RECOGNIZING SEATBELT-FASTENING BEHAVIOR WITH WEARABLE TECHNOLOGY AND MACHINE LEARNING

A Thesis

by

JAKE MITCHELL LELAND

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Tracy Hammond
Committee Members,	Theodora Chaspari
	Daniel Goldberg
Head of Department,	Dilma Da Silva

May 2019

Major Subject: Computer Science

ABSTRACT

In the case of many fatal automobile accidents, the victims were found to have not been wearing a seatbelt. This occurs in spite of the numerous safety sensors and warning indicators embedded within modern vehicles. Indeed, there is yet room for improvement in terms of seatbelt adoption. This work aims to lay the foundation for a novel method of encouraging seatbelt use: the utilization of wearable technology.

Wearable technology has enabled considerable advances in health and wellness. Specifically, fitness trackers have achieved widespread popularity for their ability to quantify and analyze patterns of physical activity. Thanks to wearable technology's ease of use and convenient integration with mobile phones, users are quick to adopt. Of course, the practicality of wearable technology depends on *activity recognition*—the models and algorithms which are used to identify a pattern of sensor data as a particular physical activity (e.g. running, sitting, sleeping). Activity recognition is the basis of this research.

In order to utilize wearable trackers toward the cause of seatbelt usage, there must exist a system for identifying whether a user has buckled their seatbelt. This was our primary goal. To develop such a system, we collected motion data from 20 different users. From this data, we identified trends which inspired the development of novel features. With these features, machine learning was used to train models to identify the motion of fastening a seatbelt in real time. This model serves as the basis for future work in systems which may provide more intelligent feedback as well as methods for interventions in dangerous user behavior.

DEDICATION

To my fiancée, Mallory.

ACKNOWLEDGMENTS

I must begin by expressing gratitude to my advisor, Dr. Tracy Hammond, who never stops empowering her students. Dr. Hammond prioritized my success and advocated for me along the way. Her continual passion, optimism, and encouragement were paramount in creating this thesis.

I would like to thank Dr. Daniel Goldberg for his flexibility and his assistance in collecting data, as well as Dr. Theodora Chaspari for her positivity and her support throughout this process.

I owe this work to the unending encouragement and assistance from the Sketch Recognition Lab. Specifically, I must thank Josh Cherian for his invaluable feedback and guidance throughout this journey. His mentorship was crucial to my success as a researcher.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Dr. Tracy Hammond (advisor) and Dr. Theodora Chaspari of the Department of Computer Science & Engineering and Dr. Daniel Goldberg of the Department of Geography.

The Phase I study in chapter 4 was conducted in conjunction with Ellen Stanfill.

All other work conducted for the thesis was completed by the student independently.

Funding Sources

Graduate study was supported by a fellowship from the College of Engineering at Texas A&M University. Equipment and materials were funded in part by the TEES AggiE_Challenge program.

NOMENCLATURE

CDC	Centers for Disease Control and Prevention
CFS	Correlation-based Feature Selection
CSV	Comma-Separated Values
FN	False Negative
FP	False Positive
GPS	Global Positioning System
IBk	Instance-Based k-NN
k-NN	k-Nearest Neighbors
SMO	Sequential Minimal Optimization
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
WEKA	Waikato Environment for Knowledge Analysis

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

Automobile accidents remain one of the leading causes of death in the United States, especially for Americans under 60 [1, 2]. Nearly half of Americans who died in car crashes in 2014 were not wearing a seatbelt at the time of the crash [3]. For Americans under 45, more than half were not wearing a seatbelt. This is in spite of widespread detection and warning systems designed specifically to enforce seatbelt use in cars. According to statistics published by the Centers for Disease Control and Prevention (CDC), young adults are the least likely age group to wear a seatbelt [4, 5], and men are less likely to wear their seatbelt than women [5]. Clearly, existing seatbelt warning systems leave room for improvement in their encouragement of safe user behavior.

Commonly, automotive safety systems vie for the driver's attention using an audible tone or visual indicator on the dashboard [6]. These warnings are triggered by sensors integrated within the vehicle [7], often the buckle [8] itself. It is fairly standard for these systems to be centered around the driver. Even if the vehicle includes sensors for the passenger, the warning indicators are often visible only to the driver. Rarely do vehicles have seatbelt sensors in any of the back seats. Furthermore, these systems can be circumvented fairly easily. For example, drivers or passengers may leave their seatbelts always buckled across their seats, sitting on top of the belt while riding in the car. It is important to note that this is a vehicle-centric paradigm. That is, the vehicle's safety systems attempt to intervene when the vehicle's sensors indicate danger. These interventions are broad and static, operating with a standard procedure regardless of the identity of the passenger. There is no precedent for altering interventions based on the behavior of individual passengers.

This work explores a human-centered paradigm for seatbelt monitoring. That is, a

seatbelt safety system built around specific users, independent of individual vehicles. With data specific to individual users, intervention tactics can be personalized. For instance, knowledge of a user's tendencies could inform a plan for improvement, and the system could intervene in ways that the user has historically been more receptive to. Notably, such a system would always remain in effect, regardless of what vehicle that user may be riding in or what seat they may be sitting in. Plus, a user's record would not be contaminated by other people driving their vehicle.

Of course, in order for this type of system to exist, there must be a human-centric way of detecting whether a user has buckled their seatbelt. Such is the motivation for research. This work seeks to use machine learning to recognize physical activities—specifically, the action of buckling a seatbelt. In recent years, activity recognition has achieved considerable success in promoting healthy user behavior. For example, many health-related applications which monitor self-care activities (such as eating habits) [9–11] and intervene to correct undesirable behaviors [12–14] depend on activity recognition. These studies, along with many others, are discussed further in chapter 2. Human-centric activity recognition is achieved most often with wearable sensors, most commonly using including accelerometers [13]. After processing this sensor data according to standard techniques [15], machine learning classifiers can be trained on the data to recognize the activities being performed. As such, activity recognition is central to achieving context-aware computing, in which the behavior of systems changes based on what the user is doing at the time of use [13].

A personalized system could take many forms, as there are many different methods for motivation. At a basic level, the watch or phone could alert the user if they neglected to buckle their seatbelt. However, as the system is tied to the individual user, it could maintain a record of seatbelt history, rather than just notifying in the moment. With historical data on the user's behavior, this system could offer motivation by way of rewards, encouraging long-term improvement. For example, many insurance companies today pro-

vide monitoring systems that customers may install in their vehicles [16–18]. Based on the customer's driving behavior, risk is determined and insurance premiums are adjusted accordingly [19]. Since the companies reward better drivers with lower rates, this encourages long-term behavioral improvement [20]. A seatbelt monitoring system could fit nicely into the calculation of insurance rates, rewarding users who always buckle their seatbelts. This could be especially beneficial for teenage drivers: parents would be able to monitor the behavior of their children and encourage safe decision making when necessary. Health wearables also encourage desirable behavior (e.g. exercising) by offering a sense of personal accomplishment when goals are met. Many corporate offices have also taken advantage of this phenomenon, offering rewards for healthy behavior in hopes of bolstering employee well-being [21]. Similarly, tracking seatbelt behavior to meet goals and feel accomplished provides positive reinforcement for good habits, and encourages long-term well being.

This work aims to provide the foundation for personalized, context-aware seatbelt monitoring systems: the activity recognition algorithm. First, the feasibility of recognition using machine learning is investigated. Using features commonly found in literature, the motion of buckling a seatbelt should be distinguishable from other similar motions. Recognition accuracy can then be improved with the incorporation of more novel features based specifically on seatbelt patterns. Finally, the model is trained and tested on more naturalistic data, verifying that the approach is effective in real-world scenarios.

In chapter 2, we conduct a review of activity recognition literature. In chapter 3, we describe the hardware and software systems which we use to collect, process, and evaluate data. In chapter 4, we conduct our first user study and analyze the results. In chapter 5, we conduct our second user study and analyze the results. In chapter 6, we recommend future research which builds on this work. In chapter 7, we draw our final conclusions.

2. PRIOR WORK

2.1 Introduction to Activity Recognition

Activity recognition is an exciting area of research that is experiencing widespread popularity across multiple disciplines. Motivated by both the push of technological advancement and the pull of potential application areas, conferences and journals now frequently enjoy the coverage of novel, state-of-the-art research developments in not only electronic sensor technology, but now also in intelligent processing of that data using various learning techniques [13].

These developments in activity recognition and sensor technology are stimulated largely by the loftier endeavor of ubiquitous computing. In the words of Óscar D. Lara and Miguel A. Labrador, "Providing accurate and opportune information on people's activities and behaviors is one of the most important tasks in pervasive computing" [10]. It is not difficult to imagine the potential benefits of such systems. To name a few, Chen et al. present "pervasive and mobile computing, surveillance-based security, context-aware computing, and ambient assistive living" [13], and Lara and Labrador reiterate "medical, military, and security applications" [10]. There is a particularly large focus on health-related activity recognition, motivated by the growing concerns surrounding sedentary lifestyles. For instance, sensor technology allows users to keep track of their levels of physical activity, thereby motivating said users to reach certain goals. This is an example of an instance where an activity recognition system can provide feedback with the intent of motivating a change in the users' behavioral patterns.

Recently, the research focus has shifted from the development of the sensors themselves to the intelligent analysis of sensor data to extract the information necessary to address real-world contexts. According to Chen et al., this process can be broken down

into four primary steps [13]:

1. Setup: select appropriate sensors and deploy them where necessary (to both objects and the environment) to adequately track a user's behavior as well as changes in the user's environment

2. Collect: capture data from the sensors, store it appropriately, and process the data to extract useful information (through data analytics and suitably abstracted knowledge representation formalisms)

3. Model: based on the collected data, generate computational activity models such that agents can conduct reasoning and manipulation

4. Develop: using the computational models, construct a final reasoning algorithm which recognizes activities amidst sensor data

As one might expect, the potential for novelty increases as the list progresses.

Already, state-of-the-art systems are able to recognize such activities as shown in Table 2.1. Presumably, this is only the beginning. As pervasive computing continues to trend toward the mainstream, activity recognition will move from being awkward and unwieldy to being convenient and unnoticed. With the integration into smart phones and smart watches, this progression is already well on its way.

If it was not already clear, activity recognition is an expansive topic. Here, we will focus specifically on two different approaches to sensor-based activity recognition, dubbed photic tracking and kinetic tracking, then on the more technical implementations of such systems, followed by popular areas of application.

2.2 Approaches to Sensor-Based Activity Recognition

The designation of "sensor-based" is a broad, encompassing qualifier. That is, sensor-based activity recognition can be achieved with a wide swath of technological equipment.

Regarding their four steps of activity recognition (enumerated in section 2.1), Chen et al. acknowledge the "raft of methods, technologies, and tools available for use" [13]. With this in mind, the authors break activity recognition into two classifications: vision-based activity recognition and sensor-based activity recognition. By vision-based, the authors refer to "visual sensing facilities", e.g., optical cameras using computer vision software. By sensor-based, the authors refer to "emerging sensor network technologies", e.g., (often wearable) sensors tracking state change with respect to time. For the purpose of this paper, we found these terms to be misleading. For instance, the distinction between vision-based and sensor-based implies that vision-based recognition does not employ sensor technology, which is incorrect. Rather, vision-based recognition makes use of vision-based sensors. To reduce confusion, we establish different terminology. The data collection hardware which Chen et al. refer to in vision-based activity recognition will be dubbed photic tracking technology, in reference to the sensors' ability to collect light (visual, infrared, etc.) data. The data collection hardware which Chen et al. refer to in sensor-based activity recognition will be dubbed kinetic tracking technology, in reference to the sensors' ability to collect position/motion (acceleration, GPS location, etc.).

2.2.1 Photic Tracking

Photic[1] tracking refers to data collection from sensors which interpret signals from light wavelengths. Most commonly, this could be thought of as an optical camera; however, such sensors are not limited to the visual spectrum alone. Non-visual wavelengths, such as those obtained by infrared sensors, are also of great utility.

Consider "A Surfaceless Pen-Based Interface" [23], wherein the researchers attempted to design a system which, when using a special pen, could track drawing movements on any given surface. The implementation worked using infrared light: the pen was equipped

[1]Photic: of, relating to, or involving light especially in relation to organisms [22]

with an infrared emitter, and the environment was fitted with an infrared camera. In this way, the tip of the pen was trackable. Their research found that while the system worked and was easy to get the hang of, it wasn't as well suited for more complex drawing tasks.

In the vein of consumer well-being, systems such as the "Driver tracking and posture detection using low-resolution infrared sensing" system [24] have much to offer. This research also uses infrared, this time to track a driver of a vehicle and identify movements and postures. Rather than using a discrete emitter and receiver, this system employs a more common low resolution infrared camera imaging system, like that of security cameras designed to operate at night. Neural networks were used to classify 16x16 images, identifying 18 different driver positions. This is the same process as is used in computer-vision/image-classification algorithms, only using an infrared camera for better low-light performance. Their research was preliminary, conducted on a small set of users, but appears promising. Consumer well-being is generating a significant amount of interest.

Another popular area of research is the study of eye-tracking systems. The eye-tracking trend is often thought of as the ability to control computers without use of the keyboard or mouse. However, eye-tracking technology can be much more sophisticated. For example, Purnendu et al. explore a system which uses eye-tracking data to perceive which activities users may be performing [25]. In this case, users were engaged in an educational problem-solving task, and the system was correctly able to identify whether users were "reading", "gazing at an image", or "problem solving". In another example, researchers attempted to use multiple eye-tracking techniques to predict the presence of certain breast-related health problems, experiencing varied results [26, 27].

A more commonly-known photic tracking device is the Leap Motion, a small peripheral that used infrared tracking to recognize hand movements and gestures. This opens many new interaction possibilities. For example, research has been conducted on the ability to "draw" three-dimensional shapes, which software could recognize with reasonable

accuracy [28].

Other commonly known examples of photic tracking include Microsoft's Kinect, which uses infrared technology to track body positioning and virtual/augmented reality headsets, which are often positioned using infrared sensors. It is worth pointing out that most of these examples make use of the infrared light wavelength. Infrared is often favored primarily due to its ability to function under any level of light, but also due to other factors, such as reduced intrusiveness [24].

Some of the difficulties associated with photic tracking are found in visual signal noise, plus limitations due to tracking resolution/framerate. Additionally, infrared sensors are presently less ubiquitous and may require more complex computer vision algorithms to recognize and track certain points of interest.

2.2.2 Kinetic Tracking

Kinetic[2] tracking refers to data collection from sensors which interpret signals from physical data like position, rotation velocity, acceleration, etc. Kinetic tracking is especially noteworthy due to the accessibility and availability of such sensors in everyday technology, such as a smartphone or smartwatch.

Within the context of kinetic tracking, Chen et al. draw a distinction between dense sensing-based activity monitoring, referring to sensors placed in the user's external environment, and wearable sensor-based activity monitoring, referring to sensors placed directly on the user [13].

2.2.2.1 Dense Sensing

Dense sensing gets its name from the density of environmental sensors required for complete user tracking. For example, cars contain sensors that detect whether the driver

[2]Kinetic: of or relating to the motion of material bodies and the forces and energy associated therewith [29]

has fastened their seatbelt [6–8]. Most cars also contain a passenger weight sensor that determines whether the passenger airbag should deploy in the case of an accident.

Another example is that of kinetic-sensor-enabled gaming controllers, like those of the Nintendo Wii game console. The Wii remote served as a convenient tool for motion-enabled game interaction, and it has also served as a basis for fun research projects, like that of the "Wiiolin" [30]. The Wiiolin made use of the Wii remote and the complementary sensor bar to mimic playing a violin or a cello.

While appropriate for simple, specific recognition tasks, dense sensing is less useful for more general, ubiquitous sensing tasks. A system designed to recognize a user's activity throughout the day would warrant a plethora of sensors placed throughout the user's path of interaction. Dense sensing does not scale well, hence the greater interest directed toward wearable sensing.

2.2.2.2 Wearable Sensing

With wearable sensing, in contrast to dense sensing, the sensors are placed on the users themselves, rather than on the objects the users are expected to interact with. Wearable sensing is attractive due to its potential for greater efficiency at the expense of less sensor hardware. Presumably, a relatively small set of sensors (e.g. on wrists, ankles, chest, etc.) can track a wide range of user activities. The challenge is now that the activity data is more obfuscated—the system doesn't inherently know what objects the user may be interacting with. Only data about the user's body and is known. Hence, with greater challenge but greater potential for reward, wearable sensing is a popular topic of research, and is the primary focus of this paper.

Lester et al. echo this sentiment in their research "A Practical Approach to Recognizing Physical Activities" [9]. Their foremost goal is to establish a system that is appealing, useful, practical, reliable, and easy to incorporate. With this purpose in mind, the researchers

established the following restraints:

1. Data should be required only from a single point on the user's body, and not necessarily the same point for everyone

2. System should be universally applicable, working "out of the box" for any given user, where system tuning and personalization may be used to improve a user's experience, but is not required

3. Recognition should be achievable with only a small, affordable set of sensors, remaining sensitive to hardware costs

Clearly, universal applicability is paramount. In order for the benefits of wearable technology to be fully realized, the system should work (intuitively) for any user, and the system should be affordable for any user. A wealth of wearable sensor research follows this general pattern. Frequently, papers are published which investigate varying placements of sensors toward the end of detecting certain types of activities.

In the same year, Pärkkä et al. published "Activity Classification Using Realistic Data From Wearable Sensors" [31], which explores sensor types and placements in even greater detail. This study, like many, focuses on health data, namely walking, running, biking, etc., which are often described together as ambulation (as seen in Table 2.1). With 35 channels of data spread across various measurement sites, the researchers concluded that the wrist and thigh locations were most ideal, and that "accelerometers proved to be the most information-rich and most accurate sensors for activity recognition" for their purposes [31]. Fortunately, accelerometers are some of the most affordable, most pervasive sensors amongst technology today.

Interestingly, in a study two years prior, Bao and Intille came to a similar conclusion. In their study, Bao and Intille requested that participants perform everyday activities (20

Group	Activities
Ambulation	Walking, running, sitting, standing still, lying, climbing stairs, descending stairs, riding escalator, and riding elevator
Transportation	Riding a bus, cycling, and driving
Phone usage	Text messaging, making a call
Daily activities	Eating, drinking, working at the PC, watching TV, reading, brushing teeth, stretching, scrubbing, and vacuuming
Exercise/fitness	Rowing, lifting weights, spinning, Nordic walking, and doing push ups
Military	Crawling, kneeling, situation assessment, and opening a door
Upper body	Chewing, speaking, swallowing, sighing, and moving the head

Table 2.1: Some examples of activities recognizable by state-of-the art sensor-driven activity recognition systems [10]. © 2013 IEEE.

types, in total) with 5 sensor placements: wrist, upper arm, hip, thigh, and ankle. In this study as well, the researchers favored thigh and wrist placement [32].

These ideas are not new. A German paper, "Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring" was published in 1999 [33]. The researchers concluded that "suitable placement of a small number of calibrated piezoresistive accelerometer devices" was sufficient to recognize everyday ambulation, with high reliability at that.

Noticeably, the basis of wearable sensor research focuses fairly consistently on carefully-placed sensors that detect a common set of ambulatory activities. More recently, this research is expanding into more novel territory. For example, Tapia et al. built upon basic ambulatory recognition to detect not only the type of activity being performed, but also the level of intensity (with mixed results) [34]. More novel expansions to the baseline model of activity recognition technology is explored in the next section. Of course, a critical part of activity recognition is the actual analysis of the sensor data. Sizable portions of the aforementioned research studies are dedicated to the algorithmic techniques of recognizing data. This is also further discussed in the next section.

2.3 Implementations and Applications of Activity Recognition

The early, foundational studies enumerated in section 2.2 set the stage for feasible activity recognition using wearable sensor technology. With this idea on the minds of academic researchers, the next challenge was the facilitation of more widespread adoption. For this to occur, Lara and Labrador lay out several factors that must be considered [10]:

1. Selection of attributes and sensors: the system need only observe those features which are most effective in distinguishing activities, and cost should be minimized through limiting the system to sensors which are absolutely necessary

2. Obtrusiveness: if everyday use is a goal, the system must be convenient and stay out of the user's way

3. Data collection protocol: the system should be trained on naturalistic data which more closely resembles real-world activity

4. Recognition performance: the system should accurately recognize activities (minimizing false positives and false negatives), and should work "out of the box" on new users

5. Energy consumption: if the system is expected to run on energy-constrained mobile devices, energy consumption must be a consideration

6. Processing: with the previous factors in mind, consider where the system's heavy-lifting should occur

7. Flexibility: consider the appropriateness of subject-dependent evaluations versus subject-independent evaluations for the given application

Increasingly, research is oriented toward vehicles that could facilitate more widespread adoption of activity recognition into everyday life.

2.3.1 Sensory Devices

One of the most recognizable factors in wearable sensor adoption is the "wearability" of the sensors. As such, considerable effort has gone into conducting recognition tasks via more common, everyday items, such as mobile devices or articles of clothing.

2.3.1.1 Mobile Devices

Examples of this were seen as early as 2006, when Maurer et al. investigated the accuracy of activity classification using only the sensors onboard an early smartwatch [35]. A common example of a mobile device is the smart watch: a mobile piece of technology which often already contains the necessary sensors (accelerometers) combined with ideal placement (wrist). Examples of this were seen as early as 2006, when Maurer et al. investigated the accuracy of activity classification using only the sensors onboard the eWatch shown in Figure 2.1 [35]. In this paper, Maurer et al. acknowledge the successes of even earlier studies which already verified that wrist-based recognition was viable [32]. With this in mind, Maurer et al. also placed the watch in other locations around the body, experiencing comparable results.

Another common example of activity recognition via everyday technology is the smartphone [36]. Regarding the processing factor of system design, the system's heavy lifting can be easily offloaded to the user's cell phone [10]. The researchers assert that complex algorithms previously better suited for desktop workstations are now manageable by mobile devices, thereby making an activity recognition less obtrusive and more flexible.

Brezmes et al. extend this notion to not only use the mobile phone for data processing, but also for accelerometer data. This approach allowed a user to place the phone in their preferred location, and it was found that after user-dependent training, the system performed adequately [37]. Kwapisz et al. reach the same conclusion, finding that cell phones' built-in accelerometers allow easy collection of "useful knowledge about the

Figure 2.1: The eWatch sensing platform, an early "smartwatch" [35]. © 2006 IEEE.

habits of millions of users passively—just by having them carry cell phones in their pockets" [38].

That said, research seems to support that smartwatch-based recognition is often more effective than smartphone-based recognition. Weiss et al. studied recognition accuracy of both hand-oriented and non-hand-oriented activities with both a smartphone and a smartwatch. They found that, unsurprisingly, smartwatches were significantly better at recognizing hand-oriented activities; however, they also found that smartwatches were more effective at recognizing the non-hand-oriented activities as well [39]. This suggests that smartwatches are better suited for recognition scenarios involving more diverse activities.

A practical application of the conclusions of Weiss et al. is found in the research of Thomaz et al., wherein the recognition of "eating moments" (i.e., different methods of eating different types of foods) is accomplished using inertial data from a wrist-mounted sensor [11].

2.3.1.2 Clothing

Perhaps a more experimental realm of pervasive sensor technology is that of clothing-embedded sensors. While arguably less scalable, such systems have the potential to offer data of much higher fidelity. Take Laerhoven and Cakmakci, who attempted to use sensor-laced pants to recognize ambulation activities with minimal user attention [40].

Consider also a more specific application of such technology: user interaction with objects in an office environment. In multiple publications, Paulson et al. explore the viability of recognizing and distinguishing 12 different office activities using "CyberGlove", which contained 22 sensors [41, 42]. The researchers discuss that, while the recognition performance could benefit from user-dependent training, the sensor-based activity recognition was more practical than RFID labeling (dense sensing, kinetic tracking) or large camera scopes (photic tracking).

Sensors worn as clothing also benefit from being more comfortable as well as discreet, enabling everyday usage without arousing suspicion or attracting too much attention. This can be particularly helpful in cases where sensor technology is used to assist men and women with disabilities. For instance, Brhlik et al. created and studied a system of sensors and feedback mechanisms to assist the visually impaired with navigation [43]. Such technology has great potential to enable greater freedom for those in situations of disability.

2.3.2 Data Analysis

As mentioned previously, a crucial piece of sensor-based activity recognition is the handling of the sensor data itself. Examples of raw sensor data are shown in Figure 2.2. At the most basic level, raw data is abstracted to a collection of features. Usually, a window of the data is isolated—say, the past 1 second—and some data points are extracted from the window (e.g. the average value of the sensor, the standard deviation of the sensor,

etc.). These extracted values are those features upon which the classification algorithms are built. The algorithms then learn to recognize activities based on the patterns deduced from the values of the features.

2.3.2.1 Features

Much research is based on fairly simple sensor data with rudimentary features. An early activity recognition study by Randell and Muller attempted to distinguish 6 different ambulation activities with only a two-axis (X and Y) accelerometer. The researchers considered only 2 features: "the RMS and integrated values of both sensors over the last 2 seconds" for each sensor, yielding only 4 features in total [44]. Even such basic data points were found to be sufficient for distinguishing the activities. A few years later, a similar study by Ravi et al. used a three-axis accelerometer to identify 8 different activities. In this study, four features were extracted from each axis (mean, standard deviation, energy, and correlation), yielding 12 features in total [45]. Even when selecting from a larger set of activities, Ravi et al. presented accuracies even higher than those of Randell and Muller, suggesting that the amount and quality of features has a large bearing on the effectiveness of the algorithm.

Unsurprisingly, a considerable amount of research has gone into the methods of extracting useful information from raw sensor data using features. Figo et al. discuss in great detail some of the differing approaches of feature extraction. They establish three categories of preprocessing techniques [15]:

1. Time Domain, i.e., time-based numerical operations
2. Frequency Domain, i.e., wave-based transformations
3. Discrete Domain, i.e., symbolic string functions

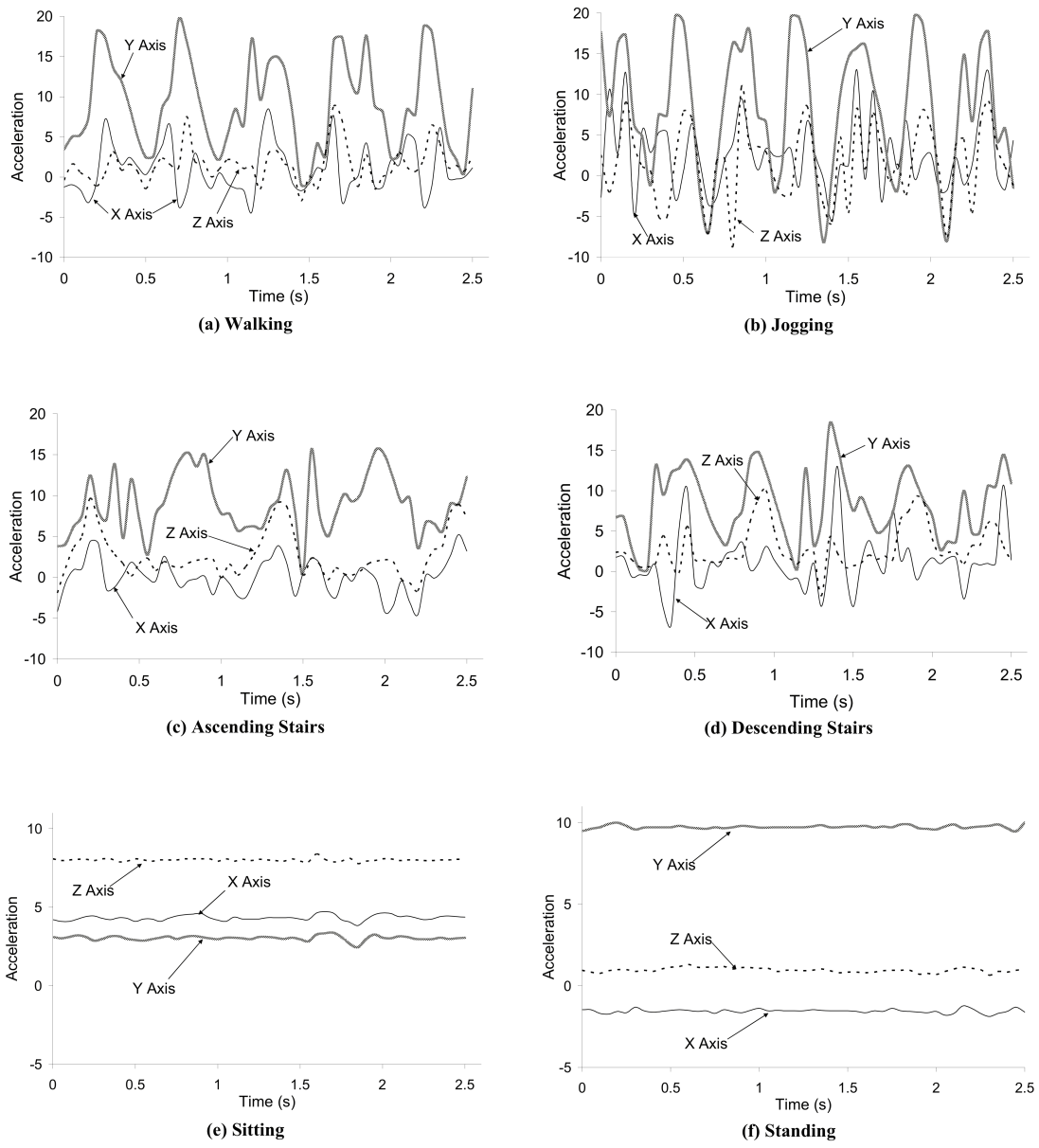


Figure 2.2: Example plots of accelerometer data for six different ambulation activities [38].

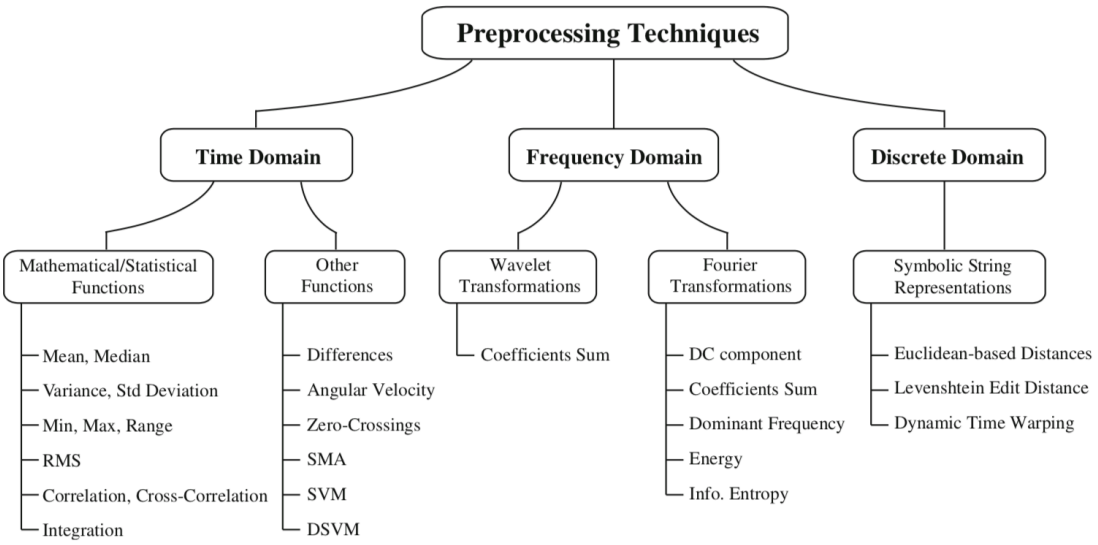


Figure 2.3: Categorization of common feature-extraction techniques applied to sensor signals for activity recognition [15]*.

The full tree of preprocessing techniques evaluated by Figo et al. can be found in Figure 2.3. The researchers then evaluate the different calculations according to computation cost, storage requirements, precision, and mobile compatibility, then tests the features in different activity contexts. In general, the frequency domain features did not perform as well as hoped, especially considering that frequency calculations are more intensive. For this reason, we focus on time domain features.

For example, Garcia-Ceja et al. built upon some of the basic data processing principles and extended them to the long-term domain [46]. That is, rather than attempting to recognize an immediate activity like ascending a flight of stairs, the researchers attempt to recognize broader activities like commuting, exercise, working, etc. In the case of this research, a user's long-term activities were segmented using "Hidden Markov Models and

*Reprinted by permission from Springer Nature: Springer-Verlag, *Personal and Ubiquitous Computing*, "Preprocessing techniques for context recognition from accelerometer data", Davide Figo, Pedro C. Diniz, Diogo R. Ferreira, João M. P. Cardoso, © 2010.

Conditional Random Fields" [46], neither of which were addressed in the Figo et al. survey [15]. Clearly, the field is still under rapid development, and there is always room for improvement.

2.3.2.2 User-Independent vs. User-Dependent

Knowing these data processing techniques, there are two main ways to go about implementing said techniques into actual systems. This has already been mentioned briefly within the flexibility factor of activity recognition system design [10]. In the most basic approach, data is collected through test instances, and the data is aggregated to create a single, universal model which is intended to apply to any user that wishes the utilize the system. This is usually referred to as user-independent or impersonal evaluation. While this approach is convenient and desirable, allowing the system to work "out-of-the-box", it has its limitations in contexts where users perform activities in unique ways.

To address such situations, researchers often train their models on a user-by-user basis. This is usually referred to as user-dependent or personal evaluation. This approach frequently enjoys higher classification accuracies and better performance for individual users; however, this adds complexity to the system, and there are drawbacks. While user-independent systems may employ pre-built classification models that were generated offline, a user-dependent system must have the capability to re-generate classification models online, effectively "learning" as the subject continues to use the system.

However, a completely user-dependent system lacks the confidence to work "out-of-the-box", as it must be trained on the individual user before it becomes useful. One approach to mitigating this disadvantage is to combine aspects of user-independent algorithms and user-dependent algorithms, i.e., the system works well enough "out-of-the-box" to be usable, but also continues to improve with prolonged used. In essence, personalized algorithms have the potential to offer much better recognition performance, but require

a more complex system design. Prior works show many examples of impersonal versus personal approaches to evaluation [24, 32, 37, 39].

2.3.3 Health and Wellness

If it was not already apparent from the plethora of examples already cited in this paper, a significant portion of activity recognition research falls into the domain of health and wellness. Without rehashing the previous references, we would like to briefly comment on this trend. Both the research community as well as industry have elected to focus on such applications. Perhaps this was kick-started by the fact that walking is one of the most rudimentary everyday activities, serving as a baseline test for new systems. Consider the widespread success of personal fitness trackers from Pebble and Fitbit, or the recent wave of smartwatches like the Apple Watch, or even simple smartphone health apps—users enjoy keeping track of their daily step counts. As our society becomes increasingly sedentary, personal fitness trackers are beginning to take hold among the health-conscious.

Information from activity recognition systems has the potential to promote behavioral modification through intelligent feedback and interventions. Take applications like Step Up Life [47], which monitors a user's level of inactivity, providing physical activity reminders when necessary, or World of Workout [12], a role-playing game in which the user's character "evolves based on the exercises the user performs in reality". Both applications were found to motivate users to engage in some sort of exercise.

Building upon the general idea of encouraging physical activity, activity recognition can be used to build systems that promote higher-level wellness. Let Me Relax [14] monitors user activity, prompting stress-relieving mental relaxation techniques when necessary. Systems may also monitor general self-maintenance activities such as the brushing of teeth, combing of hair, taking of medicine, etc. Cherian et al. propose a system which is able to recognize whether a user has brushed their teeth [48]. Such systems are useful in

scenarios involving elderly care or patients with some form of dementia, wherein the user may not have the capacity to keep track of their own activities.

Many health-related recognition systems are in their more experimental stages, but have the potential to contribute greatly to the field of medicine and to the general well-being of society.

2.4 Summary

As our world trends toward the realization of ubiquitous computing, activity recognition plays a critical role in establishing a link between human action and the appropriate computational response. The challenge becomes recognizing higher-level activity when provided with only data from rudimentary sensors. Nevertheless, the projected benefits of dependable human activity identification are so tantalizing that a wealth of novel research on the subject is beginning to emerge.

Over the past 10-20 years, activity recognition has grown and developed from simple accelerometers with a handful of features to a range of sensors with a much more diverse repertoire of feature algorithms. Activity recognition has its roots in the health industry, largely due to the social push for heathier, more active lifestyles. There are also potential applications toward general wellness, e.g., posture or mental health. Additionally, systems are proposed for the care of the elderly or those with dementia, to help keep track of mundane, but important, daily activities like brushing teeth and taking medicine.

There are many approaches to recognizing a user's activities. At a high level, regarding the sensors themselves there are two main approaches: photic tracking and kinetic tracking. Photic tracking makes use of technology which senses based on light, whether visual, infrared, or otherwise. Kinetic tracking makes use of technology which senses based on physical properties like position, velocity, or acceleration. Both approaches are appropriate for different applications, but kinetic tracking often enjoys a higher degree of

information granularity. This survey focuses on kinetic tracking. Within kinetic tracking itself, there is a distinction between sensor placement approaches: dense sensing and wearable sensing. Dense sensing entails placing sensors on items in the environment that the user is expected to interact with. Wearable sensing, instead, places the sensors on the user's body itself. Due to issues of scalability and feasibility, dense sensing is usually only appropriate for small, specific problems. For larger, more open-ended problems, wearable tracking is more universal viable. However, having more universal data unattached to any specific object means that the challenge is to accurately recognize what the user is doing, based only on their movements. This research looks specifically on wearable sensing.

Results from activity recognition studies are largely encouraging, reinforcing that the field is worth pursuing. There is great potential for activity recognition technology to play a beneficial role in daily life, especially as pervasive computing becomes more of a widely-adopted reality.

3. SYSTEM IMPLEMENTATION

3.1 Data Collection

When conducting this research, it was important that we use a system configuration that is easy and affordable. That is, we be using instruments no more advanced than what is readily available—and common—for an average user. With this in mind, we chose to use a combination of a Pebble smart watch [49] and an Android smart phone. The Android phone was selected not only for being the most common mobile platform, but also for its convenient application development tools. The Pebble watch was selected as it is a good representative of the caliber of wearable sensors on the market today. Specifically, the Pebble contains a 3-axis accelerometer (see Figure 3.1) capable of registering up to ± 4 G. The 3-axis accelerometer is all we need to effectively recognize directional motion. Furthermore, as the arm is the primary actor in a seatbelt-fastening motion, a wrist-mounded sensor is desirable. For upper body and arm activity recognition, wrist-mounted sensors are recommended by literature [11, 32, 33, 46].

To facilitate data collection and real-time testing, we developed two applications: a lightweight Pebble application and a heavyweight Android application. These applications communicate using Bluetooth. The Pebble application is very simple—its only purpose is to read the current value of the accelerometers and broadcast that data. The app broadcasts 5 packets per second, wherein each packet contains 5 samples. This results in a 25 Hz accelerometer sample rate.

When the Pebble application broadcasts a data packet, the Android application collects it and must decide what to do with it. The application has two modes: "collect" mode and "test" mode. These modes are explained in the following two paragraphs, and the high-level process diagram of this application is shown in Figure 3.2.

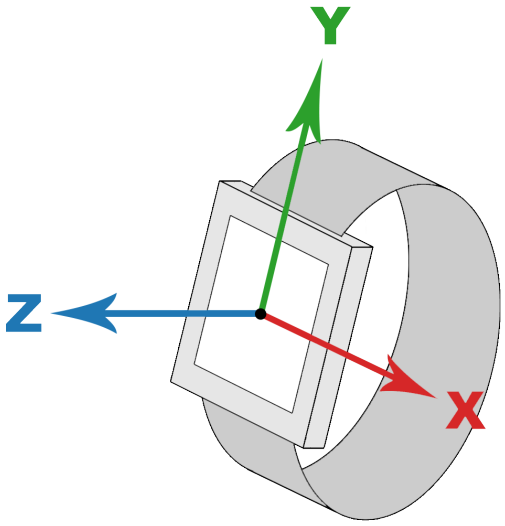


Figure 3.1: Diagram showing the conventional orientation [50, 51] of the three axes of wearable accelerometers.

The "collect" mode was developed first. The purpose of this mode is simply to collect raw accelerometer data during user studies. This motion data is stored locally on the Android device, which can then easily be offloaded to a computer to allow for researchers to manipulate and analyze the data. One of the biggest upfront costs when preparing data to train machine learning models is *labeling*. The training data must be labeled according to the ground truth. In our case, the data should be labeled according to whichever activity the user was performing at the time of collection. In the first iteration of the application used for Phase I (see chapter 4) and of our research, the application recorded only the sensor data of the Pebble watch along with a timestamp for each sample. The user id and the activity needed to be labeled by manually. To improve this process in Phase II (see chapter 5) of our work, a text field and a button were added to the application. The text field allowed the researchers to type notes describing the sample, which would be appended to every sample. The button allowed the user to mark precisely the timespan in which the

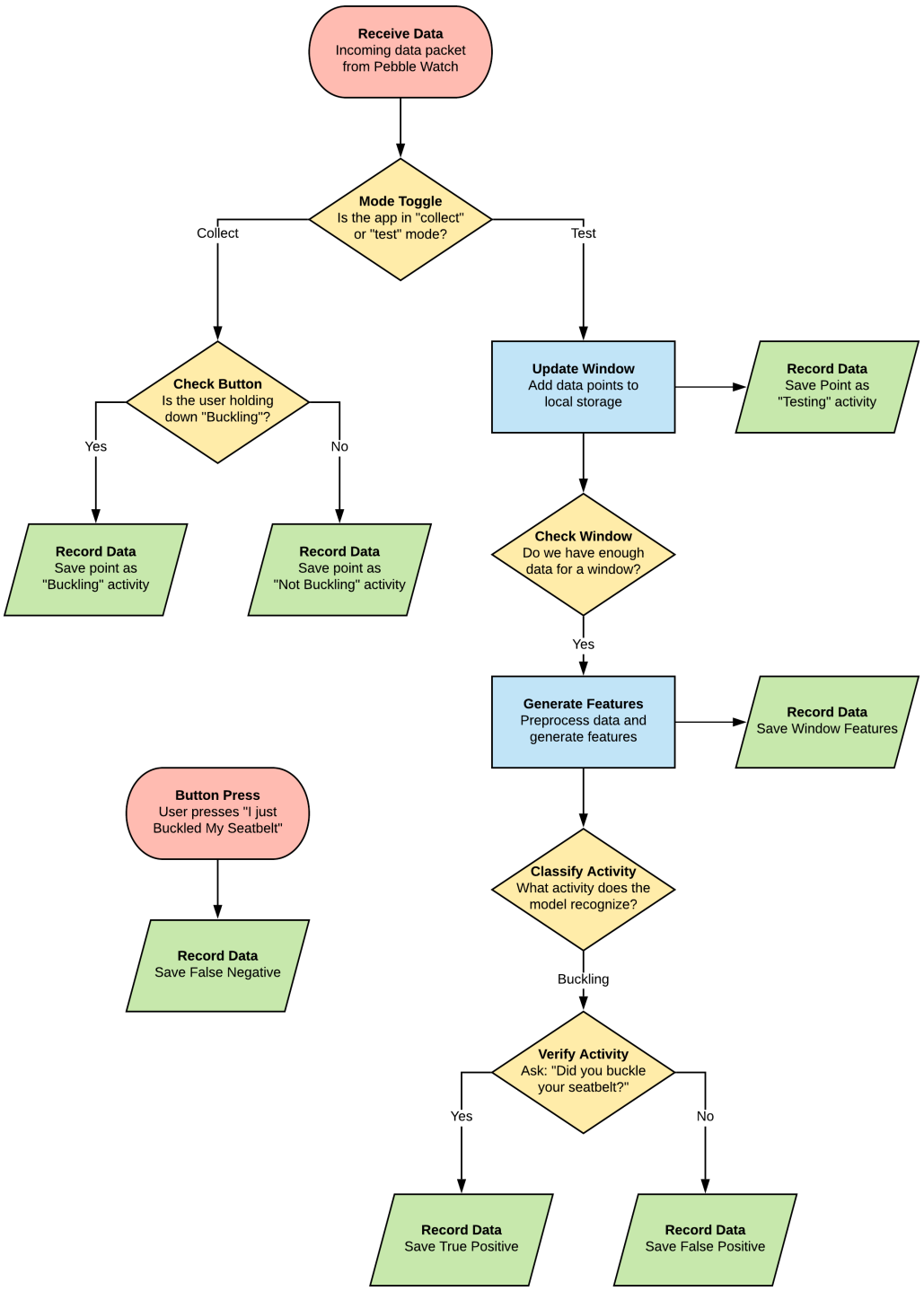


Figure 3.2: The process diagram of the Android application used for both data collection and real-time activity recognition.

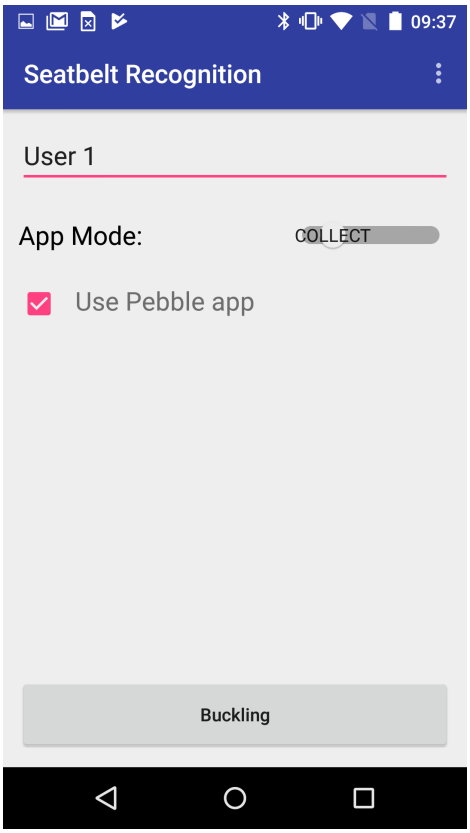


Figure 3.3: The Android application in "collect" mode, used for collecting training data. The text box at the top of the screen allows the researcher to make notes. The button at the bottom of the screen allows the researcher to indicate that the user is currently in the process of buckling their seatbelt. The application collects data so long as the "Use Pebble app" box is checked.

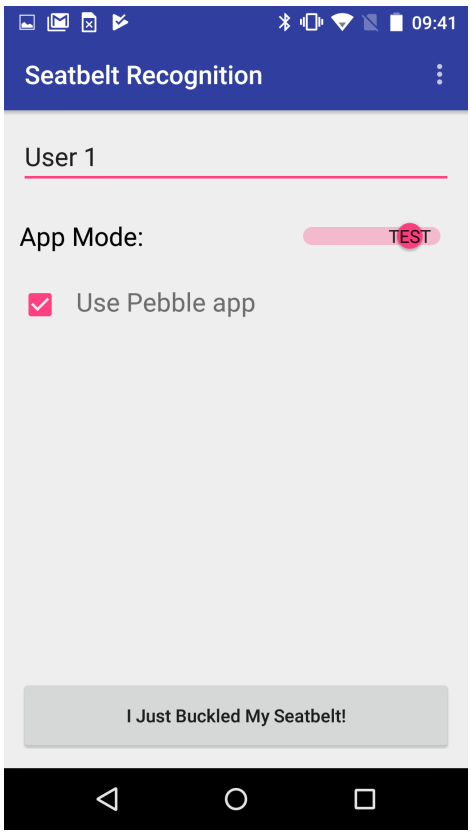
user was buckling their seatbelt. Whenever the button was held down, the corresponding samples would be automatically labeled as "Buckling". These improvements greatly increased the efficiency of data handling. A screenshot of the application in "collect" mode is shown in Figure 3.3.

The "test" mode was developed later in the research process as a way to test the real-time performance of our model, specifically during the real-time portion of Phase I (see chapter 4) and all of Phase II (see chapter 5) of our work. In "test" mode, our classifier is loaded onto the Android device itself. Now, not only does the application collect the

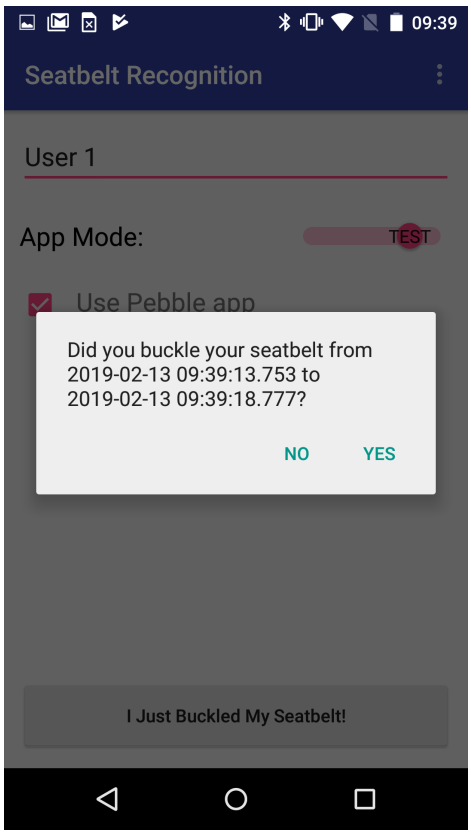
raw data points, but it can also runs our recognition algorithm: processing the data points, generating features, and classifying the activity (this algorithm is discussed in more detail in the three following methodology chapters: Phases I & II). In this case, when the Android application receives a data sample, it is automatically labeled as "testing"—a placeholder indicating that we are not actually recording the user's activity, since this is a real-time test. Here, the goal is to make sure our recognition algorithm is performing properly, making note of successful classifications and failed classifications as they arise. As such, when the application is in test mode, we record not only the raw data points, but also the processed features (more on this later) and the performance in terms of true positives, false positives, and false negatives, as explained in the next section. Just as with the "collect" mode, there is a text field allowing for notes and a button; however, the button now exists for the user to manually notify the application that they buckled their seatbelt, in case the recognition failed to recognize it. Screenshots of the application in "test" mode are shown in Figure 3.4. Although "test" mode is primarily intended for trying out the recognition algorithm in real time, it can still serve to provide data useful for training the model further. As mentioned prior, data in this mode is not explicitly labeled. However, since the time windows of true positives and false negatives are recorded, these windows can be cross-referenced with the raw data to deduce when the user was buckling.

3.2 Machine Learning

With adequate data, we must focus on the machine learning process itself. That being: if we feed our "machine" a lot of examples of both "buckling" and "not buckling" data, would it be able to identify whether a new, unknown piece of data represents buckling? Here, we introduce the notion of "features". When we have a chunk of accelerometer data, we must reduce it to some notable attributes which describe that chunk of data. For example, we could calculate the average value and standard deviation of each axis.



(a) The text box at the top of the screen allows the user to make notes. The button at the bottom of the screen allows the user to indicate a false negative (they buckled their seatbelt but the application failed to recognize the motion).



(b) A pop-up is shown when the classification algorithm detects a buckling motion. The user has the option to mark the recognition as a true positive (they did buckle) or a false positive (they did not buckle).

Figure 3.4: The Android application in "test" mode, used for testing real-time recognition.

These attributes serve as the identifying features for that chunk of data, and these are the values on which machine learning models are trained. The features used in classification play a tremendous role in the effectiveness of the algorithm, and much of our work goes toward creating and identifying the most useful features. This is discussed in detail in the methodology chapters: Phase I and Phase II.

To easily test and compare different models and parameters, we use the Waikato Environment for Knowledge Analysis, often abbreviated as WEKA [52–54]. WEKA is a very powerful open-source tool developed at the University of Waikato in New Zealand, containing pre-built foundations for many popular machine learning models, such as k-Nearest Neighbors, Bayesian Classification, Decision Trees, and even some basic Neural Networks. WEKA allows us to load our training data, visualize trends, and pick which features we want to use. To run a classifier, we can choose a model from an extensive list, customize the parameters of the model, pick an evaluation technique (including the train/test split method), and run it. This process provides rapid results as to the effectiveness of different model.

3.2.1 Classifiers

The model is the heart of machine learning. Most commonly, machine learning is used to generate classification models or regression models. Regression models act as continuous functions, where multiple input values (features) are condensed to a single numeric output. Classification models also take in multiple input values (features), but outputs a specific *class*, chosen from a pre-defined list. As our goal is to classify a motion as "buckling" or "not buckling", we concern ourselves with classification models. While we elect not to go into great detail on the inner workings of WEKA, but this section provides some high-level explanations of the classifiers which we used in our experimentation. We focused primarily on seven classification algorithms, based on trends in prior literature [10].

Unless otherwise stated, the classifiers were run according to their default configurations [52].

3.2.1.1 k-Nearest Neighbors

The k-Nearest Neighbors (k-NN) classifier, when presented with a test sample, will classify it according to the train samples which are "closest" to it [10, 40]. Conventionally, the distance between samples is calculated using Euclidean distance. k-NN relies on the intuition that similar samples will be of the same class, i.e., that classes can be expected to form clusters. k is significant in that it defines how many neighbors a test sample is compared with. In 1-NN, the test sample is assigned the class of the closest train sample. In 5-NN, the test sample is assigned whichever class appears the most among the 5 closest train samples. The choice of k can have a large impact on the effectiveness of k-NN. In WEKA, the implementation of k-NN is called IBk, short for Instance-based k-NN [55, 56]. The default setting is 1-NN, but can be easily changed. WEKA also provides a "cross validate" option which uses hold-one-out cross validation to determine the best k.

With k-NN especially, normalizing data is important. For instance, if the domain of feature a was significantly larger than the domain of feature b, then feature a would play a disproportionate role when the distance between samples was calculated, and feature b would become effectively meaningless. Fortunately, WEKA provides a lot of control over how distance is calculated. By default, features are automatically normalized, and Euclidean distance is used. We keep these settings.

3.2.1.2 C4.5 Decision Tree

The C4.5 classifier constructs a decision tree, where each "split" represents a threshold for a particular feature [10, 38]. The leaf nodes of this tree represent the final classification. This classifier can be especially useful, as it can be easily modeled in code using nested if-else statements. This makes C4.5 a good, lightweight option for mobile devices with

limited computational resources. In WEKA, the implementation of C4.5 is called J48 [57, 58]. WEKA allows for configuration of the tree pruning process, but we choose to use the default configuration.

3.2.1.3 Multilayer Perceptron

A Multilayer Perceptron classifier is a classic type of neural network with an input layer, an output layer, and hidden layers in-between [38, 59]. In our case, there is an input node for every feature, and a single binary output node indicating the class. Like many neural networks, Multilayer Perceptron uses backpropagation to optimize the network. While Neural Networks can often yield higher accuracies, but require a lot of computation to train. WEKA allows researchers to define the structure of the network's hidden layers. Researchers may add as many layers as they see fit, with however many nodes they see fit. WEKA also provides four wild-card values for specifying layer size: a, i, o, and t.

- $a = \frac{1}{2}$(num. features + num. classes)

  that is, the *average* of the size of the input layer and the size of the output layer

- $i =$ num. features

  that is, the size of the *input* layer

- $o =$ num. classes

  that is, the size of the *output* layer

- $t =$ num. features + num. classes

  that is, the *total* of the size of the input layer and the size of the output layer

The default configuration for Multilayer Perceptron in WEKA contains a single hidden layer of size a, but there is no limit as to the possible width or depth of the network. WEKA also normalizes the training data, by default, though this can be disabled.

3.2.1.4 Naive Bayes

The Naive Bayes classifier is built upon Bayes rule. Using conditional probabilities calculated from the features of the training samples, the class of a test sample is decided according to which is most likely. This process assumes that every feature is independent, even though they almost certainly are not (hence the descriptor *naive*). As such, this could lead to errors when dealing with highly-correlated features [10, 45].

3.2.1.5 Random Forest

The Random Forest classifier works by constructing a series of decision trees, with each tree based on a random subset of the training data [11, 60–63]. Each of these trees is based on WEKA's REPTree classifier. When classifying a new sample, each tree generates a prediction. The trees vote, and the sample is assigned the data point with the most votes.

3.2.1.6 Support Vector Machine

A Support Vector Machine (SVM) attempts to find an optimal hyperplane which separates the training samples in multi-dimensional space [10, 11]. Then, classification can be performed according to those features which are most helpful in separating the two sides of the plane—corresponding to the "support vectors" which appear along the hyperplane. In WEKA, the implementation of SVM is called SMO, based on John Platt's sequential minimal optimization (SMO) algorithm [64–67]. WEKA allows the specification of the underlying calibrator. By default, this is logistic regression. We keep this, as it aligns with convention.

3.2.1.7 ZeroR

The ZeroR classifier is a rudimentary classifier which classifies every new point as the class most common among the training data [38, 68]. ZeroR is most commonly used as a type of "straw man" classification. That is, ZeroR provides a baseline accuracy measure-

ment against which we can compare the accuracy of other classifiers. A classifier could not reasonably be considered useful unless the accuracy at least exceeds that of ZeroR. The name of ZeroR arises from the fact that *zero* features are used in the classification. In comparison, there exists a OneR classifier which classifies samples based on the value *one* feature alone.

3.2.2 Testing

Once we've chosen a classifier, we can feed it our data. Each entry of this data is labeled either "buckling" or "not buckling". WEKA will use this data to train the classification model to identify buckling. That is, WEKA will automatically tune the parameters of these models such that they align with the data as much as possible.

However, we need a method of testing these models. Specifically, we need data that we can use to test the effectiveness of our newly-trained classifiers. Enter the concept of a "train-test split". We have a finite amount of data from our user studies. Obviously, we want to use this data to train our models—the more data, the better. However, we must withhold some of this data so that we can use it to test our models. This withheld data is called "test" data, and the rest of it is called "train" data.

Remember, a classifier is a function which will output a class when given a data sample (which contains a bunch of features). Each of our data samples is labeled with the ground truth—either buckling or not—based on the data collected by the researchers during user studies, and this is what our classifiers are trying to predict. When we train a classifier, WEKA tunes it such that it gets as many of these predictions right as possible. However, this alone is not enough to judge how good the model is. Often, a model can be tuned so well that it gets every prediction right. We must test our model on *unseen data*, i.e., data that wasn't used during the process of training the model. Testing on unseen data ensures that our models generalized well, meaning that they recognize the underlying patterns of

seatbelt buckling in general, not just learning the specific details of the test samples that were provided. This train-test split is central to machine learning. Comparing the predicted test classes with the actual test classes is how we evaluate the quality of our models.

WEKA provides four different options for testing:

- "Use training set": This option evaluates the model on the train set alone. There is no test set in this mode. As already mentioned, this is a poor method of model evaluation. The model is tuned on train data, so its performance on that data will be very high, often 100% perfect. This option is useful only for judging how well the model is fitting the training data before testing occurs.

- "Supplied test set": This option allows us to upload a train data set separately from the test data set. Here, the model is trained on our train data, and tested on our test data. This is the classic method of machine learning model evaluation. However, we must manually split our data set into train and test data before uploading to WEKA.

- "Cross-validation": This option uses a technique known as k-fold cross validation. In k-fold cross validation, the data set is split into k equal parts, or folds. The model is then trained k times, where the test data will be one of the folds, and the train data will be the rest of the folds. Each of these iterations generates a result, and all of these results are aggregated into the final result. Figure 3.5 provides a visualization of 5-fold cross validation. A value of $k = 10$ is fairly common, and this is what we use. The name "cross *validation*" may be confusing in this sense, as we are testing, not validating. Perhaps a better way to think about it would be "cross *testing*". These points which go into these folds are selected randomly. That is, the data set is shuffled, and the test fold is selected using stratified sampling, meaning that the test fold will share the same distribution as the rest of the data. Cross validation is beneficial because it equalizes the variation in performance across the folds, giving

a better understanding of how the model performs across the data in general, rather than on just a single subset of the data.

- "Percentage split": This option is similar to the "supplied test set" option in that the data is split into a train set and a test set. However, instead of manually splitting the data and uploading two files, this option allows us to upload a single file containing our data set and specify a percent at which to split our data into train and test data. For example, a percentage split of 66% indicates that 66% of the data set will become train data, and the remaining 34% will become test data. The data that does into each of these set is shuffled and selected using stratified sampling, meaning that the train and test sets will maintain the distribution of the original data. However, there is an option to preserve the order of the data, in which case the first 66% becomes test and the last 34% becomes train.

Cross validation with $k = 10$ is the default option in WEKA, and we use it for all of our evaluations unless specified otherwise.

3.2.3 Metrics

Of course, this work ultimately comes down to the effectiveness of our classification algorithm, which uses a model generated by machine learning techniques. To measure the performance of our model, we must compare the predicted activities with the actual activities (the ground truth). When comparing a predicted value with an actual value, there are four possible outcomes: a true positive, a true negative, a false positive, and a false negative. These terms are relative to whatever activity we consider to be "positive". In our case, we care the most about how well our model recognizes "buckling", so "buckling" is considered positive, and "not buckling" is considered negative. If the model correctly predicts an instance of "buckling", this is a *true positive*. If the model correctly predicts an instance of "not buckling", this is a *true negative*. If the model incorrectly predicts

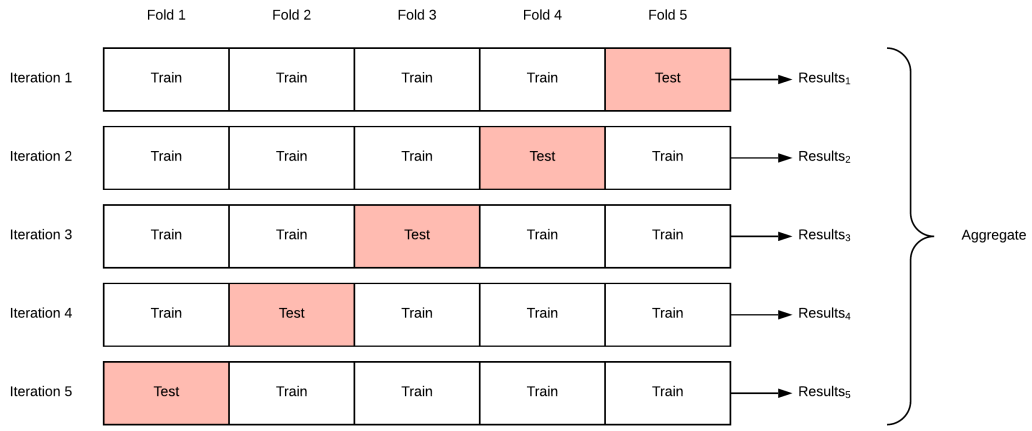


Figure 3.5: A demonstration of 5-fold cross validation. As there ate 5 folds, there are in turn 5 iterations. The results from each of these iterations are aggregated into the final result.

Prediction	Ground Truth	Result	
Buckling	Buckling	TP	True Positive
Not Buckling	Not Buckling	TN	True Negative
Buckling	Not Buckling	FP	False Positive
Not Buckling	Buckling	FN	False Negative

Table 3.1: Terms used when evaluating a model's performance on "buckling" recognition.

"buckling" when the instance was actually "not buckling", this is a *false positive* (also known as a type I error). If the model incorrectly predicts "not buckling" when the instance was actually "buckling", this is a *false negative* (also known as a type II error). These outcomes are detailed in Table 3.1.

When testing our model, it is very important to choose an appropriate evaluation metric. A classic metric is *accuracy*. Accuracy is often the default metric due to its simplicity and comprehensibility. Accuracy represents the percentage of correct predictions among the test data (see Equation 3.1).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.1}$$

While accuracy is helpful, it often fails to give an satisfactory depiction of performance. To build a more complete representation of performance, consider the notions of precision and recall. *Precision* represents how many of the positive classifications were actually positive (see Equation 3.2). *Recall* represents how many of the positive test samples were correctly classified as such (see Equation 3.3). Both precision and recall fall within the range $[0, 1]$, where 1 is the best possible value.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

Calculating the harmonic mean of precision and recall results in the *F-measure*, also known as F-score or F_1 score (see Equation 3.4). As such, the range of F-measure is $[0, 1]$, where 1 is the best possible outcome. F-measure is especially useful in areas where accuracy falls short. For example, an accuracy measure is biased toward data categories with greater representation. This is important to recognize in our problem: the majority of data collected will be "not buckling" data, whereas only a small portion of data will be "buckling" data. If a model were to simply classify everything as "not buckling", it would benefit from a very high accuracy, though it failed to recognize any instances of buckling. For this reason, F-measure is much more useful. The F-measure for "buckling" will be indicative of how well the model classified "buckling", independent of the "not buckling" performance. In our research, the "buckling" F-measure will be our primary metric of evaluation.

$$\begin{aligned}\text{F-measure} &= \left(\frac{\text{Precision}^{-1} + \text{Recall}^{-1}}{2}\right)^{-1} \\ &= 2\left(\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}\right) \\ &= \frac{2\,\text{TP}}{2\,\text{TP} + \text{FP} + \text{FN}}\end{aligned} \tag{3.4}$$

4. STUDY PHASE I*

This chapter details the research surrounding our first user study. In section 4.1, we describe the study itself. In section 4.2, we conduct *feasibility testing* to see whether seatbelt motions are unique enough to be classified. In section 4.3, we conduct *advanced testing* to try and get better results by using observations from the data. In section 4.4, we conduct *real-time testing* to assess our algorithm's ability to perform in real-world environments.

4.1 User Data

When we began studying seatbelt activity, we first wanted to ensure that a seatbelt-buckling motion could be feasibly distinguished from other similar patterns of motion. With this in mind, we developed our first study. A seatbelt-buckling motion generally consists of an arm-raising motion (wherein the user reaches up and grab the seatbelt) followed by an arm-lowering motion (wherein the user brings the seatbelt back down and buckles it in). Naturally, our study would request participants to buckle their seatbelts; however, participants were also requested to perform a number of "control motions". Specifically, users were requested to perform the following actions:

- **Buckling a seatbelt.**
- Removing something from a shirt pocket.
- Putting a phone in a pants pocket after sending a text.
- Putting a phone in a pants pocket after ending a phone call.
- Putting on a backpack.

*Part of the data reported in this chapter is adapted with permission from "Recognizing Seatbelt-Fastening Activity Using Wearable Sensor Technology" by Jake Leland and Ellen Stanfill, 2017 [69].

- Taking off glasses/sunglasses.
- Putting on a jacket.
- Reaching up and touching one's face or adjusting hair.

These control motions were carefully chosen to mimic the arm-raising/arm-lowering pattern of seatbelt motion. Presumably, if a seatbelt motion is distinguishable from the other control actions, the recognition is worth pursuing further.

This stage of the study collected data from twelve mixed-age research participants (five female, seven male). Users buckled their seatbelt 10 times, then performed each of the seven control actions 5 times. Here, data was collected in a discrete fashion, meaning that data was collected only for the duration of the action itself—the researcher started data collection, directed the participant to perform the action, then stopped data collection. Although this is not truly representative of natural motion patterns, but it provided a good baseline to start comparing isolated instances of actions and making distinctions.

Once we began conducting user studies, it became apparent that that users fasten their seatbelts in primarily three different ways:

1. Reaching up with their *left* hand, then bringing the seatbelt all the way down to buckle it.
2. Reaching up with their *right* hand, then bringing the seatbelt all the way down to buckle it.
3. Reaching up with their left hand, *transferring* the seatbelt from their left hand to their right hand, then fastening the buckle with their right hand.

This realization added a degree of complexity to our study, especially considering that only one wrist is being monitored by sensors. To provide data samples as uniform as

possible, the user was directed to wear the Pebble watch on whichever arm they used to perform the initial upward reach, as this arm is most important in performing the characteristic arm-raising/arm-lowering motion.

At the end of collection, we had sampled a total of 120 examples of buckling, plus 60 examples of each additional activity, for a total of 540 distinct samples.

4.2 Feasibility Testing

Before diving too deep into creating the "perfect" model for our classification, we opted to first conduct a sort-of feasibility test. Here, we take the most basic approach to classifying our data, using only a few arbitrary and conventional configurations. This bare-bones approach was not expected to perform remarkably, but was at least expected to give a basic sense of whether the problem was worth pursuing further, before we devoted too much time toward investigating the data.

4.2.1 Data Processing

With the Pebble watch's accelerometers providing data at a rate of 25 Hz, our user study provided a large amount of raw data. In its raw form, this data is not immediately useful. To use machine learning, we must extract features from the data. However, before we can extract features, we must clean it up. This step is often referred to as "preprocessing" [15], or simply "processing". Note that feature generation may also be considered part of the processing step, but we make a distinction.

4.2.1.1 Sorting

Due to the nature of Bluetooth transmission from the Pebble watch to the Android phone, the data packets often arrive out-of-order. When the Android application receives the raw data packets from the Pebble application, it immediately appends them to a list, which is subsequently written to a CSV file. As such, the samples in this list are ordered

according to when the Android received the sample, rather than when the Pebble recorded the sample. Fortunately, the Pebble watch records a timestamp along with every data sample. Whenever the CSV file is extracted from the Android phone, we must sort the data according to this timestamp before proceeding with anything else.

4.2.1.2 Uniquification

Also due to the nature of Bluetooth transmission, there is a high level of redundancy when sending data. This is to guard against data loss. Again, the Android application writes all of this information directly to a CSV file. As such, duplicate samples appear often in this list. These duplicate entries are not only unnecessary, but can actually interfere with later processing and feature extraction. For this reason, we must "uniquify" the data, i.e., remove duplicate entries from the list of samples.

4.2.2 Extracting Features

Once our data has been processed, we can begin to extract features from the data. Briefly introduced in chapter 3, features are the identifying characteristics of a chunk of data. Machine learning models learn to classify chunks of data according to their features, so feature selection is very important. Herein lies the creativity of machine learning: researchers suggest features which they believe may provide useful information for their particular classification problem. These features may come from classic, well-established statistical features, or they may be invented from scratch to better capture unique patterns in the data.

4.2.2.1 Feature Window

The notion of a "chunk"—or "window"—of data is also important to establish. Conventional machine learning models view only one small window of data at a time, predicting a class for that particular window before moving on to the next. Depending on the

classification problem, it may make sense for a model to classify a particular activity for multiple windows in a row. For example, a few-second window of data may be enough to recognize if a user is running, but there could be many consecutive "running" windows. Conversely, for an action that occurs in a discrete moment, like buckling a seatbelt, the entire action may fall within the timeframe of one window. Clearly, the size of the window can have a large impact on recognition, as the window size becomes the fundamental unit of time, with each of these units being classified—in our case, being identified as either "buckling" or as "not buckling".

At this stage of our research, we pick a 0.5-second window. This window size is completely arbitrary, as we are only performing feasibility testing. Later, during advanced testing (see section section 4.3), we select a window size that is founded in the data.

4.2.2.2 Feature Set

Once a window size has been established, we can extract features from the window. As mentioned prior, these features can be very simple (such as basic statistical properties) or very complex (such as custom properties based on the appearance of the data). The creation and selection of features is a core part of machine learning, and is one of the primary aims of this research. Based on the work of Figo et al., we direct our efforts toward time-domain features [15].

Again, as we are performing only feasibility testing thus far, we choose a handful of basic statistical features. These features are detailed by Equation 4.1–Equation 4.12 below. Table 4.1 also gives a concrete example of how a 0.5-second window of data (Table 4.1a) is converted to a feature vector (Table 4.1b).

μ_X, μ_Y, μ_Z: This is the average value of the points on each axis.

$$\mu_X = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.1}$$

$$\mu_Y = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{4.2}$$

$$\mu_Z = \frac{1}{n}\sum_{i=1}^{n} z_i \tag{4.3}$$

x_{min}, y_{min}, z_{min}: This is the minimum value found on each axis.

$$x_{\text{min}} = \min_{x_1 \ldots x_n} X \tag{4.4}$$

$$y_{\text{min}} = \min_{y_1 \ldots y_n} Y \tag{4.5}$$

$$z_{\text{min}} = \min_{z_1 \ldots z_n} Z \tag{4.6}$$

x_{max}, y_{max}, z_{max}: This is the maximum value found on each axis.

$$x_{\text{max}} = \max_{x_1 \ldots x_n} X \tag{4.7}$$

$$y_{\text{max}} = \max_{y_1 \ldots y_n} Y \tag{4.8}$$

$$z_{\text{max}} = \max_{z_1 \ldots z_n} Z \tag{4.9}$$

$X \times Y$, $X \times Z$, $Y \times Z$: These "multiplication" operations were performed by pairwise multiplying the axes together, which resulted in a product axis, then taking the average of that axis. This operation approximates a crude representation of correlation.

$$X \times Y = \frac{1}{n} \sum_{i=1}^{n} x_i y_i \tag{4.10}$$

$$X \times Z = \frac{1}{n} \sum_{i=1}^{n} x_i z_i \tag{4.11}$$

$$Y \times Z = \frac{1}{n} \sum_{i=1}^{n} y_i z_i \tag{4.12}$$

4.2.3 Evaluation

Using the 12 features outlined above, we ran the seven classifiers described in section 3.2. The accuracy results are shown in Table 4.2.

While these results seem nice, it is important to remember that accuracy is often not a very good metric to use for evaluating performance in the case of unbalanced data. Also note that ZeroR yields a 79.0% accuracy, so this is our baseline. As already described in section 3.2, we elect to use F-measure as our primary means of evaluation. Table 4.3 shows the F-measures of each of the seven classifiers. Remember that there is a different F-measure for each class, effectively telling us how well the model performs for that case specifically.

It is also useful to investigate the *confusion matrix* of each classifier, as this provides insight into the types of errors that are occurring. Table 4.4 demonstrates the general format of a confusion matrix. For the sake of readability, we do not include the confusion matrices within the body of this text; however, they are all included in the appendices. The confusion matrices for each of the seven classifiers are shown in Table A.1–Table A.7.

Based on these results, Random Forest performed the best. However, no classifier performed particularly well. The confusion matrices revealed a lot of errors, both false

Time Epoch (ms)	X (mG)	Y (mG)	Z (mG)	Activity
1479433861850	173	93	-1064	Buckling
1479433861900	33	163	-734	Buckling
1479433861940	-125	221	-393	Buckling
1479433861980	-221	204	-307	Buckling
1479433862020	-385	326	-91	Buckling
1479433862060	-809	245	419	Buckling
1479433862100	-554	-168	464	Buckling
1479433862140	-707	3	512	Buckling
1479433862180	-715	25	618	Buckling
1479433862220	-736	484	591	Buckling
1479433862260	-327	844	2522	Buckling
1479433862300	-673	-413	996	Buckling
1479433862340	-684	-1	838	Buckling

(a) A 0.5-second window of raw Pebble sensor data, including a timestamp (measured in milliseconds), the accelerometer values for each axis (measured in milli-Gs, where G is standard gravity), and the class.

Feature	Value
Average X	-440.769230769
Average Y	155.846153846
Average Z	336.230769231
Minimum X	-809.00
Minimum Y	-413.00
Minimum Z	-1064.00
Maximum X	173.00
Maximum Y	844.00
Maximum Z	2522.00
$X \times Y$	-68692.1893491
$X \times Z$	-148200.177515
$Y \times Z$	52400.2721893
Activity	Buckling

(b) The features generated from the 0.5-second window of data.

Table 4.1: An example of a window of raw data points being converted to a feature vector.

Classifier	Accuracy
IBk	88.4%
J48	86.7%
Multilayer Perceptron	87.8%
Naive Bayes	84.7%
Random Forest	**90.1%**
SMO	85.0%
ZeroR	79.0%

Table 4.2: Classifier accuracies for Phase I feasibility testing.

Classifier	Activity	F-measure
IBk	Buckling	0.727
	Not Buckling	0.926
J48	Buckling	0.702
	Not Buckling	0.914
Multilayer Perceptron	Buckling	0.716
	Not Buckling	0.922
Naive Bayes	Buckling	0.685
	Not Buckling	0.899
Random Forest	**Buckling**	**0.760**
	Not Buckling	0.938
SMO	Buckling	0.604
	Not Buckling	0.907
ZeroR	Buckling	0.000
	Not Buckling	0.883

Table 4.3: Classifier F-measures for Phase I feasibility testing.

Positive	Negative	← classified as
TP	FN	Positive
FP	TN	Negataive

Table 4.4: General format of a confusion matrix.

negatives and false positives.

That said, great results were not the goal of this stage of testing. We were testing whether this classification task was feasible at all. Though our results are not great, they are promising. There is potential for future models to recognize seatbelt motion.

4.3 Advanced Testing

Feasibility testing confirmed that seatbelt motion was recognizable. The next step was to study the data more closely, make more intelligent parameter decisions, and extract more useful (and novel) features.

4.3.1 Data Processing

4.3.1.1 Smoothing

When observing graphs of our raw accelerometer data (see Figure 4.2a), it became apparent that the data was very noisy. Moving forward, we choose to "smooth" the data as part of our processing step.

At a rate of 25 Hz, the Pebble watch sensors provide a high level of data granularity. While this is not inherently bad, it can result in data that is too noisy for our application. That is, while sensitive motion data may be useful for more detailed observations, it is less useful in our case. The motions involved when buckling a seatbelt are large, sweeping gestures, so we are much more concerned with the broad, high-level view of the data. Concretely, we care more about the gross motion of the arm than we do the fine tremors of the hand. In this case, such fine detail manifests itself mostly as noise. To ensure that our models focused more on the broad motions and avoided over-fitting on tremors, we ran our data through a smoothing function before extracting features (see Figure 4.2b), as recommended by literature [15]. This served two purposes:

1. The smoothing function filtered out fine movements and other anomalies, leaving

only the broad motions that we were most concerned with.

2. The smoothing function made it easier for us to visually analyze graphs of the data, identifying motion patterns as well as potential features.

We used a rolling average (also called a moving average) function to implement smoothing. At a high level, a rolling average function works by averaging each point with some of its neighboring points. The number of points included in each averaging is defined as the window size w. Note that this window is distinct from the window used in feature generation. This means that each data point p will be replaced with the average of itself and the $w-1$ surrounding points (with p centered in the window). This process is defined by Equation 4.13 and is visually demonstrated in Figure 4.1.

To smooth point p_i (the ith data point of axis p) using window size w:

$$p'_i = \frac{1}{w} \sum_{j=i-\lceil \frac{w-1}{2} \rceil}^{i+\lfloor \frac{w-1}{2} \rfloor} p_j \tag{4.13}$$

The rolling average window size w determines the balance of a trade-off between noise and information. If w is too small, the function will fail to remove noise. Conversely, if w is too large, the function will crudely remove important information. In the earlier stages of our research, we arbitrarily set $w = 10$. This was selected with the initial intent of assisting with visual examination. A window size of 10 appeared to reduce noise sufficiently without masking important information. The effect of $w = 10$ data smoothing on three different buckling instances is shown in Figure 4.2. Later, we will select w more carefully.

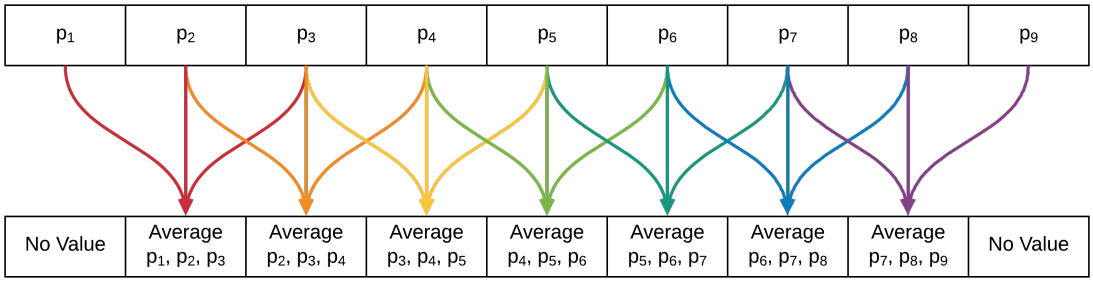


Figure 4.1: Demonstration of how a rolling average function works to smooth data. This graphic shows a window size $w = 3$. The colors exist for visual clarity and bear no significance.

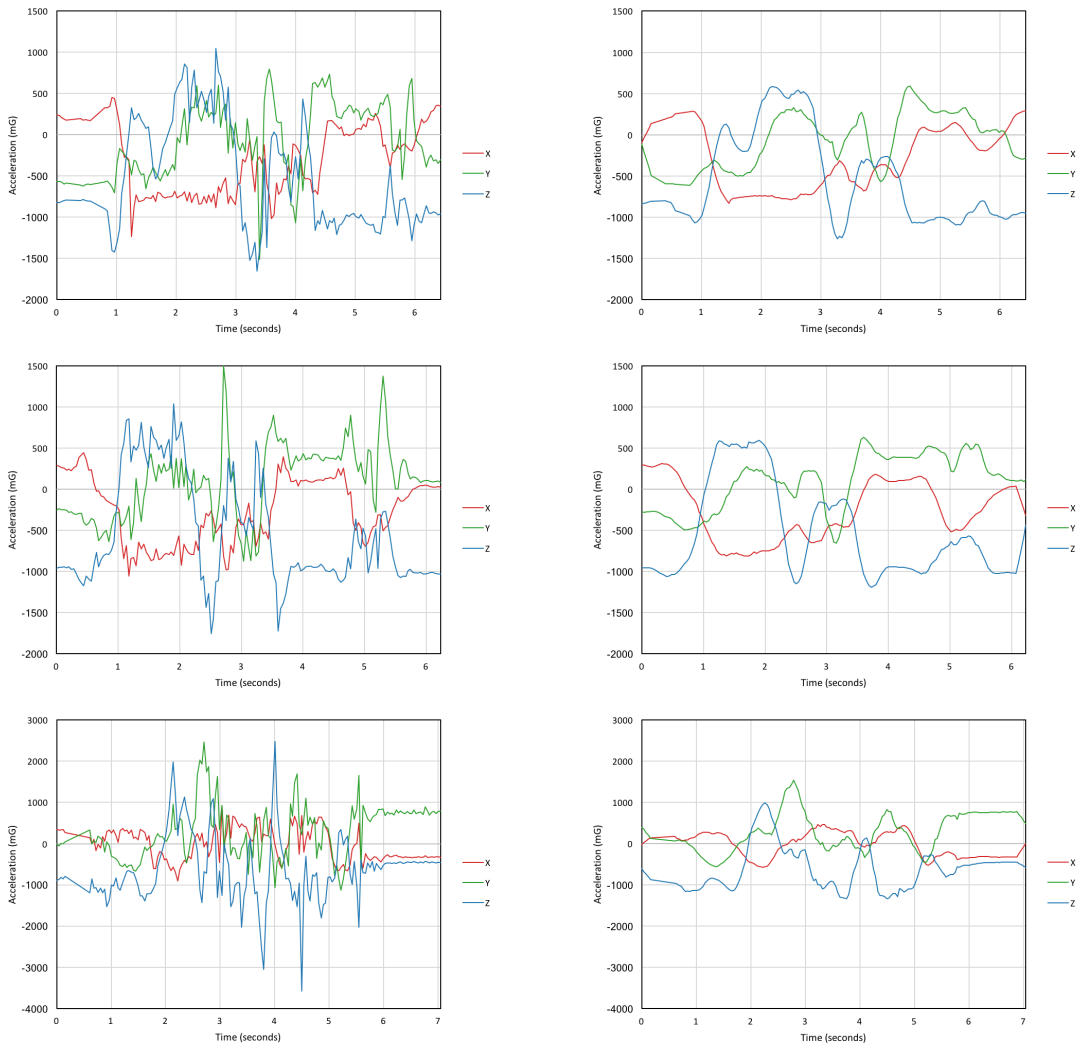
4.3.2 Extracting Features

During feasibility testing, the traditional features that we used were very generic, and not very specific to the seatbelt motion itself. To better model features for this data, we plotted individual instances of buckling and began to look for patterns which might inspire custom features.

One of the first things that we identified in the data plots was the distinctive arm-raising and arm-lowering motion. This effectively forms two "halves" to the motion: raising followed by lowering. Note that because the user wore the watch on the arm that they use to grab the seatbelt, this general structure holds regardless of how the user buckles their seatbelt. An example of a seatbelt buckling motion is shown in Figure 4.3, with the arm-raising period and arm-lowering periods highlighted.

4.3.2.1 Feature Window

Also, when investigating seatbelt motion, it became clear that the arbitrarily-picked feature window size of 0.5 seconds was not very appropriate for recognizing a seatbelt motion. Given the nature of seatbelt buckling, the activity is more of a distinct, contained motion that occurs once. As such, it would be more beneficial if the entire motion could fit



(a) Raw accelerometer data, as recorded by the Pebble watch.

(b) Smoothed accelerometer data, using a rolling average with window size 10 points.

Figure 4.2: The 3-axis accelerometer data corresponding to three separate instances of seatbelt buckling, before and after the smoothing function.

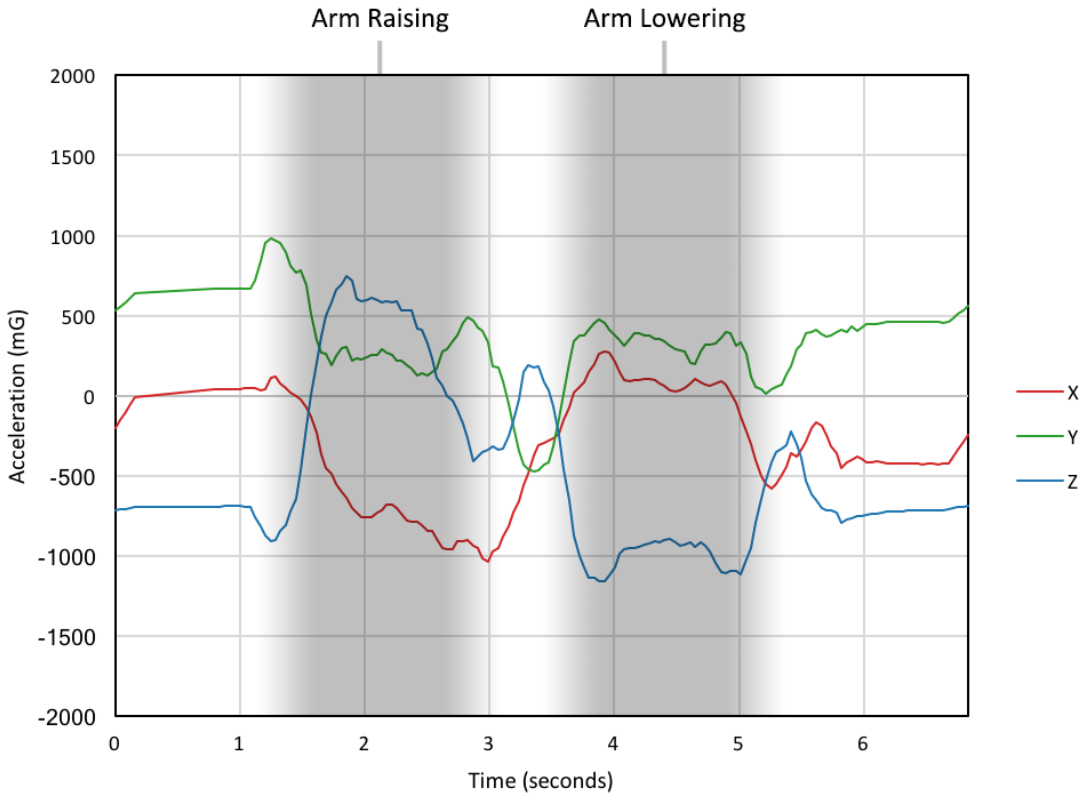


Figure 4.3: A plot of smoothed accelerometer data for one instance of buckling. The first "half" of the data corresponds to the period where the user lifts their arm to grab the buckle. The second "half" corresponds to the period where the user lowers their arm to fasten the buckle. The data shown in this graph was smoothed by rolling average with window size 10.

within a single window. With this in mind, we expanded the window size from 0.5 seconds to 5 seconds. The choice of 5 seconds was somewhat arbitrary, but based roughly on the average duration of the seatbelt motions observed.

Additionally, we identified the need for sliding windows. With such a large window size of 5 seconds, if we were to split our data evenly into 5-second chunks, there would be a high level of variation as to the position of seatbelt motions within that 5-second window. Put more simply, a seatbelt motion would likely not be centered in the window, and there is a chance of a seatbelt motion "straddling" two windows. Using sliding windows mitigates this risk by allowing windows to overlap. In our case, we chose a 3-second window overlap, meaning that each new window will overlap the previous by 3 seconds, i.e., a new window will be created every 2 seconds. This is illustrated in Figure 4.4. With this, there is a greater level of uniformity between each occurrence of seatbelt buckling, in relation to where the motion falls within the range of the window. For now, the overlap amount was selected arbitrarily. Later in our research, we select the window size and window overlap more analytically.

4.3.2.2 Window Splitting

Our window now fit to the entire seatbelt gesture more appropriately. However, if the entire gesture fits within a single window, we lose the ability to differentiate between the first half of the window and the second half of the window. To better capture the arm-raising and arm-lowering stages of the gesture, we needed to isolate the first and second half of the data. To accomplish this, we calculated all of our features twice more: once over the first half of the window, and once over the second half of the window. We call this "feature splitting"—in this case, the window is split in half. Combined with the original feature calculations performed over the entire window, this leaves us with triple the initial feature count.

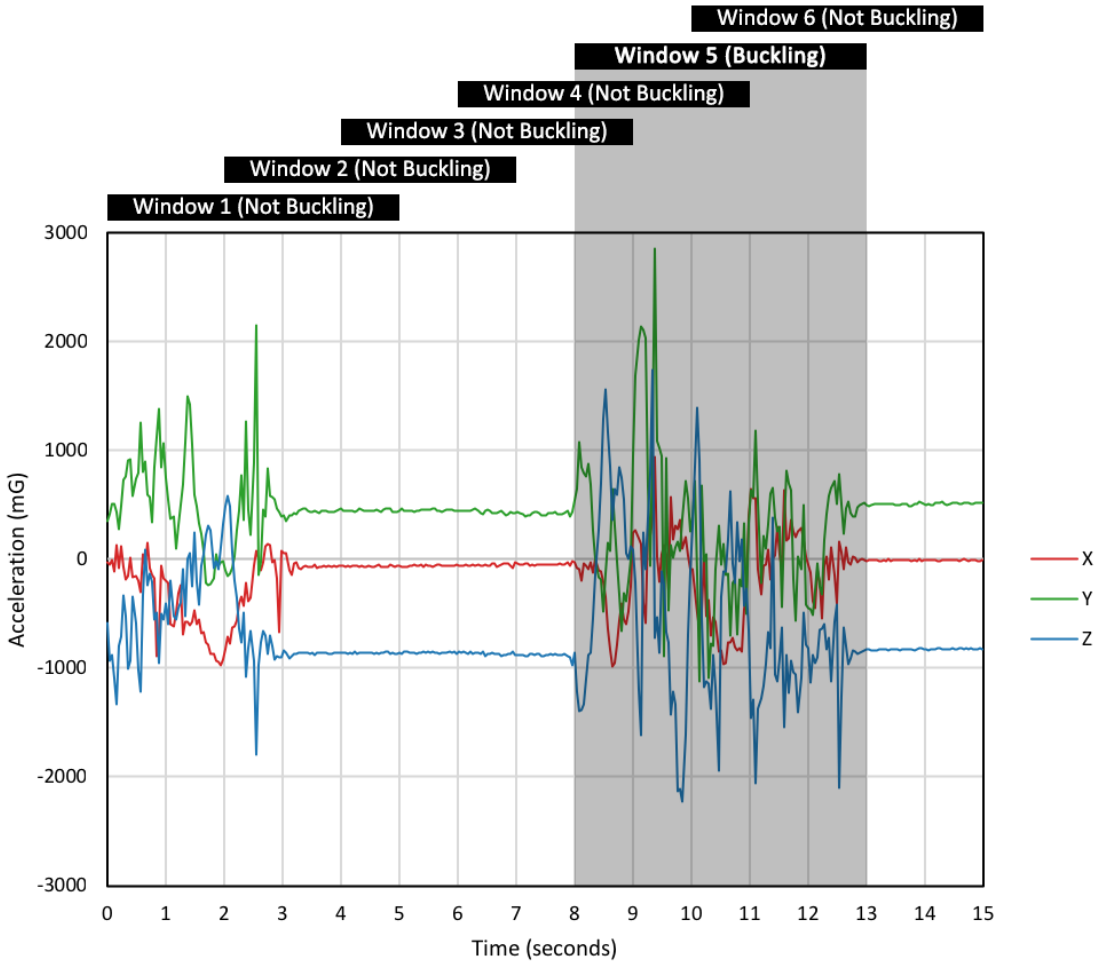


Figure 4.4: An illustration of the sliding window used when extracting features from the accelerometer data. The sliding window shown has a window size of 5 seconds and a window overlap of 3 seconds. In this example, the buckling instance falls within the fifth window, and is highlighted accordingly.

4.3.2.3 Feature Set

Plotting the gestures helped to reveal where the initial features fell short, and it helped to generate ideas for new features. We needed novel features which corresponded more directly with the seatbelt activity. Consider Figure 4.3. While the arm is being raised, the Y and Z axes appear similar in value. While the arm is being lowered, the X and Y axes appear similar and value. Throughout the gesture, the X and Y axes appear directly related, while the Z axis appears inversely related.

With this in mind, we created 8 novel features, shown in Equation 4.14–Equation 4.21 below. In each case, the formulas are applied pairwise, effectively generating a new axis. The average value of this axis is taken as the feature value. As with the previous features, these were calculated three times each: once over the first half of the window, once over the second half of the window, and once over the entirety of the window.

$X - Y, X - Z, Y - Z$: This is the average difference between each pair of axes. Based on the observation that certain axes appear very close in value during certain parts of the window, we added features representing the differences between the axes, effectively measuring how close they were. Note that this value may be positive or negative, depending on which axis is "on top".

$$X - Y = \frac{1}{n}\sum_{i=1}^{n} x_i - y_i \tag{4.14}$$

$$X - Z = \frac{1}{n}\sum_{i=1}^{n} x_i - z_i \tag{4.15}$$

$$Y - Z = \frac{1}{n}\sum_{i=1}^{n} y_i - z_i \tag{4.16}$$

$|X - Y|$, $|X - Z|$, $|Y - Z|$: This is the average of the absolute difference between each pair of axes. This is almost identical to the previous features, except that we take an absolute value. This truly measures how close the axes were, without regard for which axis was "on top".

$$|X - Y| = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i| \tag{4.17}$$

$$|X - Z| = \frac{1}{n} \sum_{i=1}^{n} |x_i - z_i| \tag{4.18}$$

$$|Y - Z| = \frac{1}{n} \sum_{i=1}^{n} |y_i - z_i| \tag{4.19}$$

$|X - Y| - |Y - Z|$: This is the difference between the X-Y absolute difference and the Y-Z absolute difference. This feature was based on the following observations:

- During the first half of the window, there exists a large difference between the X and Y axes (large $|X - Y|$ value) and a small difference between the Y and Z axes (small $|Y - Z|$ value).
- During the second half of the window, there exists a small difference between the X and Y axes (small $|X - Y|$ value) and a large difference between the Y and Z axes (large $|Y - Z|$ value).

Because these comparisons "flip" between the first half and the second half of the window, observing the difference between the values could be an interesting feature.

$$|X - Y| - |Y - Z| = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i| - |y_i - z_i| \tag{4.20}$$

Classifier	Accuracy
IBk	99.8%
J48	97.9%
Multilayer Perceptron	**100.0%**
Naive Bayes	99.4%
Random Forest	99.4%
SMO	99.8%
ZeroR	89.2%

Table 4.5: Classifier accuracies for Phase I advanced testing.

$||X-Y|-|Y-Z||$: This is the absolute difference between the X-Y absolute difference and the Y-Z absolute difference. This is closely related to the previous feature, but is now the absolute difference, without respect to the order of the calculation. Theoretically, this value should remain somewhat consistent between both halves of the window.

$$||X-Y|-|Y-Z|| = \frac{1}{n}\sum_{i=1}^{n} ||x_i-y_i|-|y_i-z_i|| \tag{4.21}$$

4.3.3 Evaluation

With our new processing techniques and new features, we ran the same classifiers again.

Looking at the accuracy results (see Table 4.5), we can already see that the models perform much better than before. All seem nearly perfect. However, we should be suspicious of 100% accuracy.

Observing the reported F-measures (see Table 4.6) confirms that the models are performing much better than before, with all Buckling F-measures at least 0.9. Again, however, we should be suspicious of the 1.0 F-measure.

Based on the confusion matrices (see Table A.8–Table A.14), we can see that our models generally have very little error. False negatives occur slightly more frequently

Classifier	Activity	F-measure
IBk	Buckling	0.990
	Not Buckling	0.999
J48	Buckling	0.902
	Not Buckling	0.988
Multilayer Perceptron	**Buckling**	**1.000**
	Not Buckling	1.000
Naive Bayes	Buckling	0.970
	Not Buckling	0.996
Random Forest	Buckling	0.970
	Not Buckling	0.996
SMO	Buckling	0.990
	Not Buckling	0.999
ZeroR	Buckling	0.000
	Not Buckling	0.943

Table 4.6: Classifier F-measures for Phase I advanced testing.

than false positives, presumably because our data set is more heavily weighted toward "not buckling" data.

The confusion matrices also help to reveal why we are seeing "perfect" results. In significantly expanding the size of the feature window (from 0.5 seconds to 5 seconds), we in turn significantly reduced the number of training data points (from 3,459 samples to 472 samples). Given the complexity of our feature set, this is not enough data to generalize well.

In spite of this, it is at least clear that our improved approach to data processing combined with our novel features made a tremendous difference in classification performance. We had realized the goal of the Phase I user study: to ensure that seatbelt motions were distinguishable from other similar motions.

4.4 Real-Time Testing

Before moving on, we paused to test the real-time validity of our approach. It was at this time that we began developing the "test" mode of our Android application, referenced in section 3.1. Up to this point, most of our data analysis was performed using Python scripting. To integrate this into a real-time Android application, we translated our algorithm into Java. This enabled the Android phone to extract features from data windows in real time, as the data flowed from the sensor. While translating the basic algorithm is simple, we must select a classifier that can be statically loaded onto the device itself. This presents two challenges:

- The model should be capable of running in a reasonable amount of time given limited computational resources.
- The model itself should require minimal storage space.

With these two restrictions, we elected to use the J48 decision tree classifier for the Android application. Though the J48 classifier did not perform the best, it is well-suited for conversion into static code. When generating a J48 model, WEKA provides the actual branching structure of the decision tree. This makes it easy to convert the model into a series of if-else statements, which are very lightweight compared to importing a WEKA model in its entirety.

With a classifier embedded in the application code, features can be processed as soon as they are extracted from the data, and the application can classify the current activity as either "buckling" or "not buckling". Refer back to Figure 3.2 for a diagram of this process. When the application detects buckling, it will notify the user. This provides an easy way to check whether the recognition is working.

4.4.1 Supplementary Data

When we first ran the application, it became immediately apparent that our models were insufficient. Without moving at all, the application was continuously recognizing seatbelt activity—it was completely unusable. In hindsight, this result should not have been surprising. Recall the user study section 4.1: we collected discrete samples of seatbelt buckling, plus a few other distinct activities. We had *not* collected any sort of data toward the everyday, average patterns of motions that the Pebble watch would most frequently experience. In our case, we had not collected any data of users standing still, so our models had nothing to train against. It just so happened that our decision tree classified stillness as "buckling".

Given the lack of naturalistic training data, we could not expect our model to perform in real time with any degree of feasibility. To correct this, we recorded a few minutes' worth of naturalistic data ourselves: standing, walking around, climbing stairs, getting in and out of a vehicle, etc. During this time we also buckled our seatbelts a few times, to test real-time recognition. Note that this was not a formal user study. We, as the researchers, simply supplemented the initial data set with some additional data to train against. This type of data collection will be formalized in the next chapter.

4.4.2 Evaluation

As the application is always collecting data, even if it is in testing mode, we can offload the data and test it offline alongside our initial training data. This allowed us to run all seven classifiers again, this time with the supplemental data.

With the inclusion of this new data, accuracy metrics (see Table 4.7) dropped slightly. While still in the 90% range, there is more variation, and there are no "perfect" classifiers. Based on accuracy alone, Multilayer Perceptron still performs the best, though only slightly. Again, accuracy measurements should not be the focus of our analysis.

Classifier	Activity
IBk	97.0%
J48	96.5%
Multilayer Perceptron	**97.6%**
Naive Bayes	90.4%
Random Forest	97.5%
SMO	97.3%
ZeroR	93.6%

Table 4.7: Classifier accuracies for Phase I real-time testing.

Classifier	Activity	F-measure
IBk	Buckling	0.775
	Not Buckling	0.984
J48	Buckling	0.734
	Not Buckling	0.981
Multilayer Perceptron	**Buckling**	**0.825**
	Not Buckling	0.987
Naive Bayes	Buckling	0.550
	Not Buckling	0.946
Random Forest	Buckling	0.772
	Not Buckling	0.987
SMO	Buckling	0.769
	Not Buckling	0.986
ZeroR	Buckling	0.000
	Not Buckling	0.967

Table 4.8: Classifier F-measures for Phase I real-time testing.

Investigating the F-measures (see Table 4.8) and confusion matrices (see Table A.15–Table A.21) reveals a better picture of the effects of adding some naturalistic data. There is, across the board, a notable drop in the buckling F-measures. Now, we have values that are more expected for real-world recognition scenarios. With an F-measure in the 0.80s, Multilayer Perceptron again performed the best. However, all other classifiers are in the 0.70s or below. Clearly, there is great potential for naturalistic recognition, but more data is needed to train the models effectively and to trust the results.

5. STUDY PHASE II

This chapter details the research surrounding our second user study. In section 5.1, we describe the study itself. In section 5.2, we conduct *baseline testing* to see how our existing algorithm from Phase I performs on the new data. In section 5.3, we conduct *advanced testing* to try and be smarter about how we choose model parameters. In section 5.4, we conduct *leave-one-out testing* to assess how well our algorithm would perform on new users. In section 5.5, we conduct *dimensionality testing* to investigate the size of our feature set.

5.1 User Data

In Phase I of our study, we learned that seatbelt-buckling motions are distinguishable from other similar motions. We also learned that lab-collected data is often not representative of real-world data. Supplementing our initial dataset with some semi-naturalistic data revealed two things:

1. Our initial results were overly optimistic and didn't translate to real-time performance.

2. We would need a lot more naturalistic data if we wanted to build a reliable classifier.

With these findings, we set out to design a second user study. It was very important that this study was naturalistic, i.e., more representative of the type of data that would be collected by a smart watch in everyday scenarios. So, rather than giving the users a specific course of action and starting and stopping collection for every instance of an activity, we let the application run continuously. As such, the study consisted of the following:

- A lot of "ambient" data, such as walking, standing, opening doors, getting in and out of the vehicle, etc.

- At least 10 instances of the user buckling their seatbelt.

As the data was collected continuously, it was important that the researcher made note of the times when the user was actually buckling their seatbelt. It was at this time we redesigned our Android application to allow the researcher to hold down a button while the user was buckling, thereby annotating the data in real time and making naturalistic data analysis much more manageable section 3.1.

Fourteen users participated in this study. As before, this group was of mixed age and gender. Six of these participants had participated in the Phase I study. This study provided roughly an hour and a half worth of naturalistic data, during which there were 184 instances of buckling.

This study, combined with the Phase I study (plus the supplementary real-time data) gave us 276 "buckling" instances from 20 different users. The number of "not buckling" instances depended greatly on the feature window size and overlap.

5.2 Baseline Testing

Before we jumped into improving our model to better handle naturalistic environments, we elected to build the classifier just as in Phase I of our research, but this time with our new data.

5.2.1 Evaluation

The accuracy results (see Table 5.1) show a notable drop in performance. Only the top three classifiers remained in the 90% range: Multilayer Perceptron, Random Forest, and IBk, respectively.

The F-measure results (see Table 5.2) are less straightforward. Compared to the previous results from Phase I, the "buckling" results were split. Some of the classifiers (IBk, Naive Bayes, and Random Forest) performed better, while others (J48, Multilayer Percep-

Classifier	Accuracy
IBk	91.2%
J48	87.7%
Multilayer Perceptron	**92.7%**
Naive Bayes	76.8%
Random Forest	92.6%
SMO	88.8%
ZeroR	80.0%

Table 5.1: Classifier accuracies for Phase II baseline testing.

tron, and SMO) performed worse. Though we don't normally concern ourselves with the "not buckling" F-measures, it is worth pointing out that they had all dropped.

Examining the confusion matrices (see Table B.1–Table B.7) confirms what the F-measures were revealing. Overall, the results are less consistent. While some of the "buckling" classifications improved, just as many worsened. Plus, all of the "not buckling" classifications worsened. This evidence suggests that our models are no longer fitting the data as well as we would need. Since the Phase II study added a lot more data variety than was captured by the Phase I study, it has become more difficult to differentiate between "buckling" and "not buckling". This imposes the need to improve and tweak our models more carefully as we continue.

5.3 Advanced Testing

During this Phase I of our study, many of our data handling techniques and parameters were arbitrary chosen. While this was sufficient for classifying our lab-collected data, it was not sufficient for scaling up to naturalistic data collection. During Phase II, we wanted to give more care to the tuning of our models.

5.3.1 Data Processing

As with Phase I, our data processing consisted of three consecutive steps:

Classifier	Activity	F-measure
IBk	Buckling	0.792
	Not Buckling	0.944
J48	Buckling	0.698
	Not Buckling	0.923
Multilayer Perceptron	**Buckling**	**0.815**
	Not Buckling	0.954
Naive Bayes	Buckling	0.603
	Not Buckling	0.836
Random Forest	Buckling	0.798
	Not Buckling	0.955
SMO	Buckling	0.678
	Not Buckling	0.933
ZeroR	Buckling	0.000
	Not Buckling	0.889

Table 5.2: Classifier F-measures for Phase II baseline testing.

1. Sorting the data

2. Uniquifying the data

3. Smoothing the data

Sorting and uniquifying are fairly straightforward, so here we focus on smoothing.

5.3.1.1 Smoothing

Recall section 4.2, where we first began smoothing our data. The data was smoothing using a rolling average function with window size w (remember, this window size should not be confused with the window size for extracting features). We arbitrarily chose $w = 10$ based mostly on the visual appearance of the data plots. Here, we choose to investigate this value more closely.

We already had processed data with a rolling average window size $w = 10$. For this test, we processed our data 14 more times, with rolling average window sizes $w =$

$1 \ldots 9, 11 \ldots 15$. Note that smoothing with $w = 1$ means that there is no smoothing whatsoever, as each point becomes the average of only itself. We then trained three classifiers—IBk, Multilayer Perceptron, and Random Forest—15 times, each with a different amount of smoothing. We chose to conduct our test with these three classifiers because they were consistently the top performers. The results of these 45 tests are shown in the form of a line graph in Figure 5.1a. As classifier performance often experiences some variance due to randomness, this graph also includes quadratic trend lines for each classifier. To aid in visualization, we also take the average performance of all three classifiers for each window size and generate a second plot of the line graph and quadratic trendline, shown in Figure 5.1b.

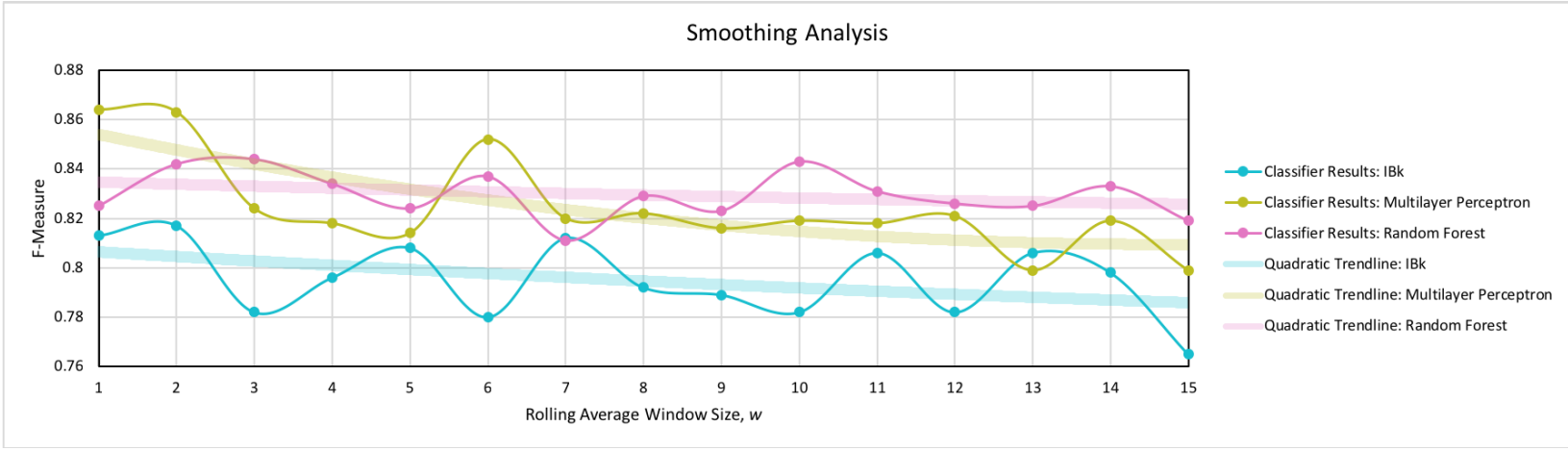
The results shown in Figure 5.1 are quite surprising. According to the trends of the graph, greater amounts of smoothing correspond with worse performance. We did not expect this. However, the graphs do at least show that the minimal amount of smoothing ($w = 2$) is still better than no smoothing at all ($w = 1$).

From this experiment, we learned the following:

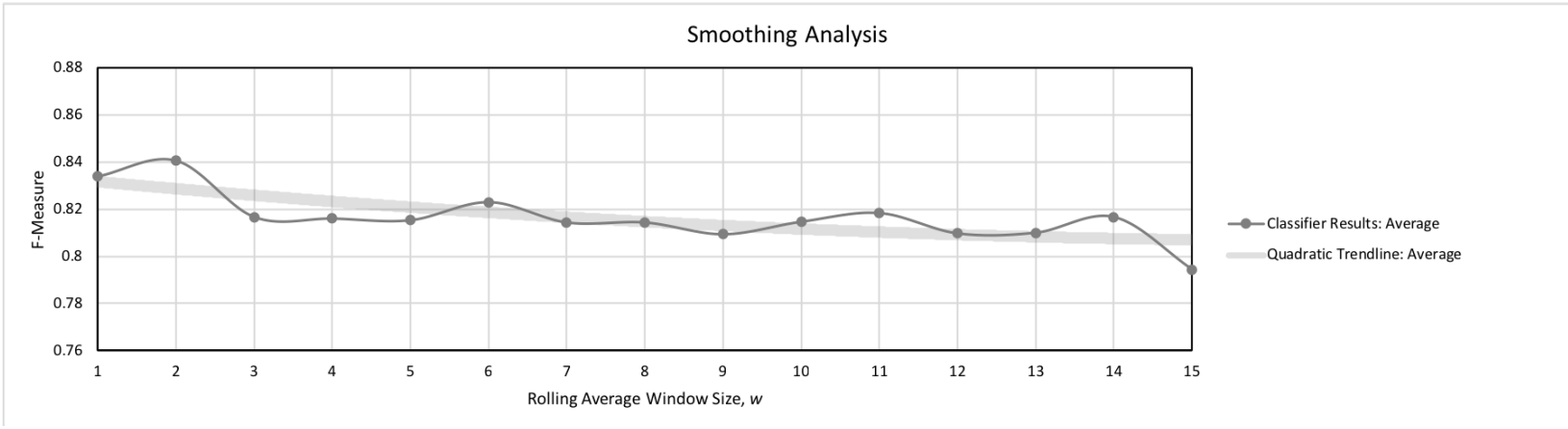
- While human analysis of a graph is often impaired by highly varied, noisy data, the machine learning analysis is not. That is, machine learning is capable of identifying trends and patterns where humans are not.

- The highly varied data likely contained useful information where where we initially assumed noise. In our effort to smooth the data and identify larger trends, we were unknowingly throwing out data that was useful to the machine learning models.

Perhaps these conclusions should not have been surprising. Indeed, machine learning models excel where humans fall short. We should trust them. Moving forward, our data should be smoothed using a rolling average window $w = 2$.



(a) The F-measures of IBk, Multilayer Perceptron, and Random Forest when trained on data with rolling average window sizes $w = 1 \ldots 15$.



(b) The average F-measure of IBk, Multilayer Perceptron, and Random Forest when trained on data with rolling average window sizes $w = 1 \ldots 15$.

Figure 5.1: The performance of classifiers when trained on data with varying amounts of smoothing.

5.3.2 Extracting Features

5.3.2.1 Feature Window

In section 4.3, we defined the feature window size to be 5 seconds, and the sliding window overlap to be 3 seconds. Both of these parameters were arbitrarily chosen, though the window size was based roughly on the average length of a seatbelt-buckling motion when viewing plots of the data. The implementation of these parameters clearly helped improve the performance of Phase I testing. However, we choose now to examine these parameters more closely.

To more accurately determine the optimal window configuration, we tested and compared classifiers trained on data with feature window sizes in 1-second increments ranging from 1 second to 7 seconds. For each of these sizes, we tested overlap amounts in 1-second increments ranging from 0 seconds to 1 second less than the window size (e.g. with a window size of 5 seconds, we tested overlaps of size 0–4 seconds). In total, this resulted in 28 different combinations of window size and window overlap. To best compare these 28 window configurations, we used the IBk and Random Forest classifiers. Consistently, the IBk, Multilayer Perceptron, and Random Forest classifiers yield the best results, so we often choose to focus on them specifically. In cases such as this where a many tests must be run, we often omit Multilayer Perceptron testing due to the amount of time required to train the models.

It would be excessive to report the full accuracies, F-measures, and confusion matrices for each of these tests, so we judge our comparison on the buckling F-measures specifically. The results of these 28 tests were compiled into the heatmaps shown in Figure 5.2. The results of the 28 IBk tests specifically are visible in Figure 5.2a, and the results of the 28 Random Forest tests specifically are visible in Figure 5.2b. These heatmaps are averaged to form the heatmap shown in Figure 5.2c, which we use for our analysis.

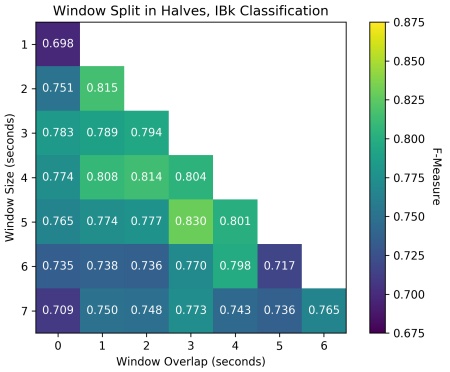
As a side note, these tests were performed before optimal smoothing was determined, so the data used for these tests was smoothing with a rolling average window $w = 10$. Furthermore, these tests were conducted using additional features which will not be introduced until later in this section.

From this heatmap, we can identify some trends. First, it appears that the classifiers perform worst at the extremes, or the corners of the triangle formed by the heatmap. Specifically, it is apparent that the values closest to the diagonal tend to perform the best. This makes sense, as the values on the diagonal represent the highest amount of window overlap. In general, more window overlap means there is less chance for a seatbelt action being "caught" between two windows. Put another way, the more windows there are, the more likely a seatbelt action will be centered in a window. If there is less overlap, there will be more variation in the location of the seatbelt action within the window. We also notice the trend that a larger window size generally leads to a higher F-measure. Of course, this also has its limit, as the 6- and 7-second window sizes displayed a drop-off in performance. This serves to confirm that our 5-second window size estimate was actually fairly well supported, as was our 3-second window overlap estimate. Too small of a window size means not enough of the buckling motion is present in the window. Too large of a window size means the window contains more than the buckling action. This heatmap informs us that the best choice for a window is a width of 5 seconds with an overlap of 3 seconds.

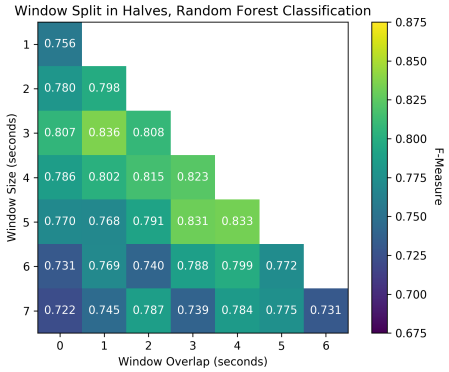
5.3.2.2 Window Splitting

Earlier in our research, we made the decision to "split" the window in half, based on the observation that a seatbelt buckling motion had two distinct motions: arm raising and arm lowering (see section 4.3 and Figure 4.3). Each of our figures was calculated on the window as normal, but then calculated again for the first half and then for the second half.

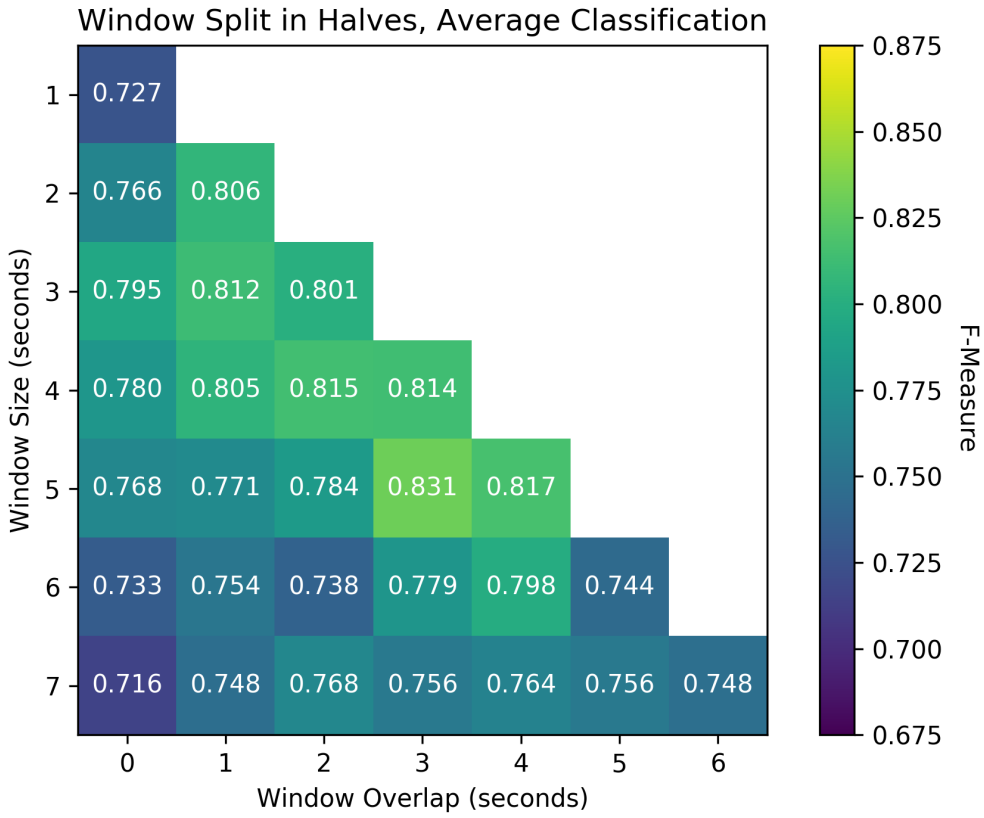
After further data observation, we began to consider the possibility of dividing the



(a) The results of IBk classifiers.



(b) The results of Random Forest classifiers.



(c) The average of IBk and Random Forest results.

Figure 5.2: This heatmap displays the buckling F-measures of many combinations of window size and window overlap. The number in each cell is the buckling F-measure, with the color of the cell corresponds with the cell's value. These tests were conducted using a feature set with the window split in halves.

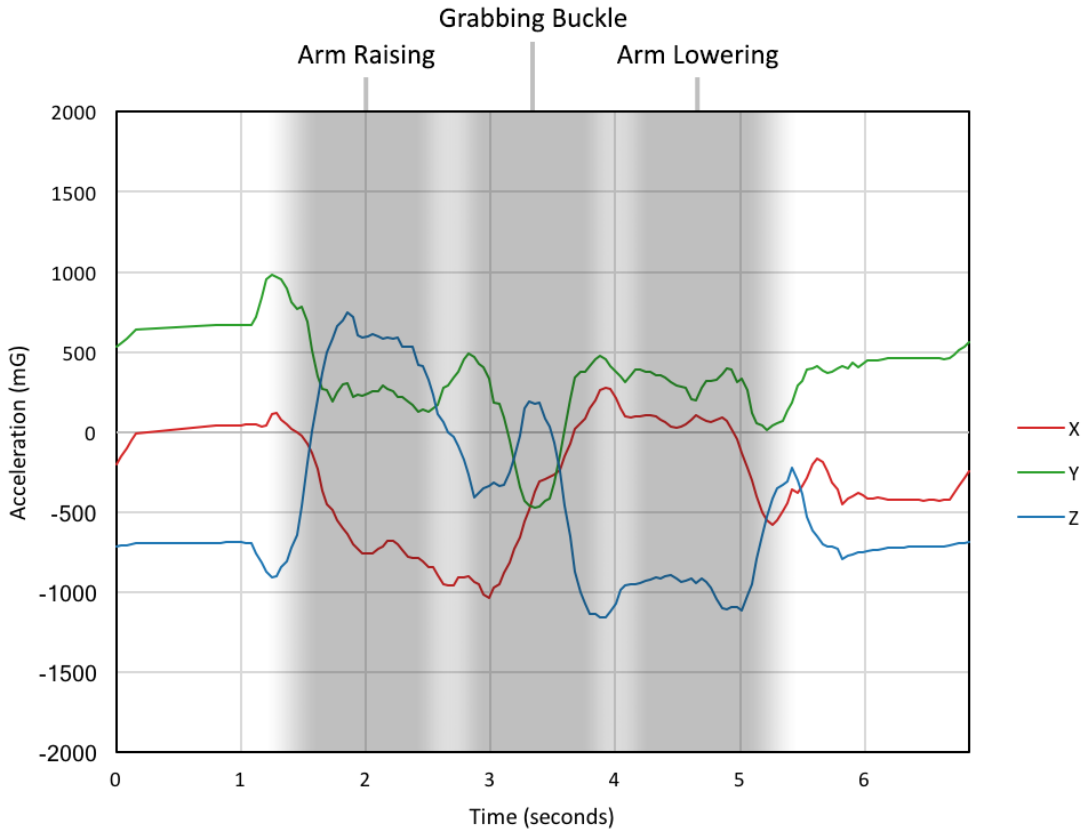


Figure 5.3: A plot of smoothed accelerometer data for one instance of buckling. The first "third" of the data corresponds to the period where the user lifts their arm to grab the buckle. The second "third" corresponds to the period where the user struggles to grab the buckle. The third "third" corresponds to the period where the user lowers their arm to fasten the buckle. The data shown in this graph was smoothed by rolling average with window size 10.

window into thirds instead. This is motivated by the period in the middle of the seatbelt motion wherein the user grabs the buckle. This period is marked by frantic movement, when compared with the periods of raising and lowering. See Figure 5.3 for a depiction of this.

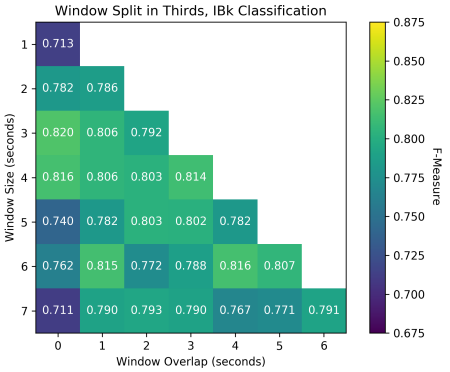
With this idea, we modified our feature extraction algorithms to split windows into thirds. Now, each feature was calculated *four* times: once over the first third, once over

the second third, once over the third third, and once over the entire window. We then re-ran the 28 tests from the prior section, generating another heatmap shown in Figure 5.4. Just as before, we used the IBk and Random Forest classifiers. The results of the IBk tests are visible in Figure 5.4a, the results of the Random Forest tests are visible in Figure 5.4b, and the average results are shown in Figure 5.2c. Again, these tests were performed using data smoothed with a rolling average window $w = 10$ and with the additional features which will be introduced soon.

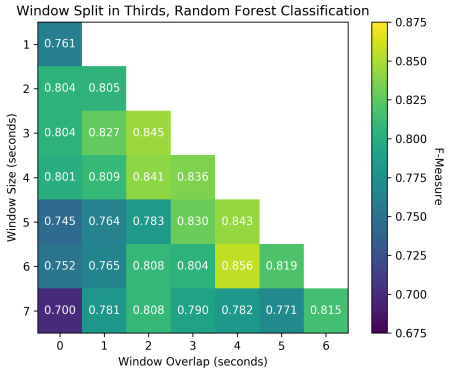
We now compare the "halves" heatmap (see Figure 5.2) and the new "thirds" heatmap (see Figure 5.4). This new heatmap displays trends similar to the previous heatmap. Namely, the classifiers perform worst at the extremes/corners, and the tests near the diagonal perform better. Small window sizes perform poorly, as do overly large window sizes. One interesting difference with the new heatmap is that the upper performance limit on window sizes expanded from 5 seconds to 6 seconds. That is, where the performance previously began to degrade above 5 seconds, the performance now begins to degrade above 6 seconds. This makes sense. As we increased the number of window "portions" from two to three, the optimal window size also increased slightly. According to this heatmap, the best window configuration for when the window is split into thirds is a window size of 6 seconds with an overlap of 4 seconds.

It is interesting to note that while configurations closer to the diagonal tend to perform better, the best results are often one away from the diagonal. Put another way, the optimal window overlap is 2 seconds less than the optimal window size, rather than 1 second less. This is seen in both cases: Figure 5.2 and Figure 5.4. We hypothesize two factors which may contribute to this:

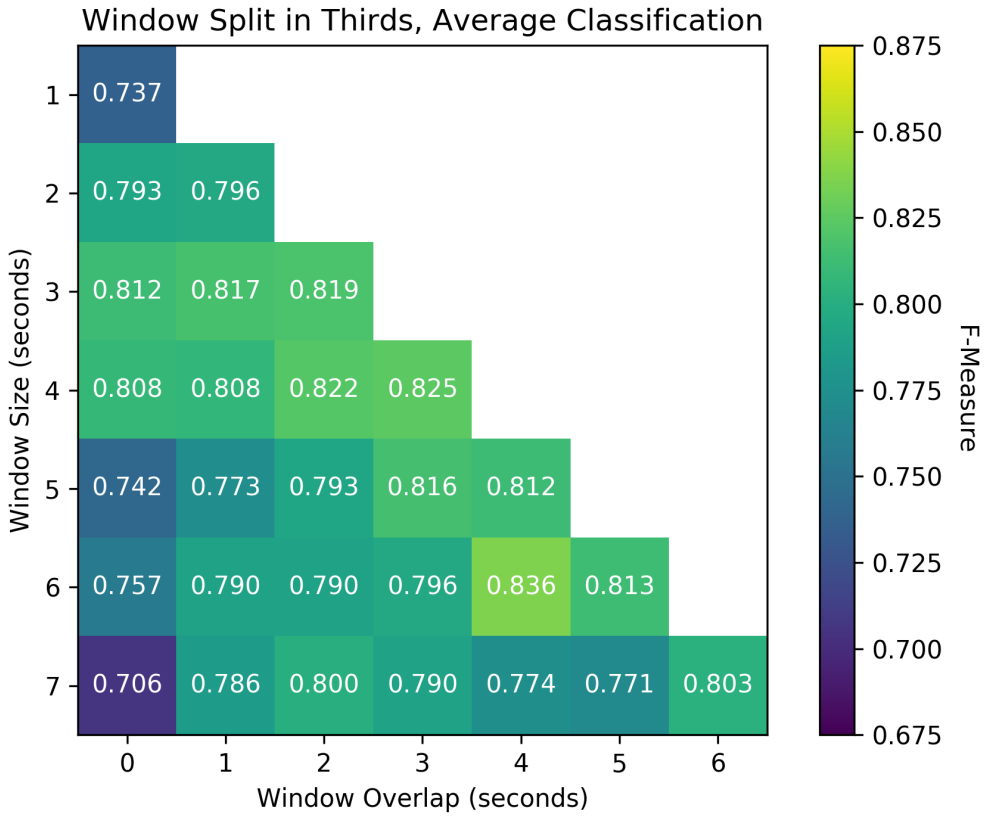
- As the window gets larger and larger, an overlap of 1 second less than the window size represents a greater and greater portion of the window being overlapped. For



(a) The results of IBk classifiers.



(b) The results of Random Forest classifiers.



(c) The average of IBk and Random Forest results.

Figure 5.4: This heatmap displays the buckling F-measures of many combinations of window size and window overlap. The number in each cell is the buckling F-measure, with the color of the cell corresponds with the cell's value. These tests were conducted using a feature set with the window split in thirds.

instance, in the case of a 5-second window with 4-second overlap, 80% of any given window will be overlapped with the following window. In the case of a 6-second window with 5-second overlap, 83% of the window is overlapped. As the percentage of the window overlapped increases, it will become more difficult to distinguish neighboring windows, as they will be mostly overlapped. This could lead to two neighboring windows both being classified as buckling.

- As the window size increases, the window is more likely to contain all of the seatbelt motion in its entirety, plus some of the ambient data surrounding the buckling motion. If the window size is larger than the size of the buckling motion, then the buckling motion could be entirely contained in two neighboring windows. This could also lead to two neighboring windows both being classified as buckling.

All this considered, the most important thing to note is that this entire heatmap, in general, displays better F-measures than the heatmap from the previous section. This suggests that a window split into thirds serves us better than a window split in half.

5.3.2.3 Feature Set

Using information gathered throughout all of our testing processes, we reconsidered our feature set. While we were still satisfied with our features, we identified the opportunity for a few more. As we are now recognizing the buckle-grabbing part of the seatbelt motion, we could make use of knowing how "calm" or "chaotic" the movement was. For this, we look at variance and standard deviation. Also, we wished to re-address our measure of correlation. Up to this point, the product of axes was used as a rough approximation of correlation. We now implement a more appropriate metric for correlation.

Here, we implemented 12 additional features (see Equation 5.1–Equation 5.12). As previously discussed, each of these features is now calculated four times each: once over

the first third of the window, once over the second third of the window, once over the third third of the window, and once over the whole window.

σ_X^2, σ_Y^2, σ_Z^2: This is the variance of each axis.

$$\sigma_X^2 = \frac{1}{n}\sum_{i=1}^{N}(x_i - \mu_X)^2 \tag{5.1}$$

$$\sigma_Y^2 = \frac{1}{n}\sum_{i=1}^{N}(y_i - \mu_Y)^2 \tag{5.2}$$

$$\sigma_Z^2 = \frac{1}{n}\sum_{i=1}^{N}(z_i - \mu_Z)^2 \tag{5.3}$$

σ_X, σ_Y, σ_Z: This is the standard deviation of each axis.

$$\sigma_X = \sqrt{\sigma_X^2} \tag{5.4}$$

$$\sigma_Y = \sqrt{\sigma_Y^2} \tag{5.5}$$

$$\sigma_Z = \sqrt{\sigma_Z^2} \tag{5.6}$$

σ_{XY}, σ_{XZ}, σ_{YZ}: This is the covariance between axes. Covariance models the strength of correlation between two axes, and is used in the calculation of statistical correlation.

$$\sigma_{XY} = \frac{1}{n}\sum_{i=1}^{N}(x_i - \mu_X)(y_i - \mu_Y) \tag{5.7}$$

$$\sigma_{XZ} = \frac{1}{n}\sum_{i=1}^{N}(x_i - \mu_X)(z_i - \mu_Z) \tag{5.8}$$

$$\sigma_{YZ} = \frac{1}{n}\sum_{i=1}^{N}(y_i - \mu_Y)(z_i - \mu_Z) \tag{5.9}$$

ρ_{XY}, ρ_{XZ}, ρ_{YZ}: This is the statistical correlation between axes. Specifically, this is modeled by Pearson's correlation coefficient. This is a true model of correlation, and is expected to perform much better than multiplying axes, as was introduced in section 4.2.

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} \tag{5.10}$$

$$\rho_{XZ} = \frac{\sigma_{XZ}}{\sigma_X \sigma_Z} \tag{5.11}$$

$$\rho_{YZ} = \frac{\sigma_{YZ}}{\sigma_Y \sigma_Z} \tag{5.12}$$

5.3.3 Evaluation

Based on our analysis of smoothing and feature window configuration, our best bet at recognition comes with a rolling average window size $w = 2$, a feature window size of 6 seconds, and a feature window overlap of 4 seconds. With this new information, plus our new features, we run all seven classifiers again, and report the results.

As seen in Table 5.3 all accuracy measurements improved, with the exception of IBk and Multilayer Perceptron, which both remained roughly the same (within 0.3%). Similarly, as seen in Table 5.4, all F-measures improved, again with the exception of IBk, which remained steady. The confusion matrices are displayed in Table B.8–Table B.14.

Something noteworthy in this round of testing is that the Random Forest classifier surpassed the Multilayer Perceptron classifier in terms of performance. In all tests leading up this point, with the exception of the very first, MultilayerPerceptron has performed the

Classifier	Accuracy
IBk	90.9%
J48	89.1%
Multilayer Perceptron	92.5%
Naive Bayes	83.8%
Random Forest	**94.0%**
SMO	91.3%
ZeroR	80.0%

Table 5.3: Classifier accuracies for Phase II advanced testing.

best.

At this point, we seem to be encountering the law of diminishing returns. While there was undoubtedly some improvement, it was not very substantial as compared to previous tests. This is a common phenomenon in machine learning, and indicates that we are likely overloading our models with more features than our data set can make use of. This is known as over-fitting, and will be explored later in this chapter.

5.4 Leave-One-Out Testing

For machine learning applications such as this, wherein the recognition algorithm is expected to perform well for users which are brand new to the application, it is important to conduct "leave-one-out" testing. Also called "leave-one-out cross validation", leave-one-out testing works by leaving one user out of the training set, and then using that user's data as the testing set. This simulates a situation where the model is already trained and a brand new user tries to use it, giving us a better idea of how our model performs on new users. This process is repeated for every user in our data set. So, in our case, this is repeated 20 times. The results are then aggregated to yield an overall metric of our model's performance on new users. The accuracy values are aggregated via averaging, but the F-measures are not. Instead of taking the average of all 20 F-measures, we sum all of

Classifier	Activity	F-measure
IBk	Buckling	0.791
	Not Buckling	0.942
J48	Buckling	0.740
	Not Buckling	0.931
Multilayer Perceptron	Buckling	0.816
	Not Buckling	0.953
Naive Bayes	Buckling	0.684
	Not Buckling	0.891
Random Forest	**Buckling**	**0.844**
	Not Buckling	0.963
SMO	Buckling	0.779
	Not Buckling	0.946
ZeroR	Buckling	0.000
	Not Buckling	0.889

Table 5.4: Classifier F-measures for Phase II advanced testing.

the confusion matrices together, then calculate the F-measure of that confusion matrix.

5.4.1 Evaluation

As leave-one-out testing involves a significant amount of data handling overhead (20 times as many tests must be conducted), in the interest of time we choose to focus on IBk and Random Forest, both for their prior performance record and for their training speed (Multilayer Perceptron can be very slow in comparison).

The accuracies shown in Table 5.5 are not awful, but are notably lower than all previous tests. Similarly, the F-measures shown in Table 5.6 are much worse than all previous tests. As a reminder, these metrics are aggregate values of all 20 leave-one-out tests. Each of the confusion matrices shown in Table B.15 and Table B.16 comprises the sum of all 20 individual confusion matrices.

Though we hoped for better results, we were not surprised by the lower performance values we encountered during leave-one-out testing. We identified two factors that likely

Classifier	Accuracy
IBk	77.5%
Random Forest	87.9%

Table 5.5: Classifier accuracies for Phase II leave-one-out testing.

Classifier	Activity	F-measure
IBk	Buckling	0.619
	Not Buckling	0.897
Random Forest	Buckling	0.683
	Not Buckling	0.932

Table 5.6: Classifier F-measures for Phase II leave-one-out testing.

contribute to this:

- As our data set is still relatively small, it is not fully representative of the general population. That is, our training will be slightly biased towards the distribution of our data set specifically.

- Unlike classic activity recognition tasks which involve a fairly standard, repetitive motion like running, walking, or climbing stairs, seatbelt-buckling is a single, distinct motion that varies from user to user. This makes the activity difficult to generalize. While we can train data on the motion patterns of prior users, a new user is likely to have a slightly different style of buckling.

Because of these reasons, a previously unseen user will not be recognized as well by the model. We discuss ways to address this later in this chapter, as well as in chapter 6.

5.5 Dimensionality Testing

Readers that have been paying close attention to our feature set will notice that it has grown to be quite large. We now have 32 unique features:

- Averages: μ_X, μ_Y, μ_Z
- Minimums: x_{min}, y_{min}, z_{min}
- Maximums: x_{max}, y_{max}, z_{max}
- Products: $X \times Y$, $X \times Z$, $Y \times Z$
- Differences: $X - Y$, $X - Z$, $Y - Z$
- Absolute Differences: $|X - Y|$, $|X - Z|$, $|Y - Z|$
- Difference in XY and YZ Absolute Differences: $|X - Y| - |Y - Z|$
- Absolute Difference in XY and YZ Absolute Differences: $||X - Y| - |Y - Z||$
- Variances: σ_X^2, σ_Y^2, σ_Z^2
- Standard Deviations: σ_X, σ_Y, σ_Z
- Covariances: σ_{XY}, σ_{XZ}, σ_{YZ}
- Correlation Coefficients: ρ_{XY}, ρ_{XZ}, ρ_{YZ}

However, as we are also calculating these features again over each third of the window independently, we are effectively quadrupling our feature count to 128. Realistically, this feature count is likely too large for our data set. This can cause problems when training our models.

One of these issues is known as the "curse of dimensionality", which asserts that as the number of dimensions increases, a larger and larger percentage of the data set is necessary for training in order to cover the same percentage of the feature space. In other words, as the number of features increases, it becomes more and more difficult for the classifier to fully and accurately model the feature space. Furthermore, with a large number of features,

there is greater risk for some features to be correlated with each other. High correlation between features manifests itself as noise when training.

Another common issue that plagues machine learning models (especially deep learning models) is that of *overfitting*. A model learns based on its provided training data. However, the goal is not to learn exactly what that specific set of data looks like; the goal is to *generalize*. Generalization refers to learning the underlying patterns present in all data—in our case, learning how to recognize a general seatbelt motion, regardless of who is doing it. This is a difficult balance to maintain. The more complex our models become, the more likely it is to overfit the data, i.e., start learning the small artifacts of our data set specifically. When overfitting starts to occur, it means that the model will start performing worse on our test data data.

There are three primary ways to combat overfitting:

- Increase the amount of training data.
- Decrease the number of features (dimensionality).
- Use regularization.

Regularization applies to regression models, not classification models, so we ignore it. With our limited data set, we should pursue a reduction in our feature set.

5.5.1 Attribute Reduction

Fortunately, WEKA provides convenient tools for attribute selection. We choose to use the Correlation-based Feature Selection (CFS) subset evaluation tool [70]. This tool evaluates each feature in our set on the basis of their ability to predict the class as well as on the basis of redundancy with other features. This evaluation technique will generate a small subset of our larger feature set, wherein the features will have a high correlation

with the class but low intercorrelation [70]. Of our 128 features, CFS Subset Evaluation selected 28. These features are shown in Table 5.7.

5.5.2 Evaluation

After using CFS Subset Evaluation to reduce our model's dimensionality, we re-evaluated the overall evaluation originally conducted in section 5.3 as well as the leave-one-out evaluation originally conducted in section 5.4.

Based on the accuracy results of our overall testing shown in Table 5.8, most classifiers dropped in accuracy by roughly 1–3%. The only exception to this was the Naive Bayes classifier, which actually improved by 3%.

The F-measures shown in Table 5.9 maintain this trend. The F-measures of most classifiers dropped by 0.030–0.090, again with the exception of Naive Bayes, which improved by 0.040.

The confusion matrices in Table B.17–Table B.23 show nothing particularly noteworthy. In general, true classifications drop slightly and false classifications rise slightly.

These results were not surprising. The performance metrics of the models were expected to drop. When we remove features, we are withholding data from our model, meaning that it will not have as much information to train on. Since the performance dropped, it is possible that our model was overfitting. We look to the leave-one-out test results for more information.

According to the accuracy results of our leave-one-out testing shown in Table 5.10, IBk increased in accuracy by about 5%, while Random forest decreased in accuracy by 1%. Meanwhile, according to Table 5.11, the F-measure of IBk stayed exactly the same, and the F-measure of Random Forest decreased by about 0.020.

The confusion matrices in Table B.24 and Table B.25 reveal an interesting trend. Overall, the models were now slightly more likely to classify results as positive. That is, in both

Feature	Full Window	First Third	Second Third	Third Third
μ_X				
μ_Y				
μ_Z				
x_{min}			X	
y_{min}			X	
z_{min}	X		X	
x_{max}			X	
y_{max}			X	
z_{max}				
$X \times Y$				
$X \times Z$				
$Y \times Z$				X
$X - Y$			X	
$X - Z$				
$Y - Z$				
$\|X - Y\|$	X			
$\|X - Z\|$				
$\|Y - Z\|$	X		X	
$\|X - Y\| - \|Y - Z\|$				
$\|\|X - Y\| - \|Y - Z\|\|$				
σ_X^2	X	X	X	
σ_Y^2	X		X	
σ_Z^2	X		X	
σ_X			X	
σ_Y				
σ_Z	X			
σ_{XY}				X
σ_{XZ}				
σ_{YZ}				
ρ_{XY}	X	X		
ρ_{XZ}	X			
ρ_{YZ}	X		X	

Table 5.7: This table contains the full feature set. Features selected by CFS Subset Evaluation are indicated with an X.

Classifier	Accuracy
IBk	89.3%
J48	87.1%
Multilayer Perceptron	91.5%
Naive Bayes	86.6%
Random Forest	**92.0%**
SMO	88.3%
ZeroR	80.0%

Table 5.8: Classifier accuracies for Phase II advanced testing with CFS feature reduction.

Classifier	Activity	F-measure
IBk	Buckling	0.755
	Not Buckling	0.931
J48	Buckling	0.673
	Not Buckling	0.920
Multilayer Perceptron	Buckling	0.789
	Not Buckling	0.947
Naive Bayes	Buckling	0.725
	Not Buckling	0.911
Random Forest	**Buckling**	**0.795**
	Not Buckling	0.950
SMO	Buckling	0.681
	Not Buckling	0.928
ZeroR	Buckling	0.000
	Not Buckling	0.889

Table 5.9: Classifier F-measures for Phase II advanced testing with CFS feature reduction.

Classifier	Accuracy
IBk	82.3%
Random Forest	87.0%

Table 5.10: Classifier accuracies for Phase II leave-one-out testing with CFS feature reduction.

Classifier	Activity	F-measure
IBk	Buckling	0.619
	Not Buckling	0.886
Random Forest	Buckling	0.662
	Not Buckling	0.919

Table 5.11: Classifier F-measures for Phase II leave-one-out testing with CFS feature reduction.

cases, the number of true positives and false positives both increased, while the number of true negatives and false negatives both decreased. While more true positives and less false negatives are a good thing, more false positives and less true negatives are a bad thing. Overall, these factors cancel each other out when calculating metrics like precision, recall, and f-measure, so our results remain nearly unchanged.

So, when testing over the entire data set using stratified cross validation, trimming our feature set down from 128 to 26 results in somewhat of a drop in performance. However, when conducting leave-one-out testing, shrinking the size of our feature set made almost no difference. Based on these observations, we conclude that while our larger set of features may not have explicitly detracted from our model's real-world performance, it has the potential to cause our testing results to appear a little bit optimistic.

5.6 Verification Testing

We performed a significant amount of experimentation to develop our models. We tested different smoothing methods, feature window configurations, feature sets, machine learning models, etc. This much experimentation could be considered high-level hyperparameter tuning. Because we used the test data set as our gauge when tuning our models, we effectively used information from the test data set to develop our models. Of course, when decisions are made based on information from the test data, we should be wary of inflated results.

Throughout this work, we tested using a form of stratified cross validation rather than a static train-test split (see section 3.2). This mitigates the risk of overfitting, so we were not as worried about inaccurate results. That said, we still acknowledge this concern. For this reason, we conducted a brief, supplemental user study and collected six additional samples. As these new samples were completely unseen throughout our research, we used them to verify the validity of our prior test results.

5.6.1 Evaluation

The accuracy results are shown in Table 5.12, the F-measure results are shown in Table 5.13, and the confusion matrices are shown in Table B.26–Table B.32. These results are in line with those of our advanced testing with feature reduction (see section 5.5). This confirms much of what we have already claimed:

- Using the entire feature set leads to overfitting, yielding overly optimistic results.
- Reducing the feature set resulted in a more accurate portrayal of model performance.
- Though we tuned using test data information, cross validation prevented overfitting.
- Our models perform as expected on unseen data.

Classifier	Activity
IBk	85.1%
J48	84.7%
Multilayer Perceptron	88.8%
Naive Bayes	71.9%
Random Forest	87.5%
SMO	**90.5%**
ZeroR	80.0%

Table 5.12: Classifier accuracies for Phase II verification testing.

Classifier	Activity	F-measure
IBk	Buckling	0.707
	Not Buckling	0.900
J48	Buckling	0.690
	Not Buckling	0.899
Multilayer Perceptron	Buckling	0.756
	Not Buckling	0.927
Naive Bayes	Buckling	0.583
	Not Buckling	0.788
Random Forest	Buckling	0.722
	Not Buckling	0.919
SMO	**Buckling**	**0.781**
	Not Buckling	0.939
ZeroR	Buckling	0.000
	Not Buckling	0.889

Table 5.13: Classifier F-measures for Phase II verification testing.

6. FUTURE WORK

6.1 Fully Naturalistic Studies

We acknowledge that our research studies were not fully representative of "real-life" usage patterns. Clearly, the Phase I study was not naturalistic. That data was collected in the form of discrete samples in a controlled environment. This study was no less important, however, for that data served a purpose in guiding us in the right direction at the onset of our research. Of course, before long, it was clear that a more naturalistic study needed to be conducted to move forward. As such, the Phase II study was conducted. This study was much more representative of real-time usage, as the data was collected non-stop from the beginning to the end of each user's data collection period.

Realistically, this Phase II data was only semi-naturalistic, not fully naturalistic. By this, we mean that even though the data was well representative of what standard real-time data would look like, the studies were still collected within a controlled period. Therefore, the data distribution is not fully reflective of what an average user's day would look like. Specifically, the concentration of seatbelt buckling was higher in our semi-naturalistic data than it would be in fully naturalistic data. This is not to say that our research is invalid. We have provided a strong basis on which to further improve the recognition as more data is collected. It is to say, however, that future work should begin with conducting more user studies to collect not only more data, but also data that is more naturalistic. With more user data, the seatbelt buckling motion will become easier to generalize, and it will become more apparent which features are most beneficial toward classification.

6.2 Active Learning

Though our classifier performance was often not as great as we would have hoped, this research was by no means unsuccessful. This process revealed a lot of insights toward

unconventional recognition tasks, like that of seatbelt-buckling classification. Throughout our analysis of results, we began to notice that seatbelt activity was in many cases more difficult to recognize than traditional ambulation activities. the following reasons help to explain why seatbelt buckling is a unique problem:

- Most classic recognition tasks involve the classification of sustained or repeated activity. Recall Table 2.1. Running, ascending stairs, riding a bike, lifting weights, typing at a computer, etc. are all activities which involve constant, repetitive motion. For activities such as this, it is common to use a technique known as two-tier recognition [48]. With this technique, a recognition algorithm divides the accelerometer data into relatively small windows, classifying each one individually. However, the overall activity determination is not made on these small windows themselves, but on the "big picture". If there are enough of these positive windows in succession, then the algorithm becomes confident in the activity and makes its determination. In our case, we cannot benefit from this technique, as the activity which we are trying to recognize is quick and discrete. There is no sustained pattern of repetitive motion which we can draw information from—we either correctly recognize the buckling motion as it occurs, or we miss it. We have one chance. These stakes are much higher than with traditional recognition applications, and our models must therefore perform much more accurately in order to get similar results.

- We also found seatbelt buckling motions to be highly personalized. That is, there was a high level variance in the patterns of motion which users exhibited while putting on their seatbelts. It might be assumed that this is because some users used their left hand and others used their right; however, we found this not to be the reason. If we give our model the information as to which hand the user used, the models performed no better. It is therefore assumed that buckling motions vary

widely from person-to-person. While we did our best to recognize the general trends in the buckling motion (reaching up, grabbing, bringing down), new users were often fairly difficult to classify well.

The second point above suggests an alternative approach to this classification problem. In the case of highly personalized motions such as this, attempting to generalize all of the population's patterns into one universal seatbelt motion may fail to attain the level of effectiveness desired for a practical, real-world applications. In this case, it may be beneficial to pursue *active learning*, also called adaptive recognition [40].

In active learning, we avoid making too many assumptions on the user ahead of time. Instead, we learn directly from the target user. That is, we have a very basic model that does an acceptable job of recognizing seatbelt buckling, but we build on this model as more and more data is collected from the user. In this way, the model learns the patterns of that specific user. This approach can be especially effective in applications like some of the ones we have described. For example, in the case of a personally-owned fitness-tracking smart watch, a user is most often tracking their personal health and performance. Here, the models can be trained specifically to that user, as they are the only ones using it. Prior work has demonstrated the effectiveness of this type of personalized approach to recognition, often called "individual clasification" [37]. Models can easily adapt to recognize the specific habits and patterns of the target user specifically.

In an attempt to briefly explore the feasibility of an active learning classifier, we looked at the data for each user individually. For each user, we used the first 75% of their data as training data, and then attempted to recognize the remaining 25%. We then aggregated this data across all 20 users, just as we did during the leave-one-out testing in the previous chapter.

The results are encouraging. IBk accuracy (see Table 6.1) improved by 9.1% compared

Classifier	Accuracy
IBk	91.4%
Random Forest	89.5%

Table 6.1: Classifier accuracies for active recognition exploratory testing.

Classifier	Activity	F-measure
IBk	Buckling	0.854
	Not Buckling	0.951
Random Forest	Buckling	0.881
	Not Buckling	0.964

Table 6.2: Classifier F-measures for active recognition exploratory testing.

to leave-one-out testing, and Random Forest improved by 2.5%. While this is nice, the F-measures are more impressive (see Table 6.2). IBk's F-measure increased by 0.235, and Random Forest's F-measure increased by 0.219. Confusion matrices are shown in Table 6.3 and Table 6.4.

Granted, these tests are conducted on data sets too small for these results to be completely conclusive. However, these brief tests are enough to make a case for active learning. After just a few examples of a particular user's buckling style, these models can already recognize that user's motion patterns better than if the models had been trained on a larger amount of data from other users. Future work should investigate active learning as a viable possibility.

Buckling	Not Buckling	← classified as
73	13	Buckling
12	275	Not Buckling

Table 6.3: IBk confusion matrix for active recognition exploratory testing.

Buckling	Not Buckling	← classified as
70	16	Buckling
3	254	Not Buckling

Table 6.4: Random Forest confusion matrix for active recognition exploratory testing.

6.3 Additional Sensors

Remember our original motivation for this work: to detect whether someone is driving without wearing their seatbelt. Knowing this use case, there are many possibilities for additional data which may be useful in recognizing this situation.

We chose to focus specifically on the seatbelt motion, as it is the most important piece of this recognition. However, other data could prove useful in validating our seatbelt recognition. For example, if sitting and standing are already recognizable by activity recognition techniques, we could incorporate this into our overall algorithm to identify whether a user appears to have just sat down into a vehicle, after which a seatbelt motion is likely to follow.

Furthermore, if the application intends to detect when a user is driving without a seatbelt, it is also important to know if the user is actually driving. Something like this could easily be accomplished using the additional sensors on the phone or wearable device [71]. For example, a GPS sensor could indicate when the user is moving faster than a predefined speed threshold.

There are many possibilities for additional recognition and sensor data that could easily be incorporated into the application based on the data and sensors already available in a Pebble watch and Android phone. This is just to say that our overall algorithm need not be limited to the seatbelt recognition component alone.

6.4 Feedback and Intervention

Finally, if the overall goal is to prevent a user from driving a vehicle without wearing their seatbelt, then future work must pursue techniques of feedback and intervention. That is, once our algorithm recognizes that a user has failed to fasten their seatbelt, what is our application to do with this information?

With regard to feedback, the application must find a way to notify the user. In our case, this notification should be clear. Classic examples of warnings include auditory signals or visual cues [14]. With the rise of smart phone and smart watch adoption, haptic[1] feedback in the form of vibrations has also increased in usage [73–76].

Of course, the end goal is not just notification, but intervention. That is, we must intervene in the user's habits with the hope of altering behavior. This presents additional challenges, as our techniques must not be so obnoxious that the user ops-out of the application all together. To this end, haptic feedback is appealing. In the end, if we wish to promote positive behavior modification, it must be a system which is not only helpful but encourages the user to participate.

Future work should investigate and compare different methods of feedback and intervention. When combined with effective feedback and intervention techniques, the algorithm proposed by this work can be used in applications which monitor and promote user safety.

[1] Haptic: relating to or based on the sense of touch [72]

7. CONCLUSION

This work was primarily focused on creating a method to recognize the motion of putting on a seatbelt using wearable technology such as a smart watch. Motivated by the fatality rates in accidents where drivers neglected to wear their seatbelts, we considered alternate forms of behavioral intervention and modification to encourage seatbelt use.

The use of wearable devices has grown increasingly popular in the rise of fitness tracking and other health-related applications [12, 14]. These approaches rely on activity recognition to correctly identify the user's motion data as specific activities. However, we found no prior work toward an activity recognition algorithm for seatbelt fastening. As any sort of real-time tracking and feedback application would require such an algorithm, we set out to create one.

Using a pebble smart watch and an Android smart phone, we conducted two studies. In our first study, we collected some discrete motion data of users buckling their seatbelts as well as performing a specific set of control motions. From this initial study, we confirmed that seatbelt-buckling was, in the very least, feasibly distinguishable from other actions involving similar patterns of motion. Encouraged by these results, we conducted a second study wherein we collected data which more closely resembled a real-world use case. Throughout the both of these studies, we identified the data processing techniques, novel features, and classification algorithms which were most appropriate to use in seatbelt-identification applications.

We primarily used F-measure to judge the performance of our models. In the end, our best performance achieved was with the Random Forest classifier at an F-measure of 0.844. In leave-one-out testing, this F-measure was 0.683. However, after reducing our feature set using Correlation-based Feature Selection, these F-measures dropped to 0.795

and 0.662, respectively.

The disparity in performance between the overall test results and the leave-one-out test results suggest that seatbelt motions are very personal—they vary from user to user. This motivates future work in active learning, wherein models can adapt specifically to their target user. A brief exploratory analysis confirmed the potential for success in this approach.

In general, our studies found that seatbelt motions are indeed recognizable, though it can be difficult to do so due to the fleeting nature of the gesture and the high level of individuality among users. Regardless, there are many possibilities for future work in augmenting our recognition algorithm.

Indeed, we have provided a suitable foundation upon which future systems may be built. This was our intent. We hope that this work motivates additional study toward creating intelligent, context-aware systems for encouraging safe habits and general well-being.

REFERENCES

[1] Centers for Disease Control and Prevention (CDC), "Faststats - leading causes of death." https://www.cdc.gov/nchs/fastats/leading-causes-of-death.htm, 2016. Accessed: 2019-02-07.

[2] Centers for Disease Control and Prevention (CDC), "Wisqars (web-based injury statistics query and reporting system)." https://www.cdc.gov/injury/wisqars/index.html, 2017. Accessed: 2019-02-12.

[3] N. H. T. S. Administration, "Traffic safety facts, 2014 data: occupant protection. Washington, DC: US department of transportation, national highway traffic safety administration; 2016," 2016.

[4] Centers for Disease Control and Prevention (CDC), "Seat belts: Get the facts." http://www.cdc.gov/motorvehiclesafety/seatbelts/facts.html, July 5, 2016. Accessed: 2017-04-09.

[5] T. W. Strine, L. Beck, J. Bolen, C. Okoro, and C. Li, "Potential moderating role of seat belt law on the relationship between seat belt use and adverse health behavior," *American journal of health behavior*, vol. 36, no. 1, pp. 44–55, 2012.

[6] R. Yano, "Seat belt buckle engagement detector and seat belt system." https://www.google.com/patents/US6498562, 2000. US Patent 6,498,562.

[7] H. Yoshihiro, "Device in an automobile for detecting a pull-out motion of a seat belt." https://www.google.com/patents/US3689881, 1971. US Patent 3,689,881.

[8] K. Aoki, T. Yasuda, S. Masanori, K. Osamu, and A. Kaisha, "Apparatus for detecting the wearing of a seat belt assembly." https://www.google.com/patents/US4885566, 1987. US Patent 4,885,566.

[9] J. Lester, T. Choudhury, and G. Borriello, "A practical approach to recognizing physical activities," in *International Conference on Pervasive Computing*, pp. 1–16, Springer, 2006.

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

[11] E. Thomaz, I. Essa, and G. D. Abowd, "A practical approach for recognizing eating moments with wrist-mounted inertial sensing," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 1029–1040, ACM, 2015.

[12] J. Bartley, J. Forsyth, P. Pendse, D. Xin, G. Brown, P. Hagseth, A. Agrawal, D. W. Goldberg, and T. Hammond, "World of workout: A contextual mobile rpg to encourage long term fitness," in *Proceedings of the Second ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health*, HealthGIS '13, (New York, NY, USA), pp. 60–67, ACM, 2013.

[13] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 790–808, Nov 2012.

[14] V. Rajanna, F. Alamudun, D. Goldberg, and T. Hammond, "Let me relax: Toward automated sedentary state recognition and ubiquitous mental wellness solutions," in *Proceedings of the 5th EAI International Conference on Wireless Mobile Communication and Healthcare*, pp. 28–33, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2015.

[15] D. Figo, P. C. Diniz, D. R. Ferreira, and J. M. Cardoso, "Preprocessing techniques for context recognition from accelerometer data," *Personal and Ubiquitous Computing*, vol. 14, no. 7, pp. 645–662, 2010.

[16] R. J. McMillan, A. D. Craig, and J. P. Heinen, "Motor vehicle monitoring system for determining a cost of insurance." https://patents.google.com/patent/US6064970, May 16 2000. US Patent 6,064,970.

[17] M. Henderson, R. J. McMillan, A. D. Craig, J. P. Heinen, B. J. Olexa, M. C. McElroy, and R. S. Lee, "Monitoring system for determining and communicating a cost of insurance." https://patents.google.com/patent/US6868386, Mar. 15 2005. US Patent 6,868,386.

[18] R. S. Ling, R. A. Hutchinson, W. J. Steigerwald III, W. A. Say, P. L. O'malley, D. A. Shrallow, W. C. Everett, and R. J. McMillan, "Vehicle monitoring system." https://patents.google.com/patent/US9754424, Sept. 5 2017. US Patent 9,754,424.

[19] G. Warren and M. Greenlee, "Calculation of driver score based on vehicle operation for forward looking insurance premiums." https://patents.google.com/patent/US20070027726, Feb. 1 2007. US Patent App. 11/409,493.

[20] P. Händel, J. Ohlsson, M. Ohlsson, I. Skog, and E. Nygren, "Smartphone-based measurement systems for road vehicle traffic monitoring and usage-based insurance," *IEEE Systems Journal*, vol. 8, pp. 1238–1248, Dec 2014.

[21] L. Giddens, D. Leidner, and E. Gonzalez, "The role of fitbits in corporate wellness programs: Does step count matter?," in *Proceedings of the 50th Hawaii International Conference on System Sciences.*, pp. 3627–3635, HICSS, 2017.

[22] Merriam-Webster, "Photic." https://www.merriam-webster.com/dictionary/photic.

[23] J. M. Peschel, B. Paulson, and T. Hammond, "A surfaceless pen-based interface," in *Proceedings of the Seventh ACM Conference on Creativity and Cognition*, C&C '09, (New York, NY, USA), pp. 433–434, ACM, 2009.

[24] I. J. Amin, A. J. Taylor, and R. M. Parkin, "Driver tracking and posture detection

using low-resolution infrared sensing," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 221, no. 9, pp. 1079–1088, 2007.

[25] P. Kaul, V. Rajanna, and T. Hammond, "Exploring users' perceived activities in a sketch-based intelligent tutoring system through eye movement data," in *ACM Symposium on Applied Perception (SAP '16)*, SAP, p. 1, 2016.

[26] F. Alamudun, H.-J. Yoon, T. Hammond, K. Hudson, G. Morin-Ducote, and G. Tourassi, "Shapelet analysis of pupil dilation for modeling visuo-cognitive behavior in screening mammography," in *Proc. SPIE*, vol. 9787 of *SPIE*, pp. 97870M–97870M–13, 2016.

[27] F. Alamudun, H.-J. Yoon, K. B. Hudson, G. Morin-Ducote, T. Hammond, and G. D. Tourassi, "Fractal analysis of visual search activity for mass detection during mammographic screening," *Medical Physics*, 2017.

[28] P. Taele and T. Hammond, "Invisishapes: A recognition system for sketched 3d primitives in continuous interaction spaces," in *Proceedings of the 2015 International Symposium on Smart Graphics, Chengdu, China*, SG, p. 12, 2015.

[29] Merriam-Webster, "Kinetic." `https://www.merriam-webster.com/dictionary/kinetic`.

[30] J. Miller and T. Hammond, "Wiiolin: A virtual instrument using the wii remote," in *Proceedings of the 2010 Conference on New Interfaces for Musical Expression (NIME)*, (Sydney, Australia), p. 497500, June 15-18 2010.

[31] J. Pärkkä, M. Ermes, P. Korpipää, J. Mäntyjärvi, J. Peltola, and I. Korhonen, "Activity classification using realistic data from wearable sensors," *IEEE Transactions on information technology in biomedicine*, vol. 10, no. 1, pp. 119–128, 2006.

[32] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," in *International Conference on Pervasive Computing*, pp. 1–17, Springer, 2004.

[33] F. Foerster, M. Smeja, and J. Fahrenberg, "Detection of posture and motion by accelerometry: a validation study in ambulatory monitoring," *Computers in Human Behavior*, vol. 15, no. 5, pp. 571–583, 1999.

[34] E. M. Tapia, S. S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman, "Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor," in *Wearable Computers, 2007 11th IEEE International Symposium on*, pp. 37–40, IEEE, 2007.

[35] U. Maurer, A. Smailagic, D. P. Siewiorek, and M. Deisher, "Activity recognition and monitoring using multiple sensors on different body positions," in *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pp. 4–pp, IEEE, 2006.

[36] N. Győrbíró, Á. Fábián, and G. Hományi, "An activity recognition system for mobile phones," *Mobile Networks and Applications*, vol. 14, no. 1, pp. 82–91, 2009.

[37] 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*, pp. 796–799, Springer, 2009.

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

[39] G. M. Weiss, J. L. Timko, C. M. Gallagher, K. Yoneda, and A. J. Schreiber, "Smartwatch-based activity recognition: A machine learning approach," in *Biomedical and Health Informatics (BHI), 2016 IEEE-EMBS International Conference on*, pp. 426–429, IEEE, 2016.

[40] K. Van Laerhoven and O. Cakmakci, *What shall we teach our pants?*, pp. 77–83. IEEE Press, 2000.

[41] B. Paulson and T. Hammond, "Office activity recognition using hand posture cues,"

in *Proceedings of the 22Nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 2*, BCS-HCI '08, (Swinton, UK, UK), pp. 75–78, British Computer Society, 2008.

[42] B. Paulson, D. Cummings, and T. Hammond, "Object interaction detection using hand posture cues in an office setting," *Int. J. Hum.-Comput. Stud.*, vol. 69, pp. 19–29, Jan. 2011.

[43] D. Brhlik, C. Young, and T. Otuyelu, "Enhancing blind navigation with the use of wearable sensor technology," undergraduate honors thesis, Texas A&M University, May 2016.

[44] C. Randell and H. Muller, "Context awareness by analysing accelerometer data," in *Digest of Papers. Fourth International Symposium on Wearable Computers*, pp. 175–176, Oct 2000.

[45] N. Ravi, N. Dandekar, P. Mysore, and M. L. Littman, "Activity recognition from accelerometer data," in *Aaai*, vol. 5, pp. 1541–1546, 2005.

[46] E. Garcia-Ceja, R. F. Brena, J. C. Carrasco-Jimenez, and L. Garrido, "Long-term activity recognition from wristwatch accelerometer data," *Sensors*, vol. 14, no. 12, pp. 22500–22524, 2014.

[47] V. Rajanna, R. Lara-Garduno, D. J. Behera, K. Madanagopal, D. Goldberg, and T. Hammond, "Step up life: a context aware health assistant," in *Proceedings of the Third ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health*, pp. 21–30, ACM, 2014.

[48] J. Cherian, V. Rajanna, D. Goldberg, and T. Hammond, "Did you remember to brush? : A noninvasive wearable approach to recognizing brushing teeth for elderly care.," in *11th EAI International Conference on Pervasive Computing Technologies for Healthcare.*, ACM, 2017.

[49] Pebble, "Pebble time steel." `https://www.pebble.com/`

pebble-time-steel-smartwatch-features. Accessed: 2017-04-09.

[50] Pebble, "Pebble accelerometer." https://developer.rebble.io/developer.pebble.com/guides/events-and-services/accelerometer/index.html. Accessed: 2019-02-05.

[51] Fitbit, "Fitbit accelerometer sensor guide." https://dev.fitbit.com/build/guides/sensors/accelerometer/. Accessed: 2019-02-05.

[52] E. Frank, M. Hall, and I. Witten, *The WEKA Workbench. Online appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. Morgan Kaufmann, 4th ed., 2016.

[53] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: An update," in *SIGKDD Explorations*, vol. 11, 2009.

[54] The University of Waikato, "Weka 3: Data mining software in java." https://www.cs.waikato.ac.nz/ml/weka/.

[55] S. Inglis, L. Trigg, and E. Frank, "weka.classifiers.lazy class IBk." http://weka.sourceforge.net/doc.dev/weka/classifiers/lazy/IBk.html. Accessed: 2017-04-09.

[56] D. Aha and D. Kibler, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.

[57] E. Frank, "weka.classifiers.trees class J48." http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html. Accessed: 2017-04-09.

[58] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[59] M. Ware, "weka.classifiers.functions class MultilayerPerceptron." http://weka.sourceforge.net/doc.dev/weka/classifiers/

functions/MultilayerPerceptron.html. Accessed: 2017-04-09.

[60] "weka.classifiers.functions class RandomForest." http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/RandomForest.html. Accessed: 2019-02-14.

[61] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[62] S. Sathyadevan and R. R. Nair, "Comparative analysis of decision tree algorithms: ID3, C4. 5 and random forest," in *Computational Intelligence in Data Mining*, vol. 1, pp. 549–562, 2015.

[63] Y. J. Sheela and S. H. Krishnaveni, "A comparative analysis of various classification trees," in *2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT)*, pp. 1–8, April 2017.

[64] E. Frank, S. Legg, and S. Inglis, "weka.classifiers.functions class SMO." http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html. Accessed: 2017-04-09.

[65] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods - Support Vector Learning* (B. Schoelkopf, C. Burges, and A. Smola, eds.), MIT Press, 1998.

[66] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy, "Improvements to platt's smo algorithm for svm classifier design," *Neural Computation*, vol. 13, no. 3, pp. 637–649, 2001.

[67] T. Hastie and R. Tibshirani, "Classification by pairwise coupling," in *Advances in Neural Information Processing Systems* (M. I. Jordan, M. J. Kearns, and S. A. Solla, eds.), vol. 10, MIT Press, 1998.

[68] E. Frank, "weka.classifiers.rules class ZeroR." http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/ZeroR.html.

[69] J. Leland and E. Stanfill, "Recognizing seatbelt-fastening activity using wearable

sensor technology," Undergraduate Honors Thesis, Texas A&M University (TAMU), College Station, TX, USA, May 2017. Advisor: Tracy Hammond.

[70] M. A. Hall, *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[71] M. Kadous, "Detecting driving with a wearable computing device." `https://www.google.com/patents/US9037125`, May 19 2015. US Patent 9,037,125.

[72] Merriam-Webster, "Haptic." `https://www.merriam-webster.com/dictionary/haptic`.

[73] M. Prasad, M. I. Russell, and T. A. Hammond, "A user centric model to design tactile codes with shapes and waveforms," in *Haptics Symposium (HAPTICS), 2014 IEEE*, pp. 597–602, Feb 2014.

[74] M. Prasad, P. Taele, A. Olubeko, and T. Hammond, "Haptigo: A navigational tap on the shoulder," in *Haptics Symposium (HAPTICS), 2014 IEEE*, pp. 339–345, Feb 2014.

[75] M. Prasad, M. Russell, and T. A. Hammond, "Designing vibrotactile codes to communicate verb phrases," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, pp. 11:1–11:21, Oct. 2014.

[76] M. Prasad, P. Taele, D. Goldberg, and T. A. Hammond, "Haptimoto: Turn-by-turn haptic route guidance interface for motorcyclists," in *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, (New York, NY, USA), pp. 3597–3606, ACM, 2014.

APPENDIX A

CONFUSION MATRICES FOR PHASE I TESTING

A.1 Phase I Feasibility Testing

Buckling	Not Buckling	← classified as
535	191	Buckling
211	2522	Not Buckling

Table A.1: IBk confusion matrix for Phase I feasibility testing.

Buckling	Not Buckling	← classified as
541	185	Buckling
275	2458	Not Buckling

Table A.2: J48 confusion matrix for Phase I feasibility testing.

Buckling	Not Buckling	← classified as
533	193	Buckling
299	2504	Not Buckling

Table A.3: Multilayer Perceptron confusion matrix for Phase I feasibility testing.

Buckling	Not Buckling	← classified as
575	151	Buckling
379	2354	Not Buckling

Table A.4: Naive Bayes confusion matrix for Phase I feasibility testing.

Buckling	Not Buckling	← classified as
541	185	Buckling
156	2577	Not Buckling

Table A.5: Random Forest confusion matrix for Phase I feasibility testing.

Buckling	Not Buckling	← classified as
396	330	Buckling
189	2544	Not Buckling

Table A.6: SMO confusion matrix for Phase I feasibility testing.

Buckling	Not Buckling	← classified as
0	726	Buckling
0	2733	Not Buckling

Table A.7: ZeroR confusion matrix for Phase I feasibility testing.

A.2 Phase I Advanced Testing

Buckling	Not Buckling	← classified as
51	0	Buckling
1	420	Not Buckling

Table A.8: IBk confusion matrix for Phase I advanced testing.

Buckling	Not Buckling	← classified as
46	5	Buckling
5	416	Not Buckling

Table A.9: J48 confusion matrix for Phase I advanced testing.

Buckling	Not Buckling	← classified as
51	0	Buckling
0	421	Not Buckling

Table A.10: Multilayer Perceptron confusion matrix for Phase I advanced testing.

Buckling	Not Buckling	← classified as
48	3	Buckling
0	421	Not Buckling

Table A.11: Naive Bayes confusion matrix for Phase I advanced testing.

Buckling	Not Buckling	← classified as
48	3	Buckling
0	421	Not Buckling

Table A.12: Random Forest confusion matrix for Phase I advanced testing.

Buckling	Not Buckling	← classified as
50	1	Buckling
0	421	Not Buckling

Table A.13: SMO confusion matrix for Phase I advanced testing.

Buckling	Not Buckling	← classified as
0	51	Buckling
0	421	Not Buckling

Table A.14: ZeroR confusion matrix for Phase I advanced testing.

A.3 Phase I Real-Time Testing

Buckling	Not Buckling	← classified as
74	18	Buckling
25	1325	Not Buckling

Table A.15: IBk confusion matrix for Phase I real-time testing.

Buckling	Not Buckling	← classified as
69	23	Buckling
27	1323	Not Buckling

Table A.16: J48 confusion matrix for Phase I real-time testing.

Buckling	Not Buckling	← classified as
80	12	Buckling
22	1328	Not Buckling

Table A.17: Multilayer Perceptron confusion matrix for Phase I real-time testing.

Buckling	Not Buckling	← classified as
85	7	Buckling
132	1218	Not Buckling

Table A.18: Naive Bayes confusion matrix for Phase I real-time testing.

Buckling	Not Buckling	← classified as
61	31	Buckling
5	1345	Not Buckling

Table A.19: Random Forest confusion matrix for Phase I real-time testing.

Buckling	Not Buckling	← classified as
65	27	Buckling
12	1338	Not Buckling

Table A.20: SMO confusion matrix for Phase I real-time testing.

Buckling	Not Buckling	← classified as
0	92	Buckling
0	1350	Not Buckling

Table A.21: ZeroR confusion matrix for Phase I real-time testing.

APPENDIX B

CONFUSION MATRICES FOR PHASE II TESTING

B.1 Phase II Baseline Testing

Buckling	Not Buckling	← classified as
234	44	Buckling
79	1033	Not Buckling

Table B.1: IBk confusion matrix for Phase II baseline testing.

Buckling	Not Buckling	← classified as
198	80	Buckling
91	1021	Not Buckling

Table B.2: J48 confusion matrix for Phase II baseline testing.

Buckling	Not Buckling	← classified as
225	53	Buckling
49	1063	Not Buckling

Table B.3: Multilayer Perceptron confusion matrix for Phase II baseline testing.

Buckling	Not Buckling	← classified as
245	33	Buckling
290	822	Not Buckling

Table B.4: Naive Bayes confusion matrix for Phase II baseline testing.

Buckling	Not Buckling	← classified as
204	74	Buckling
29	1083	Not Buckling

Table B.5: Random Forest confusion matrix for Phase II baseline testing.

Buckling	Not Buckling	← classified as
163	115	Buckling
40	1072	Not Buckling

Table B.6: SMO confusion matrix for Phase II baseline testing.

Buckling	Not Buckling	← classified as
0	278	Buckling
0	1112	Not Buckling

Table B.7: ZeroR confusion matrix for Phase II baseline testing.

B.2 Phase II Advanced Testing

Buckling	Not Buckling	← classified as
230	37	Buckling
89	1015	Not Buckling

Table B.8: IBk confusion matrix for Phase II advanced testing.

Buckling	Not Buckling	← classified as
213	63	Buckling
87	1017	Not Buckling

Table B.9: J48 confusion matrix for Phase II advanced testing.

Buckling	Not Buckling	← classified as
229	47	Buckling
56	1048	Not Buckling

Table B.10: Multilayer Perceptron confusion matrix for Phase II advanced testing.

Buckling	Not Buckling	← classified as
242	34	Buckling
190	914	Not Buckling

Table B.11: Naive Bayes confusion matrix for Phase II advanced testing.

Buckling	Not Buckling	← classified as
224	52	Buckling
31	1073	Not Buckling

Table B.12: Random Forest confusion matrix for Phase II advanced testing.

Buckling	Not Buckling	← classified as
211	65	Buckling
55	1049	Not Buckling

Table B.13: SMO confusion matrix for Phase II advanced testing.

Buckling	Not Buckling	← classified as
0	276	Buckling
0	1104	Not Buckling

Table B.14: ZeroR confusion matrix for Phase II advanced testing.

B.3 Phase II Leave-One-Out Testing

Buckling	Not Buckling	← classified as
181	85	Buckling
128	976	Not Buckling

Table B.15: IBk confusion matrix for Phase II leave-one-out testing.

Buckling	Not Buckling	← classified as
166	110	Buckling
44	1060	Not Buckling

Table B.16: Random Forest confusion matrix for Phase II leave-one-out testing.

B.4 Phase II Dimensionality Testing

Buckling	Not Buckling	← classified as
228	48	Buckling
100	1004	Not Buckling

Table B.17: IBk confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
183	93	Buckling
85	1019	Not Buckling

Table B.18: J48 confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
219	57	Buckling
60	1044	Not Buckling

Table B.19: Multilayer Perceptron confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
244	32	Buckling
153	951	Not Buckling

Table B.20: Naive Bayes confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
215	61	Buckling
50	1054	Not Buckling

Table B.21: Random Forest confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
173	103	Buckling
59	1045	Not Buckling

Table B.22: SMO confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
0	276	Buckling
0	1104	Not Buckling

Table B.23: ZeroR confusion matrix for Phase II advanced testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
197	79	Buckling
164	940	Not Buckling

Table B.24: IBk confusion matrix for Phase II leave-one-out testing with CFS feature reduction.

Buckling	Not Buckling	← classified as
177	99	Buckling
82	1022	Not Buckling

Table B.25: Random Forest confusion matrix for Phase II leave-one-out testing with CFS feature reduction.

B.5 Phase II Verification Testing

Buckling	Not Buckling	← classified as
53	6	Buckling
38	198	Not Buckling

Table B.26: IBk confusion matrix for Phase II verification testing.

Buckling	Not Buckling	← classified as
50	9	Buckling
36	200	Not Buckling

Table B.27: J48 confusion matrix for Phase II verification testing.

Buckling	Not Buckling	← classified as
51	8	Buckling
25	211	Not Buckling

Table B.28: Multilayer Perceptron confusion matrix for Phase II verification testing.

Buckling	Not Buckling	← classified as
58	1	Buckling
82	154	Not Buckling

Table B.29: Naive Bayes confusion matrix for Phase II verification testing.

Buckling	Not Buckling	← classified as
48	11	Buckling
26	210	Not Buckling

Table B.30: Random Forest confusion matrix for Phase II verification testing.

Buckling	Not Buckling	← classified as
50	9	Buckling
19	217	Not Buckling

Table B.31: SMO confusion matrix for Phase II verification testing.

Buckling	Not Buckling	← classified as
0	59	Buckling
0	236	Not Buckling

Table B.32: ZeroR confusion matrix for Phase II verification testing.