Politecnico di Torino

ICT for Smart Societies
A.a. 2022/2023

April 2023

Regression algorithms for estimation of cuff-less blood Pressure estimation

Supervisor:
Prof. Michela MEO
Prof. Guido Pagana

Candidate:
Elahe Faramini

To my parents, who have always been my source of strength, I would not be where I am today without them. This thesis is dedicated to you with love and admiration.

Abstract

Hypertension is a significant risk factor for cardiovascular diseases (CVDs), which are a leading cause of death around the world. Continuous monitoring of blood pressure (BP) is a proven tool to support patient care and when used in combination with other vital parameters such as heart rate, breath frequency, and physical activity, it can be highly effective in the prevention of CVDs. However, invasive methods are currently the most reliable way to continuously monitor BP, despite the potential for discomfort and damage to the patient. Non-invasive techniques are currently not considered optimal for continuously monitoring BP trends, as they can only return BP values every few minutes.

In this thesis work, cuff-less estimation of continuous BP through Pulse Transit Time (PTT) and Heart Rate (HR) using boosting regression techniques is investigated. This method achieves non-invasive estimation of BP with an acceptable low error, according to the AAMI guideline. The study introduces several methods, including the use of electrocardiographic (ECG), photoplethysmographic (PPG) signals and ABP (Arterial blood pressure) signals extrapolated from the MIMIC III online database, and the implementation of preprocessing of the ECG and PPG signals, and the research and processing of the features related to them in order to continuously monitor BP in a non-invasive way, exploiting boosting regression techniques. Recent studies have demonstrated that HR and PTT can be linearly combined to obtain BP values, manipulation of these two parameters is key to non-invasively estimating reliable BP values. The goal of this research is to examine the use of boosting regression techniques to estimate continuous blood pressure in a non-invasive way that equally depends- able with respect to current methods and easy for the patient to carry out, The user experience of this technology allows for the patient to easily and comfortably measure their blood pressure at various times throughout the day, regardless of their location, even if they are not in a clinical setting. This proposed approach can be viewed as the initial step towards the incorporation of these algorithms into wearable devices, particularly those developed for the SINTEC project.

Acknowledgements

I want to thank Prof. Guido Pagana, Dr. Valeria Figini, and LINKS Foundation company for believing in me and supporting me. The experience of working with them over the past several months has been both enlightening and enjoyable.

I am grateful to my supervisor, Professor Michela MEO, for accepting me to undertake this thesis.

My heartfelt thanks go to the treasures of my life, my brother and sister. I owe a particular debt of gratitude to my best friend Pouria for his unwavering support and encouragement throughout my academic journey. Without his help, I would not have been able to overcome the challenges and obstacles along the way.

I wish to acknowledge and thank Samaneh, who has supported me since the day I decided to pursue this field. Finally, I want to express my appreciation to Shahnam for being by my side through all the highs and lows.

Thank you all.

Elahe Faramini

Contents

List of Acronyms

ABP	Arterial Blood Pressure
BP	Blood Pressure
CAT Boost	Categorical Boosting
CVDs	Cardiovascular Diseases
DBP	Diastolic Blood Pressure
DB	Database
ECG	Electrocardiography/Electrocardiographic
ECG	Electrocardiogram
GB	Gradiant Boosting
HR	Heart Rate
ICT	Information and Communication Technologies
IOT	Intent Of Thing
IR	low-intensity infrared
MAE	Mean Absolute Errore
PCB	Printed Circuit Board
PPG	Photoplethysmography/Photoplethysmographic
PTT	Pulse Transit Time
SBP	Systolic Blood Pressure
SD	Standard Deviation
SINTEC	Soft Intelligence Epidermal Communication platform
XGB	EXtreme Gradient Boosting

List of Figures

Chapter 1

Introduction

The high prevalence of hypertension and its association with adverse cardiovascular outcomes make it crucial to have reliable and accurate methods for blood pressure monitoring. The importance of non-invasive technologies for monitoring health parameters cannot be overstated, especially for tracking conditions such as hypertension. This thesis focuses on examining the performance of boosting regression algorithms on the estimation of blood pressure by utilizing physiological signals collected from the MIMIC III online database. SINTEC-Soft Intelligence Epidermal Communication Platform is a European project launched in June 2019, with the goal to create a modern technology for tracking the health of its user[1]. The final product should be capable of returning the subject's heart rate, systolic (SBP), and diastolic blood pressure (DBP) during periodic monitoring. This research was conducted at LINKS Foundation, which has been operating for approximately two decades in both the national and international sphere, with the purpose of promoting, leading, and enhancing innovation processes through research projects that possess a strong innovative potential, and has the capacity to generate an impact on public production sectors whilst being competitive in an international context[2].

SINTEC aims to meet the need for the development of new interconnected technologies that are non-invasive and do not interfere with people's daily lives meanwhile speed is a key factor. Therefore the optimization of the algorithm was further investigated. Besides improvements in hardware devices, we need to promote estimation algorithms so that reaching trust-able estimated blood pressure.

1.1 Main goal

This research will explore the boosting regression algorithms in depth. The initial section of the work integrated an algorithm for extracting physiological parameters such as heart rate, pulse transit time, and systolic and diastolic blood pressure, and then the algorithms have been tested. The main focus of the LINKS Foundation in the SINTEC project is its applicability in a clinical and hospital setting. Currently, only intrusive procedures are reliable for continuous blood pressure monitoring; they involve the insertion of invasive arterial catheters (that bring potential risks to patients such as infection and vascular damage [5]. The chance to have reliable non-invasive continuous blood pressure monitoring provides a significant benefit regarding prevention and reduction of risks associated with cardiovascular diseases where hypertension is the main risk factor [6].

1.2 Hypertension and associated monitoring techniques

Hypertension is often referred to as the "silent killer" due to its difficulty in diagnosis before the signs and symptoms become visible, leading to potentially irreversible damage [7]. Despite the widely publicized advantages of lowering blood pressure (BP), a significant portion of the population still has high BP as a primary risk factor for illness and disability, with the number of people affected constantly increasing [8]. Chronic hypertension is characterized by elevated baseline BP for long periods of time [9]. It is recommended that all adults over 18 years of age have their blood pressure monitored for the purpose of diagnosing hypertension and accurately calculating cardiovascular risk [10]. More than 90% of cases of hypertension are attributable to poor nutrition, obesity, and physical inactivity. Additionally, rising blood pressure in older adults is linked to changes in arterial structure and increased arterial stiffness [10, 11].

Moreover, there is a close correlation between the increase in BP and cardiovascular risk and all evidence indicates that treating older adults hypertensive patients will reduce the risk of cardiovascular events [13].

The BP monitoring process typically involves the utilization of two distinct methodologies: invasive and noninvasive. The commonly implemented solutions in these cases are as follows:[14][52]

- The invasive arterial catheter method is employed for ongoing surveillance, but it presents potential hazards to patients in the form of infection and various vascular injuries. (Figure1.1)

- In order to intermittently monitor blood pressure, a sphygmomanometer - an arm cuff that occludes - is utilized. Blood pressure can be obtained through manual means, such as by auscultation of Korotkoff sounds or palpation, or automatically through oscillometry (palpation) or automatically (by oscillometry) [15]. The Holter blood pressure monitor (HBPM) allows for periodic readings at intervals of 15 or 30 minutes, over a duration of up to 48 hours. [10].

However, there are clinical scenarios in which it is either hard to measure blood pressure with traditional cuff-based devices or difficult to safely conduct invasive arterial monitoring [9]. The principles and techniques employed in the measurement of blood pressure without the use of a cuff have been debated and studied for decades [16]. There is a need to develop more precise methods of measuring blood pressure, to promote harmony between international hypertension guidelines, bolster diagnostic reliability and accuracy, facilitate better clinical decisions, and ultimately improve patient clinical outcomes [17]. Identifying secure and reliable estimation techniques is imperative for early diagnosis and mortality reduction in hypertension. A model incorporating these features would significantly advance the monitoring and prevention of cardiovascular disease. We hope to have a revolutionary effect on the lives of patients with CVDs with the help of these hypotheses, by giving people a convenient and accurate way to monitor their blood pressure [18]. This may prevent the emergence and damage of cardiovascular diseases, which are still the primary cause of death globally.

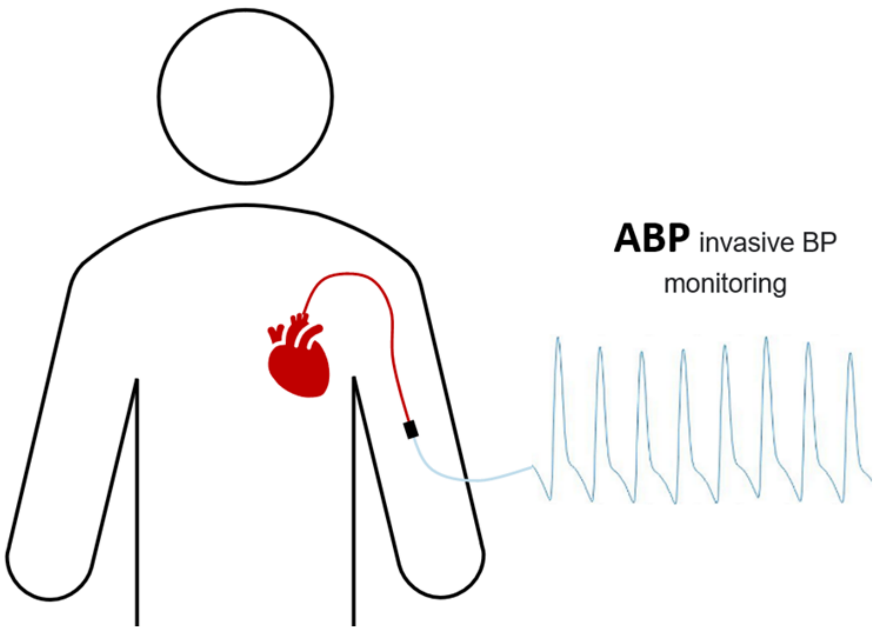


Figure 1.1: ABP signal recorded invasively[52]

1.3 Advancement Beyond the Existing

Traditional methods for estimating blood pressure, such as the auscultatory method or oscillometric method, rely on simple formulas or guidelines to determine blood pressure values. These methods can be affected by measurement errors, observer bias, and inter-observer variability, which can result in inaccurate estimations of blood pressure. In contrast, ML regression algorithms can learn complex patterns in data, account for patient-specific factors, and provide personalized and accurate predictions.

Another method for estimating blood pressure is invasive arterial blood pressure monitoring, which involves inserting a catheter into an artery to directly measure blood pressure. While this method is considered the gold standard for blood pressure monitoring, it is invasive and carries a risk of complications such as bleeding, infection, or arterial damage. ML regression algorithms can provide non-invasive and continuous monitoring of blood pressure, reducing the risks associated with invasive monitoring.

In addition, ML regression algorithms can provide real-time monitoring and updates, allowing clinicians to quickly respond to changes in blood pressure values. This is in contrast to other methods that may provide a one-time estimation of blood pressure, which may not reflect the patient's current health status.

The major objectives are:

- **Improved accuracy:** ML regression algorithms can learn complex patterns in data and make predictions based on these patterns. This can result in more accurate blood pressure estimations compared to traditional methods that rely on simple formulas or guidelines.

- **Personalized predictions:** ML algorithms can be trained on large data sets to identify patient-specific factors that influence blood pressure, such as age, sex, comorbidities, and medications. This allows for personalized predictions that can better reflect the individual patient's health status and needs.

- **Real-time monitoring:** ML algorithms can be designed to continuously monitor and update blood pressure predictions in real time based on the latest patient data. This can provide clinicians with up-to-date information and help them make timely interventions if necessary.

- **Reduced errors and variability:** Traditional methods for estimating blood pressure may have a higher degree of measurement error or variability, which can lead to miss classification and inaccurate diagnoses. ML algorithms can reduce these errors and variability, improving the accuracy of blood pressure estimations

- **Reduced workload for clinicians:** ML algorithms can automate the process of blood pressure estimation, reducing the workload for clinicians and allowing them to focus on other aspects of patient care.

1.4 Physiological signals

Recent advancements in ECG and PPG technology enable BP estimation with a satisfactory degree of accuracy [24]. PWV, PTT(Figure 1.2), and PWA-based techniques are some of the most widely used approaches for detecting continuous blood pressure without a cuff [32]. Overall, PTT has demonstrated excellent potential and delivered the best outcomes in terms of reliability and MEA. PTT can be determined by analyzing ECG and PPG signals [34, 35]. PTT is defined as the amount of time the pressure wave needs to travel from a proximal to a distal location of the body [33].

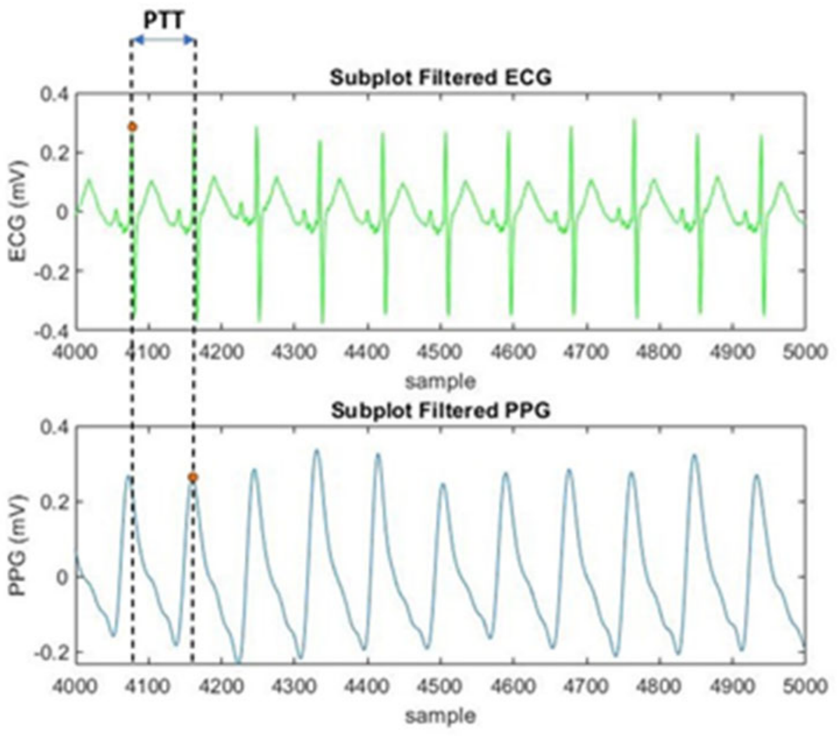


Figure 1.2: Time interval between R-peak and S-peak [52]

Previous research has established a connection between PTT and BP since the early 2000s, and machine-learning methods have been explored to address the poor accuracy of PTT [25, 26]. Subsequent investigations into the biomechanical properties of vessels and their impact on the self-regulating mechanism of blood flow led to the application of one of Moens-Korteweg's fluid dynamic laws to link PWV to DBP and SBP. Calibrations can be used to obtain pressure values for a particular subject under evaluation, as Poon and Zhang demonstrated through mathematical approximations [27]. To ensure accuracy over short- and long-term periods, multiple calibrations were necessary for the process [28, 29, 30]. DBP (also called minimum pressure), the diastolic value of blood pressure is the value observed when the heart of an individual relaxes between heartbeats [31]. SBP (or maximum pressure) is the Blood Pressure value when an individual's heart contracts, thus the Blood Pressure value with each heartbeat [31].

1.4.1 ECG

The ECG signal can be acquired through a non-invasive procedure and is a visual representation of the electrical and chemical activity of cardiac muscle fibers during the cardiac cycle. The QRS complex, consisting of three waves (Q, R, and S) which are produced by ventricular depolarization after atrial depolarization, plays an essential role in this process [36].(Figure 1.3). In this study, the R-peaks (which corresponds to left ventricle depolarization) are featured in (Figure 1.4). With the time interval,Δt, between two consecutive R$-$peaks established, it is possible to determine the HR [37] via the following equation:

$$\mathrm{HR} = \frac{1}{\Delta t} \tag{1.1}$$

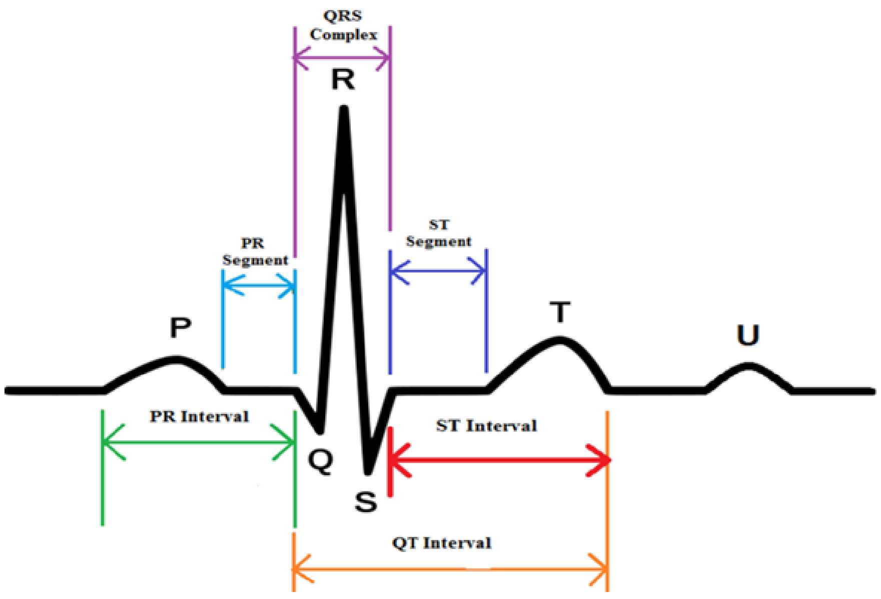


Figure 1.3: ECG wave form [75]

1.4.2 PPG

PPG instead is a simple and low-cost optical technique used to detect blood volume changes in the microvascular bed of tissue at the skin surface level [38].

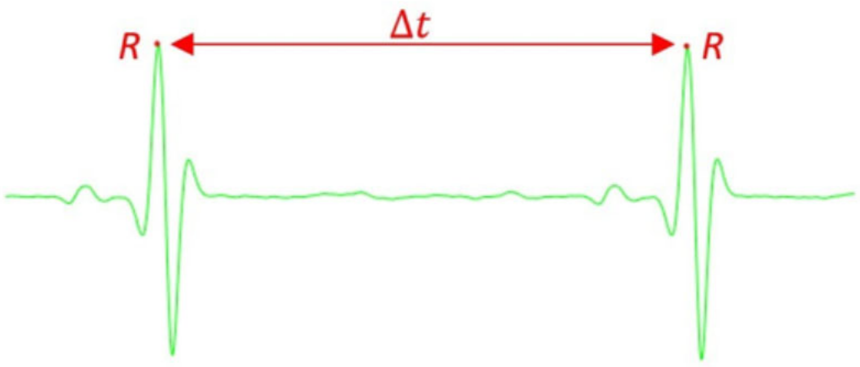


Figure 1.4: Time interval between two consecutive R−peaks[52].

PPG is an optical technique that utilizes a IR light sensor to measure fluctuations in peripheral blood volume [39, 40].

As light is more readily absorbed by blood than tissue, variations in the intensity of light can be transduced as changes in blood flow. The sensitivity of the sensor allows for even minor changes in blood volume to be detected. The PPG waveform is composed of alternating (AC) and direct (DC) components (Figure 1.5) [41].

- The AC component represents blood volume cardiac variation in each heartbeat, and it is attributed to the pulsatile behavior of the heart
- The DC component is highly correlated to central and periphery venous pressure [42]. The average blood volume experiences gradual changes over an extended period, however, abrupt fluctuations can arise due to multiple factors, e.g.breathing, the presence of a disease, vasomotor activity, sympathetic nervous system activity, and thermoregulation [43].

Utilizing this waveform, it is feasible to calculate the PTT as time interval between an ECG, R-peak, and the closest S-peak of the PPG signal (Figure1.2)

1.4.3 ABP

The arterial blood pressure signal reflects the pressure wave moving through the arteries, showing different rates of diffusion and morphology depending on the artery's cross-sectional area.

is the pressure exerted by the blood on the walls of arteries. Arterial blood pressure is one of the most important vital signs in clinical practice, and it is routinely measured in the intensive care unit (ICU) to monitor the cardiovascular health of critically ill patients. Physiologically, a pressure wave traveling through a viscoelastic tube is progressively weakened as it moves with an exponential reduction

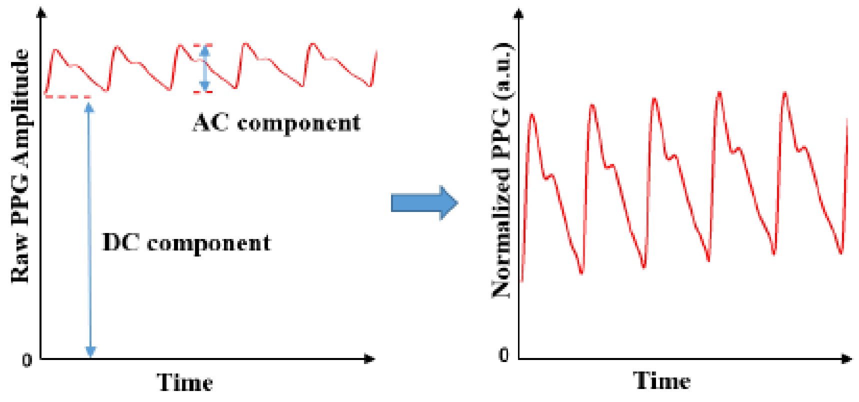


Figure 1.5: AC and DC components of the PPG waveform[76].

in speed. However, when the tube branches off into different diameters, signal amplification occurs due to reflections. In clinical settings, ABP is recorded invasively in the least rigid vessel - the aorta - where reflections are minimal [44]. From this signal, it is possible to calculate systolic blood pressure and diastolic blood pressure, which correspond to the maximum and minimum values of the signal respectively.

Chapter 2

Materials and methods

2.1 Materials

This study exploited the MIMIC III database, the most popular non-invasive pressure estimation database, created by the MIT Lab for Computational Physiology. It contains more than 60,000 acquisitions from ICU (Intensive Care Unit) patients. The database includes information such as vital signs, medications, laboratory results, demographics, and other clinical data.

The rationale for our decision is supported by the abundant availability of signals such as ECG, PPG, and ABP, which enabled us to successfully deploy the system [45], and signals are sampled at 125 Hz. In order to ensure that only records containing all of the relevant signals (ECG, PPG, ABP) and a sufficient number of samples (at least 1 min of recorded signal) were used.

2.2 Methods

It has been demonstrated in [47] that there is a strong correlation between PTT and BP. This is based on the Bramwell–Hills and Moens–Kortweg equation [48]:

$$\mathrm{PWV} = \frac{L}{PTT} = \sqrt{\frac{hE}{\rho d}} \tag{2.1}$$

In the academic context, it is noted that vessel wall thickness (h), blood density (ρ), vessel diameter (d), length (L), and vessel elastic modulus (E) are significant factors in the study of vascular mechanics. Interestingly, Leslie A. Geddes found in 1991 that there is an exponential correlation between E and pressure ρ [26], specifically:

$$\mathrm{E} = E_0 e^{-\alpha d} \tag{2.2}$$

By replacing Equation (2.2) in the Bramwell–Hills and Moens–Kortweg's Equation (2.1), PWV can be written as:

$$\mathrm{PWV} = \frac{L}{PTT} = \sqrt{\frac{hE_0 e^{-\alpha d}}{\rho d}} \tag{2.3}$$

Which leads to:

$$\mathrm{e}^{\alpha d} = \frac{\rho d L^2}{hE_0} \times \frac{1}{PTT^2} \tag{2.4}$$

and to:

$$\mathrm{P} = \frac{1}{\alpha} \ln\left(\frac{\rho d L^2}{hE_0}\right) - \frac{2}{\alpha} \ln(PTT) \tag{2.5}$$

The relationship between BP and PTT can thus be simplified as [47]:

$$\mathrm{B}P = a \ln(PTT) + b \tag{2.6}$$

Chan et al. [27] postulated that if the variation of d with the BP is negligible and if the change in the arterial wall tone (E0) is slow enough, then the second term of the right hand side of Equation (2.6) can be regarded as constant during the observation window, and it is possible that:

$$\Delta BP = \frac{2}{\alpha PTT} \Delta(PTT) \tag{2.7}$$

A linear approximation for Equation (2.7) was suggested:

$$\mathrm{B}P = \alpha PTT + b \tag{2.8}$$

Since numerous studies demonstrate the enhancement brought about by including heart rate (HR) in the equation, the mathematical relationship between BP and PTT becomes [48]:

$$\mathrm{B}P = \alpha PTT + bHR + C \tag{2.9}$$

The parameters a, b, and c are specific to each subject and require calibration for determination. The linear regression model used in this research is represented by the last equation. In order to assess the maximum and minimum blood pressure values, we conducted an analysis of arterial blood pressure by breaking down the estimation into diastolic and systolic blood pressure, as outlined in the following equation (2.10):

$$\begin{cases} \text{SBP} = a_s\text{PTT} + b_s\text{HR} + c_s \\ \text{DBP} = a_d\text{PTT} + b_d\text{HR} + c_d \end{cases} \tag{2.10}$$

This method was utilized due to the satisfactory results that were obtained. However, these results could be augmented particularly in regard to predicting more dynamic signals. The continuity of the signal was employed to address this issue. As the signal is consistent, its extracted characteristics are also continuous. It was decided to contemplate the values within a certain observation window. The duration of this observation window was determined by a trial-and-error approach.

2.2.1 Data Collection

In this work, an algorithm was initially developed that uses the signals present on the MIMIC III online database [49].

In this study, a selection process was performed during the data collection process, a total of 99 signals were obtained. The selection criteria involved :

- verifying the presence of three specific signals, namely ECG, PPG, and ABP.
- the signals were analyzed for the existence of peaks or periodicity that persisted for more than 5 seconds.
- Each signal has a length of approximately 50 seconds.

In this study, 61 signals were further analyzed to identify the peaks and extract HR and PTT features [46]. Continuous BP measurement during various activities is necessary and we suggested using HR and PTT features within a specific time window to enhance performance.

Should any of the following criteria be met, the batch of ECG, PPG, and ABP signals pertaining to a single individual was excluded:

- Absence of any peaks (leading to an erroneous estimate of HR and PTT features) was noticed for more than 1 s.
- The estimation of HR or PTT following this procedure was deemed unfeasible in physical terms. (e.g., DBP reaches 0 mmHg).
- Possible lack of synchronization between the signals.

2.2.2 Data Processing

Filtering of Signals
Band-pass filtering [50] was applied to ECG and PPG signals from the MIMIC database; this filter was not necessary for ABP signals. The 5th-order Butterworth filter utilized had upper (fH) and lower (fL) cutoff frequencies as follows: (Figure2.1)

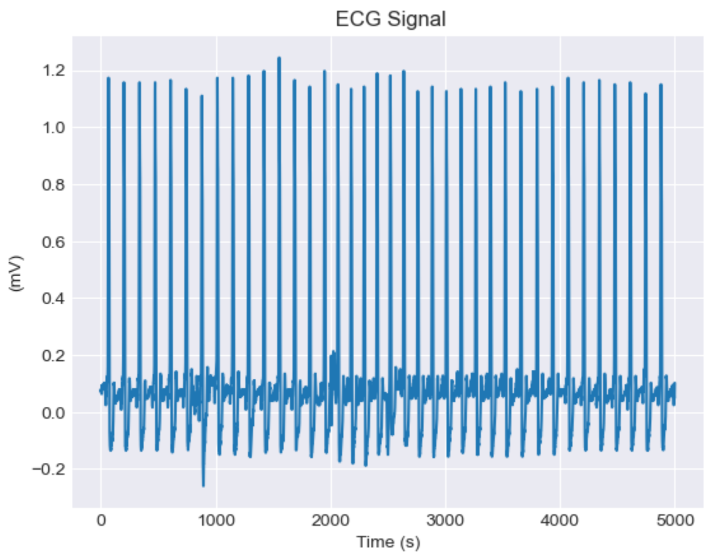
- $f_L = 1Hz$
- $f_H = 10Hz$

2.2.3 Feature Extraction

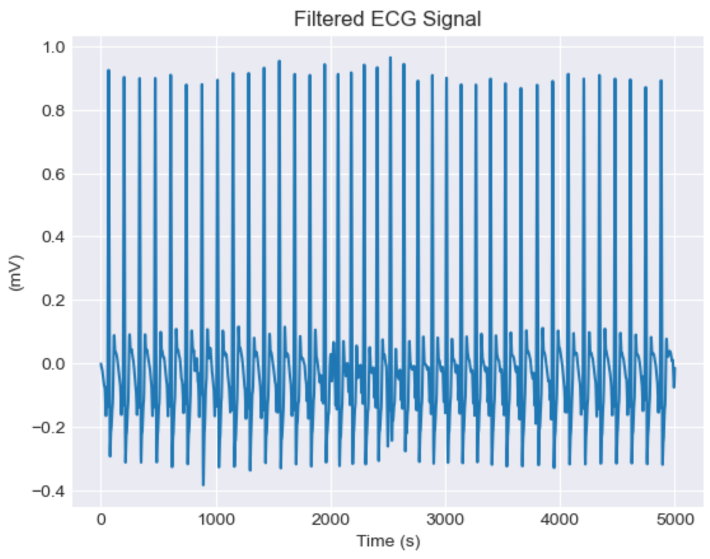
In the literature, there is a well-known correlation between PTT and HR for the estimation of BP. The relationship between PTT and HR is related to changes in blood pressure (BP). When BP increases, blood vessels become more rigid and less compliant, which leads to an increase in PTT. Conversely, when BP decreases, blood vessels become more compliant, leading to a decrease in PTT. As HR increases, PTT decreases. This is because a faster heart rate results in a shorter time between each heartbeat, which means that the pulse wave travels a shorter distance between the heart and the peripheral artery. As a result, PTT is inversely proportional to HR.
To acquire the signals from the MIMIC III database, features were collected over a time span of 1.5 seconds corresponding to two cardiac cycles. The proposed reference formula, which is an extension of Equation (2.11), is a generic formula:

$$\mathrm{B}P_i = \sum_{k=0}^{N} \alpha_k PTT_{i-k} + \sum_{k=0}^{N} b_k HR_{i-k} \tag{2.11}$$

Where N denotes the total number of samples taken during the time T and index i denotes the signal's i-th sample.



(a)



(b)

Figure 2.1: ECG signal before and after filtering with Butterworth filter.

$$\mathrm{HR}_i = \frac{60}{R_{\mathrm{peak}(i+1)} - R_{\mathrm{peak}(i)}} \tag{2.12}$$

In order to obtain PTT, it is necessary to accurately identify the R-peaks of the ECG and the PPG (see Figure 1.2). The signals need to be synchronized to the millisecond, however, this criterion is met in database [51].

2.2.4 Extraction of MIMIC III Database

Extracting the PPG peaks (S-peaks) can be challenging, as each signal of the MIMIC III database has a different amplitude, suggesting inter-subject variability in the PPG [53].

To detect peaks in arrays containing both ECG and PPG values, we developed a Python software that utilizes the `scipy.signal.find_peaks()` method. The method enables our algorithm to identify all peaks that surpass a given threshold value, which may vary between different subjects but should remain constant for measurements taken from the same individual. We also perform an additional check to ensure that no more than one peak is identified within a 0.5-second time window. Detected peaks are then stored in two different zero arrays that are of the same length as the reference timestamp array.

As signals had to be normalized to make the software applicable to any shape, multiple incorrect peaks were identified when searching for the S-peak of the PPG. To ensure that only S-peaks were detected, a kernel density estimation (KDE) of peak amplitudes was calculated and plotted (Figure 2.2) [52]. If there are two peaks in the distribution, as shown in (Figure 2.2), then only peaks above the minimum of these two values were kept. Subsequently, SBP was extracted as maximum of ABP and DBP was obtained as minimum of ABP, as shown in (Figure 2.3 c).

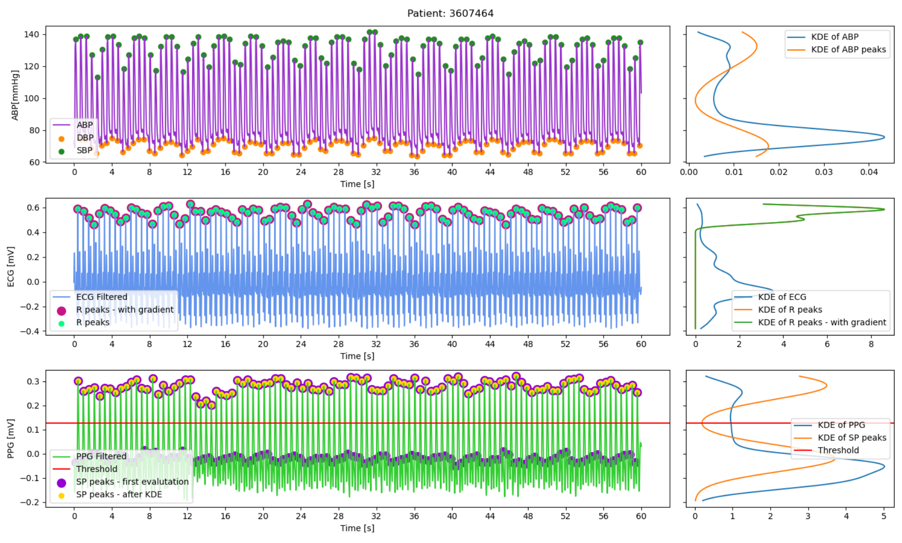


Figure 2.2: Peak finder and estimated probability density function (KDE) of the amplitude of peaks.

A more detailed flowchart of this part of the algorithm is reported in (Figure 2.4)

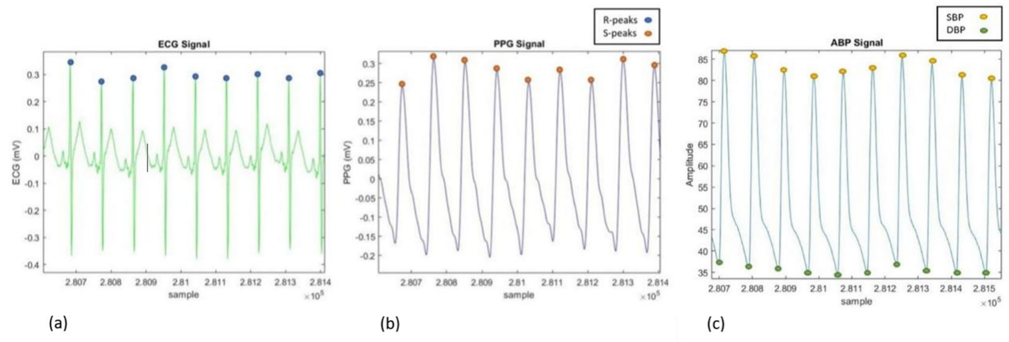


Figure 2.3: (a) Rpeaks, (b) Speaks, and (c) maxima and minima detection in signals from the MIMIC III database.

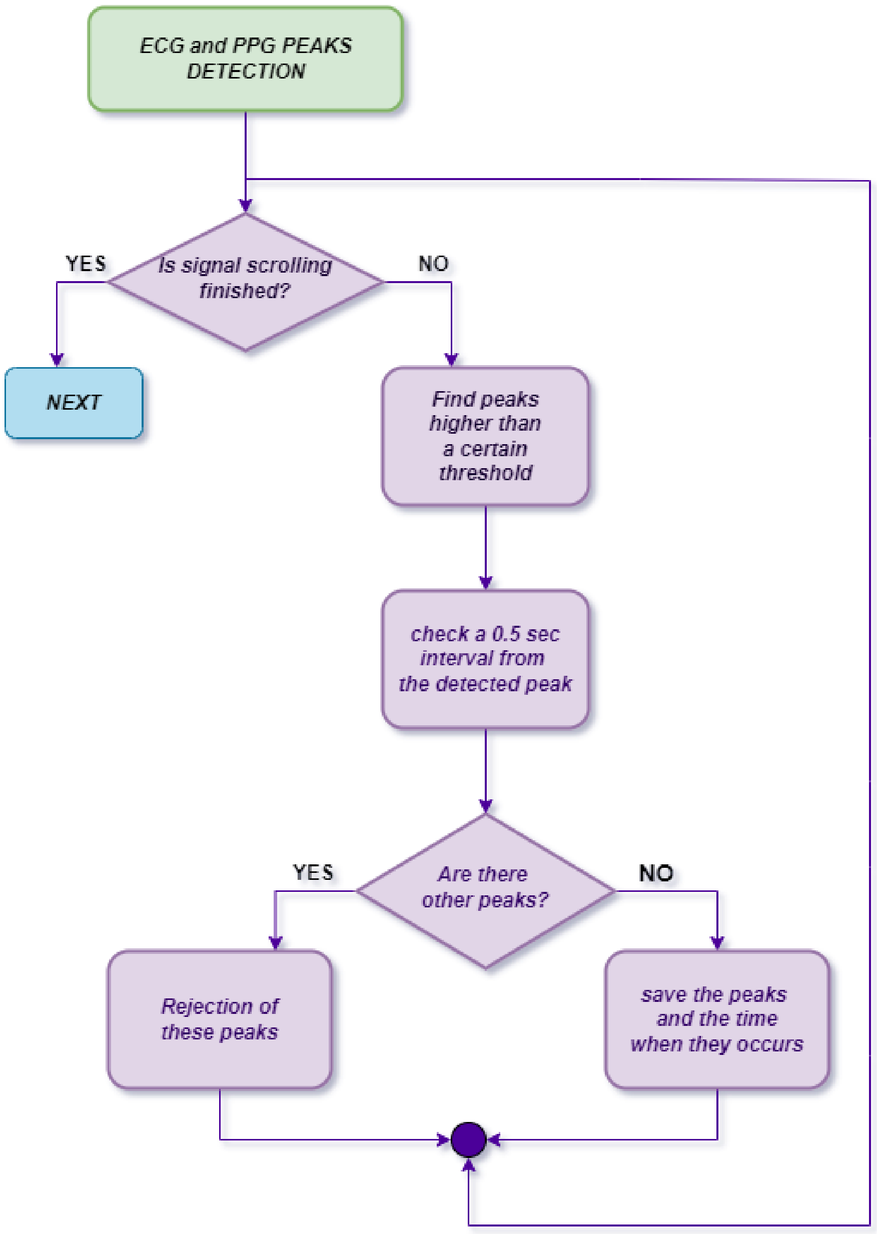

ECG and PPG PEAKS DETECTION
Is signal scrolling finished?
YES
NO
NEXT
Find peaks higher than a certain threshold
check a 0.5 sec interval from the detected peak
Are there other peaks?
YES
NO
Rejection of these peaks
save the peaks and the time when they occurs

Figure 2.4: Detailed R-peaks and S-peaks detection flowchart.

Chapter 3

The algorithm

3.1 Regression Process

Signals extracted from the MIMIC III database were used to retrieve blood pressure (BP) measurements. The regression process was structured in two phases: training and testing [53]. In the training phase, a lagged technique has been used.

creating lagged columns process is creating new columns that represent lagged values of the original columns, which will be used as predictors in a regression model. By including lagged values as predictors, the model can take into account the temporal relationship between the variables and potentially improve its accuracy. The process of creating lagged columns in a data frame for regression analysis involves several steps. For each column in the Data Frame, new lagged columns are created by shifting the values of the original column by different lag indices. This involves creating new columns, shifting values, and naming the new columns with the original column's name and the lag index. The data frame is then cleaned by dropping rows with missing values, and an intercept column of ones is added for the regression model. Finally, the columns for the regression analysis are updated to include the lagged columns and the intercept column(see Figure 3.1)[74].

The HR and PTT values were obtained by measuring the difference between each consecutive R-peak indices, then dividing this value by the sampling frequency to obtain the value in seconds (Equation (2.12)).

The PTT values were estimated by analyzing the ECG and PPG signals and comparing the positions of their peaks. Specifically, assuming that each R-peak is followed by an S-peak, only those values that adhere to this pattern are considered to be valid. The MIMIC III database's HR and PTT signals were divided into ten windows, and the mean and standard deviation (SD) were evaluated for each window. To eliminate outliers potentially caused by hardware malfunctions, all points located outside of the range of mean $\pm$ SD were substituted with a blank space, and then interpolated using the Python pandas.interpolate (method = 'polynomial',

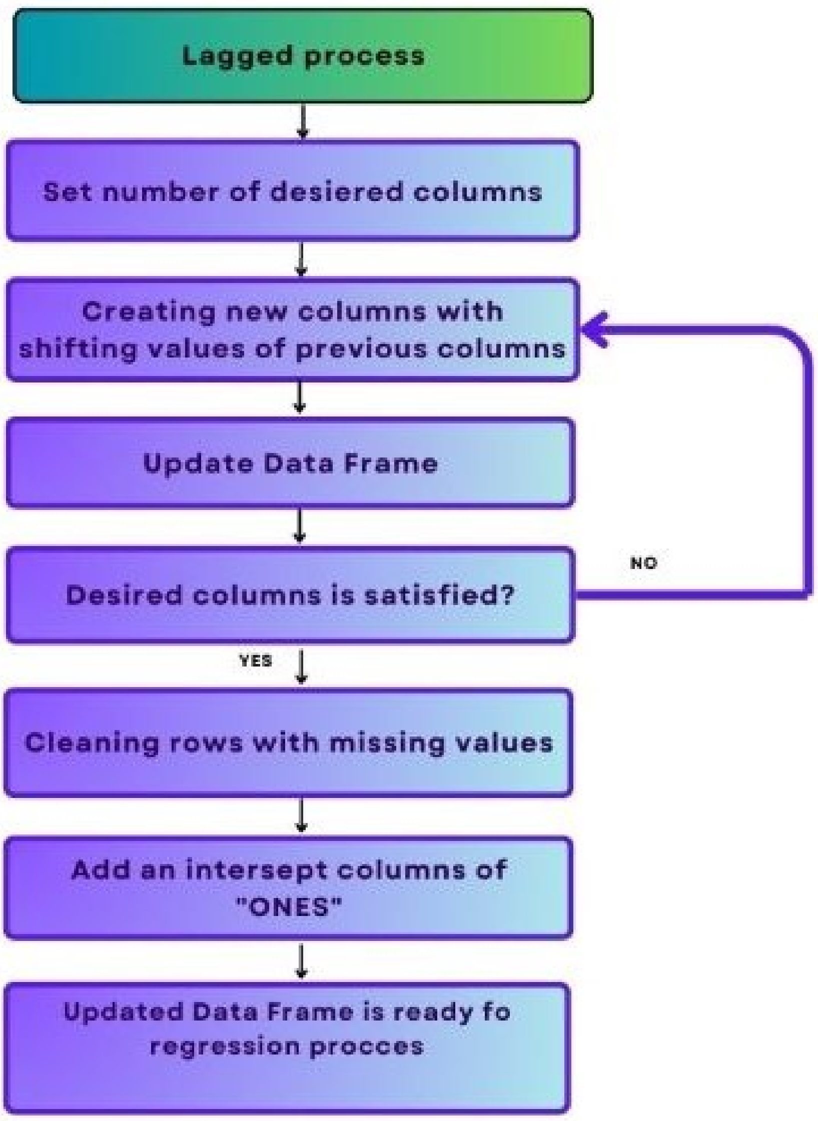

Lagged process
Set number of desiered columns
Creating new columns with shifting values of previous columns
Update Data Frame
Desired columns is satisfied?
NO
YES
Cleaning rows with missing values
Add an intersept columns of "ONES"
Updated Data Frame is ready fo regression procces

Figure 3.1: Lagged process for preparing signals from the MIMIC III database for the regression process.

order =1) function (order = 5 for HR, order = 1 for PTT).

A first-order polynomial regression can accurately fit a straight line to the data, whereas a fifth-order polynomial regression can provide a more intricate curve. While higher-order polynomials have the capacity to capture more complex trends, there is an increased risk of overfitting, whereby the interpolating function follows the noise in the data rather than the underlying pattern and thus yields unreliable results when applied to fresh data.

In the end, PTT and HR values are interpolated along the whole signals'timestamp array, ready for the feature reduction process. Also for this algorithm section, a more detailed flowchart is reported (Figure 3.2) [54].

3.2 Regression analisys

Linear regression has been around for a considerable amount of time and continues to be a commonly utilized statistical learning approach [55].

This work is based on a straightforward, simple linear approach for predicting a quantitative response, Y, on the basis of one or more predictor variables, X. It assumes there is an approximately linear relationship between X and Y [56]. To account for two predictors (PTT and HR), a multivariate regression model needs to be used with the regression line given in Equation (3.1). The intercept, β_0, and other β coefficients represent the average effect on Y of a one-unit increase in its respective predictor X while holding the other predictors fixed.

$$\mathrm{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \tag{3.1}$$

For the purpose of the work, the algorithm implements three different regression processes, from the scikit-learn, xgboost [62] and catboost Python libraries, were used for the estimation of both DBP and SBP values: each one with two distinct divisions of data sets:(Figure 3.3)

- the first (75%) of the sample (about 40 s of signal) for each batch was used for model construction
- the other (25%) (about 10–15 s of signal) for testing the algorithm

3.3 Machine learning algorithms

Gradient boosting regression trees is an ensemble method derived from a decision tree structure. This hierarchical tree structure has the potential to create overfitted models if the depth is too deep. As a remedy, ensemble methods combine multiple

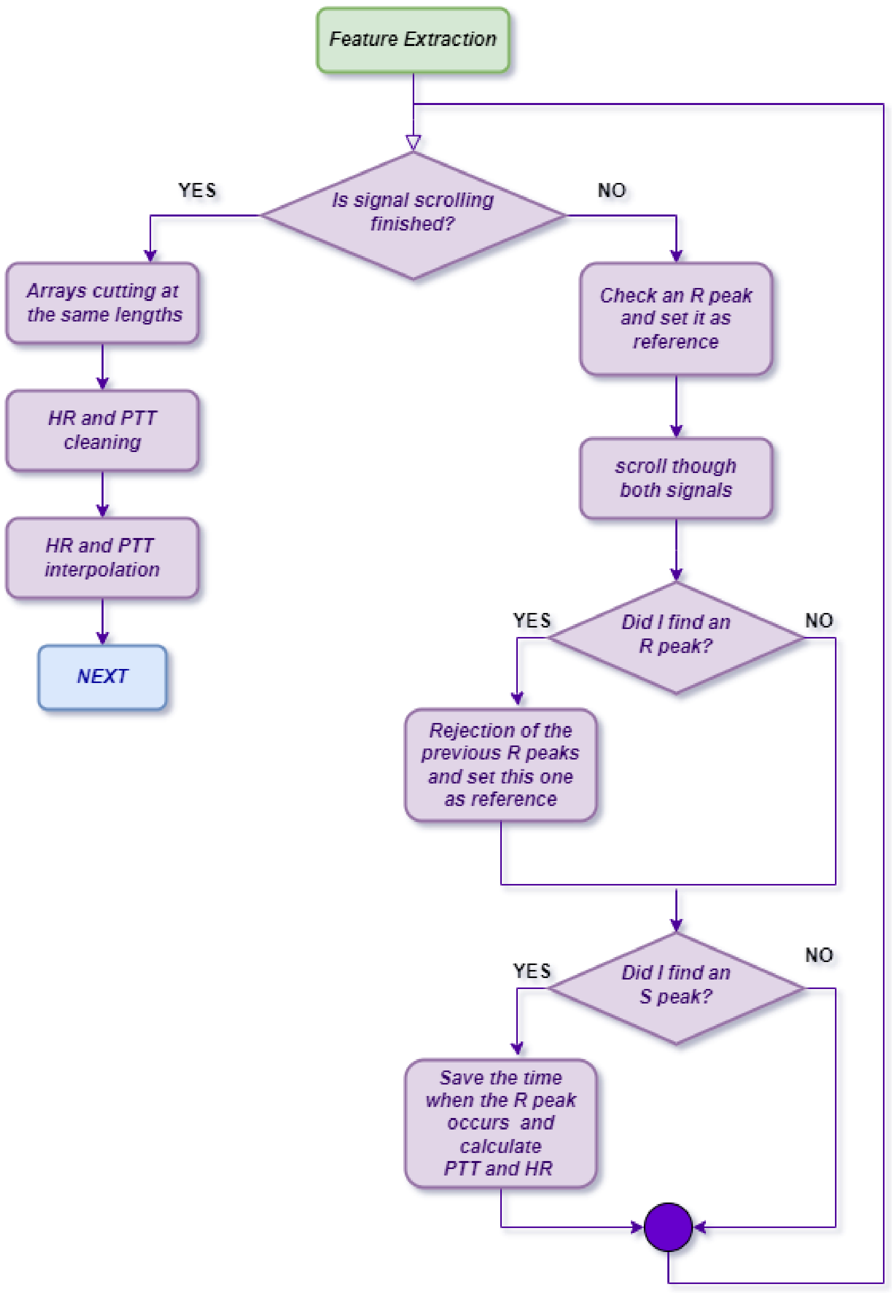

Feature Extraction
Is signal scrolling finished?
YES
NO
Arrays cutting at the same lengths
HR and PTT cleaning
HR and PTT interpolation
NEXT
Check an R peak and set it as reference
scroll though both signals
Did I find an R peak?
YES
NO
Rejection of the previous R peaks and set this one as reference
Did I find an S peak?
YES
NO
Save the time when the R peak occurs and calculate PTT and HR

Figure 3.2: Detailed feature extraction flowchart.

decision trees instead of a single one (Figure 3.4). Random forest and GB are two well-known algorithms under this category; while random forest builds multiple trees through splitting datasets based on random numbers, GB is a technique that repeatedly adds decision trees to correct the error of previous ones. Although GB requires more precise parameter settings for training, it can potentially generate

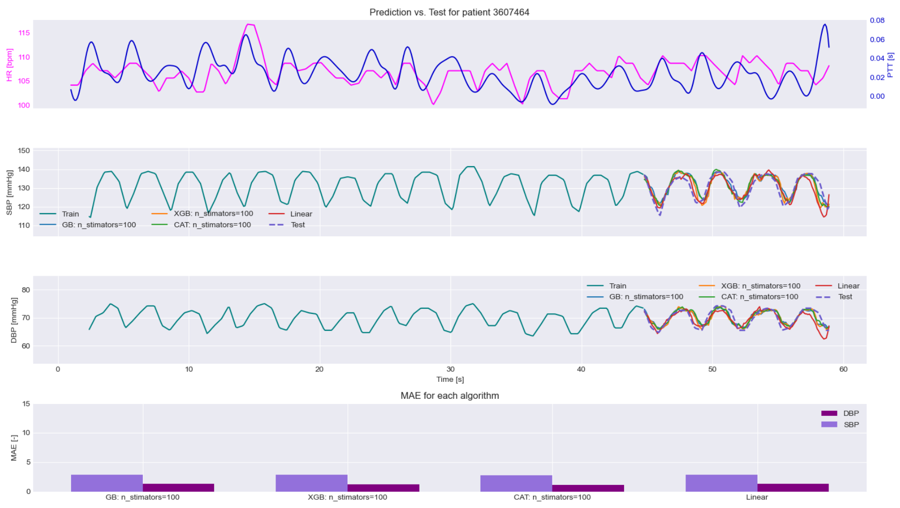


Figure 3.3: Test phase and training phase of the algorithms for the MIMIC III database signals : (a) interpolation of the input values, (b,c) results of regression, and (d) MAE values for the selected measurement.

better test results than a random forest with proper configuration.
The three different implemented regression processes are the following:

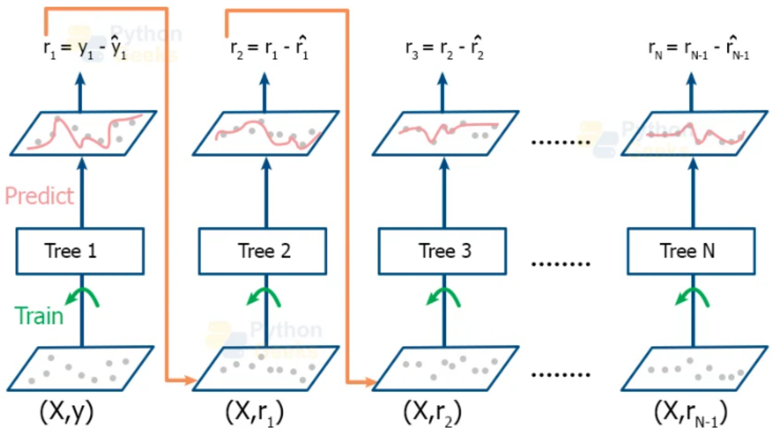


Figure 3.4: The architecture of Gradient-Boosting Decision Tree[73]

3.3.1 Gradiant Boosting Regression (GBR)

The concept of boosting originated from the question of whether a weak learner could be enhanced to improve its performance [57]. The Gradient Boosting algorithm is employed to generate an ensemble model through the aggregation of weak predictive models. This algorithm can be utilized to construct models for both regression and classification problems. Specifically, the Gradient Boosting Regression algorithm is used to fit a model that predicts a continuous value.

Gradient Boosting creates an additive model by utilizing multiple decision trees of a fixed magnitude as weak predictive models.
Gradient Boosting Regressor is a powerful machine learning algorithm that is widely used for regression problems. The algorithm involves building an ensemble of decision trees, each of which is trained to correct the errors of the previous tree. In this way, the algorithm gradually improves the accuracy of the predictions until a satisfactory level is reached.

Let's take a deep dive into the mathematical details of GBR:

Step 1: Initialization

The algorithm starts by initializing the prediction function as a constant value, typically the mean of the target variable. Let's denote this as:

$$\mathrm{F}_0(x) = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{3.2}$$

where x represents the input features, y is the target variable, and n is the number of training samples.

This initialization step sets the baseline prediction for the target variable, and the subsequent steps aim to improve this prediction by iteratively building decision trees.

Step 2: Building Trees

In each iteration, the algorithm builds a new decision tree that predicts the residuals (errors) of the current prediction function $F_{m-1}(x)$. The residual is defined as the difference between the true target value and the current prediction:

$$\mathrm{r}_{im} = y_i - F_{m-1}(x_i) \tag{3.3}$$

where m denotes the iteration number and i is the index of the training sample.

The decision tree is built using a technique called "gradient boosting". The idea is to fit a tree to the negative gradient of the loss function with respect to the current prediction function. In other words, we want to find the tree that best corrects the errors of the previous prediction function.
The negative gradient is given by:

$$-\frac{\partial F_{m-1}(x_i)}{\partial L(y_i, F_{m-1}(x_i))} \tag{3.4}$$

where L is the loss function that measures the difference between the true target value and the predicted value.
The negative gradient represents the direction and magnitude of the error i.e., how much the current prediction needs to be adjusted to reduce the error. By fitting a tree to the negative gradient, we aim to find the best split points that reduce the error the most.

Step 3: Tree Fitting

The decision tree is fitted to the negative gradient using a process called "boosting". The boosting process involves fitting the tree to the negative gradient using a technique called "gradient descent". In each iteration of the boosting process, the tree is fit to the negative gradient by minimizing the following objective function:

$$\textstyle\sum_{i=1}^{n} \frac{(\gamma_{jm} - r_{im})^2}{w_i} \tag{3.5}$$

where j is the index of the tree node, m is the iteration number, γ_{jm} is the predicted value of the tree node, and w_i is the weight of the training sample.

The weight w_i is a function of the current prediction function $F_{m-1}(x_i)$ and the iteration number m. It is given by:

$$\mathrm{w}_i = \frac{1}{1 + \exp(-2y_i F_{m-1}(x_i))} \tag{3.6}$$

The weight function is designed to give more weight to samples that are harder to predict, i.e, those that have a larger error in the current prediction.
The tree is fit using a greedy algorithm that recursively splits the data into two parts based on a threshold value that maximizes the objective function. The splitting process continues until a stopping criterion is met, e.g, when the maximum depth of the tree is reached or when the minimum number of samples per leaf node is reached.

Step 4: Updating the Prediction Function

After the tree is built, its predictions are added to the current prediction function $F_{m-1}(x)$ in a weighted manner. The weight of the tree α_m is determined by minimizing the following objective function:

$$\sum_{n}^{i=1} L(y_i) \tag{3.7}$$

Step 5: Learning Rate

In each iteration of the GBR algorithm, the contribution of the new tree to the prediction function is scaled by a learning rate η, which is a value between 0 and 1. The learning rate controls the step size of the optimization process and helps prevent overfitting. The updated prediction function is given by:

$$\mathrm{F}_m(x) = F_{m-1}(x) + \eta h_m(x) \tag{3.8}$$

where $h_m(x)$ is the new decision tree.

A smaller learning rate reduces the impact of each tree on the final prediction, which can help prevent overfitting. However, a smaller learning rate also increases the number of iterations required to converge.

Step 6: Regularization

To prevent overfitting, the GBR algorithm includes several regularization techniques, such as:

- Tree Depth: The maximum depth of the decision trees is limited to prevent them from becoming too complex and overfitting the training data.
- Minimum Samples per Leaf: The minimum number of samples required to make a split at a leaf node is limited to prevent overfitting to noise.
- Subsampling: A fraction of the training data can be randomly sampled without replacement at each iteration to reduce variance and improve generalization.
- Feature Subsampling: A fraction of the features can be randomly sampled without replacement at each split point to reduce correlation and improve generalization.

Step 7: Convergence

The GBR algorithm continues to iteratively build decision trees until the loss function converges or until a predefined maximum number of iterations is reached. The loss function measures the difference between the true target value and the predicted value, and it is used to evaluate the performance of the model.

The parameter, `n_estimators`, determines the quantity of decision trees to be implemented in the boosting stages. Gradient Boosting is distinct from AdaBoost

[58] as it utilizes decision trees of a fixed magnitude rather than decision stumps (one node and two leaves) which are used in AdaBoost. Friedman extended this framework [59] to create Gradient Boosting Machines, which are also known as Gradient Boosting or Gradient Tree Boosting.
All in all, Gradient Boosting utilizes differentiable loss functions generated by weak learners in order to extrapolate. At every boosting phase, the learner is employed to reduce the loss function in accordance with the current model. Boosting algorithms can be utilized for either regression or classification purposes.
The primary steps for implementing Gradient Boosting Regression are as follows:

- Select a weak learning model
- Utilize an additive model
- Establish a loss function
- Minimize the identified loss function

3.3.2 Extreme Gradient Boosting(XGBooste)

XGBoost, which stands for Extreme Gradient Boosting, is a parallelized and optimized version of numberient boosting algorithm. By parallelizing the entire boosting process, the training time is significantly reduced. Rather than training a single model on the data (as done in traditional methods), XGBoost trains thousands of models on various subsets of the training dataset and then combines them to create the best-performing model(Figure3.5). It is a popular implementation of gradient boosting that is optimized for both computational efficiency and predictive accuracy.
Gradient boosting is an ensemble learning method that combines multiple weak learners (usually decision trees) to create a strong learner. In XGBoost, the weak learners are decision trees, and the algorithm learns the optimal weights for each tree so that the final model predicts the target variable with high accuracy.
The XGBoost algorithm uses a gradient boosting approach, where it adds new models to the ensemble sequentially. At each iteration, the algorithm fits a new tree to the residual errors from the previous iteration. This process is repeated until the algorithm has reached a stopping condition, such as a maximum number of trees or when the improvement in the objective function is below a certain threshold.

XGBoost is designed to be highly customizable and flexible, allowing users to tweak many hyperparameters to achieve the best performance on their specific problem. Some of the most important hyperparameters in XGBoost include:

- **Learning rate (eta):** This controls the contribution of each tree to the final prediction. A smaller learning rate will result in slower learning but may lead

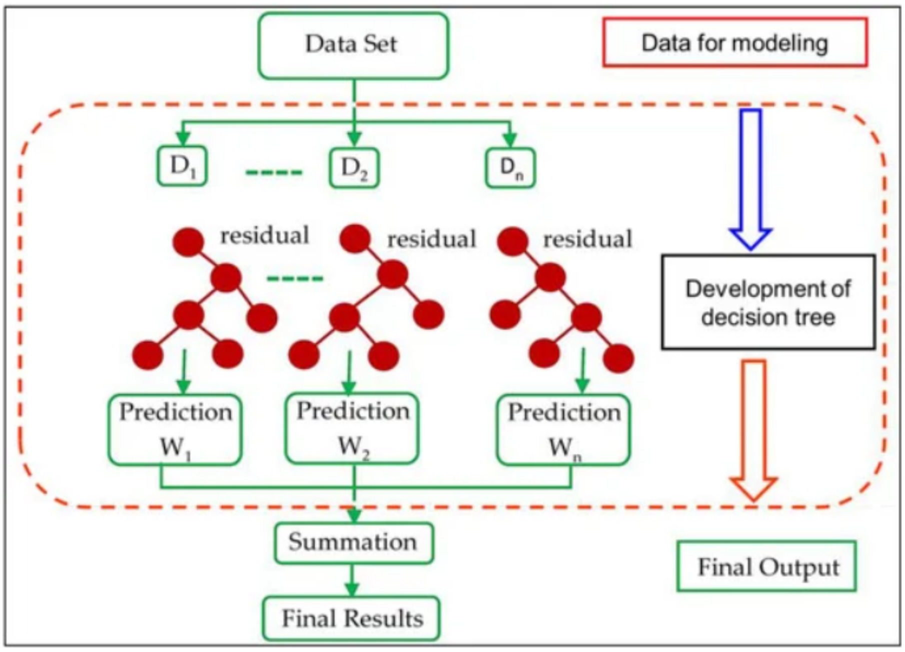


Figure 3.5: The architecture of- Extreme Gradient Boosting [81].

to better performance, while a larger learning rate will result in faster learning but may lead to overfitting.

- **Maximum depth (max_depth):** This limits the depth of each decision tree in the ensemble. A deeper tree can model more complex interactions in the data but may overfit to the training data.
- **Number of trees (n_estimators):** This controls the number of decision trees in the ensemble. Increasing the number of trees can improve performance but may lead to longer training times and overfitting.
- **Subsample:** This controls the fraction of the training data used to fit each tree. A smaller subsample can reduce overfitting but may increase variance.
- **Column subsampling (colsample_bytree):** This controls the fraction of features used to fit each tree. A smaller fraction can reduce overfitting but may increase bias.
- **Regularization parameters (lambda, alpha):** These control the strength of L1 and L2 regularization applied to the weights of the decision trees. Regularization can help prevent overfitting and improve generalization performance.

Objective function

In XGBoost, the objective function is a sum of two components: a loss function that measures the error between the predicted values and the true values, and a regularization term that penalizes complex models. The loss function is problem-specific and can be chosen from a variety of options, such as mean squared error (MSE) for regression problems or log loss for binary classification problems.

Mathematically, the objective function of XGBoost can be written as:

$$\mathrm{Obj}(\theta) = \sum_{\mathrm{i=1}}^{\mathrm{n}} \mathrm{l(y_i, \hat{y}i)} + \sum \mathrm{j} = 1^{\mathrm{m}}\Omega(\mathrm{f_j}) \tag{3.9}$$

where θ represents the model parameters, $l(y_i, \hat{y}_i)$ is the loss function that measures the error between the predicted value $\hat{y}_i$ and the true value y_i, m is the number of trees in the ensemble, f_j represents the j-th tree, and $\Omega(f_j)$ is the regularization term that penalizes the complexity of the model.

To optimize this objective function, XGBoost uses gradient descent to iteratively update the model parameters θ in the direction of the negative gradient of the objective function. This process is known as gradient boosting, where each new tree is trained to minimize the residual errors from the previous trees. The final prediction is a weighted sum.

Regularization

To prevent overfitting, XGBoost also includes several regularization techniques:

1. **Max Depth:**

   It controls the maximum depth of the tree By limiting the depth of the tree. Limiting the maximum depth of each decision tree to prevent it from becoming too complex so the model can avoid overfitting to the training data.

2. **L1 regularization (Lasso):** It adds an L1 penalty term to the cost function, which penalizes the model for using too many features or for using features that are not important. This technique helps to reduce the complexity of the model and can be used for feature selection.

3. **L2 regularization (Ridge):** It adds an L2 penalty term to the cost function, which penalizes the model for using large weights. This technique helps to reduce the impact of any individual feature and can be used to avoid overfitting.

4. **Min Child Weight:** It controls the minimum sum of weights of all observations required in a child. By setting this parameter, the model can avoid overfitting to noisy data.

5. **Gamma:** It controls the minimum loss reduction required to make a further partition on a leaf node of the tree. By setting this parameter, the model can avoid overfitting to noisy data.

6. **Subsample:** It controls the fraction of observations to be used for each tree. By setting this parameter, the model can avoid overfitting to the training data.
7. **Colsample_bytree:** It controls the fraction of features to be used for each tree. By setting this parameter, the model can avoid overfitting to the training data.

By using these regularization techniques, XGBoost can reduce overfitting and improve the performance of the model on unseen data.

In this step, I'll provide a step-by-step explanation of how XGBoost works, including the mathematical details of each step.

Step 1: Initialization:

The first step in XGBoost is to initialize the ensemble model with a constant prediction value. This value is typically set to the mean of the target variable for regression problems, or to the ratio of the number of positive and negative instances for binary classification problems. Let's call this constant prediction value $\hat{y}_0$.

Step 2: Compute the Residuals:

The next step is to compute the residuals for each training instance. The residual r_i for the i-th instance is defined as the difference between the true target value y_i and the current prediction $\hat{y}_m(x_i)$ of the ensemble model at the m-th iteration:

$$\mathrm{r}_i = y_i - \hat{y}_m(x_i) \tag{3.10}$$

At the first iteration ($m = 0$), the current prediction is the initialization value $\hat{y}_0$ for all instances, so the residuals are simply the differences between the true target values and the mean (for regression problems) or the ratio of positives and negatives (for binary classification problems).

Step 3: Train a Weak Learner:

The next step is to train a weak learner (usually a decision tree) to predict the residuals. This weak learner is trained on the features X and the residuals r computed in the previous step. Let's call this weak learner $h_m(x)$.

Step 4: Compute the Scaling Factor:

The next step is to compute a scaling factor γ_m that minimizes the loss function L for the residuals r_i and the predictions $\hat{y}_{m-1}(x_i)$. This scaling factor represents the contribution of the new decision tree $h_m(x)$ to the overall prediction of the ensemble model. Mathematically, the scaling factor is computed as follows:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \hat{y}_{m-1}(x_i) + \gamma h_m(x_i)) \tag{3.11}$$

This minimization problem can be solved using gradient descent, since the loss function L is typically differentiable.

Step 5: Update the Prediction:

The final step is to update the prediction of the ensemble model using the new decision tree and the scaling factor. The new prediction $\hat{y}_m(x)$ is computed as follows:

$$\mathrm{\hat{y}m(x) = \hat{y}m - 1(x)} + \eta\gamma_{\mathrm{m}}\mathrm{h}_{\mathrm{m}}\mathrm{(x)} \tag{3.12}$$

In this equation, $\hat{y}_{m-1}(x)$ represents the prediction of the ensemble model at the $m-1$-th iteration for the input x, $h_m(x)$ represents the new decision tree added to the ensemble model at the m-th iteration for the input x, γ_m represents the scaling factor computed in the previous step, and η represents the learning rate, which controls the contribution of the new decision tree to the overall prediction.

Some other important features of XGBoost are:[70][71][72]

- **Parallelization:**The implementation of the model allows for training with multiple CPU cores.
- **Non-linearity:** XGBoost can identify non-linear data trends and learn from them.
- **Cross-validation:**installed by default and ready-to-use .

Furthermore, by utilizing the Python implementation, users have access to a range of internal parameters that can be adjusted to optimize accuracy and precision [60, 61]

The two goals of the project align with the two objectives of using XGBoost.

- Execution Speed.
- Model Performance.

3.3.3 Categorical Boosting (CatBoost)

CatBoost is a highly effective gradient boosting algorithm applied to supervised machine learning tasks, with certain exclusive traits that set it apart from other boosting algorithms [62].

This specific assortment of Gradientgradient-boostingons has the capacity to manage categorical variables and data in general. The CatBoost object can process categorical or numeric variables, as well as datasets with a variety of types; in a more efficient and accurate manner. It was developed by Yandex and was made open-source in 2017 [77].

In traditional gradient boosting, categorical features need to be preprocessed and converted into numerical features before they can be used in the model. This is typically done by one-hot encoding, where each category is transformed into a binary vector indicating its presence or absence. However, one-hot encoding can lead to high-dimensional feature spaces and may result in overfitting, especially when dealing with sparse or high-cardinality categorical features.

CatBoost addresses this issue by implementing a novel algorithm that can directly handle categorical features without the need for one-hot encoding or other preprocessing steps. The algorithm is based on the idea of ordered boosting, where categories are sorted based on their target statistics and are split into subsets using the optimal threshold.

Furthermore, it is able to exploit unlabelled examples and analyze the impact of kernel size on speed during training. The Ordered Boosting (OB) algorithm is a novel method of encoding categorical features, which has been demonstrated to improve performance on datasets with categorical features. This algorithm encodes the categories based on the target variable's mean and variance values, rather than relying on frequency as a basis for encoding.
The CatBoost algorithm consists of the following steps:

- **Initialization:** A constant prediction value is set for all instances, which is typically the mean or median value of the target variable.
  For each iteration compute the negative gradient of the loss function with respect to the predictions of the previous iteration F_{m-1}.

- **Gradient computation:** The gradient of the loss function with respect to the prediction is computed for each instance.

- **Sorting and splitting of categorical features:** Categorical features are sorted based on their target statistics (e.g. mean, sum) and are split into subsets using the optimal threshold.

- **Tree building:** A decision tree is trained on each subset of categorical features and the numerical features. Each leaf node of the tree represents a categorical feature value or a numerical range.

- **Gradient update:** The negative gradient of the loss function with respect to the prediction is computed for each instance and is used to update the prediction values.

- **Regularization:** A regularization term is added to the objective function to penalize complex models and prevent overfitting.

The objective function of CatBoost can be expressed as:

$$obj(\Theta) = L(y, F(\Theta)) + \Omega(\Theta) \tag{3.13}$$

where Θ denotes the model parameters, y denotes the true values, F(Θ) denotes the predicted values, L is the training loss function, and ω is the regularization term.
In CatBoost, the training loss function is typically the log loss function for binary classification or the multinomial log loss function for multiclass classification.
catBoost works by iteratively adding decision trees to the model. At each iteration, the algorithm fits a decision tree to the gradient of the loss function with respect to the predictions of the previous iteration.
Some key features of CatBoost:

- **Handling categorical features:** CatBoost's main benefits over conventional gradient boosting are its capacity to handle categorical features without preprocessing and its effective implementation, which can drastically reduce training time and memory requirements. It is widely used in both industry and academics and has been demonstrated to deliver state-of-the-art performance on a variety of datasets.[78].

- **Regularization:** To further prevent overfitting, CatBoost uses two types of regularization: L2 regularization and a novel technique called Categorical Feature Combination. L2 regularization penalizes large weights in the decision trees and helps to reduce overfitting. Categorical Feature Combination, on the other hand, works by combining the categorical values into new, higher-level features. This reduces the number of features in the model and improves its generalization ability.

- **Robust to outliers:** CatBoost is robust to outliers in the target variable, making it suitable for noisy data.

- **Handling missing values:** Handling missing values in categorical boosting involves making a decision about how to treat missing data, such as ignoring them, imputing them, or creating a separate category. The best approach will depend on the nature of the data and the goals of the model.

- **GPU support:** CatBoost can utilize GPUs to speed up the training process.

CatBoost Regressor is an effective algorithm that is capable of accommodating large datasets with categorical features and has been commonly used in various domains such as finance, health care, and marketing [63].

All in all Boosting algorithms can offer several advantages for estimating blood pressure from ECG and PPG signals. Some of these advantages include:

1. **Improved accuracy:** Boosting algorithms, such as XGBoost and CatBoost, can improve the accuracy of predictions by reducing bias and variance in the model. This can be particularly useful when estimating blood pressure from ECG signals, which can be complex and noisy signals.

2. **Handling of non-linear relationships:** Boosting algorithms can capture non-linear relationships between the ECG signals and blood pressure values. This can be useful in cases where the relationship between the two variables is not linear, as is often the case in medical applications.

3. **Handling of missing data:** Boosting algorithms, such as CatBoost, can handle missing data effectively, reducing the need for imputation and minimizing the impact of missing values on the accuracy of the model.

4. **Handling of categorical data:** CatBoost can handle categorical data effectively, which can be useful in cases where ECG signals are categorized based on their characteristics.

5. **Fast computation:** Boosting algorithms are known for their fast computation time, which can be important in medical applications where time is critical.

3.3.4 Hyper parameters Selection

In boosting regression, hyperparameters are tuning parameters that are set before training the model and can greatly affect the model's performance. The best values for hyperparameters depend on the specific data and problem at hand and can be found through a process called hyperparameter tuning [79].

There are several approaches to finding the best hyperparameters in boosting regression. Finding the best hyperparameters in boosting regression involves exploring different values for each hyperparameter and selecting the combination that yields the best performance. Different approaches such as grid search, random search, Bayesian optimization, and ensemble methods can be used to find the optimal hyperparameters. It is important to keep in mind that the best hyperparameters may vary depending on the specific data and problem being solved, and hyperparameter tuning should be performed carefully to avoid overfitting the training data [80].

in this work, we select hyperparameters with a grid search. Grid search involves specifying a range of values for each hyperparameter and then exhaustively

searching through all possible combinations of hyperparameters to find the best combination. While this approach can be computationally expensive, it is simple to implement and can be effective in finding the best hyperparameters.
Some advantages of using grid search are:

- **Comprehensive:** Grid search systematically searches over a range of hyperparameters and evaluates all possible combinations, ensuring that no parameter configuration is overlooked.

- **Easy to implement:** Grid search is a simple and easy-to-implement method for hyperparameter tuning, making it accessible to both beginners and experts.

- **Reproducible:** Grid search is a reproducible method, as it evaluates all possible hyperparameter combinations, ensuring that the optimal set of parameters is selected each time.

- **Can handle multiple metrics:** Grid search can be used to optimize multiple metrics simultaneously, allowing for more complex model evaluations.

We define a parameter grid to search using different values for the regressor hyperparameters. Then create an instance of the regressor and an instance of **GridSearchCV**, passing the regressor and the parameter grid as arguments. We fit the GridSearchCV instance on our data and then print the best parameters.

Chapter 4

Results

The algorithm has been tested for all the measurements in the database.

4.1 Algorithm test

The prediction performance was assessed by utilizing the Mean Absolute Error (MAE) value. [64, 65, 66].
In order to determine the validity of measurements, one must refer to the AAMI /ESH/ISO guidelines. According to these guidelines, a pass/fail criterion should be applied to general population samples (at least 85 measurements) as well as populations with a smaller sample size (at least 35 measurements). This criterion requires that the mean difference between test and reference blood pressure measurements be less than or equal to 5 mmHg with a standard deviation of 8 mmHg or less for both systolic and diastolic blood pressure values.[68]

$$\begin{cases} \text{MAE} \leq 5\text{mmHg} \\ \text{SD} \leq 8\text{mmHg} \end{cases} \tag{4.1}$$

In this study, all of the algorithms were evaluated using data from 90 patients in the MIMIC III database showed an average error value below 5 mmHg for both DBP and SBP values [69](Table 4.1).and almost in (98%) of them avarage of MAE is less than 2 in dbp error and less than 3 in sbp error.

In this study, all regression methods are considered suitable since they yield nearly equal Mean Absolute Error (MAE) and Standard Deviation (SD) values for each prediction.

Compared to previous works boosting regression could be chosen because it is characterized by a shorter computational time. From a first glance to figure 3.4 it can be seen that these regressors are able to follow the trend of the signal testing phase, being able to recognize the variability, also demonstrated stability when the

Table 4.1: Average MAE and average SD values in the testing phase with signals

AVERAGE MAE (mmHg) $\pm$ AVERAGE SD (mmHg)

	Gradiant Boosting	**XG Boosting**	**CAT Boosting**
SBP	2.9 ± 2.8	3.1 ± 2.7	3 ± 3
DBP	2.2 ± 2.8	2.3 ± 2.7	2.3 ± 3

input feature values were quite dissimilar to the previous ones.

4.2 Calibration Time

In order to evaluate the calibration time in this work, for a test vector with a fixed length (25%), of the total data set in the first step, just (10%) of the length of the training set vector was considered and the length of training set was increased in each iteration. As shown in(Figures 4.1, 4.2, 4.3, 4.4), in comparison with Linear regression these boosting algorithms have better calibration time and even with a limited amount of data they can estimate blood pressure with MAE less than 5 mmHg.

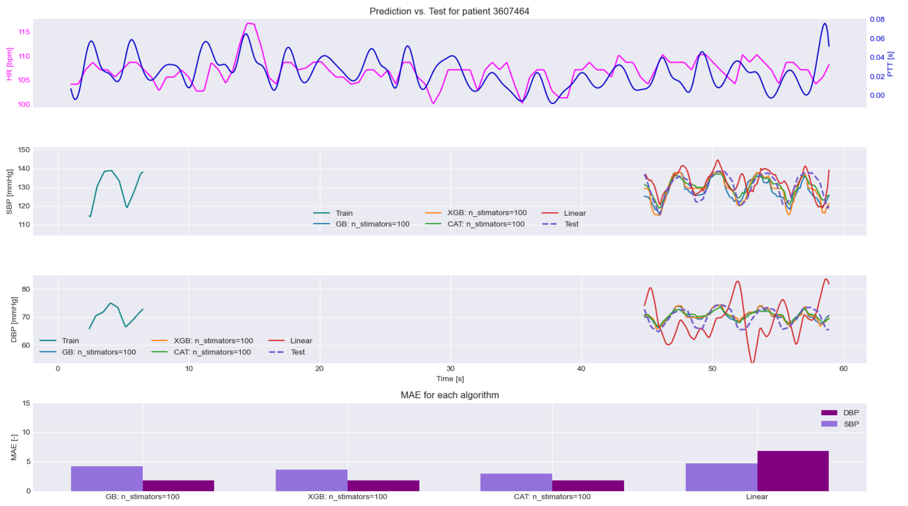


Figure 4.1: Training first (10%) of original training set and test last (25%) of data set .

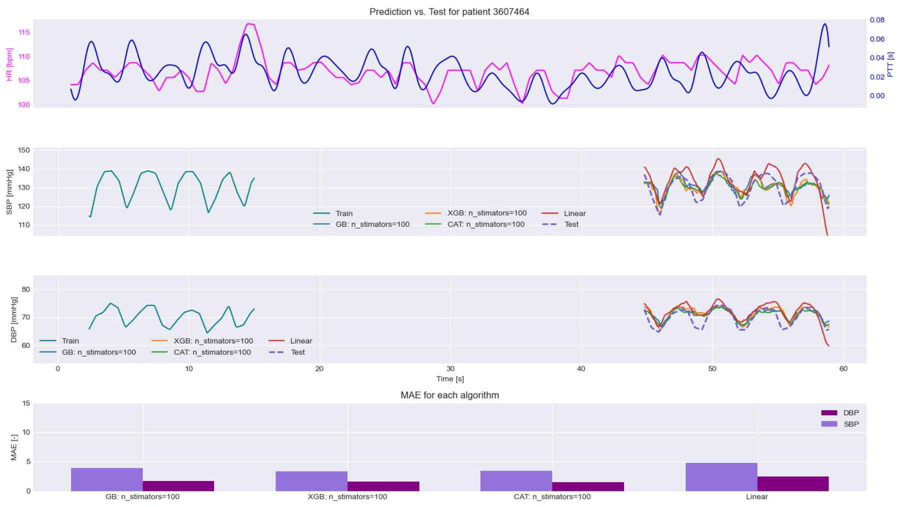


Figure 4.2: Training first (30%) of original training set and test last (25%) of data set .

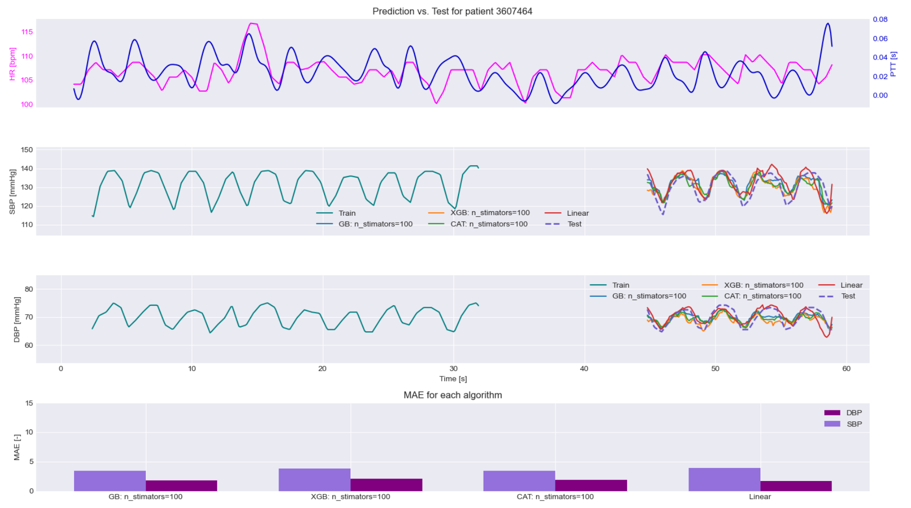


Figure 4.3: Training first (70%) of original training set and test last (25%) of data set.

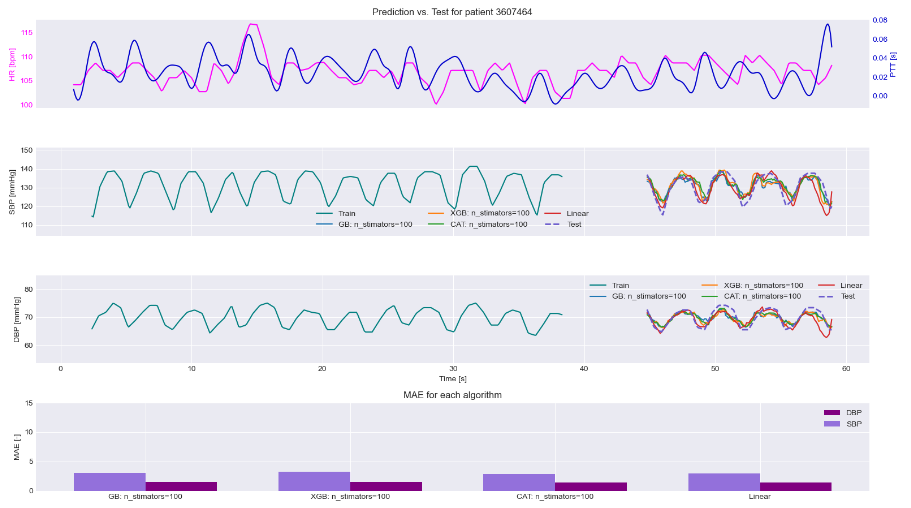


Figure 4.4: Training first (85%) of original training set and test last (25%) of the data set.

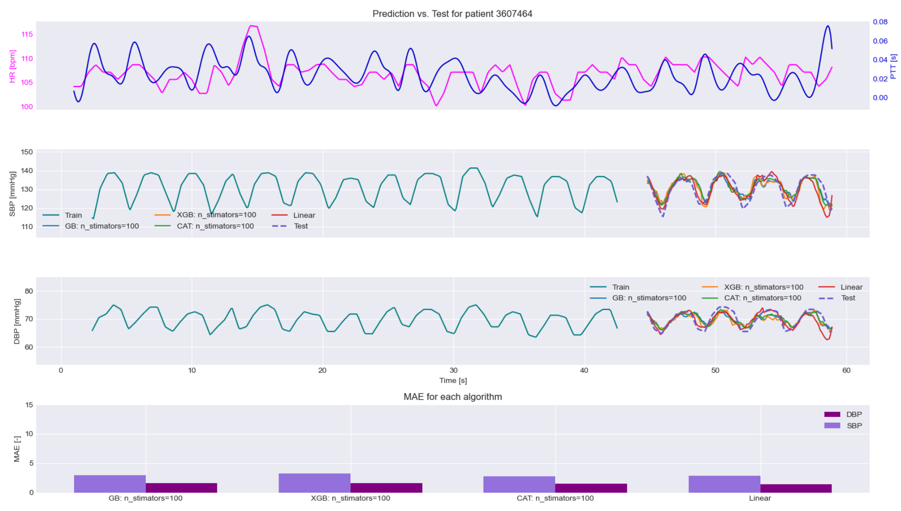


Figure 4.5: Training of first (95%) of original training set and test last (25%) of the data set.

Chapter 5

Conclusion

The high prevalence of hypertension and its association with negative cardiovascular outcomes necessitates reliable and accurate methods for monitoring blood pressure. The importance of non-invasive technologies for tracking health parameters, especially hypertension, cannot be overstated. The purpose of this thesis was to identify an approach for non-invasive monitoring of blood pressure. This goal was accomplished by pre-processing electrocardiogram (ECG) and photoplethysmography (PPG) signals and utilizing boosting regression techniques with related features, namely heart rate and pulse transit time.

The study revealed that the boosting algorithms provide BP values with an acceptable error rate (in accordance with the AAMI/ISO/ESH guidelines) using a large number of measurements taken from the MIMIC III online database[64].

Previous research has demonstrated that similar processing of electrocardiogram (ECG), photoplethysmograph (PPG), and arterial blood pressure (ABP) signals extracted from MIMIC III database can be performed[52], estimation of blood pressure values with an error of less than 5mmHg are returned, but slower and with longer calibration time.

5.1 Future developments

There are aspects that could be improved.

Comparison with other non-invasive methods:
While the proposed method is shown to have an acceptable low error, it would be useful to compare it with other non-invasive methods of continuous blood pressure monitoring, such as pulse wave analysis and photoplethysmography. This will help to determine which method is the most reliable and accurate.

Long-term monitoring:
Continuous monitoring of blood pressure over a long period of time is important for

the prevention of cardiovascular diseases. Future research could focus on developing methods that can reliably monitor blood pressure over a longer period of time, without causing discomfort or damage to the patient.

Clinical trials:
In order to validate the effectiveness of the proposed method in a clinical setting, clinical trials could be conducted. This will help to determine the usability, reliability, and accuracy of the proposed method and its potential for wider adoption in clinical practice.

Investigating the impact of physical activity:
As mentioned before, continuous monitoring of blood pressure when used in combination with other vital parameters such as heart rate, breath frequency, and physical activity, can be highly effective in the prevention of CVDs. Therefore, future research could investigate the impact of physical activity on the accuracy of the proposed method. This could help to determine whether the method is equally reliable during periods of physical activity and whether adjustments to the algorithm are needed to improve accuracy.

Further development of the algorithm:
While the proposed method is shown to have an acceptable low error, future research could focus on the further development of the algorithm to improve accuracy. This could involve the incorporation of additional physiological signals or the use of more advanced machine learning techniques.

References

[1] Annualreport-SINTEC".In:2020.

[2] Links Foundation. (2021). Chi siamo: Fondazione Links. Retrieved November, 2021, from https://linksfoundation.com/chi-siamo/fondazione-links/.

[3] Park, S., and Jayaraman, S. (2003). Enhancing the quality of life through wearable technology. IEEE Engineering in Medicine and Biology Magazine, 22(3), 41-48.

[4] Lymberis, A. (2003). Smart wearables for remote health monitoring, from prevention to rehabilitation: current RandD, future challenges. In Proceedings of the 4th International IEEE EMBS Special Topic Conference on Information Technology Applications in Biomedicine (pp. 272-275). IEEE.

[5] Lehman, H., Li-wei, H.,and et al. (2013). Methods of blood pressure measurement in the ICU. Critical Care Medicine, 41(1), 34.

[6] Franklin, S. S., Wong, N. D. (2013). Hypertension and cardiovascular disease:Contributions of the Framingham Heart Study. Global Heart, 8(1), 49 57.

[7] Sawicka,K., Szczyrek, M., Kucharska, A. Z. & Owczarek, A. (2011). Hypertension – the silent killer. Journal of Pre-Clinical and Clinical Research, 5(2).

[8] Rahimi, K.; Emdin, C.A.; MacMahon, S. The Epidemiology of Blood Pressure and Its Worldwide Management. Circ. Res. 2015, 116, 925–936. https://www.ahajournals.org/doi/full/10.1161/circresaha.116.304723

[9] Martínez, G.; Howard, N.; Abbott, D.; Lim, K.; Ward, R.; Elgendi, M. Can Photoplethysmography Replace Arterial Blood Pressure in the Assessment of Blood Pressure? J. Clin. Med. 2018, 7, 316. https://doi.org/10.3390/jcm7100316.

[10]] Shimbo, D.; Abdalla, M.; Falzon, L.; Townsend, R.R.; Muntner, P. Role of Ambulatory and Home Blood Pressure Monitoring in Clinical Practice: A Narrative Review. Ann. Intern. Med. 2015, 163, 691–700. https://doi.org/10.7326/M15- 1270.

[11]] Sun, Z. Aging, arterial stiffness, and hypertension. Hypertension 2015, 65, 252–256. https://doi.org/10.1161/hypertensionaha.114.03617.

[12] Callaghan, F.J.; Geddes, L.A.; Babbs, C.F.; Bourland, J.D. Relationship between pulse-wave velocity and arterial elasticity. Med. Biol. Eng. Comput. 1986, 24, 248–254. https://doi.org/10.1007/bf02441620.

[13]] Chen, W.; Kobayashi, T.; Ichikawa, S.; Takeuchi, Y.; Togawa, T. Continuous estimation of systolic blood pressure using the pulse arrival time and intermittent calibration. Med. Biol. Eng. Comput. 2000, 38, 569–574. https://doi.org/10.1007/BF02345755.

[14]] Chen, W.; Kobayashi, T.; Ichikawa, S.; Takeuchi, Y.; Togawa, T. Continuous estimation of systolic blood pressure using the pulse arrival time and intermittent calibration. Med. Biol. Eng. Comput. 2000, 38, 569–574. https://doi.org/10.1007/BF02345755.

[15] Mukherjee, R.; Ghosh, S.; Gupta, B.; Chakravarty, T. A literature review on current and proposed technologies of noninvasive blood pressure measurement. Telemed. e-Health 2017, 24, 185–193.

[16] Solà, J.; Delgado-Gonzalo, R. Pulse arrival time techniques. In The Handbook of Cuffless Blood Pressure Monitoring; Delgado- Gonzalo, J.S.R., Ed.; Springer Nature: Berlin/Heidelberg, Germany, 2019; pp. 43–59.

[17] Sharman, J. E.,and Marwick, T. H. (2019). Accuracy of blood pressure monitoring devices: A critical need for improvement that could resolve discrepancy in hypertension guidelines.

Journal of Human Hypertension, 33(2), 89-93.

[18] Funke, K., Parry, M., Crouch, R., Duffin, D., Haverkamp, M.,and Hoeper, K. (2004). Continuous noninvasive blood pressure measurement by pulse transit time. In Proceedings of the 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (pp. 738-741). IEEE.

[19] In:Visitedinnovember2021. url: https://cordis.europa.eu/nproject/id/824984/reporting.

[20] Binkley,P. F. (2003). Predicting the potential of wearable technology. IEEE Engineering in Medicine and Biology Magazine, 22(3), 23-27.

[21] Zoschke, K., Osmani, V., Dang, T., & Paradiso, J. A. (2020). Flexible and Stretchable Systems for Healthcare and Mobility. In Flexible, Wearable, and Stretchable Electronics (pp. 269-282). CRC Press.

[22] Rahman, Z., Alam, M., Hossain, M. A., & Islam, M. S. (2019). Smart Health Monitoring with On-demand Data Acquisition and Analysis. In Proceedings of IEEE 5th International Conference on Computer and Communications (pp. 347-351). Retrieved from https://www.researchgate.net/publication/334586404 $_Smart\_Health\_Monitoring\_with\_Ondemand\_Data\_Acquisition\_and\_Analysis$.

[23] Rahman, Z., Bhuiyan, M. I. H., Islam, M. S., Hossain, M. A. (2019). Smart health monitoring with on-demand data acquisition and analysis. In Proceedings of IEEE 5th International Conference on Computer (pp. 430 434). Retrieved from https://www.researchgate.net/publication/ 334586404$_Smart_ealth_Monitoring_with_Ondemand_Data_Acquisition_and_Analysis$.

[24] Ding, X.; Yan, B.P.; Zhang, Y.; Liu, J.; Zhao, N.; Tsang, H.K. Pulse Transit Time Based Continuous Cuffless Blood Pressure Estimation: A new extension and a comprehensive evaluation. Sci. Rep. 2017, 7, 1–11.

[25] Chen, W.; Kobayashi, T.; Ichikawa, S.; Takeuchi, Y.; Togawa, T. Continuous estimation of systolic blood pressure using the pulse arrival time and intermittent calibration. Med. Biol. Eng.

Comput. 2000, 38, 569–574. https://doi.org/10.1007/BF02345755.

[26]]. Simjanoska, M.; Gjoreski, M.; Gams, M.; Bogdanova, A.M. Non-Invasive Blood Pressure Estimation from ECG Using Machine Learning Techniques. Sensors 2018, 18, 1160. https://doi.org/10.3390/s18041160

[27] Gesche, H.; Grosskurth, D.; Küchler, G.; Patzak, A. Continuous blood pressure measurement by using the pulse transit time: Comparison to a cuff-based method. Eur. J. Appl. Physiol. 2012, 112, 309–315. https://doi.org/10.1007/s00421-011-1983-3.

[28] Wong, M.Y.; Poon, C.C.; Zhang, Y.T. An evaluation of the cuffless blood pressure estimation based on pulse transit time technique: A half year study on normotensive subjects. Cardiovasc. Eng. 2009, 9, 32–38. https://doi.org/10.1007/s10558-009-9070-7.

[29] WPoon, C.; Zhang, Y. Cuff-less and noninvasive measurements of arterial blood pressure by pulse transit time. In Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference, Shanghai, China, 17–18 January 2006

[30] Wang, R.; Jia, W.; Mao, Z.H.; Sclabassi, R.J.; Sun, M. Cuff-Free Blood Pressure Estimation Using Pulse Transit Time and Heart Rate. In Proceedings of the 12th International Conference on Signal Processing (ICSP), Hangzhou, China, 19–23 October 2014. https://doi.org/10.1109/ICOSP.2014.7014980

[31] American Heart Association, Inc., Campbell, N. R. C., Izzo Jr, J. L., Lichtenstein, A. H. (2017). What is high blood pressure? South Carolina State Documents Depository.

[32] Esmaili, A.; Kachuee, M.; Shabany, M. Nonlinear Cuff-less Blood Pressure Estimation of Healthy Subjects Using Pulse Transit Time and Arrival Time. IEEE Trans. Instrum. Meas. 2017, 66, 3299–3308. https://doi.org/10.1109/TIM.2017.2745081.

[33] Moharram, M.; Wilson, L.; Williams, M.; Coffey, S. Beat-to-beat blood pressure measurement using a cuffless device does not accu-rately reflect invasive blood pressure. Int. Cardiol. Hypertens. 2020, 5, 100030.

https://doi.org/10.1016/j.ijchy.2020.100030.

[34] Sharma, M.; Barbosa, K.; Ho, V.; Griggs, D.; Ghirmai, T.; Krishnan, S.; Hsiai, T.; Chiao, J.C.; Cao, H. Cuff-Less and Continuous Blood Pressure Monitoring: A Methodological Review. Technologies 2017, 5, 21.

[35] Mukherjee, R.; Ghosh, S.; Gupta, B.; Chakravarty, T. A literature review on current and proposed technologies of noninvasive blood pressure measurement. Telemed. e-Health 2017, 24, 185–193.

[36] Rajni, R.; Inderbir, K. Electrocardiogram Signal Analysis—An Overview. Int. J. Comput. Appl. 2013, 84, 22–25. https://doi.org/10.5120/14590-2826. [37] Yu, Q.; Liu, A.; Liu, T.; Mao, Y.; Chen, W.; Liu, H. ECG R-wave peaks marking with simultaneously recorded continuous blood pressure. PLoS ONE 2019, 14, e0214443. https://doi.org/10.1371/journal.pone.0214443

[37] Yu, Q.; Liu, A.; Liu, T.; Mao, Y.; Chen, W.; Liu, H. ECG R-wave peaks marking with simultaneously recorded continuous blood pressure. PLoS ONE 2019, 14, e0214443. https://doi.org/10.1371/journal.pone.0214443.

[38] Allen, J. (2007). Photoplethysmography and its application in clinical physiological measurement. Physiological measurement, 28(3), R1.

[39] Kavya, R.; Nayana, N.; Karangale, K.B.; Madhura, H.; Sheela, S.J. Photoplethysmography—A Modern Approach and Applications. In Proceedings of the International Conference for Emerging Technology (INCET), Belgaum, India, 5–7 June 2020; pp. 1– 4. https://doi.org/10.1109/IN-CET49848.2020.9154139.

[40]] Ding, X.R.; Zhang, Y.T. Photoplethysmogram intensity ratio: A potential indicator for improving the accuracy of PTT-based cuffless blood pressure estimation. In Proceedings of the 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Milan, Italy, 25–29 August 2015. https://doi.org/10.1109/EMBC.2015.7318383.

[41] NEWS Medical Life Sciences: Photoplethysmography (PPG). Available online: https://www.news-medical.net/health/Photoplethysmography-(PPG).aspx (accessed on 1 October 2020).

[42] Lee, C.; Shin, H.S.; Lee, M. Relations between ac-dc components and optical path length in photoplethys-mography. J. Biomedical. Opt. 2011, 16, 077012. https://doi.org/10.1117/1.3600769.

[43] Allen, J. Photoplethysmography and its application in clinical physiological measurement. Physiol. Meas. 2007, 28, R1.

[44] Nichols, W.W.; Denardo, S.J.; Wilkinson, I.B.; McEniery, C.M.; Cockcroft, J.; ORourke, M.F. Effects of Arterial Stiffness, Pulse Wave Velocity, and Wave Reflections on the Central Aortic Pressure Waveform. J. Clin. Hypertens. 2008, 10, 295–303. https://doi.org/10.1111/j.1751-7176.2008.04746.x.

[45] Johnson, A.E.; Pollard, T.J.; Shen, L.; Lehman LW, H.; Feng, M.; Ghassemi, M.; Mark, R.G. MIMIC-III, a freely accessible critical care database. Sci. Data 2016, 3, 1–9. Available online: http://www.nature.com/articles/sdata201635 (accessed on 1 September2020).

[46] Mukkamala, R.; Hahn, J.O.; Inan, O.T.; Mestha, L.K.; Kim, C.S.; Töreyin, H.; Kyal, S. Toward Ubiquitous Blood Pressure Monitoring via Pulse Transit Time: Theory and Practice. IEEE Trans. Biomed. Eng. 2015, 62, 1879–1901. https://doi.org/10.1109/TBME.2015.2441951.

[47] Proença, J.; Muehlsteff, J.; Aubert, X.; Carvalho, V. Is pulse transit time a good indicator of blood pressure changes during short physical exercise in a young population? Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. 2010, 2010, 598–601. https://doi.org/10.1109/IEMBS.2010.5626627.

[48] , Y.; Zhang, Y. A novel calibration method for non-invasive blood pressure measurement using pulse transit time. In Proceedings of the 4th IEEE/EMBS International Summer School and Symposium on Medical Devices and Biosensors, Cambridge, UK, 19–22 August 2007.

[49] PhysioNet. (n.d.). PhysioBank ATM. Retrieved September 1,

2020, from https://archive.physionet.org/cgi-bin/atm/atm

[50] Barlett, M.S. Periodogram analysis and continuous spectra. Biometrika 1950, 37, 1–16.

[51] Lazazzera, R.; Belhaj, Y.; Carrault, G. A New Wearable Device for Blood Pressure Estimation Using Photoplethysmogram. Sensors 2019, 19, 2557. https://doi.org/10.3390/s19112557

[52] Figini, V., Galici, S., Russo, D., Centonze, I., Visintin, M.,and Pagana, G. (2022). Improving Cuff-Less Continuous Blood Pressure Estimation with Linear Regression Analysis. Electronics, 11(9), 1442.

[53] , Umang and Abbas, Sherif N and Hatzinakos, Dimitrios. Evaluation of PPG biometrics for authentication in different states. In Proceedings of the 2018 International Conference on Biometrics (ICB), Gold Coast, QLD, Australia, 20–23 February 2018.

[54] Galici, S. (2022). Blood pressure monitoring in a non-invasive way using Regression techniques (Doctoral dissertation, Politecnico di Torino).

[55] Seber, G. A. F.,and Lee, A. J. (2012). Linear regression analysis. Vol. 329. John Wiley Sons.

[56] Montgomery, D. C., Peck, E. A., and Vining, G. G. (2021). Introduction to linear regression analysis. John Wiley Sons.

[57] Kearns, M. (1988). Thoughts on hypothesis boosting, ML class project.

[58] Freund, Y., Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In Computational Learning Theory: Second European Conference, EuroCOLT'95 Barcelona, Spain, March 13–15, 1995 Proceedings 2 (pp. 23-37). Springer Berlin Heidelberg

[59] Breiman, L. (1999). Prediction games and arcing algorithms. Neural computation, 11(7), 1493-1517.

[60] Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho,

H., ...and Zhou, T. (2015). Xgboost: extreme gradient boosting. R package version 0.4-2, 1(4), 1-4.

[61] Chen, T., Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794).

[62] Dorogush, A. V., Ershov, V.,and Gulin, A. (2018). CatBoost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363.

[63] Wu, L., Huang, G., Fan, J., Zhang, F., Wang, X., and Zeng, W. (2019). Potential of kernel-based nonlinear extension of Arps decline model and gradient boosting with categorical features support for predicting daily global solar radiation in humid regions. Energy Conversion and Management, 183, 280-295.

[64] Stergiou, G.S.; Alpert, B.; Mieke, S.; Asmar, R.; Atkins, N.; Eckert, S.; Frick, G.; Friedman, B.; Grassl, T.; Ichikawa, T.; et al. A universal standard for the validation of blood pressure measuring devices: Association for the Advancement of Medical Instrumentation/ European Society of Hypertension/ International Organization for Standardization (AAMI/ESH/ISO) Collaboration Statement. J. Hypertens. 2018, 36, 472–478. https://doi.org/10.1097/HJH.0000000000001634.

[65] Vernier: What Are Mean Squared Error and Root Mean Squared Error? Available online: https://www.vernier.com/til/1014 (accessed on 1 November 2020).

[66] Wang, W.; Lu, Y. Analysis of the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) in Assessing Rounding Model. IOP Conf. Ser. Mater. Sci. Eng. 2018, 324, 012049. https://doi.org/10.1088/1757-899X/324/1/012049.

[67] Guidelines on medical devices. European Commission Dg Internal Market, Industry, Entrepreneurship and SMEs Consumer, Environmental and Health Technologies Health technology and Cosmetics (June 2016).

Available online: https://ec.europa. eu/docsroom/documents/17522/attachments/1/translations/en/renditions/native (accessed on 1 January 2022).

[68] Stergiou, G. S., Alpert, B., Mieke, S., Asmar, R., Atkins, N., Eckert, S., Frick, G., Friedman, B., Grassl, T., Ichikawa, T., Ioannidis, J. P., Lacy, P., McManus, R. J., Murray, A., Myers, M.

[69] Jaiswal, S. (2021, April 15). Introduction to XGBoost Algorithm. Analytics Vidhya. Retrieved March 11, 2023, from https://medium.com/analytics-vidhya/introduction-to-xgboost-algorithm-d2e7fad76b04G., Palatini, P., Parati, G., Quinn, D., Sarkis, J., Shennan, A., Usuda, T., Wang, J., and O'Brien, E. (2018). A universal standard for the validation of blood pressure measuring devices: Association for the Advancement of Medical Instrumentation/European Society of Hypertension/International Organization for Standardization (AAMI/ESH/ISO) Collaboration Statement. Hypertension, 71(3), 368-374.

[70] neptune Krasnikov, D. (2021, January 27). XGBoost: Everything you need to know. Neptune. Retrieved March 11, 2023, from https://neptune.ai/blog/xgboost-everything-you-need-to-know.

[71] kaggle Kaggle. (2019, July 15). How to analyze the frequency of text data? [Question and Answer]. Retrieved March 11, 2023, from https://www.kaggle.com/questions-and-answers/302677.

[72] medium Jaiswal, S. (2021, April 15). Introduction to XGBoost Algorithm. Analytics Vidhya. Retrieved March 11, 2023, from https://medium.com/analytics-vidhya/introduction-to-xgboost-algorithm-d2e7fad76b04.

[73] Python Geeks. (2022, March 5). Working of Gradient Boosting Algorithm. Python Geeks. https://pythongeeks.org/working-of-gradient-boosting-algorithm/

[74] Machine Learning Mastery. (2019, January 10). Basic feature engineering on time series data with Python. Retrieved from https://machinelearningmastery.com/basic-feature-engineering-time-series-data-python/.

[75] Mishra, A., Chakraborty, B., Das, D., Bose, P. (2018). AD8232 based smart healthcare system using internet of things (IoT). Int. J. Eng. Res. Technol.(IJERT), 7(4), 13-16.

[76] Taha, Z., Shirley, L., & Razman, M. A. M. (2017). A review on non-invasive hypertension monitoring system by using photoplethysmography method. Movement, Health Exercise, 6(1), 47-57.

[77] Yandex. (2017). CatBoost. CatBoost Documentation. https://catboost.ai/docs/.

[78] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., Gulin, A. (2018). CatBoost: Unbiased boosting with categorical features. In Proceedings of the 32nd International Conference on Neural Information Processing Systems (pp. 6638-6646). Curran Associates, Inc.

[79] Schmitz, E. A., and Remington, R. A. (2018). Hyperparameter Tuning in Boosting-Based Regression. Journal of Machine Learning Research, 19(30), 1-33.

[80] López García, D., Rodríguez González, A. B., and González Sánchez, A. (2017). Hyperparameter Tuning for Gradient Boosting Machines Using Grid Search. Lecture Notes in Computer Science, 10586, 180-191.

[81] Sharma, P., and Bora, B. J. (2022). A Review of Modern Machine Learning Techniques in the Prediction of Remaining Useful Life of Lithium-Ion Batteries. Batteries, 9(1), 13.

Appendix A

This appendix displays the Python code that was used to implement the complete algorithm.

```python





# from math import nan
# from typing import final
# from numpy. core.fromnumeric import ptp
from sklearn.model_selection import GridSearchCV
# import csv
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mplt
import scipy.fft
import scipy.signal
from scipy import stats
from pprint import pprint
from sklearn.linear_model import Ridge
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error,
    mean_squared_error

from sklearn.model_selection import train_test_split,
    cross_val_score
from sklearn import datasets, ensemble
import xgboost
from xgboost import XGBRegressor
from catboost import CatBoostRegressor

import time


# sklearn.metrics has a mean_squared_error function with a
    squared kwarg (defaults to True). Setting squared to False
     will return the RMSE.


class SintecProj(object):
        """docstring for SintecProj"""
    def __init__(self):
        self.fs = 125
        self.mmHg_thresh = [5,10]
        self.PREV_VAL = 15 # X * 0.1 = [s]
        self.patient_path = str(os.getcwd())+'\\Patients'
        self.dataset_path = str(os.getcwd())+'\\Dataset'
        self.plot_setup()
        self.save_figure = False
        self.signal_list = [
            '3001689','3001203','3000714','3515650','3516310','3510820',
            '3513879','3513631','3511504','3512125','3513230','3503726',
            '3509498','3509505','3508696','3508299','3506991','3505101',
            '3507993','3508009','3505162','3505174','3503945','3503406',
            '3503404','3502786','3403213','3700665','3700837','3703763',
            '3703856','3703872','3704307','3704658','3704803','3705715',
            '3705993','3402408','3402291','3600293','3602237','3602666',
            '3600490','3600620','3601272','3403274','3604430','3604660',
            '3604404','3605744','3904308','3603256','3604217','3607634',
            '3608436','3608706','3609155','3609182','3609463','3606882',
            '3602521','3602766','3602772','3603658','3604352','3607711',
            '3605724','3904396','3606358','3607077','3907039','3607464',
            '3606909','3609839','3800183','3800350','3900487','3901160',
            '3901339','3905772','3903282','3901654','3902124','3902445',
            '3902729','3902894','3905695','3904550','3902994','3904246']


    def create_path(self, path):
        if not os.path.exists(path):
```

```python
            os.makedirs(path)

    def plot_setup(self):
        self.figsize = (15,9)
        self.create_path("Plots")
        self.plot_path = os.getcwd()+'\\Plots'
        plt.style.use('seaborn-darkgrid')

    def plot(self, df, pat_name):
        fig, axs = plt.subplots(2,1,sharex=True)
        fig.set_size_inches(self.figsize)
        axs[0].set_ylabel('ABP [mmHg]')
        axs[1].set_ylabel('[mV]')
        df['ABP'].plot(ax=axs[0])
        plt.suptitle(f'Patient: {pat_name}')
        df[['II','PLETH']].plot(ax=axs[1])
        plt.tight_layout()
        if self.save_figure: plt.savefig(f'{self.plot_path}\\{pat_name}.png')
        plt.close()
        if pat_name in self.signal_list: self.save_df(df,pat_name)

    def save_df(self,df,pat_name):
        self.create_path("Dataset")
        df.to_csv(f'{os.getcwd()}\\Dataset\\{pat_name}.csv')

    def data_reader(self):
        for n,file in enumerate(os.listdir(self.patient_path)):
            pat_name = file.split('_')[0]
            print(f'Patient: {pat_name} - {n}\{len(os.listdir(self.patient_path))}')
            df = pd.read_csv(f'{self.patient_path}\\{file}',quotechar="'",sep=',',skiprows=[1])

            if df.iloc[0][0][0] == '"':
                df.columns = [x.replace('"',"") for x in df.columns]
                df.columns = [x.replace("'","") for x in df.columns]

                df['Time'] = df['Time'].apply(lambda x: x[3:-2])
                df[df.columns[-1]] = df[df.columns[-1]].apply(lambda x: x[:-1])
                df = df.replace('-', np.nan)
                df.index = df['Time']

                def_columns = []
                for x in df.columns:
                    if 'ABP' in x or x=='II' or 'PLETH' in x:
                        def_columns.append(x)
                df = df[def_columns]
                df = df.astype(float)
                self.plot(df,pat_name)

            else:
                df['Time'] = df['Time'].apply(lambda x: x[1:-1])
                df.index = df['Time']
                df = df.replace('-', np.nan)
                df = df[['ABP','PLETH','II']]
                df = df.astype(float)
                self.plot(df,pat_name)


    def peak_finder(self):
        self.create_path('Plots\\Peaks')
        tmp_path = self.plot_path+'\\Peaks'
        file_lst = [x for x in os.listdir(self.dataset_path) if x.endswith('.csv')]
        # file_lst = [x for x in os.listdir(self.dataset_path) if '3601140' in x]

        for file in file_lst:
            patient = file.split('.')[0]
            print(f'Patient: {patient}')
            print()
            df = pd.read_csv(f'{self.dataset_path}\\{file}').dropna()
            df.index = range(0,len(df))

            #print(df)
```

```
                        #plt.plot(df['II'])
                        #plt.xlabel('Time (s)')
                        #plt.ylabel('(mV)')
                        #plt.title(' ECG Signal')
                        #plt.show()

                        # Filtering the signal
                        b, a = scipy.signal.butter(N=5,
                                Wn=[1,10],
                                btype='band',
                                analog=False,
                                output='ba',
                                fs=125
                                )
                        ecg_filt = scipy.signal.filtfilt(b,a,df['II'])
                        ecg_diff = np.gradient(np.gradient(ecg_filt))
                        ppg_filt = scipy.signal.filtfilt(b,a,df['PLETH'])

                        #plt.plot(ecg_filt)
                        #plt.xlabel('Time (s)')
                        #plt.ylabel('(mV)')
                        #plt.title('Filtered ECG Signal')
                        #plt.show()

                        #find DBP/SBP points
                        DBPs,_ = scipy.signal.find_peaks(-df['ABP'],prominence=.5,distance=60,width=10)
                        SBPs,_ = scipy.signal.find_peaks(df['ABP'],prominence=.5,distance=60,width=10)
                        x_abp, kde_abp, kde_pks = self.gaussian_distributions(df['ABP'],np.concatenate((DBPs, SBPs), axis=None))

                        #print(SBPs)


                        #find R peaks
                        Rs,_ = scipy.signal.find_peaks(ecg_filt,distance=60)
                        Rs_diff,_ = scipy.signal.find_peaks(-ecg_diff,distance=60) #discarded because of patient 3600490
                        x_ecg, kde_ecg, kde_rs = self.gaussian_distributions(ecg_filt,Rs)
                        x_ecg1, kde_ecg1, kde_rs1 = self.gaussian_distributions(ecg_filt,Rs_diff)

                        #find SP peaks
                        SPs,_ = scipy.signal.find_peaks(ppg_filt,prominence=.05,width=10)
                        SPs_new, [kde_ppg, kde_sp, x_ppg, min_] = self.PPG_peaks_cleaner(ppg_filt, SPs)
                        # print(SPs_new)
                        if True:
                                plt.style.use('default')
                                fig, axs = plt.subplots(3,2,sharex=True)
                                fig.set_size_inches(self.figsize)
                                gs = mplt.gridspec.GridSpec(3, 2, width_ratios=[3,1])

                                # PLOT
                                axs[0,0] = plt.subplot(gs[0,0])
                                axs[0,0].plot(df['ABP'],label='ABP')
                                axs[0,0].scatter(DBPs,df['ABP'][DBPs],label='DBP',c='r')
                                axs[0,0].scatter(SBPs,df['ABP'][SBPs],label='SBP',c='g')
                                axs[0,0].set_ylabel('ABP[mmHg]')

                                # Gaussian dist. - ABP
                                axs[0,1] = plt.subplot(gs[0,1])
                                axs[0,1].plot(kde_abp(x_abp),x_abp,label='KDE of ABP')
                                axs[0,1].plot(kde_pks(x_abp),x_abp,label='KDE of ABP peaks')

                                axs[1,0] = plt.subplot(gs[1,0])
                                axs[1,0].plot(ecg_filt,label='ECG Filtered')
                                # axs[1].plot(df['II'],label='ECG')
                                axs[1,0].scatter(Rs_diff,ecg_filt[Rs_diff],label='R peaks - with gradient',s=100,c='r')
                                axs[1,0].scatter(Rs,ecg_filt[Rs],label='R peaks',c='y')
                                axs[1,0].set_ylabel('ECG [mV]')

                                # Gaussian dist. - ECG
                                axs[1,1] = plt.subplot(gs[1,1])
                                axs[1,1].plot(kde_ecg(x_ecg),x_ecg,label='KDE of ECG')
                                axs[1,1].plot(kde_rs(x_ecg),x_ecg,label='KDE of R peaks')
                                axs[1,1].plot(kde_rs1(x_ecg1),x_ecg1,label='KDE of R peaks - with gradient')

                                axs[2,0] = plt.subplot(gs[2,0])
                                axs[2,0].plot(ppg_filt,label='PPG Filtered')
                                # axs[2].plot(df['PLETH'],label='PPG')
                                axs[2,0].scatter(SPs,ppg_filt[SPs],label='SP peaks - first evalutation',s=100,c='r')
                                axs[2,0].scatter(SPs_new,ppg_filt[SPs_new],label='SP peaks - after KDE',c='y')
```

```
                axs[2,0].set_ylabel('PPG [mV]')

                # Gaussian dist. - PPG
                axs[2,1] = plt.subplot(gs[2,1])
                axs[2,1].plot(kde_ppg(x_ppg), x_ppg,label='KDE of PPG')
                axs[2,1].plot(kde_sp(x_ppg), x_ppg,label='KDE of SP peaks')

                if min_ != None:
                        axs[2,1].axhline(min_,c='red',label='Threshold')
                        axs[2,0].axhline(min_,c='red',label='Threshold')

                [axs[x,0].legend(loc='lower left', facecolor='white', framealpha=.8) for x in range(3)]
                [axs[x,1].legend(facecolor='white', framealpha=.8) for x in range(3)]
                [axs[x,1].set_yticklabels([]) for x in range(3)]

                x_ticks = np.arange(0,len(ppg_filt)+1,500)
                for x in range(3):
                        axs[x,0].set_xlabel('Time [s]')
                        axs[x,0].set_xticks(x_ticks)
                        axs[x,0].set_xticklabels((x_ticks/self.fs).astype(int))

                # print(f'ECG vector: {ecg_filt}')
                # print(f'PPG vector: {ppg_filt}')
```

```
                                plt.suptitle(f'Patient: {patient}')
                                plt.tight_layout()
                                if self.save_figure: plt.savefig(f'{tmp_path}\\{patient}')
                        plt.show()

                        dataset = self.find_PTT(ecg_filt,Rs,ppg_filt,SPs_new,patient)
                        df.index = np.array(list(df.index))/self.fs
                        print(df)
                        print(dataset)
                        dataset['DBP'] = df['ABP'].iloc[DBPs]
                        dataset['SBP'] = df['ABP'].iloc[SBPs]

                        print(dataset)
                        regr_path = self.dataset_path+'\\Regression'
                        self.create_path(regr_path)
                        dataset.to_csv(f'{regr_path}\\{patient}.csv')

                        dff = pd.read_csv(f'{regr_path}\\{patient}.csv')
                        print(dff)

        def gaussian_distributions(self,curve,peaks):
                x = np.arange(min(curve),max(curve),.001)
                kde_curve = stats.gaussian_kde(curve)
                kde_peaks = stats.gaussian_kde(curve[peaks])
                return x, kde_curve, kde_peaks

        def PPG_peaks_cleaner(self, ppg, SP):
                check_plot = False

                x_ppg, kde_ppg, kde_sp = self.gaussian_distributions(ppg, SP)
                if check_plot: plt.figure()
                peak_sp,_ = scipy.signal.find_peaks(kde_sp(x_ppg))
                n_peaks = len(peak_sp)
                minimum = None

                if n_peaks == 2:
```

```
                # sp_idx = np.argmin(kde_sp(x_ppg)[peak_sp])
                minimum = scipy.signal.find_peaks(-kde_sp(x_ppg)[peak_sp[0]:peak_sp[1]])
                minimum = x_ppg[peak_sp[0]:peak_sp[1]][minimum[0]][0]
                if minimum < .90*max(x_ppg):
                    SP = [x for x in SP if ppg[x] > minimum]
                    if check_plot:
                        plt.plot(x_ppg, kde_sp(x_ppg))
                        plt.plot(x_ppg[peak_sp[0]:peak_sp[1]],kde_sp(x_ppg)[peak_sp[0]:peak_sp[1]])
                        plt.plot(x_ppg[peak_sp[0]:peak_sp[1]],-kde_sp(x_ppg)[peak_sp[0]:peak_sp[1]])
                        plt.axvline(x_ppg[peak_sp[0]],ls='--',label='1st peak')
                        plt.axvline(x_ppg[peak_sp[1]],ls='-.',label='2nd peak')
                        plt.axvline(minimum, label='Minimum')
                        plt.legend()
                else: minimum = None


        curves = [kde_ppg, kde_sp, x_ppg, minimum]
        if check_plot: plt.show()

        return SP, curves

    def find_PTT(self,ECG,ECG_peaks,PPG,PPG_peaks,patient):
        #ECG_peaks,PPG_peaks: vectors containig indices of peaks
        #ECG,PPG: vectors containig ECG/PPG curves
        plt.style.use('seaborn-darkgrid')
        self.create_path('Plots\\HR and PTT')
        tmp_path = self.plot_path+'\\HR and PTT'

        #transofrm in time series:
        ecg_TS = np.array(ECG_peaks)/self.fs
        ppg_TS = np.array(PPG_peaks)/self.fs
        # print(f'ECG in seconds:{ecg_TS}')
        # print(f'PPG in seconds:{ppg_TS}')

        #HR evaluation:
        HR = 60/(ecg_TS[1::] - ecg_TS[0:-1])
        # print(f'HR:{HR}')

        plt.close('all')
        fig, axs = plt.subplots(2,1,sharex=True)
        fig.set_size_inches(self.figsize)

        axs[0].plot(ecg_TS[1:],HR,label='Heart Rate')
        axs[0].set_title(f'HR and SP peaks cleaning for patient: {patient}')

        #HR cleaning:
        LEN_WDW = int(len(HR)/5)
        for x in range(10):
                HR_tmp = HR[int(LEN_WDW*x*.5):int(LEN_WDW*(1+x*.5))]
                if np.std(HR_tmp) > 3:
                        up_bound, low_bound = np.mean(HR_tmp)+np.std(HR_tmp), np.mean(HR_tmp)-np.std(HR_tmp)
                        # axs[0].axhline(np.mean(HR_tmp),c='r',lw=4, label='Mean Value')
                        axs[0].fill_between(ecg_TS[1:][int(LEN_WDW*x*.5):int(LEN_WDW*(1+x*.5))], low_bound, up_bound, alpha=0.15, color='tab:red', lw=4)
```

```
                                # else: axs[0].fill_between(ecg_TS[int(LEN_WDW*x*.5):int(LEN_WDW*(1+x*.5))], low_bound, up_bound, alpha=0.15, color='tab:red', lw=4)
                                nan_idx = np.concatenate((np.argwhere(HR_tmp<=low_bound),np.argwhere(HR_tmp>=up_bound)))
                                nan_idx = list([int(LEN_WDW*x*.5)+y[0] for y in nan_idx])
                                HR[nan_idx] = np.nan
                HR = pd.DataFrame(HR).interpolate(method='polynomial',order=5)
                axs[0].plot(ecg_TS[1:],HR.values.tolist(),label='HR - cleaned',c='g')
                axs[0].legend()
                axs[0].set_ylabel('HR [mmHg]')
                # print(f'HR: {HR}')

                #PTT evaluation:
                time = np.arange(0,max(max(ECG_peaks),max(PPG_peaks)),1)
                #print(time)
                real_time = time/self.fs

                y = time*0
                for k in time:
                        if time[k] in ECG_peaks:
                                y[k]=1
                        if time[k] in PPG_peaks:
                                y[k]=2

                index = np.argwhere(y>0).flatten()
                yy = list(y[index])
                time1 = time[index]

                results = []
                b = [1,2]
                results = [i for i in range(len(yy)) if yy[i:i+len(b)] == b]
                index_rf = time1[np.array(results)]
                index_spf = time1[np.array(results)+1]

```

```
        time_rf = real_time[index_rf]
        time_spf = real_time[index_spf]

        # rf_diff = np.diff(time_rf)
        ptt = time_spf - time_rf

        #PTT cleaning:
        axs[1].hlines(ptt, xmin=real_time[index_rf], xmax=real_time[index_spf], colors='tab:green', linestyles='solid', label='ptt')
        # print(f'PTT: {pd.DataFrame(ptt)}')
        for x in range(10):
                PTT_tmp = ptt[int(LEN_WDW*x*.5):int(LEN_WDW*(1+x*.5))]
                # print(PTT_tmp)
                if np.std(PTT_tmp) > .05:
                        print(np.std(PTT_tmp))
                        up_bound, low_bound = np.mean(PTT_tmp)+np.std(PTT_tmp), np.mean(PTT_tmp)-np.std(PTT_tmp)
                        # axs[0].axhline(np.mean(HR_tmp),c='r',lw=4, label='Mean Value')
                        # axs[1].fill_between(ecg_TS[1:][int(LEN_WDW*x*.5):int(LEN_WDW*(1+x*.5))], low_bound, up_bound, alpha=0.15, color='tab:red', lw=4)
                        nan_idx = np.concatenate((np.argwhere(PTT_tmp<=low_bound),np.argwhere(PTT_tmp>=up_bound)))
                        nan_idx = list([int(LEN_WDW*x*.5)+y[0] for y in nan_idx])
                        ptt[nan_idx] = np.nan
        TEMP = pd.DataFrame(columns=['Before','After'])
        TEMP['Before'] = ptt
        # print(f'Before: {ptt}')
        ptt = pd.DataFrame(ptt).interpolate(method='polynomial',order=5)
        TEMP['After'] = ptt

        axs[1].set_ylabel('PTT [s]')
        axs[1].plot(ecg_TS,ECG[ECG_peaks],'o-',label='R peaks')
        axs[1].plot(real_time[index_rf],ECG[index_rf],'o-',label='R peaks - newly found')
        axs[1].plot(ppg_TS,PPG[PPG_peaks],'o-',label='SP peaks')
        axs[1].plot(real_time[index_spf],PPG[index_spf],'o-',label='SP peaks - newly found')
        axs[1].hlines(ptt, xmin=real_time[index_rf], xmax=real_time[index_spf], colors='tab:red', linestyles='solid', label='ptt - newly found')
        axs[1].legend()
        plt.tight_layout()
        if self.save_figure: plt.savefig(f'{tmp_path}\\{patient}')

        tmp_df_hr = pd.DataFrame(HR)
        tmp_df_hr.index = ecg_TS[1:]

        tmp_df_ptt = pd.DataFrame(ptt)
        tmp_df_ptt.index = real_time[index_rf]


        df = pd.DataFrame({'Time':real_time})

        df.index = df['Time']
        #rint(df)

        df['HR'] = tmp_df_hr
        #print(df)
        df['PTT'] = tmp_df_ptt
        #print(df)

        return df.drop('Time',axis=1)

    def regression_process(self):
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.linear_model import LinearRegression
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.datasets import make_regression

        TRAIN_PERC = .75
        regr_path = 'Dataset\\Regression'
        dbp_errors, sbp_errors = pd.DataFrame(), pd.DataFrame()
        file_lst = os.listdir(regr_path)
        final_dict_dbp, final_dict_sbp = {}, {}
        # file_lst = file_lst[40::]
        # file_lst = [x for x in os.listdir(regr_path) if '3601140' in x]
        for file in file_lst:
            patient = file.split('.')[0]
            final_dict_sbp.update({patient:{}})
            final_dict_dbp.update({patient:{}})
            print(f'Patient: {patient}')

            fig, axs = plt.subplots(2,1,sharex=True)
            fig.set_size_inches((16,9))

            df = pd.read_csv(regr_path+'\\'+file).set_index('Time')
            df = df.dropna(how='all')
            #print(df)
            x_final = np.arange(0, 60,.1)
            for i in x_final:
                try:
                    df.loc[i]
                except:
                    df.loc[i] = [np.nan,np.nan,np.nan,np.nan]
            df = df.sort_values(by='Time')


            df[['HR','SBP','DBP']].plot(style='o', ax=axs[0])
            df[['PTT']].plot(style='o', ax=axs[1])
            df[['HR','SBP','DBP']] = df[['HR','SBP','DBP']].interpolate(method='polynomial',order=1)
            df['PTT'] = df['PTT'].interpolate(method='polynomial',order=5)


```

```
                        df = df.loc[x_final].dropna()
                        print(df)

                        [axs[0].plot(df[x],'*',alpha=.4,label=y) for x,y in zip(['HR','SBP','DBP'],['HR - resampled','SBP - resampled','DBP - resampled'])]
                        axs[1].plot(df['PTT'],'*',alpha=.4,label='PTT - resampled')
                        [axs[i].legend() for i in range(2)]
                        axs[1].set_xlabel('Time [s]')


                        plt.tight_layout()
                        self.create_path('Plots\\interpolation')
                        if self.save_figure: plt.savefig(f'Plots\\interpolation\\{patient}.png')
                        plt.show()

                        plt.close()

                        #REGRESSION
                        fig, axs = plt.subplots(4,1)
                        fig.set_size_inches((16,9))
                        axs[1].sharex(axs[0])
                        axs[2].sharex(axs[0])

                        train_cols = ['HR','PTT']
                        axs[0].set_ylabel('HR [bpm]', color='tab:red')
                        axs[0].plot(df['HR'],c='tab:red')
                        axs[0].tick_params(axis='y', labelcolor='tab:red')
                        axs_b = axs[0].twinx()
                        axs_b.set_ylabel('PTT [s]', color='tab:blue')
                        axs_b.plot(df['PTT'],c='tab:blue')
                        axs_b.tick_params(axis='y', labelcolor='tab:blue')
                        [x.grid() for x in [axs[0], axs_b]]

                        for x in train_cols:
                          for y in range(1,self.PREV_VAL):
                                df[f'{x}-{y}'] = df[x].shift(y)
                        df = df.dropna()
                        print(df.head(20))
                        df['ones'] = np.ones(len(df))

                        train_cols = ['HR','PTT','ones']
                        for x in range(1,self.PREV_VAL):
                                train_cols.append(f'HR-{x}')
                                train_cols.append(f'PTT-{x}')
                        # final_cols = df.columns

                        # f = scipy.signal.resample(df, 550)
                        # beg,end = df.index[0],df.index[-1]
                        # xnew = np.linspace(beg,end, 550, endpoint=True)
                        # df = pd.DataFrame(f)
                        # df.index = xnew
                        # x_final = np.arange(5, 60,.1)
                        # tmp_df = pd.DataFrame(np.nan, index=x_final, columns=df.columns)
                        # df = df.append(tmp_df)
                        # df = df.sort_index().interpolate(method='polynomial',order=3)
                        # df = df.loc[x_final].dropna()
                        # df.columns = final_cols

                        #print(df)
                        #dfs=int(.10*len(df.index))
                        #df=df.iloc[0:dfs]
                        print(df)

                        #DBP Prediction
                        test_size = int(TRAIN_PERC*len(df.index))
                        calib_size=int(1* test_size)
                        X_train_dbp,y_train_dbp = df[train_cols].iloc[0:calib_size], df['DBP'].iloc[0:calib_size]
                        X_test_dbp,y_test_dbp = df[train_cols].iloc[test_size::], df['DBP'].iloc[test_size::]

                        X_dbp,y_dbp=df[train_cols] , df['DBP']

                        #SBP Prediction
                        X_train_sbp,y_train_sbp = df[train_cols].iloc[0:calib_size], df['SBP'].iloc[0:calib_size]
                        X_test_sbp,y_test_sbp = df[train_cols].iloc[test_size::], df['SBP'].iloc[test_size::]

                        X_sbp,y_sbp=df[train_cols] , df['SBP']

                        # axs[0].plot(X_train_dbp['HR'],'o')
                        # axs[1].plot(X_train_dbp['PTT'],'o')
                        # axs[2].plot(y_train_dbp,'o')
                        # # plt.show()
                        axs[1].plot(y_train_sbp,label='Train')
                        axs[2].plot(y_train_dbp,label='Train')
                        maes_dbp, maes_sbp = [],[]
                        DEV_sbp,DEV_dbp = [],[]
                        count_dbp, count_sbp = [],[]
                        x_labs = []



#============================================================
#gradient boosting
                        nTrees=[100]
                        [x_labs.append(f'GB: n_stimators={x}') for x in nTrees]
                        for trees in nTrees:
                                regr=ensemble.GradientBoostingRegressor(n_estimators=trees,learning_rate=0.1,max_depth=5,loss='lad')
                                y_hat_dbp = self.regression(regr,y_train_dbp,X_train_dbp,X_test_dbp)


                                # y_hat_dbp = regr.predict(X_test_dbp)
                                axs[2].plot(y_test_dbp.index,y_hat_dbp,label=f'GB: n_stimators={trees}')
                                MAE_dbp = round(mean_absolute_error(y_test_dbp, y_hat_dbp),2)
                                count_dbp.append(self.count_diff(y_test_dbp, y_hat_dbp, 'GB-DBP'))
                                maes_dbp.append(MAE_dbp)
                #DEV_dbp.append(,DEV_dbp)

                                regr.fit(X_train_sbp, y_train_sbp)
                                y_hat_sbp = self.regression(regr,y_train_sbp, X_train_sbp,X_test_sbp)
                                axs[1].plot(y_test_sbp.index,y_hat_sbp,label=f'GB: n_stimators={trees}')
                                MAE_sbp = round(mean_absolute_error(y_test_sbp, y_hat_sbp),2)
                                count_sbp.append(self.count_diff(y_test_sbp, y_hat_sbp, 'GB-SBP'))
                                maes_sbp.append(MAE_sbp)
                                #DEV_sbp.append(DEV_sbp)

                                scoresg = cross_val_score(regr, X_dbp, y_dbp, cv=5)
                                scoresg = np.abs(scoresg)
                                mean_scoreg = np.mean(scoresg)
                                std_scoreg = np.std(scoresg)

                # Print the mean and standard deviation of the scores
                                print("Mean scoreg: {:.2f}".format(mean_scoreg))
                                print("Standard deviation: {:.2f}".format(std_scoreg))



```

```


#============================================================
#XG boosting
                        nTrees=[100]
                        [x_labs.append(f'XGB: n_stimators={x}') for x in nTrees]
                        for trees in nTrees:
                                regr=XGBRegressor(n_estimators=trees,learning_rate=0.1,max_depth=5,eta=0.1,subsample=0.7,colsample_bytree=0.8)
                                y_hat_dbp = self.regression(regr,y_train_dbp,X_train_dbp,X_test_dbp)


                                #y_hat_dbp = regr.predict(X_test_dbp)
                                axs[2].plot(y_test_dbp.index,y_hat_dbp,label=f'XGB: n_stimators={trees}')
                                MAE_dbp = round(mean_absolute_error(y_test_dbp, y_hat_dbp),2)
                                count_dbp.append(self.count_diff(y_test_dbp,y_hat_dbp, 'XGB-DBP'))
                                maes_dbp.append(MAE_dbp)

                                regr.fit(X_train_sbp,y_train_sbp)
                                y_hat_sbp = self.regression(regr,y_train_sbp,X_train_sbp,X_test_sbp)
                                axs[1].plot(y_test_sbp.index,y_hat_sbp,label=f'XGB: n_stimators={trees}')
                                MAE_sbp = round(mean_absolute_error(y_test_sbp, y_hat_sbp),2)
                                count_sbp.append(self.count_diff(y_test_sbp,y_hat_sbp, 'XGB-SBP'))
                                maes_sbp.append(MAE_sbp)

                                scores = cross_val_score(regr, X_dbp, y_dbp, cv=5)
                                scores = np.abs(scores)
                                mean_score = np.mean(scores)
                                std_score = np.std(scores)

                # Print the mean and standard deviation of the scores
                                print("Mean score: {:.2f}".format(mean_score))
                                print("Standard deviation: {:.2f}".format(std_score))

#==============================================================
#CAT boosting
                        nTrees=[100]
                        [x_labs.append(f'CAT: n_stimators={x}') for x in nTrees]
                        for trees in nTrees:
                                regr=CatBoostRegressor(iterations=trees, learning_rate=0.1, depth=5, verbose=False)

                                y_hat_dbp = self.regression(regr,y_train_dbp, X_train_dbp,X_test_dbp)


                                # y_hat_dbp = regr.predict(X_test_dbp)

                                axs[2].plot(y_test_dbp.index, y_hat_dbp,label=f'CAT: n_stimators={trees}')
                                MAE_dbp = round(mean_absolute_error(y_test_dbp, y_hat_dbp),2)
                                count_dbp.append(self.count_diff(y_test_dbp, y_hat_dbp, 'CAT-DBP'))
                                maes_dbp.append(MAE_dbp)

                                regr.fit(X_train_sbp,
                                   y_train_sbp)
                                y_hat_sbp = self.regression(
                                   regr,y_train_sbp,
                                   X_train_sbp,X_test_sbp)
                                axs[1].plot(y_test_sbp.index,
                                   y_hat_sbp,label=f'CAT:
                                   n_stimators={trees}')
                                MAE_sbp = round(
                                   mean_absolute_error(
                                   y_test_sbp, y_hat_sbp),2)
                                count_sbp.append(self.
                                   count_diff(y_test_sbp,
                                   y_hat_sbp, 'CAT-SBP'))
                                maes_sbp.append(MAE_sbp)

                                scores1 = cross_val_score(
                                   regr, X_dbp, y_dbp, cv=5)
                                scores1 = np.abs(scores)
                                mean_score1 = np.mean(scores1
                                   )
                                std_score1 = np.std(scores1)

                # Print the mean and standard deviation of
                   the scores
                                print("Mean score1: {:.2f}".
                                   format(mean_score1))
                                print("Standard deviation:
                                   {:.2f}".format(std_score1)
                                   )

#============================================================
# Linear regression
                        start_time = time.time()
                        w_dbp = (np.linalg.inv(X_train_dbp.
                           values.T@X_train_dbp.values))@(
                           X_train_dbp.values.T@y_train_dbp.
                           values)
                        print("Training time of dbp in Linear
                            regression : ", time.time() -
                           start_time, "seconds")
                        y_hat_dbp = X_test_dbp.values@w_dbp
                        print("Prediction time of dbp in
                           Linear regression : ", time.time()
                            - start_time, "seconds")
                        axs[2].plot(y_test_dbp.index, y_hat_dbp,label='Linear')
                        MAE_dbp = round(mean_absolute_error(y_test_dbp, y_hat_dbp),2)
                        maes_dbp.append(MAE_dbp)
                        count_dbp.append(self.count_diff(y_test_dbp, y_hat_dbp, 'Lin-DBP'))


                        w_sbp = (np.linalg.inv(X_train_sbp.values.T@X_train_sbp.values))@(X_train_sbp.values.T@y_train_sbp.values)
                        y_hat_sbp = X_test_sbp.values@w_sbp
                        axs[1].plot(y_test_sbp.index, y_hat_sbp,label='Linear')
                        MAE_sbp = round(mean_absolute_error(y_test_sbp, y_hat_sbp),2)
                        maes_sbp.append(MAE_sbp)
                        count_sbp.append(self.count_diff(y_test_sbp, y_hat_sbp, 'Lin-SBP'))
                        x_labs.append(f'Linear')




#============================================================
                        axs[0].set_title(f'Prediction vs. Test for patient {patient}')

                        # axs[0].sharex(axs[2])
                        width = 0.35
                        axis = np.arange(len(maes_dbp))
                        axs[3].bar(axis+width/2,maes_dbp,width,label='DBP')
                        axs[3].bar(axis-width/2,maes_sbp,width,label='SBP')
                        axs[3].set_ylim(0,15)
                        axs[3].set_title('MAE for each algorithm')
                        # x_labs = [f'POL: {x}' for x in pol_orders]
                        # [x_labs.append(f'SVR: {x}') for x in Cs]
                        # x_labs.append('SVR: Best')
                        axs[3].set_xticks(range(len(x_labs)))
```

```
            axs[3].set_xticklabels((x_labs))
            axs[3].set_ylabel('MAE [-]')
            axs[3].legend()

            axs[1].plot(y_test_sbp,label='Test',ls='--',lw=2,c='tab:blue')
            axs[1].set_ylabel('SBP [mmHg]')
            axs[1].set_ylim(min(df['SBP'])-10,max(df['SBP'])+10)
            axs[1].legend(ncol=3)

            axs[2].plot(y_test_dbp,label='Test',ls='--',lw=2,c='tab:blue')
            axs[2].set_ylabel('DBP [mmHg]')
            axs[2].set_ylim(min(df['DBP'])-10,max(df['DBP'])+10)
            axs[2].set_xlabel('Time [s]')
            axs[2].legend(ncol=3)
            for ax in plt.gcf().axes[0:2]:
                try:
                    ax.label_outer()
                except:
                    pass

            #print(df)
            plt.tight_layout()
            self.create_path('Plots\\Regression')
            if self.save_figure: plt.savefig(f'Plots\\Regression\\{patient}.png')
            dbp_errors[patient] = maes_dbp
            sbp_errors[patient] = maes_sbp
            for lab,err_sbp,err_dbp in zip(x_labs,count_sbp,count_dbp):
                for cnt,thresh in enumerate(self.mmHg_thresh):
                    # print(lab,err)
                    final_dict_sbp[patient].update({f'{lab} > {thresh}':err_sbp[cnt]})
                    final_dict_dbp[patient].update({f'{lab} > {thresh}':err_dbp[cnt]})
                plt.show()

                pd.DataFrame(final_dict_sbp).to_excel(f'{self.dataset_path}\\sbp_thresh_errors.xlsx')
                pd.DataFrame(final_dict_dbp).to_excel(f'{self.dataset_path}\\dbp_thresh_errors.xlsx')
                dbp_errors.index = x_labs
                sbp_errors.index = x_labs
                # print(dbp_errors)
                # print(sbp_errors)
                dbp_errors.to_excel(f'{self.dataset_path}\\dbp_errors.xlsx')
                sbp_errors.to_excel(f'{self.dataset_path}\\sbp_errors.xlsx')


        def regression(self,clf,y_train,X_train,X_test):
                from sklearn.preprocessing import RobustScaler
                scaler_x = RobustScaler()
                # scaler_y = StandardScaler()
                # print(y_train)
                X_train = (scaler_x.fit_transform(X_train))
                # y_train = scaler_y.fit_transform(np.array(y_train).reshape(1,-1))
                # print(y_train)
                clf.fit(X_train, y_train)
                pred = clf.predict(scaler_x.transform(X_test))
                # pred = scaler_y.inverse_transform(np.array(pred).reshape(1,-1))
                return pred

        def GS_regression(self,clf,params,y_train,X_train,X_test):
                clf = GridSearchCV(clf, params)
                clf.fit(X_train,y_train)
                pred_gs = clf.predict(X_test)
                return pred_gs

        def count_diff(self, test, pred, alg_type):
                count_perc = []
                for thresh in self.mmHg_thresh:
                        test, pred= np.array(test), np.array(pred)
                        diff = np.abs(test - pred)
                        count = sum(i > thresh for i in diff)
```

```
                                count_perc.append(round(100*count/len
                                    (test),1))
                    #print(f'{count_perc}[%] > {thresh} [mmHg]
                        for {alg_type}')
                    #print()
                    return count_perc
```