SenseBody: A Multi-agent Human Activity Recognition System Using Wearable Devices

---

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

Kai Lin

May 2021

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: Kai Lin

This Thesis has been read and approved by the examing committee:

Advisor: Bradford Campbell

Advisor:

Committee Member: Yuan Tian

Committee Member: Tom Fletcher

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

May 2021

Abstract

The exploding of the Internet of Things (IoT) has brought in an enormous number of smart devices into humans' everyday lives. People start to carry more devices on the body to monitor vital signals or increase their lifestyle convenience, for instance, wearing an Apple Watch or an Airpods. These new devices bring new opportunities for accurate body gesture recognition. However, traditional methods that rely on a single device cannot precisely recognize more complex gestures. Cooperating between multiple devices to achieve better accuracy while minimizing computation and communication costs is still a challenge for existing methods. Recognizing complex gestures with various devices is still kind of left blank. This thesis presents SenseBody, a system that utilizes multiple wearable sensors and devices in the different body parts to enable gesture recognition. Our system utilizes machine learning techniques focusing on its easy pairing and energy preserving techniques. We use the current state-of-art machine-learning methods to classify the current dataset and collaborate different body part sensor data to make better classifications. We built a prototype based on COTs on-body wearable devices and evaluated the system's accuracy and energy performance. Our results have shown that by applying the proposed multi-agent framework, we can achieve an average of 98.3% of recognition accuracy on nine everyday activities and achieve more than 98.9% auto-pairing accuracy. From an energy-saving perspective, our result also shows that our system can save more than 90% of the energy consumed in data transmission over traditional methods while only sacrificing 3% of accuracy. Our evaluation has demonstrated great potential for future human activity recognition systems.

Acknowledgements

Two years of my master's journey is too short for me to explore and enjoy life at the University of Virginia, one page of acknowledgment can not express how I am grateful for all people who make this work possible and be there with me during my grad school.

To begin with, I would like to thank my advisor Brad Campbell, who gave me a chance to learn and work with him at Link Lab and meet more labmates. I still remember the first time I was chatting with him about different projects the group was doing and discussed some potential thesis topics in his office, it was a shiny afternoon, and it made me love the research since then. Prof. Brad has given me so many valuable insights and suggestions on my projects. I also want to appreciate my committee members - Prof. Yuan Tian and Prof. Tom Fletcher, who have provided me valuable feedback on this thesis. Further, I want to thank Prof. Sebastian Elbaum and Prof. Yangfeng Ji, whom I have learned a lot from during the coursework.

I would also acknowledge my labmates Wenpeng Wang, Jiechao Gao, and Nabeel Nasir, who have supported me through this project and guide me for the first several months of joining the lab. I am grateful to be at Brad's lab working with them and the rest of the fantastic people in the lab.

Outside of the lab, I am grateful to have all my CS classmates and teammates, Chijung Jung, Yiwen Su, Dexuan Zhang, Anastasia Lalamentik, Zhanhong Tian, and Yufeng Yan, who give me valuable domain knowledge that constantly broadens my horizons.

Besides, I am lucky to have all people from the badminton team, David Xiao, Zihe Ye, Emily Zou, Leo Wang, and Danial Xiao, to give me a chance to walk outside the department of computer science and have a wonderful time in different tournaments and learning time. I am also proud to be part of the VISAS program and become good friends with Ariana Kim, Joyce Lee, Wynter Early, and Ciarra Moses. They have helped me learn the language and the US culture. I also want to thank Dr. Xue Feng at Carina Medical and my co-workers at eBay for giving me the flexibility to finish this thesis while as an employee.

In addition, I want to thank my parents, who have shown their massive support during grad school. People come and go, but they are always on my side, and I can not express how wonderful they are to help me learn, grow and be independent.

Last but definitely not least, I want to express my gratitude to Diyu Zheng. My grad school life will still exist without her, but it will never become my once-in-a-lifetime experience. She has provided many insights and help of this work and offered continued support in my course works, job-seeking, and badminton courts.

Dedication

To my parents, for their continued, selfless, dedicated support, without expecting anything in return.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Body Sensor Networks

As the world of IoT and mobile devices continues to increase, we continue to transform everyday items into digital accessories. Internet of Things is bringing far more connected devices than before. Analysts from government agencies have predicted an estimation of 25 to 50 billion connected devices by the year 2025 [22, 49, 42, 36].

These new digital technologies purport to enhance our quality of life and productivity while also offering the freedom of wireless communications. With more research happened on the wearables and body sensor networks (BSNs) [57, 19], which are a branch of wireless sensor networks(WSNs), wearable products have been coming out to join the BSNs [33]. More researchers have put effort into investigating use cases, devices, and topics on the BSNs. Wearable devices, such as smartwatches, fitness trackers, and earphones, are becoming commonplace in commercial products and research projects. With people regularly wearing multiple devices, full-fledged body sensor networks (BSNs) are no longer impossible. In the recent future, wearables will no longer be limited to smartwatches and fitness trackers. Other products including smart-eyeglasses [61, 27], smart-bandages [40, 39] and smart-shoes [55, 18, 26] are under development in research and will soon join the BSNs.

1.2 Human Activity Recognition

BSNs enable new applications like human activity recognition (HAR). Being able to automatically identify what a user is doing using on-body wearable sensors has many use

cases [34], such as in healthcare [51, 43, 58, 30] to manage and reduce the risk of many diseases including obesity, cardiovascular and diabetes, in entertainment [16, 5] to develop sensor-based games like ring fit adventure and fitness boxing [4], and in fitness [13, 3, 5] to help improve body strength and flexibility.

One key characteristic of human activities is that different activities require different body parts to move. However, many of the current HAR devices using on-body sensors only utilize one body part's sensor data, such as one mobile phone and one smartwatch. The current state-of-art algorithms perform activity recognition using CNN [59, 44], or spatial-temporal attention methods on smart devices such as smartphones and smartwatches [38, 15]. Using a single sense point is sufficient for detecting activities that involve most body parts, for example running or walking. Still, systems with a single sensor fail to accurately classify more specific actions or activities involving only one or more body parts. For exercises such as pull-ups or push-ups, using smartwatch data to classify will fail since the wrist barely moves. Similarly, for activities like sit-ups or weight lifting, a smartphone will not be able to classify the activity since the waist sees little movement. So in certain situations, using only a single sensor to perform activity recognition is insufficient and will result in inaccurate activity recognition. Take Just Dance [4] from Nintendo Switch as an example; its score system will only base on the gestures from a single Joy-Con, which means you can achieve a high score by only waving your hands around as shown in Figure 1.1.

One potential remedy to improve activity recognition for more subtle gestures is to use multiple on-body sense points. Luckily, the increasing number of on-body devices and sensors enables multi-devices coordination. About one-in-five Americans have at least one wearable [1]. The number of connected wearable devices worldwide has exceeded 550 million [7] and wrist-worn wearable have reached 150 million in 2020 [8]. At least 13 types of sensors [33] are present in different kinds of wearable devices. Other than the existing smartwatches and smartphones, more devices now include accelerometer and gyroscope sensors, for example, recent smart earphones [2].

Previous works have shown accuracy improvements with multiple wearable devices and state of the art algorithm implementation [15, 52] with the development of machine learning and deep learning for classification tasks. Accuracy increases as the number of devices increases. However, this leads to more devices to charge, configure and coordinate with higher energy consumption.

Figure 1.1: Nintendo Switch game - Just Dance: Since this game only registers your Joy-Con movements, you could technically just wave your arms around without even moving your whole body.

1.3 Contributions

In this thesis, we focus on the challenge of energy efficiency and pairing overhead. We develop a multi-agent HAR coordination system that lowers the total energy use while maintaining high accuracy. Our key insight is a new strategic duty cycling approach tuned with multi-agent HAR classification output. The energy preserving mainly deals with the sampling frequency and communication frequency for sensors can be as high as 100Hz [34, 14], which would consume an enormous portion of energy when doing communication inside the BSNs. However, if the classification confidence is high enough for the given sampling frequency for the specific device, or the device did not involve a lot in this activity, we can lower the sampling and communication frequency by prioritizing the importance of the devices and only send necessary features instead of raw data to reduce the energy cost.

Second, we also address the overhead of pairing multiple on-body devices. The pairing effort is related to the human effort that each time the new devices are attached to the body, we need to let the new device join the current BSNs, the current state of art techniques

includes NFC pairing [48], smart pairing using Bluetooth technology [41]. However, the BSN has its features because of attaching to the same person. It will follow a similar moving trend, which can facilitate the pairing phase easier with the human's body movements.

The main contributions of this thesis are as follows:

- We propose SenseBody, a system with multi-agent human activity recognition that focuses on easy pairing and energy efficiency using COTs wearable devices.
- We design a new pairing and data transmission framework among wearable devices that minimized pairing and data transmission costs.
- We perform a thorough evaluation of SenseBody, demonstrates the high accuracy (more than 98%) for both HAR tasks and auto-pairing, we can also sacrifice approximately 3% accuracy for 10x less energy consumption.

1.4 Thesis Overview

This thesis is organized as follows.

Chapter 2 describes the related work of this thesis, including sensor-based HAR, vision-based HAR, and energy management in wireless sensor networks. We compared the advantages and disadvantages of different methods. We also emphasize differences compared to this project.

Chapter 3 describes the SenseBody system overview. It shows the scenario of this work, what devices and components are included in the system and how they interact as a whole. Furthermore, it provides the goals of this work that will be accomplished by the rest of the thesis.

Chapter 4 describes how we design and build the SenseBody system, including the energy optimization techniques and machine learning techniques we used. We also run a thorough evaluation of the system to demonstrate the system's accuracy and energy consumption.

Chapter 5 describes the auto-pairing we developed for the SenseBody system. It describes why this application matters in the future development of multi-agent HAR and how we design and implement auto-pairing. We also evaluate the auto-pairing accuracy to demonstrate the great potential of this application.

Chapter 6 and Chapter 7 summarize the work in this thesis, point out limitations and possible future works of this system.

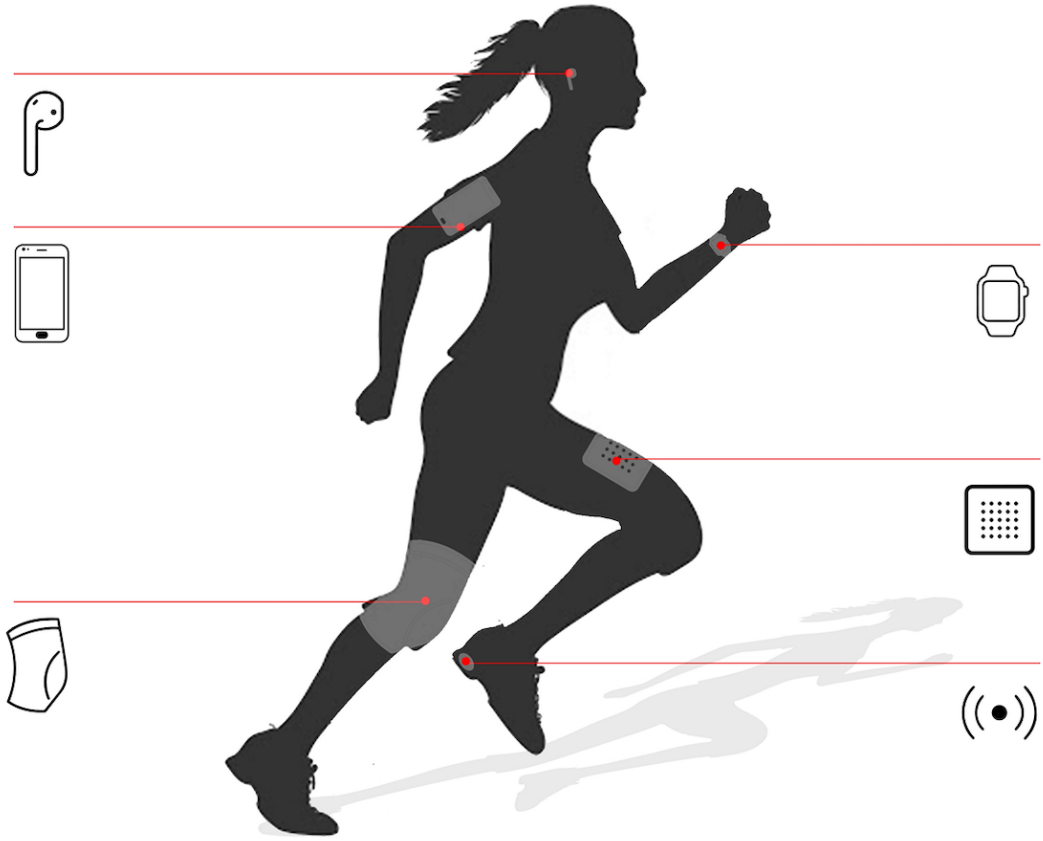

Figure 1.2: A person is running with multiple wearable devices. The trend of IoT is bringing more on-body devices, making it possible to collaborate between them and perform complex tasks that can't be done on a single wearable device in the past.

Chapter 2

Related Work

Our work focuses on the energy and pairing optimization on sensor-based multi-agent HAR using only wearable devices. Some previous work on HAR prioritizes accuracy improvement using one device, while others combine additional environmental sensors with improving accuracy without considering the synchronization overhead and configuration efforts.

2.1 Sensor Based HAR

2.1.1 Single Agent HAR

Triaxial accelerometers and gyroscopes are the most broadly used sensors for HAR [11, 20, 21]. Artificial-neural-nets (ANNs) perform well with up 98% accuracy [28] on recognizing ambulation activities such as walking, running, and lying, and is present in any of today's wearable devices and smartphones. However, other activities such as doing sit-ups or weight lifting are challenging to recognize from a single device point of view [34].

2.1.2 Multi Agent HAR

Multi-agent activity recognition [24, 47, 60] have been done largely based on the public HAR dataset such as Opportunity human activity dataset [14], and MIT PlaceLab dataset [23]. However, these datasets have environmental sensors involved. The Opportunity dataset is specific to the morning activities setting, and the activities defined at MIT PlaceLab dataset are vague, like talking on the phone and searching for items. These

factors will cause the accuracy drop if only involving body sensors. Most importantly, even though these projects tried to accomplish a higher accuracy than a single device, they fail to consider the energy consumption of wearable devices.

2.2 Vision Based HAR

HAR tasks can also be achieved through videos taken by various cameras, majorly based on the computer vision [50] has also evolved in recent years. Methods including optical flow [54, 32], Kalman filtering [46, 12], Hidden Markov models [35, 25], etc. have been applied on several projects under different modalities such as single-camera, stereo, and infrared. In addition, researchers have considered multiple aspects of this topic, including single pedestrian tracking, group tracking, and detecting dropped objects. However, vision-based HAR will consume much more energy than the sensor-based HAR due to camera usage, and the model will be complex due to the high dimension of the computer vision techniques. User privacy is also a major roadblock for applying vision-based HAR tasks.

2.3 Energy Management in WSNs

Energy management in wireless sensor networks and body sensor networks has also been identified and investigated [10, 29, 53, 45]. These projects tried to find the minimum sampling frequency with different sensors while maintaining relatively high accuracy. Our work differs from these projects by using the state-of-art minimum sampling rate and further save energy by dynamically reducing the duty cycle and active devices for the HAR tasks,

There are other methods for performing HAR, with most prior works only focused on algorithm improvements, which is highly affected by the training data [31]. Some additionally require that environmental sensors are involved in reaching high accuracy. This is not entirely practical since the model inside wearable devices cannot include a complex model to handle different environments. Our work overcomes some of these limitations by focusing on on-body sensors only and their optimizations. SenseBody is less dependent on the environment and has better energy performance.

Chapter 3

SenseBody Overview

Our system involves several wearable devices trying to classify both daily activities such as sitting, walking, running and complex training activities, including sit-ups and weight lifting with features calculating from triaxial accelerometers and gyroscopes.

Wearable devices will automatically join the network if they are nearby. And these devices will be separated into two categories, a central device and several auxiliary devices. A central device is usually a smartphone or smartwatch with strong computation power to handle system-level tasks. Auxiliary devices have two statuses in the system: joined (existing auxiliary device) and not joined(new auxiliary device). These devices can be any device shipping the sensors and provide help and better classification for the central device.

The central device includes three main components: the pairing controller, the energy controller and the activity recognition controller. The pairing controller is responsible for letting the new devices automatically pair to the existing SenseBody network. The energy controller handles all policies to save energy in both the central device and auxiliary devices. The activity recognition controller is responsible for collecting features from different devices, applying machine learning models to the data, and classifying activities. It also provides data to the pairing controller and energy controller in the central device.

New auxiliary devices communicate with the existing central device using the pairing controller. The existing auxiliary devices send features to the activity recognition controller in the central device and get feedback from the energy controller. Detailed explanations for these communications will be shown in Chapter 4 and chapter 5.

Project goals Our goal is to develop a system that can coordinate wearable devices and sensors to accurately determine a person's activity in cases where a single device is

insufficient. The approach will also reduce human effort in an easy pairing way and will save the total energy usage for devices.

To achieve this research goal, there are several sub-goals that we completed at each stage of the research:

- Reduce the required sensor data for at least 90% comparing to the traditional multi-agent HAR
- Achieve higher accuracy compared to the single device HAR,
- Hands-free pairing procedure with at least 95% auto-pairing accuracy

Chapter 4

Multi-Agent HAR Coordination in SenseBody

4.1 Design

Our design has three main components as highlighted in Figure 4.1. When a new device is attached to the human body and actively moving, it will communicate with the pairing controller inside the existing central device to make auto-pairing to check if it belongs to the same person. After the network has formed, the central device will coordinate with the existing auxiliary device to make activity recognition within the activity recognition controller. The energy controller will prioritize the device and make a judgment on changing the duty cycle of the existing devices.

4.1.1 Energy optimization design

Energy is one of the critical considerations in developing this system. The battery capacity in the wearable device is limited to its size, so we have come up with methods that can reduce energy consumption. In that way, we can preserve more energy and possibly stop recharge with the development of energy harvesting techniques. We lower the energy consumption in 3 different techniques: the network transmission optimization, which reduces the overhead to transmit the time series signals; the device duty cycle reduction, which reduces the working time for the sensor; and the feature reduction, which reduces the model complexity, benefit for computation and network transmission.

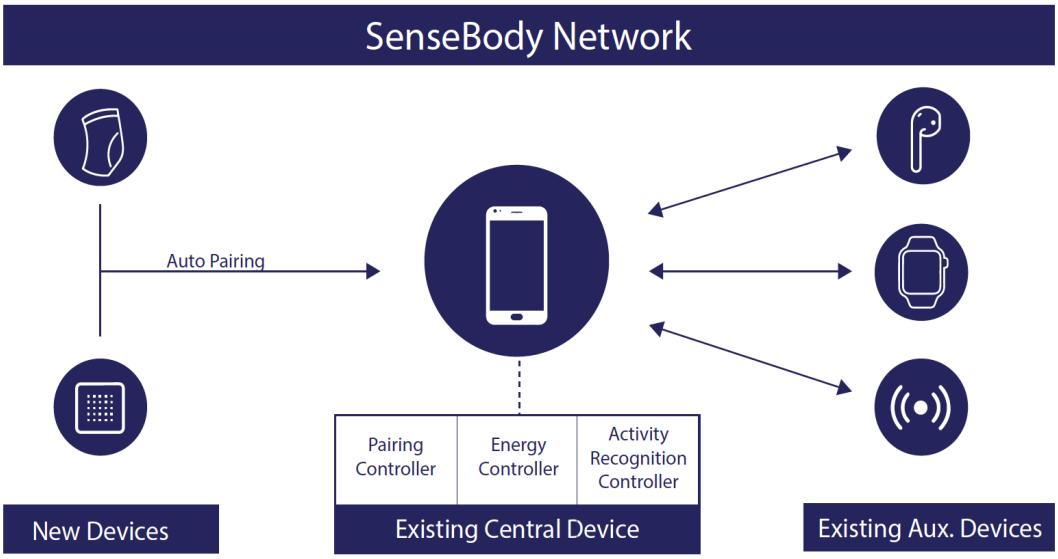

Figure 4.1: Overall architecture of SenseBody. The central device manages the existing auxiliary devices, and control the auto-pairing of new devices. Energy control and activity recognition tasks are all done on the central node.

Network transmission optimization

We firstly reduce our energy consumption by optimizing the network flow between devices. Three levels of cross-device communications are identified in our projects, raw-data level, feature-level, and gesture-level.

The naive way to perform multi-agent HAR is to send raw sensor signals to the central device. The central device will group those signals and perform activity recognition tasks. However, due to the high sampling rate (at least 30Hz) needed for HAR, it would be unrealistic to send each data record in real-time. The advanced way to perform multi-agent HAR is to calculate features from the recording window on each sensor and then send these features to the central device, and the central device will group these features from different devices to make the HAR tasks. The third way is to perform gesture classification in each distributed device, and the central device will make a prediction based on the gesture the auxiliary devices send.

Figure 4.2 shows the comparison between these three levels of cross-device communications. In our design, we use feature level to optimize the network transmission as well as the human labeling effort. Furthermore, this will increase user privacy since features

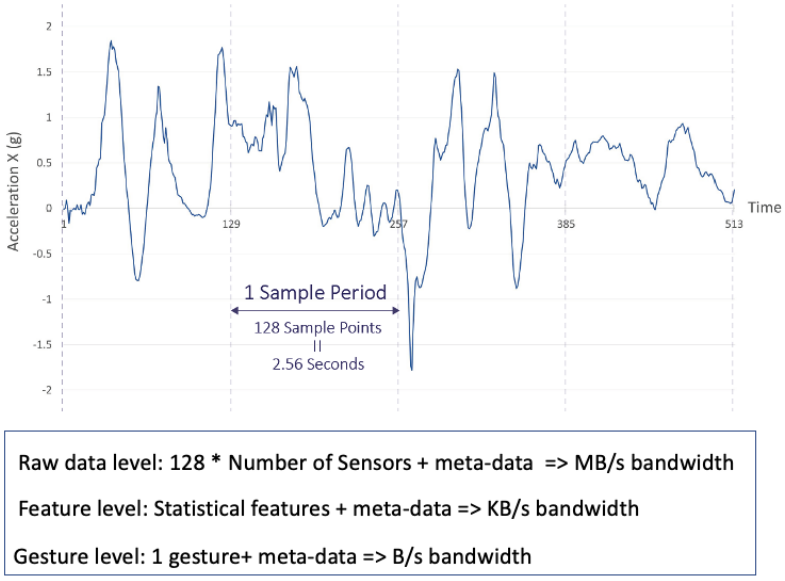

Figure 4.2: Three levels of network transmission optimization and comparison explained in a single sensor (accelerometerX)

might be hard to guess and infer information even if being exposed to a malicious attacker. The gesture classification is the best to optimize the network transmission. However, this method will need a large gesture dataset in the different body parts and require substantial human effort to segment gestures and label the segmented data. We will accomplish this in future work.

Devices duty cycle reduction

We use the exponential backoff algorithm to reduce our duty cycle. Since activity will last for several seconds or several minutes depending on activities, there is no need to keep sending features to the central device to perform HAR tasks. When the same activity happens in consecutive sampling points, we gradually increase the gap between the sample period (i.e. reduce the duty cycle). We store the last saved activity in the energy controller, and when the new activity from activity recognition controller comes in, and a change of an activity is detected by the energy controller, we decrease the gap between the sample period (increase the duty cycle). Figure 4.3 shows how we optimize the duty cycle.

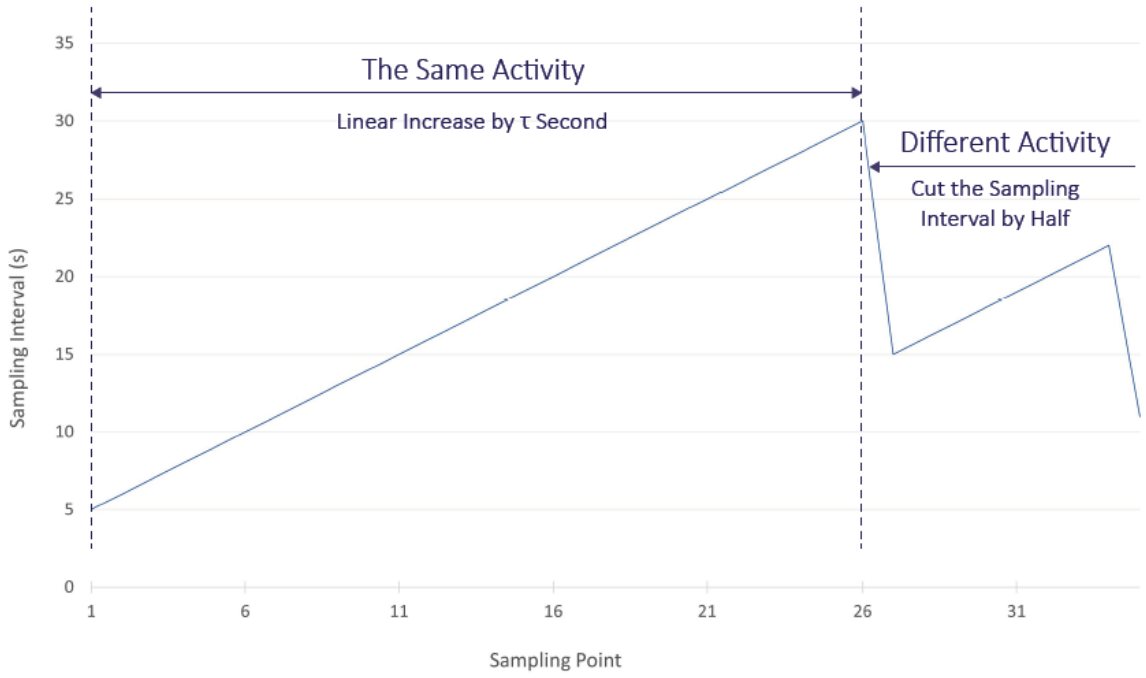

Figure 4.3: Duty cycle optimization design

Linear Increase The initial sampling interval will be Ti(s). When the current activity is the same activity of the previous record, we gradually increase the sampling interval by τ(s), and the new sampling interval will be the $(Ti + \tau)(s)$, and after N same activity, the sampling interval will become $(Ti + N * \tau)$(s)

Exponential Decrease When activity changes, we reduce sampling interval exponentially by a factor of 2, so the new sampling interval will become $(Ti + N * \tau)/2$(s)

Feature reduction

Even though the computation power for wearable devices is becoming stronger, we still need to only calculate the most valuable features because of the limited battery life a wearable device has. We reduce our features by identifying the most critical features during the training phase and limiting our features to the target numbers to maintain high accuracy while saving energy for the wearable devices.

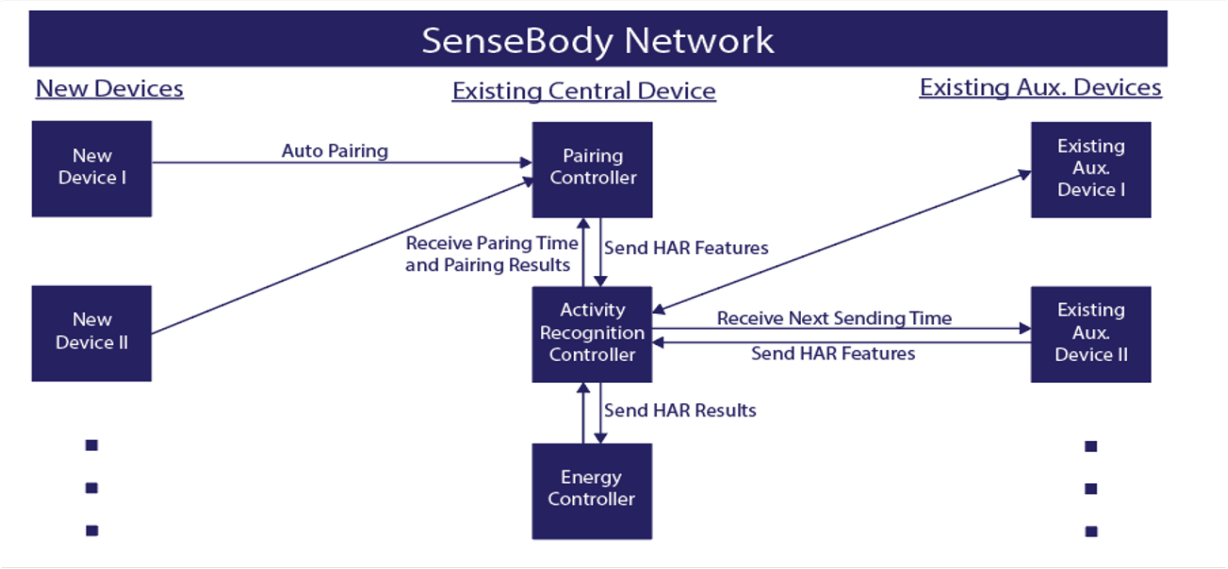

Figure 4.4: SenseBody system implementation

4.1.2 Activity recognition design

SenseBody system will synchronize clock and agree on the next checking time, all devices will send another sampling record (features) to the central device, and the central device will invoke the model and provide a prediction, after then, the central device will talk to auxiliary devices for the next checking time.

We follow the state-of-art design for HAR tasks and features in this project. We use 50Hz, 2.56s, with 128 sampling points as a window and with 50% overlap for collecting data. Detailed implementation is shown in the following section.

4.2 System Implementation

We implemented a prototype of SenseBody in Raspberry Pi in python, which can be used to run the auto-pairing algorithm and multi-agent human activity recognition. We use sensorLog[9] application using two apple watch, two iPhone and an iPad to simulate sensor data and monitor the energy consumption. Due to the engineering effort for the IOS and WatchOS devices, we decided to collect data from these devices first and later run the system using the collected data.

Figure 4.5: SenseBody environment setup, we use Raspberry PI to act like central device, and use SensorLog in iPhone, apple watch attached to different body part to record sensor data

4.2.1 Environment setup

Figure 4.5 shows devices we use in this experiment. We attach different devices into four different body parts for demonstration, including waist, wrist, shoulder and ankle. We then perform various activities as shown in table 4.1.

The data collection happens on the Raspberry pi using multiple nc command, the algorithm and ML development are using jupyter notebook in Python3, related libraries are `numpy`, `pandas`, `sklearn`, and `matplotlib` packages.

4.2.2 Energy Management

We implemented energy optimization methods based on the Section 4.1.1. Since we run our experiments on Raspberry pi, we can not measure the effect of only transmitting features instead of raw data.

When we optimize the duty cycle, We chose $T_i = 15.36$s, which is six times one window

Table 4.1: Activities summary

Activities	**Time collected (s)**	**Label**
Sit	180.48	0
Walk	134.4	1
Run	138.24	2
Push up	84.48	3
Sit up	92.16	4
Squat	115.2	5
Lying	138.24	6
Dumbbell Lateral Raise	99.84	7
Front Dumbbell Raise	80.64	8

time. Within 18s, we can consider the activity is highly likely to be the same and so that only one feature sample point needs to be calculated and send to the central device. We also implemented the maximum sampling interval time T_{max} so that the time interval will not be too much and cause an accuracy drop. Whenever the current interval time hits the maximum sampling interval, the system will not increase the sampling interval.

To further optimize the feature numbers and reduce the calculation, training, and testing overhead, we use the Random Forest classifier to determine the importance of variables. We then use the get_support function to select the features we are going to use automatically. Later, when training and applying models, well will only use filtered features.

4.2.3 Activity recognition

For the activity recognition, we invoke models trained from Section 4.3 and then predict both auto-pairing and HAR tasks.

4.3 Machine Learning Implementation

Figure 4.6 shows the overall flow chart for the implementation of the machine learning. Three phases will be discussed in the following sections, data preparation, feature selection and classifications.

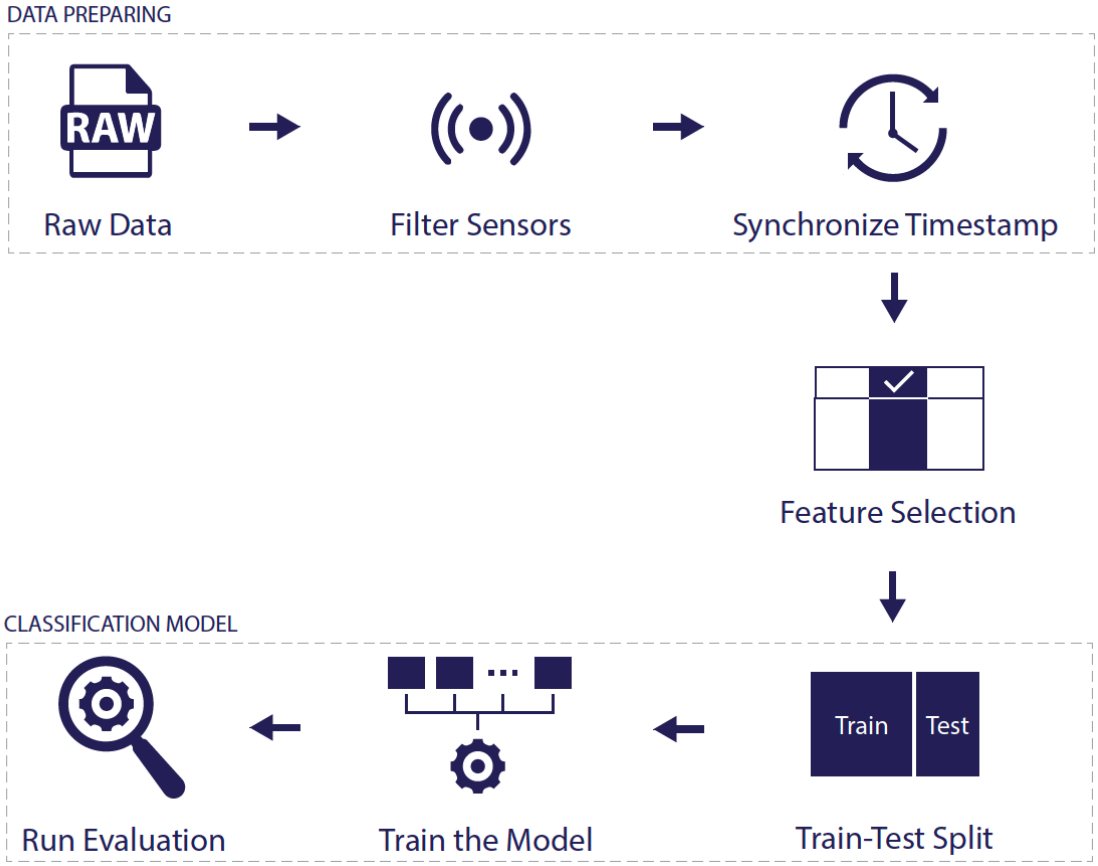

Figure 4.6: Machine learning implementation flow chart

4.3.1 Data preparation

We open the 5004 - 5009 port on the central device to establish a TCP connection for the different wearable devices. After collecting data, we filter the sensor data to what we use in the classification tasks. We delete data involving user privacy, including latitude, longitude, altitude, and IP address. Also, we remove unnecessary data such as magnetic fields and pedometer. The final raw data including device id, timestamp, tri-axial accelerometer and gyroscope data, and heading data.

Finally, we synchronize the time of collected raw data and make a further calibration to make sure that the collected sensor data from different devices are happening simultaneously.

4.3.2 Feature selection

After pre-processing the raw sensor data, we first calculate the statistical features of the tri-axial accelerometers, gyroscope, and headings, including mean value, maximum value, minimum value, standard deviation, and window. And then We also use the random forest classifier to determine the importance of variables. This can be used to filter the most important features.

4.3.3 Classifications

For the classification tasks, we separate the dataset into 67% for training and 33% for testing. We used several different machine learning models to perform the classification, including SVM, decision tree, neural network.

We also utilize a stacking classifier [17], which consists of stacking the output of individual estimators and use a classifier to compute the final prediction. Stacking allows utilizing the strength of each individual estimator by using their output as input of a final estimator.

4.4 Evaluation

4.4.1 Evaluation Metrics

We followed the state-of-the-art classification evaluation metrics [34] in this work. This includes accuracy score, precision, F1 score, and recall rate. These are measured by different equations for true positive (TP), true negative (TN), false positive (FP), and false negative (FN).

Accuracy in multi-label classification is the percentage the predicted label matches the actual label.

Precision is calculated with the ratio of $\frac{P_T}{P_T+P_F}$ where P_T is the number of true positives and P_F is the number of false positives.

F1 score can be interpreted as a weighted average of the precision and recall. It reaches its best value at 1 and its worse value at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is: $F = \frac{2pr}{p+r}$ where F is the F1 score, p is the precision, and r is the recall.

Recall is the ratio of $\frac{P_T}{P_T+N_F}$ where P_T is the number of true positives and N_F is the number of false negatives. The recall is, intuitively, the ability of the classifier to find all the positive samples.

When doing multi-label classification, the evaluation score can be calculated by doing macro, micro or weighted and None. Macro calculates metrics for each label and finds their unweighted mean. This does not take label imbalance into account. Micro calculates metrics globally by counting the total number of true positives, false negatives, and false positives. In our model comparison, we use the macro to eliminate data imbalance when the data is not at the same frequency.

Since embedded devices have limitations in computing power and memory, we evaluate the CPU and memory usage on our central device to show how running the classification model would affect normal operations of our embedded central node. For visualization, we select the famous Grafana platform to demonstrate the result.

Energy consumption is another key component for battery-powered devices. We evaluate the battery level change on each auxiliary sensor to demonstrate the effect on the battery life of these devices. We test the system performance under three different energy modes: no energy optimization, optimization designed by SenseBody, and low sample rate optimization. SensorLog is applied to report the battery level of each device under three scenarios to the central node.

4.4.2 Result

Table 4.2: Activity Recognition evaluation results

ML method	F1-score	Accuracy	Precision	Recall
SVM(rbf)	0.799	0.834	0.831	0.798
Decision Tree	0.969	0.971	0.974	0.967
Random Forest	0.973	0.975	0.974	0.974
Ridge Classifier	0.914	0.928	0.925	0.913
Logistic Regression	0.921	0.931	0.927	0.918
Stacking Classifier	0.982	0.983	0.981	0.981

Table 4.2 shows the activity recognition evaluation results versus different ML methods. The result follows SVM < Ridge Classifier ≈ Logistic Regression < Decision Tree ≈ Random

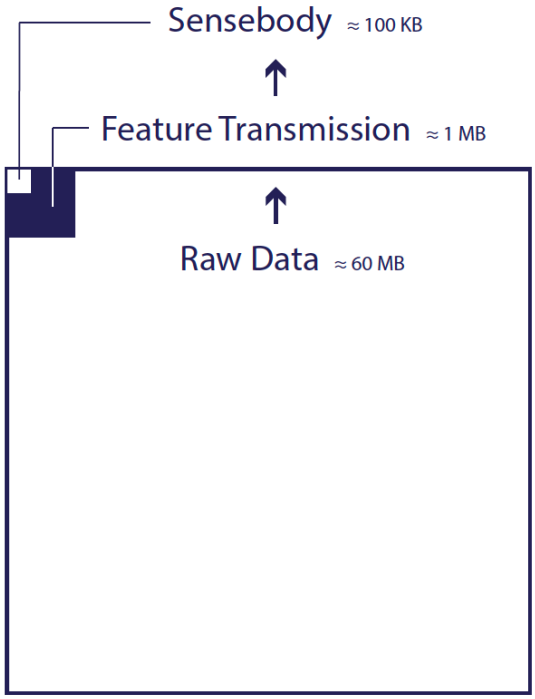

Figure 4.7: Data size compression process

Forest < Stacking Classifier in both F1-score, accuracy, precision and recall due to the same reason as we discussed in Table 5.1.

Figure 4.7 shows the compression process of our proposed system. The size of raw data is 60 MB. After the feature transmission process, the size of transmitted data reduced to 1MB. Then we optimize the duty cycle further to reduce the size of transmitted data into 100KB.

Figure 4.8 shows the memory and CPU usage while we are running the proposed system. We demonstrate a test case between 19:38 and 19:41. The average memory usage increases from 0 to 3 percent during the system running time for memory usage. CPU usage first has a massive jump of about 20 percent, which is caused by the python programming problem [37]. Then it shows an increase of 5 percent on average. Since in our system, we proposed the energy management optimization mechanism to benefit the system for computation and network transmission, so we can achieve lower overhead in memory and CPU usage while the system is running.

Figure 4.9 and Figure 4.10 show the F1 score and data size transmitted within 5 min-

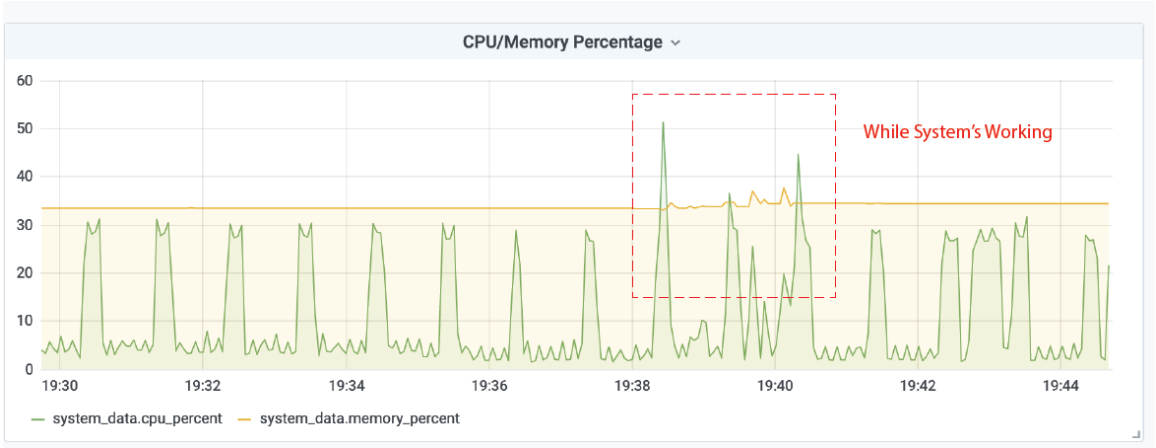

Figure 4.8: CPU and memory consumption visualization using Grafana of SenseBody

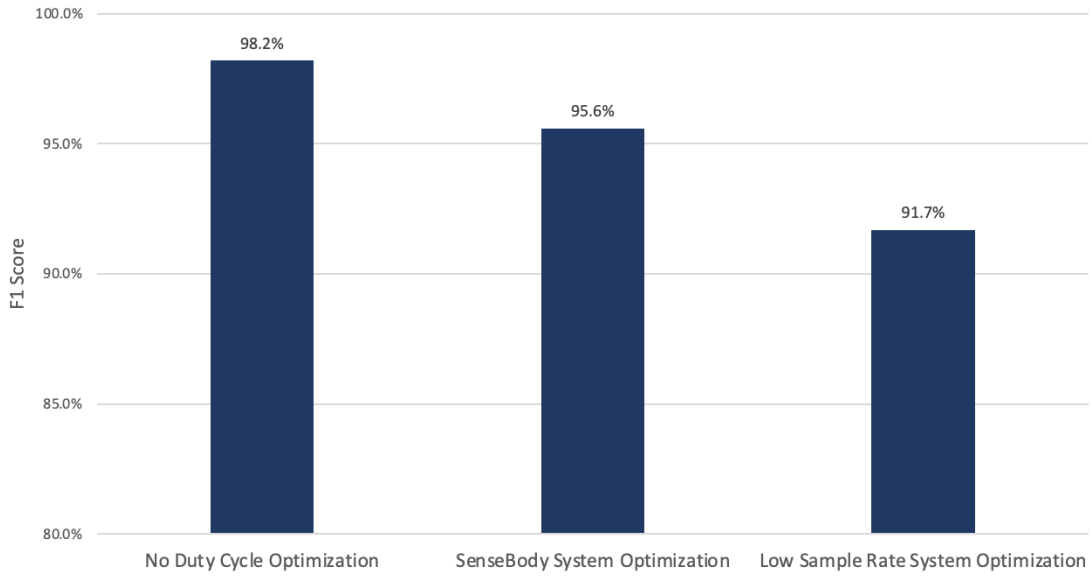

Figure 4.9: F1 score comparison with different optimization mechanism

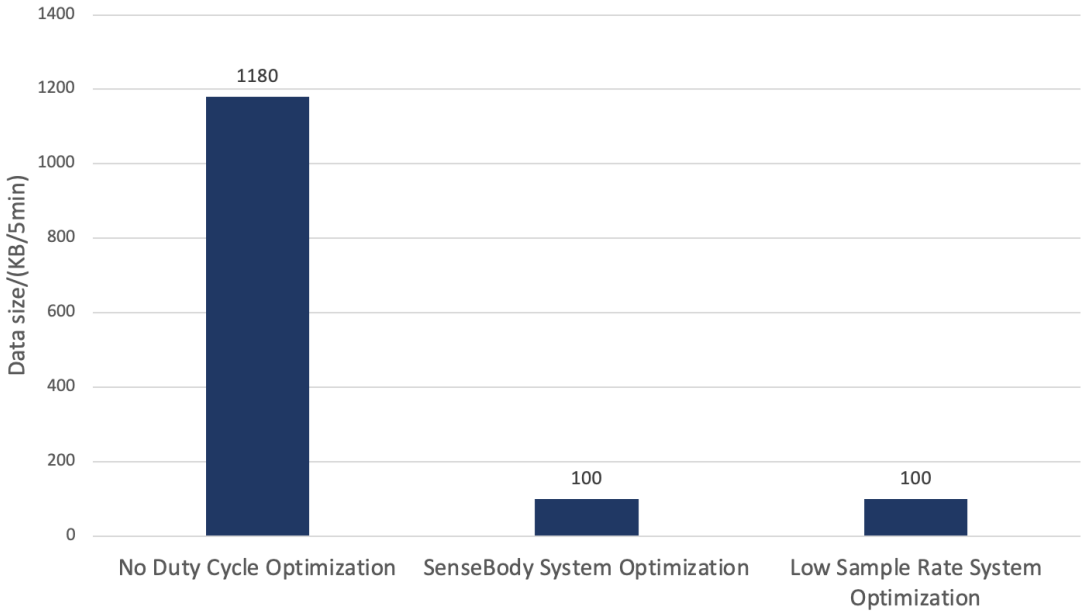

Figure 4.10: Data size optimization with different optimization mechanism

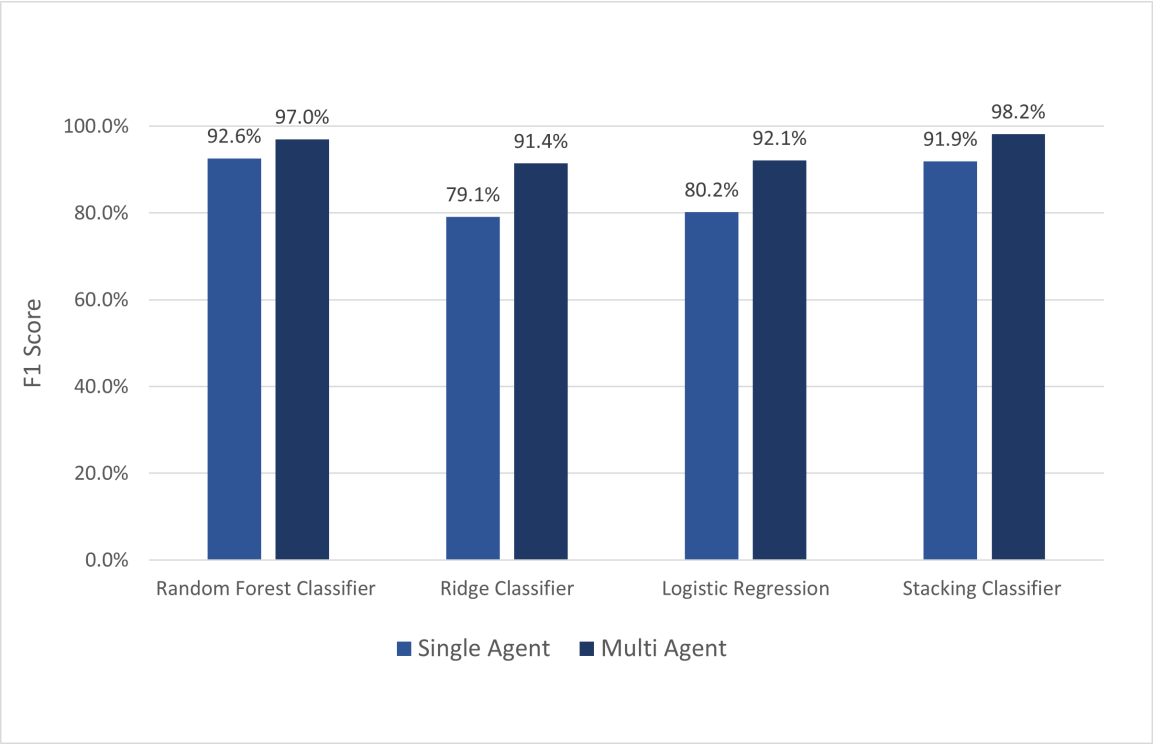

Figure 4.11: F1 score comparison for single agent and multi agent

utes versus different optimization methods respectively. The F1 score follows No duty cycle optimization > SenseBody System Optimization > Low Sample Rate System Optimization. The data size optimization follows No duty cycle optimization > SenseBody System Optimization = Low Sample Rate System Optimization. For No Duty Cycle Optimization, all data are transmitted between the devices, which leads to the highest F1 score but also causes the most energy since the total data transmission rate is the highest. For SenseBody System Optimization, we utilize the advantage of network transmission and duty cycle reduction to achieve 12x energy saving with only 2.6% F1 score lose. Low Sample Rate System Optimization can also save 12x energy compared to No duty cycle optimization. It sacrifices a 6.5% F1 score, which is much higher than SenseBody System Optimization.

Figure 4.11 shows the F1 score versus different classifiers in both single-agent and multi-agent. The result follows Stacking > Random Forest > Logistic > Ridge Classifier in both single and multi-agent cases due to the same reason as we discussed in Table 5.1. We also observe that, in both 4 classfier methods, multi agent is about 6% higher in F1 score comparing with single agent, we are safe to say that the performance for multi agent is better than single agent in our system.

Figure 4.12 show the confusion matrix for both single-agent and multi-agent system. As shown in the figure, the Y-axis is the predicted label and X-axis is the actual label, which means the squares on the catercorner diagonal are the correct prediction result. For the single-agent HAR, we can see that gestures 2,3,4 and 8 achieve lower results than the rest of the gestures. For multi-agent, all gestures achieve 90% or higher accuracy, which can illustrate the same result as shown in Figure 4.11.

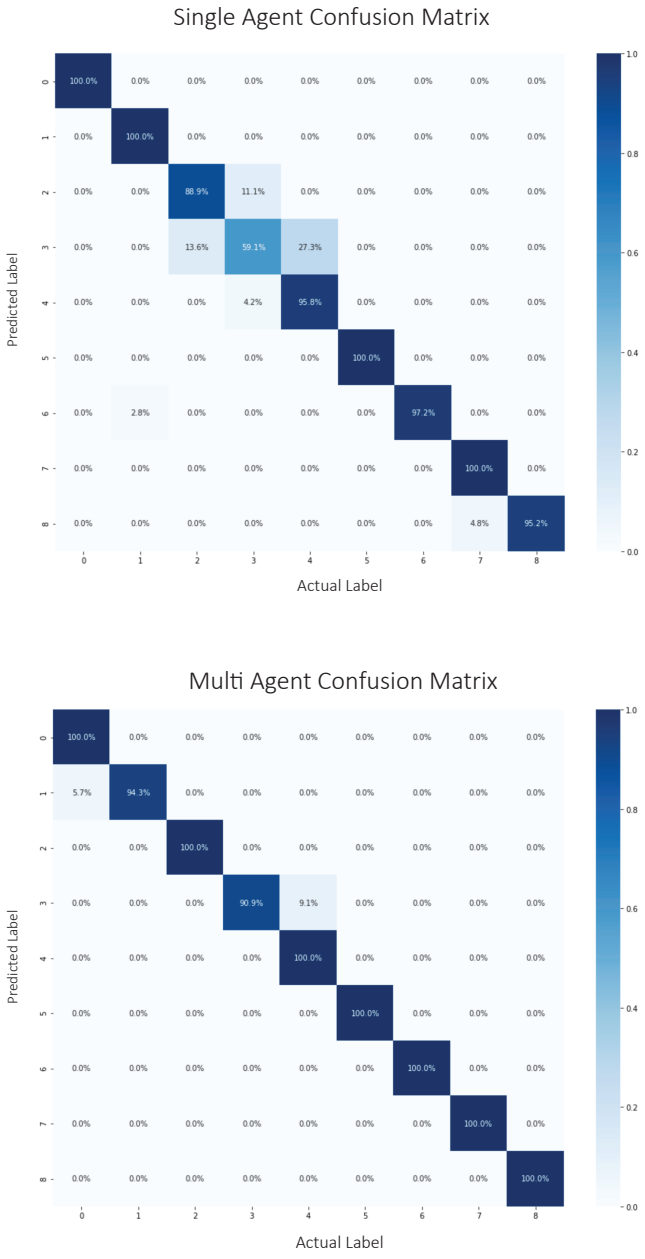

Figure 4.12: HAR tasks confusion matrix for single agent and multi agent

Chapter 5

Auto Pairing in SenseBody

5.1 Background

With more wearable devices existing on the markets and being developed, we are getting more functionalities enhancing our lives as well as putting more effort when connecting and pairing different devices. Traditionally, we use NFC technology to hold devices near each other, use QR code or Bluetooth pairing code to establish a pairing between devices, and this works fine if there are only limited devices. However, if more wearable devices are connected to the BSNs, it will cause more human efforts to pair these wearable devices.

Figure 5.1 shows one example of what will happen with more wearable devices. This is how researcher collected different sensor data from Opportunity dataset [14]. Even though we will not have these wires attached to our body, how to easily connect and configure these devices remain challenging

Due to the fact that wearable devices will be on the same person and follow similar activity patterns, we can actually utilize these sensor data to provide authentication for the pairing for wearable devices. We can group the existing device's accelerometer and gyroscope data with the new device's sensor data to make a binary classification on whether these two devices belong to the same person and use this authentication to pair new devices.

5.2 Auto Pairing Design

Figure 5.2 shows the design of the auto-pairing workflow. When a new device is nearby the existing network, it will search and communicates with the existing central device to

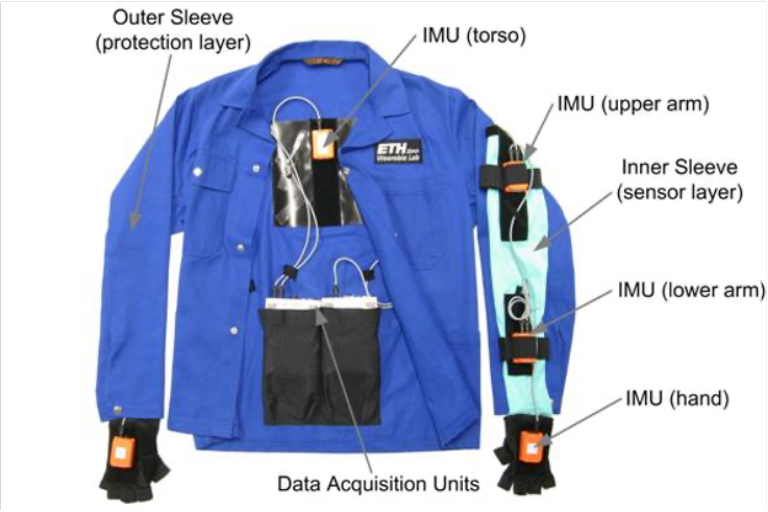

Figure 5.1: Opportunity dataset collecting figure [14]. More wearable devices attached to human body will result in more pairing effort.

perform auto-pairing.

A new device will first communicate to a public access point generated by the pairing controller and then synchronize the time with the central device. Then they will agree on a time point to start sampling and calculating features, the new device will then send its pairing features to the central device, and the central device will check with its activity features with the existing activity features to judge whether the new device should join the current SenseBody network. The central device will share the authentication to join the SenseBody network, and it will become an auxiliary device within the SenseBody.

5.3 Implementation

Our implementation for pairing is when the new device connects to the pairing controller. It will send its timestamp and the pairing controller will check the central device to get the time difference between the new device and the central device and arrange it for the next sending time for the new device. When the new device sends features to the pairing controller, Algorithm 1 will be invoked and generate a result, either a token or an error message. We collected data from 3 collecting points, two of them belong to the same person while the other one belongs to the other person, and ask them to do daily activities.

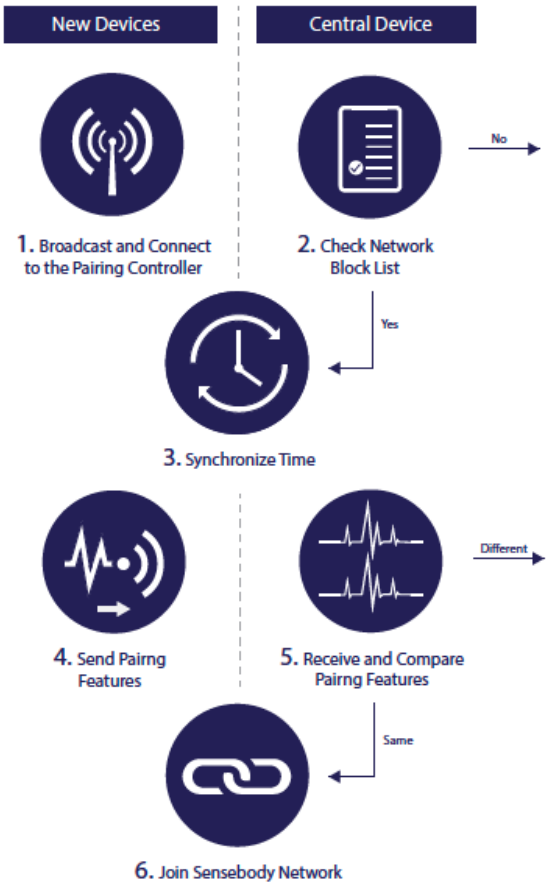

Figure 5.2: Auto-pairing design. New devices going through a 6-step joining procedure to join the network.

5.4 Evaluation

Table 5.1 shows the auto-pairing evaluation results versus different ML methods. The result follows SVM < Ridge Classifier ≈ Logistic Regression < Decision Tree ≈ Random Forest < Stacking Classifier in both F1-score, accuracy, precision and recall. The result is that, for SVM, the feature dimension is 136, and the dataset size for each gesture is around 100. So the performance for SVM is not promising. On the other hand, our problem is non-linear, so it's hard to choose the kernel function. In our experiment, we choose poly, RBF, and sigmoid as our kernel function, respectively, and the best result is SVM with RBF as kernel function, which achieves 0.948 in F1-score and 0.955 in accuracy. The

Algorithm 1: Auto Pairing procedures

Input: E: The existing set of devices. $e_i \in E$ is i-th device
PM: Pairing model we trained for the auto pairing
Output: res: Either a valid token or not able to connect error message

```
1 procedure pairNewDevice(PM)
2     res ← error
3     curFeature = getNewActivityFeatures ()
4     for e_i ∈ E do
5         if PM.predict (curFeature, e_i) == 1 then
6             token = generateNewToken ()
7             res = token
8             break
9     return res
```

stacking classifier contains a meta classifier to multiple aggregate classifier to perform a better classification result, which can achieve 0.982 in F1-score and 0.991 in accuracy.

Table 5.1: Auto pairing evaluation results

ML method	F1-score	Accuracy	Precision	Recall
SVM(rbf)	0.948	0.955	0.965	0.934
Decision Tree	0.980	0.982	0.986	0.973
Random Forest	0.983	0.985	0.989	0.977
Ridge Classifier	0.969	0.973	0.980	0.959
Logistic Regression	0.959	0.964	0.971	0.948
Stacking Classifier	0.989	0.991	0.993	0.986

Chapter 6

Discussion

In this section, we discuss some of our system's limitations for the implementation, as well as some future directions and work that can be done based on the system.

6.1 Limitations

Even though our SenseBody system indicates that we can improve HAR tasks accuracy when coordinating different devices in an energy-efficient way, and provides possible applications like auto-pairing. There are still challenges existing since the system is not built in different platforms/operating systems due to the engineering efforts. Still, we do expect that a standard protocol needs to be made and adopted into wearable devices to facilitate the interaction among devices.

Also, the current data only records one person for demonstration purposes. We expect to get more data from different people and come up with better results and more persuasive results.

6.2 Future work

6.2.1 Algorithm Enhancements

Our algorithms are built in a relatively simple configuration with one person involved due to the Covid-19. Thus, the model effectiveness and robustness still need to be improved.

We have provided two insights for improving the algorithms, handling different gestures for the same activities, and dealing with dynamic sampling windows to fit in various activities.

Different gestures for the same activities

Our classification model is built based on prior data collected for specific gestures. Thus, with the fact that different people might have different gestures for a particular meaning, our model needs to be calibrated before the model can apply it to different users. To make our system more adaptive to various users, we will need to enlarge our dataset and include more gestures from different people. We might use some other machine learning techniques such as reinforcement learning or federated machine learning to improve our model.

Dynamic sampling window to fit in different activities

SenseBody is currently using a fixed sampling window for all activities. However, in our experiment, we have witnessed different duration times for different activities. Optimizing sampling window for different activities have more energy-saving potential for these embedded wearable devices. Activities with longer duration and fewer movements can have a longer sampling period. For example, the sampling rate on running could be faster than lying since the latter activity is more stable and static.

6.2.2 New Applications

We have built our prototype on existing COTs devices and have demonstrated that cooperating between multiple devices can greatly increase activity recognition accuracy on the human body. As we have mentioned in the previous subsection, we plan to extend our algorithm to more generalized scenarios. This includes extending our system to a generalized scenario, creating a data collection and activity recognition framework that can be used on any on-body devices. There are two major directions for the potential new applications:

Opportunities for human computer interaction

One direction is to extend our recognition system to complex gestures on different body parts; this requires more on-body sensors and even some device-free sensing techniques. Achieving complex gesture recognition on top of activity recognition can help better understanding what the user is doing and is introducing more opportunities for the user to

control surrounding devices. For example, when the system recognizes the user is lying on a sofa, the user can then control the volume of music by just waving his/her hand without using the remote control. This brings significant benefits for the HCI in future smart homes.

Adaptive duty cycling method

Another direction is to design an adaptive duty cycling method to fit the need for different activities. For example, increase the duty cycle in sports like playing tennis can record each stroke and achieve higher accuracy. For activities like sitting and lying, we can reduce the duty cycle and extend the data collection period to prolong the operating time for wearable devices.

Chapter 7

Conclusion

Our work evaluates the possibility and energy efficiency for connecting wearable devices in the BSNs, which is critical for human activity recognition using multiple on-body devices. The SenseBody demonstrates how we design a secure data transmission and storage among these wearable devices and collaborate to auto-pairing and conduct HAR tasks. The evaluation results validate the contributions of the proposed system. Our results show that we can achieve an average of 98.3% of recognition accuracy on nine common activities running our proposed multi-agent framework and achieve more than 98.9% auto-pairing accuracy. From an energy-saving perspective, our result also shows that our system can save more than 90% of energy consumed in data transmission over traditional methods while only sacrificing 3% of accuracy.

References

[1] About one-in-five americans use a smart watch or fitness tracker. *https://www.pewresearch.org/fact-tank/2020/01/09/about-one-in-five-americans-use-a-smart-watch-or-fitness-tracker/: :text=As 2020 begins – and health,June 3-17, 2019.*, [Accessed in JUL. 2020].

[2] Airpods pro - technical specifications. *https://www.apple.com/airpods-pro/specs/*, [Accessed in JUL. 2020].

[3] Apple fitness. *https://www.apple.com/apple-fitness-plus/*, [Accessed in JUL. 2020].

[4] Nintendo switch system software. *https://www.nintendo.com/switch/*, [Accessed in JUL. 2020].

[5] Ring fit adventure. *https://en.wikipedia.org/wiki/Ring_Fit_Adventure*, [Accessed in JUL. 2020].

[6] Smartwatch unit sales in the us 2016-2020. *https://www.statista.com/statistics/381696/wearables unit-sales-forecast-united-states-by-category/*, [Accessed in JUL. 2020].

[7] Wearables sales worldwide by region 2015-2022. *https://www.statista.com/statistics/490231/wearable-devices-worldwide-by-region/*, [Accessed in JUL. 2020].

[8] Wrist-worn wearable shipments worldwide 2019-2024. *https://www.statista.com/statistics/296565/wearables-worldwide-shipments/*, [Accessed in JUL. 2020].

[9] Hasan Faik Alan, Bert Arnrich, Cem Ersoy, and Burcu Cinaz. Sensor log: A mobile data collection and annotation application. In *2014 22nd Signal Processing and Communications Applications Conference (SIU)*, pages 1375–1378. IEEE, 2014.

[10] C. Alippi, G. Anastasi, M. Di Francesco, and M. Roveri. An adaptive sampling algorithm for effective energy management in wireless sensor networks with energy-hungry sensors. *IEEE Transactions on Instrumentation and Measurement*, 59(2):335–344, Feb 2010.

[11] Ling Bao and Stephen S Intille. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*, pages 1–17. Springer, 2004.

[12] Robert Bodor, Bennett Jackson, and Nikolaos Papanikolopoulos. Vision-based human tracking and activity recognition. In *Proc. of the 11th Mediterranean Conf. on Control and Automation*, volume 1. Citeseer, 2003.

[13] Ryan Burchfield and S Venkatesan. A framework for golf training using low-cost inertial sensors. In *2010 International Conference on Body Sensor Networks*, pages 267–272. IEEE, 2010.

[14] Ricardo Chavarriaga, Hesam Sagha, Alberto Calatroni, Sundara Tejaswi Digumarti, Gerhard Tröster, José del R Millán, and Daniel Roggen. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recognition Letters*, 34(15):2033–2042, 2013.

[15] Kaixuan Chen, Lina Yao, Dalin Zhang, Bin Guo, and Zhiwen Yu. Multi-agent attentional activity recognition. *arXiv preprint arXiv:1905.08948*, 2019.

[16] Luke Conroy, Ciarán Ó Conaire, Shirley Coyle, Graham Healy, Philip Kelly, Damien Connaghan, Noel E O'Connor, Alan F Smeaton, Brian Caulfield, and Paddy Nixon. Tennissense: a multi-sensory approach to performance analysis in tennis. *27th International Society of Biomechanics in Sports Conference*, 2009.

[17] Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine learning*, 54(3):255–273, 2004.

[18] Bjoern M Eskofier, Sunghoon Ivan Lee, Manuela Baron, André Simon, Christine F Martindale, Heiko Gaßner, and Jochen Klucken. An overview of smart shoes in the internet of health things: gait and mobility assessment in health promotion and disease monitoring. *Applied Sciences*, 7(10):986, 2017.

[19] Raffaele Gravina, Parastoo Alinia, Hassan Ghasemzadeh, and Giancarlo Fortino. Multi-sensor fusion in body sensor networks: State-of-the-art and research challenges. *Information Fusion*, 35:68–80, 2017.

[20] Yuya Hanai, Jun Nishimura, and Tadahiro Kuroda. Haar-like filtering for human activity recognition using 3d accelerometer. In *2009 ieee 13th digital signal processing workshop and 5th ieee signal processing education workshop*, pages 675–678. IEEE, 2009.

[21] Zhen-Yu He and Lian-Wen Jin. Activity recognition from acceleration data using ar model representation and svm. In *2008 international conference on machine learning and cybernetics*, volume 4, pages 2245–2250. IEEE, 2008.

[22] McKinsey Global Institute. The Internet of Things: Mapping the Value Beyond the Hype, 2015.

[23] Stephen S Intille, Kent Larson, J Beaudin, E Munguia Tapia, Pallavi Kaushik, Jason Nawyn, and Thomas J McLeish. The placelab: A live-in laboratory for pervasive computing research (video). *Proceedings of PERVASIVE 2005 Video Program*, 2005.

[24] Stephen S Intille, Kent Larson, Emmanuel Munguia Tapia, Jennifer S Beaudin, Pallavi Kaushik, Jason Nawyn, and Randy Rockinson. Using a live-in laboratory for ubiquitous computing research. In *International Conference on Pervasive Computing*, pages 349–365. Springer, 2006.

[25] Ahmad Jalal, Shaharyar Kamal, and Daijin Kim. A depth video-based human detection and activity recognition using multi-features and embedded hidden markov models for health care monitoring systems. *International Journal of Interactive Multimedia & Artificial Intelligence*, 4(4), 2017.

[26] Yong Won Jang. Smart shoes, method of providing sensor information to smart shoes, smart device and method of providing guidance program via smart device, November 12 2019. US Patent 10,473,483.

[27] Ga-hyun Joo. Wearable glasses and method of providing content using the same, July 31 2018. US Patent 10,037,084.

[28] Adil Mehmood Khan, Young-Koo Lee, Sungyoung Y Lee, and Tae-Seong Kim. A triaxial accelerometer-based physical-activity recognition via augmented-signal features and a hierarchical recognizer. *IEEE transactions on information technology in biomedicine*, 14(5):1166–1172, 2010.

[29] Aftab Khan, Nils Hammerla, Sebastian Mellor, and Thomas Plötz. Optimising sampling rates for accelerometer-based human activity recognition. *Pattern Recognition Letters*, 73:33–40, 2016.

[30] Yasser Khan, Aminy E Ostfeld, Claire M Lochner, Adrien Pierre, and Ana C Arias. Monitoring of vital signs with flexible and wearable medical devices. *Advanced Materials*, 28(22):4373–4395, 2016.

[31] Eunju Kim, Sumi Helal, and Diane Cook. Human activity recognition and pattern discovery. *IEEE pervasive computing*, 9(1):48–53, 2009.

[32] S Santhosh Kumar and Mala John. Human activity recognition using optical flow based feature set. In *2016 IEEE international Carnahan conference on security technology (ICCST)*, pages 1–5. IEEE, 2016.

[33] Xiaochen Lai, Quanli Liu, Xin Wei, Wei Wang, Guoqiao Zhou, and Guangyi Han. A survey of body sensor networks. *Sensors*, 13(5):5406–5447, 2013.

[34] Oscar D Lara and Miguel A Labrador. A survey on human activity recognition using wearable sensors. *IEEE communications surveys & tutorials*, 15(3):1192–1209, 2012.

[35] Young-Seol Lee and Sung-Bae Cho. Activity recognition using hierarchical hidden markov models on a smartphone with 3d accelerometer. In *International conference on hybrid artificial intelligence systems*, pages 460–467. Springer, 2011.

[36] Knud Lasse Lueth. State of the iot 2018: Number of iot devices now at 7b – market accelerating, Aug 2018.

[37] Mark Lutz. *Programming python.* " O'Reilly Media, Inc.", 2001.

[38] Haojie Ma, Wenzhong Li, Xiao Zhang, Songcheng Gao, and Sanglu Lu. Attnsense: Multi-level attention mechanism for multimodal human activity recognition. In *IJCAI*, pages 3109–3115, 2019.

[39] Anna McLister, Jolene McHugh, Jill Cundell, and James Davis. New developments in smart bandage technologies for wound diagnostics. *Advanced Materials*, 28(27):5732–5737, 2016.

[40] Pooria Mostafalu, Ali Tamayol, Rahim Rahimi, Manuel Ochoa, Akbar Khalilpour, Gita Kiaee, Iman K Yazdi, Sara Bagherifard, Mehmet R Dokmeci, Babak Ziaie, et al. Smart bandage for monitoring and treatment of chronic wounds. *Small*, 14(33):1703509, 2018.

[41] Adam E Newham. Smart pairing using bluetooth technology, November 17 2015. US Patent 9,191,988.

[42] UK Government Office of Science. The Internet of Things: Making the Most of the Second Digital Revolution, 2014.

[43] Godwin Ogbuabor and Robert La. Human activity recognition for healthcare using smartphones. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, pages 41–46, 2018.

[44] Madhuri Panwar, S Ram Dyuthi, K Chandra Prakash, Dwaipayan Biswas, Amit Acharyya, Koushik Maharatna, Arvind Gautam, and Ganesh R Naik. Cnn based approach for activity recognition using a wrist-worn accelerometer. In *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 2438–2441. IEEE, 2017.

[45] X. Qi, M. Keally, G. Zhou, Y. Li, and Z. Ren. Adasense: Adapting sampling rates for activity recognition in body sensor networks. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 163–172, 2013.

[46] Muhammad Asif Razzaq, Javier Medina Quero, Ian Cleland, Chris Nugent, Usman Akhtar, Hafiz Syed Muhammad Bilal, Ubaid Ur Rehman, and Sungyoung Lee. umodt: an unobtrusive multi-occupant detection and tracking using robust kalman filter for real-time activity recognition. *Multimedia Systems*, 26(5):553–569, 2020.

[47] Hesam Sagha, Sundara Tejaswi Digumarti, José del R Millán, Ricardo Chavarriaga, Alberto Calatroni, Daniel Roggen, and Gerhard Tröster. Benchmarking classification techniques using the opportunity human activity dataset. In *2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 36–40. IEEE, 2011.

[48] Stephen Schooley, Paul Heninwolf, and Christopher Jones. Home automation device pairing by nfc-enabled portable device, July 30 2013. US Patent 8,498,572.

[49] Congressional Research Service. The Internet of Things(IoT): An Overview, 2020.

[50] Roshan Singh, Ankur Sonawane, and Rajeev Srivastava. Recent evolution of modern datasets for human activity recognition: a deep survey. *Multimedia Systems*, pages 1–24, 2019.

[51] Abdulhamit Subasi, Mariam Radhwan, Rabea Kurdi, and Kholoud Khateeb. Iot based mobile healthcare system for human activity recognition. In *2018 15th Learning and Technology Conference (L&T)*, pages 29–34. IEEE, 2018.

[52] Halim Tannous, Dan Istrate, Aziz Benlarbi-Delai, Julien Sarrazin, Didier Gamet, Marie Christine Ho Ba Tho, and Tien Tuan Dao. A new multi-sensor fusion scheme to improve the accuracy of knee flexion kinematics for functional rehabilitation movements. *Sensors*, 16(11):1914, 2016.

[53] Emmanuel Munguia Tapia, Stephen S Intille, and Kent Larson. Activity recognition in the home using simple and ubiquitous sensors. In *International conference on pervasive computing*, pages 158–175. Springer, 2004.

[54] Amin Ullah, Khan Muhammad, Javier Del Ser, Sung Wook Baik, and Victor Hugo C de Albuquerque. Activity recognition using temporal optical flow convolutional features and multilayer lstm. *IEEE Transactions on Industrial Electronics*, 66(12):9692–9702, 2018.

[55] Thomas L Wood. Smart shoes, December 20 1994. US Patent 5,373,651.

[56] Allen Y Yang, Roozbeh Jafari, S Shankar Sastry, and Ruzena Bajcsy. Distributed recognition of human actions using wearable motion sensor networks. *Journal of Ambient Intelligence and Smart Environments*, 1(2):103–115, 2009.

[57] Guang-Zhong Yang and Guangzhong Yang. *Body sensor networks*, volume 1. Springer, 2006.

[58] Jerald Yoo, Long Yan, Seulki Lee, Hyejung Kim, and Hoi-Jun Yoo. A wearable ecg acquisition system with compact planar-fashionable circuit board-based shirt. *IEEE Transactions on Information Technology in Biomedicine*, 13(6):897–902, 2009.

[59] Ming Zeng, Le T Nguyen, Bo Yu, Ole J Mengshoel, Jiang Zhu, Pang Wu, and Joy Zhang. Convolutional neural networks for human activity recognition using mobile sensors. In *6th International Conference on Mobile Computing, Applications and Services*, pages 197–205. IEEE, 2014.

[60] Mi Zhang and Alexander A Sawchuk. Usc-had: a daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1036–1043, 2012.

[61] Rui Zhang and Oliver Amft. Monitoring chewing and eating in free-living using smart eyeglasses. *IEEE journal of biomedical and health informatics*, 22(1):23–32, 2017.

APPENDICES

Appendix A

Code Sample and Link

The code and sample data will be available at https://github.com/Lichking1/senseBody

A.1 Insert system data into Grafana using influxdb

```
import psutil
from influxdb import InfluxDBClient
import time

client = InfluxDBClient(host='localhost', port=8086)
client.create_database('system')

measurement_name = 'system_data'
data_end_time = int(time.time() * 1000)
data = []
cpu_p, mem_p, disk_read, disk_write, net_sent_now, net_recv_now, temp, \
    boot_time, net_sent_prev, net_recv_prev = \
    0, 0, 0, 0, 0, 0, 0, 0, \
    psutil.net_io_counters().bytes_sent, psutil.net_io_counters().
        ↪ bytes_recv


def get_system_data():
```

```
    global cpu_p, mem_p, disk_write, disk_read, net_recv_now, net_sent_now
        ↪ ,\
        temp, boot_time, data_end_time
    data_end_time = int(time.time() * 1000)
    cpu_p = psutil.cpu_percent()
    mem_p = psutil.virtual_memory().percent
    disk_read = psutil.disk_io_counters().read_count
    disk_write = psutil.disk_io_counters().write_count
    net_sent_now = psutil.net_io_counters().bytes_sent
    net_recv_now = psutil.net_io_counters().bytes_recv
    boot_time = psutil.boot_time()

    data.append(
        {
            "measurement": "system_data",
            "tags": {
                "boot_time": boot_time
            },
            "fields": {
                "cpu_percent": cpu_p,
                "memory_percent": mem_p,
                "disk_read": disk_read,
                "disk_write": disk_write,
                "net_sent": net_sent_now-net_sent_prev,
                "net_received": net_recv_now-net_recv_prev,
                "temperature": temp,
            },
            "time": data_end_time
        }
    )

    client.write_points(data, database='system', time_precision='ms',
                        protocol='json')


def run(interval=3): # interval in seconds
    global net_recv_prev, net_sent_prev
    print("Script␣is␣running,␣press␣Ctrl+C␣to␣stop!")
```

```
    while 1:
        try:
            get_system_data()
            net_sent_prev = psutil.net_io_counters().bytes_sent
            net_recv_prev = psutil.net_io_counters().bytes_recv
            time.sleep(interval)
        except KeyboardInterrupt:
            quit()
        except Exception as e:
            print(e)
            pass


run()
```

A.2 Main jupyter notebook code for SenseBody

```
#!/usr/bin/env python
# coding: utf-8

#

import seaborn as sns
from sklearn.metrics import confusion_matrix
import warnings
import time
from sklearn.metrics import *
from sklearn.neighbors import *
from sklearn.svm import *
from sklearn.naive_bayes import *
from sklearn.linear_model import *
from sklearn.model_selection import *
from sklearn.preprocessing import *
from sklearn.feature_selection import *
from sklearn.tree import *
from sklearn.ensemble import *
```

```
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import numpy as np
import os
import pandas as pd
from IPython.display import display, HTML


sz = 10000000
# Header function


def hf(string):
    print("========" + string + "==========")


rawPCols = ['loggingTime', 'loggingSample', 'locationTimestamp_since1970',
    ↪  'locationLatitude', 'locationLongitude',
           'locationAltitude', 'locationSpeed', 'locationCourse', '
              ↪ locationVerticalAccuracy',
           'locationHorizontalAccuracy', 'locationFloor', '
              ↪ locationHeadingTimestamp_since1970',
           'locationHeadingX', 'locationHeadingY', 'locationHeadingZ', '
              ↪ locationTrueHeading', 'locationMagneticHeading',
           'locationHeadingAccuracy', 'accelerometerTimestamp_sinceReboot'
              ↪ , 'accelerometerAccelerationX',
           'accelerometerAccelerationY', 'accelerometerAccelerationZ', '
              ↪ gyroTimestamp_sinceReboot', 'gyroRotationX',
           'gyroRotationY', 'gyroRotationZ', '
              ↪ magnetometerTimestamp_sinceReboot', 'magnetometerX', '
              ↪ magnetometerY',
           'magnetometerZ', 'motionTimestamp_sinceReboot', 'motionYaw', '
              ↪ motionRoll', 'motionPitch', 'motionRotationRateX',
           'motionRotationRateY', 'motionRotationRateZ', '
              ↪ motionUserAccelerationX', 'motionUserAccelerationY',
           'motionUserAccelerationZ', 'motionAttitudeReferenceFrame', '
              ↪ motionQuaternionX', 'motionQuaternionY',
```

```
            'motionQuaternionZ', 'motionQuaternionW', 'motionGravityX', '
               ↪ motionGravityY', 'motionGravityZ',
            'motionMagneticFieldX', 'motionMagneticFieldY', '
               ↪ motionMagneticFieldZ', 'motionHeading',
            'motionMagneticFieldCalibrationAccuracy', '
               ↪ activityTimestamp_sinceReboot', 'activity',
            'activityActivityConfidence', 'activityActivityStartDate', '
               ↪ pedometerStartDate',
            'pedometerNumberofSteps', 'pedometerAverageActivePace',
            'pedometerCurrentPace', 'pedometerCurrentCadence', '
               ↪ pedometerDistance', 'pedometerFloorAscended',
            'pedometerFloorDescended', 'pedometerEndDate', '
               ↪ altimeterTimestamp_sinceReboot', 'altimeterReset',
            'altimeterRelativeAltitude', 'altimeterPressure', '
               ↪ IP_Timestamp_since1970', 'IP_en0', 'IP_pdp_ip0', 'deviceID
               ↪ ',
            'deviceOrientationTimeStamp_since1970', 'deviceOrientation', '
               ↪ batteryTimeStamp_since1970', 'batteryState',
            'batteryLevel',
            'avAudioRecorder_Timestamp_since1970', '
               ↪ avAudioRecorderPeakPower', 'avAudioRecorderAveragePower',
               ↪ 'label']

p2Cols = ['locationTimestamp_since1970',
          'locationHeadingX', 'locationHeadingY', 'locationHeadingZ',
          'accelerometerAccelerationX', 'accelerometerAccelerationY', '
             ↪ accelerometerAccelerationZ',
          'gyroRotationX', 'gyroRotationY', 'gyroRotationZ', 'batteryLevel'
             ↪ ]


p1Cols = ['locationTimestamp_since1970(s)',
          'accelerometerAccelerationX(G)', 'accelerometerAccelerationY(G)',
             ↪  'accelerometerAccelerationZ(G)',
          'motionRotationRateX(rad/s)', 'motionRotationRateY(rad/s)', '
             ↪ motionRotationRateZ(rad/s)',
          'deviceOrientation(Z)',
          'batteryLevel(Z)']
```

```
rawWCols = ['locationTimestamp_since1970(s)', 'locationLatitude(WGS84)',
           'locationLongitude(WGS84)', 'locationAltitude(m)', '
              ↪ locationSpeed(m/s)', 'locationCourse()',
           'locationVerticalAccuracy(m)', 'locationHorizontalAccuracy(m)',
              ↪  'locationFloor(Z)',
           'accelerometerTimestamp_sinceReboot(s)', '
              ↪ accelerometerAccelerationX(G)',
           'accelerometerAccelerationY(G)', 'accelerometerAccelerationZ(G)
              ↪ ', 'motionTimestamp_sinceReboot(s)',
           'motionYaw(rad)', 'motionRoll(rad)', 'motionPitch(rad)', '
              ↪ motionRotationRateX(rad/s)',
           'motionRotationRateY(rad/s)', 'motionRotationRateZ(rad/s)', '
              ↪ motionUserAccelerationX(G)',
           'motionUserAccelerationY(G)', 'motionUserAccelerationZ(G)', '
              ↪ motionAttitudeReferenceFrame(txt)',
           'motionQuaternionX(R)', 'motionQuaternionY(R)', '
              ↪ motionQuaternionZ(R)', 'motionQuaternionW(R)',
           'motionGravityX(G)', 'motionGravityY(G)', 'motionGravityZ(G)',
              ↪ 'motionMagneticFieldX(T)',
           'motionMagneticFieldY(T)', 'motionMagneticFieldZ(T)', '
              ↪ motionHeading()',
           'motionMagneticFieldCalibrationAccuracy(Z)', '
              ↪ activityTimestamp_sinceReboot(s)', 'activity(txt)',
           'activityActivityConfidence(Z)', 'activityActivityStartDate(txt
              ↪ )', 'pedometerStartDate(txt)',
           'pedometerNumberofSteps(N)', 'pedometerAverageActivePace(s/m)',
              ↪  'pedometerCurrentPace(s/m)',
           'pedometerCurrentCadence(steps/s)', 'pedometerDistance(m)', '
              ↪ pedometerFloorAscended(N)',
           'pedometerFloorDescended(N)', 'pedometerEndDate(txt)', '
              ↪ altimeterTimestamp_sinceReboot(s)',
           'altimeterReset(bool)', 'altimeterRelativeAltitude(m)', '
              ↪ altimeterPressure(kPa)', 'batteryState(N)',
           'batteryLevel(R)', 'deviceID(txt)']
complexWCols = ['locationTimestamp_since1970(s)',
```

```
                'accelerometerAccelerationX(G)', 'accelerometerAccelerationY
                    ↪ (G)', 'accelerometerAccelerationZ(G)',
                'motionYaw(rad)', 'motionRoll(rad)', 'motionPitch(rad)',
                'motionRotationRateX(rad/s)', 'motionRotationRateY(rad/s)',
                    ↪ 'motionRotationRateZ(rad/s)',
                'motionUserAccelerationX(G)',
                'motionUserAccelerationY(G)', 'motionUserAccelerationZ(G)',
                    ↪ 'motionAttitudeReferenceFrame(txt)',
                'motionQuaternionX(R)', 'motionQuaternionY(R)', '
                    ↪ motionQuaternionZ(R)', 'motionQuaternionW(R)',
                'motionGravityX(G)', 'motionGravityY(G)', 'motionGravityZ(G)
                    ↪ ', 'motionMagneticFieldX(T)',
                'motionMagneticFieldY(T)', 'motionMagneticFieldZ(T)', '
                    ↪ motionHeading()',
                'motionMagneticFieldCalibrationAccuracy(Z)', '
                    ↪ activityTimestamp_sinceReboot(s)', 'activity(txt)',
                'activityActivityConfidence(Z)', 'activityActivityStartDate(
                    ↪ txt)', 'pedometerStartDate(txt)',
                'pedometerNumberofSteps(N)', 'pedometerAverageActivePace(s/m
                    ↪ )', 'pedometerCurrentPace(s/m)',
                'pedometerCurrentCadence(steps/s)', 'pedometerDistance(m)',
                    ↪ 'pedometerFloorAscended(N)',
                'pedometerFloorDescended(N)', 'pedometerEndDate(txt)', '
                    ↪ altimeterTimestamp_sinceReboot(s)',
                'altimeterReset(bool)', 'altimeterRelativeAltitude(m)', '
                    ↪ altimeterPressure(kPa)', 'batteryState(N)',
                'batteryLevel(R)', 'deviceID(txt)']

wCols = ['locationTimestamp_since1970(s)',
         'accelerometerAccelerationX(G)', 'accelerometerAccelerationY(G)',
             ↪ 'accelerometerAccelerationZ(G)',
         'motionRotationRateX(rad/s)', 'motionRotationRateY(rad/s)', '
             ↪ motionRotationRateZ(rad/s)',
         'motionHeading()',
         'batteryLevel(R)']

lCols = ['locationTimestamp_since1970(s)',
```

```
        'accelerometerAccelerationX(G)', 'accelerometerAccelerationY(G)',
           ↪ 'accelerometerAccelerationZ(G)',
        'gyroRotationX(rad/s)', 'gyroRotationY(rad/s)', 'gyroRotationZ(rad
           ↪ /s)', 'batteryLevel(Z)', 'label(N)']


folder = './data7/'
p2TimeStamp = 'locationTimestamp_since1970'
wTimeStamp = 'locationTimestamp_since1970(s)'


def getCols(k):
    return p1Cols if k[0:2] == 'p1' else wCols if k[0] == 'w' else p2Cols
       ↪ if k[0:2] == 'p2' else lCols


def readData(path):
    res = {}
    dataDir = os.listdir(path)

    for fileName in dataDir:
        if os.stat(os.path.join(path, fileName)).st_size == 0:
            continue
        if fileName[0] != 'p' and fileName[0] != 'w' and fileName[0] != 'l'
           ↪ :
            continue
        key = fileName[0:2]
        hf(key)
        curFile = os.path.join(path, fileName)
        print(getCols(key))
        csvContent = pd.read_csv(curFile,
                                 usecols=getCols(key))
        print(list(csvContent))
# print("============" + key + "========" + str(len(list(csvContent))))
        res[key] = csvContent
    return res
```

```
rawData = readData(folder)


def visualizeData(rawData):
    hf("Visualize␣Data")
    for key in rawData.keys():
        print(key + ":␣" + str(len(rawData[key])))
        display(HTML(rawData[key].head(3).to_html()))


visualizeData(rawData)


def getCurrentTimeStampHeader(k):
    # return pTimeStamp if k[0] == 'p'else wTimeStamp if k[0] == 'w' else
    ↪  'NA'
    return p2TimeStamp if k[0:2] == 'p2'else wTimeStamp


def findTimeInterval(rawData):
    hf("Find␣global␣time␣interval")
    maxStart = 0.0
    minEnd = float("inf")
    for key in rawData.keys():
        maxStart = max(maxStart, rawData[key][0:1]
                       [getCurrentTimeStampHeader(key)].values[0])
        minEnd = min(minEnd, rawData[key][-2:-1]
                     [getCurrentTimeStampHeader(key)].values[0])
    print("start␣Time:␣" + str(maxStart))
    print("end␣Time:␣" + str(minEnd))
    return maxStart, minEnd


start, end = findTimeInterval(rawData)


def preProcessingDiscardGarbageTime(rawData, startTime, endTime):
    hf("Disgard␣out␣of␣bound␣time")
```

```
    for key in rawData.keys():
        df = rawData[key]
        df.drop(df[df[getCurrentTimeStampHeader(key)]
               <= startTime + 1].index, inplace=True)
        df.drop(df[df[getCurrentTimeStampHeader(key)]
               >= endTime - 1].index, inplace=True)
        df.drop(columns=[getCurrentTimeStampHeader(key)], inplace=True)


preProcessingDiscardGarbageTime(rawData, start, end)


visualizeData(rawData)


singleMap = {}


def calSingle(df):
    res = []
    res = np.append(res, df.mean().to_numpy())
    res = np.append(res, df.max().to_numpy())
    res = np.append(res, df.min().to_numpy())
    res = np.append(res, df.std().to_numpy())
# If the accuracy is not that high enough, consider adding energy, and
    ↪ other features in
    return res


# Pre process to calculate features, and save as key-features.csv
def preProcessingCalculateFeatures(rawData):
    X = []
    Y = []
    endFlag = False
    i = 127
    while endFlag == False:
        i += 1
        if i % 64 != 0:
```

```
            continue
        cur = []
        for key in rawData.keys():
            df = rawData[key]
            if(i >= len(df)):
                endFlag = True
                break
            skipX = 0
            if key == "la":
                curLa = df['label(N)'].iloc[i]
                if curLa == 0:
                    skipX = 1
                    break
                Y.append(curLa)
                continue
            cur = np.append(cur, calSingle(df[i-128: i]))

        if (endFlag != True) and (skipX == 0):
            X.append(cur)
    return X, Y


X, Y = preProcessingCalculateFeatures(rawData)


def saveCSV():
    np.savetxt(folder+"X.csv", X, delimiter=",")
    np.savetxt(folder+"Y.csv", Y, delimiter=",")


print(type(X))
print(len(X))
print(len(Y))
print(Y)


saveCSV()
```

```
warnings.filterwarnings("ignore")


classifiers = [
    KNeighborsClassifier(5),
    SVC(kernel="rbf"),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    GaussianNB(),
    RidgeClassifier(),
    LogisticRegression(max_iter=200)
]


def evalRes(X, Y):
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=0.33, random_state=12)

    f_score(X_train, X_test, y_train, y_test)
    estimators = [
        ('RFC', RandomForestClassifier(n_estimators=500, random_state=42)),
        ('KNC', KNeighborsClassifier(5)),
        ('DTC', DecisionTreeClassifier()),
        ('SVC', SVC(kernel="rbf")),
        ('RC', RidgeClassifier()),
    ]
    clf = StackingClassifier(
        estimators=estimators,
        final_estimator=GradientBoostingClassifier()
    )
    s = time.time()
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    e = time.time()
    print(f"time␣consumed:␣{round(e-s,3)}" + "␣for␣stacking␣classifier")
    print(f1_score(y_true=y_test, y_pred=y_pred, average="macro"))
# print(clf.predict(X))
```

```
    cf_matrix = confusion_matrix(y_test, y_pred)
# sns.heatmap(cf_matrix, annot=True)
    print("===========")
    sns.heatmap(cf_matrix/np.sum(cf_matrix), annot=True,
                fmt='.2%', cmap='Blues')
    return clf


def f_score(X_train, X_test, y_train, y_test):
    for clf in classifiers:
        s = time.time()
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        f = f1_score(y_true=y_test, y_pred=y_pred, average="macro")
        e = time.time()
        print(
            f"Score: {round(f,3)} \t Time(in secs): {round(e-s,3)} \t
                ↪ Classifier: {clf.__class__.__name__}")
        print("Acc:" + str(accuracy_score(y_test, y_pred)))
        print("Prec:" + str(precision_score(y_test,
              y_pred, average='macro', zero_division=1)))
        print("Recall:" + str(recall_score(y_test,
              y_pred, average='macro', zero_division=1)))


# Multi agent
clf = evalRes(X, Y)


def optimizeDutyCycleEval(clf, Y, full_Y_pred):
    skip = 1
    prevStat = None

    cur = 3

    newY_pred = []
    cnt = 0
    skipmax = 0
```

```
    for rec in full_Y_pred:
        if cur < skip:
            newY_pred.append(prevStat)
            cur += 1
            cnt += 1
            continue
        else:
            cur = 0
            if rec == prevStat:
                skip += 1
            else:
                skip = skip // 2 + 1

            skipmax = max(skipmax, skip)
            newY_pred.append(rec)
            prevStat = rec

    print(len(Y))
    print(cnt)
    print(skipmax)
    print(f1_score(y_true=Y, y_pred=newY_pred, average="macro"))


optimizeDutyCycleEval(clf, Y, clf.predict(X))


def lowIntervalEval(clf, Y, full_Y_pred):
    skip = 10
    prevStat = None

    cur = 100

    newY_pred = []
    cnt = 0
    skipmax = 0
    for rec in full_Y_pred:
        if cur < skip:
            newY_pred.append(prevStat)
```

```
            cur += 1
            cnt += 1
            continue
        else:
            cur = 0
            newY_pred.append(rec)
            prevStat = rec

    print(len(Y))
    print(cnt)
    print(skipmax)
    print(f1_score(y_true=Y, y_pred=newY_pred, average="macro"))


lowIntervalEval(clf, Y, clf.predict(X))


# Pre process to calculate features, and save as key-features.csv
def calSingleFunc(rawData):
    # X= np.array([], dtype=np.float64)
    X = []
    Y = []
    endFlag = False
    i = 127
    while endFlag == False:
        i += 1
        if i % 64 != 0:
            continue
        cur = []
        for key in rawData.keys():
            df = rawData[key]
            if(i >= len(df)):
                endFlag = True
                break
            if key == "la":
                Y.append(df['label(N)'].iloc[i])
                continue
            if key != "":
```

```
                continue
            cur = np.append(cur, calSingle(df[i-128: i]))

        if(endFlag != True):
            X.append(cur)
    return X, Y


sinX, sinY = calSingleFunc(rawData)

# Single agent
print(len(sinX))
print(len(sinX[0]))
clf = evalRes(sinX, sinY)
```