Machine Learning Estimation of COVID-19 Social Distance Using Smartphone Sensor Data

by

Oleksandr Semenov

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfilment of the requirements for the

Degree of Master of Science

in

Electrical and Computer Engineering

by

____

August 2021

APPROVED:

____

Professor Emmanuel Agu, Major Thesis Advisor

____

Professor Kaveh Pahlavan

ABSTRACT

COVID-19 is spread from an infected to a healthy person when they are within 6 feet from each other for longer than 15 minutes. To limit disease transmission, there is a need for technology that could identify whether subjects were near each other longer than 15 minutes. In this thesis we systematically investigate Machine Learning (ML) methods to detect proximity by analyzing data gathered from smartphones' built-in Bluetooth, accelerometer, and gyroscope sensors. We show that the proximity classification (< 6ft or not) can achieve 72%-90% accuracy using the accelerometer, 78%-84% accuracy using gyroscope sensor, and 76%-92% accuracy with the Bluetooth radio, while sensor fusion shows accuracy as high as 97%. Our model outperforms current state-of-the-art methods using neural networks and achieved Normalized Decision Cost Function (nDCF) score of 0.34 with Bluetooth radio and 0.36 with sensor fusion.

ACKNOWLEDGMENTS

This thesis is a summary of my research work under the guidance of Professor Emmanuel Agu over the past year in pursuit of my Master of Science Degree in Electrical and Computer Engineering at Worcester Polytechnic Institute.

Firstly, I would like to express my deepest gratitude to my research advisor, Professor Emmanuel Agu who guided me in this research, shared his ideas, reviewed my work, offered support and encouragement when I encountered difficulties in my research. He is a wise man and a great advisor.

I am grateful to have Professor Kaveh Pahlavan as my committee member and for providing additional guidance. Thank you, Professor Pahlavan, for reviewing my work, providing comments, and sharing your knowledge on wireless signal propagation. I also would like to say thanks to Ph.D. student Zhuoran Su and undergraduate student Haowen Wei for helping me with radio signal specific feature calculation and running experiments on the remote server.

Finally, I would like to thank my parents and my brother for their continuous support and encouragement in pursuing my graduate studies.

Contents

LIST OF FIGURES

List of Tables

CHAPTER 1 INTRODUCTION

1.1. Motivation

COVID-19 is a highly infectious airborne transmittable disease that has currently infected over 213 million people with a total of 4.44 million deaths globally to date [46], and has caused severe negative impact on global economy. The risk of airborne infectious diseases such as COVID-19 increases when healthy people are within 6 feet of infected people for longer than 15 minutes [25]. This has led to research interest in estimating the distance between smartphone users by analyzing data from smartphone built in sensors such as Bluetooth, accelerometer, and gyroscope.

1.2. Background on COVID-19

COVID-19 is a viral, highly transmittable respiratory disease of a coronavirus family of the diseases. It gets its name from crown-like spikes on the surface of the virus [37]. Coronavirus diseases cause severe acute respiratory syndrome (SARS) and Middle East respiratory syndrome (MERS) diseases in humans [37]. According to the Center for Disease Control and Prevention (CDC) people with COVID-19 may show following symptoms:

- Fever and chills
- Cough

- Shortness of breath or difficulty breathing
- Fatigue
- Muscle or body aches
- Headache
- New loss of taste or smell
- Sore throat
- Congestion or runny nose
- Nausea or vomiting
- Diarrhea

This list is not complete and it should be noted that an individual infected with COVID-19 can exhibit other symptoms [38].

The current strain of coronavirus was first discovered in Wuhan, China in December of 2019 which is why it is also commonly known under the name of SARS-CoV-19. The virus quickly spread around the world causing a global pandemic. The rapid spread of SARS-CoV-19 is partially due to a long incubation period relative to other viruses. Its incubational period can be up to 14 days with a median of 4-5 days [38]. Currently, COVID-19 can be found on all continents except Antarctica [37]. Since it first originated in 2019, the virus mutated resulting in a new strain of the virus; some strains more contagious than others. At the time of writing this thesis, the most prevalent strain of the virus is the Delta variant, which is a more transmittable and more contagious strain than the original strain first discovered in Wuhan, China. Even though, vaccines were developed and there have been to vaccinate large amounts of the global population, due to mutations of the virus it has continued to spread around the world causing new outbreaks and high rates of hospitalizations.

To prevent the spread of the virus and its mutations, CDC recommends following preventive measures:

- Get vaccinated
- Wear a mask
- Stay 6 feet from others
- Avoid crowded and poorly ventilated spaces
- Wash your hands often
- Cover coughs and sneezes
- Clean and disinfect
- Monitor your health daily [39].

Figure 1 shows the infographic developed by the CDC which advertises behavior recommended to stop the spread of the virus.

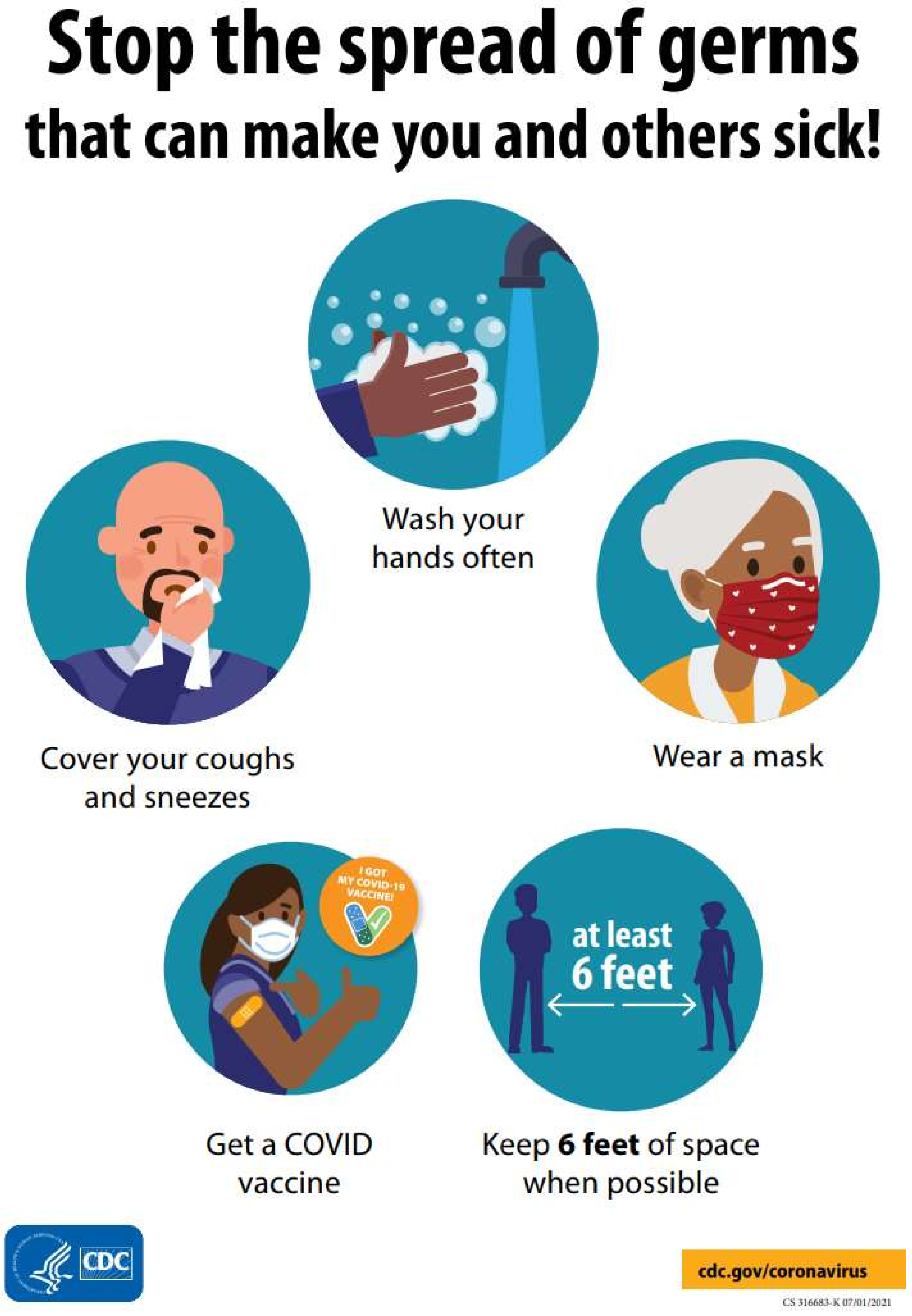

Figure 1 Stop the spread infographic [40].

1.3. Contact Tracing

Since SARS-CoV-19 is a respiratory transmittable disease, it spreads from an infected to a healthy person through airborne particles, droplets. The transmission happens when two subjects are in the close contact with each other. CDC defines close contact by proximity and exposure. The close contact is defined as 6 feet of an infected person for a cumulative total of 15 minutes or more over a 24-hour period [41]. The people who were exposed to an infected person are asked to quarantine at home for 14 days [37]. This led to a need for contact tracing technology that could automatically detect a person's close contacts and notify them when they have been exposed to an infected person and need to quarantine. The National Institute of Standards and Technology (NIST) and Massachusetts Institute of Technology's (MIT) Private Automated Contact Tracing (PACT) collaboration led to initiative for development of a protocol, which defined a data collection procedure [42]. Their data collection protocol defines the procedure, in which participants of the study follow a smartphone application developed for this study to collect data [43]. Smartphones were chosen as a data collection tool since most of the modern smartphones present a vast variety of sensors such as integrated measurement unit (IMU), Wi-Fi and Bluetooth radios, and global positioning system (GPS). Smartphones are also widely available to the public which is helpful in large scale deployment of developed technology. However, some of the sensors available in mobile phones such as GPS present privacy concerns. PACT collaboration focuses on use of IMU and Bluetooth radio which would preserve privacy of the users.

1.4. Challenges with infectious disease tracing using smartphones

Tracing infectious disease spread often means tracing social interactions which presents privacy issues. When developing technology for contact tracing it is important to be mindful of the potential privacy issues presented by this technology. Bluetooth radio signal and IMU signals do not contain information about the user which makes these sensors a good candidate for contact tracing. The challenge with using IMU is that these sensors do not carry any information about other subjects, meaning they are unable to send signals to other smartphone users. On the other hand, Bluetooth radio can transmit and receive signal from other mobile devices in its vicinity. Signal strength of Bluetooth can be used to infer the distance. Unfortunately, measuring strength of Bluetooth signal reliably is challenging. Bluetooth signal is subject to reflections and attenuations on its propagation path that create inaccuracies. By developing machine learning models, we attempt to minimize this uncertainty and accurately detect range between the subjects which would identify if subjects are closer than 6 feet of each other. The duration of the interaction can be tracked with an accurate clock available in smartphones. Time tracking is much more trivial task those in this thesis we focus on accurate proximity detection.

1.5. Goals

In this thesis, we utilize PACT MITRE Range and Angle (Unstructured) (MRAU) dataset [1] and ML to develop regression and classification models for proximity detection. We attempt to accurately detect exact distance between two subjects and classify if two subjects are closer than 6 feet of each other. Models described in this thesis could be used as a backbone for contact tracing smartphone application which would send a notification if a healthy person were in close proximity of an infected person.

1.6. Approach to contact tracing

To develop accurate proximity detection models, we start with reading and splitting the data from the MARU dataset. We then filter accelerometer and gyroscope data and remove outliers in Bluetooth radio data. We proceed with preparing data for the ML classification and regression model. The data is first split into training and test splits and then normalized. Three different methods of splitting the data and accuracy of proximity detection are presented in this thesis. We explore 19 different ML models to identify the one that works best with smartphone sensor data. The developed classification models are used to identify subject's context and this information then used as a feature to aid regression model in range estimation. In this thesis, we present 20 different features derived from accelerometer and gyroscope sensors signals, and 28 features derived from Bluetooth radio signal. Additionally, we present two different methods for sensor fusion. One at a feature level and another at decision level. Finally, we present accuracy of range estimation and ability to detect close contact (< 6 feet) with our model. Our model outperforms current state-of-the-art methods using neural networks and achieved nDCF score of 0.34 with Bluetooth radio and 0.36 with sensor fusion. Results are presented with three different cross-validation techniques using four different evaluation metrics.

1.7. Contributions of this thesis

We careful evaluated a variety of ML methods for both regression and classification tasks and experimented with two different sensor fusion methods. As a result of our experiments, we found the following:

1. Elliptical filtering of the accelerometer and gyroscope signals improves the regression R^2 by 0.32 and 0.06 using accelerometer and gyroscope data respectively.
2. Ensemble ML classification methods (boosted and bagged trees) classified < 6ft or not between subjects with 72%-90% accuracy and 0.43 nDCF score using the accelerometer, 78%-84% accuracy and 0.37 nDCF score using gyroscope sensor.

3. The regression tree classification method classified < 6ft or not between subjects with 76%-92% accuracy and 0.36 nDCF score using Bluetooth.
4. Regression tree methods to estimate the actual distance between users when utilizing Bluetooth data, achieving an R^2 between 0.64 to 0.99.
5. Sensor fusion methods were found to be more robust at estimating distance than each sensor individually and were able to classify whether subjects were closer than 6 feet or not with up to 97% accuracy and 0.36 nDCF score.
6. The most important features were z-axis mean and autocorrelation for the accelerometer, z-axis mean and y-axis mean for the gyroscope sensor, and advertiser time, mean RSSI, and peak of Doppler spectrum for Bluetooth.
7. Recognizing user context and classifying proximity using context-specific ML models improved the performance of range regression but not classification.

1.8. Roadmap

The rest of this thesis is as follows: Chapter 2 presents background information of sensing social interactions in the past. Chapter 3 describes our approach to develop accurate ML models for proximity estimation. Chapter 4 demonstrates the results of our experiments. In Chapter 5, we discuss sources of error and why our model is not perfect. Finally, in Chapter 6 we conclude our findings.

CHAPTER 2 BACKGROUND

Sensing social interactions has been explored by many researchers in the past. The main objective of such research is to track human mobility and the spread of epidemics [2], human behavior in organizational settings and how it shapes individuals and organizations [3], and propagation of information [4]. One approach to track social interactions relies on self-reporting. However, information collected in such a way is subjective, as it relies on the study participant's memory. Another approach is to automatically detect social interactions. In order to classify interaction between two people, it is important to understand the proximity and orientation of the individuals. In the past, one group of researchers relied on custom hardware designed specifically for the purpose of identifying subjects carrying such devices, while other groups opportunistically utilized signals from the sensors available on mobile phones. Olguin, D.O et al [3] utilized a custom RFID tag called Sociometer to identify social interactions. Sociometer implements multimodal sensing which utilizes the microphone, accelerometer, Bluetooth and IR sensor to detect interactions. Unfortunately, in using the microphone this approach raises privacy concerns. Similar technology was employed by Isella, Lorenzo et al [2] which counts social interaction when two subjects are within a range of 1-2m from each other. The RFID tag employed in their study was not able to sense interactions beyond 2 meters. Huang, William et al [14] developed a custom sensor that uses ultrasonic radio to communicate among its networks. Their approach demonstrated highly accurate proximity detection with an error of 5cm. Approaches using custom hardware to measure and detect social interactions are not scalable for large scale deployment. In the past, researchers experimented with using combinations of various sensors built into mobile phones to increase the accuracy of social interaction detection. Data from sensors such as: accelerometer [3, 5, 6, 7], gyroscope [5, 6], magnetometer [8], microphone [3,6], and GPS [7] were combined to estimate proximity of two or multiple subjects.

Most modern mobile phones have built-in Wi-Fi and Bluetooth radios, which have provided the opportunity for prior research to utilize Wi-Fi [8, 9, 10] and Bluetooth [3, 5, 9, 11, 12, 13] signals to detect proximity of two or more subjects. Distance estimation using mobile radios relies on Received Signal Strength Indicator (RSSI) [5, 9,11, 12] and Time Difference of Arrival (TDoA) [14]. Researchers predominantly used the Path Loss Model (PLM) [5] to estimate distance from signal power. However, a new form of the PLM, which takes into account the relative orientation of mobile phones was proposed [13] and showed improved distance estimation when the separation between devices is less than 8 feet. In addition to classic proximity estimation techniques such as PLM, prior research also utilized Machine Learning (ML) based solutions. Palaghias, Niklas et al [11] studied social interactions using mobile phones and analyzed Bluetooth RSSI using the

MultiBoostAB (variation of AdaBoost) ML algorithm. In their work, the authors were not concerned with estimating the exact distance between subjects. Instead, their work focused on determining mutual orientation between participants in the study and classifying people's interaction into public, social, and personal zones. In our work, we also employ Bluetooth RSSI but try to estimate distance in feet between subjects. Additionally, while we analyzed data from a variety of phone models, the research of Palaghias, Niklas et al was limited to only one type of phone (HTC One S). Katevas, Kleomenis et al [5] explored multiple ML models such as: XGBoost, Linear Regression, SVM, and Random Forest for identifying social interactions and not to estimate distance. They utilized the same sensors we used in our work accelerometer, gyroscope, and Bluetooth. They defined social interaction as two subjects located within 0.5m-1.5m of each other, giving a range of 1 meter (3.28 feet). Our proposed model achieves higher accuracy even with leave-one-out cross validation method.

In addition to distance estimation, prior researchers attempted to identify contexts in which social interactions occur. Vaizman et al [6] created an application that offered over 100 context labels. Their study attempts to classify the context the smartphone user is in and proposed a context recognition system that utilizes a combination of multi-modal sensors available on mobile devices to increase accuracy of context recognition.

Convolutional neural networks (CNNs) are a popular deep learning technique commonly used in domains such as computer vision and image recognition. CNNs are typically used with 2D image datasets. However, research in the areas of speech recognition [31, 32] and biomedical signal processing [33, 35] have also applied CNNs to 1D datasets. Tsinalis, Orestis et al proposed CNN architecture for sleep stage classification. The network takes an unfiltered EEG signal and passes it through 20 different filters, performing 1D convolution followed by max pooling. Afterwards, the filtered signals are stacked and 2D convolution is performed. The purpose of stacking signals is to find a relationship between different 1D filtered signals. This approach tries to discover relationships across the 3 axes of inertial sensors. Tsinalis, Orestis et al reported an F1 score of 81% for sleep stage classification using CNNs.

Other works attempting to estimate social distance have been published. Shankar, Sheshank et al. attempt to estimate proximity between two subjects by utilizing Bluetooth Low Energy (BLE) in combination with the other sensors available on mobile devices such as: accelerometer, gyroscope, and magnetometer. Their approach treated sensors data as a time-series that were concatenated data into a vector and was analyzed using 1D CNNs. However, in addition to sensor data they also added encoded information about orientation between transmitting and receiving devices. Such information would normally not be available on mobile devices, raising questions about real world applicability of their approach.

In our conference paper, [36] we explored different ML algorithms, filter types, and features based on the user context. In this thesis we evaluate much broader set of statistical features and some radio signal specific features. We also present two different sensor fusion methods and do comprehensive evaluation of model performance. We investigate how accurately the proximity of two smartphones can be estimated using data from their built-in accelerometer, gyroscope sensors and Bluetooth radio. We analyzed these data using ML algorithms to estimate range. A novel aspect of our work is that we are the first to explore whether context-specific ML models are more accurate than general ones. Specifically, we explored whether first recognizing the user's context such as whether the user is indoors or outdoors, room size, user's pose and location of the transmitting device on the body and providing this context information as an input feature improves ML proximity estimation. We found that adding recognizing and using context information as a feature improved the accuracy of ML regression (distance estimate) but not ML proximity classification (< 6ft or not). Additionally, we present two different sensor fusion techniques to combine information from multiple sensors in order to improve regressor performance.

Range estimation using Bluetooth radio is challenging because the RSSI varies continuously due to multipath fading, the transmission environment, room size, the presence of obstructions and the number of people in the room.

CHAPTER 3 METHODOLOGY FOR PROXIMITY DETECTION

3.1. Dataset overview

In our experiments, we utilized the publicly available PACT MITRE Range and Angle (Unstructured) (MRAU) dataset [1] to develop our ML proximity classification and regression models. Table 1 summarizes the data in the MRAU dataset.

Table 1 Summary of the MITRE range and angle (unstructured) dataset

Gyroscope			Context	
x-axis	y-axis	z-axis	Indoor	Sitting
Accelerometer			Outdoor	Standing
x-axis	y-axis	z-axis	Large Room	Hold to Right Ear
Bluetooth			Medium Room	Front Pants Pocket
RSSI	TSSI	Advertiser Timestamp	Small Room	In Hand
Response			Center Congested	In Purse
			Center Open	Rear Pants Pocket
Range	Angle		Near Wall Congested	Shirt Pocket
			Near Wall Open	

The RSSI signal measurements were taken at increments of 2 feet from 2 to 16 feet between transmitting and receiving devices. Since participants did not follow a defined protocol, not all recorded RSSI at all distances, which presents additional challenge when developing and evaluating models on the MRAU dataset. Figure 2 depicts measurements scenarios.

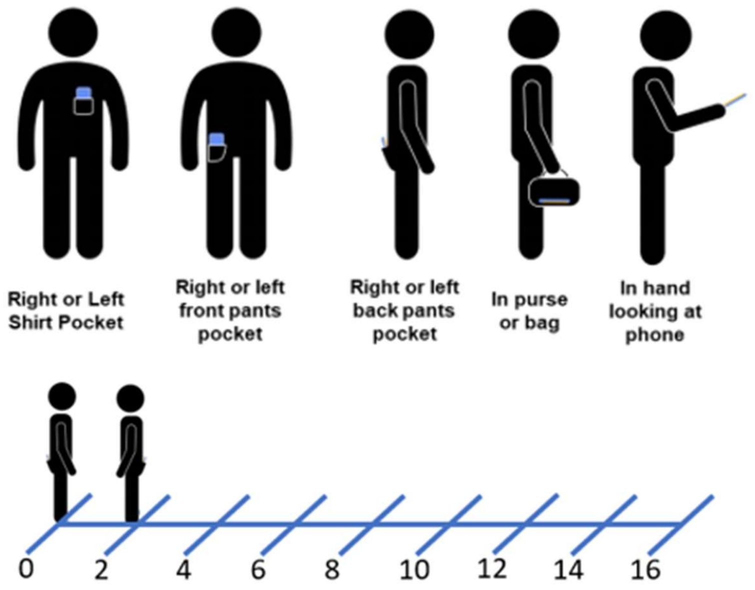
Right or Left Shirt Pocket
Right or left front pants pocket
Right or left back pants pocket
In purse or bag
In hand looking at phone
0
2
4
6
8
10
12
14
16

Figure 2 MARU measurement scenarios

The data in MARU dataset was collected for approximately 60 seconds at each distance and orientation. Accelerometer and gyroscope data were sampled at ~4Hz while Bluetooth data was sampled at 4Hz-10Hz. Various mobile devices were used to capture the data such as: Pixel, Pixel 3, iPhone 6s, iPhone 7, iPhone 8, iPhone 11, iPhone XR, Galaxy S9, and LG G7, which explains why sampling frequency is inconsistent.

3.2. Approach to proximity detection

The main steps in our machine learning pipeline are shown in Figure 3.

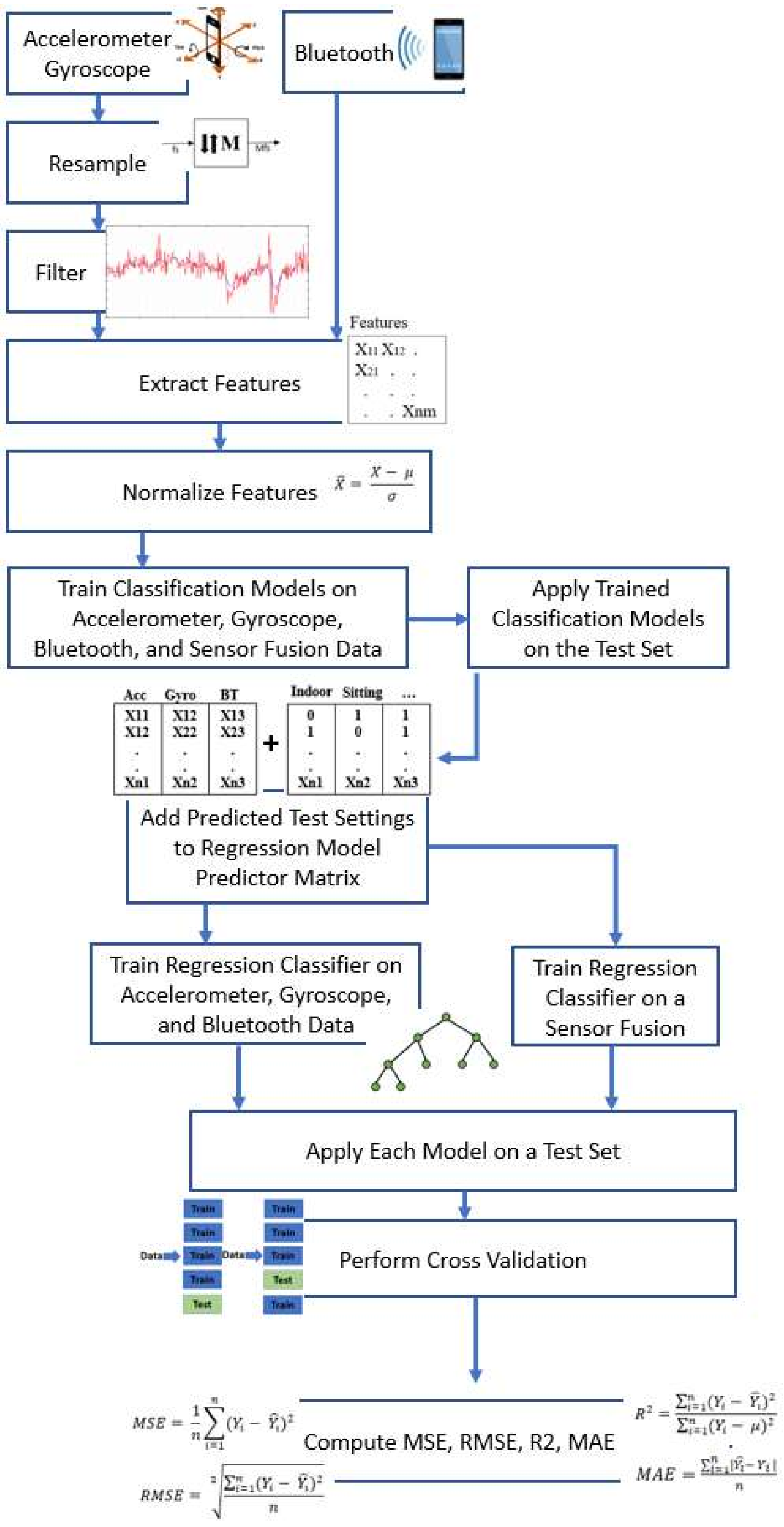
Accelerometer Gyroscope
Bluetooth
Resample
Filter
Features
Extract Features
Normalize Features
Train Classification Models on Accelerometer, Gyroscope, Bluetooth, and Sensor Fusion Data
Apply Trained Classification Models on the Test Set
Acc Gyro BT
Indoor Sitting ...
Add Predicted Test Settings to Regression Model Predictor Matrix
Train Regression Classifier on Accelerometer, Gyroscope, and Bluetooth Data
Train Regression Classifier on a Sensor Fusion
Apply Each Model on a Test Set
Perform Cross Validation
Compute MSE, RMSE, R2, MAE

Figure 3 Overview of our ML proximity regression pipeline using Accelerometer, Gyroscope, Bluetooth, and sensor fusion data.

First, the raw sensor signals were resampled since the sampling rate is not consistent across all subjects. Next, the accelerometer and gyroscope sensors signals were filtered using a low-pass filter. Then, various statistical features found to be predictive in prior work were extracted. These features including the user's context were then classified or regressed to predict the user's proximity/range using ML methods. In addition to training a classifier and regressor on individual sensors we used a combination of all three sensors for range estimation. Inspired by Vaizman et al's work on context recognition, we experimented with two different sensors fusion methods.

Sensor fusion: There are three commonly known sensor fusion methods which combine data from multiple sources at data level, feature level, or decision level. We explored sensor fusion at feature level and decision level. The first sensor fusion method combines data at feature level using a feature matrix extracted from raw sensor data (accelerometer, gyroscope and Bluetooth). The feature data was concatenated horizontally, resulting in n by 58 feature matrices, where n is a number of samples. The second sensor fusion method combines data at decision level using a two-stage approach. First the ML model was trained on data from each sensor individually. The proximity probabilities output by each ML were combined resulting in n by 3 matrices. Finally, the matrix of probabilities was input into the ML regressor for range estimation. Our data fusion method at decision level is similar to the popular Dempster-Shafer theory where hypothesis H is calculated as $m_1 \oplus m_2(H) = \frac{\sum_{X \cap Y = H} m_1(X) m_2(Y)}{1 - \sum_{X \cap Y = \emptyset} m_1(X) m_2(Y)}$ [44]. The difference with our method is that after obtaining probabilities (hypothesis), we let ML model decide how to combine the data instead of assigning a defined formula. Figure 4 illustrates how the data from multiple sensors is arranged for the two sensor fusion methods explored in this thesis.

Signal filtering: We evaluated the utility of 5 filter types for ML range estimation: Butterworth [16], Chebyshev [16], Elliptical [16], Median [17], moving average and moving average with overlapping windows. In order to determine frequencies of interest, the Fast Fourier Transforms (FFTs) of the sensor signals were computed. Most of the signal energy was found to be concentrated in the 0-0.2Hz, 0.3Hz-0.5Hz, and 1.3Hz-1.5Hz bands. The Kaiser window FIR filter defined by $\omega_k(nT) = \begin{cases} \frac{I_0\left(2\beta\sqrt{\frac{n}{N-1} - (\frac{n}{N-1})^2}\right)}{I_0(\beta)} & 0 \le n \le N-1 \\ 0 & elsewhere \end{cases}$ where β controls side lobe level and $I_0(x)$ is modified zero-order Bessel function [26]. A combination of three bands was tested. Another experiment was conducted by using the Discrete Wavelet Transform (DWT) to filter data, but it did not improve the accuracy of distance estimation.

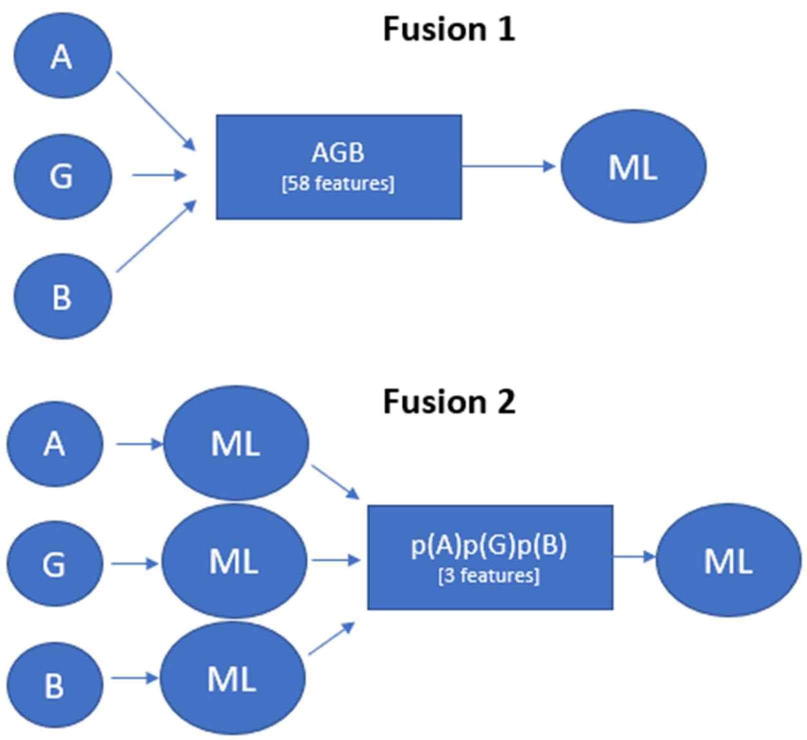

Figure 4 Sensor fusion methods

Feature extraction: Table 2 summarizes the radio propagation and statistical features we computed from accelerometer, gyroscope, and Bluetooth data. Additionally, for the accelerometer and gyroscope sensors, the mean and standard deviation for each axis were computed as well as autocorrelation between the xy, xz, and yz axes. A total of 20 features was extracted for each sensor. Radio propagation-specific features along with statistical features based on RSSI, transmitted signal strength, and advertiser time were extracted for Bluetooth radio. All radio propagation specific features for Bluetooth radio were calculated based on RSSI.

Table 2 Features computed from accelerometer, gyroscope, and Bluetooth data

Sensor	*Feature*	*Formula*	*Ref*
A, G	*Magnitude*	$magnitude = \sqrt{x^2 + y^2 + z^2}$	-
A, G, B	*Mean*	$\mu = \frac{1}{N}\sum_{i=1}^{N} X_i n$	-
A, G, B	*Standard deviation*	$S = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\lvert X_i - \mu\rvert^2}$	-
A, G, B	*Third and fourth momentum*	$m_k = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^k$, k=3, 4	-
A, G, B	*Percentile*	The score at k percentile for k = 25, 50, 75	-
A, G, B	*Value entropy*	$H_x(y) = -K\sum_{i=1}^{N} p_i \log p_i$	15
A, G	*Time entropy*	$H_x(y) = -K\sum_{i=1}^{N}\lvert m_i\rvert \log(\lvert m_i\rvert)$	15
A, G, B	*Autocorrelation*	$r_k = \frac{c_k}{c_0}$	18
A, G	*Autocovariance*	$K_X(t,s) = cov[X(t), X(s)] = E\{[X(t) - \mu_X(t)][X(s) - \mu_X(s)]\}$	19
B	*Delta*	$delta = RSS - TSS$	-
B	*Peak-to-peak change*	$\{y(n)\}\rvert_{max} - \{y(n)\}\rvert_{min}$	29
B	*Doppler spectrum peak*	$P_{D(\lambda)} = \max_{\lambda\epsilon(-f_m, f_m)} D(\lambda)$	28
B	*Doppler spectrum mean*	$\mu_D = \frac{\sum_{-f_m}^{f_m} D(\lambda)}{\lvert 2f_m\rvert}$	28
B	*Doppler spectrum RMS*	$P_{RMS} = \sqrt{\frac{\int \lambda^2 D(\lambda)d\lambda}{\int D(\lambda)d\lambda}}$	29
B	*Energy*	$E = \int D(\lambda)d\lambda$	29

B	*Laplacian best fit*	$D(\lambda) = \dfrac{a}{1 + b\lambda^2}$	29
B	*Gaussian fit*	$D(\lambda) = \sum_{i=0}^{i=n-1} a_i \exp\,(-(\lambda - \ b_i)^2 /c_i^2)$	27
B	*Polynomial fit*	$D(\lambda) = \sum_{i=0}^{i=n-1} a_i \lambda^i$	27
B	*Fade duration*	$\tau(\rho) = \dfrac{e^{\rho^2} - 1}{\sqrt{2\pi}\rho P_{RMS}}$	29
B	*Level crossing*	$N(\rho) = \sqrt{2\pi}\rho P_{RMS} e^{-\rho^2}$	29
B	*Rayleigh parameter*	$f_{ray}(r) = \dfrac{y}{\sigma^2} \exp\left(-\dfrac{y^2}{2\sigma^2}\right), y \geq 0$	29

In addition to statistical features, we employed classification models to estimate the environment in which measurements were taken. There were five types of environment labels: 1) Indoors or outdoors, 2) Room size: large, medium or small room, 3) Transmitting device's location in the room: center congested, center open, near wall congested, and near wall open, 4) Pose of the test subject: sitting or standing, and 5) Phone placement: held to right ear, front pants pocket, in hand, in purse, rear pants pocket, or shirt pocket. All the features were computed over windows of 10 samples of continuously sampled sensors' signals. Before training a regression classifier on the dataset, the data were normalized using one of two methods: 1) z-score $\bar{X} = \frac{X-\mu}{\sigma}$ or 2) min-max normalization $\bar{X} = \frac{X-X_{min}}{X_{max}- X_{min}}$.

Machine learning algorithms: In this thesis, several regression models were examined. A linear regression model assumes that regression function is linear. Linear regression model is defined by $f(X) = \beta_0 + \sum_{j=1}^{p} X_j\beta_j$ [20]. Several methods were examined to find the best fitting the linear regression model. The stepwise linear method automatically adds or removes predictors X_j to the model based on the goodness of fit evaluated by computing a sum of squared errors. It is expected that a model with more predictors will have lower error but could also easily overfit the data. For that reason, we evaluated various combinations of predictors in the ML model. The linear regression method is somewhat similar to stepwise linear method but instead of adding or removing predictors, it tries to discover interactions between predictors. The robust linear regression method is a type of linear regression that attempts to identify outliers in the dataset which would affect the goodness of fit. The measure of the effect of each observation in the dataset can be found by computing H-matrix $H = X(X^TX)^{-1}X^T$ where each diagonal term is the effect of an observation on outcome y and X is the data matrix [22]. Outliers are undesirable data points that do not follow the pattern of the other samples. Removing outliers may affect the distribution of the data, which may be undesirable. Tukey's biweight applies weight to the data to handle outliers and was used to handle outliers in our dataset. Tukey's biweight function is defined as:

$$\begin{array}{ll} \psi(x) = x(1-x^2)^2 & for\ x < 1 \\ \psi(x) = 0 & otherwise \end{array}$$ [17].

We evaluated linear regression [20, 22], regression trees [21], Support Vector Machines (SVM) [23], ensemble methods [20], and Gaussian Process Regression (GPR) [24]. We validated the results of regression models using the 5-fold cross validation technique to avoid overfitting and to robustly determine the optimal ML model. The best performing model was also evaluated using leave-one-out cross-validation with subject level splitting, wherein in each subject's data occurs either in the training or test set but not partially in both.

The regression tree splits data space into disjointed regions A_k and provides a fitted value $E\ (Y \mid X \in A_k)$ within each region [21]. Each split in the tree is made to decrease impurity. Deviance is used as a definition of impurity and defined as $D = \sum_{cases\ j}(y_i - \mu_{|i|})^2$ where $\mu_{|i|}$ is the mean of values in the node where case j belongs to [21]. Three different types of trees were examined in this thesis: fine, medium, and coarse. The difference between the tested tree types is that the minimum of the fine tree leaf size is 4. The medium tree has a minimum leaf size of 12 and the coarse tree has a minimum leaf size of 36. A Fine tree with smaller leaf size tends to overfit the data. Trees with larger leaf sizes could help improve accuracy on the test set. Our experiments showed the highest accuracy achieved with a fine tree.

Support Vector Machines (SVM) regression is similar to linear regression attempts to split feature space by creating linear boundary defined by $f(x) = x^T\beta + \beta_0$. In addition to hyperline that attempts to split the feature space, SVM allows deviation from the hyperline by a margin M, defined as $M = \frac{1}{\|\beta\|}$ [20]. SVM constructs a decision boundary of a type $f(x) = sign(\sum_{support\ vectors} y_i\alpha_i K(x_i, x) - b)$ where $K(x_i, x)$ is a convolution of the inner product between support vector and the vector of the feature space [23]. Different functions can be used for convolution of the inner product $K(x_i, x)$ which can create different types of non-linear decision boundaries. In this thesis, linear, quadratic, cubic, and three types of Gaussian function were examined. The linear decision boundary is defined by $K(x_i, x) = x_i \cdot x$. To construct the polynomial decision boundary the following function for convolution of the inner product is used $K(x_i, x) = [(x_i \cdot x) + 1]^d$ where d is the degree of the polynomial [11]. For quadratic functions d=2 and cubic d=3 are used. The gaussian function uses the following decision rule $f(x) = sign(\sum_{i=1}^{N} \alpha_i K_\gamma(|x - x_i|) - b)$ where $K_\gamma(|x - x_i|) = exp\{-\gamma|x - x_i|^2\}$ [11]. Fine, medium, and coarse Gaussian SVMs are different in how the feature space is scaled. For Fine Gaussian SVM, features are scaled by $\frac{\sqrt{P}}{4}$ where P is the number of features. Medium Gaussian SVM uses $\sqrt{P}$ to scale the features and Coarse Gaussian SVM utilizes $\sqrt{P} * 4$.

Table 3 Evaluation metrics

Mean Squared Error (MSE)	$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2$
Root-Mean-Square Error (RMSE)	$\text{RMSE} = \sqrt[2]{\frac{\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2}{n}}$
R^2	$\text{R}^2 = \frac{\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2}{\sum_{i=1}^{n}(Y_i - \mu)^2}$
Mean Absolute Error (MAE)	$\text{MAE} = \frac{\sum_{i=1}^{n}\lvert\widehat{Y_i} - Y_i\rvert}{n}$
Balanced Accuracy (BA)	$\text{BA} = \frac{TPR + TNR}{2}$
F1	$F1 = \frac{2 * TPR * precision}{TPR * precision}$
True Positive Rate (TPR)	$\text{TPR} = \frac{TP}{TP+FN}$
True Negative Rate (TNR)	$\text{TNR} = \frac{TN}{TN+FP}$
Precision	$precision = \frac{TP}{TP + FP}$
Normalized Decision Cost Function (nDCF)	$nDCF = \frac{w_{miss}P_{miss} + w_{fa}P_{fa}}{\min\,(w_{miss}, w_{fa})}$

Ensemble Regression model utilizes several simple models to build stronger prediction models. Two different ensemble regression methods were explored in this thesis. Bagged Trees, an ensemble method, utilizes simple regression trees to build stronger predictor. The bagging estimate is defined by $\hat{f}_{bag} = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{xb}(x)$ where $\hat{f}(x)$ is the prediction at input *x*, *b* is a bootstrap sample, and $\hat{f}^{xb}(x)$ are independent predictions generated by each regression tree [20]. Bagging trees helps reduce variance and thus improve goodness of fit. Boosted Trees is a technique that combines several individual regression models to produce a more powerful one. A prediction from several individual classifiers $G_m(x)$ combined through weighted majority voting produces a final decision $G(x) = sign(\sum_{m=1}^{M} \alpha_m G_m(x))$ where α_m are weights for each individual classifier $G_m(x)$ [20]. Weights applied to each individual classifier help more accurate classifiers influence the final prediction. α_m computed as $\alpha_m = \log(\frac{1-err_m}{err_m})$; error computed as $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$ where $w_i = \frac{1}{N}$ and N is a number of observations [20].

Gaussian Process Regression (GPR) can be viewed as a linear regression but with added noise. The noise in the GPR model has a Gaussian distribution $\varepsilon \sim N(0, \sigma_n^2)$ with zero mean and variance of σ_n^2, those GPR model has the form $y = f(x) + \varepsilon$ where $f(x) = x^T w$ [24]. The Gaussian process is specified by the mean function $m(x)$ and covariance function or kernel $k(x, x')$, where $m(x) = E[f(x)]$ and $k(x, x') = E[(f(x) - m(x))(f(x') - m(x'))]$ [24]. In this thesis, several kernel functions were examined. We examined squared exponential kernel function defined as $k(r) = \exp(-\frac{r^2}{2l^2})$ where l is characteristic length scale and $r = |x - x'|$, the Matern 5/2 kernel defined as $k(r) = \frac{2^{1-v}}{\Gamma(v)} \left(\frac{\sqrt{2vr}}{l}\right)^v K_v(\frac{\sqrt{2vr}}{l})$ where $v = 5/2$ and K_v is a modified Bessel function, an exponential kernel defined as $k(r) =$

$\exp\left(-(r/l)^{\gamma}\right)$ for $0 < \gamma \leq 2$, and rational quadratic kernel defined as $k(r) = (1 + \frac{r^2}{2\alpha l^2})^{-\alpha}$ [24].

Evaluation metrics: Table 3 summarizes evaluation metrics used to measure performance of regression and classification models.

CHAPTER 4 EVALUATION AND RESULTS

4.1. Data preparation

MARU dataset contains 24 different folders. We treated each folder as a different subject. We first load the data into computer memory and organized data by sensor and subject. Accelerometer and gyroscope data matrices have x, y, and z axes data, information on range, angle, and user context. Bluetooth contains advertiser time stamp, RSSI, transmitter signal strength, range, angle, and information on user context. User context at this stage is *n x 17* matrix consisting of ones and zeros that tell if certain label applicable to a given data point or not. We continue by filtering data using routine in appendix B.1 and creating hand-crafted features using routine in appendix B.2. The functions used to calculate features presented in appendix B.6.

Now we have data ready for classification to identify user context. Routines presented in appendix B.3 first combines 17 individual context labels into five categories then performs classification to predict user context and finally, adds this information to the test and training matrices. The final step in which the data was normalized, ML models trained, and the regression tasks performed to estimate range between a subject in order to classify it two are closer than six feet presented in Appendix B.4. Supporting functions, such as the one used to align sensor data for sensor fusion experiments, calculate performance matrices, and to group context information into five categories are presented in Appendix B.5.

4.2. Signal filtering

We tested various types of filters and cutoff frequencies ranging from 0.1 Hz to 1.8 Hz and found that the Elliptical filter of the 9th order performed best. Table 4 shows a ranking of the performance of different filter types.

The highest regression R^2 for the accelerometer was 0.63 at cut-off frequency of 0.2 Hz and 0.27 for the gyroscope at a cut-off frequency of 0.1 Hz. This was a significant improvement over the best R^2 achieved using unfiltered data: 0.31 for the accelerometer and 0.21 for the gyroscope. The constructed bandpass filter performed worse than a low-pass filter. Thus, we proceeded to utilize the 9th order Elliptical filter low-pass filter in subsequent experiments.

Table 4 Filter type rating

Rating	Accelerometer	Gyroscope	Sensor Fusion
1	Elliptical 9th	Elliptical 10th	Elliptical 9th
2	Chebyshev 5th	Butterworth 10th	Butterworth 6th
3	Butterworth 6th	Chebyshev 3rd	Chebyshev 5th
4	Median 7th	MA with OL	MA (window 4)
5	MA (window 8)	MA (window 3)	MA with OL
6	MA with OL	Median 6th	Median 10th

4.3. Normalization

Min-max and z-score normalization techniques showed similar performance. The R^2 score for accelerometer was 0.5365, gyroscope – 0.3808, and Bluetooth - 0.9983 using min-max normalization, while with z-score normalization accelerometer R^2 – 0.6767, gyroscope – 0.2935, and Bluetooth – 0.9980. Consequently, we proceeded with z-score normalization.

4.4. Classification performance on various labels

Table 5 summarizes the results of classifying various labels in the MRAU dataset. Overall, Bluetooth radio had very high accuracy for recognizing all the context variables with BA and F1 values of 0.99 for all. Accelerometer data yielded good accuracy for recognizing when the subject was sitting or standing and also, in detecting the size of the room. We discovered that including additional features improved the ML model's performance of the model when validated using 5-fold cross-validation. Figures 5 through 7 show the importance of each feature for various range estimation tasks. In addition to hand-crafted features, a much broader set of features was evaluated, and results are presented in Appendix A. Feature importance was calculated as the difference in the node risk between parent and children's nodes $\frac{R_1 - R_2 - R_3}{N_{branch}}$ where risk is defined as a node error.

Table 5 F1 and BA classifier performance

Classification Model Accuracy	Accelerometer		Gyroscope		Bluetooth	
	F1	*BA*	*F1*	*BA*	*F1*	*BA*
Indoor	0.8512	0.9014	0.7595	0.8371	0.9981	0.9991
Outdoor	0.7812	0.8626	0.4939	0.7098	0.9999	1
Large Room	0.7946	0.8708	0.7152	0.8465	0.9971	0.9988
Medium Room	0.9035	0.9041	0.7793	0.7843	0.9992	0.9992
Small Room	0.8466	0.9005	0.5839	0.74	0.9999	1
Center Congested	0.5835	0.7511	0.1614	0.59	0.9936	0.9999
Center Open	0.8511	0.8921	0.6514	0.7466	0.9995	0.9995
Near Wall Congested	0.8657	0.8993	0.7512	0.8113	0.9999	1
Near Wall Open	0.9035	0.9309	0.7357	0.8026	0.9999	0.9999
Sitting	0.8815	0.8987	0.7401	0.782	0.9979	0.998
Standing	0.9151	0.8987	0.8195	0.782	0.9979	0.998
Hold to Right Ear	0.8546	0.9033	0.7712	0.8362	0.9992	0.9996
Front Pants Pocket	0.635	0.7677	0.3487	0.6125	0.9943	0.9975
In Hand	0.7518	0.8048	0.5636	0.6611	0.9929	0.9929
In Purse	0.6756	0.7956	0.2771	0.5835	0.9796	0.992
Rear Pants Pocket	0.6292	0.7419	0.456	0.6573	0.985	0.9895
Shirt Pocket	0.5803	0.7863	0.1325	0.5656	0.989	0.9987
Average:	0.7826	0.8535	0.573	0.7264	0.9955	0.9978

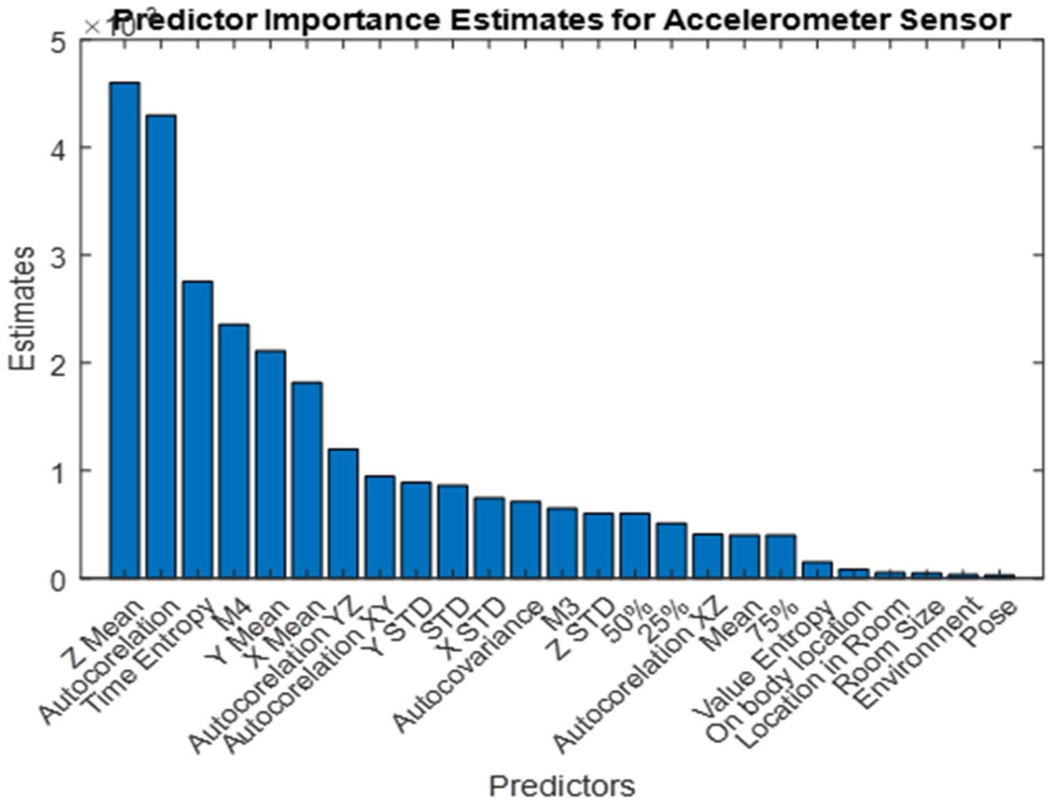

Figure 5 Predictor importance for accelerometer sensor data

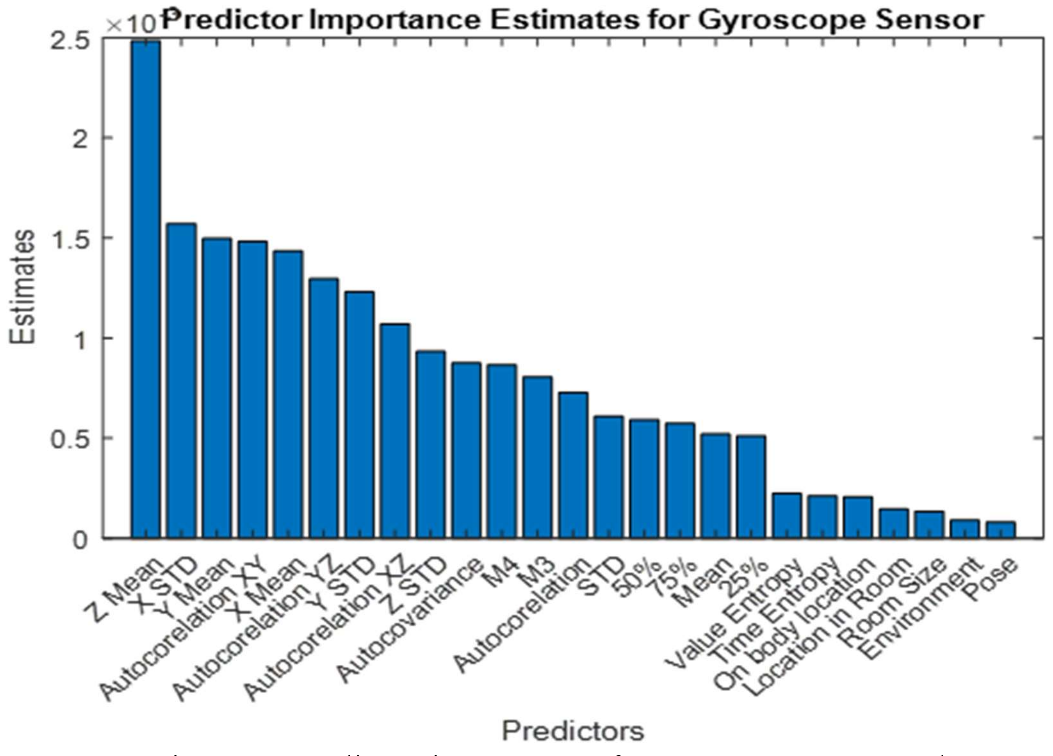

Figure 6 Predictor importance for gyroscope sensor data

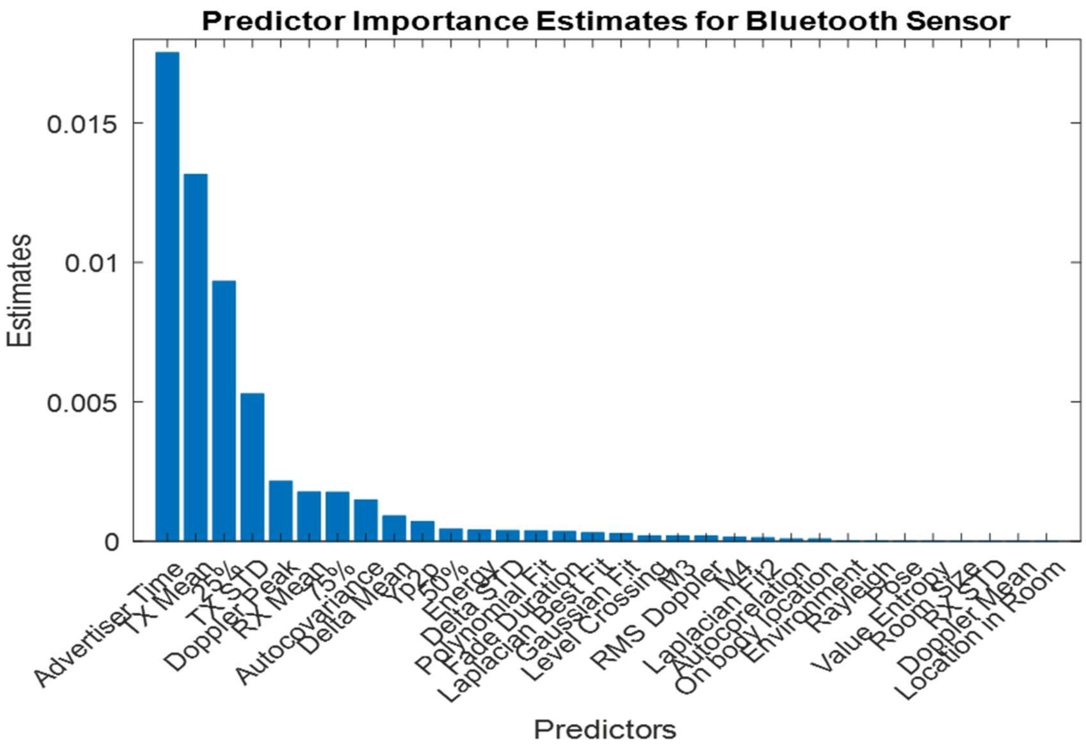

Figure 7 Predictor importance for Bluetooth sensor data.

4.5. Best performing ML classification algorithm

Table 6 summarizes and compares the performance of various machine learning classification algorithms. Regression Trees performed best with Bluetooth sensor data, and Bagged Trees performed best on the accelerometer and gyroscope data. The highest ML regression fit was observed with the Bluetooth sensor.

Table 6 Results of an optimal classifier search

Model	Method	RMSE score		
		A	*G*	*B*
Linear	Linear	4.4223	4.6607	4.7696
	Interactions Linear	4.6399	7.938	5.82E+08
	Robust Linear	4.4308	4.6633	12.719
	Stepwise Linear	3.9749	4.6461	7.0528
Tree	Fine Tree	3.5293	5.1107	0.2289
	Medium Tree	3.3486	4.7299	0.3087
	Coarse Tree	3.4537	4.4817	0.4765
SVM	Linear SVM	4.528	4.7148	4.6823
	Quadratic SVM	4.2603	7.1004	19.766
	Cubic SVM	30.453	105.29	26940
	Fine Gaussian SVM	3.234	4.2502	3.7406
	Medium Gaussian SVM	3.3354	4.2912	4.2992
	Coarse Gaussian SVM	4.2538	4.5428	4.5327
Ensemble	Boosted Trees	3.5044	4.2923	2.1337
	Bagged Trees	2.837	4.0989	0.5902
GPR	Squared Exponential	3.0078	4.1947	3.6705
	Matern 5/2	2.9709	4.1685	3.5152
	Exponential	2.9223	4.1153	2.0701
	Rational Quadratic	2.9249	4.1399	2.1051

4.6. Cross-validation

In addition to the 5-fold cross validation technique, we also validated the performance of our model using leave-one-out cross validation, and subject level splitting. For leave-one-out cross-validation, we split the data into training and test sets by subject, where one subject's data was put in the test set and all others in the training set. For this test we only used subjects that had data from all three sensors which resulted in a total of 17 subjects. Since the performance of the regression model can vary significantly from subject to subject, we performed leave-one-out cross-validation 17 times, each time testing on a new subject while the remaining 16 were used to train the model. This validation approach most closely mimics real-world environments. Figure 8 summarizes the performance of the regression model and depicts the distribution of regressor error. It should be noted that the regression model was found to be capable of producing repeatable results when trained on the accelerometer, gyroscope, Bluetooth, or the first method of sensor fusion. The regression model trained on gyroscope sensor data showed the lowest average RMSE error of 4.57 while Bluetooth had an error of 5.65. However, some subjects showed higher range estimation accuracy using Bluetooth sensor data, while others had better results with gyroscope or accelerometer data. Since it is difficult to predict which sensor will work best for a random subject, it is best to combine data from all three sensors as proposed in the first sensor fusion method. Even though the average

RMSE for the first sensor fusion method is 5.23, it never performed worse than the worst performing sensor out of three for a given subject. The second sensor fusion method did not show good results, had wide distribution error and thus should not be considered for real-word deployment.

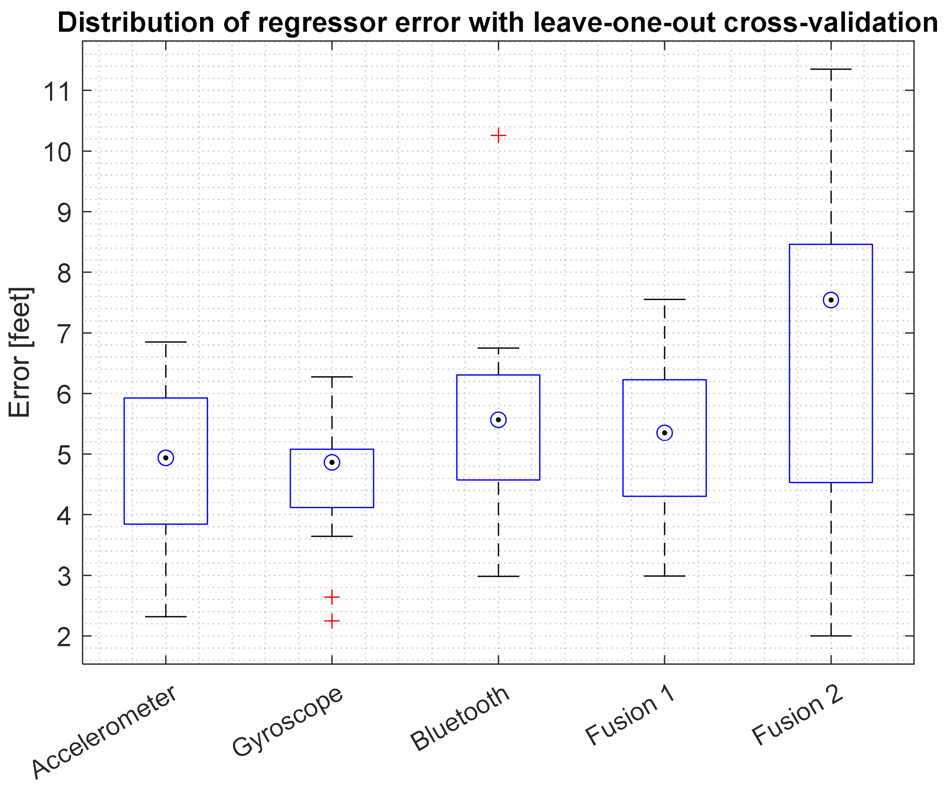

Figure 8 Distribution of regressor error with leave-one-out cross-validation.

For subject level cross-validation, we used a 70/30 train/test split (12 subjects in training set, 5 subjects in test set). This method is a good alternative to leave-one-out cross-validation as it is very rigorous and mimics a real-world scenario but also allows us to have a larger test set. Tables 7 through 9 show the results of cross-validation.

Table 7 Regression model results with 5-fold cross-validation

	Accelerometer		Gyroscope		Bluetooth	
	Handcrafted Features	*All Features*	*Handcrafted Features*	*All Features*	*Handcrafted Features*	*All Features*
MSE	7.1917	7.4384	15.9857	16.2524	0.0409	0.0485
RMSE	2.6817	2.7273	3.9982	4.0314	0.2023	0.2202
R2	0.6874	0.6767	0.3051	0.2935	0.9983	0.998
MAE	1.8938	1.9426	3.2899	3.3445	0.0112	0.0122

Table 8 Regression model results with leave-one-out cross-validation

	Accelerometer		Gyroscope		Bluetooth	
	Handcrafted Features	*All Features*	*Handcrafted Features*	*All Features*	*Handcrafted Features*	*All Features*
MSE	43.8925	43.2142	39.2562	39.0704	33.2625	12.9921
RMSE	6.6251	6.5738	6.2655	6.2506	5.7674	3.6045
R2	-0.2007	-0.1822	-0.0739	-0.0688	0.0901	0.6446
MAE	5.9464	5.9096	5.7377	5.7527	4.5635	2.2201

Table 9 Regression model results with subject level splitting cross-validation

	Accelerometer		Gyroscope		Bluetooth	
	Handcrafted Features	*All Features*	*Handcrafted Features*	*All Features*	*Handcrafted Features*	*All Features*
MSE	11.7313	11.1554	24.4948	23.3507	35.929	41.4525
RMSE	3.4251	3.34	4.9492	4.8323	5.9941	6.4384
R2	0.5098	0.5339	-0.0235	0.0243	-0.5012	-0.732
MAE	2.8614	2.7524	4.3665	4.2637	4.5213	4.8695

CHAPTER 5 ANALYSIS OF PROXIMITY ESTIMATION ERROR

5.1. Analysis of error

We found that ensemble methods estimated range between two subjects within 2.8 feet RMSE when using accelerometer data, and 4 feet RMSE with gyroscope data, while the regression trees yielded 0.2 feet RMSE score using Bluetooth data. Additionally, low pass filtering sensor data improved the performance of ML algorithms for range estimation. We observed an R^2 improvement of 0.32 and 0.06 with accelerometer and gyroscope data respectively. Unfortunately, the performance of the regressor dropped significantly when validated using subject level splitting. The R^2 of the best performing ML regression model using Bluetooth reduced by 0.35 R^2, and by 0.49 and 0.22 when using accelerometer and gyroscope data respectively. We observed that adding context information as an input feature to the machine learning model improved regression model performance but not on classification models validated either using 5-fold cross-validation or with leave-one-out cross-validation, but context features helped during validation.

We believe that the reduction in regression model performance was due to inadequate training data. Figure 9 shows how much data was available for each subject. As we can see, some subjects had very little data and very few available distance settings which results in a model being trained at a given distance not on all

17 subjects but on much smaller dataset as well as a much less diverse training set. This also affects validation since only a few samples end up in a test set when validating with leave-one-out cross-validation. Figures 10 to 12 show the predicted versus actual when validated using leave-one-out cross validation.

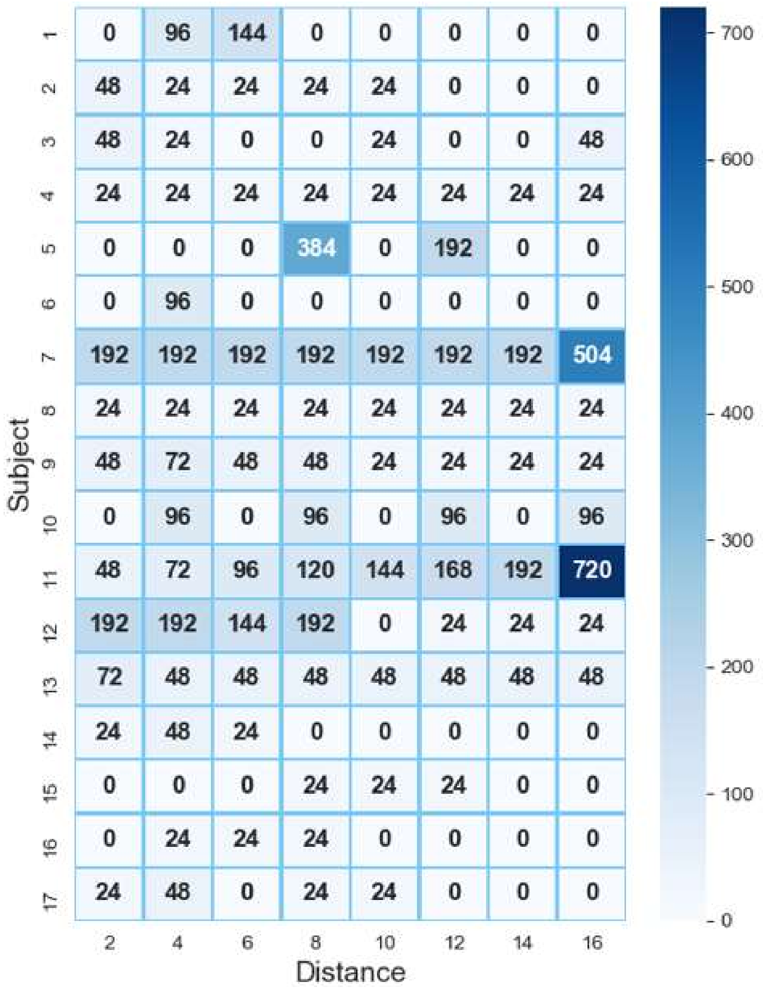

Figure 9 Amount of data in the test set.

Accelerometer and gyroscope data have a high variance, which can be improved by training the regression model on more data. In contrast, the Bluetooth regressor had both high variance and bias. Thus, adding more diverse Bluetooth data could improve regression performance.

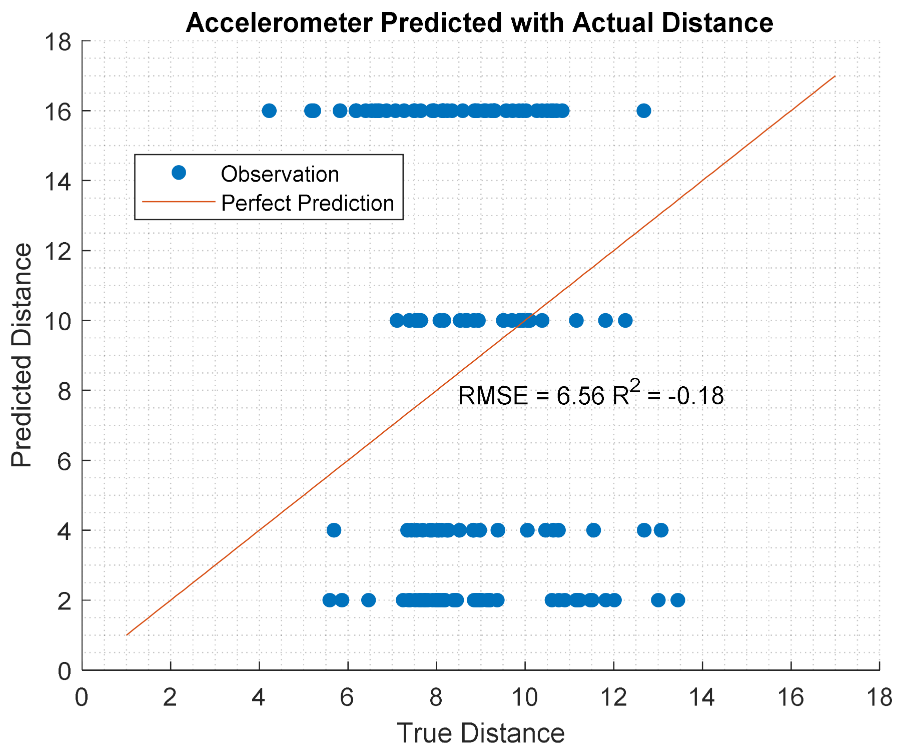

Figure 10 Predicted versus actual distance for regression model trained on accelerometer data and validated with leave-one-out cross-validation

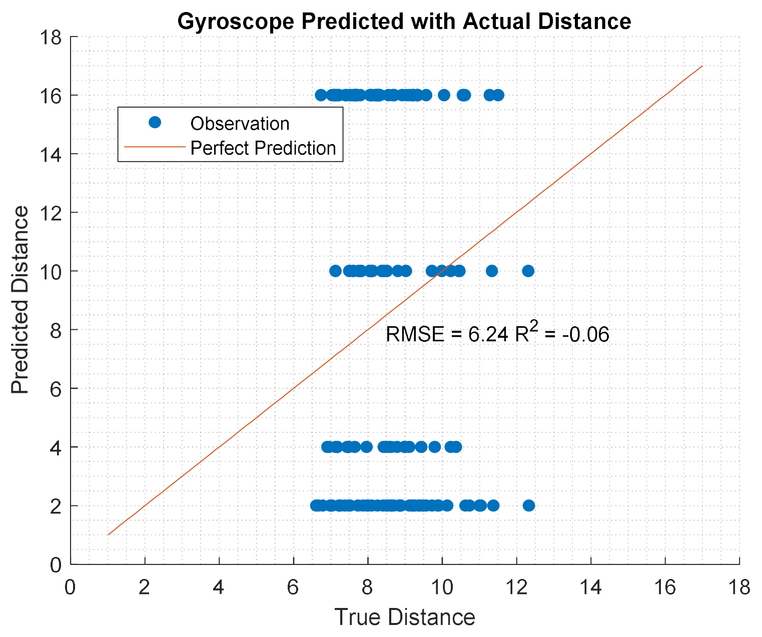

Figure 11 Predicted versus actual distance for regression model trained on gyroscope data and validated with leave-one-out cross-validation

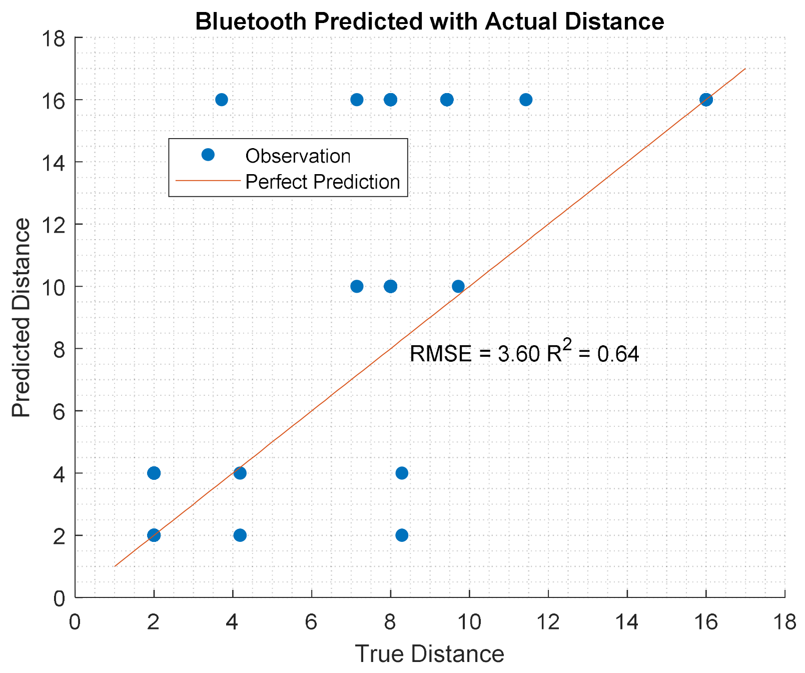

Figure 12 Predicted versus actual distance for regression model trained on Bluetooth radio data and validated with leave-one-out cross-validation

5.2. Synthetic minority over-sampling (SMOTE)

To train our model on more samples, we generated more data using the Synthetic Minority Over-sampling Technique (SMOTE). In comparison to traditional over-sampling or under-sampling techniques, SMOTE operates in the “feature space: rather than “data space” [30]. The additional samples generated along the line that joins all k nearest neighbors where k is determined by the oversampling rate. New samples generated by the following formula $x' = x + rand(0,1)|x - x_k|$, where x is the feature vector under consideration and x_k its nearest neighbor [30]. Table 10 depicts results of comparison between original data set and the one with synthetic data. It should be noted that the regression model results depicted in Table 10 were obtained with leave-one-out cross-validation in which only one randomly selected subject was chosen for the test set. Unfortunately, training our regression model on synthetic data did not improve the accuracy of range estimation.

5.3. Best performing features

In some cases, adding additional features to ML model improved the accuracy of the model. In our experiments, we used 20 hand-crafted features for accelerometer and gyroscope, and 28 hand-crafted features for Bluetooth. However, we found that not all features were useful for range estimation. Based on predictor importance we selected only the 15 best-performing features and repeated our tests using leave-one-out cross validation, where one subject would be kept in the test set and the rest put in the training set. We tested all 17 subjects, and the results are shown in Figure 13 and Table 11.

Table 10 Original with synthetic dataset regression model performance when validated with leave-one-out cross-validation

	Accelerometer		Gyroscope		Bluetooth	
	Ref.	SMOTE	Ref.	SMOTE	Ref.	SMOTE
MSE	35.8281	43.5135	32.6668	35.7631	30.498	89.5644
RMSE	5.9857	6.5965	5.7155	5.9802	5.5225	9.4638
R2	-1.0572	-1.0271	-0.8756	-0.6761	-0.7511	-3.178
MAE	5.294	5.6685	5.0777	5.1284	4.8512	8.1768

	Fusion 1		Fusion 2	
	Ref.	SMOTE	Ref.	SMOTE
MSE	46.2692	47.6334	128.8	101.658
RMSE	6.8021	6.9017	11.349	10.0826
R2	-1.6567	-1.2324	-6.3954	-3.7642
MAE	5.9693	5.9333	10.5538	8.9624

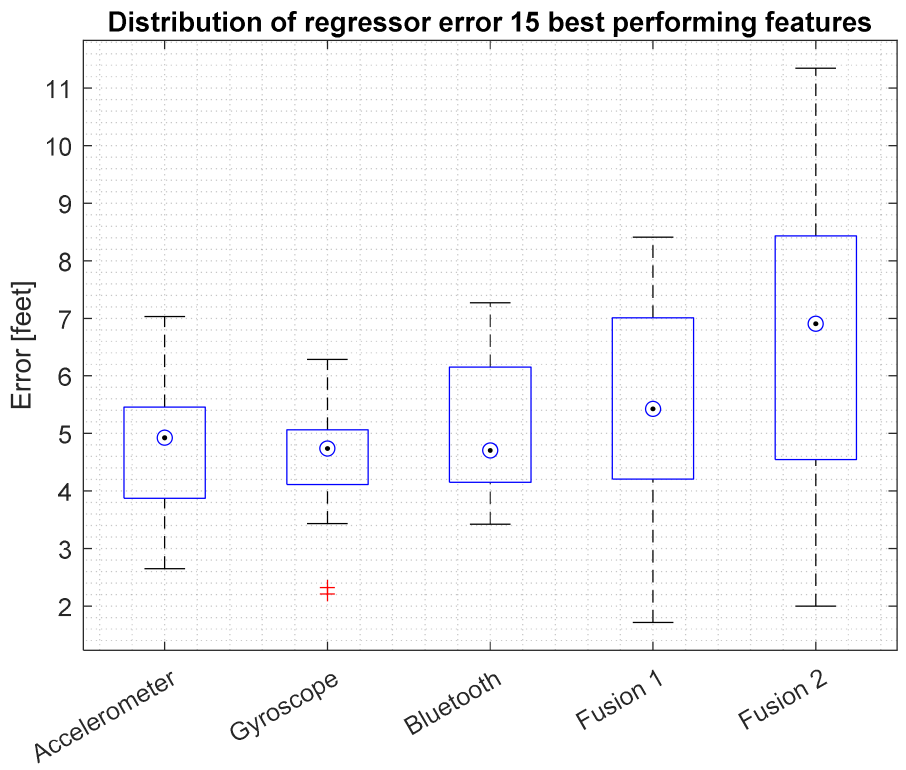

Figure 13 Distribution of regressor error with leave-one-out cross-validation using only 15 best performing features.

Table 11 Regression model results with leave-one-out cross-validation when trained on 15 best performing features

	Accelerometer	Gyroscope	Bluetooth	Fusion 1	Fusion 2
MSE	24.5158	21.6165	27.5294	32.8085	48.5999
RMSE	4.8283	4.5220	5.1097	5.4426	6.4561
R2	-2.4943	-2.1205	-4.0736	-4.1276	-5.2881
MAE	4.1838	3.9787	4.2420	4.6338	5.5489

Using only the best performing features improved mean RMSE by 0.0035 for a model trained on accelerometer sensor data, by 0.0493 on gyroscope data, and by 0.5436 on Bluetooth radio data. However, it did affect the performance of the first sensor fusion method by 0.2098. The regression model based on Bluetooth sensor data had the most features (28 features). Thus, removing the worst-performing features yielded the highest improvement in the accuracy of range estimation. We believe that using the best performing features versus more futures yields higher accuracy regression and classification models.

5.4. Social distance classification

Overall, the approach presented in this thesis can detect with high accuracy whether two subjects are within 6 feet of each other. Table 12 shows the classification results for all sensors with leave-one-out cross-validation technique when using all features and when using only the best performing.

We found that using only the best performing features significantly improves classification model trained on Bluetooth data, improving the F1 score by 0.17. Classification results are even better when validated with 5-fold cross-validation. However, 5-fold cross-validation are not as rigorous as leave-one-out cross-validation, which is the best approximation to the real-world deployment scenario.

Table 13 presents classification results for all sensors and sensor fusion methods when validated with 5-fold cross-validation.

Table 12 Accuracy of estimation if subjects are closer than 6 feet with leave-one-out cross-validation

	Accelerometer		Gyroscope	
	All features	15 Best feat.	All features	15 Best feat.
F1	0.6359	0.7274	0.7345	0.7813
BA	0.4348	0.4599	0.4698	0.4978
	Bluetooth			
	All features	15 Best feat.		
F1	0.5885	0.7589		
BA	0.5318	0.5984		

Table 13 Accuracy of estimation if subjects are closer than 6 feet with 5-fold cross-validation

	Accelerometer	Gyroscope	Bluetooth	Fusion 1	Fusion 2
F1	0.9036	0.8358	0.9182	0.9687	0.9511
BA	0.7925	0.5575	0.9238	0.9625	0.9497

Table 14 Accuracy of estimation if subjects are closer than 6 feet with leave-one-out cross-validation

	Accelerometer	Gyroscope	Bluetooth	Fusion 1	Fusion 2
F1	0.7274	0.7813	0.7589	0.7651	0.0000
BA	0.4599	0.4978	0.5984	0.5918	0.4375

We can see that sensor fusion classified proximity of 6 feet or less better than any other individual sensor. To verify whether that remains true when validating with leave-one-out cross validation, we tested every subject in our dataset and the average results of leave-one-out cross-validation are captured in Table 14. First sensor fusion method performed slightly worse than the model trained on gyroscope data only if comparing by F1 metric. However, sensor fusion performed better than the gyroscope when compared using the Balanced Accuracy (BA) metric. Sensor fusion showed very promising results at estimating proximity and classifying if subjects are closer than 6 feet and we believe it is the optimal way to use sensors available in mobile phones for real-world deployment scenarios.

In addition to balanced accuracy and F1-score, we computed nDCF which is the required performance metric for National Institute of Standards and Technology (NIST) TC4TL challenge [45]. nDCF measures misclassified predictions – false negatives (FN) and false positives (FP). The sum of the misclassification normalized per number of samples so that sets of different sample size can be compared. nDCF results for classifying social distance are captured in Figure 14. We found that our model, trained on each individual sensor and on the sensor fusion at the future level, outperformed state of the art methods using neural networks developed by Shankar, Sheshank et al. and described in their paper “Proximity Sensing: Modeling and Understanding Noisy RSSI-BLE Signals and Other Mobile Sensor Data for Digital Contact Tracing.” [34].

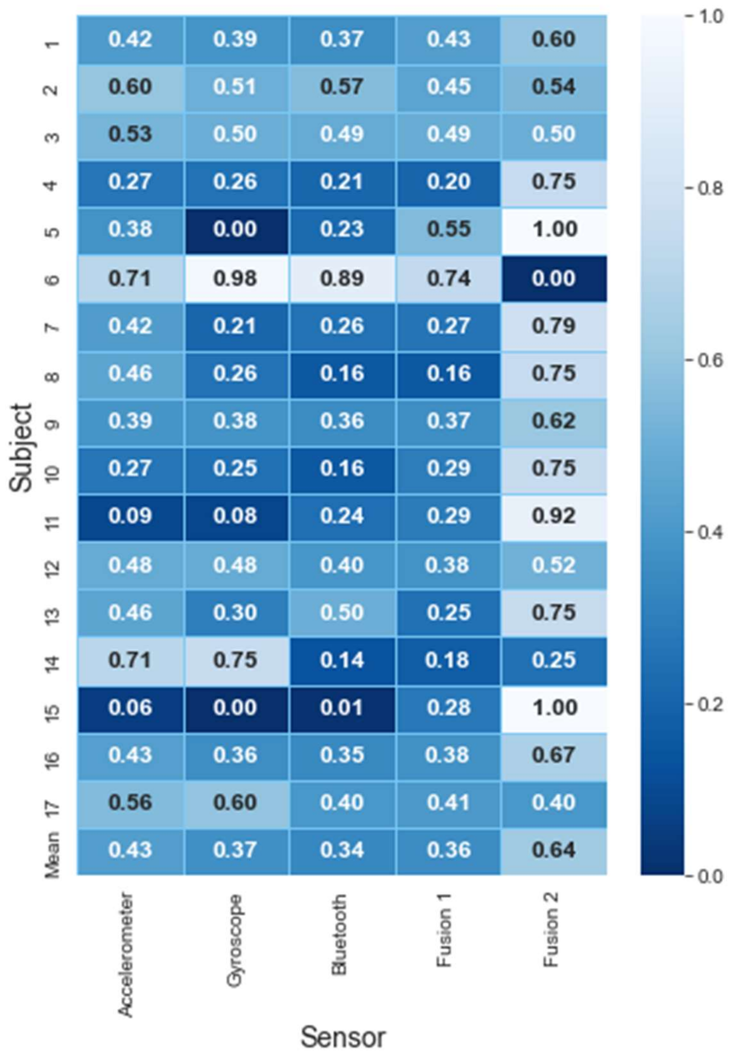

Figure 14 Accuracy of ≤6 feet estimation – nDCF metric.

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

In this thesis, we have presented research that demonstrates that accurate range estimation with accelerometer, gyroscope sensors and Bluetooth radio are possible with high accuracy in certain scenarios. We found that ensemble ML models worked best with accelerometer and gyroscope sensors data, while regression trees performed best with Bluetooth radio data. We found that the Elliptical low-pass filter of 9th order with cut-off frequency of 0.2 Hz for accelerometer and 0.1 Hz for gyroscope performed best. Z-axis mean autocorrelation were the most important features in ML model developed for accelerometer sensor, z-axis mean y-axis mean worked best for gyroscope sensor, and advertiser time and mean RSSI worked best with Bluetooth radio. In addition to handcrafted features, this thesis shows that adding context to predictor matrix could improve regression model performance. Our classification model was able to detect context in which measurements took place with an average BA of 0.85 using accelerometer sensor, 0.73 using gyroscope sensor, and 0.99 with Bluetooth radio. We explored two methods of combining sensors data. Finally, we presented results of an ML classification model trained on sensors fusion data that showed 77%-97% accuracy estimating whether two subjects are closer than 6 feet. We observed that different validation methods could have significant impact on performance of our model. We performed thorough evaluation of our model. The presented model was evaluated with three different cross-validation methods. However, we only split data into a training and test set due to lack of samples and subjects in available dataset. In the future, if a larger dataset becomes available, it would be worth repeating our experiments with the data split to training, validation, and test sets.

REFERENCE

[1] "Pact datasets and evaluation," PACT, Boston, MA, 2020. [Online]. Available: https://mitll.github.io/PACT/datasets.html# datasets-submit

[2] Isella, Lorenzo et al. "What's in a Crowd? Analysis of Face-to-Face Behavioral Networks." Journal of theoretical biology 271.1 (2011): 166–180. Web.

[3] Olguin, D.O et al. "Sensible Organizations: Technology and Methodology for Automatically Measuring Organizational Behavior." IEEE transactions on systems, man and cybernetics. Part B, Cybernetics 39.1 (2009): 43–55. Web.

[4] Pentland, Alex (Sandy). "Automatic Mapping and Modeling of Human Networks." *Physica A* 378.1 (2007): 59–67. Web.

[5] Katevas, Kleomenis et al. "Finding Dory in the Crowd: Detecting Social Interactions Using Multi-Modal Mobile Sensing." (2018): n. pag. Print.

[6] Vaizman, Yonatan, Katherine Ellis, and Gert Lanckriet. "Recognizing Detailed Human Context in the Wild from Smartphones and Smartwatches." IEEE pervasive computing 16.4 (2017): 62–74. Web.

[7] Ouchi, Kazushige, and Miwako Doi. "Indoor-Outdoor Activity Recognition by a Smartphone." ACM Ubicomp 2012. 600-1.

[8] Matic, A et al. "Multi-Modal Mobile Sensing of Social Interactions.", IEEE Pervasive Health Workshop 2012. 105-14.

[9] Banerjee, Nilanjan et al. "Virtual Compass: Relative Positioning to Sense Mobile Social Interactions." Pervasive Computing. Berlin, Heidelberg: Springer Berlin Heidelberg. 1–21. Web.

[10] Sapiezynski, Piotr et al. “Inferring Person-to-Person Proximity Using WiFi Signals.” ACM IMWUT 1.2 (2017): 1–20. Web.

[11] Palaghias, Niklas et al. “Accurate Detection of Real-World Social Interactions with Smartphones.” 2015 IEEE ICC 2015. 579–585.

[12] Katevas, Kleomenis et al. “Detecting Group Formations Using iBeacon Technology.” Proc. ACM Ubicomp 2016. 742–752.

[13] Ghose, Avik, Chirabrata Bhaumik, and Tapas Chakravarty. “BlueEye: A System for Proximity Detection Using Bluetooth on Mobile Phones.” Adj. Proc. ACM Ubicomp 2013. 1135–1142.

[14] Huang, William et al. “Opo: a Wearable Sensor for Capturing High-Fidelity Face-to-Face Interactions.” In Proc ACM Conference on Embedded Network Sensor Systems.2014. 61–75.

[15] Shannon, C. “A Mathematical Theory of Communication.” Mobile computing and communications review 5.1 (2001): 3–55.

[16] Schlichthärle, Dietrich. Digital Filters: Basics and Design. Second edition. Heidelberg: Springer, 2011.

[17] Tukey, John Wilder. Exploratory Data Analysis. Reading, Mass: Addison-Wesley Pub. Co., 1977.

[18] George E.P. Box. Time Series Analysis: Forecasting and Control. 5th ed. New York: Wiley, 2016.

[19] Hsu, Hwei P. Schaum's Outline of Theory and Problems of Probability, Random Variables, and Random Processes. Place of publication not identified: McGraw Hill, 1997.

[20] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. Elements of Statistical Learning: Data Mining, Inference, and Prediction. New York: Springer, 2009.

[21] Breiman, Leo. Classification and Regression Trees. Belmont, Calif: Wadsworth International Group, 1984.

[22] Huber, P. J. Robust Statistics. New York: Wiley, 1981.

[23] Vapnik, Vladimir N. The Nature of Statistical Learning Theory. New York, NY: Springer New York, 1995.

[24] Rasmussen, Carl Edward, and Christopher K. I Williams. Gaussian Processes for Machine Learning. Cambridge: MIT Press, 2005. Print.

[25] Greenberg, C. and, D. (2020), NIST Pilot Too Close for Too Long (TC4TL) Challenge Evaluation Plan, NIST TC4TL Challenge, [online], https://tsapps.nist.gov/publication/get_pdf. cfm?pub_id=930486, https://www .nist.gov/itl/iad/mig/nist-tc4tl-challenge (Accessed May 3, 2021)

[26] Chen, Wai-Kai. Passive, Active, and Digital Filters. Boca Raton: Taylor & Francis, 2006. Print.

[27] Ruijun Fu et al. "Doppler Spread Analysis of Human Motions for Body Area Network Applications." 2011 IEEE 22nd International Symposium on Personal, Indoor and Mobile Radio Communications. IEEE, 2011. 2209–2213. Web.

[28] Dong, Zehua et al. “Indoor Motion Detection Using Wi-Fi Channel State Information in Flat Floor Environments Versus in Staircase Environments.” Sensors (Basel, Switzerland) 18.7 (2018): 2177–. Web.

[29] Su, Zhuoran, Kaveh Pahlavan, and Emmanuel Agu. “Performance Evaluation of COVID-19 Proximity Detection Using Bluetooth LE Signal.” IEEE access 9 (2021): 38891–38906. Web.

[30] Chawla, N. V et al. “SMOTE: Synthetic Minority Over-Sampling Technique.” The Journal of artificial intelligence research 16 (2002): 321–357. Web.

[31] Palaz, Dimitri, Mathew Magimai-Doss, and Ronan Collobert. “Convolutional Neural Networks-Based Continuous Speech Recognition Using Raw Speech Signal.” 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015. 4295–4299. Web.

[32] Swietojanski, Pawel, Arnab Ghoshal, and Steve Renals. “Convolutional Neural Networks for Distant Speech Recognition.” IEEE signal processing letters 21.9 (2014): 1120–1124. Web.

[33] Kiranyaz, Serkan et al. “Convolutional Neural Networks for Patient-Specific ECG Classification.” 2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (2015): 2608–2611. Web.

[34] Shankar, Sheshank et al. “Proximity Sensing: Modeling and Understanding Noisy RSSI-BLE Signals and Other Mobile Sensor Data for Digital Contact Tracing.” (2020): n. pag. Print.

[35] Tsinalis, Orestis et al. "Automatic Sleep Stage Scoring with Single-Channel EEG Using Convolutional Neural Networks." (2016): n. pag. Print.

[36] Semenov, Oleksandr, Emmanuel Agu, and Kaveh Pahlavan. "Machine Learning Estimation of COVID-19 Social Distance using Smartphone Sensor Data"." 2021 43rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (2021). Web. (Accepted, to appear)

[37] "Coronavirus, Covid-19" *Cleveland Clinic*, 12 Nov. 2020, my.clevelandclinic.org/health/diseases/21214-coronavirus-covid-19.

[38] "Interim Clinical Guidance for Management of Patients with Confirmed Coronavirus Disease (COVID-19)." *CDC*, Center for Disease Control and Prevention, 12 Feb. 2021, www.cdc.gov/coronavirus/2019-ncov/hcp/clinical-guidance-management-patients.html. Accessed 15 Aug. 2021

[39] "How to Protect Yourself & Others." *CDC*, Center for Disease Control and Prevention, 13 Aug. 2021, https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/prevention.html. Accessed 15 Aug. 2021

[40] "Coronavirus disease 2019 (COVID-19) Factsheet." *CDC*, Center for Disease Control and Prevention, 7 Jan. 2021, https://www.cdc.gov/coronavirus/2019-ncov/downloads/stop-the-spread_poster.pdf. Accessed 15 Aug. 2021

[41] "Appendices." *CDC*, Center for Disease Control and Prevention, 5 Aug. 2021, https://www.cdc.gov/coronavirus/2019-ncov/php/contact-tracing/contact-tracing-plan/appendix.html. Accessed 15 Aug. 2021

[42] “PACT: Private Automated Contact Tracing” PACT, Massachusetts Institute of Technology, 19 May 2020, https://pact.mit.edu/project-description. Accessed 16 August 2021

[43] “Structured Contact Tracing Protocol, V. 2.0 (1.5)” *MITLL*, Massachusetts Institute of Technology Lincoln Laboratory, 2020, https://mitll.github.io/PACT/files/Structured%20Contact%20Tracing%20Protocol,%20V.%202.0%20(1.5).pdf. Accessed 15 Aug. 2021

[44] Castanedo, Federico. “A Review of Data Fusion Techniques.” The Scientific World 2013 (2013): 704504–19. Web.

[45] “NIST Pilot Too Close for Too Long (TC4TL) Challenge Evaluation Plan.” *NIST TC4TL Challenge*, NIST, 1 July 2020, www.nist.gov/system/files/documents/2020/07/01/2020_NIST_Pilot_TC4TL_Challenge_Evaluation_Plan_v1p3.pdf.

[46] “WHO Coronavirus (COVID-19) Dashboard.” *World Health Organization*, World Health Organization, 25 Aug. 2021, covid19.who.int/.

APPENDIX A FEATURE IMPORTANCE

A.1 Predictor Importance for Features Based on Accelerometer Sensor Data

Feature	Importance
acc_x__absolute_sum_of_changes	2.92496E-05
acc_x__fft_coefficient__attr_"abs"__coeff_1	2.46921E-05
acc_x__mean_abs_change	2.33099E-05
acc_y__mean_abs_change	2.18487E-05
acc_z__fft_coefficient__attr_"abs"__coeff_1	2.16607E-05
acc_z__mean_abs_change	2.13031E-05
acc_z__absolute_sum_of_changes	2.04745E-05
acc_y__fft_coefficient__attr_"abs"__coeff_1	2.04742E-05
acc_y__absolute_sum_of_changes	1.92937E-05
acc_x__quantile__q_0.6	1.61013E-05
acc_x__time_reversal_asymmetry_statistic__lag_1	1.56442E-05
acc_x__autocorrelation__lag_0	1.35567E-05
acc_x__cwt_coefficients__coeff_0__w_5__widths_ (2, 5, 10, 20)	1.19505E-05
acc_x__minimum	1.16347E-05
acc_x__quantile__q_0.8	1.16339E-05
acc_x__cwt_coefficients__coeff_0__w_20__widths_ (2, 5, 10, 20)	1.15786E-05
acc_x__abs_energy	1.13793E-05
acc_x__quantile__q_0.9	1.11264E-05
acc_x__cwt_coefficients__coeff_1__w_2__widths_ (2, 5, 10, 20)	1.10877E-05
acc_x__quantile__q_0.4	1.10048E-05
acc_x__quantile__q_0.7	1.0521E-05
acc_y__cwt_coefficients__coeff_1__w_2__widths_ (2, 5, 10, 20)	1.02811E-05
acc_x__mean	1.01694E-05
acc_z__quantile__q_0.2	9.30498E-06
acc_x__fft_coefficient__attr_"real"__coeff_1	9.29676E-06

acc_x__length	9.21368E-06
acc_y__fft_coefficient__attr_"real"__coeff_1	9.19955E-06
acc_y__autocorrelation__lag_0	9.19913E-06
acc_x__benford_correlation	9.13743E-06
acc_y__cwt_coefficients__coeff_0__w_20__widths_ (2, 5, 10, 20)	9.07809E-06
acc_z__fft_coefficient__attr_"real"__coeff_1	8.94405E-06
acc_z__quantile__q_0.8	8.93077E-06
acc_y__benford_correlation	8.82741E-06
acc_y__quantile__q_0.7	8.77025E-06
acc_z__quantile__q_0.3	8.74172E-06
acc_y__quantile__q_0.6	8.5821E-06
acc_z__cwt_coefficients__coeff_0__w_5__widths_ (2, 5, 10, 20)	8.56369E-06
acc_y__cwt_coefficients__coeff_0__w_10__widths_ (2, 5, 10, 20)	8.53453E-06
acc_y__quantile__q_0.8	8.52491E-06
acc_x__quantile__q_0.3	8.45743E-06
acc_x__cwt_coefficients__coeff_0__w_10__widths_ (2, 5, 10, 20)	8.36018E-06
acc_z__abs_energy	8.29184E-06
acc_z__cwt_coefficients__coeff_1__w_2__widths_ (2, 5, 10, 20)	8.13314E-06
acc_z__cwt_coefficients__coeff_0__w_20__widths_ (2, 5, 10, 20)	7.99846E-06
acc_y__abs_energy	7.98002E-06
acc_y__length	7.87085E-06
acc_z__autocorrelation__lag_0	7.77286E-06
acc_y__quantile__q_0.3	7.71416E-06
acc_z__benford_correlation	7.70786E-06
acc_y__quantile__q_0.2	7.60758E-06
acc_x__quantile__q_0.2	7.43027E-06
acc_z__quantile__q_0.6	7.39519E-06
acc_y__quantile__q_0.9	7.33523E-06
acc_y__minimum	7.10305E-06
acc_z__quantile__q_0.4	6.94949E-06
acc_y__cwt_coefficients__coeff_0__w_5__widths_ (2, 5, 10, 20)	6.75632E-06
acc_z__mean	6.75479E-06

acc_y__mean	6.6312E-06
acc_z__cwt_coefficients__coeff_0__w_10__widths_ (2, 5, 10, 20)	6.52357E-06
acc_y__quantile__q_0.4	6.37809E-06
acc_z__quantile__q_0.7	6.3578E-06
acc_z__minimum	6.15688E-06
acc_z__quantile__q_0.9	5.72905E-06
acc_z__length	5.25758E-06
acc_y__time_reversal_asymmetry_statistic__lag_1	2.75666E-06
acc_z__time_reversal_asymmetry_statistic__lag_1	2.69566E-06
acc_x__lempel_ziv_complexity__bins_2	5.58921E-07
acc_x__fft_coefficient__attr_"angle"__coeff_1	5.08107E-07
acc_x__approximate_entropy__m_2__r_0.1	2.58461E-07
acc_x__count_below__t_0	2.25775E-07
acc_y__lempel_ziv_complexity__bins_2	6.01714E-08
acc_y__fft_coefficient__attr_"angle"__coeff_1	5.26534E-08
acc_y__count_below__t_0	4.87347E-08
acc_y__approximate_entropy__m_2__r_0.1	3.24629E-08
acc_z__range_count__max_0__min_1000000000000.0	3.18586E-08
acc_z__lempel_ziv_complexity__bins_2	2.93348E-08
acc_z__approximate_entropy__m_2__r_0.1	1.7963E-08
acc_z__fft_coefficient__attr_"angle"__coeff_1	1.78132E-08
acc_z__count_below__t_0	1.5534E-08
acc_y__range_count__max_0__min_1000000000000.0	1.39696E-08

A.2 Predictor Importance for Features Based on Gyroscope Sensor Data

Feature	Importance
gyro_z__fft_coefficient__attr_"abs"__coeff_1	2.84421E-05
gyro_y__absolute_sum_of_changes	2.80925E-05
gyro_z__mean_abs_change	2.71903E-05
gyro_y__mean_abs_change	2.60896E-05
gyro_y__fft_coefficient__attr_"abs"__coeff_1	2.57663E-05

gyro_z__absolute_sum_of_changes	2.31541E-05
gyro_x__fft_coefficient__attr_"abs"__coeff_1	2.24368E-05
gyro_x__absolute_sum_of_changes	2.02626E-05
gyro_x__mean_abs_change	1.96577E-05
gyro_y__time_reversal_asymmetry_statistic__lag_1	1.68127E-05
gyro_z__time_reversal_asymmetry_statistic__lag_1	1.65649E-05
gyro_x__time_reversal_asymmetry_statistic__lag_1	1.42673E-05
gyro_x__autocorrelation__lag_0	1.11642E-05
gyro_x__quantile__q_0.2	1.10066E-05
gyro_x__quantile__q_0.7	1.09897E-05
gyro_x__cwt_coefficients__coeff_1__w_2__widths_ (2, 5, 10, 20)	9.95369E-06
gyro_y__quantile__q_0.9	9.869E-06
gyro_x__length	9.69879E-06
gyro_y__quantile__q_0.4	9.69441E-06
gyro_z__cwt_coefficients__coeff_0__w_10__widths_ (2, 5, 10, 20)	9.69363E-06
gyro_x__fft_coefficient__attr_"real"__coeff_1	9.58692E-06
gyro_y__quantile__q_0.6	9.47557E-06
gyro_x__benford_correlation	9.42223E-06
gyro_y__fft_coefficient__attr_"real"__coeff_1	9.30378E-06
gyro_z__quantile__q_0.2	9.27752E-06
gyro_z__quantile__q_0.3	9.1353E-06
gyro_x__cwt_coefficients__coeff_0__w_5__widths_ (2, 5, 10, 20)	9.06342E-06
gyro_z__quantile__q_0.6	9.06283E-06
gyro_z__quantile__q_0.7	8.95777E-06
gyro_y__autocorrelation__lag_0	8.8109E-06
gyro_z__minimum	8.70725E-06
gyro_z__abs_energy	8.62714E-06
gyro_z__quantile__q_0.8	8.39803E-06
gyro_z__length	8.39759E-06
gyro_z__mean	8.35594E-06
gyro_y__minimum	8.35335E-06
gyro_y__quantile__q_0.3	8.34377E-06
gyro_y__cwt_coefficients__coeff_0__w_10__widths_ (2, 5, 10, 20)	8.33001E-06
gyro_x__mean	8.22334E-06
gyro_y__quantile__q_0.2	8.1304E-06

gyro_x__quantile__q_0.9	8.09564E-06
gyro_x__quantile__q_0.3	8.00055E-06
gyro_z__quantile__q_0.4	7.97899E-06
gyro_y__cwt_coefficients__coeff_0__w_5__widths_ (2, 5, 10, 20)	7.95473E-06
gyro_z__fft_coefficient__attr_"real"__coeff_1	7.95095E-06
gyro_z__quantile__q_0.9	7.92326E-06
gyro_y__abs_energy	7.90241E-06
gyro_y__benford_correlation	7.84763E-06
gyro_z__cwt_coefficients__coeff_0__w_20__widths_ (2, 5, 10, 20)	7.77894E-06
gyro_z__cwt_coefficients__coeff_1__w_2__widths_ (2, 5, 10, 20)	7.7693E-06
gyro_z__cwt_coefficients__coeff_0__w_5__widths_(2, 5, 10, 20)	7.66005E-06
gyro_x__quantile__q_0.6	7.61073E-06
gyro_z__benford_correlation	7.49373E-06
gyro_x__cwt_coefficients__coeff_0__w_20__widths_(2, 5, 10, 20)	7.45128E-06
gyro_y__quantile__q_0.8	7.34678E-06
gyro_x__quantile__q_0.4	7.17652E-06
gyro_x__abs_energy	7.15876E-06
gyro_y__length	7.10458E-06
gyro_y__mean	6.93309E-06
gyro_y__cwt_coefficients__coeff_1__w_2__widths_(2, 5, 10, 20)	6.84515E-06
gyro_y__quantile__q_0.7	6.79714E-06
gyro_y__cwt_coefficients__coeff_0__w_20__widths_(2, 5, 10, 20)	6.66148E-06
gyro_z__autocorrelation__lag_0	6.3747E-06
gyro_x__quantile__q_0.8	6.32307E-06
gyro_x__cwt_coefficients__coeff_0__w_10__widths_(2, 5, 10, 20)	6.03442E-06
gyro_x__minimum	5.88025E-06
gyro_z__lempel_ziv_complexity__bins_2	6.57121E-07
gyro_y__lempel_ziv_complexity__bins_2	6.37286E-07
gyro_z__fft_coefficient__attr_"angle"__coeff_1	5.55441E-07
gyro_y__fft_coefficient__attr_"angle"__coeff_1	5.43819E-07
gyro_x__fft_coefficient__attr_"angle"__coeff_1	5.22187E-07

gyro_x__lempel_ziv_complexity__bins_2	4.72632E-07
gyro_y__approximate_entropy__m_2__r_0.1	4.04478E-07
gyro_y__count_below__t_0	3.11687E-07
gyro_z__count_below__t_0	3.02908E-07
gyro_z__approximate_entropy__m_2__r_0.1	2.83714E-07
gyro_x__count_below__t_0	2.19721E-07
gyro_x__approximate_entropy__m_2__r_0.1	1.84156E-07

APPENDIX B SOURCE CODE

B.1 Data filtering

```
%-----------------------------------------------------------------%
% Routine to filter sensor's data
%-----------------------------------------------------------------%
clear
load('test_acc_user11c.mat')
load('test_gyro_user11c.mat')
load('test_bt_user11c.mat')
fc_acc = 0.2; %accelerometer cut-off frequency
fc_gyro = 0.1; %gyroscope cut-off frequency
%fc_bt = 1;
fs = 4; %sampling frequency

%moving avearage filter
% windowSize = 10;
% b1 = (1/windowSize)*ones(1,windowSize);
% a1 = 1;
%butterworth filter 5th order
%[b2,a2] = butter(9,fc/(fs/2));
%FIR
%frequency bands for a band-pass filter
% fcuts = [0.16 0.22 0.28 0.3 0.45 0.5 1.28 1.35 1.5 1.55 ];% 3 Frequencie bands
% mags = [1 0 1 0 1 0];
% devs = [0.05 0.01 0.05 0.01 0.05 0.01];
% fcuts = [0.16 0.22 0.28 0.3 0.45 0.5 ];% 1,2 Frequencie bands
% mags = [1 0 1 0];
% devs = [0.05 0.01 0.05 0.01];
% fcuts = [0.16 0.22 1.28 1.35 1.5 1.55 ];% 1,3 Frequencie bands
% mags = [1 0 1 0];
% devs = [0.05 0.01 0.05 0.01];
% fcuts = [0.28 0.3 0.45 0.5 1.28 1.35 1.5 1.55 ];% 2,3 Frequencie bands
% mags = [0 1 0 1 0];
% devs = [0.01 0.05 0.01 0.05 0.01];
% [n,Wn,beta,ftype] = kaiserord(fcuts,mags,devs,fs);          % Kaiser Window FIR
Specification
% n = n + rem(n,2);
% hh = fir1(n,Wn,ftype,kaiser(n+1,beta),'noscale');          % Filter Realisation
% figure
% freqz(hh,1,2^14,fs)
% set(subplot(2,1,1), 'XLim',[0 2]);                         % Zoom Frequency Axis
% set(subplot(2,1,2), 'XLim',[0 2]);                         % Zoom Frequency Axis
% acc_filt_x = filtfilt(hh, 1, acc_data_proc.acc_x(:,1));
% acc_filt_y = filtfilt(hh, 1, acc_data_proc.acc_x(:,2));
% acc_filt_z = filtfilt(hh, 1, acc_data_proc.acc_x(:,3));
%
% gyro_filt_x = filtfilt(hh, 1, gyro_data_proc.gyro_x(:,1));
% gyro_filt_y = filtfilt(hh, 1, gyro_data_proc.gyro_x(:,2));
% gyro_filt_z = filtfilt(hh, 1, gyro_data_proc.gyro_x(:,3));

%eliptical
[b_acc,a_acc] = ellip(9,5,20,fc_acc/(fs/2)); %(10,10,20,fc_acc/(fs/2))
```

```
[b_gyro,a_gyro] = ellip(7,5,20,fc_gyro/(fs/2));
%filter the data
acc_filt_x = filtfilt(b_acc,a_acc, acc_data_proc.acc_x(:,1));
acc_filt_y = filtfilt(b_acc,a_acc, acc_data_proc.acc_x(:,2));
acc_filt_z = filtfilt(b_acc,a_acc, acc_data_proc.acc_x(:,3));

gyro_filt_x = filtfilt(b_gyro,a_gyro, gyro_data_proc.gyro_x(:,1));
gyro_filt_y = filtfilt(b_gyro,a_gyro, gyro_data_proc.gyro_x(:,2));
gyro_filt_z = filtfilt(b_gyro,a_gyro, gyro_data_proc.gyro_x(:,3));
%Raised cosine filter
%h1 = rcosdesign(0.25,4,6,'normal');
%chabyshev
%[b,a] = cheby1(10,10,fc/(fs/2));
%median filter
% m_filt_order = 4;
% acc_filt_x = medfilt1(acc_data_proc.acc_x(:,1), m_filt_order);
% acc_filt_y = medfilt1(acc_data_proc.acc_x(:,2), m_filt_order);
% acc_filt_z = medfilt1(acc_data_proc.acc_x(:,3), m_filt_order);
%
% gyro_filt_x = medfilt1(gyro_data_proc.gyro_x(:,1), m_filt_order);
% gyro_filt_y = medfilt1(gyro_data_proc.gyro_x(:,2), m_filt_order);
% gyro_filt_z = medfilt1(gyro_data_proc.gyro_x(:,3), m_filt_order);
%
% bt_filt_d = medfilt1(bt_data_proc.bt_x(:,1), m_filt_order);
% bt_filt_rx = medfilt1(bt_data_proc.bt_x(:,2), m_filt_order);
% bt_filt_tx = medfilt1(bt_data_proc.bt_x(:,3), m_filt_order);
%no filtering on Bluetooth signal
```

```
bt_filt_d = bt_data_proc.bt_x(:,1);
bt_filt_rx = bt_data_proc.bt_x(:,2);
bt_filt_tx = bt_data_proc.bt_x(:,3);

%compute magnitude of a filtered signal
acc_magnitude = sqrt(acc_filt_x.^2 + acc_filt_y.^2 + acc_filt_z.^2);
acc_xy_mag = sqrt(acc_filt_x.^2 + acc_filt_y.^2);
acc_xz_mag = sqrt(acc_filt_x.^2 + acc_filt_z.^2);
acc_yz_mag = sqrt(acc_filt_y.^2 + acc_filt_z.^2);
%conditions
ax_range = acc_data_proc.ax_range;
ax_angle = acc_data_proc.ax_angle;
acc_env_categories = acc_data_proc.acc_env_categories;
acc_pos_loc_categories = acc_data_proc.acc_pos_loc_categories;
%compute magnitude of a filtered signal
gyro_magnitude = sqrt(gyro_filt_x.^2 + gyro_filt_y.^2 + gyro_filt_z.^2);
gyro_xy_mag = sqrt(gyro_filt_x.^2 + gyro_filt_y.^2);
gyro_xz_mag = sqrt(gyro_filt_x.^2 + gyro_filt_z.^2);
gyro_yz_mag = sqrt(gyro_filt_y.^2 + gyro_filt_z.^2);
gyro_range = gyro_data_proc.gyro_range;
gyro_angle = gyro_data_proc.gyro_angle;
gyro_env_categories = gyro_data_proc.gyro_env_categories;
gyro_pos_loc_categories = gyro_data_proc.gyro_pos_loc_categories;
%cary over bluetooth data
bt_rx = bt_data_proc.bt_x(:,2);
bt_tx = bt_data_proc.bt_x(:,3);
bt_time = bt_data_proc.bt_x(:,4);
```

```
bt_delta = bt_rx - bt_tx;
bt_range = bt_data_proc.bt_range;
bt_angle = bt_data_proc.bt_angle;
bt_env_categories = bt_data_proc.bt_env_categories;
bt_pos_loc_categories = bt_data_proc.bt_pos_loc_categories;
bt_x = [bt_filt_d bt_filt_rx bt_filt_tx bt_time];
%bt_x = [bt_delta bt_rx bt_tx bt_time];
acc_x = [acc_filt_x acc_filt_y acc_filt_z acc_magnitude acc_xy_mag acc_xz_mag
acc_yz_mag ];
gyro_x = [gyro_filt_x gyro_filt_y gyro_filt_z gyro_magnitude gyro_xy_mag
gyro_xz_mag gyro_yz_mag ];
%processed data
acc_data_proc = table(acc_x, ax_range, ax_angle, acc_env_categories,
acc_pos_loc_categories);
gyro_data_proc = table(gyro_x, gyro_range, gyro_angle, gyro_env_categories,
gyro_pos_loc_categories);
bt_data_proc = table(bt_x, bt_range, bt_angle, bt_env_categories,
bt_pos_loc_categories);

%save data to mat file
save('train_acc_user11SMOTE.mat','acc_data_proc')
save('train_gyro_user11SMOTE.mat','gyro_data_proc')
save('train_bt_user11SMOTE.mat','bt_data_proc')

% visualize filtered data
% figure
% plot(acc_data_proc.acc_x(1:300,1),'c')
% hold on
% plot(acc_filt_but(1:300),'r')
% hold on
% plot(acc_filt_ma(1:300),'b')
% hold on
% plot(acc_filt_ep(1:300),'g')
% hold on
% plot(acc_filt_ch(1:300),'k')
% hold off
% legend('Raw data','Butterworth 5th', 'Moving Average w5', 'Eliptical 5th',
'Chabishev I 5th')
% title('Filter Types vs Accelerometer Data')
% grid minor
%--------------------------------------------------------------%
```

B.2 Routine to generate hand-crafted features

```
%--------------------------------------------------------------------%
% Routine to compute hand crafted features
%--------------------------------------------------------------------%
clear
% load the data
load('test_acc_user11cf.mat')
load('test_gyro_user11cf.mat')
load('test_bt_user11cf.mat')

%----fix irroneous transmit level power------5-15-2021----%
bt_data_proc((bt_data_proc.bt_x(:,3)<-20),:)=[];
bt_data_proc((bt_data_proc.bt_x(:,3)>20),:)=[];
%bt_data_proc.bt_x((bt_data_proc.bt_x(:,3)>-4),3)=-10;
bt_data_proc((bt_data_proc.bt_x(:,1)>-20),:)=[];
%-----------------------------------------------------%
window_size = 10; % window size over which statistical features calculated
%average_over = 3;
%---------------------------------------------------------------%
%calculate moving average
% acc_data_proc.acc_x = movmean(acc_data_proc.acc_x, average_over);
% gyro_data_proc.gyro_x = movmean(gyro_data_proc.gyro_x, average_over);
% bt_data_proc.bt_x = movmean(bt_data_proc.bt_x, average_over);
%statistics of magnitude signal
%calculate magnitude mean
acc_mag_mean = Fmean(acc_data_proc.acc_x(:,4),window_size);
gyro_mag_mean = Fmean(gyro_data_proc.gyro_x(:,4),window_size);
bt_mag_mean = Fmean(bt_data_proc.bt_x(:,1),window_size);
%calculate standard deviation of magnitude
acc_mag_std = Fstd(acc_data_proc.acc_x(:,4),window_size);
gyro_mag_std = Fstd(gyro_data_proc.gyro_x(:,4),window_size);
bt_mag_std = Fstd(bt_data_proc.bt_x(:,1),window_size);
%calculate third and fourth moment of magnitude
[acc_M3, acc_M4] = Fmoment(acc_data_proc.acc_x(:,4),window_size);
[gyro_M3, gyro_M4] = Fmoment(gyro_data_proc.gyro_x(:,4),window_size);
[bt_M3, bt_M4] = Fmoment(bt_data_proc.bt_x(:,1),window_size);
%calculate 25th, 50th, and 75th percentile
[acc_P25, acc_P50, acc_P75] = Fprctile(acc_data_proc.acc_x(:,4),window_size);
```

```
[gyro_P25, gyro_P50, gyro_P75] =
Fprctile(gyro_data_proc.gyro_x(:,4),window_size);
[bt_P25, bt_P50, bt_P75] = Fprctile(bt_data_proc.bt_x(:,1),window_size);
%value and time entopy
[acc_ValE, acc_TimeE] = Fentropy(acc_data_proc.acc_x(:,4),window_size);
[gyro_ValE, gyro_TimeE] = Fentropy(gyro_data_proc.gyro_x(:,4),window_size);
[bt_ValE, ~] = Fentropy(bt_data_proc.bt_x(:,1),window_size);
%----------------------------------------------------------------%
%auto-correlation and auto-covariance
[acc_Acov, acc_Acor] = Fauto(acc_data_proc.acc_x(:,4),window_size);
[gyro_Acov, gyro_Acor] = Fauto(gyro_data_proc.gyro_x(:,4),window_size);
[bt_Acov, bt_Acor] = Fauto(bt_data_proc.bt_x(:,1),window_size);
%----------------------------------------------------------------%
%statistics of 3-axis
%accelerometer
%mean
acc_x_mean = Fmean(acc_data_proc.acc_x(:,1),window_size);
acc_y_mean = Fmean(acc_data_proc.acc_x(:,2),window_size);
acc_z_mean = Fmean(acc_data_proc.acc_x(:,3),window_size);
%standard deviation
acc_x_std = Fstd(acc_data_proc.acc_x(:,1),window_size);
acc_y_std = Fstd(acc_data_proc.acc_x(:,2),window_size);
acc_z_std = Fstd(acc_data_proc.acc_x(:,3),window_size);
%correlation xy, xz, yz
acc_corr = Fcorr(acc_data_proc.acc_x(:,1), acc_data_proc.acc_x(:,2), ...
    acc_data_proc.acc_x(:,3), window_size);
%gyroscope
```

```
%mean
gyro_x_mean = Fmean(gyro_data_proc.gyro_x(:,1),window_size);
gyro_y_mean = Fmean(gyro_data_proc.gyro_x(:,2),window_size);
gyro_z_mean = Fmean(gyro_data_proc.gyro_x(:,3),window_size);
%standard deviation
gyro_x_std = Fstd(gyro_data_proc.gyro_x(:,1),window_size);
gyro_y_std = Fstd(gyro_data_proc.gyro_x(:,2),window_size);
gyro_z_std = Fstd(gyro_data_proc.gyro_x(:,3),window_size);
%correlation xy, xz, yz
gyro_corr = Fcorr(gyro_data_proc.gyro_x(:,1), gyro_data_proc.gyro_x(:,2), ...
    gyro_data_proc.gyro_x(:,3), window_size);
%bluetooth
%mean
bt_rx_mean = Fmean(bt_data_proc.bt_x(:,2),window_size);
bt_tx_mean = Fmean(bt_data_proc.bt_x(:,3),window_size);
%standard deviation
bt_rx_std = Fstd(bt_data_proc.bt_x(:,2),window_size);
bt_tx_std = Fstd(bt_data_proc.bt_x(:,3),window_size);
%--------additional features for bluetooth
%RMS Doppler Spread
bt_rx_dopler = FRmsDoppler(bt_data_proc.bt_x(:,2),window_size);
%Yp2p
bt_rx_yp2p = Yp2p(bt_data_proc.bt_x(:,2),window_size);
%Rayleigh
bt_rx_rayleigh = Frayleigh(bt_data_proc.bt_x(:,2),window_size);
%Fade Duration
bt_rx_fadeDuration = FfadeDuration(bt_data_proc.bt_x(:,2),window_size);
```

```
%Level Crossing
bt_rx_levelX = FlevelX(bt_data_proc.bt_x(:,2),window_size);
%Energy
bt_rx_energy = Fenergy(bt_data_proc.bt_x(:,2),window_size);
%Laplacian Best Fit
%bt_rx_laplacian = Flaplacian(bt_data_proc.bt_x(:,2),window_size);
%Laplacian Best Fit v2
%bt_rx_laplacian2 = Flaplacian2(bt_data_proc.bt_x(:,2),window_size);
%Gaussian fit
%bt_rx_gauss = Fgauss(bt_data_proc.bt_x(:,2),window_size);
%Polynomial Fit
%bt_rx_poly = Fpoly(bt_data_proc.bt_x(:,2),window_size);
%peak of Doppler spread
bt_rx_peakD = FpeakDoppler(bt_data_proc.bt_x(:,2),window_size);
%mean of Doppler spread
bt_rx_meanD = FmeanDoppler(bt_data_proc.bt_x(:,2),window_size);
%---------------------%
%bt stats on advertiser time
bt_time_mean = Fmean(bt_data_proc.bt_x(:,4),window_size);
bt_time_std = Fstd(bt_data_proc.bt_x(:,4),window_size);
[bt_time_P25, bt_time_P50, bt_time_P75] =
Fprctile(bt_data_proc.bt_x(:,4),window_size);
[bt_time_M3, bt_time_M4] = Fmoment(bt_data_proc.bt_x(:,4),window_size);
%-----------------------------------------------------------------%
%get labels
%accelerometer
acc_range = acc_data_proc.ax_range(1:window_size:end);
```

```
acc_angle = acc_data_proc.ax_angle(1:window_size:end);
acc_categories = [acc_data_proc.acc_env_categories(1:window_size:end,:)...
    acc_data_proc.acc_pos_loc_categories(1:window_size:end,:)];

acc_stats = [acc_mag_mean, acc_mag_std, acc_M3, acc_M4, acc_P25, acc_P50, ...
    acc_P75, acc_ValE, acc_TimeE, acc_Acov, acc_Acor, acc_x_mean,
acc_y_mean, ...
    acc_z_mean, acc_x_std, acc_y_std, acc_z_std, acc_corr];
%acc_stats = acc_data_proc.acc_x(:,1:3);
%  acc_range = acc_range(1:end-1);
%  acc_angle = acc_angle(1:end-1);
%  acc_categories = acc_categories(1:end-1,:);
acc_dat_test = table(acc_stats, acc_range, acc_angle, acc_categories);
%-----------------------------------------------------------------%
gyro_range = gyro_data_proc.gyro_range(1:window_size:end);
gyro_angle = gyro_data_proc.gyro_angle(1:window_size:end);
gyro_categories = [gyro_data_proc.gyro_env_categories(1:window_size:end,:)...
    gyro_data_proc.gyro_pos_loc_categories(1:window_size:end,:)];

gyro_stats = [gyro_mag_mean, gyro_mag_std, gyro_M3, gyro_M4, gyro_P25,
gyro_P50, ...
    gyro_P75, gyro_ValE, gyro_TimeE, gyro_Acov, gyro_Acor, gyro_x_mean,
gyro_y_mean, ...
    gyro_z_mean, gyro_x_std, gyro_y_std, gyro_z_std, gyro_corr];
%gyro_stats = gyro_data_proc.gyro_x(:,1:3);

%  gyro_range = gyro_range(1:end-1);
% gyro_angle = gyro_angle(1:end-1);
% gyro_categories = gyro_categories(1:end-1,:);
gyro_dat_test = table(gyro_stats, gyro_range, gyro_angle, gyro_categories);
%-----------------------------------------------------------------%
bt_range = bt_data_proc.bt_range(1:window_size:end);
% bt_range = bt_range(1:end-1);
bt_angle = bt_data_proc.bt_angle(1:window_size:end);
 %bt_angle = bt_angle(1:end-1);
bt_time = bt_data_proc.bt_x(1:window_size:end,4);
%bt_time = bt_time(1:end-1);
bt_categories = [bt_data_proc.bt_env_categories(1:window_size:end,:)...
    bt_data_proc.bt_pos_loc_categories(1:window_size:end,:)];
 %bt_categories = bt_categories(1:end-1,:);
bt_stats = [bt_rx_mean,bt_tx_mean, bt_time,bt_mag_mean, bt_mag_std, bt_M3,
bt_M4, bt_P25, ...
    bt_P50, bt_P75, bt_ValE, bt_Acov, bt_Acor, ...
    bt_rx_std, bt_tx_std, bt_rx_yp2p, bt_rx_rayleigh, bt_rx_fadeDuration,...
    bt_rx_levelX, bt_rx_energy, bt_rx_dopler,...
    bt_rx_peakD, bt_rx_meanD, ...
    bt_time_mean,bt_time_std, bt_time_P25,bt_time_P50, bt_time_P75,
bt_time_M3, bt_time_M4];
%bt_rx_laplacian, bt_rx_laplacian2, bt_rx_gauss, bt_rx_poly,
%bt_stats = bt_data_proc.bt_x(:,2:4);
bt_dat_test = table(bt_stats, bt_range, bt_angle, bt_categories);

%save data to mat file
save('test_acc_user11cff4.mat','acc_dat_test')
save('test_gyro_user11cff4.mat','gyro_dat_test')
save('test_bt_user11cff4.mat','bt_dat_test')
%-------------------------------------------------------------------------------%
```

B.3 Routine to classify user context

```
%-------------------------------------------------------------------------------%
% routine to classify user context
%-------------------------------------------------------------------------------%
clear
load('test_acc_user11cff3.mat')
load('test_gyro_user11cff3.mat')
load('test_bt_user11cff3.mat')

load('train_acc_user11cff3.mat')
load('train_gyro_user11cff3.mat')
load('train_bt_user11cff3.mat')
%randomly shufle the data
%test data
h = height(acc_dat_test);
idx = randperm(h);
acc_data_rand_test = acc_dat_test(idx,:);
gyro_data_rand_test = gyro_dat_test(idx,:);

h1 = height(bt_dat_test);
idx1 = randperm(h1);
bt_data_rand_test = bt_dat_test(idx1,:);
%training data
```

```
h = height(acc_dat_train);
idx = randperm(h);
acc_data_rand_train = acc_dat_train(idx,:);
gyro_data_rand_train = gyro_dat_train(idx,:);

h1 = height(bt_dat_train);
idx1 = randperm(h1);
bt_data_rand_train = bt_dat_train(idx1,:);
%Normalize features data
%accelerometer
acc_norm_test = featureNormalize2(acc_data_rand_test.acc_stats, 'Zscale');
acc_norm_train = featureNormalize2(acc_data_rand_train.acc_stats, 'Zscale');
%gyroscope
gyro_norm_test = featureNormalize2(gyro_data_rand_test.gyro_stats, 'Zscale');
gyro_norm_train = featureNormalize2(gyro_data_rand_train.gyro_stats, 'Zscale');
%bluetooth
bt_norm_test = abs(featureNormalize2(bt_data_rand_test.bt_stats, 'Zscale'));
bt_norm_test(:,12) = [];
bt_norm_train = abs(featureNormalize2(bt_data_rand_train.bt_stats, 'Zscale'));
bt_norm_train(:,12) = [];
%prepare data for sensor fusion
[acc_lbl_train, acc_lbl_test, gyro_lbl_train, gyro_lbl_test, ...
   bt_lbl_train, bt_lbl_test, all_cnt]= group_cat(acc_data_rand_test, ...
   acc_data_rand_train, gyro_data_rand_test, gyro_data_rand_train, ...
   bt_data_rand_test, bt_data_rand_train);
%-------------------------------------------------------------%
for c = 1:size(acc_lbl_train,2)
```

```
    t2 = templateTree('MinLeafSize', 1);
%train -------------------------------------------------------------%
    cost = compute_cost_multi(acc_lbl_train(:,c),c);

    Mdl_acc = fitcensemble(acc_norm_train, acc_lbl_train(:,c),'Method','Bag',...
        'NumLearningCycles',15,'Learners',t2,'Cost', cost); %try 265 Nlerncycles

    cost = compute_cost_multi(gyro_lbl_train(:,c),c);
    Mdl_gyro = fitcensemble(gyro_norm_train, gyro_lbl_train(:,c),'Method','Bag' ,...
        'NumLearningCycles',15,'Learners',t2, 'Cost', cost); %try 265 Nlerncycles

    cost = compute_cost_multi(bt_lbl_train(:,c),c);
    Mdl_bt = fitctree(bt_norm_train, bt_lbl_train(:,c),'MinLeafSize', 7, ...
        'Surrogate', 'off', 'Cost', cost);
%-------------------------------------------------------------------%
    %apply model on a test set
    [prediction_acc(:,c), ~]= predict(Mdl_acc, acc_norm_test);
    [prediction_gyro(:,c), ~]= predict(Mdl_gyro, gyro_norm_test);
    [prediction_bt(:,c), ~]= predict(Mdl_bt, bt_norm_test);

    %apply model on a train set
    [prediction_acc_t(:,c), ~]= predict(Mdl_acc, acc_norm_train);
    [prediction_gyro_t(:,c), ~]= predict(Mdl_gyro, gyro_norm_train);
    [prediction_bt_t(:,c), ~]= predict(Mdl_bt, bt_norm_train);
%-------------------------------------------------------------------%
end
```

```
%performance metrics
[F1_test_acc, BA_test_acc] = scores_multi(acc_lbl_test,prediction_acc);
[F1_test_gyro, BA_test_gyro] = scores_multi(gyro_lbl_test,prediction_gyro);
[F1_test_bt, BA_test_bt] = scores_multi(bt_lbl_test,prediction_bt);
%-------------------------------------------------------------%
lbl_txt = ["Indoor" "outdoor" "large room" "medium room"...
   "small room" "center congested" "center open" "near wall congested"...
   "near wall open" "Sitting" "Standing" "Hold to right ear"...
   "front pants pocket" "in hand" "in purse" "rear pants pocket" "shirt pocket"];
%number of positive samples
n_e_acc = sum(acc_data_rand_test.acc_categories)';
n_e_gyro = sum(gyro_data_rand_test.gyro_categories)';
n_e_bt = sum(bt_data_rand_test.bt_categories)';

n_te_acc = sum(acc_data_rand_train.acc_categories)';
n_te_gyro = sum(gyro_data_rand_train.gyro_categories)';
n_te_bt = sum(bt_data_rand_train.bt_categories)';
%save results
results_ba = table(lbl_txt', n_te_acc, n_e_acc, BA_test_acc, n_te_gyro, ...
   n_e_gyro, BA_test_gyro, n_te_bt, n_e_bt, BA_test_bt);
results_f1 = table(lbl_txt', n_te_acc, n_e_acc, F1_test_acc, n_te_gyro, ...
   n_e_gyro, F1_test_gyro, n_te_bt, n_e_bt, F1_test_bt);

writetable(results_ba, 'results_pact_BA_user11tsfresh.xlsx');
writetable(results_f1, 'results_pact_F1_user11tsfresh.xlsx');

%---------------------------------------------------%
```

```
%add classified context to the data file as a feature
%training set
acc_stats = [acc_dat_train.acc_stats, prediction_acc_t];
acc_range = acc_dat_train.acc_range;
acc_angle = acc_dat_train.acc_angle;
acc_dat_train = table(acc_stats, acc_range, acc_angle);

gyro_stats = [gyro_dat_train.gyro_stats, prediction_gyro_t];
gyro_range = gyro_dat_train.gyro_range;
gyro_angle = gyro_dat_train.gyro_angle;
gyro_dat_train = table(gyro_stats, gyro_range, gyro_angle);

bt_stats = [bt_dat_train.bt_stats, prediction_bt_t];
bt_range = bt_dat_train.bt_range;
bt_angle = bt_dat_train.bt_angle;
bt_dat_train = table(bt_stats, bt_range, bt_angle);

%save data to mat file
save('train_acc_user11cffc3.mat','acc_dat_train')
save('train_gyro_user11cffc3.mat','gyro_dat_train')
save('train_bt_user11cffc3.mat','bt_dat_train')
%--------------------------------------------------%
%test set
acc_stats = [acc_dat_test.acc_stats, prediction_acc];
acc_range = acc_dat_test.acc_range;
acc_angle = acc_dat_test.acc_angle;
acc_dat_test = table(acc_stats, acc_range, acc_angle);
```

```
gyro_stats = [gyro_dat_test.gyro_stats, prediction_gyro];
gyro_range = gyro_dat_test.gyro_range;
gyro_angle = gyro_dat_test.gyro_angle;
gyro_dat_test= table(gyro_stats, gyro_range, gyro_angle);

bt_stats = [bt_dat_test.bt_stats, prediction_bt];
bt_range = bt_dat_test.bt_range;
bt_angle = bt_dat_test.bt_angle;
bt_dat_test = table(bt_stats, bt_range, bt_angle);

%save data to mat file
save('test_acc_user11cffc3.mat','acc_dat_test')
save('test_gyro_user11cffc3.mat','gyro_dat_test')
save('test_bt_user11cffc3.mat','bt_dat_test')
%------------------------------------------------------------%
```

B.4 Routines to perform range estimation

```
%------------------------------------------------------------------%
% Routine to do range estimation using 5fold cross validation
%------------------------------------------------------------------%
clear
load('data_acc_plus_class.mat')
load('data_gyro_plus_class.mat')
load('data_bt_plus_class.mat')
%randomly shuffle the data
h = height(acc_dat);
```

```
idx = randperm(h);
acc_data_rand = acc_dat(idx,:);
gyro_data_rand = gyro_dat(idx,:);

h1 = height(bt_dat);
idx1 = randperm(h1);
bt_data_rand = bt_dat(idx1,:);
%normalize features
%accelerometer
acc_norm = featureNormalize2(acc_data_rand.acc_stats(:,1:20), "Zscale");
acc_norm =[acc_norm, acc_data_rand.acc_stats(:,21:end)];
%acc_norm =acc_data_rand.acc_stats(:,21:end);
acc_range= acc_data_rand.acc_range;
%gyroscope
gyro_norm = featureNormalize2(gyro_data_rand.gyro_stats(:,1:20), "Zscale");
gyro_norm = [gyro_norm, gyro_data_rand.gyro_stats(:,21:end)];
%gyro_norm = gyro_data_rand.gyro_stats(:,21:end);
gyro_range = gyro_data_rand.gyro_range;
%bluetooth
bt_norm = featureNormalize2(bt_data_rand.bt_stats(:,1:14), "Zscale");
bt_norm = [bt_norm, bt_data_rand.bt_stats(:,15:end)];
bt_range = bt_data_rand.bt_range;
%----------------------------------------------------------------%
%regrassion on range
predNames = {'Mean', 'STD', 'M3', 'M4', '25%', '50%', '75%',...
    'Value Entropy', 'Time Entropy', 'Autocorelation', 'Autocovariance',...
    'X Mean', 'Y Mean', 'Z Mean', 'X STD', 'Y STD', 'Z STD', ...
    'Autocorelation XY', 'Autocorelation XZ', 'Autocorelation YZ'};
predNamesBT = {'RX Mean', 'TX Mean', 'Advertiser Time', 'Delta Mean',...
    'Delta STD', 'M3', 'M4', '25%', '50%', '75%',...
    'Value Entropy', 'Autocorelation', 'RX STD', 'TX STD'};
%range
t2 = templateTree('MinLeafSize', 1);
Mdl_acc_reg = fitrensemble(acc_norm, acc_range, ...
    'Method','Bag','NumLearningCycles', 492,'Learners',t2, 'CrossVal','on',...
    'KFold',5); %try 265 Nlerncycles , 'PredictorNames', predNames
fitted_acc= kfoldPredict(Mdl_acc_reg);

Mdl_gyro_reg = fitrensemble(gyro_norm, gyro_range, ...
    'Method','Bag','NumLearningCycles',477,'Learners',t2, 'CrossVal','on',...
    'KFold',5); %try 265 Nlerncycles , 'PredictorNames', predNames
fitted_gyro= kfoldPredict(Mdl_gyro_reg);

Mdl_bt_reg = fitrtree(bt_norm, bt_range,'MinLeafSize', 2, ...
    'Surrogate', 'off', 'CrossVal','on','KFold',5); %, 'PredictorNames', predNamesBT
fitted_bt= kfoldPredict(Mdl_bt_reg);
%------------------------------------------------------------%
%Sensor fusion
%prepare training data
[ef_data_r, lbl_r] = prep_ef_data_reg2(acc_norm, gyro_norm,...
    bt_norm, acc_range, gyro_range, bt_range);
Mdl_ef = fitrtree(ef_data_r, lbl_r, 'MinLeafSize', 2, ...
    'Surrogate', 'off', 'CrossVal', 'on', 'KFold',5);
prediction_ef = kfoldPredict(Mdl_ef);
```

```
%--------------------------------------------------------------%
%Fusion2
acc_nn = ef_data_r(:,1:37); %with labels (:,1:37) ///without labels (:,1:20)
gyro_nn = ef_data_r(:,38:74); %with labels (:,38:74) ///without labels )(:,21:40)
bt_nn = ef_data_r(:,75:end); %with labels (:,75:end) ///without labels(:,41:end)
%first layer
Mdl_acc_nn = fitrensemble(acc_nn, lbl_r, 'Method','Bag',...
    'NumLearningCycles',30,'Learners',t2, 'CrossVal','on','KFold',5);
Mdl_gyro_nn = fitrensemble(gyro_nn, lbl_r, 'Method','Bag',...
    'NumLearningCycles',30,'Learners',t2, 'CrossVal','on','KFold',5);
Mdl_bt_nn = fitrtree(bt_nn, lbl_r,'MinLeafSize', 4, ...
    'Surrogate', 'off', 'CrossVal','on','KFold',5);

acc_nn_l1 = kfoldPredict(Mdl_acc_nn);
gyro_nn_l1 = kfoldPredict(Mdl_gyro_nn);
bt_nn_l1 = kfoldPredict(Mdl_bt_nn);
%layer 2 (output layer)
nn_dat_l2 = [acc_nn_l1 gyro_nn_l1 bt_nn_l1];
nn_dat_l2_norm = featureNormalize(nn_dat_l2); %try without normalization
Mdl_nn_l2 = fitrensemble(nn_dat_l2, lbl_r, 'Method','Bag',...
    'NumLearningCycles',30,'Learners',t2, 'CrossVal','on','KFold',5);
prediction_nn = kfoldPredict(Mdl_nn_l2);
%--------------------------------------------------------------%
% fit error/performance metrics
[mse_acc, rmse_acc, r2_acc, mae_acc] = fit_error(acc_range, fitted_acc);
[mse_gyro, rmse_gyro, r2_gyro, mae_gyro] = fit_error(gyro_range, fitted_gyro);
[mse_bt, rmse_bt, r2_bt, mae_bt] = fit_error(bt_range, fitted_bt);
```

```
[mse_ef, rmse_ef, r2_ef, mae_ef] = fit_error(lbl_r, prediction_ef);
[mse_nn, rmse_nn, r2_nn, mae_nn] = fit_error(lbl_r, prediction_nn);
%save results
methods = ["MSE" "RMSE" "R2" "MAE"];
results_reg = table(methods', [mse_acc rmse_acc r2_acc mae_acc]',...
    [mse_gyro rmse_gyro r2_gyro mae_gyro]',...
    [mse_bt rmse_bt r2_bt mae_bt]', ...
    [mse_ef rmse_ef r2_ef mae_ef]', ...
    [mse_nn rmse_nn r2_nn mae_nn]');

writetable(results_reg, 'results_pact_linReg13xLablBest3.xlsx');
%-----------------------------------------------------------------%



%----------------------------------------------------------------------%
% Routine to do range estimation using subject-level splitting cross-validation
%----------------------------------------------------------------------%
clear
load('train_acc_user11cffc3.mat')
load('train_gyro_user11cffc3.mat')
load('train_bt_user11cffc3.mat')

load('test_acc_user11cffc3.mat')
load('test_gyro_user11cffc3.mat')
load('test_bt_user11cffc3.mat')
% %randomly shufle the data
h = height(acc_dat_test);
```

```
idx = randperm(h);
acc_data_test_rand = acc_dat_test(idx,:);
gyro_data_test_rand = gyro_dat_test(idx,:);

h1 = height(bt_dat_test);
idx1 = randperm(h1);
bt_data_test_rand = bt_dat_test(idx1,:);
%-------------------------------
h = height(acc_dat_train);
idx = randperm(h);
acc_data_train_rand = acc_dat_train(idx,:);
gyro_data_train_rand = gyro_dat_train(idx,:);

h1 = height(bt_dat_train);
idx1 = randperm(h1);
bt_data_train_rand = bt_dat_train(idx1,:);
%normalize features
%accelerometer
acc_norm_train = featureNormalize2(acc_data_train_rand.acc_stats(:,1:20),
"Zscale");
acc_norm_train =[acc_norm_train,
featureNormalize2(acc_data_train_rand.acc_stats(:,21:end), "Zscale")];

acc_norm_test = featureNormalize2(acc_data_test_rand.acc_stats(:,1:20),
"Zscale");
```

```
acc_norm_test
=[acc_norm_test,featureNormalize2(acc_data_test_rand.acc_stats(:,21:end),
"Zscale")];
acc_train_range= acc_data_train_rand.acc_range;
acc_test_range= acc_data_test_rand.acc_range;
%gyroscope
gyro_norm_train = featureNormalize2(gyro_data_train_rand.gyro_stats(:,1:20),
"Zscale");
gyro_norm_train = [gyro_norm_train,
featureNormalize2(gyro_data_train_rand.gyro_stats(:,21:end), "Zscale")];

gyro_norm_test = featureNormalize2(gyro_data_test_rand.gyro_stats(:,1:20),
"Zscale");
gyro_norm_test = [gyro_norm_test,
featureNormalize2(gyro_data_test_rand.gyro_stats(:,21:end), "Zscale")];
gyro_train_range = gyro_data_train_rand.gyro_range;
gyro_test_range = gyro_data_test_rand.gyro_range;
%bluetooth
bt_norm_train = featureNormalize2(bt_data_train_rand.bt_stats(:,1:23), "Zscale");
%(:,1:23)
bt_norm_test = featureNormalize2(bt_data_test_rand.bt_stats(:,1:23), "Zscale");
%(:,1:23)

bt_norm_train = ( [bt_norm_train,
featureNormalize2(bt_data_train_rand.bt_stats(:,24:end), "Zscale")]);
bt_norm_test = ( [bt_norm_test,
featureNormalize2(bt_data_test_rand.bt_stats(:,24:end), "Zscale")]);
```

```
bt_train_range = bt_data_train_rand.bt_range;
bt_test_range = bt_data_test_rand.bt_range;
%-------------------------------------------------------------%
%regrassion on range
%range train
t2 = templateTree('MinLeafSize', 1);
Mdl_acc_reg = fitrensemble(acc_norm_train, acc_train_range, ...
   'Method','Bag','NumLearningCycles', 265,'Learners',t2 );

Mdl_gyro_reg = fitrensemble(gyro_norm_train, gyro_train_range, ...
   'Method','Bag','NumLearningCycles',265,'Learners',t2);

Mdl_bt_reg = fitrtree(bt_norm_train, bt_train_range,'MinLeafSize', 7, ...
   'Surrogate', 'off'); %, 'PredictorNames', predNamesBT
%-------------------------------------------------------------%
%Sensor fusion
%prepare training data
[ef_data_r, lbl_r] = prep_ef_data_reg2(acc_norm_train, gyro_norm_train,...
   bt_norm_train, acc_train_range, gyro_train_range, bt_train_range);
Mdl_ef = fitrtree(ef_data_r, lbl_r, 'MinLeafSize', 11, ...
   'Surrogate', 'off');
%-------------------------------------------------------------%
%Fusion 2
acc_nn = ef_data_r(:,1:25); %with labels (:,1:37) ///without labels (:,1:20)
gyro_nn = ef_data_r(:,26:50); %with labels (:,38:74) ///without labels (:,21:40)
bt_nn = ef_data_r(:,51:end); %with labels (:,75:end) ///without labels(:,41:end)
%first layer
```

```
Mdl_acc_nn = fitrensemble(acc_nn, lbl_r, 'Method','Bag',...
    'NumLearningCycles',225,'Learners',t2);
Mdl_gyro_nn = fitrensemble(gyro_nn, lbl_r, 'Method','Bag',...
    'NumLearningCycles',225,'Learners',t2);
Mdl_bt_nn = fitrtree(bt_nn, lbl_r,'MinLeafSize', 9, ...
    'Surrogate', 'off');

acc_nn_l1 = predict(Mdl_acc_nn, acc_nn);
gyro_nn_l1 = predict(Mdl_gyro_nn, gyro_nn);
bt_nn_l1 = predict(Mdl_bt_nn, bt_nn);
%layer 2 (output layer)
nn_dat_l2 = [acc_nn_l1 gyro_nn_l1 bt_nn_l1];
nn_dat_l2_norm = featureNormalize2(nn_dat_l2, "Zscale"); %try without
normalization
Mdl_nn_l2 = fitrensemble(nn_dat_l2, lbl_r, 'Method','Bag',...
    'NumLearningCycles',30,'Learners',t2);
%-----------------------------------------------------------------%
%test
fitted_acc = predict(Mdl_acc_reg, acc_norm_test);
fitted_gyro = predict(Mdl_gyro_reg, gyro_norm_test);
fitted_bt = predict(Mdl_bt_reg, bt_norm_test);
%-----------------------------------------------------------------%
%Sensor fusion test
[ef_data_test, lbl_test] = prep_ef_data_reg3(acc_norm_test, gyro_norm_test,...
    bt_norm_test, acc_test_range, gyro_test_range, bt_test_range);
prediction_ef = predict(Mdl_ef, ef_data_test);
%-----------------------------------------------------------------%
```

```
%Fusion 2 test
acc_nn_test = ef_data_test(:,1:25); %with labels (:,1:37) ///without labels (:,1:20)
gyro_nn_test = ef_data_test(:,26:50); %with labels (:,38:74) ///without labels
)(:,21:40)
bt_nn_test = ef_data_test(:,51:end); %with labels (:,75:end) ///without
labels(:,41:end)
acc_nn_l1_test = predict(Mdl_acc_nn, acc_nn_test);
gyro_nn_l1_test = predict(Mdl_gyro_nn, gyro_nn_test);
bt_nn_l1_test = predict(Mdl_bt_nn, bt_nn_test);
%layer 2 (output layer)
nn_dat_l2_test = [acc_nn_l1_test gyro_nn_l1_test bt_nn_l1_test];
nn_dat_l2_test_norm = featureNormalize2(nn_dat_l2_test, "Zscale");
prediction_nn = predict(Mdl_nn_l2, nn_dat_l2_test_norm);
%--------------------------------------------------------------%
%fit error/performance metrics
[mse_acc, rmse_acc, r2_acc, mae_acc] = fit_error(acc_test_range, fitted_acc);
[mse_gyro, rmse_gyro, r2_gyro, mae_gyro] = fit_error(gyro_test_range,
fitted_gyro);
[mse_bt, rmse_bt, r2_bt, mae_bt] = fit_error(bt_test_range, fitted_bt);
[mse_ef, rmse_ef, r2_ef, mae_ef] = fit_error(lbl_test, prediction_ef);
[mse_nn, rmse_nn, r2_nn, mae_nn] = fit_error(lbl_test, prediction_nn);
%safe regression results
methods = ["MSE" "RMSE" "R2" "MAE"];
results_reg = table(methods', [mse_acc rmse_acc r2_acc mae_acc]',...
    [mse_gyro rmse_gyro r2_gyro mae_gyro]',...
    [mse_bt rmse_bt r2_bt mae_bt]', ...
    [mse_ef rmse_ef r2_ef mae_ef]', ...
    [mse_nn rmse_nn r2_nn mae_nn]');
%accuracy of predicting <6 feet
[F1_acc, BA_acc] = scores((acc_test_range >= 6),(fitted_acc >= 6));
[F1_gyro, BA_gyro] = scores((gyro_test_range >= 6),(fitted_gyro >= 6));
[F1_bt, BA_bt] = scores((bt_test_range >= 6),(fitted_bt >= 6));
%save <6 feet classification results
b_methods = ["F1" "BA"];
results_b = table(b_methods', [F1_acc BA_acc]', [F1_gyro BA_gyro]',...
    [F1_bt BA_bt]','VariableNames',{'Metric' 'Accelerometer' 'Gyroscope'
'Bluetooth'});
writetable(results_b, 'results_pact_LOO_6feet_user17.xlsx');
%-----------------------------------------------------------------%
```

B.5 Support functions

```
%-----------------------------------------------------------------%
%Function to normalize features
%Inputs: X - feature vector/matrix to normalize
%       method - normalization method ('Zscore','MinMax', or 'Log')
%Output: X_norm - normalized feature vector/matrix
%-----------------------------------------------------------------%
function X_norm = featureNormalize2(X, method)

epsilon = 0.0001; %to avoid division by 0
if method == "Zscale"
    temp = bsxfun(@minus, X, mean(X));
    X_norm = bsxfun(@rdivide, temp, (std(X)+epsilon));
elseif method == "MinMax"
```

```
     X_norm = (X - min(X))./(max(X) - min(X));
elseif method == "Log"
    X_norm = log10(X);
else
    disp('No scaling method selected')
end
end
%---------------------------------------------------------------------%
% Function that creates 5 groups from user context data and prepares it for
% sensor fusion model
% Inputs: training and test data
% Outputs: training, test data, and labels spareratly
%---------------------------------------------------------------------%
function [acc_lbl_train, acc_lbl_test, gyro_lbl_train, gyro_lbl_test, ...
    bt_lbl_train, bt_lbl_test, all_cnt]= group_cat(acc_dat_test, ...
    acc_dat_train, gyro_dat_test, gyro_dat_train, bt_dat_test, bt_dat_train)
%accelerometer
%---------------------------------------------------------------%
%training set
acc_cat_train = acc_dat_train.acc_categories;
%Inside or outside
acc_env1_train = acc_cat_train(:,1)+(acc_cat_train(:,2).*2);
%Size of the room
acc_env2_train = acc_cat_train(:,3) + acc_cat_train(:,4).*2 + acc_cat_train(:,5).*3;
%Where in the room
acc_env3_train = acc_cat_train(:,6) + acc_cat_train(:,7).*2 + acc_cat_train(:,8).*3
+ acc_cat_train(:,9).*4 ;
```

```
%Sitting or Standing
acc_pose_train = acc_cat_train(:,10)+(acc_cat_train(:,11).*2);
%Phone location on user body
acc_loc_train = acc_cat_train(:,12) + acc_cat_train(:,13).*2 +...
    acc_cat_train(:,14).*3 + acc_cat_train(:,15).*4 + acc_cat_train(:,16).*5 +
acc_cat_train(:,17).*6 ;
%test set
acc_cat_test = acc_dat_test.acc_categories;
acc_env1_test = acc_cat_test(:,1)+(acc_cat_test(:,2).*2);
acc_env2_test = acc_cat_test(:,3) + acc_cat_test(:,4).*2 + acc_cat_test(:,5).*3;
acc_env3_test = acc_cat_test(:,6) + acc_cat_test(:,7).*2 + acc_cat_test(:,8).*3 +
acc_cat_test(:,9).*4 ;
acc_pose_test = acc_cat_test(:,10)+(acc_cat_test(:,11).*2);
acc_loc_test = acc_cat_test(:,12) + acc_cat_test(:,13).*2 +...
    acc_cat_test(:,14).*3 + acc_cat_test(:,15).*4 + acc_cat_test(:,16).*5 +
acc_cat_test(:,17).*6 ;

acc_lbl_train = [acc_env1_train, acc_env2_train, acc_env3_train, ...
    acc_pose_train, acc_loc_train];
acc_lbl_test = [acc_env1_test, acc_env2_test, acc_env3_test, ...
    acc_pose_test, acc_loc_test];
%------------------------------------------------------------%
%gyroscope
%------------------------------------------------------------%
gyro_cat_train = gyro_dat_train.gyro_categories;
gyro_env1_train = gyro_cat_train(:,1) + gyro_cat_train(:,2).*2;
```

```
gyro_env2_train = gyro_cat_train(:,3) + gyro_cat_train(:,4).*2 +
gyro_cat_train(:,5).*3;
gyro_env3_train = gyro_cat_train(:,6) + gyro_cat_train(:,7).*2 +
gyro_cat_train(:,8).*3 + gyro_cat_train(:,9).*4 ;
gyro_pose_train = gyro_cat_train(:,10)+ gyro_cat_train(:,11).*2;
gyro_loc_train = gyro_cat_train(:,12) + gyro_cat_train(:,13).*2 +...
    gyro_cat_train(:,14).*3 + gyro_cat_train(:,15).*4 + gyro_cat_train(:,16).*5 +
gyro_cat_train(:,17).*6 ;

gyro_cat_test = gyro_dat_test.gyro_categories;
gyro_env1_test = gyro_cat_test(:,1) + gyro_cat_test(:,2).*2;
gyro_env2_test = gyro_cat_test(:,3) + gyro_cat_test(:,4).*2 + gyro_cat_test(:,5).*3;
gyro_env3_test = gyro_cat_test(:,6) + gyro_cat_test(:,7).*2 + gyro_cat_test(:,8).*3
+ gyro_cat_test(:,9).*4 ;
gyro_pose_test = gyro_cat_test(:,10)+ gyro_cat_test(:,11).*2;
gyro_loc_test = gyro_cat_test(:,12) + gyro_cat_test(:,13).*2 +...
    gyro_cat_test(:,14).*3 + gyro_cat_test(:,15).*4 + gyro_cat_test(:,16).*5 +
gyro_cat_test(:,17).*6 ;

gyro_lbl_train = [gyro_env1_train, gyro_env2_train, gyro_env3_train, ...
    gyro_pose_train, gyro_loc_train];
gyro_lbl_test = [gyro_env1_test, gyro_env2_test, gyro_env3_test, ...
    gyro_pose_test, gyro_loc_test];
%------------------------------------------------------------%
%Bluetooth
%------------------------------------------------------------%
bt_cat_train = bt_dat_train.bt_categories;
```

```
bt_env1_train = bt_cat_train(:,1) + bt_cat_train(:,2).*2;
bt_env2_train = bt_cat_train(:,3) + bt_cat_train(:,4).*2 + bt_cat_train(:,5).*3;
bt_env3_train = bt_cat_train(:,6) + bt_cat_train(:,7).*2 + bt_cat_train(:,8).*3 +
bt_cat_train(:,9).*4 ;
bt_pose_train = bt_cat_train(:,10)+ bt_cat_train(:,11).*2;
bt_loc_train = bt_cat_train(:,12) + bt_cat_train(:,13).*2 +...
    bt_cat_train(:,14).*3 + bt_cat_train(:,15).*4 + bt_cat_train(:,16).*5 +
bt_cat_train(:,17).*6 ;

bt_cat_test = bt_dat_test.bt_categories;
bt_env1_test = bt_cat_test(:,1) + bt_cat_test(:,2).*2;
bt_env2_test = bt_cat_test(:,3) + bt_cat_test(:,4).*2 + bt_cat_test(:,5).*3;
bt_env3_test = bt_cat_test(:,6) + bt_cat_test(:,7).*2 + bt_cat_test(:,8).*3 +
bt_cat_test(:,9).*4 ;
bt_pose_test = bt_cat_test(:,10)+ bt_cat_test(:,11).*2;
bt_loc_test = bt_cat_test(:,12) + bt_cat_test(:,13).*2 +...
    bt_cat_test(:,14).*3 + bt_cat_test(:,15).*4 + bt_cat_test(:,16).*5 +
bt_cat_test(:,17).*6 ;

bt_lbl_train = [bt_env1_train, bt_env2_train, bt_env3_train, ...
    bt_pose_train, bt_loc_train];
bt_lbl_test = [bt_env1_test, bt_env2_test, bt_env3_test, ...
    bt_pose_test, bt_loc_test];
%-----------------------------------------------------------------%
%calculate number of positive samples
acc_test_cnt = sum(acc_cat_test);
acc_train_cnt = sum(acc_cat_train);
```

```
gyro_test_cnt = sum(gyro_cat_test);
gyro_train_cnt = sum(gyro_cat_train);

bt_test_cnt = sum(bt_cat_test);
bt_train_cnt = sum(bt_cat_train);

all_cnt = [acc_test_cnt', acc_train_cnt', gyro_test_cnt', gyro_train_cnt', ...
    bt_test_cnt', bt_train_cnt'];
end
%-----------------------------------------------------------%


%-----------------------------------------------------------------%
% Function to compute cost of misclassification for multiclass
% classification
% Input: class vector
% Output: cost matrix
%-----------------------------------------------------------------%
function cost = compute_cost_multi(array_classes,c)
%classes
all_classes = [0 1 2 0 0 0 0;
    0 1 2 3 0 0 0;
    0 1 2 3 4 0 0;
    0 1 2 0 0 0 0;
    0 1 2 3 4 5 6;];
%unique classes
```

```
u1 = unique(array_classes);
    i1 = ~eye(length(u1));
    for z = 1:length(u1)
        u2(z) = sum(array_classes==u1(z));
        if u2(z) == 0
            u2(z)=1;
        end
    end
    cost = [];
    for z = 1:length(u1)
        cost = [cost;u2];
    end
    %misclassification matrix
    cost=(cost).*i1;
    u2=[];
end
%-------------------------------------------------------------------%

%-------------------------------------------------------------------%
% Function to compute F1 score and blanced accuracy
% Input: ground truth vector and predicted class vector
% Output: F1 score and balanced accuracy
%-------------------------------------------------------------------%
function [F1_all , BA_all] = scores_multi(groundTruth, prediction)
%classes
all_classes = [0 1 2 0 0 0 0;
    0 1 2 3 0 0 0;
    0 1 2 3 4 0 0;
    0 1 2 0 0 0 0;
    0 1 2 3 4 5 6;];

epsilon = 0.0001; %to avoid division by 0

[n,~] = size(all_classes); %n-number of groups
F1_all = [];
BA_all = [];
for c=1:n
    u = unique(all_classes(c,:)); %number of test settings (lables) per group
    for i=2:length(u)
        [TP, FP, TN, FN] = calError(prediction(:,c) == u(i), groundTruth(:,c) == u(i));
        TPR = TP./(TP+FN+epsilon);
        TNR = TN./(TN+FP+epsilon);
        prec = TP./(TP+FP+epsilon);
        BA(i-1) = (TPR+TNR)./2;
        F1(i-1) = (2.*TPR.*prec)./(TPR+prec+epsilon);
    end
    F1_all = [F1_all; F1'];
    BA_all = [BA_all; BA'];
    BA = [];
    F1 = [];
end

end
%-------------------------------------------------------------------%
```

```
%-------------------------------------------------------------------%
% Function to concatinate horizontaly data of different length (vertical length)
% from three sensors
% Input: each sensor data and range vectors
% Output: concatinated data and range vector
%-------------------------------------------------------------------%
function [ef_dat, lbls] = prep_ef_data_reg2(acc_x, gyro_x, bt_x, ...
    acc_y, gyro_y, bt_y )
%get all unique distanses (expected 2:2:16)
dist = unique(acc_y);

if length(dist)~=8
    disp('Distance range is not what expected');
end
%find data point associated with each distance
acc_dist2 = acc_x(find(acc_y==dist(1)),:);
acc_dist4 = acc_x(find(acc_y==dist(2)),:);
acc_dist6 = acc_x(find(acc_y==dist(3)),:);
acc_dist8 = acc_x(find(acc_y==dist(4)),:);
acc_dist10 = acc_x(find(acc_y==dist(5)),:);
acc_dist12 = acc_x(find(acc_y==dist(6)),:);
acc_dist14 = acc_x(find(acc_y==dist(7)),:);
acc_dist16 = acc_x(find(acc_y==dist(8)),:);

gyro_dist2 = gyro_x(find(gyro_y==dist(1)),:);
gyro_dist4 = gyro_x(find(gyro_y==dist(2)),:);
gyro_dist6 = gyro_x(find(gyro_y==dist(3)),:);
```

```
gyro_dist8 = gyro_x(find(gyro_y==dist(4)),:);
gyro_dist10 = gyro_x(find(gyro_y==dist(5)),:);
gyro_dist12 = gyro_x(find(gyro_y==dist(6)),:);
gyro_dist14 = gyro_x(find(gyro_y==dist(7)),:);
gyro_dist16 = gyro_x(find(gyro_y==dist(8)),:);

bt_dist2 = bt_x(find(bt_y==dist(1)),:);
bt_dist4 = bt_x(find(bt_y==dist(2)),:);
bt_dist6 = bt_x(find(bt_y==dist(3)),:);
bt_dist8 = bt_x(find(bt_y==dist(4)),:);
bt_dist10 = bt_x(find(bt_y==dist(5)),:);
bt_dist12 = bt_x(find(bt_y==dist(6)),:);
bt_dist14 = bt_x(find(bt_y==dist(7)),:);
bt_dist16 = bt_x(find(bt_y==dist(8)),:);
%concatenate using the shortest vector those cropping vectors for the
%sensors that have more data for given distance
min2 = min([length(acc_dist2), length(gyro_dist2), length(bt_dist2)]);
min4 = min([length(acc_dist4), length(gyro_dist4), length(bt_dist4)]);
min6 = min([length(acc_dist6), length(gyro_dist6), length(bt_dist6)]);
min8 = min([length(acc_dist8), length(gyro_dist8), length(bt_dist8)]);
min10 = min([length(acc_dist10), length(gyro_dist10), length(bt_dist10)]);
min12 = min([length(acc_dist12), length(gyro_dist12), length(bt_dist12)]);
min14 = min([length(acc_dist14), length(gyro_dist14), length(bt_dist14)]);
min16 = min([length(acc_dist16), length(gyro_dist16), length(bt_dist16)]);

dist2 = [acc_dist2(1:min2,:) gyro_dist2(1:min2,:) bt_dist2(1:min2,:)
ones(min2,1)*2];
```

```
dist4 = [acc_dist4(1:min4,:) gyro_dist4(1:min4,:) bt_dist4(1:min4,:)
ones(min4,1)*4];
dist6 = [acc_dist6(1:min6,:) gyro_dist6(1:min6,:) bt_dist6(1:min6,:)
ones(min6,1)*6];
dist8 = [acc_dist8(1:min8,:) gyro_dist8(1:min8,:) bt_dist8(1:min8,:)
ones(min8,1)*8];
dist10 = [acc_dist10(1:min10,:) gyro_dist10(1:min10,:) bt_dist10(1:min10,:)
ones(min10,1)*10];
dist12 = [acc_dist12(1:min12,:) gyro_dist12(1:min12,:) bt_dist12(1:min12,:)
ones(min12,1)*12];
dist14 = [acc_dist14(1:min14,:) gyro_dist14(1:min14,:) bt_dist14(1:min14,:)
ones(min14,1)*14];
dist16 = [acc_dist16(1:min16,:) gyro_dist16(1:min16,:) bt_dist16(1:min16,:)
ones(min16,1)*16];
%combine data for each distance and randomly shuffle it
dat_all = [dist2; dist4; dist6; dist8; dist10; dist12; dist14; dist16];
[h,~] = size(dat_all);
idx = randperm(h);
data_rand = dat_all(idx,:);
%split data and range vector
ef_dat = data_rand(:,1:end-1);
lbls = data_rand(:,end);
end
%------------------------------------------------------------------%


%------------------------------------------------------------------%
```

```
% Function to concatinate horizontaly data of different length (vertical length)
% from three sensors
% Input: each sensor data and range vectors
% Output: concatinated data and range vector
%--------------------------------------------------------------------%
function [ef_dat, lbls] = prep_ef_data_reg3(acc_x, gyro_x, bt_x, ...
    acc_y, gyro_y, bt_y )
dist = unique(acc_y); %get all unique distanses (expected 2:2:16)
dat_all = [];
for i=1:length(dist)
   %find data point associated with each distance
   acc_dist = acc_x(find(acc_y==dist(i)),:);
   gyro_dist = gyro_x(find(gyro_y==dist(i)),:);
   bt_dist = bt_x(find(bt_y==dist(i)),:);
   %concatenate using the shortest vector those cropping vectors for the
   %sensors that have more data for given distanc
   min_d = min([length(acc_dist(:,1)), length(gyro_dist(:,1)), length(bt_dist(:,1))]);
   dist_m = [acc_dist(1:min_d,:) gyro_dist(1:min_d,:) bt_dist(1:min_d,:)
ones(min_d,1)*dist(i)];
   dat_all = [dat_all; dist_m];
end
%combine data for each distance and randomly shuffle it
[h,~] = size(dat_all);
idx = randperm(h);
data_rand = dat_all(idx,:);
%split data and range vector
ef_dat = data_rand(:,1:end-1);
```

```
lbls = data_rand(:,end);
end
%-------------------------------------------------------------------%


%-------------------------------------------------------------------%
% Function to compute RMSE, R^2, MSE, MAE
% Input: predicted range and ground truth
% Output: MSE, RMSE, R^2, MAE
%-------------------------------------------------------------------%
function [mse, rmse, r2, mae] = fit_error(tru, fitted)
s = sum((tru-fitted).^2);
n = length(tru);

mse = s/n; %MSE

rmse = sqrt(s/n);%RMSE

r2 = 1 - (s/sum((tru-mean(tru)).^2)); %R^2

mae = sum(abs(fitted-tru))/n; %MAE

end
%-------------------------------------------------------------------%


%-------------------------------------------------------------------%
% Function to compute F1 score and balanced accuracy
% Input: predicted range and ground truth
% Output: F1 and balanced accuracy
%-------------------------------------------------------------------%
function [F1,BA] = scores(groundTruth, prediction)
epsilon = 0.0001; %to avoid division by 0
[TP, FP, TN, FN] = calError(groundTruth,prediction);
TPR = TP./(TP+FN+epsilon);
TNR = TN./(TN+FP+epsilon);
prec = TP./(TP+FP+epsilon);
BA = (TPR+TNR)./2; %Balanced accuracy
F1 = (2.*TPR.*prec)./(TPR+prec+epsilon); %F1 score
end
%-------------------------------------------------------------------%


%-------------------------------------------------------------------%
% This function calculates True Positives, False Positives, True Negatives
% and False Negatives for two matrices of equal size assuming they are
% populated by 1's and 0's.
% Inputs: trueMat contains the actual true values while the predictedMat
% contains the 1's and 0's predicted from the algorithm used.
% Output: confucion matrix
%-------------------------------------------------------------------%
function [TP, FP, TN, FN] = calError(trueMat, predictedMat)

adder = trueMat + predictedMat;
TP = length(find(adder == 2));
```

```
TN = length(find(adder == 0));
subtr = trueMat - predictedMat;
FP = length(find(subtr == -1));
FN = length(find(subtr == 1));
end
%-------------------------------------------------------------------%
```

B.6 Functions to calculate features

```
%-----------------------------------------------------------------%
% Function to compute mean
% Input: X - data vector
%            n - window over which mean computed
%-----------------------------------------------------------------%
function Y = Fmean(X,n)
   Y = arrayfun(@(i) mean(X(i:i+n-1)),1:n:length(X)-n+1)';
End

%-----------------------------------------------------------------%
% Function to compute standard deviation
% Input: X - data vector
%        n - window over which standard deviation computed
%-----------------------------------------------------------------%
function Y = Fstd(X,n)
   Y = arrayfun(@(i) std(X(i:i+n-1)),1:n:length(X)-n+1)';
end


%-----------------------------------------------------------------%
```

```
% Function to compute skewness and kurtosis
% Input: X - data vector
%        n - window over which skewness and kurtosis computed
%-----------------------------------------------------------------%
function [M3, M4] = Fmoment(X,n)
%calculate third moment - skewness and fourth moment - kurtosis
   M3 = arrayfun(@(i) moment(X(i:i+n-1),3),1:n:length(X)-n+1)';
   M4 = arrayfun(@(i) moment(X(i:i+n-1),4),1:n:length(X)-n+1)';
end

%-----------------------------------------------------------------%
% Function to compute 25th, 50th, and 75th percentile
% Input: X - data vector
%        n - window over which statistic calculation performed
%-----------------------------------------------------------------%
function [P25, P50, P75] = Fprctile(X,n)
%calculate 25th, 50th, 75th percentile
   P25 = arrayfun(@(i) prctile(X(i:i+n-1),25),1:n:length(X)-n+1)';
   P50 = arrayfun(@(i) prctile(X(i:i+n-1),50),1:n:length(X)-n+1)';
   P75 = arrayfun(@(i) prctile(X(i:i+n-1),75),1:n:length(X)-n+1)';
end

%-----------------------------------------------------------------%
% Function to value entropy and time entropy
% Input: X - data vector
%        n - window over which statistic calculation performed
%-----------------------------------------------------------------%
```

```
function [ValE, TimeE] = Fentropy(X,n)
    %value entropy
    val = arrayfun(@(i) hist(X(i:i+n-1),20),1:n:length(X)-
n+1,'UniformOutput',false)';
    ValE = zeros(length(val),1);
    for i = 1:length(val)
        ValE(i) = entropy(val{i})';
    end
    TimeE = arrayfun(@(i) timeE(X(i:i+n-1)),1:n:length(X)-n+1)';
end

function te = timeE(X)
    %time entropy
    normX = X./max(abs(X));
    te = entropy(normX);
end


%-----------------------------------------------------------------%
% Function to compute autocovariance and autocorrelation
% Input: X - data vector
%        n - window over which statistic calculation performed
%-----------------------------------------------------------------%
function [Acov, Acor] = Fauto(X,n)
    Acor = arrayfun(@(i) autoCor(X(i:i+n-1),n),1:n:length(X)-n+1)';
    Acov = arrayfun(@(i) autoCov(X(i:i+n-1),n),1:n:length(X)-n+1)';
end
```

```
%autocorrelation
function Y = autoCor(X,n)
    z = circshift(X,floor(n/2));
    Y=X'*z;
end
%autocovariance
function Y = autoCov(X,n)
    z = circshift(X,floor(n/2));
    Y = ((X-mean(X))'*(z-mean(z)))/(std(X)*std(z));
end

%------------------------------------------------------------------%
% Function to compute correlation between axes
% Input: X - data vector
%        n - window over which statistic calculation performed
%------------------------------------------------------------------%
function cor = Fcorr(x, y, z, n)
%computes correlation between three axis
xy = arrayfun(@(i) corr(x(i:i+n-1), y(i:i+n-1),  'Type', 'Pearson'),1:n:length(x)-
n+1)';
xz = arrayfun(@(i) corr(x(i:i+n-1), z(i:i+n-1),  'Type', 'Pearson'),1:n:length(x)-
n+1)';
yz = arrayfun(@(i) corr(y(i:i+n-1), z(i:i+n-1),  'Type', 'Pearson'),1:n:length(x)-
n+1)';
cor = [xy, xz, yz];
end
```

```
%-----------------------------------------------------------------%
% Function to compute RMS of Doppler spectrum
% Input: X - data vector
%        n - window over which output parameter calculated
%-----------------------------------------------------------------%
function Y = FRmsDoppler(X,n)
   Y = arrayfun(@(i) doppler(X(i:i+n-1)),1:n:length(X)-n+1)';
end

function  B_rms= doppler(rssi_1)
w0=-pi:pi/1000:pi;
X0=zeros(1,length(w0));
j0=length(rssi_1);
for i=1:length(w0)
   for j=1:j0
   X0(i)=X0(i)+sqrt(rssi_1(j)).*exp(-1i*w0(i)*j); % Discrete Time Fourier
Transform
   end
end
P_doppler = 10*log10(abs(X0).^2);

N_slice = size(rssi_1,1);

f = w0*N_slice/2/pi;
B_rms = sqrt(sum(f.^2.* P_doppler)/sum(P_doppler));

end
```

```
%------------------------------------------------------------------%
% Function to compute peak-to-peak signal change
% Input: X - data vector
%        n - window over which output parameter calculated
%------------------------------------------------------------------%
function Y = Yp2p(X,n)
    Y = arrayfun(@(i) (max(X(i:i+n-1)) - min(X(i:i+n-1))),1:n:length(X)-n+1)';
end

%------------------------------------------------------------------%
% Function to compute signal energy
% Input: X - data vector
%        n - window over which output parameter calculated
%------------------------------------------------------------------%
function Y = Fenergy(X,n)
    Y = arrayfun(@(i) doppler(X(i:i+n-1)),1:n:length(X)-n+1)';
end
function  energy= doppler(rssi_1)
w0=-pi:pi/1000:pi;
X0=zeros(1,length(w0));
j0=length(rssi_1);
for i=1:length(w0)
    for j=1:j0
    X0(i)=X0(i)+sqrt(rssi_1(j)).*exp(-1i*w0(i)*j); % Discrete Time Fourier
Transform
    end
```

```
end
P_doppler = 10*log10(abs(X0).^2);
dopler_size = size(P_doppler,2);
energy = sum(P_doppler)/dopler_size; %signal energy
end

%------------------------------------------------------------------%
% Function to compute Rayleigh parameter
% Input: X - data vector
%        n - window over which output parameter calculated
%------------------------------------------------------------------%
function Y = Frayleigh(X,n)
   Y = arrayfun(@(i) raylfit(X(i:i+n-1)),1:n:length(X)-n+1)';
end


%------------------------------------------------------------------%
% Function to compute fade duration
% Input: X - data vector
%        n - window over which output parameter calculated
%------------------------------------------------------------------%
function Y = FfadeDuration(X,n)
   Y = arrayfun(@(i) (exp(ro(X(i:i+n-1))^2)-1)/(sqrt(2*pi)*ro(X(i:i+n-
1))*doppler(X(i:i+n-1))),...
       1:n:length(X)-n+1)';
end
```

```
function y = ro(X)
   %a = median(X);
   arms=rms(X);
   a = rms(X) - 3; %RMS - 3dB (try different values e.g 3, 2, 1 dB)
   y=(a)/(arms);
end
function  B_rms= doppler(rssi_1)
w0=-pi:pi/1000:pi;
X0=zeros(1,length(w0));
j0=length(rssi_1);
for i=1:length(w0)
   for j=1:j0
   X0(i)=X0(i)+sqrt(rssi_1(j)).*exp(-1i*w0(i)*j); % Discrete Time Fourier
Transform
   end
end
P_doppler = 10*log10(abs(X0).^2);

N_slice = size(rssi_1,1);

f = w0*N_slice/2/pi;
B_rms = sqrt(sum(f.^2.* P_doppler)/sum(P_doppler));

end

%-------------------------------------------------------------------%
% Function to compute level crossing
% Input: X - data vector
%        n - window over which output parameter calculated
%-------------------------------------------------------------------%
function Y = FlevelX(X,n)
    Y = arrayfun(@(i) sqrt(2*pi)*ro(X(i:i+n-1))*doppler(X(i:i+n-1))*exp(-
ro(X(i:i+n-1))^2),...
        1:n:length(X)-n+1)';
end

function y= ro(X)
    %a = median(X);
    arms=rms(X);
    a = arms - 3;
    y=sum(a)/sum(arms);
end

function  B_rms= doppler(rssi_1)
w0=-pi:pi/1000:pi;
X0=zeros(1,length(w0));
j0=length(rssi_1);
for i=1:length(w0)
    for j=1:j0
    X0(i)=X0(i)+sqrt(rssi_1(j)).*exp(-1i*w0(i)*j); % Discrete Time Fourier
Transform
    end
end
P_doppler = 10*log10(abs(X0).^2);
```

```
N_slice = size(rssi_1,1);

f = w0*N_slice/2/pi;
B_rms = sqrt(sum(f.^2.* P_doppler)/sum(P_doppler));

end

%-----------------------------------------------------------------%
% Function to compute peak of Doppler spectrum
% Input: X - data vector
%        n - window over which output parameter calculated
%-----------------------------------------------------------------%
function Y = FpeakDoppler(X,n)
   Y = arrayfun(@(i) doppler(X(i:i+n-1)),1:n:length(X)-n+1)';
end

function  P_max= doppler(rssi_1)
w0=-pi:pi/1000:pi;
X0=zeros(1,length(w0));
j0=length(rssi_1);
for i=1:length(w0)
   for j=1:j0
   X0(i)=X0(i)+sqrt(rssi_1(j)).*exp(-1i*w0(i)*j); % Discrete Time Fourier
Transform
   end
end
```

```
P_doppler = 10*log10(abs(X0).^2);
P_max = max(P_doppler);
end

%-----------------------------------------------------------------------%
% Function to compute mean of Doppler spectrum
% Input: X - data vector
%        n - window over which output parameter calculated
%-----------------------------------------------------------------------%
function Y = FmeanDoppler(X,n)
    Y = arrayfun(@(i) doppler(X(i:i+n-1)),1:n:length(X)-n+1)';
end

function  P_mean= doppler(rssi_1)
w0=-pi:pi/1000:pi;
X0=zeros(1,length(w0));
j0=length(rssi_1);
for i=1:length(w0)
    for j=1:j0
    X0(i)=X0(i)+sqrt(rssi_1(j)).*exp(-1i*w0(i)*j); % Discrete Time Fourier
Transform
    end
end
P_doppler = 10*log10(abs(X0).^2);
P_mean = mean(P_doppler);
end
```

```
%-----------------------------------------------------------------%
% Function to compute polynomial fit
% Input: X - data vector
%        n - window over which output parameter calculated
%-----------------------------------------------------------------%
function Y = Fpoly(X,n)
%     a=1;
%     b=0.7;
    Y = arrayfun(@(i) lap_fit(X(i:i+n-1)),1:n:length(X)-n+1)';
end
function lbf = lap_fit(X)

w0=-pi:2*pi/(length(X)-1):pi;
ydata = X;
xdata = w0';
opts = optimset('Display','off');
fun = @(x, xdata) x(1)*xdata.^3;
x0 = 0.1;
fit_param = lsqcurvefit(fun,x0,xdata,ydata, [],[],opts); %
lbf = real(fit_param(1));
end


%-----------------------------------------------------------------%
% Function to compute Laplacian best fit
% Input: X - data vector
%        n - window over which output parameter calculated
```

```
%-----------------------------------------------------------------%
function Y = Flaplacian(X,n)
   Y = arrayfun(@(i) lap_fit(X(i:i+n-1)),1:n:length(X)-n+1)';
end

function lbf = lap_fit(X)
w0=-pi:2*pi/(length(X)-1):pi;
ydata = X;
xdata = w0';
opts = optimset('Display','off');
fun = @(x, xdata) 1./(((xdata-x(3))./x(1)).^(2*x(2)));
x0 = [0.1,0.1,1e-10];
fit_param = lsqcurvefit(fun,x0,xdata,ydata, [],[],opts); %
lbf = real(fit_param(2));
end

%-----------------------------------------------------------------%
% Function to compute Gaussian best fit
% Input: X - data vector
%        n - window over which output parameter calculated
%-----------------------------------------------------------------%
function Y = Fgauss(X,n)
   Y = arrayfun(@(i) lap_fit(X(i:i+n-1)),1:n:length(X)-n+1)';
end
function lbf = lap_fit(X)
w0=-pi:2*pi/(length(X)-1):pi;
ydata = X;
```

```
xdata = w0';
opts = optimset('Display','off');
fun = @(x, xdata) (x(1)*exp(-(-xdata-x(2)).^2)./(x(3)^2));
x0 = [0.1,0.1,1e-10];
fit_param = lsqcurvefit(fun,x0,xdata,ydata, [],[],opts); %
lbf = real(fit_param(2));
end
```