickly converging to 100%), the subsequent model testing clearly shows no confidence on the

Accuracy	AccuracyLower	AccuracyUpper	AccuracyPValue
0.8276083	0.8225385	0.8325909	0.5043485

Table 5.3: Accuracy results for the test data on sorted data

Accuracy	AccuracyLower	AccuracyUpper	AccuracyPValue
0.9719532	0.9696796	0.9741003	0.0000000

Table 5.4: Accuracy results for the test data on shuffled data

testing data accuracy of 0.82 (the p-value of 0.5 in Table 5.3) and the confusion matrix (Figure 5.23, left) shows clearly how the test data was completely misclassified.

We subsequently shuffled the provided MIT-BIH data before training the same model. Figure 5.22 (right) shows clearly the normality of the training process, with the training accuracy slowly converging to the final value in the high 90s. The subsequent model testing on the provided MIT-BIH test data shows a mastery of 96.9% with high confidence (as indicated by the p-value ≈ 0 in Table 5.4). The corresponding confusion matrix for the test data classification (Figure 5.23, right) confirms the highly accurate multiclass classification.

It is worth mentioning that the ANN multiclass classifier exhibits behavior consistent with the results we observed in our simple analysis carried out in Section 5.2.1. Namely, the classifier performs poorly on identifying abnormal types 3, which we found having very similar features to normal types 0. Figure 5.23 shows confusion between abnormal type 3 and normal type 0, indeed.

CHAPTER 6
CONCLUSION AND FUTURE WORK

In this work, we introduce Artificial Neural Network (ANN) based models for assisting experts in the medical field in assessing various heart diseases based on automatically analyzing patients' electrocardiogram (ECG) data. We have addressed two case scenario; (i) binary decision models, where the ECG data of patients are analyzed, and patients are categorized as normal or abnormal; (ii) multi-decision models, where the ECG data is automatically analyzed and a diagnostic of normal or 4 possible heart anomalies (arrhythmia, myocardial infarct, etc.) is produced. Our models are mainly characterized by their simplicity, as they do not require significant computational power to deliver the results.

We have implemented and tested our models on real patient data from the PTB Diagnostic ECG Database [1, 6] (for the binary model), and the MIT-BIH Arrhythmia Database [14] (for the multi-decision model). Our model proved to be very reliable, with average accuracies of 95.96% and 97.19%, respectively. These results are superior to previous work results on the same datasets [8].

An exciting idea raised from our ECG signal study was to perform an analysis based on simple measures introduced in Chapter 5 (means, standard deviations, and sums) by considering all of them at the same time (rather than one at the time). A classification model based on these signal features not only will be significantly simplified but also independent of the ECG signals space dimension. That is, the requirement to have precisely 187 samples for each ECG signal can be dropped. We believe such an analysis would greatly simplify the classification model and would be of great practical interest.

Our unsupervised learning analysis produced another interesting conclusion. Namely, the spaces of normal and abnormal ECG signals are separable using feature similarity analysis. As we focused on the ANN model classification in this study, we only conjecture that unsupervised analysis can produce significant practical results, and we leave it for possible future work.

Finally, another interesting analysis would be knowledge transfer from one dataset to another: can one train a model with data from one dataset and perform accurate classification of data from the other set? We leave this investigation for future work as well.

REFERENCES

[1] R. Bousseljot, D. Kreiseler, and A. Schnabel, *Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet*, Biomedical Engineering **40** (1995), no. s1, 317–318.

[2] François Chollet et al., *Keras*, https://keras.io, 2015.

[3] C. J. Clopper and E. S. Pearson, *The use of confidence or fiducial limits illustrated in the case of the binomial*, Biometrika **26** (1934), no. 4, 404–413.

[4] Research Gate, *Schematic representation of normal ECG waveform*, https://research gate.net/, March 2020.

[5] Geekymedic, *How to Read an ECG*, https://geekymedics.com/how-to-read-an-ecg, February 2020.

[6] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley, *PhysioBank, PhysioToolkit, and PhysioNet*, Circulation **101** (2000), no. 23, e215–e220.

[7] Imotions, *Heart Rate Variability*, https://imotions.com/blog/heart-rate-variability/, February 2020.

[8] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh, *ECG Heartbeat Classification: A Deep Transferable Representation*, (2018).

[9] Jesse H. Krijthe, *Rtsne: T-Distributed Stochastic Neighbor Embedding using Barnes-Hut Implementation*, 2015, R package version 0.15.

[10] Carlos Lastre, Yuriy Shmaliy, Oscar Ibarra-Manzano, J. Munoz-Minjares, and Luis Morales-Mendoza, *Ecg signal denoising and features extraction using unbiased fir smoothing*, BioMed Research International **2019** (2019), 1–16.

[11] LearnOpenCV, *Understanding activation functions in deep learning*, http://learnopencv.com, February 2020.

[12] Quinn McNemar, *Note on the sampling error of the difference between correlated proportions or percentages*, Psychometrika **12** (1947), no. 2, 153–157.

[13] Mohamed Hammad and Asmaa Maher and Kuanquan Wang and Feng Jiang and Moussa Amrani, *Detection of abnormal heart conditions based on characteristics of ECG signals*, Measurement **125** (2018), 634 – 644.

[14] G. B. Moody and R. G. Mark, *The impact of the mit-bih arrhythmia database*, IEEE Engineering in Medicine and Biology Magazine **20** (2001), no. 3, 45–50.

[15] Physionet, *Data Description MIT*, https://doi.org/10.13026/C2F305, February 2020.

[16] ______, *Data Description PTB*, https://physionet.org/content/ptbdb/1.0.0/, February 2020.

[17] V. Seena and J. Yomas, *A review on feature extraction and denoising of ecg signal using wavelet transform*, (2014), 1–6.

[18] L.J.P. van der Maaten, *Accelerating t-SNE using Tree-Based Algorithms*, Journal of Machine Learning Research **15** (2014), 3221–3245.

[19] L.J.P. van der Maaten and G.E. Hinton, *Visualizing High-Dimensional Data Using t-SNE*, Journal of Machine Learning Research **9** (2008), 2579–2605.

[20] Wikihow, *How to read an EKG*, https://www.wikihow.com/Read-an-EKG, February 2020.

[21] Wikimedia, *Electrocardiography*, http://commons.wikimedia.org, February 2020.

[22] Ali Yasoubi, Reza Hojabr, and Mehdi Modarressi, *Power-efficient accelerator design for neural networks using computation reuse*, IEEE Computer Architecture Letters **16** (2016), 1–1.

APPENDIX A
THE COMPLETE R CODE FOR THE EXPERIMENTAL RESULTS

As described in Chapter 5 we have run two sets of experiments for each data set. The first two experiments (entitled below as Experiment 1 and Experiment 2) used the PTB Diagnostic ECG Database [1, 6] and trained an ANN model with (i) data as provided (in Experiment 1) and (ii) shuffled and re-sampled data (in Experiment 2). Similarly, the Experiment 3 and Experiment 4 were performed on the MIT-BIH Arrhythmia Database [14], with (i) data as provided and (ii) shuffled and re-sampled data, respectively. We present the complete R code of these experiments in the subsequent sections of this Appendix.

A.1 Experiment 1

```
###############################################################
#
# Classification of ECG data using ANN
#
# Version 1.0: un-shuffled training data; poor results
#
###############################################################

library(keras)
#install_keras() #only first time

#data folder (relative to current)
srcdir <- "../data/"

#data files
```

```
srcfile1 <- "mitbih_test.csv"
srcfile2 <- "mitbih_train.csv"
#srcfile3 <- "ptbdb_abnormal.csv"
#srcfile4 <- "ptbdb_normal.csv"

#read data
srcfile <- srcfile2
traindata <- read.csv(paste0(srcdir, srcfile),header = F)
srcfile <- srcfile1
testdata <- read.csv(paste0(srcdir, srcfile),header = F)

#some info from training data
unique(traindata[,188])
sum(traindata[,188] == 0)
sum(traindata[,188] == 1)
sum(traindata[,188] == 2)
sum(traindata[,188] == 3)
sum(traindata[,188] == 4)


#retrieve inputs/outputs from data
x_train <- as.matrix(traindata[,1:187])
y_train <- to_categorical(traindata[,188], 5)


#Test: retrieve inputs/outputs from data
```

```
x_test <- as.matrix(testdata[,1:187])
y_test <- to_categorical(testdata[,188], 5)

#some useful plots
colors <- c("blue", 'red', 'green', 'grey', 'black')
#1. Normal vs. Abnormal time series; exp2res1
yn <- traindata[traindata[,188] == 0,-188][1,]
ya1 <- traindata[traindata[,188] == 1,-188][1,]
ya2 <- traindata[traindata[,188] == 2,-188][1,]
ya3 <- traindata[traindata[,188] == 3,-188][1,]
ya4 <- traindata[traindata[,188] == 4,-188][1,]
plot(1:length(yn), yn, col = "blue", type = 'l',
     main = "Normal vs. Abnormal time series",
     xlab = "signal sample",
     ylab = "normalized signal amplitude")
lines(1:length(ya1), ya1, col = colors[2])
lines(1:length(ya2), ya2, col = colors[3])
lines(1:length(ya3), ya3, col = colors[4])
lines(1:length(ya4), ya4, col = colors[5])
legend("topright", legend=c("Normal", "Abnormal 1",
              "Abnormal 2", "Abnormal 3", "Abnormal 4"),
       col=colors, lty=1, cex=0.8)


#2, 3, 4, 5, 6 Median distributions for normal and abnormal
hist(rowMeans(traindata[traindata[,188] == 0,-188]),
     xlab = "ECG signal means",
     main = "Normal ECG: median distribution")
hist(rowMeans(traindata[traindata[,188] == 1,-188]),
     xlab = "ECG signal means",
     main = "Abnormal ECG 1: median distribution")
hist(rowMeans(traindata[traindata[,188] == 2,-188]),
     xlab = "ECG signal means",
     main = "Abnormal ECG 2: median distribution")
hist(rowMeans(traindata[traindata[,188] == 3,-188]),
     xlab = "ECG signal means",
     main = "Abnormal ECG 3: median distribution")
hist(rowMeans(traindata[traindata[,188] == 4,-188]),
     xlab = "ECG signal means",
     main = "Abnormal ECG 4: median distribution")


#7, 8, 9, 10, 11 SD distributions for normal and abnormal
hist(rowSds(as.matrix(traindata[traindata[,188] == 0,-188])),
     xlab = "ECG signal SDs",
     main = "Normal ECG: SDs distribution")
hist(rowSds(as.matrix(traindata[traindata[,188] == 1,-188])),
     xlab = "ECG signal SDs",
     main = "Abnormal ECG 1: SDs distribution")
hist(rowSds(as.matrix(traindata[traindata[,188] == 2,-188])),
     xlab = "ECG signal SDs",
     main = "Abnormal ECG 2: SDs distribution")
```

```
hist(rowSds(as.matrix(traindata[traindata[,188] == 3,-188])),
     xlab = "ECG signal SDs",
     main = "Abnormal ECG 3: SDs distribution")
hist(rowSds(as.matrix(traindata[traindata[,188] == 4,-188])),
     xlab = "ECG signal SDs",
     main = "Abnormal ECG 4: SDs distribution")


#12, 13, 14, 15, 16 Sums distributions for normal and abnormal
hist(rowSums(traindata[traindata[,188] == 0,-188]),
     xlab = "ECG signal sums",
     main = "Normal ECG: sums distribution")
hist(rowSums(traindata[traindata[,188] == 1,-188]),
     xlab = "ECG signal sums",
     main = "Abnormal ECG 1: sums distribution")
hist(rowSums(traindata[traindata[,188] == 2,-188]),
     xlab = "ECG signal sums",
     main = "Abnormal ECG 2: sums distribution")
hist(rowSums(traindata[traindata[,188] == 3,-188]),
     xlab = "ECG signal sums",
     main = "Abnormal ECG 3: sums distribution")
hist(rowSums(traindata[traindata[,188] == 4,-188]),
     xlab = "ECG signal sums",
     main = "Abnormal ECG 4: sums distribution")
```

```
#create the model
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = c(187)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 5, activation = 'softmax')
#model %>%
# layer_dense(units = 64, activation = 'relu',
input_shape = c(187)) %>%
  # layer_dropout(rate = 0.4) %>%
  # layer_dense(units = 32, activation = 'relu') %>%
  # layer_dropout(rate = 0.3) %>%
  # layer_dense(units = 5, activation = 'softmax')



  #print some model info (if wanted)
  summary(model)


#assemble the model
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)



#train (exp2res17)
model.info <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)

#plot the model info
#plot(model.info)
model %>% evaluate(x_test, y_test)

#test on test data
test.results <- model %>% predict_classes(x_test)

#####################
#confusion matrix (exp2res18)
library('caret')
cm <- confusionMatrix(factor(test.results),
      factor(testdata[,188]))
cm$overall
```

```
table <- data.frame(cm$table)



library(ggplot2)
library(dplyr)

#table <- data.frame(confusionMatrix(pred, truth)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse
         (table$Prediction == table$Reference,
          "correct", "incorrect")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))
#confusion matrix, expres14
#fill alpha relative to sensitivity/specificity by
#proportional outcomes within reference groups (see
#dplyr code above as well as original confusion matrix
#for comparison)

ggplot(data = plotTable, mapping = aes(x = Reference,
                                       y = Prediction,
                                       fill = goodbad,
                                       alpha = prop)) +
  geom_tile() + geom_text(aes(label = Freq),
  vjust = .5, fontface  = "bold", alpha = 1) +
  scale_fill_manual(values = c(correct = "green",
  incorrect = "red"))+
  theme_bw() +
  xlim(rev(levels(table$Reference)))



##############################################
#some nice plots with t-SNE (exp2res19, 20)
library(Rtsne)

per <- .25
N <- nrow(testdata)
tsnedata <- testdata[sample(N, as.integer(per * N)),]

Labels<-tsnedata[,188]
tsnedata[,188]<-as.factor(tsnedata[,188])
## for plotting
colors = rainbow(length(unique(tsnedata[,188])))
names(colors) = unique(tsnedata[,188])

## Executing the algorithm on curated data
tsne <- Rtsne(tsnedata[,-188], dims = 2, perplexity=30,
        verbose=TRUE, max_iter = 500)
```

```
#exeTimeTsne<- system.time(Rtsne(tsnedata[,-188],dims = 2,
perplexity=30, verbose=TRUE, max_iter = 500))

## Plotting
plot(tsne$Y, t='n', xlab = 'X', ylab = 'Y', xlim = c(-30, 45),
main="T-distributed Stochastic Neighbor Embedding\n(test
                                                    data)")
text(tsne$Y, labels=tsnedata[,188],
                            col=colors[tsnedata[,188]])
legend("topright", legend=c("Normal", "Abnormal 4",
              "Abnormal 3", "Abnormal 2", "Abnormal 1"),
       col=colors[unique(tsnedata[,188])],
       pch = c("0", "4", "3", "2", "1"))



per <- .15
N <- nrow(traindata)
tsnedata <- traindata[sample(N, as.integer(per * N)),]

Labels<-tsnedata[,188]
tsnedata[,188]<-as.factor(tsnedata[,188])
## for plotting
colors = rainbow(length(unique(tsnedata[,188])))
names(colors) = unique(tsnedata[,188])

## Executing the algorithm on curated data
```

```
tsne <- Rtsne(tsnedata[,-188], dims = 2, perplexity=30,
              verbose=TRUE, max_iter = 500)
#exeTimeTsne<- system.time(Rtsne(tsnedata[,-188], dims = 2,
perplexity=30, verbose=TRUE, max_iter = 500))

## Plotting
plot(tsne$Y, t='n', xlab = 'X', ylab = 'Y',
     xlim = c(-30, 45),
     main="T-distributed Stochastic Neighbor Embedding\n
                                         (training data)")
text(tsne$Y, labels=tsnedata[,188],
        col=colors[tsnedata[,188]])
legend("topright", legend=c("Normal", "Abnormal 4",
            "Abnormal 3", "Abnormal 2", "Abnormal 1"),
       col=colors[unique(tsnedata[,188])],
       pch = c("0", "4", "3", "2", "1"))
```

A.2 Experiment 2

```
############################################################
#
# Classification of EEG data using ANN
#
# Version 0.2: shuffled training data, multi-class
# classification, confusion matrix
```

```
#
##############################################################
library(keras)
library(signal)
library('seewave')
library('tuneR')
library('spectral')

sensors <- c("AF3", "F7", "F3", "FC5", "T7", "P7", "O1",
              "O2","P8", "T8", "FC6", "F4", "F8", "AF4")
df <- read.csv("../data/AllNeuroMarketingEEGs.csv",
      header = T)
#this holds the whole data
#extract the list of subjects
subjects <- unique(df[,3])#this is the list of 25 subjects

#select data for model
sensor <- sensors[3]       #this selects the sensor from list
                              above
subject <- subjects[3]      #this selects the user

data <- df[(df[,4]==sensor),5:ncol(df)]
#from the whole data, retain sensor and subject
data <- data[sample(nrow(data)), ]#shuffle
```

```
####################### FFT ##########################

#IFT
# returns the x.n time series for a given time sequence (ts)
# and a vector with the amount of frequencies k in the
# signal (X.k)
get.trajectory <- function(X.k, minh = 1, maxh = -1) {
  N   <- length(X.k)
  NH <- ifelse(maxh > 0, min(maxh, N), N)
  if (minh < 1 | minh > maxh) {
    minh <- 1
  }
  X.k <- X.k[minh:NH]
  i   <- complex(real = 0, imaginary = 1)
  x.n <- rep(0,N)      #create vector to keep the trajectory
  ks  <- 0:(length(X.k)-1)

  for(n in 0:(N-1)) {  # compute each time point x_n based on
  freqs X.k  x.n[n+1] <- sum(X.k * exp(i*2*pi*ks*n/N)) / N
  }

  return (x.n)
}
#plot FFT spectrum
plot.frequency.spectrum <- function
     (X.k,xlimits=c(0,length(X.k)),main = '') {
```

```
  plot.data  <- cbind(0:(length(X.k)-1), Mod(X.k))

  # TODO: why this scaling is necessary?
plot.data[2:length(X.k),2] <- 2*plot.data[2:length(X.k),2]

plot(plot.data, t="h", lwd=2, main=main,
       xlab="Frequency (Hz)", ylab="Strength",
       xlim=xlimits, ylim=c(0,max(Mod(plot.data[,2]))))
}



set.seed(101)

#extract a like vs. a dislike signal
#plot a like vs. a dislike signal
yl <- data[data[,1]==T,-1]
yd <- data[data[,1]==F,-1]
x <- 1:length(y)
plot(x, yd[2,], col = "red", type = 'l',
main = "Like vs. dislike signals", xlab = "signal sample",
     ylab = "signal amplitude",
     ylim = c(min(yd[2,],yl[1,]),max(yd[2,],yl[1,])))
lines(x, yl[1,], col = 'green')

legend("topright", legend=c("Like", "Dislike"),
       col=c("green", "red"), lty=1, cex=0.8)
```

```
#perform FFT
acq.freq <- 1000
ts <- x
trajectory <- unlist(yl[1,])

X.k <- fft(trajectory)
plot.frequency.spectrum(X.k,xlimits=c(0,acq.freq))

#restore the original
maxhar <- 15 #max harmonics (-1 = all)
minhar <- 10
x.n <- get.trajectory(X.k, minh = minhar, max = maxhar)
plot(ts,Mod(x.n), type="l")
points(ts,trajectory,col="red",type="l")
#compare with original
#points(ts,Re(x.n),col="blue",type="l")

#extract and compare components from like(green)/dislike(red)
acq.freq <- 1000
ts <- x
trajectoryl <- unlist(yl[1,])
trajectoryd <- unlist(yd[1,])

par(mfrow=c(2,1))
Xl.k <- fft(trajectoryl)
```

```
plot.frequency.spectrum(Xl.k,xlimits=c(0,acq.freq),
                        main = 'Specturm LIKE')
Xd.k <- fft(trajectoryd)
plot.frequency.spectrum(Xd.k,xlimits=c(0,acq.freq),
                        main = 'Spectrum DISLIKE')

#restore the original
minhar <- 1
maxhar <- 4 #max harmonics (-1 = all)

xl.n <- get.trajectory(Xl.k, minh = minhar, max = maxhar)
plot(ts,Mod(xl.n), type="l", main = 'Like')
xd.n <- get.trajectory(Xd.k, minh = minhar, max = maxhar)
plot(ts,Mod(xd.n), type="l", main = 'DisLike')






#normalize then perform FFT
yl <- yl / max(yl)
trajectory <- unlist(yl[1,])

X.k <- fft(trajectory)
plot.frequency.spectrum(X.k,xlimits=c(0,acq.freq))
```

```
#restore the original
maxhar <- 1 #max harmonics (-1 = all)
minhar <- 1
x.n <- get.trajectory(X.k, minh = minhar, max = maxhar)
plot(ts,Mod(x.n), type="l", ylim = c(0,1))
points(ts,trajectory,col="red",type="l")
# compare with original
#points(ts,Re(x.n),col="blue",type="l")




#################### end FFT ######################



#smooth, nonlinear
#lfdata <- data
#for (r in 1:nrow(data)) {
# lowpass.spline <- smooth.spline(x,data[r,-1], spar = 0.1)
## Control spar for amount of smoothing
#  ys <- predict(lowpass.spline, x)
#  lfdata[r,-1] <- sin(as.numeric(ys$y))
#}
```

```
##################################### ANN #################
TRAINING <- sample(0.8*nrow(data))
#normal data sample for training
traindata <- data[TRAINING,]
testdata <- data[-TRAINING,]

#some info from training data
unique(traindata[,SLENGTH])
sum(traindata[,SLENGTH] == 0)
sum(traindata[,SLENGTH] == 1)


#retrieve inputs/outputs from data
x_train <- as.matrix(traindata[,1:(SLENGTH-1)])
y_train <- to_categorical(traindata[,SLENGTH], 2)


#Test: retrieve inputs/outputs from data
x_test <- as.matrix(testdata[,1:(SLENGTH-1)])
y_test <- to_categorical(testdata[,SLENGTH], 2)
```

```
#create the model
model <- keras_model_sequential()
# model %>%
#  layer_dense(units = 256, activation = 'tanh',
input_shape = c(SLENGTH-1)) %>%
  #  layer_dense(units = 128, activation = 'tanh') %>%
  #  layer_dense(units = 2, activation = 'softmax')

  model %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = c(SLENGTH-1)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = 'softmax')
#model %>%
# layer_dense(units = 64, activation = 'relu',
input_shape = c(SLENGTH-1)) %>%
  # layer_dropout(rate = 0.4) %>%
  # layer_dense(units = 32, activation = 'relu') %>%
  # layer_dropout(rate = 0.3) %>%
  # layer_dense(units = 2, activation = 'softmax')



  #print some model info (if wanted)
  summary(model)



#assemble the model
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)



#train
model.info <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.25
)

#plot the model info
plot(model.info)
model %>% evaluate(x_test, y_test)

#test on test data
test.results <- model %>% predict_classes(x_test)
#create and show the confusion matrix
library('caret')
```

```
confusionMatrix(factor(test.results),
factor(testdata[,SLENGTH]))




#some nice plots with t-SNE
library(Rtsne)

per <- .4
N <- nrow(testdata)
tsnedata <- testdata[sample(N, as.integer(per * N)),]

Labels<-tsnedata[,SLENGTH]
tsnedata[,SLENGTH]<-as.factor(tsnedata[,SLENGTH])
## for plotting
colors = rainbow(length(unique(tsnedata[,SLENGTH])))
names(colors) = unique(tsnedata[,SLENGTH])

## Executing the algorithm on curated data
tsne <- Rtsne(tsnedata[,-SLENGTH], dims = 2,
        perplexity=30, verbose=TRUE, max_iter = 500)
exeTimeTsne<- system.time(Rtsne(tsnedata[,-SLENGTH],
dims = 2, perplexity=30, verbose=TRUE, max_iter = 500))

## Plotting
plot(tsne$Y, t='n', main="tsne")
```

```
text(tsne$Y, labels=tsnedata[,SLENGTH],
col=colors[tsnedata[,SLENGTH]])
```

A.3 EXPERIMENT 3

```
########################################################
#
# Classification of ECG data using ANN
#
# Version 1.0: non-shuffled training data,
# binary classification, confusion matrix
########################################################
library(matrixStats)
library(keras)
#install_keras() #only first time

#data folder (relative to current)
srcdir <- "../data/"

#data files
#srcfile1 <- "mitbih_test.csv"
#srcfile2 <- "mitbih_train.csv"
srcfile3 <- "ptbdb_abnormal.csv"
srcfile4 <- "ptbdb_normal.csv"

#read data
```

```
srcfile <- srcfile3
adata <- read.csv(paste0(srcdir, srcfile),header = F)
srcfile <- srcfile4
ndata <- read.csv(paste0(srcdir, srcfile),header = F)

#some useful plots
#1. Normal vs. Abnormal time series
yn <- ndata[13,-188]
ya <- adata[12, -188]
plot(1:length(yn), yn, col = "blue", type = 'l',
     main = "Normal vs. Abnormal time series",
     xlab = "signal sample",
     ylab = "normalized signal amplitude")
lines(1:length(ya), ya, col = 'red')
legend("topright", legend=c("Normal", "Abnormal"),
       col=c("blue", "red"), lty=1, cex=0.8)

#2. Normal times series
N <- 40
yn <- ndata[sample(nrow(ndata), N),-188]
plot(1:length(yn[1,]), yn[1,], col = "blue", type = 'l',
     main = "Multiple normal time series",
     xlab = "signal sample",
     ylab = "normalized signal amplitude")
for (i in 2:N) {
  lines(1:length(yn[i,]), yn[i,], col = 'blue')
```

```
}


#3. Abnormal times series
N <- 40
yn <- adata[sample(nrow(adata), N),-188]
plot(1:length(yn[1,]), yn[1,], col = "red", type = ’l’,
     main = "Multiple abnormal time series",
     xlab = "signal sample",
     ylab = "normalized signal amplitude")
for (i in 2:N) {
  lines(1:length(yn[i,]), yn[i,], col = ’red’)
}



#4. Multiple Normal vs. Abnormal times series
N <- 40
yn <- ndata[sample(nrow(ndata), N),-188]
ya <- adata[sample(nrow(adata), N),-188]
plot(1:length(yn[1,]), yn[1,], col = "blue", type = ’l’,
     main = "Multiple normal vs. abnormal time series",
     xlab = "signal sample",
     ylab = "normalized signal amplitude")
for (i in 2:N) {
  lines(1:length(yn[i,]), yn[i,], col = ’blue’)
}
for (i in 1:N) {
  lines(1:length(ya[i,]), ya[i,], col = 'red')
}
legend("topright", legend=c("Normal", "Abnormal"),
       col=c("blue", "red"), lty=1, cex=0.8)

#5. Multiple Abnormal vs. Normal times series
N <- 40
yn <- ndata[sample(nrow(ndata), N),-188]
ya <- adata[sample(nrow(adata), N),-188]
plot(1:length(ya[1,]), ya[1,], col = "red", type = 'l',
     main = "Multiple normal vs. abnormal time series",
     xlab = "signal sample",
     ylab = "normalized signal amplitude")
for (i in 2:N) {
  lines(1:length(ya[i,]), ya[i,], col = 'red')
}
for (i in 1:N) {
  lines(1:length(yn[i,]), yn[i,], col = 'blue')
}
legend("topright", legend=c("Normal", "Abnormal"),
       col=c("blue", "red"), lty=1, cex=0.8)

#6,7 Median distributions for normal and abnormal
hist(rowMeans(ndata[,-188]), xlab = "ECG signal means",
     main = "Normal ECG: median distribution")
hist(rowMeans(adata[,-188]), xlab = "ECG signal means",
     main = "Abnormal ECG: median distribution")


#8,9 SD distributions for normal and abnormal
hist(rowSds(as.matrix(ndata[,-188])),
     xlab = "ECG signal SDs",
     main = "Normal ECG: SDs distribution")
hist(rowSds(as.matrix(adata[,-188])),
     xlab = "ECG signal SDs",
     main = "Abnormal ECG: SDs distribution")


#10,11 Sums distributions for normal and abnormal
hist(rowSums(ndata[,-188]),  xlab = "ECG signal sums",
     main = "Normal ECG: sums distribution")
hist(rowSums(adata[,-188]),  xlab = "ECG signal sums",
     main = "Abnormal ECG: sums distribution")


#dev.copy(png,'results/expres5.png', device = x11)
#dev.print(width = 791, height = 529)
#dev.off()

####################### ANN model binary classification
seeds <- c(2020, 123, 451, 8720, 2130,
           9901, 2504, 5432, 4710, 8315)
```

```
cm <- list()
for (expi in 1:length(seeds)) {
  set.seed(seeds[expi])
  NDTRAINING <- sample(0.8*nrow(ndata))
  #normal data sample for training
  ADTRAINING <- sample(0.8*nrow(adata))
  #abnormal data sample for training



traindata <- rbind(ndata[NDTRAINING,], adata[ADTRAINING,])
#traindata <- traindata[sample(nrow(traindata)), ]

testdata <- rbind(ndata[-NDTRAINING,], adata[-ADTRAINING,])

  #some info from training data
  unique(traindata[,188])
  sum(traindata[,188] == 0)
  sum(traindata[,188] == 1)
  sum(testdata[,188] == 0)
  sum(testdata[,188] == 1)



  #retrieve inputs/outputs from data
  x_train <- as.matrix(traindata[,1:187])
  y_train <- to_categorical(traindata[,188], 2)
```

```
#Test: retrieve inputs/outputs from data
x_test <- as.matrix(testdata[,1:187])
y_test <- to_categorical(testdata[,188], 2)




#create the model
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = c(187)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = 'softmax')
#model %>%
#  layer_dense(units = 64, activation = 'relu',
input_shape = c(187)) %>%
#  layer_dropout(rate = 0.4) %>%
#  layer_dense(units = 32, activation = 'relu') %>%
#  layer_dropout(rate = 0.3) %>%
#  layer_dense(units = 5, activation = 'softmax')
```

```
  #print some model info (if wanted)
  summary(model)



#assemble the model
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)



#train (plot expres12)
model.info <- model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)

#plot the model info (plot expres13)
plot(model.info)
model %>% evaluate(x_test, y_test)

#test on test data
test.results <- model %>% predict_classes(x_test)
#create and show the confusion matrix (expres14)
```

```
library('caret')
cm[[expi]] <- confusionMatrix(factor(test.results),
                              factor(testdata[,188]))
}
allresults <- data.frame(matrix(cm[[1]]$overall
                         [c(1,3,4,6)],nrow = 1))
colnames(allresults)<-rownames(as.matrix(cm[[1]]
                               $overall[c(1,3,4,6)]))
for (i in 2:length(seeds)) {
  allresults <- rbind(allresults, cm[[i]]$overall
                                   [c(1,3,4,6)])
}
#plot mean and extreme accuracies over all experiments
#(expres15)
plot(allresults$Accuracy, col = 'blue', xlab = 'Experiment',
     ylab = 'Accuracy',
     ylim = c(min(allresults$AccuracyLower),
              max(allresults$AccuracyUpper)+0.05),
   main = 'Accuracy of binary classification\n(sorted data)',
     lwd = 2, type = 'l')
lines(allresults$AccuracyLower, col = 'green')
lines(allresults$AccuracyUpper, col = 'red')
legend("topright", legend=c("Upper", "Mean", "Lower"),
       col=c("green", "blue", "red"), lty=1, cex=0.8)

mean(allresults$Accuracy)
```

```
sd(allresults$Accuracy)

#export to LaTeX
allresults

table <- data.frame(cm[[1]]$table)



library(ggplot2)
library(dplyr)

#table <- data.frame(confusionMatrix(pred, truth)$table)

plotTable <- table %>%
  mutate(goodbad = ifelse(table$Prediction == table$Reference,
                          "correct", "incorrect")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))
#confusion matrix, expres14
# fill alpha relative to sensitivity/specificity by
#proportional outcomes within reference groups
#(see dplyr code above as well as original confusion matrix
#for comparison)

ggplot(data = plotTable, mapping = aes(x = Reference,
       y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
  geom_text(aes(label = Freq), vjust =.5, fontface = "bold",
               alpha = 1)+
  scale_fill_manual(values = c(correct = "green",
               incorrect = "red")) +
  theme_bw() +
  xlim(rev(levels(table$Reference)))



############## unsupervised learning #########################


#some nice plots with t-SNE (expres16,17)
library(Rtsne)


per <- .75
N <- nrow(traindata)
tsnedata <- unique(traindata[sample(N, as.integer(per * N)),])


Labels<-tsnedata[,188]
tsnedata[,188]<-as.factor(tsnedata[,188])
## for plotting
colors = rainbow(length(unique(tsnedata[,188])))
names(colors) = unique(tsnedata[,188])


## Executing the algorithm on curated data
tsne <- Rtsne(tsnedata[,-188], dims = 2, perplexity=30,
        verbose=TRUE, max_iter = 500)
#exeTimeTsne<- system.time(Rtsne(tsnedata[,-188], dims = 2,
perplexity=30, verbose=TRUE, max_iter = 500))

## Plotting
#classes <- sort(as.numeric(as.vector(unique(tsnedata[,188]))))
plot(tsne$Y, t='n', xlab = 'X', ylab = 'Y',
     main="T-distributed Stochastic Neighbor Embedding\n
                                        (training data)")
legend("topright", legend=c("Abnormal", "Normal"),
       col=colors[unique(tsnedata[,188])], pch = c("1", "0"))
text(tsne$Y, labels=tsnedata[,188], col=colors[tsnedata[,188]])

#now for test data
per <- 1.0
N <- nrow(testdata)
tsnedata <- unique(testdata[sample(N, as.integer(per * N)),])

#Labels<-tsnedata[,188]
tsnedata[,188]<-as.factor(tsnedata[,188])
## for plotting
#colors = rainbow(length(unique(tsnedata[,188])))
#names(colors) = unique(tsnedata[,188])

## Executing the algorithm on curated data
tsne <- Rtsne(tsnedata[,-188], dims = 2, perplexity=30,
        verbose=TRUE, max_iter = 500)
#exeTimeTsne<- system.time(Rtsne(tsnedata[,-188], dims = 2,
perplexity=30, verbose=TRUE, max_iter = 500))

## Plotting
#classes <- sort(as.numeric(as.vector(unique(tsnedata[,188]))))
plot(tsne$Y, t='n', xlab = 'X', ylab = 'Y',
     main="T-distributed Stochastic Neighbor Embedding\n
                                              (test data)")
legend("topright", legend=c("Abnormal", "Normal"),
       col=colors[unique(tsnedata[,188])], pch = c("1", "0"))
text(tsne$Y, labels=tsnedata[,188], col=colors[tsnedata[,188]])
```

A.4 EXPERIMENT 4

```
################################################################
#
# Classification of ECG data using ANN
#
# Version 1.0: shuffled training data, binary classification,
# confusion matrix
#
#################################################################

library(keras)
#install_keras() #only first time
```

```
#data folder (relative to current)
srcdir <- "../data/"

#data files
#srcfile1 <- "mitbih_test.csv"
#srcfile2 <- "mitbih_train.csv"
srcfile3 <- "ptbdb_abnormal.csv"
srcfile4 <- "ptbdb_normal.csv"

#read data
srcfile <- srcfile3
adata <- read.csv(paste0(srcdir, srcfile),header = F)
srcfile <- srcfile4
ndata <- read.csv(paste0(srcdir, srcfile),header = F)

#ANN classification
seeds <- c(2020, 123, 451, 8720, 2130,
           9901, 2504, 5432, 4710, 8315)
cm <- list()
for (expi in 1:length(seeds)) {
  NDTRAINING <- sample(0.8*nrow(ndata))
  #normal data sample for training
  ADTRAINING <- sample(0.8*nrow(adata))
  #abnormal data sample for training
```

```
traindata <- rbind(ndata[NDTRAINING,], adata[ADTRAINING,])
traindata <- rbind(traindata, traindata)
     #resampling (bootstrap)
traindata <- traindata[sample(nrow(traindata)), ] #shuffle

testdata <- rbind(ndata[-NDTRAINING,], adata[-ADTRAINING,])

#some info from training data
unique(traindata[,188])
sum(traindata[,188] == 0)
sum(traindata[,188] == 1)
sum(testdata[,188] == 0)
sum(testdata[,188] == 1)



#retrieve inputs/outputs from data
x_train <- as.matrix(traindata[,1:187])
y_train <- to_categorical(traindata[,188], 2)



#Test: retrieve inputs/outputs from data
x_test <- as.matrix(testdata[,1:187])
y_test <- to_categorical(testdata[,188], 2)
```

```
#create the model
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = ’relu’,
              input_shape = c(187)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = ’relu’) %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 2, activation = ’softmax’)
#model %>%
# layer_dense(units = 64, activation = ’relu’,
              input_shape = c(187)) %>%
# layer_dropout(rate = 0.4) %>%
# layer_dense(units = 32, activation = ’relu’) %>%
# layer_dropout(rate = 0.3) %>%
# layer_dense(units = 5, activation = ’softmax’)



#print some model info (if wanted)
summary(model)


#assemble the model
model %>% compile(
    loss = ’categorical_crossentropy’,
    optimizer = optimizer_rmsprop(),
    metrics = c(’accuracy’)
  )



  #train
  model.info <- model %>% fit(
    x_train, y_train,
    epochs = 30, batch_size = 128,
    validation_split = 0.2
  )

  #plot the model info
  plot(model.info)
  model %>% evaluate(x_test, y_test)

  #test on test data
  test.results <- model %>% predict_classes(x_test)
  #create and show the confusion matrix
  library(’caret’)
  cm[[expi]] <- confusionMatrix(factor(test.results),
                                factor(testdata[,188]))
}

allresults <- data.frame(matrix(cm[[1]]$overall[c(1,3,4,6)],
                              nrow = 1))
colnames(allresults) <- rownames
                    (as.matrix(cm[[1]]$overall[c(1,3,4,6)]))
for (i in 2:length(seeds)) {
allresults <- rbind(allresults, cm[[i]]$overall[c(1,3,4,6)])
}
#plot mean and extreme accuracies over all experiments
#(expres15_2)
plot(allresults$Accuracy, col = 'blue', xlab = 'Experiment',
     ylab = 'Accuracy',
     ylim = c(min(allresults$AccuracyLower),
              max(allresults$AccuracyUpper)+0.05),
   main = 'Accuracy of binary classification\n(shuffled data;
             resampling)',
     lwd = 2, type = 'l')
lines(allresults$AccuracyLower, col = 'green')
lines(allresults$AccuracyUpper, col = 'red')
legend("topright", legend=c("Upper", "Mean", "Lower"),
       col=c("green", "blue", "red"), lty=1, cex=0.8)

mean(allresults$Accuracy)
sd(allresults$Accuracy)

#export to LaTeX
allresults
```

```
table <- data.frame(cm[[1]]$table)



library(ggplot2)
library(dplyr)

#table <- data.frame(confusionMatrix(pred, truth)$table)
# (expres14_2)
plotTable <- table %>%
mutate(goodbad = ifelse(table$Prediction == table$Reference,
                          "correct", "incorrect")) %>%
  group_by(Reference) %>%
  mutate(prop = Freq/sum(Freq))

# fill alpha relative to sensitivity/specificity by
# proportional outcomes within reference groups (see
# dplyr code above as well as original confusion matrix
# for comparison)
ggplot(data = plotTable, mapping = aes(x = Reference
          y = Prediction, fill = goodbad, alpha = prop)) +
  geom_tile() +
geom_text(aes(label = Freq), vjust = .5, fontface  = "bold",
            alpha = 1) +
  scale_fill_manual(values = c(correct = "green",
            incorrect = "red")) +
  theme_bw() +
xlim(rev(levels(table$Reference)))
```