2017

Wearable Computing: Accelerometer-Based Human Activity Classification Using Decision Tree

Chong Li
Utah State University

WEARABLE COMPUTING: ACCELEROMETER-BASED HUMAN ACTIVITY CLASSIFICATION USING DECISION TREE

by

Chong Li

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

_____	_____
Xiaojun Qi, Ph.D. Major Professor	Kyumin Lee, Ph.D. Committee Member
_____	_____
Haitao Wang, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2017

ABSTRACT

Wearable Computing: Accelerometer-Based Human Activity Classification Using Decision Tree

by

Chong Li, Master of Science

Utah State University, 2017

Major Professor: Xiaojun Qi, Ph.D.
Department: Computer Science

This study focused on the use of wearable sensors in human activity recognition and proposes an accelerometer-based real-time human activity recognition approach using the decision tree as the classifier. We aimed to create an approach that requires only one accelerometer to be worn on the user's wrist and recognizes activities in real-time based on the acceleration data. The decision tree was adopted as the classification algorithm and a classifier simplification technique and a novel decision tree storage structure were designed. Feature selection and tree pruning were applied to reduce the decision tree complexity. With this approach, the designed system has fairly low computational cost and consumes small memory space, and therefore can be easily implemented to a wristband or smart watch that has an embedded accelerometer.

The proposed approach follows a process of feature extraction, feature selection, decision tree training, and decision tree pruning. We categorized human daily activities into three activity states, including stationary, walking, and running. Through

experiments, the effects of feature extraction window length, feature discretization intervals, feature selection, and decision tree pruning were tested. On top of this process, we also implemented misclassification correction and decision tree simplification to improve classification performance and reduce classifier implementation size. The experimental results showed that based on the particular set of data we collected, the combination of 2-second window length and 8 intervals yielded the best decision tree performance. The feature selection process reduced the number of features from 37 to 7, and increased the classification accuracy by 1.04%. The decision tree pruning slightly decreased the classification performance, while significantly reducing the tree size by around 80%. The proposed misclassification mechanism effectively eliminated single misclassifications caused by interruptive activities. In addition, with the proposed decision tree simplification approach, the trained decision tree could be saved to three arrays. The implemented decision tree could be initiated simply by reading configurations from the three arrays.

(60 pages)

PUBLIC ABSTRACT

Wearable Computing: Accelerometer-Based Human Activity Classification Using Decision Tree

Chong Li

In this study, we designed a system that recognizes a person's physical activity by analyzing data read from a device that he or she wears. In order to reduce the system's demands on the device's computational capacity and memory space, we designed a series of strategies such as making accurate analysis based on only a small amount of data in the memory, extracting only the most useful features from the data, cutting unnecessary branches of the classification system, etc. We also implemented a strategy to correct certain types of misclassifications, in order to improve the performance of the system.

We categorized a person's daily activities into three activity states, including stationary, walking, and running. Based on data collected from five subjects, we trained a classification system that provides an activity state feedback every second and yields a classification accuracy of 94.82%. Our experiments also demonstrated that the strategies applied to reduce system size and improve system performance worked well.

ACKNOWLEDGMENTS

I would like to give my special thanks to my major professor, Dr. Xiaojun Qi, for giving me the opportunity of participating in this study, as well as her continuous support, generous guidance, and great expertise that helped me complete this study. I enjoyed working with her and without her patience and understanding, I would not have made it this far.

I am grateful to my committee members, Dr. Haitao Wang and Dr. Kyumin Lee for their support and assistance in completing this thesis.

I would also like to express my gratitude to the Alcatel team of TCL Corporation for making available to me the accelerometer data used in this study.

I duly acknowledge the help, direct or indirect from the Computer Science department during my study here at Utah State University.

Finally and most importantly, huge thanks to my husband Guang for his endless support and especially to our baby Tony who was born in Logan and brought tons of joy to my life.

Chong Li

CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

Human activity recognition has been widely studied in recent years, mostly because of its important role in healthcare technologies. With one or more sensors and a computing device, human activities can be recognized in real life settings. This greatly helps with the design of smart homes [1], post-surgery rehabilitation at home [2], healthcare for elderly people [2], etc. A more common use of human activity recognition is daily activity monitoring, especially for fitness training. Many commercially available products provide such uses and are worn at different places, mostly on the wrist (Garmin Vivosmart HR, Casio WSD-F10, Samsung Gear Fit 2), and some on the foot (Kinematic Tune), hand (Zepp Golf 2), head (XMetrics Pro), or body (Fitbit Zip) [3].

Two kinds of sensors are generally used for human activity recognition. Environmental sensors, such as cameras [4] and depth sensors [5], are used to track a person's motion, location, and object interaction, usually in a smart house or for rehabilitation purposes. Wearable sensors [6], such as accelerometers, are usually attached to a person's body to track the motion, location, temperature, etc. Both approaches have been demonstrated effective in various studies.

This study focuses on the use of wearable sensors in human activity recognition. In existing studies, accelerometer data have been used to recognize relatively simple and common daily activities, including standing, walking, and jogging [7]–[9], as well as more complex daily activities such as cleaning, cooking, washing hands and so on [9].

Those studies generally adopt a similar approach of supervised machine learning. One or multiple classifiers are trained with features extracted from annotated data collected by one or more accelerometers worn by the participants. Two factors, features and classifiers, distinguish those studies from each other.

In this study, we propose the use of decision tree in the real-time classification of several human activities based on data collected from accelerometers. The next section discusses the related work that has been done on this topic, and describes the technique that will be used in this study, as well as the contributions of this work. Section 3 details the approach and Section 4 presents the experiments designed to evaluate the proposed approach and analyzes the experimental results. Section 5 concludes this study.

CHAPTER 2

RELATED WORK

Kwapisz et al. [7] studied the recognition of activities including walking, jogging, climbing up/down stairs, sitting, and standing based on accelerometer data from an Android phone worn by a user in his/her front pants leg pocket. Data were collected from 29 subjects at a sampling frequency of 20 Hz. The collected data were divided into 10-second non-overlapping windows and 43 features were extracted from each window. The features are variants of six basic features along three axes including mean, standard deviation, average absolute difference, average resultant acceleration, time between peaks, and binned distribution. With the 4526 samples extracted from the collected data, ten-fold validations were performed by using three classification algorithms separately. The three algorithms are J48, logistic regression, and multi-layer perception. The overall accuracies for the aforementioned three algorithms are 85.1%, 78.1% and 91.7%, respectively. However, each algorithm performs inconsistently when recognizing different activities.

Anguita et al. [8] used SVM (Support Vector Machine) to recognize activities including standing, walking, laying, sitting, walking upstairs, and walking downstairs. They collected 3-axial acceleration data and angular velocity data from the accelerometer and gyroscope embedded in an Android phone at a sampling rate of 50 Hz. Data were collected by 30 subjects carrying the Android phone on his/her waist. After noise filtering, 17 features were extracted from the data with a 2.56-second sliding window and a 50% overlapping. Those features include mean, standard deviation, signal magnitude area,

entropy, signal-pair correlation, etc. of both accelerometer and gyroscope data. The multi-class hardware-friendly SVM they proposed achieved a classification accuracy of 89%. The approach requires less memory, processor time, and power consumption, while the use of gyroscope data and the noise filtering step added complexity to the design.

Dernbach et al. [9] tried to recognize simple activities including biking, climbing stairs, driving, lying, running, sitting, standing, and walking, as well as complex activities including cleaning, cooking, medication, sweeping, washing hands, and watering plants from accelerometer data. They collected acceleration data from 10 participants, each wearing an Android smartphone at no predetermined location or orientation. Raw data were collected at the sampling rate of 80 Hz, and then features were extracted with sliding windows of 1, 2, 4, 8, 12, or 16 seconds. The features are mean, min, max, standard deviation, zero-cross, and correlation of the accelerometer data in three axes. They used six classifiers including multilayer perception, Naïve Bayes, Bayesian network, decision table, best-first tree, and K-star algorithms to classify the activities. For simple activities, all activities, and complex activities, all algorithms (except for Naïve Bayes) reached accuracies of over 90%, 70%, and 45%, respectively. They also concluded that the window length has little effect on the accuracy for simple activities. Meanwhile, when window sliding is not used, recognizing complex activities, which has rarely been done, could achieve an accuracy of 78%. However, although not stated by the authors, the performance improvement in recognizing complex activities may compromise the performance of recognizing simple activities as not using a sliding window significantly reduces the number of training samples. Nevertheless, the system has high demand on the

phone's power usage, which restrains its implementation.

Bayat et al. [10] combined six classifiers to recognize daily activities including slow walking, fast walking, running, stairs-up, stairs-down, and aerobic dancing. From accelerometer data collected by smartphones worn by four participants in hands or pockets, features including mean, standard deviation, RMS (Root Mean Square), correlation, difference, etc. were extracted. By using the combined probability determined by six classifiers (multilayer perception, SVM, random forest, LMT (Logistic Model Tree), simple logistic, and Logit boost), an accuracy of 91.15% was obtained. The combination of a number of classifiers is a novel design. However, the system is very complex as it uses complicated features and requires several algorithms to be implemented.

Zhang et al. [11] categorized daily living activities into four categories including walking, running, household, or sedentary activities, and developed methods to recognize them based on raw acceleration data from the GENEA (Gravity Estimator of Normal Everyday Activity). They also compared the classification accuracies from a wrist-worn GENEA and a waist-worn GENEA. Sixty participants, each wearing three accelerometers (one at the waist, one on the left wrist, and one on the right wrist), completed an ordered series of 10-12 semi-structured activities in laboratory and outdoor environments. Features obtained from both FFT (Fast Fourier transform) and DWT (Discrete Wavelet Transform) were extracted, and machine learning algorithms were used to classify the four types of daily activities. With their proposed approach, they were able to reach high overall classification accuracy for both waist-worn GENEA (99%) and wrist-worn

GENEAs (right wrist: 97%; left wrist: 96%).

Mannini et al. [12] replicated the algorithm of Zhang et al. [11] and tested it on a dataset with 33 participants performing a set of daily physical activities. Various combinations of window lengths (i.e., the amount of data acquired to give a single classification output) and feature sets (sets of variables used for classification purposes) were tested to develop an algorithm. With a 4-second window length and the same features as those in the study of Zhang et al., the algorithm yielded an accuracy of 84.2% for wrist data. The study validated the feasibility of the design of Zhang et al.

Gao et al. [13] proposed an activity recognition approach that requires multiple sensors to be worn on distributed body locations. They designed a distributed computing-based sensor system to run “light-weight” signal processing algorithms on multiple computational efficient nodes to achieve higher recognition accuracy. Through comparison of six decision tree-based classifiers employing single or multiple sensors, they proposed a multi-sensor system consisting of four sensors that can achieve an overall recognition accuracy of 96.4% by adopting the mean and variance features. They further evaluated different combinations of sensor positioning and classification algorithms. However, wearing multiple sensors on the subject’s body restrains the design from being adopted in a daily life setting.

Some studies designed user-specific classifiers. A user’s activity data were collected first to train a classifier, which was then used to classify the user’s future activities. In this way, a real-time monitoring is realized. In the study of Brezmes et al. [14], a kNN (k-Nearest Neighbors) algorithm is used to classify activities including

walking, climbing up stairs, climbing down stairs, sitting-down, standing-up, and falling. Accuracies of 70% to 90% were reached for the six activities.

Many probability-based algorithms have been used for activity recognition. For example, Kwapisz et al.'s study adopted the decision tree and Brezme et al.'s study used kNN. In the study of Dernbach et al., six different algorithms were adopted and resultant accuracies were compared. The compared algorithms include multilayer perception, naïve Bayes, Bayesian network, decision table, best-first tree, and K-star. Little difference showed among the different algorithms' accuracies. Bayat et al. combined several algorithms together for classification.

Although the topic has been extensively studied, there is still more to explore. In this study, we propose a real-time single accelerometer-based activity recognition approach that makes the following contributions:

- Requiring only one accelerometer instead of multiple sensors worn on the subject's wrist (left or right) to increase portability, reduce cost, and broaden the applications.
- Recognizing activities in real-time without requiring user-specific classifier training.
- Adopting the decision tree as the classification algorithm and designing decision tree simplification technique to store the trained decision tree in fairly small memory.
- Reducing the complexity of the decision tree by applying feature selection and tree pruning and therefore allowing the system to have low computational cost

and consume small memory size.

- Studying the effects of window length, feature discretization, feature selection, and decision tree pruning on the activity recognition performance and providing insightful information for future studies.

CHAPTER 3

METHODOLOGY

A four-step approach is designed in this study for the activity recognition task. Three kinds of activity states including stationary, walking, and running, are recognized from accelerometer data. The four steps are data collection and preprocessing, feature extraction, feature selection, and classifier learning as summarized below:

- Data collection and preprocessing. Data are collected at the sampling frequency of 31.5 Hz in a controlled manner. Five subjects are supervised to perform different activities and the recordings are annotated after collection. Data preprocessing is then performed to remove the noisy data collected at the beginning and towards the end of each collection process to ensure valid data are used to train the classifier.
- Feature extraction. Based on analysis of data and review of related work, 37 features, newly developed or previously published, are selected and extracted from the raw accelerometer data. The data recordings are divided into windows of certain lengths, and a set of features is extracted from each window and labeled. Each two consecutive windows overlap by a half of the window length.
- Feature selection. Feature selection aims to reduce the number of features so the complexity of the classifier will be reduced and the recognition accuracy will be improved. A two-step approach is adopted. First, a random forest-

based R package, Boruta, is used to rank the features. Then, a sequential feature selection (SFS) algorithm is performed on the features that are marked important by the Boruta package. Based on the feature selection result, we determine the optimal feature subset for the classifier.

- Classifier learning. A simple and efficient algorithm, the decision tree, is used to learn a classifier for activity recognition. A TDIDT (Top-Down Induction of Decision Trees) process is used to train an ID3 decision tree, which uses information gain to decide the splitting criteria. A simple structure is designed to compactly store the trained decision tree in small memory spaces. The reduced error pruning strategy is also used in the tree pruning process to reduce the complexity of the decision tree and improve the activity recognition accuracy. A misclassification correction mechanism is also employed to improve classification performance.

For each step, we perform investigation and evaluate potential approaches to find a tradeoff between recognition accuracy and classifier complexity. In the following subsections, each step is explained in detail.

3.1 Data Collection and Preprocessing

3.1.1 Data collection

The data collection is conducted by using a prototype TCL Watch. Five participants, each wearing two watches on the left and right wrists, perform 13 daily activities and categorize them into three activity states, namely, stationary, walking, and

running. Table 3-1 summarizes the activities performed by the participants. As the subjects perform activities, accelerometer data are collected and annotated. Data collected from the left wrist and data collected from the right wrist are treated as two individual sets of data. In other words, only one accelerometer is needed in final implementation.

The data are collected at a sampling frequency of 31.5 Hz, which means 31.5 data points are collected per second. Each data point contains a timestamp and three values, which correspond to the acceleration along the x-axis (horizontal movement), y-axis (upward/downward movement), and z-axis (forward/backward movement), respectively. Fig. 3-1 shows some representative plots of each state. Eight seconds of data is shown in

Table 3-1 Accelerometer data collection

State	**Activity**	**Details**	**Left/Right wrist**
Stationary	Standing	5 minutes without doing anything	both
	Answering phone	5 minutes, standing and talking on the phone	both
	Typing	5 minutes, sitting and typing	both
	Writing	5 minutes, sitting and writing	dominant hand
	Reading	5 minutes, sitting and reading	both
	Drinking	5 minutes, sitting and drinking water	dominant hand
	Eating	5 minutes, sitting and eating	dominant hand
Walking	Slow walking	5 minutes at slower than 1 m/s	both
	Normal walking	5 minutes at about 1.4 m/s	both
	Fast walking	5 minutes at faster than 2 m/s	both
Running	Slow running	5 minutes at about 2 m/s	both
	Normal running	400 meters at about 4 m/s	both
	Fast running	100 meters fast run	both

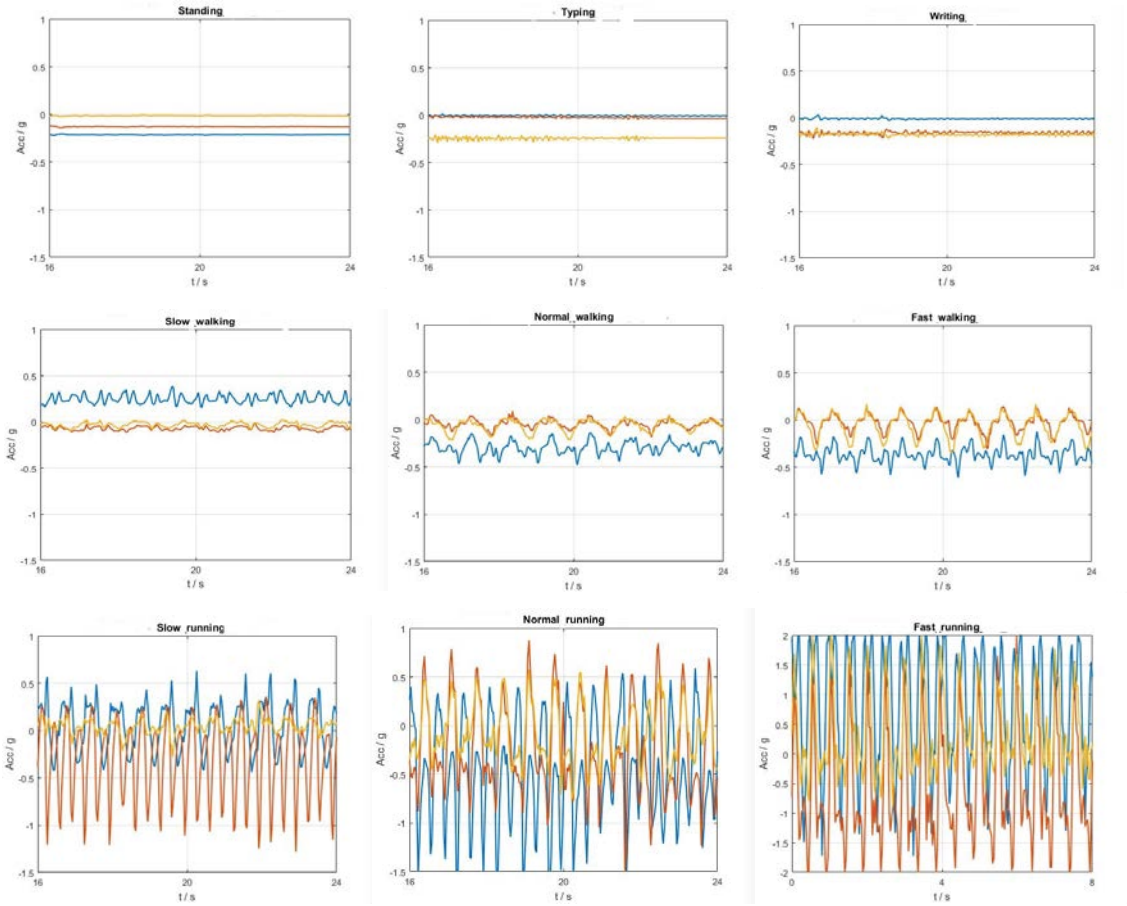

Fig. 3-1: Examples of raw accelerometer signals (each showing an 8-second segment of data) in three axes for different daily activities.

each plot. The value range of the collected acceleration is [-2, 2], whereas the value range of [-1.5, 1] is used in the plots (except for fast running) to clearly reflect the repetitive motions in the walking state and the small fluctuations in the stationary state.

3.1.2 Data preprocessing

In order for the final implementation of the system to be able to process real-time accelerometer data without performing massive computation, signal preprocessing is not

designed in this study. However, a simple procedure is performed to eliminate the potential annotation errors during data collection. For example, when a subject is collecting fast running data, he might need two seconds to activate both collectors on his wrists; after he stops running, he would also need a few second to calm down and deactivate the collectors. Such a process inevitably produces noise at the beginning and towards the end of the data collection process. As a result, the features extracted from those noisy portions of data cannot correctly reflect the annotated activity state. In order to have correct data for classifier learning, the first 5 and the last 5 seconds of data are truncated from each recording.

3.2 Feature Extraction

Since the collected raw data are time-series data, we cannot train or run classification algorithms directly on those data. Therefore, the raw data are divided into segments of a specific length and then informative features are extracted from each segment for classifier training.

3.2.1 Sliding windows

The raw accelerometer data are broken into windows of the same duration in order to capture its characteristics. As seen in Fig. 3-1, the accelerometer data of most activities show repetitive patterns. In each window, there should be enough repetitions of motion to distinguish different activities. Meanwhile, since we develop a real-time classifier, the time between each two classifications (feedbacks) should be reasonable for

users to monitor their activities. Consecutive windows overlap by a half of the window length. This means each data point contributes to two windows. This strategy, on one hand, yields more useful training data. On the other hand, it benefits the misclassification correction mechanism adopted in the classification stage.

The length of each window is a significant factor influencing our classifier performance. Having a smaller window length means fewer motion repetitions are included in each window, which may result in lower classification accuracy, while the feedback is more frequent during real-time monitoring. Meanwhile, having a larger window length means more motion repetitions are included in each window, while the feedback frequency may not be satisfying. A trade-off between classification accuracy and feedback frequency must be found. In order to determine the optimal window length, we experiment with the lengths of 2, 4, 6, and 8 seconds. Based on the experimental result, a window length is determined for the subsequent steps.

3.2.3 Feature extraction

Various features are used in existing works. They typically are extracted from either the frequency domain or the time domain or both. Most of the features used in this study are extracted from the time domain to reduce the computational cost and make the activity recognition system real time.

A variety of features, newly developed or previously published, are extracted. Table 3-2 lists the 37 features used in this study together with their feature IDs. Each feature is extracted for *x*-, *y*-, and *z*-axes, except for the simplified RMS feature, which is

Table 3-2 Features extracted from tri-axial accelerometer data

Feature	**Axis**	**Feature (Variable) ID**	**Description**	**References**
Average Increment	X, Y, Z	V1, V2, V3	The average absolute increment (increase or decrease) of acceleration values from one data point to the next within the window	
Standard Deviation	X, Y, Z	V4, V5, V6	The standard deviation in the accelerations of each axis within the window	[7]–[9], [15]
Mean	X, Y, Z	V7, V8, V9	The mean acceleration within the window	[7]–[9], [15]
Simplified RMS (Root Mean Square)	X, Y, Z, M	V10, V11, V12, V13	RMS are replaced with the absolute values The M-axis is a virtual combination of the three axes	
Binned distribution	X, Y, Z	V14-V18, V19-V23, V24-V28	The number of acceleration values falling into each one of the five bins of each axis	[7]
Mean-Cross	X, Y, Z	V29, V30, V31	The number of mean-crossings [3], a variant of zero-crossing	
Pairwise Correlation	X-Y, Y-Z, X-Z	V32, V33, V34	The pairwise correlations between the three axes	[8], [9], [15]
Simplified energy	X, Y, Z	V35, V36, V37	The sum of the squared acceleration values	[15]

also extracted for a virtual axis *m*, a combination of the three axes. The features without any references are the features developed by ourselves.

Assuming that each window contains *n* data points, and each data point is a tuple (x_i, y_i, z_i) $(1 \leq i \leq n)$, we can calculate the features as follows (The calculations for the *x*-axis are presented as examples).

a. The average increment (AveInc) feature describes the absolute difference between each two data points, and it is designed to capture the intensity of each axis'

b. The standard deviation (*SD*) feature is one of the most commonly used features in

machine learning. It quantifies the variation of the data. It is calculated by Equation (2).

$$SD_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \quad (2)$$

c. The mean is also one of the most commonly used features in machine learning to describe the expected value of the data. It is calculated by Equation (3).

$$Mean_x = \frac{\sum_{i=1}^{n} x_i}{n} \quad (3)$$

d. Typically, root mean square (*RMS*) is defined as the square root of the arithmetic mean of the squares of a set of numbers (Equation (4)). Here, we use a simplified version of root mean square to reduce the amount of computation. The arithmetic mean of the absolute values of the series of data is used, as Equation (5) shows (it is still denoted as RMS). The RMS for the virtual m-axis is calculated by Equation (6).

$$RMS_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n} {x_i}^2} \quad (4)$$

$$RMS_x = \frac{\sum_{i=1}^{n}|x_i|}{n} \quad (5)$$

$$RMS_m = \frac{\sum_{i=1}^{n}\frac{1}{3}(|x_i|+|y_i|+|z_i|)}{n} \quad (6)$$

e. The binned distribution is used to describe the value of distribution of each axis. The value range of acceleration [-2, 2] is divided into five ranges, [-2, -1.2), [-1.2, -0.4), [-0.4, 0.4), [0.4, 1.2), and [1.2, 2.0]. For each axis, the number of values that fall in each range is counted.

f. Zero-crossing is often used in image processing for edge detection or gradient

filtering. For a mathematical function, when its graph crosses the axis (zero value), it is called a zero-crossing point. In this study, we use a variant of zero-crossing during feature extraction. For each axis, the mean in a window is first calculated. Then, the number that the values cross the mean is counted. This feature is called mean cross (*MC*). Since the mean is also used as a feature, counting the mean-crossings adds little computational complexity to the algorithm.

g. The pairwise correlation (*Corr*) is used to capture the correlation between the data points of each pair of axes. Specifically, it computes the correlation between data points along *x*- and *y*-axes, along *x*- and *z*-axes, and along *y*- and *z*-axes. It is calculated by Equation (7), where $Corr_{xy}$ is the correlation between *x*- and *y*-axes. The correlation between *x*- and *z*-axes and between *y*- and *z*-axes can be computed similarly by replacing the data points at the corresponding axis.

$$Corr_{xy} = \frac{Cov_{xy}}{SD_x * SD_y} \tag{7}$$

$$Cov_{xy} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \tag{8}$$

h. Simplified energy is a simplified version of energy to measure the total of the magnitude of the power spectrum of the data. Originally, energy is the sum of the squared discrete FFT component magnitudes of the signal [15]. However, performing FFT transformation of the accelerometer data means massive computation, which is not desired in this study. Therefore we use a simplified version of this feature without performing FFT transformation.

$$Energy_x = \sum_{i=1}^{n} {x_i}^2 \tag{9}$$

From the acceleration data points (e.g., 60 data points when using 2-second

windows) in each window, one record is obtained, consisting of 37 values of the extracted features and a class label that marks the activity state that this record reflects. After the feature extraction, feature selection is performed to reduce the number of features in order to further reduce the complexity of the classifier. We use the feature selection approach presented in Section 3.3 to keep the most effective features in distinguishing the three activity states, i.e. stationary, walking, and running.

3.2.3 Feature discretization

Since both the extracted feature data and the selected feature data after applying feature selection are continuous data with hundreds of distinct values for each feature, discretization methods may need to be applied to convert the continuous data to discretized (i.e. categorical) data depending on the chosen classification method. Some classification algorithms, such as C4.5 decision tree, are able to work on continuous data by discretizing them during the algorithm learning process. However, many algorithms, especially the ID3 decision tree algorithm we adopt in this study, work better on discretized training data. Studies [16], [17] have shown that classifiers construct faster and with proper optimal interval values, perform better when continuous data are discretized prior to training. Therefore, we apply a simple equal width binning technique to transform the selected continuous feature data into discrete values. For each feature, its value range is divided into k equally sized intervals and the values falling into each interval is replaced by a distinct value. Since we have dozens of features, the value of k is the same for all extracted features to reduce the computation for decision tree training as

well as the classification.

To set a base value of k, we use $k_i = \max\{1, 2\times\log(l_i)\}$ [17], where l is the number of unique observed values for the i-th feature. For our features extracted with different window sizes (each set considered separately), the maximum value of k ranges from 6 to 10. We extend this range and use the values of 5 to 11 for k. An experiment (Experiment 3) is designed to determine the optimal number of features in this range. The result of this experiment is illustrated in Section 4.

3.3 Feature Selection

For the training of a predictive model, feature selection is a crucial step. Although 37 features are extracted from the raw data, it is not ideal to use all of them for the classification for two reasons. First, some features may be irrelevant to the categorization. Second, two features may play the same role for identifying a record's class, making one of them redundant. With those irrelevant or redundant features included in the training of the classifier, the generated decision tree may have a lot of redundant branches and be over-fitted. Feature selection is adopted to solve these problems.

Generally, three types of feature selection algorithms are available [18]. They include filter methods such as Chi squared test and correlation coefficient scores, wrapper methods such as recursive feature elimination algorithm, and embedded methods such as Ridge Regression. Filter methods rank the features with importance scores and are often used as a pre-selection method. Wrapper methods attempt to find a subset of the features that yields the highest classification performance. Embedded methods include the feature

selection process in the classifier training and are specific to classifiers.

In this study, we adopt a two-step feature selection. In the first step, a filter-based feature selection approach is employed. We use the Boruta package [19], [20] in R on the labeled feature data to eliminate a portion of the unimportant features. Boruta algorithm is built around the random forest classification algorithm. The algorithm first adds shuffled copies of all features and then trains a random forest classifier on the extended feature data. Based on the maximum Z score of the shadow features (MZSF), it confirms the original features that have Z scores significantly higher than the MZSF and rejects those with significantly lower Z scores than the MZSF. The shadow features are then removed and new shadow features added to repeat this process. The algorithm stops when all features gets either confirmed or rejected or it reaches a specified limit of random forest runs.

In the second step of the feature selection, we remove the features that are marked unimportant or tentative from the labeled feature data and adopt a wrapper-based method on the new feature data. Heuristically, a wrapper approach means to examine every possible subset of the features and find the one that produces the highest classifier performance using the target classification algorithm. However, it means 2^n tests are required if n features are selected in the first step. Unless n is a really small number, this amount of tests is not ideal. To avoid this massive amount of tests, we adopt the sequential feature selection (SFS) algorithm [18] instead. The algorithm tests each of the rest features' performance using the target classification algorithm (i.e. decision tree) together with the features in a current subset. When an additional feature is added to the

current subset, a decision tree classifier is trained on the subset, the classification performance is recorded, and the feature is then removed from the current subset. This process is repeated until all the features that are not in the current subset are tested. The feature that gives the highest classification accuracy is permanently added to the subset and the algorithm moves on to the next step until the required number of features is included. In this study, the algorithm starts with an empty subset and ends until all features are added to the subset, and we analyze the accuracy change to decide the optimal subset.

In Experiment 3, the performance of the feature selection approach is tested and the result is presented in Section 4. Feature selection is performed directly on the continuous feature data extracted from the accelerometer data.

3.4 Classifier Learning

Decision tree has been a popular algorithm in machine learning. In 1979, Quinlan proposed the ID3 algorithm [21] based on Shannon's information theory (1949). The ID3 algorithm is mainly for training the decision tree from discrete attributes by using information gain to select the splitting criterion. We adopt a TDIDT strategy to learn the classifier in this study. This process is a mature methodology with efficient learning and classification on categorical attributes.

3.4.1 Tree training

The TDIDT strategy we adopt is a greedy algorithm and is by far the most

common strategy for learning decision trees from data. The source data set is split into subsets based on an attribute that is determined by using a certain purity measure. This process is repeated on each derived subset in a recursive manner. The recursion is completed when all the data in the subset at a node are in the same class or all the attributes have been used as splitting criteria, i.e. the branch cannot be split again, in which case we assign the majority class of the subset to a leaf node.

In this study, information gain is used as the purity measure to select the splitting criterion at a splitting node. If the sample is completely homogeneous, the entropy is zero; if the sample is equally divided, it has entropy of one. The attribute that carries the most information gain draws more clear boundaries among the classes and is thus used as the splitting criterion. No stop-splitting rule is set for the recursive partitioning.

To avoid bias during tree learning, the same number of samples of the three activity states is used to train the classifier. It should be noted that the input to the training is the discretized features extracted from the overlapping sliding windows.

3.4.2 Tree pruning

Tree pruning is the method to cope with one of the most common and important problems in TDIDT, namely overfitting. There are commonly two ways to prune decision trees, one is pre-pruning and the other is post-pruning. Pre-pruning methods set stopping rules to prevent redundant branches from growing, whereas post-pruning methods let the decision tree fully grow and then retrospectively remove redundant branches. In this study, we adopt a post-pruning approach. The sections of the tree that provide little or

adverse power to classifying instances are removed. By doing this, the final classifier is less complex, and also has higher predictive accuracy. Since we look for a classifier that is small sized, pruning is a crucial process in this study.

Post-pruning can be performed in either a top-down or a bottom-up fashion, and the latter is adopted in this study. Typically, a bottom up pruning starts at the leaf nodes. Since we use the leaf nodes to store the final class, the pruning starts at the lowest rightmost parent nodes. We adopt the reduced error pruning (REP) strategy [22] in our study. Specifically, a pruning set of data is used to test the performance of the decision tree as branches are being pruned. Starting at the last parent node, each parent one (i.e. a branch) is replaced with its most popular class that is denoted by the leaf node with the largest number of samples. Intuitively, if the tree's prediction performance is downgraded by the deletion of a parent node, the deletion is reversed; if not, the change is kept. This process is iterated until the left-most child of the root is processed.

3.4.3 Classifier simplification

To implement the classifier into a real-time activity monitor, the trained classifier needs to be compactly saved in the memory of the wearable device and be quickly accessed to make a decision for newly extracted features. In order to do so, a novel approach is designed to store the decision tree in files of small sizes. We include three pieces of information for the trained decision tree as arrays. As detailed below, the three arrays store the feature discretization information, each node's splitting criterion or class, and each internal node's location in the decision tree, respectively. An example of the

three arrays is shown in Fig. 3-2.

- The first array is a two-dimensional array; the first column of the array stores the smallest value of each feature and the second column stores the interval size. For example, if the i^{-th} row of the array is {*min*, *interval*}, then for the i^{-th} feature, the values falling in the range of [*min*, *min*+*k***interval*) will be replaced by a value of *k*-1 during the feature discretization process.
- The second array is a one-dimensional array storing the splitting criteria of tree nodes, which are recorded in a breadth first manner. For the i-th node, if the array stores a digit *k*, it means the node is an internal node and its branches are splitted by the value of the k-th feature. If the array stores a capital letter 'A' for a node, it means the node is a leaf and the activity state it represents is "stationary". Similarly, the letter 'B' and 'C' represents the states of "walking" and "running", respectively.

Array 1 (Feature discretization information):
{{0.001887, 0.287572},{0.002376, 0.332635},{0.003153, 0.229805},{0.001814, 0.534820},{0.002286, 0.577197},{0.003045, 0.436870},{0.002124, 0.607137},{0.003410, 0.607108},{0.003790, 0.417544}} ;

Array 2 (Decision tree reconstruction information):
{'4','6','7','C','C','C','C','C','C','A','8','B','B','A','A','A','A','8','C','C','C','C','C','C','C','3','7','B','B','B','B','B','B','C','B','B','B','B','B','B','7','B','B','B','B','B','B','B','5','A','A','A','A','A','A','A','5','5','B','B','B','B','B','B','3','A','A','A','A','A','A','A','2','B','B','B','B','B','B','B','B','2','B','B','B','B','B','B','B','2','A','A','A','A','A','A','A','1','B','B','B','B','B','B','B','1','B','B','B','B','B','B','B','1','A','A','A','A','A','A','A','0','B','B','B','B','B','B','B','0','B','B','B','B','B','B','B','0','A','A','A','A','A','A','A','A','B','B','B','B','B','B','B','B','B','B','B','B','B','B','B','B','B','A','A','A','A','A','A','A'};
Array 3 (Decision tree reconstruction information):
{0, 1, 2, 10, 17, 25, 26, 41, 49, 57, 58, 65, 73, 81, 89, 97, 105, 113, 121, 129, 137};

Fig. 3-2: An example of the three arrays storing the trained decision tree classifier

- The third array is a one-dimensional array that stores each internal node' location in the decision tree. During activity classification, the classifier can locate a node's children quickly in the second array based on its location information.

  Each internal node of the trained tree has a specific number of children, i.e. the number of intervals used during feature discretization. During classification, the classifier first reads a window length of accelerometer data, extracts features, and discretizes the feature data based on the information in the first array. Assuming *NumberOfIntervals* intervals are used during discretization and the discretized feature data is stored in a *testData* array, the classification process is as shown in Fig. 3-3.

Through this conversion of decision tree into three arrays, the classification can be performed without reconstructing a decision tree. This way, the classifier is significantly reduced and can be easily implemented to a device with fairly small memory size.

3.4.4 Classification

A decision tree can easily be transformed to a set of rules by mapping from the root node to the leaf nodes one by one. A specific set of values of the features leads to a specific class. Assuming that each feature is discretized into three intervals, Fig. 3-4 illustrates the transformation of a decision tree to a set of rules. In this decision tree, with the value of *Mean* being 0 and the value of *AveInc* being 0, no matter what values the other features (if any) have, the activity state will be classified as *Stationary*.

```
childIndex = 0;
while (Array2[childIndex] != 'A', 'B', or 'C') {
        splittingCriterion = Array2[childIndex];
        attributeValue = testData[splittingCriterion];
        for (i = 0; i < SizeOfArray3; i++) {
                if (Array3[i] = ChildIndex)
                        temp = i;
                        break;
        }
        childIndex = temp*NumberOfIntervals + 1 + attributeValue;
}
return Array2[childIndex] as activity state feedback;
```

Fig. 3-3: Activity classification from decision tree information stored in three arrays.

During the real-time monitoring/classification process, accelerometer data flows into the classifier as they are collected. As shown in Fig. 3-5, after data of the pre-defined window length arrives, features are extracted and discretized. Based on the discretized values, the activity is determined through a simple process detailed in Section 3.4.4.

3.4.5 Misclassification correction

Many scenarios of daily activities may cause misclassification. For example, when a user is running, he or she might lift his or her arm to wipe off sweat from his or

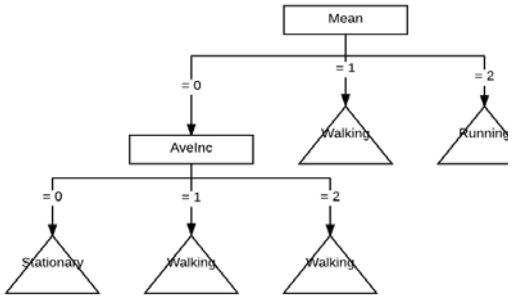

Fig. 3-4: Conversion of a decision tree (left) to a set of decision rules (right).

her forehead. At this time, the data collected by the wristband might be very different from the data collected one second before and one second after. Such interruptions of continuous motion will certainly lead to misclassification. In order to reduce this kind of misclassification, a correction mechanism is designed in this study. When a state transition occurs during monitoring, we assume that the user is still performing the previous activity, until two classifications of the same result have been made. For example, the classifier has made two classifications of the running activity, and a new classification result of a walking activity is given. In such case, the result will be corrected to be the running state. However, the result of walking is still stored and used for the next classification. In other words, for each classification, three windows are considered and the majority activity is given as the classification result.

As seen in Fig. 3-6, we aim to eliminate the misclassification cases shown in the upper scenario. With the two previous classifications of *Running* state, the classification result of *Stationary* is corrected to *Running*. In the lower scenario, a second *Stationary* state is detected after the misclassification correction. In this case, the classifier will

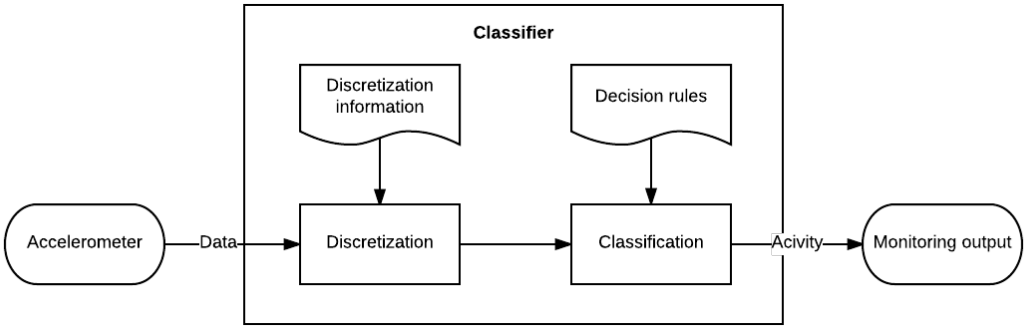

Fig. 3-5: Real-time activity monitoring process.

reckon that the user has stopped running and gives the classification result of *Stationary*. Apparently, this strategy would also cause misclassifications as we see in the lower scenario of Fig. 3-6. However, it is corrected after half a window length and we can consider it negligible.

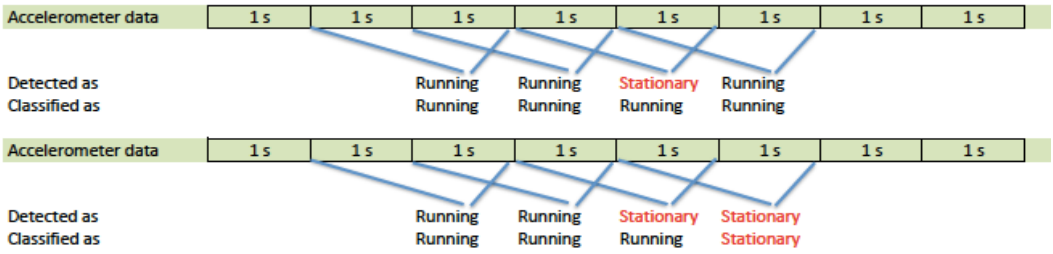

Fig. 3-6 Misclassification correction scenarios (with window lengths of 2 seconds).

CHAPTER 4

EXPERIMENTS AND RESULTS

In order to validate the proposed activity recognition approach, we design a set of experiments. These experiments test the effects of window length, the number of intervals in feature discretization, feature selection, tree pruning, and misclassification correction on the activity recognition accuracy. Through these experiments, the optimal configuration for the classifier training is determined.

4.1 Design of Experiments

The following experiments are designed to validate the proposed approach. The parameter setting of each experiment is dependent on the result of the previous one.

- The First Experiment: The window lengths of 4 seconds, 6 seconds, and 8 seconds are tested to decide the optimal window length. 37 features and various numbers of intervals are used in this experiment. The classifiers are trained without feature selection and tree pruning.
- The Second Experiment: Five to eleven intervals are tested to determine the optimal number of intervals. Various window lengths and 37 features are used in this experiment. The classifiers are trained without feature selection and tree pruning. Experiments 1 and 2 are combined as they are performed.
- The Third Experiment: The performances of two classifiers, one trained with 37 features and one trained with the feature subset selected by the adopted

feature selection approach, are compared. The window length determined in the first experiment and the number of intervals determined in the second experiment are used in this experiment. The classifiers are trained without tree pruning.

- The Fourth Experiment: The performance of the pruned decision tree is compared to the original decision tree learned from the features determined in the third experiment. The window length determined in the first experiment and the number of intervals determined in the second experiment are used in this experiment.
- The Fifth Experiment: With a decision tree trained with the parameters determined in the first four experiments, the classification performance with misclassification correction incorporated is compared with the classification performance without incorporating misclassification correction.

A 5-fold cross-validation is employed to verify the accuracy of the classifier. Since we collected data from five subjects, we intuitively divide the data into five sets, with the data collected from each subject being one dataset. In each round of the 5-fold cross-validation, one dataset is used as the testing set and the other four used as the training set (as seen in Fig. 4-1). This process is repeated five times, with each of the five subsets used once as the testing set. This way, all the collected data are used for both training and testing, whereas in each round, the test data is unseen (new) to the classifier. The classification results obtained from the five tests are put together to obtain the overall classification accuracy. Meanwhile, we process the training set and the testing set

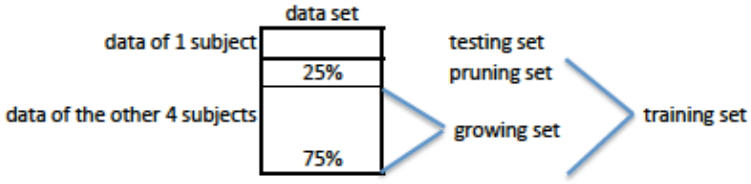

Fig. 4-1: Data set division for the experiments.

differently.

- As can be seen in Table 3-1, we collected data for a longer duration in stationary state than in walking or running states. As a result, more feature records can be extracted for stationary state than for the other two states. Fig. 4-2 shows the distributions of the feature records. For the training set, we randomly discard a portion of the feature records of each state so that the number of feature records of the three states is the same. For example, when a 4-second window length is used for feature extraction, we obtain 1400 records of each state in the training set, and when a 2-second window length is used we obtain 2876 records of each state in the training set. This is to avoid bias in the classifier learning process.
- The above balancing strategy is not applied to the testing set. In other words, the testing set contains more data of stationary state than that of walking state, and more data of walking state than that of running state, as shown in Fig. 4-2. Since in daily life, people generally stay inactive a longer time than active (walking or running), such a testing set better simulates the situation in which the classifier will be used.

Fig. 4-2 Feature records distributions (left: training set, right: test set).

- The training set consists of labeled feature data whereas the testing set consists of raw accelerometer data files. Throughout the experiments, decision trees are learned from labeled feature data. During the classification process, the raw accelerometer data files are input to the algorithm with a label of the annotated activity state and the trained decision tree performs classification on the features that are extracted from the raw data. The classification result is then compared to the activity state label to determine the classification accuracy.

For the fourth experiment that tests the effect of tree pruning, the training set is further divided into a growing set and a pruning set, as shown in Fig. 3-5. The growing set is a random 75% of the training set and the pruning set is the other 25%.

4.2 Result Analysis and Discussion

This section describes and analyzes the results of the designed experiments.

4.2.1 Experiments 1 & 2

Considering that the feature selection window size and the number of discretization intervals may have influences on each other, Experiments 1 and 2 are performed together to determine the optimal values for both parameters. Fig. 4-3 shows the results of the two sets of experiments. As discussed in Sections 3.2.1 and 3.2.3, four window lengths (2, 4, 6, and 8 seconds) and 7 interval numbers (5 to 11) are tested. Therefore, we obtain 4*7 values of classification accuracy as shown in Fig. 4-3.

As seen in Fig. 3-6, there are several combinations of window length and interval number that yield relatively high accuracies. It is apparent that the combination of 2-s windows and 8 intervals produces the highest accuracy. However, it would be imprudent to use this configuration directly. In Fig.s 4-4 and 4-5, we plot the experimental result in bar charts to have a clearer idea of the effect of each parameter alone.

In Fig. 4-4, the classification accuracies are grouped by window length to show

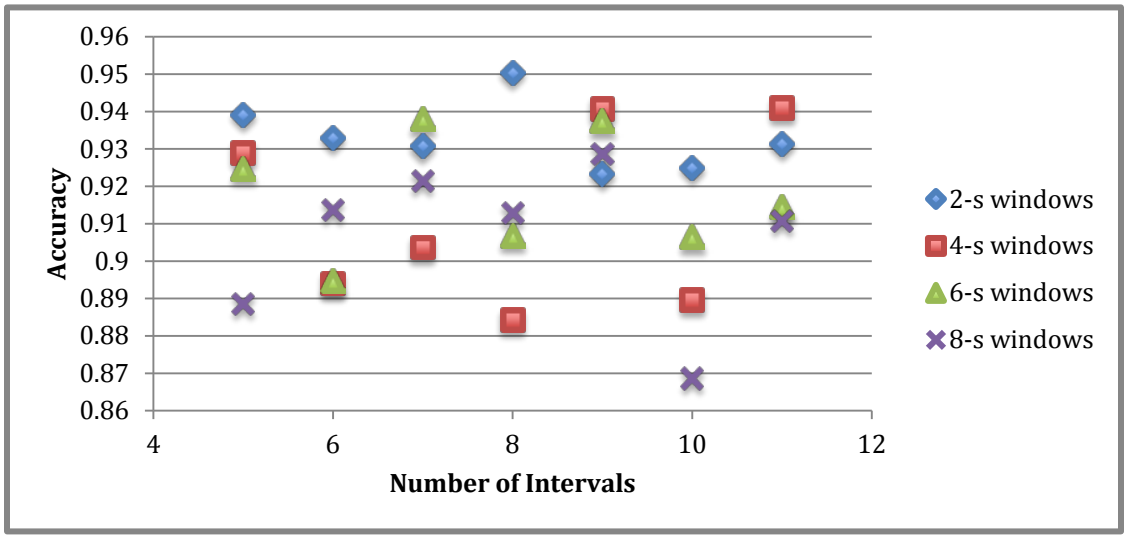

Fig. 4-3: Decision tree classification accuracies with different feature extraction

window the effect of window length regardless of interval number. As can be seen, for the window lengths of 4, 6, and 8 seconds, the classification accuracy varies greatly as the interval number changes. Meanwhile, no matter what number of intervals is used, the sizes and various numbers of discretization intervals window length of 2 seconds always results in relatively high performance. In other words, the performance of 2-s windows is more stable than those of other window lengths.

In Fig. 4-5, the classification accuracies are grouped by the number of intervals to show the effect of interval number regardless of window length. Similarly, we look for the interval number that performs stably. As can be seen, no matter what window length is used, the interval number of 9 always results in relatively high performance.

Based on Fig. 4-4 and 4-5, we know that the window length of 2 seconds and the interval number of 9 are optimal choices for the two parameters. However, as can be seen in Fig. 4-3, the combination of these two settings yields an accuracy that is lower than

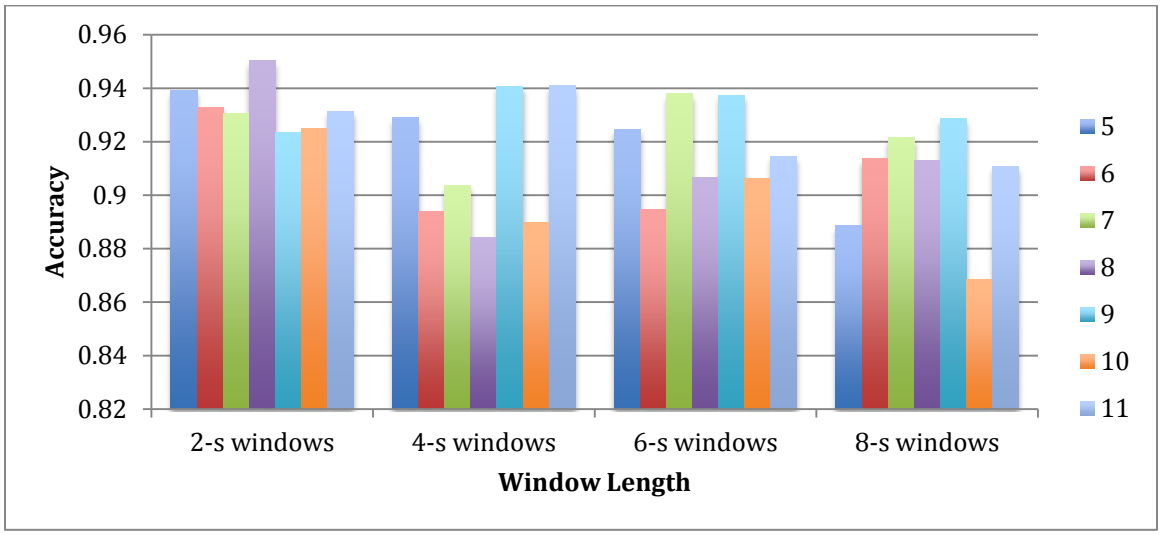

Fig. 4-4: Decision tree classification accuracies plotted by window length.

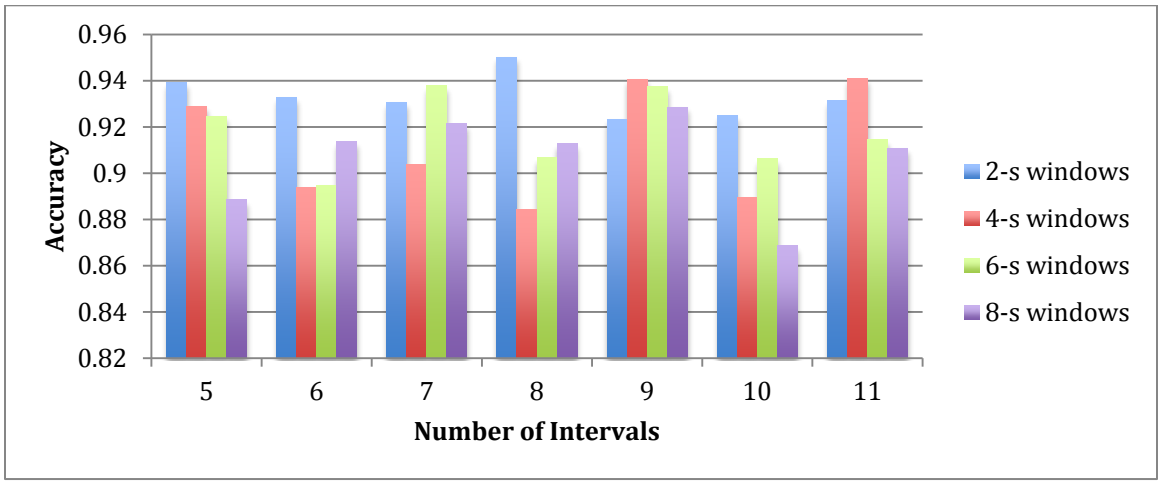

Fig. 4-5: Decision tree classification accuracies plotted by number of intervals.

those of many other combinations. Therefore, we decide to further experiment with the optimal combinations with either of these two settings. Since 8 intervals achieve the best performance when using 2-second windows and 4-second windows achieve the best performance when using 9 intervals, these two combinations are tested in subsequent experiments. In Experiments 3 and 4, we experiment with these two sets of configurations and determine the optimal choice between them based on the results.

4.2.2 Experiment 3

To reduce the number of features, simplify the trained classifier, and improve the classifier performance, we adopt a two-step feature selection approach. First we use the Boruta package in R to perform an initial selection. On this basis, the SFS algorithm is applied to find the optimal feature subset. Since we determined two optimal combinations of window length and interval number in the last experiment, two sets of tests are performed in this experiment.

Fig. 4-6 plots the importance score of each feature after applying the Boruta package on the 37 features extracted from collected accelerometer data with the window length of 4 seconds. Each feature is given an importance score as indicated in the black bar in the middle of each box together with its smallest and largest possible importance scores as indicated in the left and the right of each box, respectively. Each feature is marked as confirmed (shown in green), tentative (shown in yellow), or rejected (shown in red). It should be noted that shadow features shown in blue are not considered since they are shuffled copies of the original features. For the 4-s-window data, 10 features are marked tentative or rejected and are excluded in the second step of feature selection.

Fig. 4-7 plots the importance score of each feature for the features extracted with the window length of 2 seconds. For the 2-s-window data, 7 features are marked tentative or rejected. These 7 features, including the 1^{st} and 5^{th} binned distribution for all three axes and the 4^{th} binned distribution for y-axis, are marked as tentative or rejected for the 4-s-window data as well. Apparently, regardless of the window length, the importance of features does not change significantly.

In the second step of feature selection, the SFS algorithm is performed on the features that are marked as important (confirmed) by the Boruta package. Assuming there are n candidate features, at the i^{-th} step of the SFS algorithm, the current subset contains i-1 features. Each of the rest n-i+1 features is temporarily added to the current subset and a five-fold cross validation is performed with the i features in the subset. The feature that yields the highest accuracy is permanently included in the current subset for the $(i+1)^{-th}$ step of the SFS algorithm. The algorithm starts with an empty current subset and ends

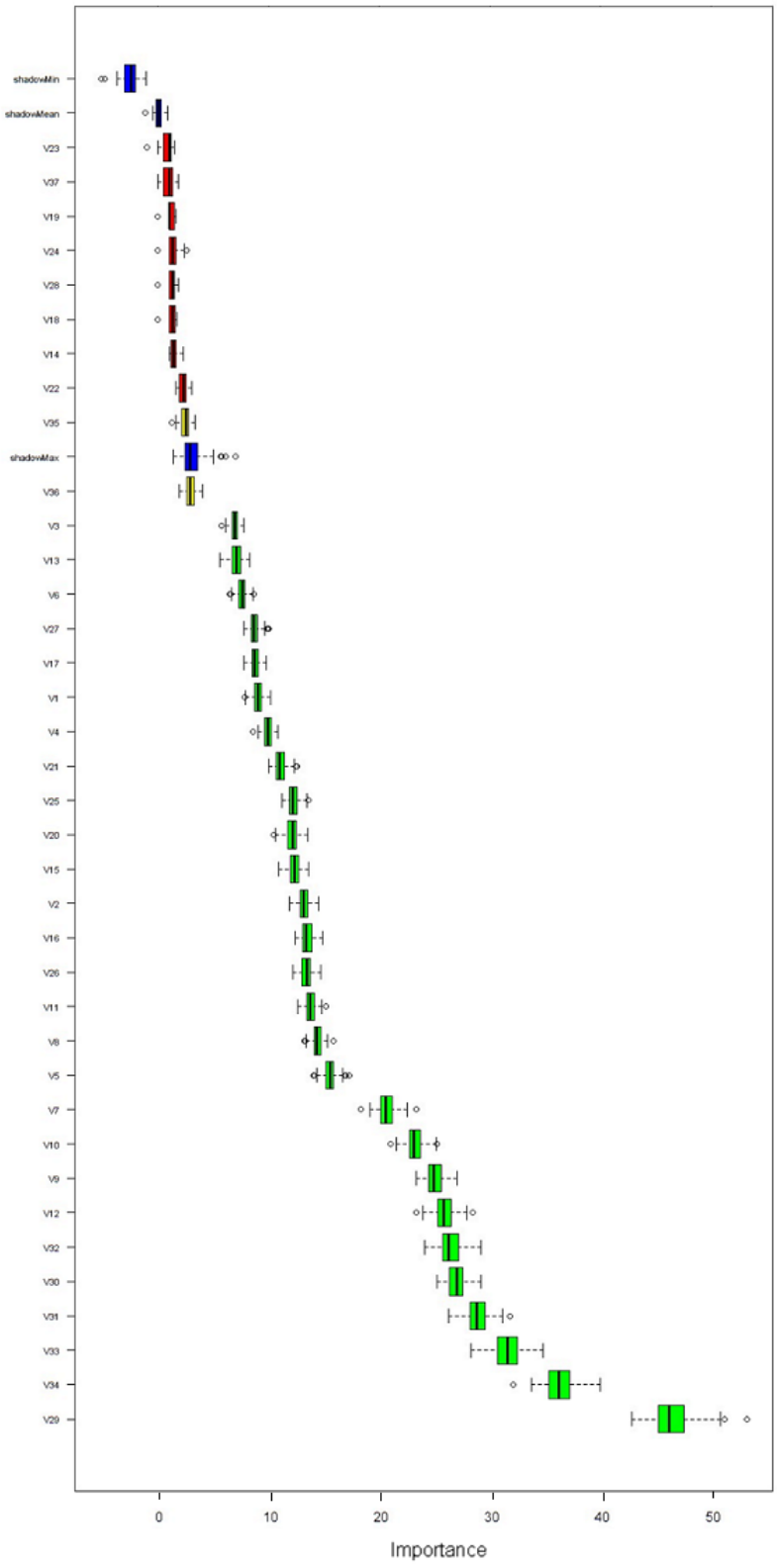

Fig. 4-6: Importance scores for features extracted with 4-s window length.

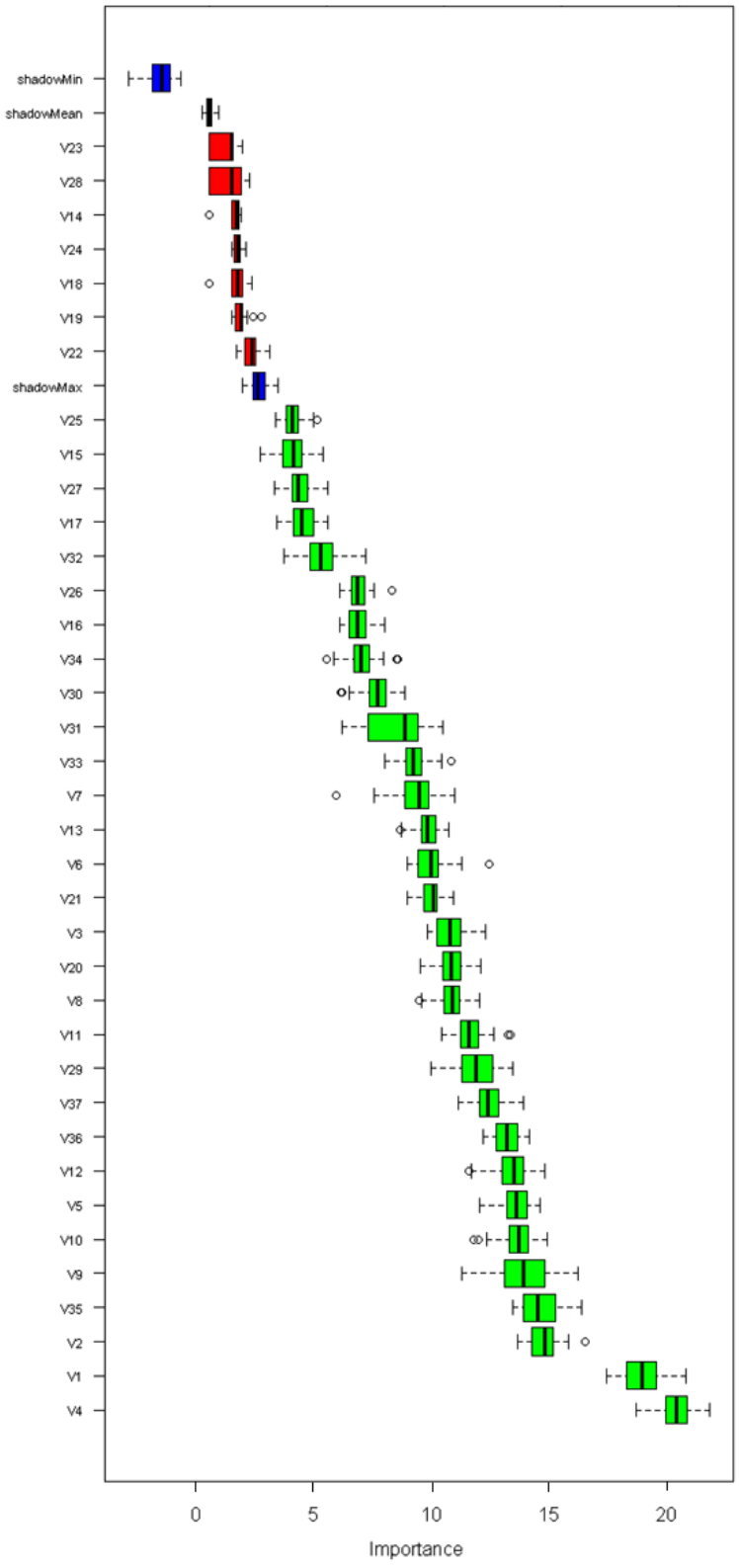

Fig. 4-7: Importance scores for features extracted with 2-s window length.

with all *n* features in the subset.

Since the Boruta package generated slightly different results for the two window lengths of 4 seconds and 2 seconds, two sets of tests are performed in this step as well. At each step of the SFS algorithm, in addition to adding the feature that produces the highest accuracy to the current subset, the highest accuracy obtained is recorded as well. Fig. 4-8 and 4-9 illustrate the change of the classifier's best performance as features are added to the current subset. Tables 4-1 and 4-2 tabulate the experimental result to show more details of the performance change.

From the performance change during the sequential feature selection, we can observe the following:

a) For both sets of configurations, the classification accuracy first increases rapidly and then stabilizes. As the number of features in the subset increases to a larger value, the classification accuracy decreases.

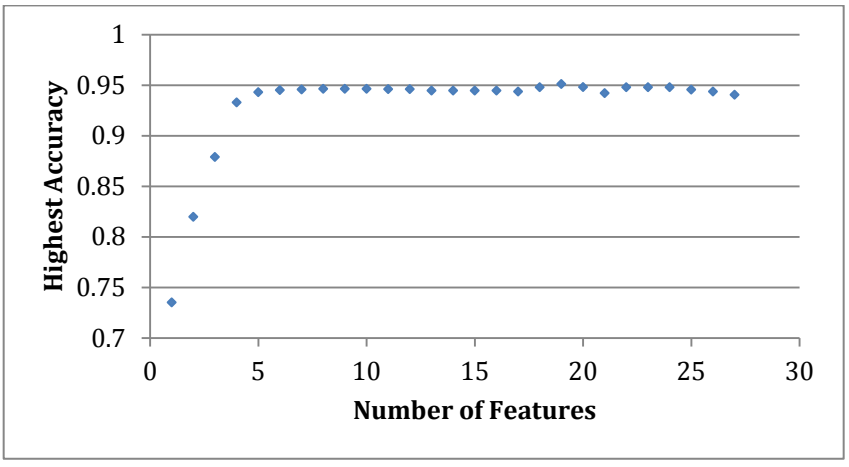

Fig. 4-8: Accuracy change with the number of features for 4-s-window and 9-interval data.

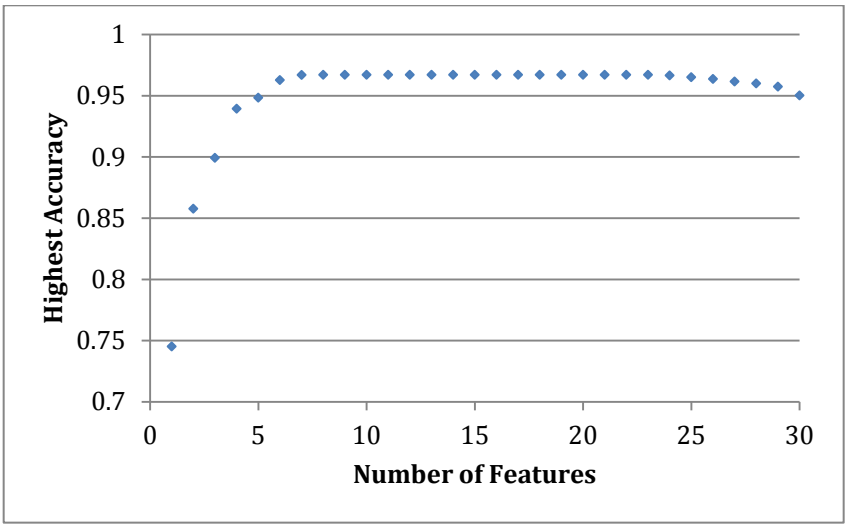

Fig. 4-9: Accuracy change with the number of features for 2-s-window and 8-interval data.

b) For both sets of configurations, the performance with the entire feature set can be achieved with as few as 5 to 6 features. This indicates that the feature selection algorithm is effective and able to eliminate redundant features. As can be seen in Table 4-1, when using 4-second windows and 9 intervals, the classifier learned from all 27 features achieves a classification accuracy of 94.07%, whereas the highest accuracy of classifiers learned from subsets containing 5 features reaches 94.31%. As seen in Table 4-2, for the configuration of 2-s windows and 8 intervals, the classifier learned from all 30 features achieves a classification accuracy of 95.03%, whereas the highest accuracy of classifiers learned from subsets containing 6 features reaches 96.28%.

As the number of features in the subset increases, the complexity of classifier increases as well. In order to determine the optimal subset and find a balance between classification accuracy and classifier complexity, we introduce a complexity penalty

Table 4-1: Accuracy change with the number of features for 4-s-window and 9-interval data

Number of Features	Highest Accuracy	Number of Features	Highest Accuracy	Number of Features	Highest Accuracy
1	0.7352	10	0.9465	19	0.9513
2	0.8198	11	0.9461	20	0.9482
3	0.879	12	0.9461	21	0.9422
4	0.933	13	0.9447	22	0.9481
5	0.9431	14	0.9447	23	0.9481
6	0.9452	15	0.9447	24	0.9481
7	0.9458	16	0.9447	25	0.9457
8	0.9465	17	0.9438	26	0.9438
9	0.9465	18	0.9481	27	0.9407

Table 4-2: Accuracy change with the number of features for 2-s-window and 8-interval data

Number Of Features	Highest Accuracy	Number Of Features	Highest Accuracy	Number Of Features	Highest Accuracy
1	0.7452	11	0.9672	21	0.9672
2	0.8578	12	0.9672	22	0.9672
3	0.8994	13	0.9672	23	0.9672
4	0.9395	14	0.9672	24	0.9667
5	0.9485	15	0.9672	25	0.9652
6	0.9628	16	0.9672	26	0.9638
7	0.9670	17	0.9672	27	0.9617
8	0.9672	18	0.9672	28	0.9601
9	0.9672	19	0.9672	29	0.9575
10	0.9672	20	0.9672	30	0.9503

when selecting the optimal subset [23]. When comparing the performances of two subsets, a penalty of 0.1% is applied for each one more feature. The final subsets chosen for the two configurations are shown in Table 4-3. The features in the optimal subsets are listed in the orders that they are selected by the SFS algorithm.

Obviously, the features in the final subsets are not the ones that are given higher

Table 4-3: Final subsets chosen after two-step feature selection.

	Window length	Number of intervals	Number of features in optimal subset	Features in optimal subset
Configuration 1	4 seconds	9	6	mean cross of *y*-axis mean cross of *x*-axis RMS of *x*-axis standard deviation of *x*-axis correlation between *x*- and *z*-axes standard deviation of *z*-axis
Configuration 2	2 seconds	8	7	mean cross of *x*-axis mean of *y*-axis RMS of *x*-axis standard deviation of *y*-axis correlation between *x*- and *y*-axes mean of *x*-axis 4th binned distribution of *z*-axis

importance scores in the first step. For Configuration 1, the final features rank 5th, 1st, 9th, 21st, 2nd and 24th places in the result of Boruta. For Configuration 2, the final features rank 11th, 13th, 6th, 7th, 26th, 19th, and 28th places in the result of Boruta. This may be attributable to the following reasons.

a) The Boruta package is a filter-based feature selection method that considers each feature individually. It scores a feature's importance based on its relevance to the class label. Therefore, if the values of a feature discriminate the class more clearly, the feature is given a higher rank. For both configurations, the performance of using all the confirmed (green) features is the same as the performance of using all 37 features, which means the tentative or rejected features do not contribute to the classification accuracy, or in other words are irrelevant.

b) The sequential feature selection algorithm tries to find an optimal feature subset that best discriminate the class. In other words, it considers the relevance of each subset as a whole with the class. In some cases the combination of two features

with lower importance scores may be more relevant to the class labels than another feature with much higher importance score, and in many cases including one more feature in the subset decreases the performance instead of increases it.

4.2.3 Experiment 4

In order to solve the overfitting problem that is common to TDIDT algorithms, we adopt a reduced error post-pruning strategy. Similar to the 5-fold cross validations performed in previous experiments, in each test, the accelerometer data collected from 4 of the subjects are used to train a decision tree and the data of the other subject (testing set) is used to evaluate the classification performance. The overall performance is then calculated from the result of the 5 tests.

For reduced error pruning (REP), a separate set of data is needed to test the classification performance as branches of the decision tree are being pruned. Therefore, each of the training sets used in the previous experiments is divided into two sets, the growing set and the pruning set [24]. First, a decision tree is learned from the growing set, and we call it the grown tree. Then, in each step of REP, a parent node is replaced with its most popular class and the new tree's performance is tested on the pruning set. If the performance is reduced, the deletion of the node is reversed, and otherwise it is kept. After the pruning is completed, the pruned decision tree's performance is evaluated on the testing set. In this study, we use a random 75% of the training set as the growing set and the other 25% as the pruning set.

We evaluate two effects of pruning: the performance change and the tree size

change. As each decision tree is pruned, we record its performance change and the number of parent nodes left in the tree. Similar to previous experiments, a five-fold cross validation is performed in each pruning step. The experimental results are shown in Table 4-4.

As can be seen in Table 4-4, the classification accuracies of the pruned trees decrease by 0.76% and 1.88%, respectively. Meanwhile, 80% of internal nodes are pruned, which means the pruning significantly reduced the decision trees' complexity. The performance reduction may be attributable to that the pruning process makes the decision tree more specific to the pruning set, i.e. perform better on the pruning set, and compromises its performance on the test data. Although the tree pruning decreases the classification accuracies, the tree size reduction compensates the performance drop since one goal of this study is to design a real-time classifier that has little computational cost and consumes little memory space.

4.2.4 Experiment 5

As described in Section 3.4.6, we adopt a misclassification correction mechanism

Table 4-4: Experimental results of tree pruning

	Classification accuracy		Average number of internal nodes	
	Before pruning	After pruning	Before pruning	After pruning
Configuration 1 (4-s windows and 9 intervals)	94.52%	93.76%	75.2	14.4
Configuration 2 (2-s windows and 8 intervals)	96.70%	94.82%	101.6	20.6

to reduce the misclassification in the case where interruptive activities occur. In previous experiments, this mechanism is adopted for all tests. In this experiment, we test the performance without using this misclassification correction mechanism and compare it with that when the mechanism is applied.

Table 4-5 presents the classification confusion matrices of the decision trees trained with the two configurations and with or without the misclassification correction mechanism applied. In the table, the letters 'S', 'W', and 'R' are short for stationary, walking, and running states, respectively. Each number indicates the number of records that are actually the state on its left and are correctly or incorrectly classified as the state above it. For example, the highlighted number 349 means 349 stationary records are misclassified as walking. Each confusion matrix in the table is a sum of the classification results of the five tests performed in cross validation.

Table 4-5: Experimental results of misclassification correction

Configuration 1 without misclassification correction				Configuration 2 without misclassification correction			
	S	W	R		S	W	R
S	4626	349	14	S	9294	805	1
W	249	2322	19	W	270	4968	9
R	42	9	1708	R	7	3	3603
Accuracy = 92.70%				Accuracy = 94.22%			
Configuration 1 with misclassification correction				Configuration 2 with misclassification correction			
	S	W	R		S	W	R
S	4687	300	2	S	9359	741	0
W	234	2351	5	W	235	5010	2
R	37	5	1717	R	4	0	3609
Accuracy = 93.76%				Accuracy = 94.82%			

As can be seen, applying the misclassification correction algorithm reduces the number of misclassifications of all types, such as stationary records misclassified as walking or running, running records misclassified as stationary or walking, etc. The accuracy changes (92.70% to 93.76% for configuration 1 and 94.22% to 94.82% for configuration 2) clearly show that the mechanism is effective. However, since the mechanism only corrects single misclassifications, the performance improvement is not significant.

4.2.5 Final configuration

Based on the results of Experiments 1 and 2, two sets of window length and interval number were used in the subsequent experiments. For the final classifier, we need to determine one set of configuration. Table 4-6 summarizes the experimental results of the third and fourth experiments for comparison. Based on the results, we decide to use configuration 2 (2-second windows and 8 intervals) for the following reasons.

Table 4-6: Summary of experimental results of experiments 3 and 4

	Feature selection effects (Experiment 3)			Tree pruning effects (Experiment 4)				
	Number of features	Classifier performance		Number of features	Accuracy		Average number of internal nodes	
		Before	After		Before	After	Before	After
Configuration 1 (4-s windows and 9 intervals)	37 to 6	94.07%	94.52%	6	94.52%	93.76%	75.2	14.4
Configuration 2 (2-second windows and 8 intervals)	37 to 7	95.03%	96.07%	7	96.70%	94.82%	101.6	20.6

- For both configurations, the classification accuracies of the pruned trees are only slightly lower than those before feature selection, with performance drops of 0.31% and 0.21%, respectively. Although not significant, the performance drop of configuration 2 is smaller.
- The performance of configuration 2 is higher than that of configuration 1 before feature selection, after feature selection, and after tree pruning.
- Using configuration 2 means that during real-time monitoring, feedbacks are provided more frequently (every second).

CHAPTER 5

CONCLUSIONS

In this study, we propose an accelerometer-based real-time human activity recognition approach using the decision tree as the classifier. The major contributions are as follows:

- Requiring only one accelerometer instead of multiple sensors worn on the subject's wrist (left or right) to increase portability, reduce cost, and broaden the applications. The system can be easily implemented to a wristband or smart watch that has an embedded accelerometer for users.
- Recognizing activities in real-time without requiring user-specific classifier training. The classifier is learned from accelerometer data collected by participants of the study instead of the new user. In other words, a new user can directly pass accelerometer data to the classifier and get activity feedback.
- Adopting the decision tree as the classification algorithm and designing a simple structure and using a decision tree simplification technique to store the trained decision tree in fairly small memory. The complexity of the decision tree is reduced by applying feature selection and tree pruning, allowing the system to have low computational cost and consume small memory space.

The proposed approach follows a process of feature extraction, feature selection, decision tree training, and decision tree pruning. Through experiments, the effects of feature extraction window length, feature discretization intervals, feature selection, and

decision tree pruning are tested. On top of this process, we also implement misclassification correction and decision tree simplification to improve classification performance and reduce classifier implementation size. The experimental results show the following:

- With the combination of 2-second window length and 8 intervals, the extracted feature data produces the best decision tree performance.
- Through feature extraction, the number of features is reduced from 37 to 7. On a subset of 7 features, the trained decision tree performs better than the one trained with 37 features, with a classification accuracy increase of 1.04%.
- Decision tree pruning slightly decreases the classification performance, while it significantly reduces the tree size. The number of internal decision tree nodes is decreased from 101.6 to 20.6, which equals a remarkable reduction in tree size.
- The proposed misclassification mechanism effectively eliminates single misclassifications caused by interruptive activities.
- With the proposed decision tree simplification approach, the trained decision tree can be saved to three arrays. The implemented decision tree can be simply initiated by reading configurations from the three arrays.

REFERENCES

[1] Jalal, N. Sarif, J. T. Kim, and T.-S. Kim, “Human Activity Recognition via Recognized Body Parts of Human Depth Silhouettes for Residents Monitoring Services at Smart Home,” *Indoor Built Environ.* , vol. 22, no. 1, pp. 271–279, 2013.

[2] S. R. Ke, H. Thuc, Y. J. Lee, J. N. Hwang, J. H. Yoo, and K. H. Choi, “A Review on Video-Based Human Activity Recognition,” *Computers*, vol. 2, pp. 88–131, 2013.

[3] Vandrico. (2016). *Wearable devices that have an accelerometer*. [Online]. Available: http://vandrico.com/wearables/device-categories/components/accelerometer

[4] S. Althloothi, M. H. Mahoor, X. Zhang, and R. M. Voyles, “Human activity recognition using multi-features and multiple kernel learning,” *Pattern Recognition*, vol. 47, no. 5, pp. 1800–1812, 2014.

[5] L. Chen, H. Wei, and J. Ferryman, “A survey of human motion analysis using depth imagery,” *Pattern Recognition Letter*, vol. 34, no. 15, pp. 1995–2006, 2013.

[6] O. D. Lara and A. L. Miguel, "A survey on human activity recognition using wearable sensors," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1192–1209, 2013.

[7] J. Kwapisz, G. Weiss, and S. Moore, "Activity recognition using cell phone accelerometers," *ACM SigKDD Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011.

[8] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz,, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International Workshop on Ambient Assisted Living*, Springer Berlin Heidelberg, 2012, pp. 216–223.

[9] S. Dernbach, B. Das, N. C. Krishnan, B. L. Thomas, and D. J. Cook, "Simple and complex activity recognition through smart phones," in *Intelligent Environments (IE), 2012 8th International Conference on*, IEEE, 2012, pp. 214–221.

[10] A. Bayat, M. Pomplun, and D. A. Tran, "A study on human activity recognition using accelerometer data from smartphones," *Procedia Computer Science,* vol. 34, pp. 450–457, 2014.

[11] S. Zhang, A. V. Rowlands, P. Murray, and T. L. Hurst, “Physical activity classification using the GENEA wrist-worn accelerometer,” *Medicine and Science in Sports and Exercise*, vol. 44, no. 4, pp. 742–748, 2012.

[12] A. Mannini, S. S. Intille, M. Rosenberger, A. M. Sabatini, and W. Haskell, "Activity recognition using a single accelerometer placed at the wrist or ankle," *Medicine and Science in Sports and Exercise*, vol. 45, no. 11, pp. 2193–2203, 2013.

[13] L. Gao, A. K. Bourke, and J. Nelson, "Evaluation of accelerometer based multi-sensor versus single-sensor activity recognition systems," *Medical Engineering & Physics*, vol. 36 no. 6, pp. 779–785, 2014.

[14] T. Brezmes, J. L. Gorricho, and J. Cotrina, "Activity recognition from accelerometer data on a mobile phone," in *International Work-Conference on Artificial Neural Networks*, Springer Berlin Heidelberg, 2009, pp. 796-799.

[15] N. Ravi, N. Dandekar, P. Mysore, P., & Littman, M. L., "Activity recognition from accelerometer data." *AAAI*, vol. 5, pp. 1541–1546, 2005.

[16] F. Kaya. (2008). *Discretizing continuous features for naïve Bayes and C4.5 classifiers* [online], Available: http://cgis.cs.umd.edu/grad/scholarlypapers/papers/fatih-kaya.pdf

[17] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features," *Machine Learning: Proceedings of the Twelfth International Conference*, vol. 12, pp. 194–202, 1995.

[18] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no.1 pp. 16–28, 2014.

[19] D. Dutta. (2016, March 22). *How to perform feature selection (i.e. pick important variables) using Boruta Package in R?* [Online]. Available: https://www.analyticsvidhya.com/blog/2016/03/select-important-variables-boruta-package/

[20] B. K. Kursa and R. R. Witold, "Feature selection with the Boruta package," *Journal of Statistical Software*, vol. 36, no. 11, pp. 1–13, 2010.

[21] J. R. Qinlan, "Introduction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[22] J. R. Qinlan, "Introduction of decision trees," *International Journal of Man-Machine Studies*, vol. 27, no. 3, pp. 221–234, 1987.

[23] R. Kohavi and H. J. George, "Wrappers for feature subset selection," *Artificial Intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[24] F. Esposito, D. Malerba, G. Semeraro, and J. Kay, "A comparative analysis of methods for pruning decision trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 476–491, 1997.