UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

RESPIRATORY RATE ESTIMATION USING WIFI CHANNEL STATE INFORMATION – A MACHINE LEARNING APPROACH

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

By

NIKA MOSTAHINIC
Norman, Oklahoma
2020

RESPIRATORY RATE ESTIMATION USING WIFI CHANNEL STATE INFORMATION – A MACHINE LEARNING APPROACH

A THESIS APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

Dr. Hazem H. Refai, Chair

Dr. Thordur Runolfsson

Dr. Samuel Cheng

Dedication

To my parents,

Tamara and Dean,

&

my beloved siblings,

Marko, Stjepan and Maura

Acknowledgements

I wish to show my gratitude to my advisor and mentor, Dr. Hazem Refai, for his guidance through each stage of the process. Without his persistent help, this project would not have reached its goal. I want to thank Dr. Thordur Runolfsson and Dr. Samuel Cheng for taking their time to review this thesis. My thanks to Dr. Mohamad Omar Al Kalaa for providing his software and to Mohamed Irfan Ali, Joseph Sullivan, and Nabil Asfari for participating in my study. Finally, I want to acknowledge the support and great love of my family and friends.

Table of Contents

List of Tables

List of Figures

Abstract

Respiratory rate (RR) is an important vital sign for diagnosing and treating a number of medical conditions. Current respiration monitoring systems require that a special device is continuously attached to the human body. However, contactless respiration monitoring systems have recently been developed to overcome this inconvenience. Research has shown that channel state information (CSI) measured by WiFi devices can be used for estimating RR. Although pattern-based respiration detection has been used to extract RR from periodic changes in CSI, systems based on this method do not perform well when channel conditions are not favorable. This thesis highlights newly introduced learning-based approaches used for RR estimation. Off-the-shelf WiFi devices were used to collect fine-grained wireless CSI data, which was then used to train and evaluate machine learning models.

Results show that classification algorithms, including KNN, SVM, Random Forest, Logistic Regression and MLP, achieve over 96% accuracy when predicting RR. Regression models were compared to an existing pattern-based system, demonstrating that the majority of regression models have better performance when estimating RR. For instance, Logistic Regression's Root Mean Square Error (RMSE) is 0.35, while pattern-based system's RMSE is 2.7. It is important to note that classification and regression models cannot be generalized, nor can they accurately predict respiratory rate using the data collected from a new and previously unseen subject. To improve and make the models more generalizable, data used to train the models must be collected from a larger number of subjects.

Chapter 1: Introduction

Vital signs are collected to measure essential body functions. Respiratory rate (RR)—the number of breaths taken per minute—is a critical vital sign for assessing an individual's health. Normal RR for an adult at rest is 12 to 20 breaths per minute (bpm). An RR below 12 or above 25 bpm is considered abnormal [1]. RR consists of important information for detecting and monitoring medical problems. In fact, daily monitoring of RR could help diagnose and treat a variety of pathological conditions (e.g., respiratory, metabolic, and cardiovascular disorders, to name a few [2]). Moreover, RR monitoring has aided in diagnosing pulmonary disease, heart failure, anxiety, and sleep disorders. Obstructive sleep apnea syndrome (OSAS) and chronic obstructive pulmonary disease (COPD) are among the chronic diseases requiring constant RR monitoring [3]. Each year, cardiovascular diseases account for 17.9 million deaths worldwide, while chronic respiratory diseases account for 3.9 million [4]. Respiratory monitoring is important for both in-patient and in-home health care settings.

1.1 Contact-based RR Monitoring Systems

Traditional technologies used for measuring RR are contact-based, which means that they require attaching a sensor to a subject's body. Figure 1-1 shows contact-based techniques for measuring RR, as well as the related human body areas where sensors should be attached [5]. These include the face, neck, chest, wrists, fingers, and abdominal area. Contact-based technologies are intrusive, limiting a subject's activity and mobility. For example, a patient might be required to wear a chest or wrist band, nasal probe or finger clip. The need to constantly wear such technologies also renders them inconvenient, causing discomfort, especially for people with sensitive or burned skin. Furthermore, there

is a general risk of a sensor becoming detached; also some sensors must be wired to a monitor.

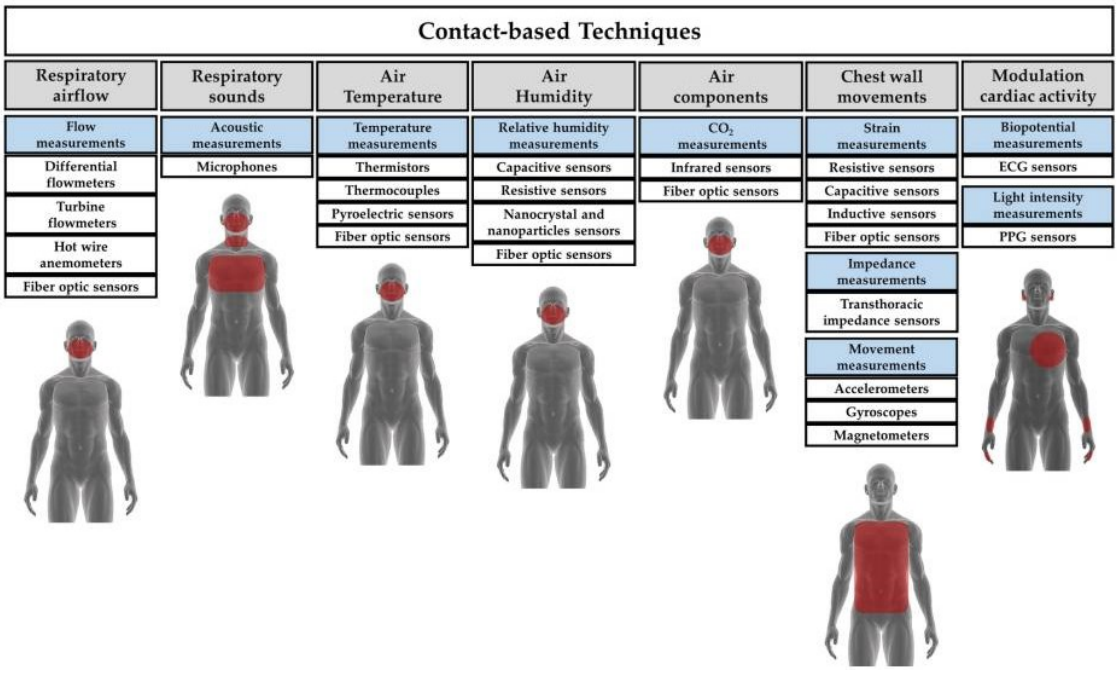

Figure 1-1. Contact-based techniques and locations.

1.2 Contactless RR Monitoring Systems

The many disadvantages of contact-based sensors have been the impetus for developing a number of contactless RR monitoring systems. A camera-based system, which extracts raw breathing signals from the video stream and measures RR without any body contact, was proposed in [6]. Notably, system performance relies on appropriate lightning conditions and the availability of direct line of sight (LoS). Such vision-based approaches often raise privacy concerns, as well. Authors of [7] describe an RR monitoring system that leverages a smartphone's microphone to capture human breathing sound and

measure RR based on signal envelope detection. System drawbacks include sensitivity to background noise and short sensing distance.

Radio frequency (RF)-based systems have also been proposed. These are neither intrusive nor sensitive to lighting conditions. A transmitter first sends a signal to a receiver, and then the signal's amplitude and phase are modulated by the subject's breath-induced inhalation and exhalation chest movements. Next, signal changes are measured at the receiver [8]. RF systems include Doppler [9], ultra-wideband (UWB) [10], and Frequency Modulated Carrier Waves (FMCW) radar [11]. Although these systems accurately measure RR, they require costly, highly complex, specialized devices, making them difficult to deploy in a home setting.

To address this limitation, narrowband commodity off-the-shelf (COTS), device-based RR monitoring systems have been developed. These utilize widely available Wi-Fi infrastructure that is both cheap and easy to deploy. Similar Wi-Fi devices are becoming more common in homes and in buildings due to the growth of Internet of Things (IoT). Many are currently used for wirelessly transmitting data, all the while their channel measurements can be used for RR monitoring. Notably, although Wi-Fi received signal strength (RSS) from COTS devices can be used to extract a person's breathing pattern [12], systems leveraging Wi-Fi channel state information (CSI) have shown superior performance as a result of fine signal granularity. CSI measurements describe the amplitude and phase of the wireless channel at the sub-carrier level, while the RSS provides a single measurement averaged over the entire channel. The RR monitoring system presented in this work focuses on Wi-Fi CSI.

1.3 Background of CSI

A growing demand for wireless data traffic has been the impetus for Wi-Fi to leverage multiple input multiple output (MIMO) technology. With this solution, data rates are high, because multiple antennas are placed at the transmitter and receiver to create multiple spatial streams [13]. In a Wi-Fi system using MIMO and orthogonal-division multiplexing (OFDM), CSI can be obtained for every transmitter and receiver antenna pair at each sub-carrier frequency. CSI of every sub-carrier is a complex number, which represents amplitude attenuation and phase shift impacted by multi-path effects (See Figure 1-2) [14]. For a packet transmitted using M number of transmitting and N number of receiving antennas on a 20 MHz wide channel, CSI is a 3D complex matrix of size MxNx56. The number 56 represents the number of subcarriers, which is a result of dividing a 20 MHz channel by OFDM.

Given that a 40 MHz-wide channel is used for transmission, the number of subcarriers is 114 [14]. CSI is measured in the following way. First, a Wi-Fi transmitter sends long training symbols (LTFs) to the receiver, wherein the LTF packet preamble contains pre-defined symbols for each sub-carrier. After LTFs are received, the receiver estimates CSI matrix using the original LTFs and the received signal. The Wi-Fi channel for each sub-carrier is modeled using the following formula

$$y = Hx + n, \qquad \text{Eq. 1}$$

where y is the received signal; H is the CSI matrix; x is the pre-defined transmitted signal; and n is the noise. The receiver estimates H using x and y signals [15]. CSI of a single subcarrier is defined as

$$h = |h|e^{jsin\angle \mathrm{h}}, \qquad \text{Eq. 2}$$

where |h| is the CSI amplitude, and ∠h is the CSI phase.

CSI describes the way in which a Wi-Fi signal propagates from transmitter to receiver at different sub-carrier frequencies through multiple paths. CSI is sensitive to the presence and movements of humans and objects; therefore, a time series of CSI data can be used for various wireless sensing purposes. For example, variations in CSI amplitude can be used for human-presence detection, motion detection, activity recognition, gesture recognition, and human identification. CSI phase shifts can be used for human localization and tracking [15]. CSI measurements can also be used to estimate RR. Amplitude and CSI phase on many subcarriers are affected by breathing-induced chest movement. CSI-based RR estimation can be divided into two categories: pattern-based and model-based, which are described in sections 1.4 and 1.5, respectively.

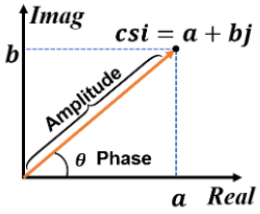

Figure 1-2. CSI format.

1.4 Pattern-based RR Estimation

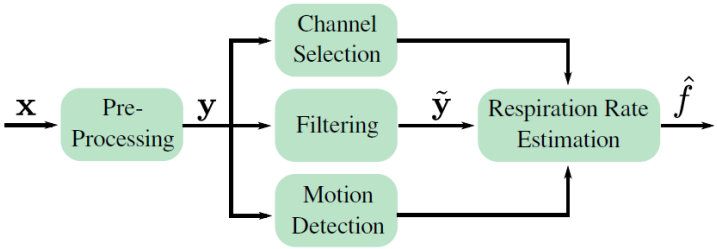

Figure 1-3. RR estimation block diagram.

Pattern-based RR estimation studies changes in the Wi-Fi CSI patterns to extract RR. Methods that are most commonly used in pattern-based RR estimation are depicted in Figure 1-3. First, pre-processing is used to remove the noise form CSI measurements. Then, filtering is used to remove the unwanted frequency content to extract a breathing signal. Next, channel selection is performed to select the stream or subcarrier with the greatest potential of showing changes due to person's breathing. This selection is needed because due to the constructive and destructive interference of multipath signal components, some streams are more sensitive to person's respiration than the others. Motion detection is used to flag time periods during which RR estimation is not reliable because the signal is affected by person's movements. Lastly, the selected stream is utilized for RR estimation.

There are two methods commonly used for estimating RR. The first method, which is called power spectral density (PSD) method, obtains average PSD in a 10 to 30 s measurement window. PSD is computed between a minimum and maximum frequency, which accounts for a range of normal breathing rates. Estimated RR is the frequency at which PSD is maximum. The second method is called inter-breath interval (IBI) method. Using this method, the peaks of the stream are identified and the time difference between the peaks is calculated. Estimated RR is the inverse of average time difference [8].

1.5 Model-based RR Estimation

Model-based RR estimation uses physical theories or statistical models to relate breathing to received CSI measurements. The most common model-based algorithm for RR estimation application is Fresnel Zone Model. Fresnel zones (See Figure 1-4) are concentric elliptical regions with foci in a pair of transceivers used in radio propagation

theory to study diffraction loss caused by an obstruction between transmitter and receiver. The radius of the *n*th Fresnel zone can be expressed using the following formula:

$$r_n = \sqrt{\frac{n\lambda d_1 d_2}{d_1 + d_2}}, \quad d_1, d_2 \gg r_n \,, \qquad \text{Eq. 3}$$

where λ is the radio wavelength; d_1 is the distance from the transmitter; and d_2 is the distance from the receiver. The important zones for transmission are the first 8 to 12 zones. Additionally, more than 70% of the energy is transferred via the first Fresnel zone (FFZ). Movements in this zone can greatly affect received signal amplitude and phase [16]. Moreover, when an object moves across a series of Fresnel zones, received signal looks like a continuous sinusoid.

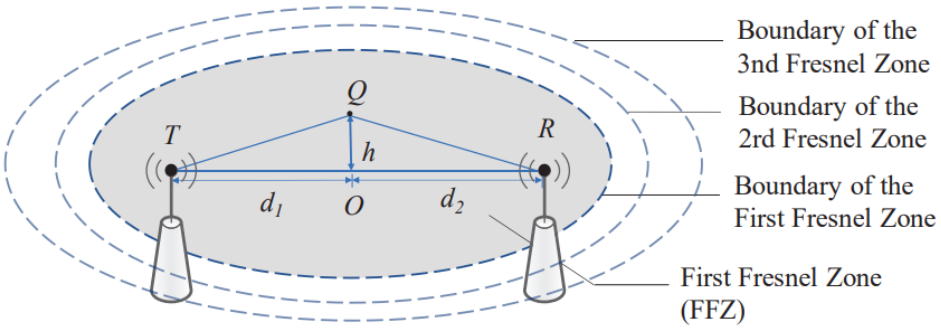

Figure 1-4. Geometry of the Fresnel diffraction at point Q [16].

Chapter 2: Related Work

In recent years, contactless RR monitoring systems based on Wi-Fi CSI have sparked significant interest in research. There has been a growing interest in these systems because they are non-invasive, low cost and easy to implement. Previous work can be divided into two primary types: 1) pattern-based and 2) model-based RR estimation systems.

2.1 Pattern-based RR Estimation Systems

Liu, *et al.* (2014) [17] developed Wi-Sleep—the first system analyzing CSI data from COTS Wi-Fi devices for monitoring human respiration during sleep. This work was extended in [18], where abnormal breathing and varied sleeping postures were examined. The authors discovered that the ripple-like pattern in the CSI amplitude is related to chest movement. Accordingly, they used the CSI amplitude as an input for respiration monitoring. In this work, all CSI streams from all subcarriers were combined and weighted based on their periodicity to obtain an RR estimate.

Liu, *et al.* (2015) [19] developed a CSI-based system to track RR and heart rate during sleep for one- and two-person scenarios. CSI amplitude streams for one-person RR estimation were selected and weighted according to their variance. For two-person RR estimation, the PSD for each selected subcarrier was obtained. K-means clustering was applied to classify the strong peaks into two clusters relative to PSD amplitude and the frequency. Estimated RRs of two persons were the average values of frequencies in two clusters. One notable issue was the difficulty in determining which RR belonged to whom.

Wu, *et a.l* (2015) [20] developed a system, called DeMan, that did not require that a person lays on a bed. Instead, respiration was detected for a person in a standing position.

The DeMean system investigated the likelihood that a measured signal has the same frequency as human breathing. Given that the estimated frequency falls within the range of human breathing frequencies, a stationary person can be detected.

So far, the systems that use the amplitudes of Wi-Fi CSI measurements have been discussed. They do not use the CSI phase information due to large variations caused by asynchronous times and frequencies of the transmitter and receiver. TensorBeat [21] and PhaseBeat [22] systems were the first to utilize CSI-phase difference data for two receiver antennas to monitor RR. Researchers found CSI-phase difference after appropriate calibration.

Although pattern-based RR estimation systems have shown encouraging results, they are mainly based on empirical experiments. Also, they do not perform well when the fading conditions are unfavorable. If multipath components are added destructively at the receiver, the breathing signal will be hidden in noise. When this occurs, it is difficult to estimate RR for a majority of selected streams.

2.2 Model-based RR Estimation Systems

Wang *et al.* (2016) [23] introduced a Fresnel model for indoor Wi-Fi radio propagation. Researchers applied this model to an RR detection system using COTS Wi-Fi devices. They used the system developed in [19] to validate their theory. Moreover, they investigated how user location, body orientation, and frequency diversity affect system performance. Results showed that user location and body orientation influence CSI signal quality. Likewise, blind-spot locations in the sensing range of a transceiver range where the RR detection is not guaranteed. Specifically, the worst location for RR sensing was around the boundary within each Fresnel Zone; the best location was in the middle.

Wang *et al.* (2017) [24] used multiple transmitter and receiver antenna pairs to improve RR detection by overlapping multiple Fresnel Zones. Researchers proposed an approach for multi-user respiration detection. In their system, a receiver is placed beside each user. In a multi-user scenario, respiration information of a user is found in the shortest reflection path. Hence, for each user, they filter out the data that is greatly affected by the longer paths. In fact, the data whose time of arrival (TOA) is greater than a truncation threshold was filtered out.

Zeng *et al.* (2018) [25] eliminated blind spot locations where respiration couldn't be detected by combining both CSI amplitude and phase. Researchers observed that an undetectable location wherein CSI amplitude is used for RR estimation might be a detectable area when CSI phase is used, and vice versa. Accordingly, a conjugate multiplication (CM) of CSI between two receiver antennas was used as an input for RR estimation.

Pattern-based and modeling-based systems typically require a lot of signal processing. Model-based algorithms are generally not reusable or robust enough for new scenarios and environments. To the best of my knowledge, learning-based approaches have not been used for estimating RR based on CSI data. Learning-based algorithms attempt to learn a function for estimating RR by using labeled training samples of CSI measurements. The advantage of this method over previous ones is that very little or no signal processing is required; also the method is evolvable, meaning estimated RR could improve with more training data.

Chapter 3: Predicting RR Using Machine Learning Algorithms

Learning-based algorithms were applied to estimate RR based on collected CSI measurements. Given the model expressed by

$$y = f(x), \qquad \text{Eq. 4}$$

where y are RR estimation results and x are CSI measurements, the goal of the algorithm is learning mapping function f by training samples of x and y. After CSI data was collected and analyzed, machine learning models were trained on training data. Finally, models were used to make predictions on test data, and then evaluated based on the predictions.

3.1 Data Collection

3.1.1 Equipment

The data collection was performed using two TP-Link TL-WDR4300 routers. One served as a transmitter (Tx), and the other as a receiver (Rx). Each router had three external omnidirectional antennas (See Figure 3-1).

Figure 3-1. TP-Link TL-WDR4300 router.

Atheros CSI tool [14], which is an OpenWrt firmware for CSI acquisition, was installed on the routers. OpenWrt was chosen for the operating system (OS), since it is a commonly used Linux OS for embedded devices. The network setup used to obtain CSI data is shown in Figure 3-2. Routers were configured to utilize IEEE 802.11n protocol and operate in the 5 GHz frequency band, primarily because this frequency band offers lower levels of interference and improved spectrum efficiency compared with the 2.4 GHz frequency band. Wi-Fi channel 20 was selected for transmission with 5.2 GHz center frequency. Channel bandwidth was 20 MHz, meaning that for every packet received, N_{Tx} x N_{Rx} x 56 CSI measurements were available. N_{Tx} represents the number of transmitting antennas; N_{Rx} represents the number of receiver antennas; and 56 is the number of subcarriers. In the tested scenario, 3 x 3 x 56 CSI matrix was obtained for every received packet.

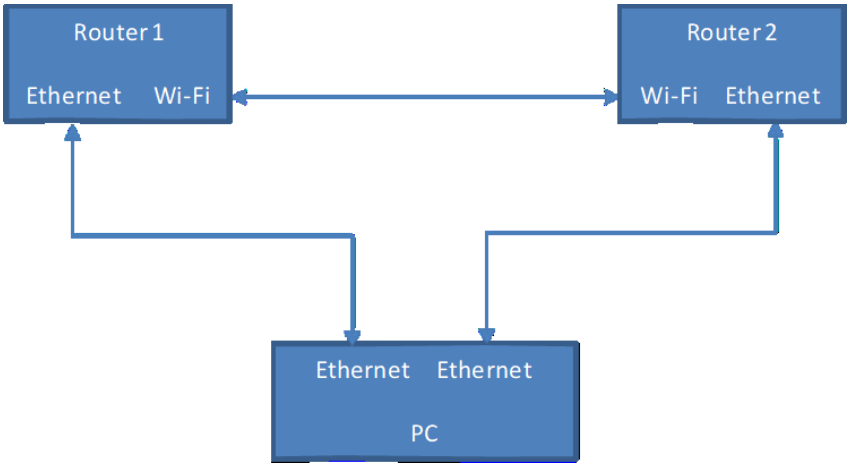

Figure 3-2. Network setup.

3.1.2 Test Setup

CSI data was collected separately from four human subjects in a laboratory room located in a semi-underground space. Subjects were approximately 25 years old with slightly different heights and weights. Routers were placed on wooden tables at an elevation of 1 m and separated by a distance of 2 m (See Figure 3-3 and Figure 3-4). Each

subject sat still in a chair with his or her chest perpendicular to and between both routers. Only the test subject was present in the room, and the only movements in the room were caused by subject's chest during data collection.

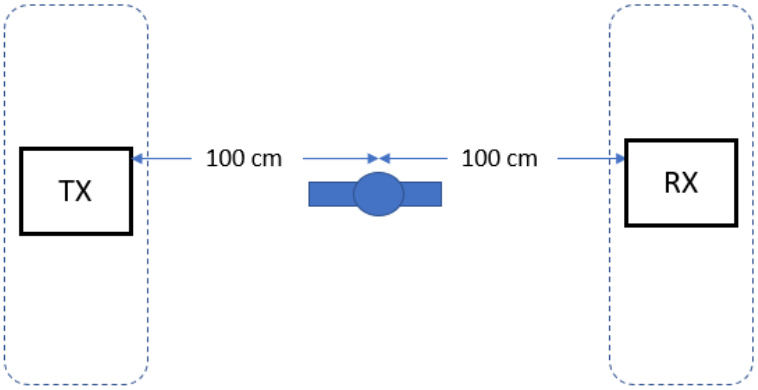

Figure 3-3. Data collection setup scheme.

Figure 3-4. Data collection setup.

CSI data for each subject was collected for one minute during each test, wherein 10 packets were sent per second, for a total of 600 packets. At first, test subjects were breathing at a 12 bpm RR. During each follow-up test, RR was incremented by 1 bpm until 23 bpm was reached. Each test was repeated three times. A spectrum analyzer was used to ensure no unintended signals were present in the room.

3.1.3 Ground Truth

To label each test, ground truth RR values were recorded using the counting method and the respiration monitor belt logger sensor NUL-236 [26], which is shown in Figure 3-5. This sensor measures the air pressure in the belt, which changes according to the breathing of the subject, and it calculates the RR based on those air pressure measurements. The sensor uses the piezoresistive effect to monitor respiration. Its transducer composed of silicon between metal foils changes resistance according to pressure and outputs a voltage depending on absolute pressure. In this way, when a subject breathes, pressure applied to the respiration monitor belt is detected by the sensor and converted to a voltage. The voltage reading is further converted into arbitrary units to monitor RR. The sensor's specifications are listed in Table 1.

NeuLog's software application was used to display respiration data in the form of a graph; arbitrary units were plotted versus time. An example of the graph is shown in Figure 3-6. In this graph, each wave represents one breath. Respiratory data was exported to a .csv file, and a Matlab program was used to count peaks. Since the duration of each test was 60 seconds, RR is determined by counting the number of peaks in each graph. To eliminate detecting fake peaks, minimum peak prominence was set to 200 and minimum peak distance was set to 2.4. An example of the Matlab program output is shown in Figure 3-7.

Table 1. Respiration Monitor Belt Logger Sensor NUL-236 Specifications

Range and operation modes	ADC resolution	Resolution	Max Sample Rate (S/sec)
0 to 20,000 Arbitrary units	13 bit	1	100

Figure 3-5. Respiration monitor belt logger sensor NUL-236.

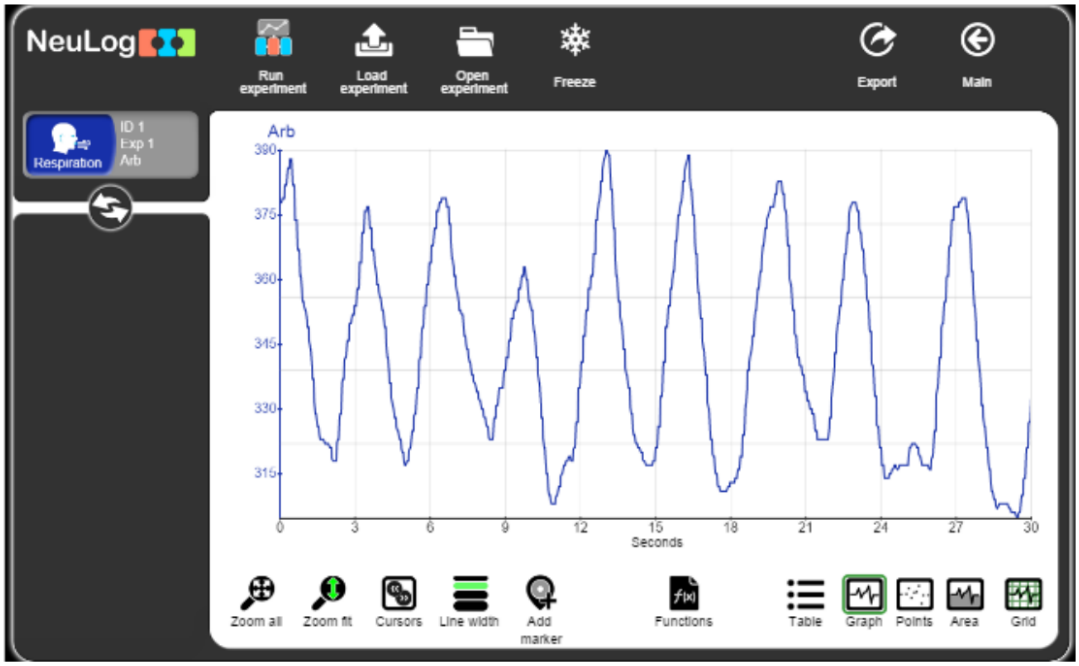

Figure 3-6. NeuLog software application.

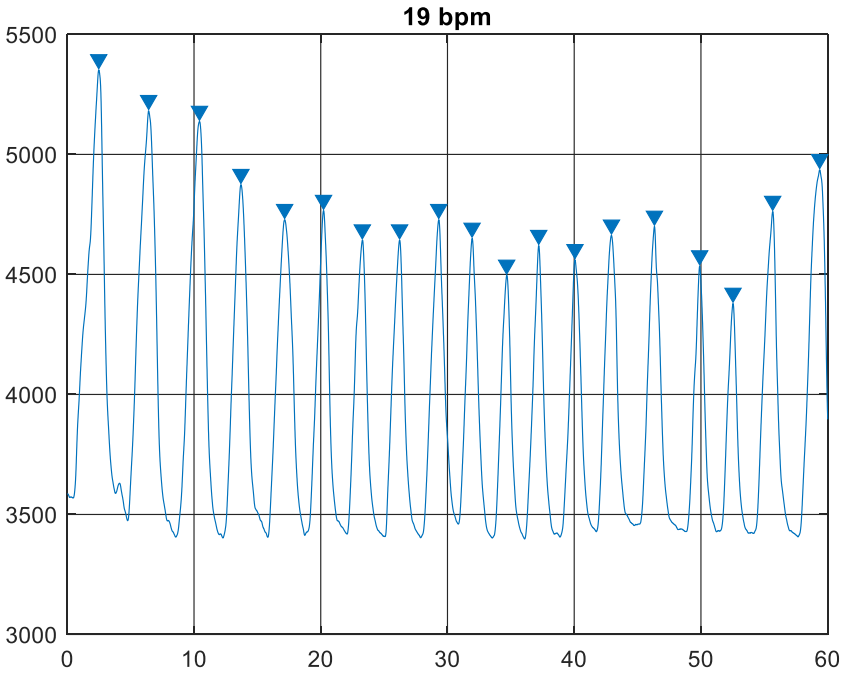

Figure 3-7. Peak counting to obtain RR.

3.2 Exploratory Data Analysis

The CSI data collected for each test was stored as an N_{pk} x 504 (3x3x56) matrix, where N_{pk} is the number of received and CSI decoded packets. Since packet transmission rate was 10 packets per second, expected number of received packets during a 60 second test was 600. However, not all 600 packets were received and decoded during each test; the number of decoded packets varied from 500 to 600. This phenomenon could be explained by several reasons: a) wireless interference was coming from surrounding buildings, b) receiving a non-sounding packet was received, or c) a problem with the firmware. For consistency, CSI data for the first 500 packets was used in the analysis. Fifty-six sub-carriers were counted for each Tx-Rx antenna pair, and three antennas were used at Tx and Rx, totaling 504 sub-carriers. The absolute value of the CSI matrix was obtained to determine amplitude values. The matrix was transposed so that each row represented a sub-carrier and each column represented CSI amplitude value for consecutive packets. Hence, collected data for each test was plotted in a 500 x 504 complex matrix.

CSI amplitude variations reported in each row (i.e., sub-carrier) are the result of measured breathing. In fact, every row (i.e. sub-carrier) is used as a data observation labeled by the subject's breathing rate. For example, after a one-minute data collection for a subject breathing at 12 bpm, 504 data rows were collected and each row was labeled with the number "12." CSI amplitude for one of the tests where a subject was breathing at 12 bpm is displayed in Figure 3-8. The y-axis shows sub-carrier index, and the x-axis shows the packet index. Figure 3-8 illustrates the way in which periodic changes in CSI amplitude represent the breathing signal for different sub-carriers.

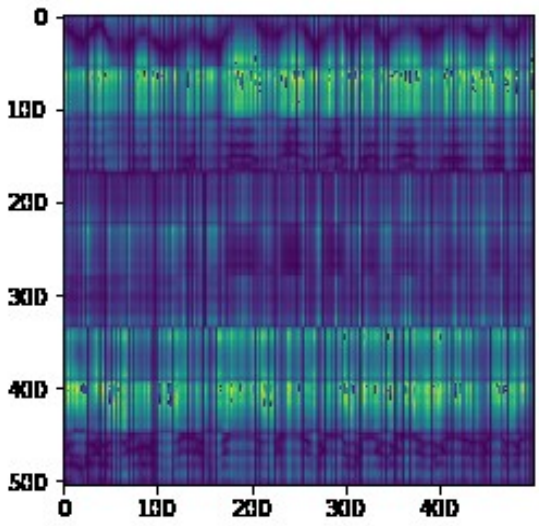

Figure 3-8. Image of a 12 bpm test.

Data was collected individually from four subjects for 12 different breathing rates, ranging from 12-23 bpm. Each RR represented a class, and data collection for each class was repeated three times. Figure A-1 and Figure A-2 in Appendix A offer image examples of classes for four subjects, namely A, B, C and D. Figure 3-9 shows CSI amplitude variations for a randomly chosen single subcarrier for each class. Total data collected had 72, 576 rows or observations, and 500 columns or features.

The number of rows was obtained, as follows: 504 observations/test * 3 tests * 4 subjects * 12 classes/ subject = 72, 576 observations. The dataset is balanced, and its class balance is shown in Figure 3-10. Number of data observations for each class is 6,048.

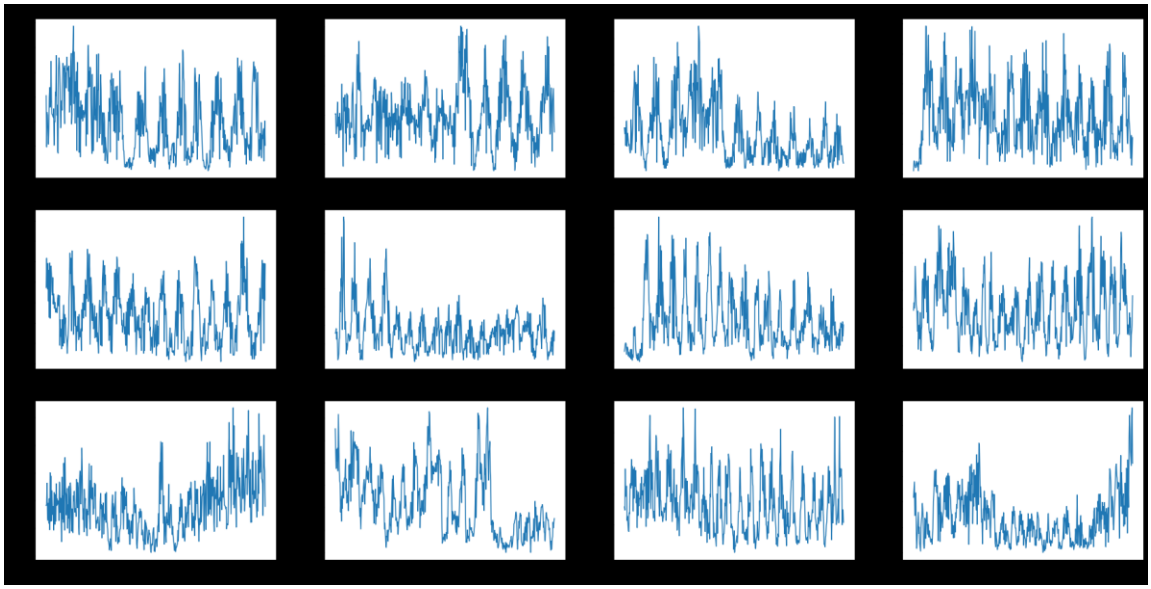

Figure 3-9. CSI amplitude variations for a single subcarrier for each class.

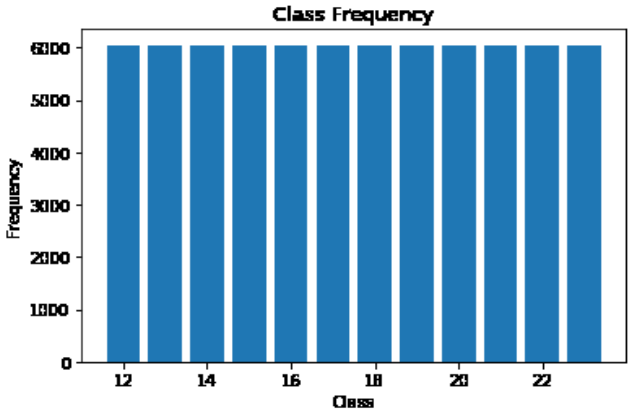

Figure 3-10. Class frequency.

The dataset was divided into two subsets: 1) a training set containing 70% of the original dataset and is used to train the model, and 2) a test set containing the remaining 30% of the original dataset and is used to test the trained model. Python's scikit-learn library was used to split the dataset into training and test-data subsets. The resulting subsets were shuffled and were characterized by the same proportions of class labels as the input dataset (i.e., balanced). The new training dataset contained 50,803 data instances, while the

test dataset contained 21,773. Figure 3-11 and Figure 3-12 show the plots of the first four training and the test data instances with their labels, respectively.

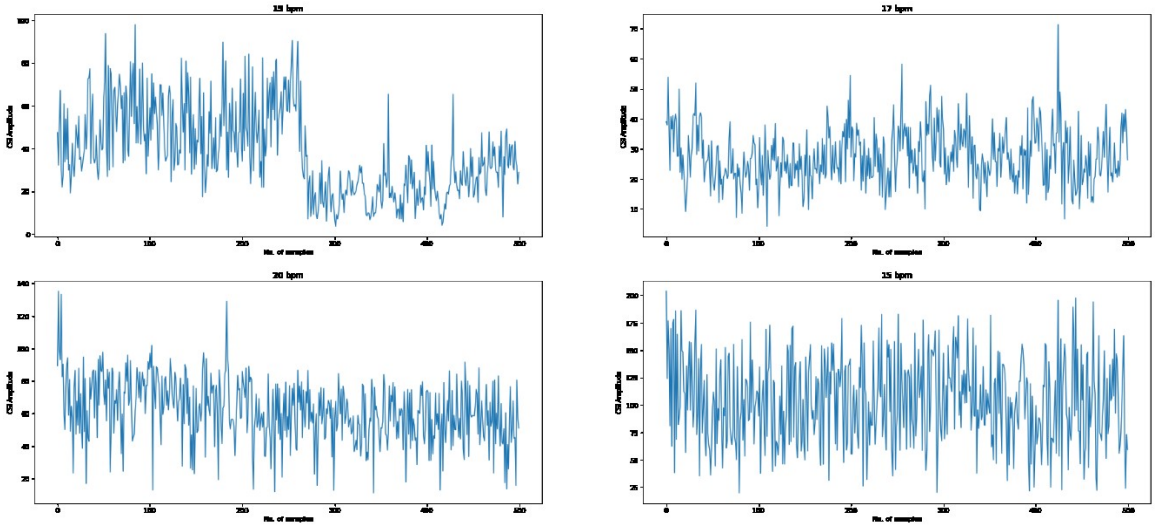

Figure 3-11 Plot of the first four training data instances.

3.3 Data Preparation

Before training the models, features were scaled (or normalized) so that they can be uniformly evaluated. For example, KNN calculates the distance between two data points by measuring the Euclidean distance. If one of the features' values vary widely, the distance will be governed by this feature. Hence, it is important to normalize the data so that every feature's contribution to the final distance is proportional. Scaling the features was done using StandardScaler class from scikit-learn package, which standardizes features by removing the mean and scaling to unit variance. Standardization (or Z-score normalization) is expressed by

$$x' = \frac{x - \bar{x}}{\sigma}, \qquad \text{Eq. 5}$$

where $\bar{x}$ is the mean of the feature vector, and σ is its standard deviation. This method is widely used in many machine learning algorithms because it helps objective functions to work properly and it helps the gradient descent to converge faster.

3.4 Classification Models to Predict RR

Since RR variable can take a discrete set of values, predicting RR can be a classification problem. Classification algorithms have been applied to develop models for predicting RR based on CSI collected from a person breathing when sitting still between two routers. The classification algorithms that were used in this work are k-Nearest Neighbor (KNN), Support Vector Machine (SVM), Decision Tree, Random Forest, Naïve Bayes, Logistic Regression, and a Neural Network (NN).

Classification algorithms were implemented in Python. To build the models, the proper packages, and their functions and classes were used. NumPy—a fundamental package for scientific and numerical computing in Python—was used because it allows high-performance operations on single- and multi-dimensional arrays. Scikit-learn [27], which is a widely used Python library for machine learning, was used for data preprocessing, and for implementing classification algorithms. Matplotlib was a package used for data visualization and for visualization of classification results.

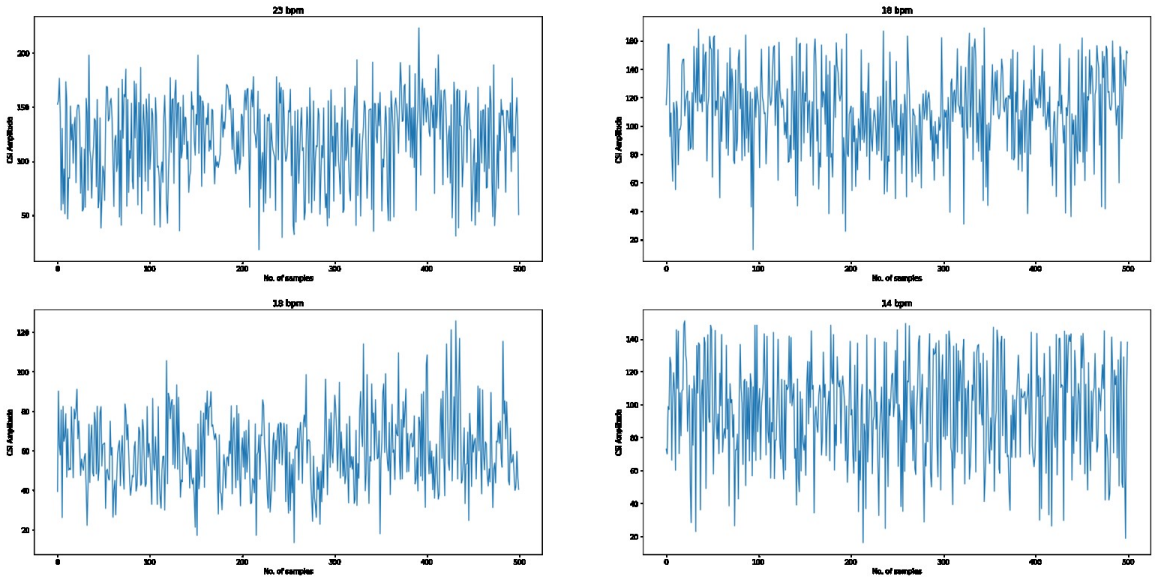

Figure 3-12. Plot of the first four test data instances.

3.4.1 KNN

K-Nearest Neighbor (KNN) classifier takes a test data sample and it finds the closest k number of the training samples. Predicted label for the test sample is the most frequent label of the closest k training samples. Notably, the distance used to determine the closest samples can be any metric measure, although Euclidean distance is the most common choice. An example of KNN with K=3 is shown in Figure 3-13 [28]. To predict blue star class, the three nearest samples are considered. Because the three closest points belong to red circles, a blue star is classified as a red circle, as well.

KNN was implemented in Python by using scikit-learns's sklearn.neighbors.KNeighborsClassifier class. *K* number of neighbors must be chosen to build the model. To find the optimal number of *K*, 10% of training data was used as a validation set. KNN classifier models were built for k=1, 3, 5, 7, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, and 100. To build each model, KnearestNeighbor class was used from Python's sklearn package. Each model was evaluated on the validation set. Figure 3-14 illustrates

model accuracy versus number of neighbors. The figure shows that the most accurate model evaluated on validation set is the model with k=1.

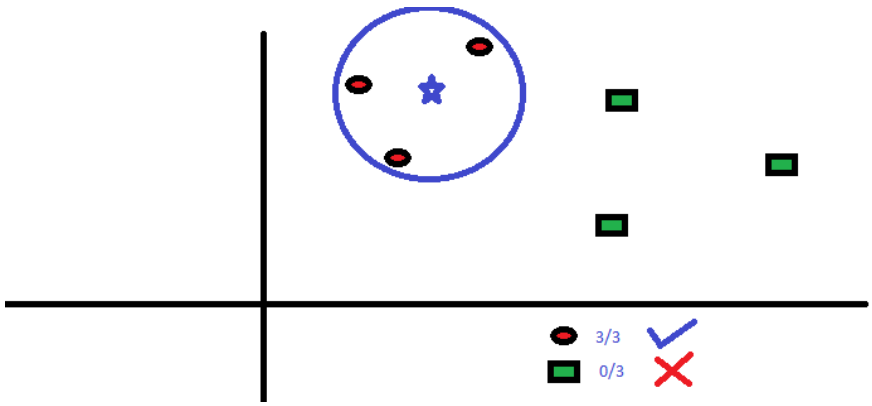

Figure 3-13. Visualization of KNN.

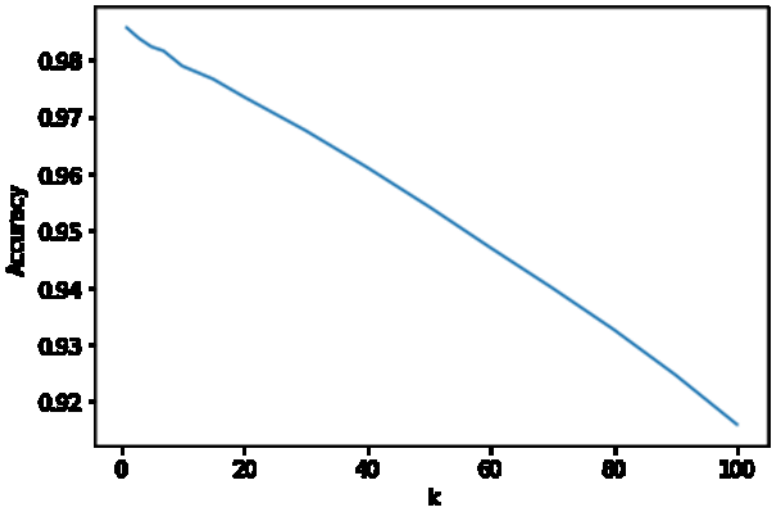

Figure 3-14. KNN accuracy vs number of K.

After k=1 was chosen as the optimal hyperparameter, the model was evaluated using test data; model accuracy was 96.38%. The accuracy of the model is the number of correct predictions divided by the number of total predictions. Figure 3-15 shows the confusion matrix, which visualizes model performance, shows that all classes were

predicted nearly 100%, accuracy, except for classes 14 bpm and 15 bpm. Nearly 20% of 14 bmp data samples were classified as 15 bpm and vice versa.

Table **2** shows the classification report with the most common metrics. Class precision is the number of correctly predicted data samples for any given class, divided by the number of total predictions for the class. On the other hand, class recall is the number of correctly predicted data samples for the class, divided by the actual number of data samples belonging to the same class. The F-1 score is a weighted harmonic mean of precision and recall, with best value at 1 and worst at 0. The support is the number of occurrences for each class. Table 2 shows that the precision and recall are 100% for all classes, except for classes 14 bpm and 15 bpm.

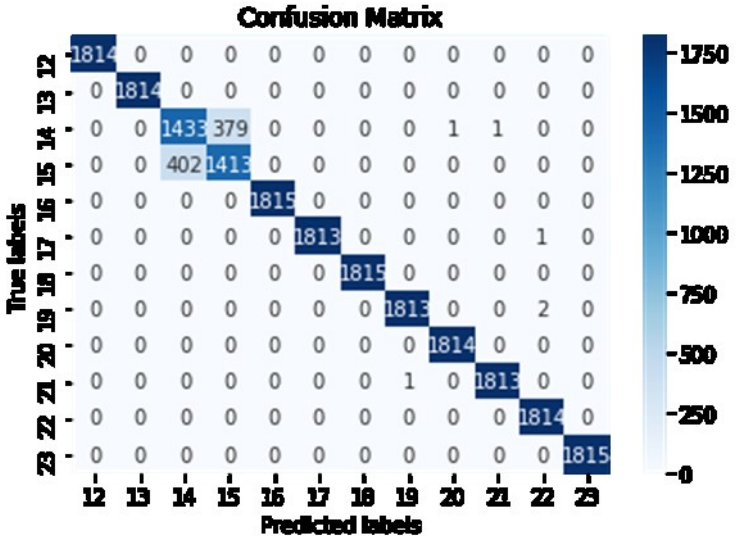

Figure 3-15. KNN confusion matrix.

Table 2. KNN Classification Report

	precision	recall	f1-score	support
12	1.00	1.00	1.00	1814
13	1.00	1.00	1.00	1814
14	0.78	0.79	0.79	1814
15	0.79	0.78	0.78	1815
16	1.00	1.00	1.00	1815
17	1.00	1.00	1.00	1814
18	1.00	1.00	1.00	1815
19	1.00	1.00	1.00	1815
20	1.00	1.00	1.00	1814
21	1.00	1.00	1.00	1814
22	1.00	1.00	1.00	1814
23	1.00	1.00	1.00	1815
accuracy			0.96	21773

3.4.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) classifier attempts to make a decision boundary so that the separation between classes is as wide as possible. SVM algorithm finds the points (i.e., support vectors) that are closest to the line between the classes. The distance between line and support vectors is called the margin, and the goal of the algorithm is maximizing margin. The optimal hyperplane is the one with the largest margin (See Figure 3-16) [29].

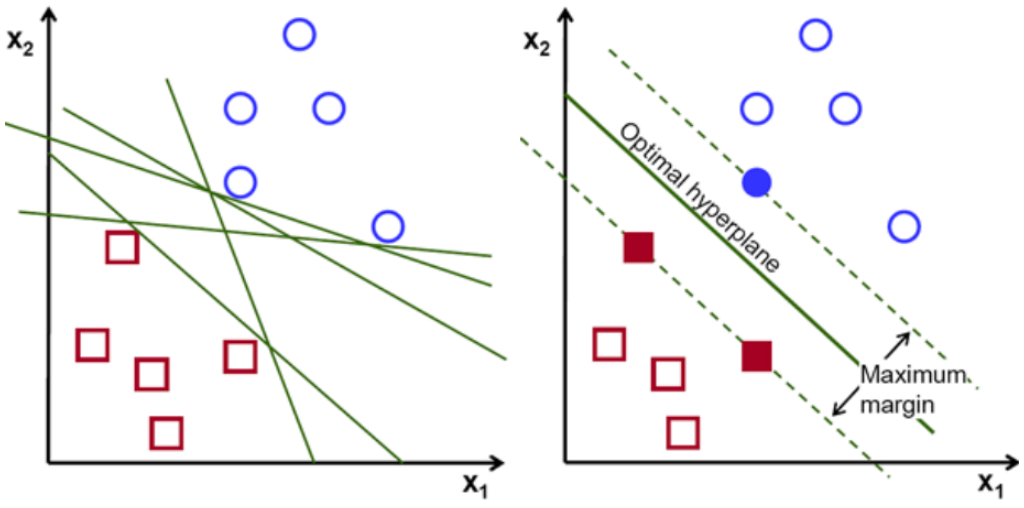

Figure 3-16. Visualization of SVM.

To build an SVM classifier in Python, sklearn.svm.SV class from scikit-learn library was used, and the default parameters were passed to the class. The model was trained using training data, and then evaluated using test data. The model predicted 96.55% of the test data samples correctly. Figure 3-17 shows the confusion matrix, and Table 3 shows the classification report. SVM classified approximately 15% of the data samples belonging to 14 bpm class as 15 bpm, and vice versa. In fact, SVM's performance was similar to KNN performance.

Table 3. SVM Classification Report

	precision	recall	f1-score	support
12	1.00	0.99	0.99	1814
13	1.00	0.99	0.99	1814
14	0.85	0.84	0.85	1814
15	0.84	0.85	0.85	1815
16	1.00	0.99	0.99	1815
17	0.99	0.99	0.99	1814
18	1.00	0.99	0.99	1815
19	1.00	0.99	0.99	1815
20	1.00	0.99	1.00	1814
21	1.00	0.99	1.00	1814
22	1.00	0.98	0.99	1814
23	0.92	1.00	0.96	1815
accuracy			0.97	21773

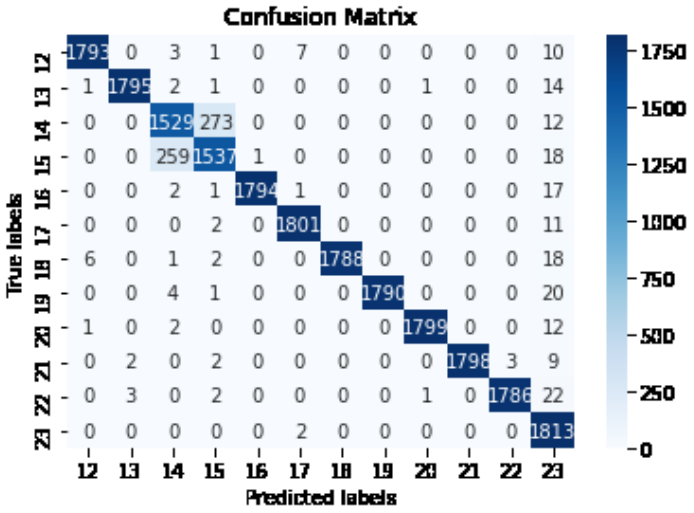

Figure 3-17. SVM confusion matrix.

3.4.3 Decision Tree

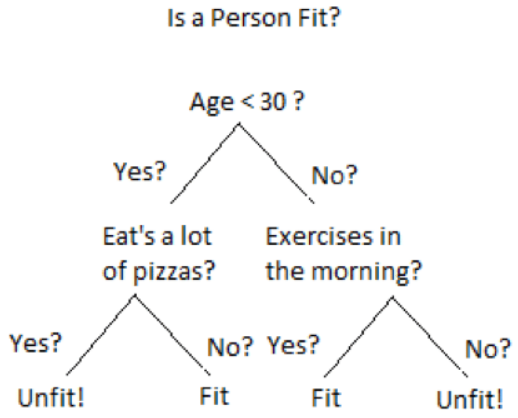

Figure 3-18. Decision tree example.

The goal of a decision trees is creating a predictive model by using learning decision rules, in form of if-then-else statements inferred from the data features. Decision trees build a classification model in the form of a tree structure. An example of a decision tree is shown in Figure 3-8 [30]. Decision trees are built using an algorithm that determines how to split a data set based on different conditions. Constructing a decision tree includes deciding on which features to choose and which conditions should be used for splitting. Various split points are tested, and then a cost function is used to select the best splits. The final decision tree consists of decision nodes, which are split into branches and leaf nodes that cannot be further split, representing a classification or decision. The root node is the uppermost decision node and also the best predictor. The longest path from a root to a leaf defines the depth of a decision tree. The complexity of decision rules and fitness of the model increases with the depth of the tree.

Scikit-learn's DecisionTreeClassifier class was used to build a decision tree classifier in Python. This function takes criterion as a parameter, which measures the

quality of a split. Gini impurity—a measure of how often a randomly chosen element from the set would be incorrectly labeled—was used as a criterion. The maximum tree depth was set to *none*, meaning that the nodes are expanded until all leaves are pure or until all leaves contain less than the minimum number of samples required to split an internal node. The minimum number of samples required to split an internal node was set to the default number of 2, and the minimum number of samples required to be at a leaf node was set to the default number of 1. The model was trained using training data. Model accuracy on predictions made using test data was 88.84%. The corresponding confusion matrix is shown in Figure 3-19, and the classification report is shown in Table 4.

Table 4. Decision Tree Classification Report

	precision	recall	f1-score	support
12	0.92	0.90	0.91	1814
13	0.93	0.92	0.92	1814
14	0.71	0.74	0.72	1814
15	0.73	0.71	0.72	1815
16	0.90	0.92	0.91	1815
17	0.93	0.92	0.92	1814
18	0.93	0.92	0.92	1815
19	0.93	0.93	0.93	1815
20	0.92	0.92	0.92	1814
21	0.93	0.93	0.93	1814
22	0.94	0.93	0.93	1814
23	0.91	0.92	0.91	1815
accuracy			0.89	21773

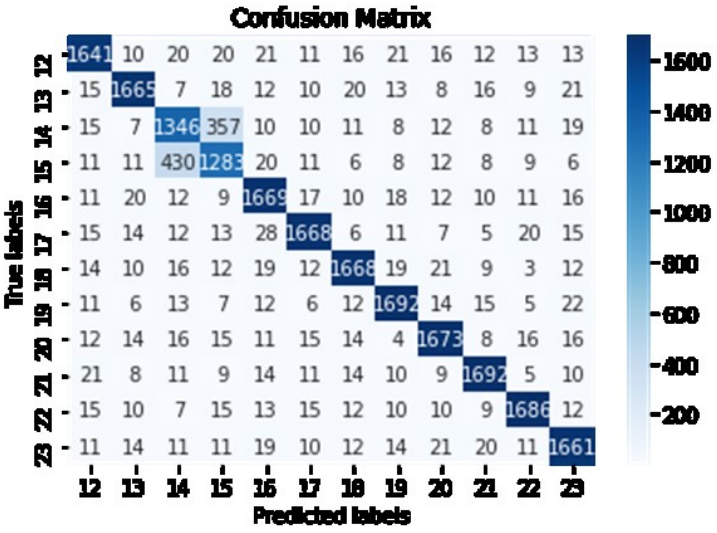

Figure 3-19. Decision tree confusion matrix.

Decision tree classifier achieved an accuracy less than 10% lower than SVM and KNN. Similar to KNN and SVM performance, decision tree classified approximately 20% of the 14 bpm data samples as 15 bpm, and vice versa. A section of a resulting decision tree is displayed in Figure 3-20, even though this particular decision tree is too complex to visualize or interpret.

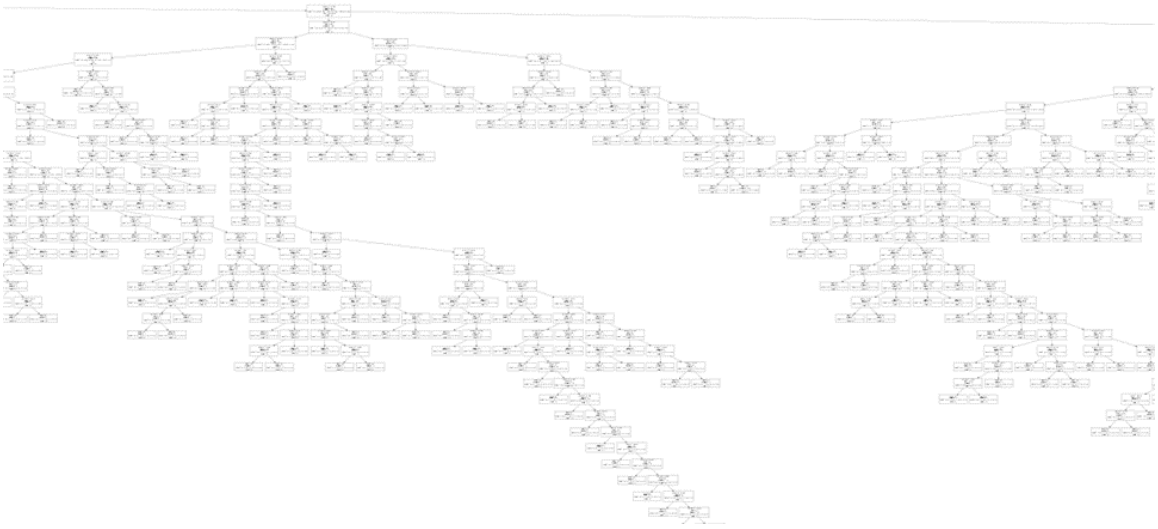

Figure 3-20. Section of the decision tree.

3.4.4 Random Forest

Random forest consists of several individual decision trees that operate as an ensemble. Each tree in the random forest predicts a class, and the class with the most votes is model's prediction. Figure 3-21 shows an example of how a random forest model makes a prediction [31]. RandomForestClassifier class from scikit-learn library was used to build the classifier. The number of trees in the forest was the default number of 100. Gini impurity was used to measure the quality of the split. The maximum depth of the trees was not specified; hence, the nodes are expanded until all leaves are pure or until all leaves contain less than two samples. After the model was trained and evaluated using test data, a

96.28% accuracy was achieved. Figure 3-22 shows the confusion matrix, and the classification report is shown in Table 5. Results are consistent with KNN, SVM, and decision tree results.

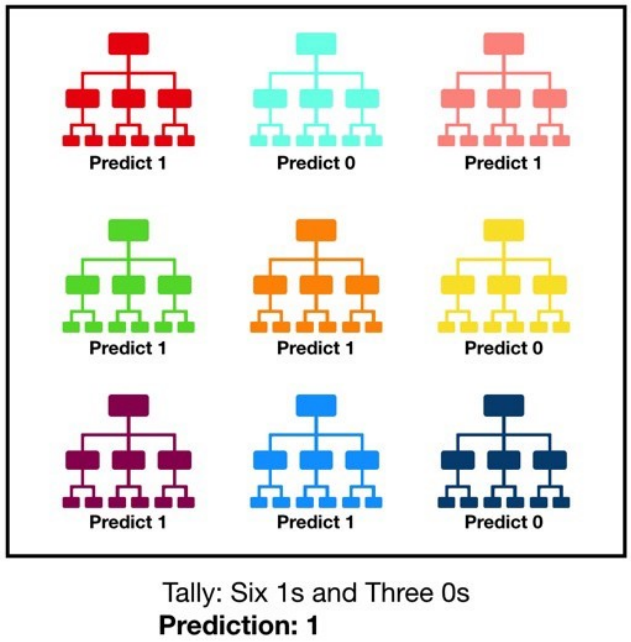

Figure 3-21. Visualization of a random forest model making a prediction

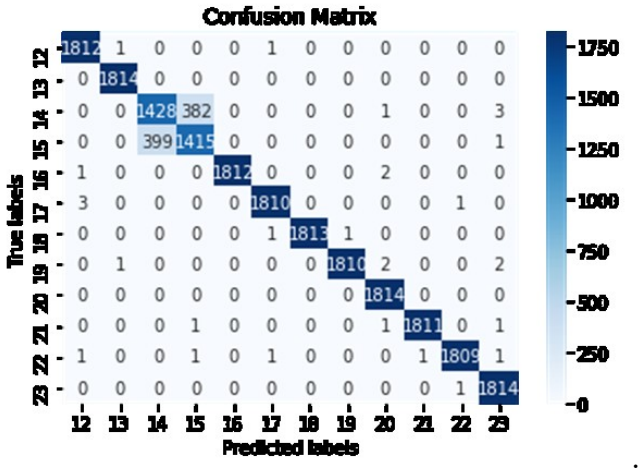

Figure 3-22. Random forest confusion matrix.

Table 5. Random Forest Classification Report

	precision	recall	f1-score	support
12	1.00	1.00	1.00	1814
13	1.00	1.00	1.00	1814
14	0.78	0.79	0.78	1814
15	0.79	0.78	0.78	1815
16	1.00	1.00	1.00	1815
17	1.00	1.00	1.00	1814
18	1.00	1.00	1.00	1815
19	1.00	1.00	1.00	1815
20	1.00	1.00	1.00	1814
21	1.00	1.00	1.00	1814
22	1.00	1.00	1.00	1814
23	1.00	1.00	1.00	1815
accuracy			0.96	21773

3.4.5 Naïve Bayes

Naive Bayes classifier uses Bayes Theorem, expressed by Eq. 6. In Eq. 6, A and B are events and P(B) is not equal to zero. P(A|B) and P(B|A) are conditional probabilities. P(A|B) is the probability of event A occurring, given that B is true. P(A) and P(B) are probabilities of A and B occurring, respectively. For every data point, Naïve Bayes classifier predicts membership probabilities for each class. The class with the highest probability is the predicted class. Naïve Bayes classifier assumes that features are unrelated to each other, and conditionally independent [32]. Gaussian Naïve Bayes assumes that the probability of features is Gaussian, which is described in Eq. 7. In Eq. 7, x_i represents the features, and y represents the output. The parameters σ_y and μ_y are estimated using maximum likelihood.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \quad \text{Eq. 6}$$

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \quad \text{Eq. 7}$$

GaussianNB class from Pyhton's scikit-learn library was used to build a Naive Bayes model. The model was trained on training data and evaluated on test data. The model achieved 15.43% accuracy, which is significantly lower than model accuracy for KNN, SVM, decision tree, and random forest classifier. The confusion matrix is shown in Figure 3-23, revealing that a large number of data samples for every class has been misclassified as a 14 bpm class. The classification report is shown in Table 6. Naive Bayes classifier did not perform well because of the "naive" assumption of conditional independence between every pair of features, given the value of the class variable.

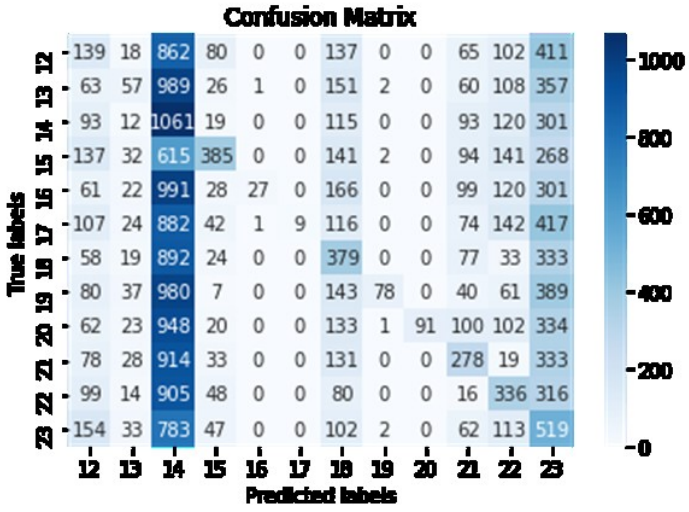

Figure 3-23. Naive Bayes confusion matrix.

Table 6. Naive Bayes Classification Report

	precision	recall	f1-score	support
12	0.12	0.08	0.09	1814
13	0.18	0.03	0.05	1814
14	0.10	0.58	0.17	1814
15	0.51	0.21	0.30	1815
16	0.93	0.01	0.03	1815
17	1.00	0.00	0.01	1814
18	0.21	0.21	0.21	1815
19	0.92	0.04	0.08	1815
20	1.00	0.05	0.10	1814
21	0.26	0.15	0.19	1814
22	0.24	0.19	0.21	1814
23	0.12	0.29	0.17	1815
accuracy			0.15	21773

3.4.6 Logistic Regression

The logistic regression classification algorithm uses a Sigmoid function, expressed by Eq. 8, as the prediction function that returns a probability value that can then be mapped into discrete classes. The output of the Sigmoid function are the probability estimates that fall between 0 and 1. To predict the label of a data point, the class with the highest score or probability is chosen [33].

$$S(z) = \frac{1}{1+e^{-z}} \qquad \text{Eq. 8}$$

Class sklearn.linear_model.LogisticRegression was used to implement logistic regression in Python,. After training the model, test data evaluation showed 97.46% accuracy. The confusion matrix is shown in Figure 3-24, and the classification report is shown in Table 7. Model performance was very similar to KNN, SVM, decision tree, and random forest classifiers.

Table 7. Logistic Regression Classification Report

	precision	recall	f1-score	support
12	0.99	1.00	0.99	1814
13	1.00	1.00	1.00	1814
14	0.86	0.87	0.87	1814
15	0.87	0.86	0.86	1815
16	1.00	1.00	1.00	1815
17	0.99	1.00	1.00	1814
18	1.00	0.99	1.00	1815
19	1.00	1.00	1.00	1815
20	1.00	1.00	1.00	1814
21	1.00	1.00	1.00	1814
22	0.99	1.00	1.00	1814
23	1.00	0.99	1.00	1815
accuracy			0.97	21773

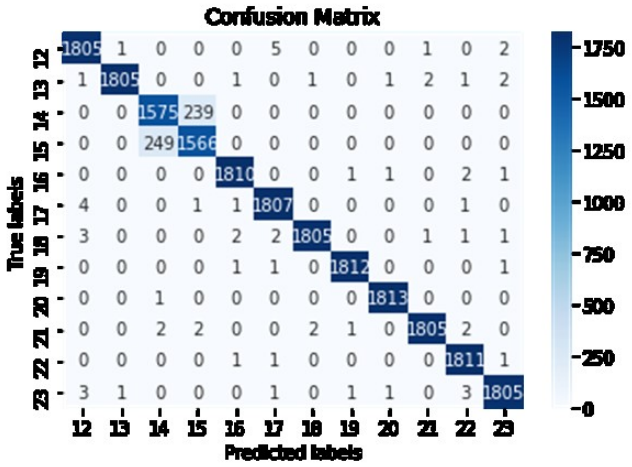

Figure 3-24. Logistic regression confusion matrix.

3.4.7 Neural Network Model (MLP)

Multilayer perceptron (MLP) is a class of feedforward neural network (FFNN)—a classification algorithm that learns a function $f(\cdot): R^m \rightarrow R^o$ by training on data, where m is the number of dimensions for input, and o is the number of dimensions for output. Using a set of features $X = x_1, x_2, \ldots x_m$ and a target y, MLP learns a non-linear approximator for classification. It differs from logistic regression because it has one or more hidden layers between the input and the output layer. There can be many such hidden layers making the architecture deep. Figure 3-25 shows the architecture of MLP with one hidden layer.

There are three steps in training the MLP model, which are forward pass, calculating the loss, and the backward pass. In the forward pass, input is passed to the model. At each layer, the input received to the layer is multiplied with weights and bias is added. In fact, each neuron in the hidden layer uses a weighted linear summation $x_1 w_1 + x_2 w_2 + \cdots + x_m w_m$ to transform values from the previous layer. Model output is calculated using an activation function. The second step in MLP training is calculating error or loss by comparing the output with the ground truth labels. The third step is a backward pass,

where backpropagation is used from the output layer to the previous layers to minimize loss function. The gradient descent is used to update the weights [34]. The algorithm stops when the preset number of maximum iterations is reached, or when the loss falls below a certain threshold.

Scikit-learn's class MLPClassifier implements an MLP algorithm, which uses backpropagation for training. The number of hidden layers was set to 4, and the number of neurons in hidden layers were 200, 150, 100, and 50, respectively. The activation function for hidden layers was relu, which is the rectified linear unit function expressed by Eq. 9.

$$f(x) = \max(0, x) \qquad \text{Eq. 9}$$

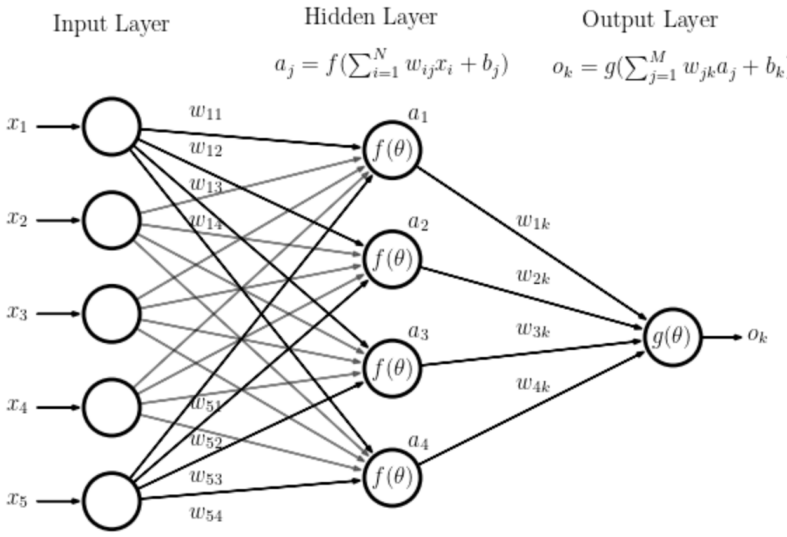

Figure 3-25. MLP with one hidden layer,

Adam, which is a stochastic gradient-based optimizer, was chosen as the solver for weight optimization. MLP trained on two arrays: training data array of size (n_samples, n_features), and array y of size (n_samples,), which consists of class labels for training data samples. The model achieved 97.3% accuracy. The confusion matrix is shown in

Figure 3-26, and the classification report is shown in Table 8. Model performance is similar to other models, with the exception of Naïve Bayes.

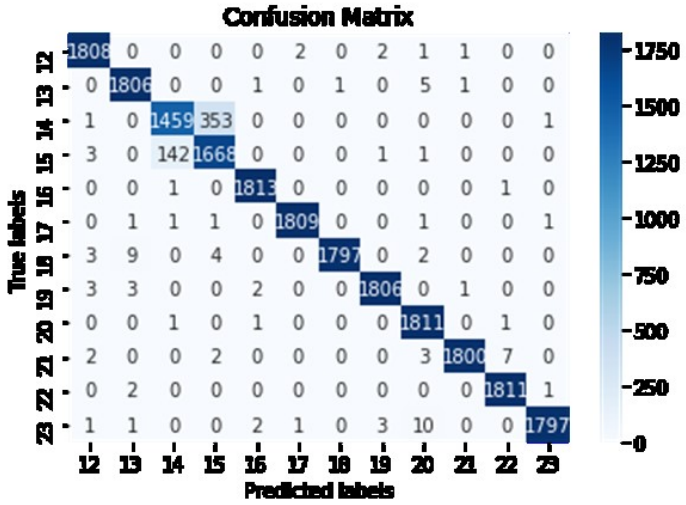

Figure 3-26. MLP confusion matrix.

Table 8. MLP Classification Report

	precision	recall	f1-score	support
12	0.99	1.00	0.99	1814
13	0.99	1.00	0.99	1814
14	0.91	0.80	0.85	1814
15	0.82	0.92	0.87	1815
16	1.00	1.00	1.00	1815
17	1.00	1.00	1.00	1814
18	1.00	0.99	0.99	1815
19	1.00	1.00	1.00	1815
20	0.99	1.00	0.99	1814
21	1.00	0.99	1.00	1814
22	1.00	1.00	1.00	1814
23	1.00	0.99	0.99	1815
accuracy			0.97	21773

3.4.8 Summary of Classification Modelling Results

Model accuracies are summarized in Figure 3-27. Logistic Regression proved to be the model with the highest accuracy, followed by MLP, SVM, KNN, and random forest. Together with the decision tree, these often misclassify 14 bpm as 15 bpm, and vice versa. Naive Bayes classifier achieved significantly lower accuracy than other models, primarily

because Naïve Bayes assumes that all predictors are independent of the others. This is a very strong assumption and it would be difficult to claim that it is realistic for this problem.

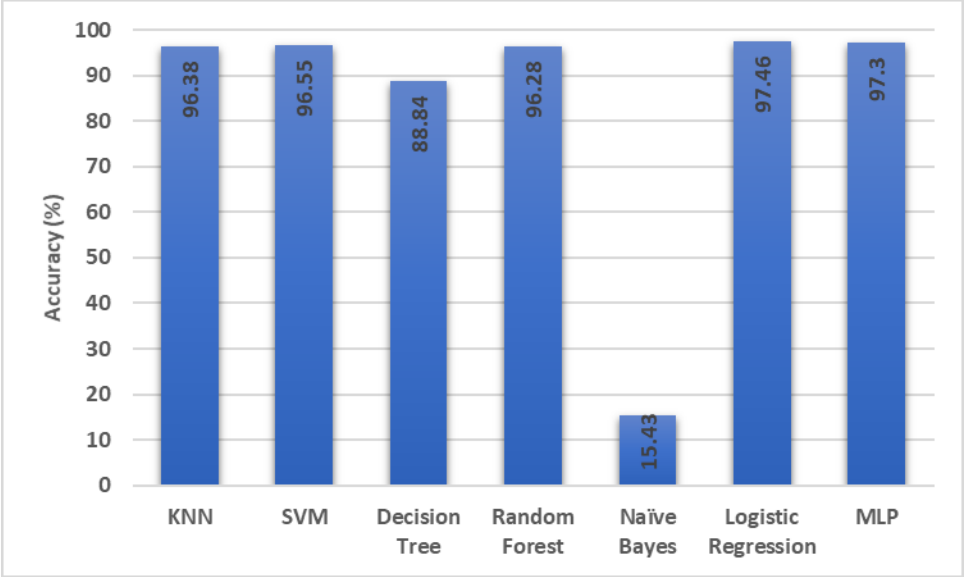

Figure 3-27. Classification model accuracies.

3.5 Regression Models to Estimate RR

Estimating RR is a regression problem because RR variable takes continuous values or real numbers. Regression models were trained using collected CSI data to estimate the RR of a subject breathing and sitting still between the transmitting and receiving routers. Models were trained on 70% of the collected data and tested on the remaining 30% of the collected data. Logistic Regression, SVM, decision tree, and random forest algorithms, which were explained in Section 3.4, can also be used for regression. Additional regression algorithms for developing the models include Linear Regression, Least Absolute Shrinkage, and Selection Operator (LASSO). To evaluate regression model performance, the following four metrics were used, namely Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared. R-squared is the proportion of variance in the

observed data that is explained by the model. R-squared normally takes values between 0 and 1. Values closer to 1 are superior, as more variance is explained by the model. Metrics are calculated using the following formulas:

$$MAE = \frac{1}{n}\sum_{i,j=1}^{n} |y_j - y_i| \quad \text{Eq. 10}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i,j=1}^{n} \left(y_j - y_i\right)^2} \quad \text{Eq. 11}$$

$$\text{and } R^2 = \frac{SS_{reg}}{SS_{tot}} = \frac{\sum_j (y_j - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}, \quad \text{Eq. 12}$$

where y_j refers to predicted labels, and y_i refers to true labels. SS_{reg} is the regression sum of squares (i.e., explained sum of squares), and SS_{tot} is the total sum of squares, which is proportional to the variance of the data.

3.5.1 Linear Regression

Linear regression is a linear model, meaning it assumes a linear relationship between input variables and a single output variable. More specifically, a target can be calculated from a linear combination of the features. For example, a model for a simple linear regression problem with a single feature x and target y can be expressed by

$$y = \beta_0 + \beta_1 x, \quad \text{Eq. 13}$$

where β_0 and β_1 are model parameters, referred to as regression coefficients. The goal of linear regression is finding regression coefficients that minimize a cost function. The class sklearn.linear_model.LinearRegression was used to implement linear regression in Python. The model was fit using training data, and then used to make predictions on test data. Figure 3-28 shows true values and predicted values for the first 10 test-data samples. Regression

metrics are shown in Table 9. Notably, the value of root mean squared error is 2.17, which is more than 10% of the mean value for RR, which is 17.5. This means that although the linear regression algorithm is not very accurate, it is able to make reasonably good predictions.

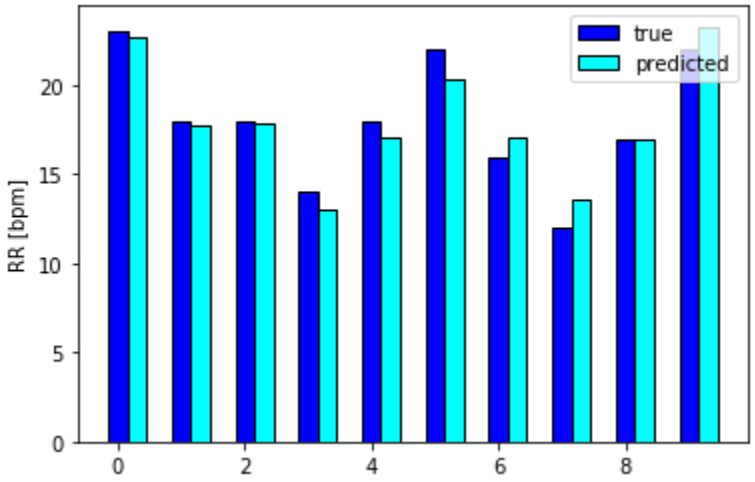

Figure 3-28. Linear regression estimated and true values

Table 9. Linear Regression Metrics

Mean Absolute Error	1.69
Root Mean Squared Error	2.17
R-squared	0.60

3.5.2 LASSO

LASSO is a type of linear regression that uses shrinkage, where data values are shrunk towards a central point. LASSO regression performs L1 regularization, which limits the size of coefficients. L1 penalty equals to the absolute value of coefficient magnitude. The goal of LASSO is minimizing Eq. 15, which is the sum of squares with a constraint on coefficients.

$$Function\ to\ minimize = \sum_{i=1}^{n}\left(y_i - \sum_j x_{ij}\beta_j\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \qquad \text{Eq. 14}$$

Regression metrics of LASSO are shown in Table 10. Since R-squared is extremely close to 0, the model explains none of the observed data variance. MAE and RMSE are greater when compared with linear regression, rendering LASSO estimation less accurate. Figure 3-29 shows predicted and true RR values for the first 10 test data samples. LASSO estimated RR at 17.5 bpm for the entire test data; hence, it cannot be used to accurately estimate RR.

Table 10. LASSO Regression Metrics

Mean Absolute Error	3.0
Root Mean Squared Error	3.45
R-squared	4.43e-9

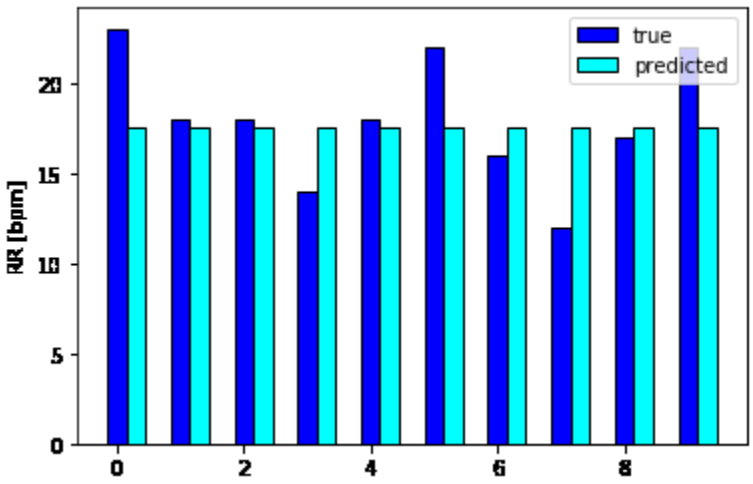

Figure 3-29. LASSO regression estimated and true values.

3.5.3 Summary of Regression Modelling Results

Regression metrics for all models are listed in Table 11. Considering errors and R-squared value, logistic regression offers superior performance, and LASSO offers the worst. SVR, decision tree, and random forest regressors also accurately estimated RR based on test data samples, as evidenced by insignificant error and an R-squared value close to 1.

Table 11. Regression Metrics for All Regression Models

Regression Model	MAE	RMSE	R-squared
Linear Regression	1.69	2.17	0.60
LASSO	3.0	3.43	4.43e-9
SVR	0.58	1.16	0.89
Decision Tree Regressor	0.35	1.32	0.85
Random Forest Regressor	0.34	0.68	0.96
Logistic Regression	0.04	0.35	0.97

3.6 Evaluating Classification and Regression Models Using a Blind Test Set

Data collected from all four subjects was divided into 70% training- and 30% test-data. Most models showed promising performance when evaluated using test-data. To determine if models could perform similarly when making predictions for a new and unknown subject, data collected from subjects A, B, and D were used to train the models. Data collected from subject C was then used as test-data to evaluate the models. Subject C data served as a blind test data, as none of the models had seen samples from the data. The

training data consisted of 54,432 data samples, each with 500 attributes from three subject. The test data was 18,144 data observations, with 500 attributes from a single subject. After training data was used to fit the models, predictions were made on test data.

All classification models achieved approximately 7 to 8% accuracy using the blind test data. Results were significantly lower than when models were trained using 70% data from all subjects. Comparisons are shown in Figure 3-30. Table 12 shows the regression metrics of the regression models obtained when evaluated on blind test-data. Regression models showed significantly higher errors when compared with previous testing. In addition, R-squared values were negative, zero, or close to zero, meaning that models performed very poorly.

Results show that models cannot be generalized, as they are unable to adapt properly to new and previously unseen subjects. The reason for this could be due to the fact that the data used for training was collected from only three subjects. To make models more generalized, data should be collected from a large number of subjects. Also, models were not generalizable because in this test, each subject significantly changed the signal multi-path in a different way.

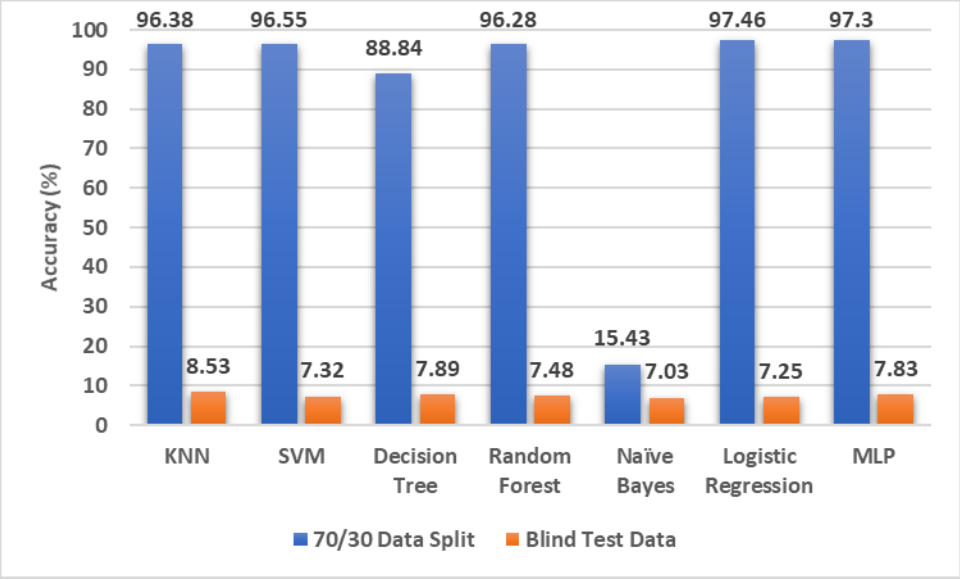

Figure 3-30. Comparison of performance when 70/30 split was used vs. blind test data.

Table 12. Regression Metrics for All Regression Models – Blind Test Set

Regression Model	**MAE**	**RMSE**	**R-squared**
Linear Regression	4.61	6.04	-2.07
LASSO	3.0	3.45	0.00
SVR	3.36	4.10	-0.41
Decision Tree Regressor	3.87	4.74	-0.88
Random Forest Regressor	3.04	3.56	-0.06
Logistic Regression	4.51	5.46	0.07

Chapter 4: Comparison with an Existing Pattern-based System

4.1 Description of the Pattern-based System

One of the existing pattern-based system that uses Wi-Fi CSI data for estimating RR was developed by a previous OU student, Mohamad Omar Al Kalaa. He developed a full control, processing and RR estimation Matlab program.

4.1.1 Tx and Rx Configuration

In this program, a new cmd process was started, and the ssh command was repeatedly executed for logging into the routers. After exiting this process, the program returned to the Matlab process. The original Atheros tool source code for sending and receiving data to obtain CSI was modified. The sender and receiver programs were placed in the root directory of the transmitting and receiving router, respectively. The Matlab program first started the CSI receiving program on the Rx device, and then commenced transmission from the Tx device. Packet transmission rate was 10 packets per second, for a total of 60 seconds. Data acquisition was then performed. The program included signal processing, stream selection, and RR estimation.

4.1.2 Pre-processing

Raw CSI data comprised a complex 3x3x56 matrix for each packet received. Absolute values of data were obtained to determine the CSI amplitude values. Data was then resampled to 10 Hz, and the negative values were removed. CSI amplitude values were converted to dB scale, and the DC component was removed. Finally, the signal was smoothed with a moving average using 10 samples, which is the message frequency.

4.1.3 Stream Selection

During the stream selection process, CSI amplitude streams were selected on subcarriers most sensitive to breathing. Each time subcarrier selection was performed, a signal with maximum power in the frequency band (i.e., frequency limits 0.1 and 1.6 Hz) was selected. The remainder of subcarriers were filtered out and not used for RR estimation. The program offers real-time RR monitoring, in which RR is reported every second after 10 seconds. In this case, a new subcarrier was selected and used for RR estimation in every 10 second window. The program also offers post-processing, where RR is estimated after the data collection. During post-processing, only one subcarrier was selected over the entire test period and used for RR estimation.

4.1.4 RR Estimation

To calculate the RR, CSI amplitude stream peaks are determined so that the minimum peak-to-peak distance is one sample and minimum peak prominence is one dB. Setting minimum peak-to-peak distance and prominence helps to prevent fake peaks that are not caused by breathing. Inter-breath interval (IBI) method was used to obtain an RR estimate. First, the mean time difference between the peaks is calculated. Then, the inverse of the mean is computed to obtain the estimated RR. Given that the peaks are found at time $\tau_1, \tau_2, \dots, \tau_n$, then the estimated RR is calculated using the following formula,

$$\mathrm{RR} = \frac{60}{\frac{1}{n-1}\sum_{i=1}^{n-1}(\tau_{i+1} - \tau_i)}, \qquad \text{Eq. 15}$$

where the factor of 60 is used to convert the frequency from Hz to bpm. Examples of post-processing and real-time RR estimation program output are shown in Figure 4-1 and Figure 4-2, respectively. These examples are test outputs for a subject breathing at a rate of 19

bpm for a time period of 60 seconds. The real-time system showed the last 10-second window. The legend for both graphs shows the index of a selected subcarrier.

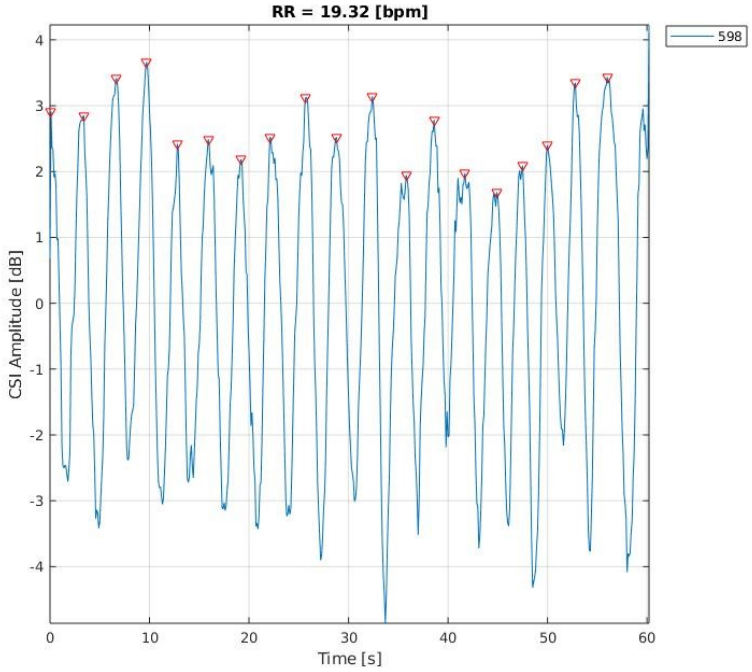

Figure 4-1. Post-processing RR estimation example.

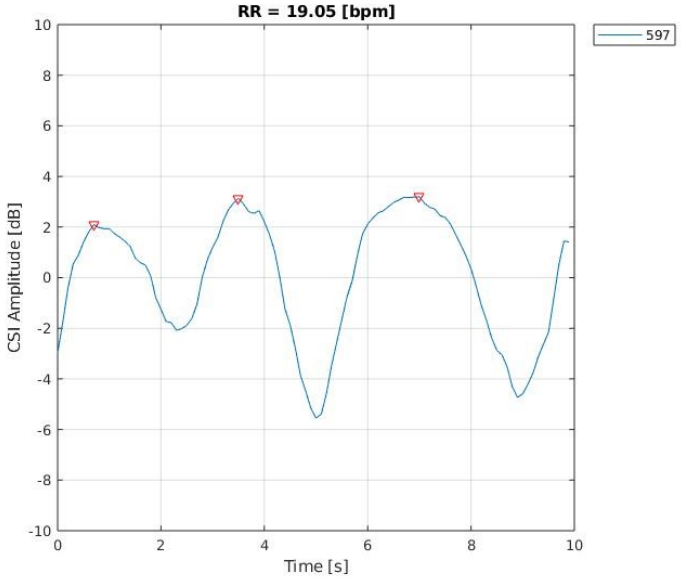

Figure 4-2. Real-time RR monitoring example.

4.2 Evaluation of the Pattern-based System and Comparison with the Regression Models

The pattern-based system described in Section 4.1 was used to estimate RR using the collected data to train and evaluate the machine learning models described in Chapter 3. The system was evaluated using regression metrics described in Section 3.5. Both post-processing and real-time RR estimation were evaluated. Since real-time system reports an RR value every second, the estimated RR represents the average value over 60 seconds. Results are shown in Table 13 and show that the real-time system estimated RR values slightly better than when post-processing was used. The reason for these results is due to the fact that real-time monitoring selects a new subcarrier most sensitive to breathing every 10 seconds, and during post-processing, only one subcarrier is selected for RR estimation.

Table 13. Pattern-based System Regression Metrics

Pattern-based System	MAE	RMSE	R-squared
Postprocessing RR Estimation	2.14	2.9	0.49
Real-time RR Estimation	0.78	2.7	0.60

When comparing regression models (See Table 11), all—except for LASSO—have lower error and higher R-squared value than the pattern-based system. This indicates that learning-based models perform better when estimating RR based on the collected CSI data. One drawback of regression models is that they cannot be generalized, thus no ability to accurately predict blind test data from a new subject. It is likely, however, performance could improve with additional training data.

Conclusion and Future Work

The work in this thesis presented machine learning models used to predict RR using CSI collected from four human subjects who were breathing while sitting still in a chair located between two WiFi routers. Seventy percent of the collected data used for training classification and regression models; the remaining 30% was used for evaluating the models. Model accuracy was used to evaluate classification models. The logistic regression model demonstrated the highest accuracy rate of 97.46%. Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared metrics were used to evaluate regression models. Logistic regression also demonstrated both the lowest error and the highest R-squared. Regression model performance was compared with existing pattern-based system performance. Results show that with the exception of LASSO, all regression models were able to predict RR more accurately than pattern-based systems. Drawbacks of models developed for this thesis are that they cannot be generalized, and they do not perform well when predicting RR using unseen CSI data collected from a new subject.

In future work, CSI data used to train the models should be collected from a large number of subjects. Accordingly, models might show improvement and become more generalizable. Also, the data should be collected in various environments under different scenarios with a variety of Tx-Rx distances, subject body positions, and orientations. Because the work in this thesis focused on predicting RR for a single subject, future work should consider multi-user scenarios.

References

[1] Cleveland Clinic, “Vital Signs,” 2019.

[2] S. Lapi *et al.*, “Respiratory rate assessments using a dual-accelerometer device,” *Respir. Physiol. Neurobiol.*, vol. 191, pp. 60–66, Jan. 2014, doi: 10.1016/J.RESP.2013.11.003.

[3] S. Shi, Y. Xie, M. Li, A. X. Liu, and J. Zhao, “Synthesizing Wider WiFi Bandwidth for Respiration Rate Monitoring in Dynamic Environments,” *Proc. - IEEE INFOCOM*, vol. 2019-April, pp. 181–189, 2019, doi: 10.1109/INFOCOM.2019.8737553.

[4] WHO, “Noncommunicable diseases,” *Fact sheet*, 01-Jun-2018.

[5] C. Massaroni, A. Nicolò, D. Lo Presti, M. Sacchetti, S. Silvestri, and E. Schena, “Contact-based methods for measuring respiratory rate,” *Sensors (Switzerland)*, vol. 19, no. 4, pp. 1–47, 2019, doi: 10.3390/s19040908.

[6] M. Bartula, T. Tigges, and J. Muehlsteff, “Camera-based system for contactless monitoring of respiration,” *Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBS*, pp. 2672–2675, 2013, doi: 10.1109/EMBC.2013.6610090.

[7] Y. Ren, C. Wang, J. Yang, and Y. Chen, “Fine-grained sleep monitoring: Hearing your breathing with smartphones,” in *Proceedings - IEEE INFOCOM*, 2015, doi: 10.1109/INFOCOM.2015.7218494.

[8] P. Hillyard *et al.*, “Comparing Respiratory Monitoring Performance of Commercial Wireless Devices,” 2018.

[9] A. D. Droitcour, O. Boric-Lubecke, and G. T. A. Kovacs, “Signal-to-noise ratio in doppler radar system for heart and respiratory rate measurements,” *IEEE Trans. Microw. Theory Tech.*, 2009, doi: 10.1109/TMTT.2009.2029668.

[10] S. Venkatesh, C. R. Anderson, N. V. Rivera, and R. M. Buehrer, “Implementation and analysis of respiration-rate estimation using impulse-based UWB,” in *Proceedings - IEEE Military Communications Conference MILCOM*, 2005, doi: 10.1109/MILCOM.2005.1606167.

[11] F. Adib, H. Mao, Z. Kabelac, D. Katabi, and R. C. Miller, “Smart homes that monitor breathing and heart rate,” *Conf. Hum. Factors Comput. Syst. - Proc.*, vol. 2015-April, pp. 837–846, 2015, doi: 10.1145/2702123.2702200.

[12] H. Abdelnasser, K. A. Harras, and M. Youssef, “UbiBreathe: A ubiquitous non-invasive wifi-based breathing estimator,” in *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2015, doi: 10.1145/2746285.2755969.

[13] N. Mostahinic and H. Refai, “Spectrum Occupancy for 802 . 11a / n / ac Homogeneous and Heterogeneous Networks,” *2019 15th Int. Wirel. Commun. Mob. Comput. Conf.*, pp. 1690–1695, 2019.

[14] Y. Xie, Z. Li, and M. Li, “Precise power delay profiling with commodity WiFi,” in *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM*, 2015, doi: 10.1145/2789168.2790124.

[15] Y. Ma, G. Zhou, and S. Wang, “WiFi sensing with channel state information: A survey,” *ACM Comput. Surv.*, vol. 52, no. 3, 2019, doi: 10.1145/3310194.

[16] F. Zhang *et al.*, “From Fresnel Diffraction Model to Fine-grained Human Respiration Sensing with Commodity Wi-Fi Devices,” *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 2, no. 1, pp. 1–23, 2018, doi: 10.1145/3191785.

[17] X. Liu, J. Cao, S. Tang, and J. Wen, “Wi-sleep: Contactless sleep monitoring via WiFi signals,” *Proc. - Real-Time Syst. Symp.*, vol. 2015-Janua, no. January, pp. 346–355, 2015, doi: 10.1109/RTSS.2014.30.

[18] X. Liu, J. Cao, S. Tang, J. Wen, and P. Guo, “Contactless Respiration Monitoring Via Off-the-Shelf WiFi Devices,” *IEEE Trans. Mob. Comput.*, vol. 15, no. 10, pp. 2466–2479, 2016, doi: 10.1109/TMC.2015.2504935.

[19] J. Liu, Y. Wang, Y. Chen, J. Yang, X. Chen, and J. Cheng, “Tracking vital signs during sleep leveraging off-the-shelf WiFi,” *Proc. Int. Symp. Mob. Ad Hoc Netw. Comput.*, vol. 2015-June, pp. 267–276, 2015, doi: 10.1145/2746285.2746303.

[20] C. Wu, Z. Yang, Z. Zhou, X. Liu, Y. Liu, and J. Cao, “Non-invasive detection of moving and stationary human with WiFi,” *IEEE J. Sel. Areas Commun.*, vol. 33, no. 11, pp. 2329–2342, 2015, doi: 10.1109/JSAC.2015.2430294.

[21] X. Wang, C. Yang, and S. Mao, “TensorBeat: Tensor Decomposition for Monitoring Multiperson Breathing Beats with Commodity WiFi,” *ACM Trans. Intell. Syst. Technol.*, vol. 9, no. 1, pp. 1–28, 2017, doi: 10.1145/3078855.

[22] X. Wang, C. Yang, and S. Mao, “PhaseBeat: Exploiting CSI Phase Data for Vital Sign Monitoring with Commodity WiFi Devices,” *Proc. - Int. Conf. Distrib. Comput. Syst.*, pp. 1230–1239, 2017, doi: 10.1109/ICDCS.2017.206.

[23] H. Wang *et al.*, “Human respiration detection with commodity WiFi devices: Do user location and body orientation matter?,” *UbiComp 2016 - Proc. 2016 ACM Int. Jt. Conf. Pervasive Ubiquitous Comput.*, pp. 25–36, 2016, doi: 10.1145/2971648.2971744.

[24] P. Wang, B. Guo, T. Xin, Z. Wang, and Z. Yu, “TinySense: Multi-user respiration detection using Wi-Fi CSI signals,” *2017 IEEE 19th Int. Conf. e-Health Networking, Appl. Serv. Heal. 2017*, vol. 2017-Decem, pp. 1–6, 2017, doi:

10.1109/HealthCom.2017.8210837.

[25] Y. Zeng, D. Wu, R. Gao, T. Gu, and D. Zhang, “FullBreathe,” *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 2, no. 3, pp. 1–19, 2018, doi: 10.1145/3264958.

[26] “NeuLog Respiration Monitor Belt logger sensor NUL-236.” [Online]. Available: https://neulog.com/respiration-monitor-belt/.

[27] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, 2011.

[28] T. Srivastava, “Introduction to k-Nearest Neighbors: A powerful Machine Learning Algorithm(with implementation in Python),” *Analytics Vidhya*, 2014. .

[29] Pier Paolo Ippolito, “SVM: Feature Selection and Kernels,” 2019.

[30] Mayur Kulkarni, “Decision Trees for Classification: A Machine Learning Algorithm,” 2017.

[31] Tony Yiu, “Understanding Random Forest: How the Algorithm Works and Why is It So Effective,” 2019.

[32] Rohith Gandhi, “Naive Bayes Classifier,” 2018.

[33] M. Kuhn and K. Johnson, *Applied predictive modeling*. 2013.

[34] Nitin Kumar Kain, “Understanding of Multilayer Perceptron (MLP),” 2018.

Appendix A: Example Images of 12 Classes for 4 Subjects

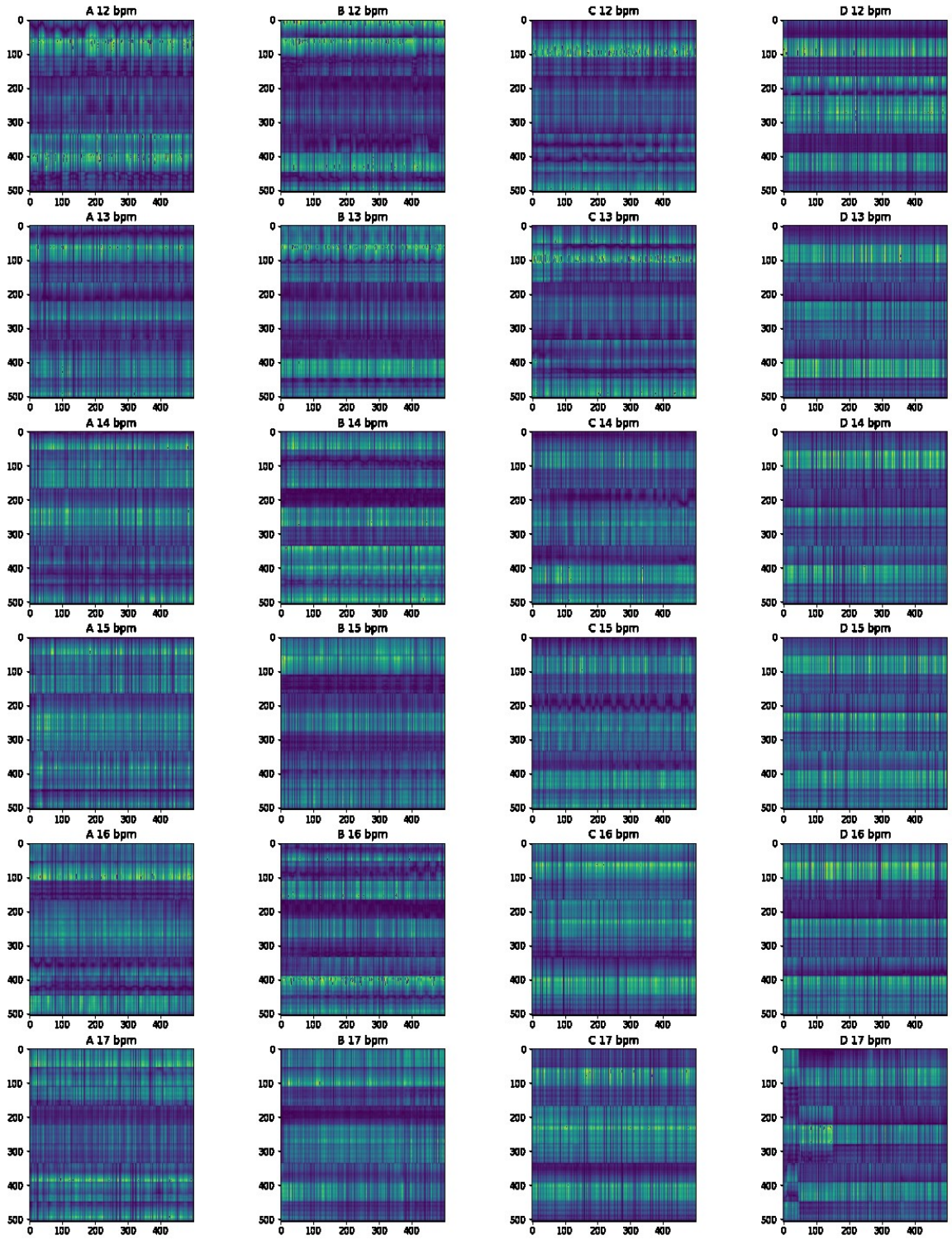

Figure A-1. Example image of classes 12-17 for all subjects.

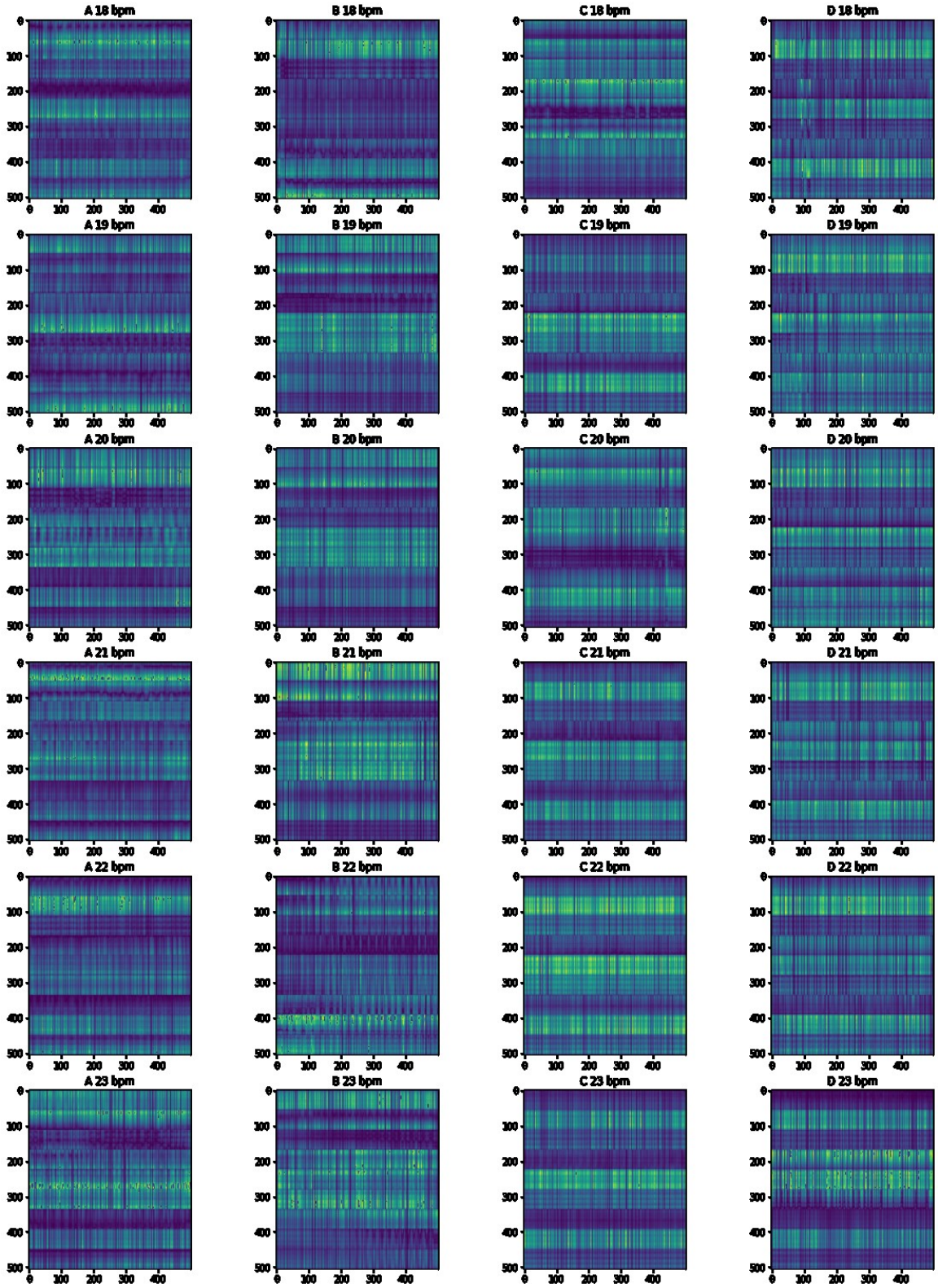

Figure A-2. Example mage of classes 17-23 for all subjects.