UNIVERSITY OF CALIFORNIA

Los Angeles

Context Guided and Personalized Activity Classification System for Wireless Health

A thesis submitted in partial satisfaction

of the requirements for the degree Master of Science

in Electrical Engineering

By

James Yan Xu

2011

The thesis of James Yan Xu is approved.

_____
William J. Kaiser

_____
Lara Dolecek

_____
Gregory J. Pottie, Committee Chair

University of California, Los Angeles

2011

This thesis is dedicated to my parents, Dr. Peter Xu and Helen Li, for without their unwavering support this great adventure would not have been possible.

I would like to thank Professor Gregory J. Pottie and Professor William J. Kaiser, for their guidance, inspirations, and financial support.

I would like to thank Professor Lara Dolecek, for her exceptional instructions on topics behind many parts of this thesis.

Incoming fire has the right of way.

Old artilleryman's maxim

Table of Contents

List of Figures

List of Tables

ABSTRACT OF THE THESIS

Context Guided and Personalized Activity Classification System

for Wireless Health

by

James Yan Xu

Master of Science in Electrical Engineering

University of California, Los Angeles, 2011

Professor Gregory J. Pottie, Chair

Continued rapid progress in the development of embedded motion sensing enables wearable devices that provide fundamental advances in the capability to monitor and classify human motion, detect movement disorders, and estimate energy expenditure. With this progress, it is becoming possible to provide, for the first time, evaluation of outcomes of rehabilitation interventions and direct guidance for advancement of subject health, wellness, and safety. The progress in motion classification relies on both the performance of new sensor fusion methods that provide inference, and the energy efficiency of energy-constrained monitoring sensors. As will be described here, both of these objectives require advances in the capability of detecting and classifying

the location and environmental context. Context directly enables both enhanced motion classification accuracy and speed through reduction in search space, and reduced energy demand through context-aware optimization of sensor sampling and operation schedules. There have been attempts to introduce context awareness into activity monitoring with limited success, due to the ambiguity in the definition of context, and the lack of a system architecture that enables the adaptation of signal processing and sensor fusion algorithms specific to the task of personalized activity monitoring. In this thesis we present a novel end-to-end system that provides context guided personalized activity classification. With a refined concept of context, the system introduces interface models that feature a context classification committee, the concept of context specific activity classification, the ability to manage sensors given context, and the ability to operate in real time using wireless sensors. We also present an implementation that demonstrates accurate context classification, accurate activity classification using context specific models with improved accuracy and speed.

Chapter 1

Introduction

1.1 Background

The rapid advance in microelectronics has provided MEMS inertial sensors, low power processors, and low cost monitoring systems applicable to human motion classification. Many of the most urgent problems in health and wellness promotion, diagnostics and treatment of neurological condition and even athletic performance advancement are now possible. The wireless health community exploits this along with smartphone technologies for integration of monitoring and in field guidance for both advancing and evaluating treatment outcomes.

Recently developed solutions monitor a subject's physical activity, for example walking gait speed monitoring for recovering stroke patients in the field of wireless healthcare [1,2]. For many applications, there is also a need for personalized, targeted monitoring for specific activities, in specific environments. For example, a stroke patient benefits from monitoring of gait speed while in the hospital and then at home to ensure that their mobility is sufficient to enable safe passage through urban areas.

Also, these subjects benefit from monitoring and guidance for aerobic exercises while at home to maximize the effectiveness of recovery routines [3].

A large body of work has focused on the accurate detection of physical activities, using a diverse range of classification and feature extraction techniques [1,2,4,5,6-9]. These methods are confronted with the challenge of classification of a specific, correct motion among many possibilities at any observation time. As the number of potential motions increase, classification reliability is degraded.

In fields including wireless sensor networking, pervasive computing, and others, the concept of context-awareness has been introduced with the objectives of improving human machine interaction, and enabling low energy operation while retaining system performance. Many architectures have been proposed to bring personalization and adaptation to a system [10], and recent attempts have been made to introduce context into activity classification [11,12]. These systems experienced limited success due to ambiguity in the definition of context, and a lack of an appropriate system architecture that is specific to the task of personalized activity monitoring.

1.2 Aim, Objectives and Contributions

This thesis aims at integrating context into activity classification, and based on this, proposes a novel system architecture for personalized, context-guided wireless human motion classification. The objectives of the thesis work cover reliable wireless sensor

instrumentation, context detection, activity classification using context, mobile platform development, and mass data collection.

There are four major contributions in the work presented in this thesis. First, in addressing the deficiencies outlined in Introduction, a novel end-to-end system architecture has been proposed that provides context guided personalized activity classification. The novelty of the proposed architecture lies in three areas: 1) The ability to accurately detect context with multiple sensing modes; 2) The use of context to improve classification accuracy and speed; and 3) The ability to target specific physical activities of interest under selected contexts.

Second, in implementing the proposed architecture, a comprehensive study of the Bayesian Networks (BN), and its uses in activity classification have been presented. Detailed discussions include: 1) The implementation of BN with corresponding message passing inference algorithms; 2) The ability to exploit a BN's visualization advantages to better assist domain experts who are otherwise not trained in machine learning to construct meaningful models with powerful mathematical machinery as a backend; and 3) The difficulties with using inertial data in BNs, and the solution using specially designed discretizers that takes into account a BN's structure.

Third, a number of practical issues have been solved with regards to instrumenting multiple unreliable sensors in a wireless environment. Specifically, the thesis describe ways to reliably control and collect data from them.

Finally, a novel way has been developed to carry out long periods of data collection on multiple subjects. Problems surrounding long recording labeling, unreliable labeling, and synchronization have been solved.

1.3 Related Work

Many investigations in medical science over the last decade have demonstrated the critical benefits of activity monitoring for applications ranging from health and wellness promotion to disease treatment, to performance advancement and injury risk reduction in athletics. One example is the use of motion and sound data sources in an application that provides telemonitoring for elderly individuals living independently [6]. Here, a method was developed that can detect when a user requires attention (as a result of a fall or long periods of inactivity). In another study, accelerometer sensor data sources and machine learning algorithms were applied for monitoring intervention effectiveness of acute stroke patients [1]. The technology provides physicians with the ability to directly measure a patient's activity level, even after discharge. This improves on the surrogate laboratory measurements, administered only in a clinical setting. An example of applications in athletics was presented in [7], where multiple accelerometers were used for ambulatory monitoring of elite athletes in both competitive and training environments. For swimmers, the characteristics of strokes can be captured and analyzed. For rowers, the addition of an impeller combined with accelerometer data was used to recover intra and inter stroke phases for

performance analysis. This system was used by Australian Olympic athletes in training for competition in the 2004 Olympic Games.

Using sensors for activity monitoring has been studied extensively. In [8], a system using iPhone and Nike+iPod sport kit was proposed for classifying human activities. The activities considered include running, walking, bicycling, and sitting. In [4], a complex environment with many microphones, video sources and other sensors was designed. The study attempted to accurately track movements of arms and hands. Activities considered there are bathing, dressing, toileting, eating, and others. Results indicated that using one third of the 300 available sensors in the specially designed lab, tasks can be detected with an accuracy of 90%. A specially designed glove was introduced for activity classification [5]. The glove detects and records objects a user touches using an RFID reader. In this system, all the objects being monitored (such as utensils, toothbrushes, and appliances) need to have RFID tags instrumented.

Most of the studies above are restricted in the number of activities they can detect accurately. These systems are designed either for a specific set of activities that may not be easily modified, or have a high system installation cost with the requirement to modify environments and also monitor subjects only when they are present in these environments.

The recognition of user and environment context has been identified as a primary capability for advancing the performance and capability of human-computer interfaces in many fields [11]. Studies have emerged recently in wireless health that attempt to

combine context and activity classification. In [9], a multi-sensor wearable system was proposed that enables a context that largely consists of physical activities. There, 30 sensors were embedded into a garment, with multiple processing nodes responsible for distributed processing of sensor data. This study treated physical activities as contexts, and focused on the sensor fusion development. A system for a context-aware mobile phone named Sensay was developed in [12]. This includes context defined as a set of user states (normal, idle, uninterruptable). By introducing light, motion and microphone sensors, Sensay is able to detect these contexts and manipulate ringer volume, vibration, and phone alerts. At MIT media lab, a system using audio information to obtain environmental context is described in [6]. The system adopted HMM algorithm to perform the classification, which can classify contexts such as office, supermarket and busy street in real-time.

In the investigations conducted so far, the definition of context has varied significantly between investigations. It is particularly important for activity classification systems to define contexts such that they do not contain physical activities, as these should be classified after a context has been determined. A new definition of context will be introduced for the proposed system architecture in this thesis study.

Chapter 2

System Design

This chapter starts with the definition of contexts and scenarios, then presents a high level description of the system, followed by describing the system architecture in detail.

2.1 Context

First, we present the concept of a context, as it is fundamental to the rest of the thesis. When addressing context, many investigations use the important definition by Dey [13]. While powerful, this definition of context that includes every characteristics of a given situation, in terms of both the environment and the user, is very broad. Useful for some applications, it is not suitable for leveraging context in monitoring physical activities, as in many cases a context contains physical activities that are underlying in the definition. There are a number of alternative definitions available in the field of pervasive computing, offering different selection of divisions, such as external and internal contexts [14,15]. These definitions are usually narrower, but still contain a mix of physical activities with other environmental attributes.

In this thesis, a context is defined thusly:

> **Definition 1: A context is a subset of all attributes that characterizes an environment or situation, external to the user**

This definition clearly distinguishes between the external environment, and the user's physical activities. By means of this definition, there is a clear guideline for deciding which attribute is associated with context and which is associated with physical activity. For example, a "meeting" environment is a context, and its characteristics may involve certain sound profiles and a set of possible locations. "Sitting in a meeting" in contrast is not a context, as it contains the user's physical activity of "sitting".

2.2 Scenario

The term scenario is used throughout this thesis, and refers to a collection of contexts and activities of interest for a particular user. Within each context there is a set of activities of interest, and we can build models of these activities. As an example, suppose we believe that by using an accelerometer on the ankle and looking at the standard deviation in the horizontal direction, we can tell if a person is walking, running or standing still. Here the accelerometer on the ankle is a sensor, the standard deviation on the horizontal axis is a feature (derived from sensor data), and activities (walking, running, standing) are classes. Further assuming that the distribution on the standard deviation is Gaussian, we obtain a model that links a feature to a set of

classes (and the model has only two parameters: mean and variance). From here, we can carry out a number of different methods to distinguish between the classes given an observation of the feature set, and these methods are called classifiers. A simple example using Naive Bayes classifiers can be found in [16].

With this knowledge, a scenario is defined as:

> **Definition 2: A scenario consists of a set of interested contexts with a model for distinguishing them. Under each context, a model is required that describes a set of interested activities .**

Using the concept of a scenario, we can now describe the proposed system. For example scenarios, refer to Section 2.9.

2.3 High Level Description

High level functionalities of the system can be described by Figure 2.1. Through instrumentation of various sensors, we obtain both inertial data which describe motions, and environmental data which describe contexts. The data then flow through a signal processing pipeline which would determine the user's context, and produce a motion classification based on the context and inertial sensor data. The set of contexts and motions that are considered, as well as their relationships are determined by a scenario fed into the signal processing pipeline. This scenario can be built by domain experts in areas such as primary health care and personal training. The final output of

the system is a user's current context and motion, which can be consumed by any application that requests this data. These two pieces of information provide two entirely new dimensions which could enhance the capability of many software and applications.

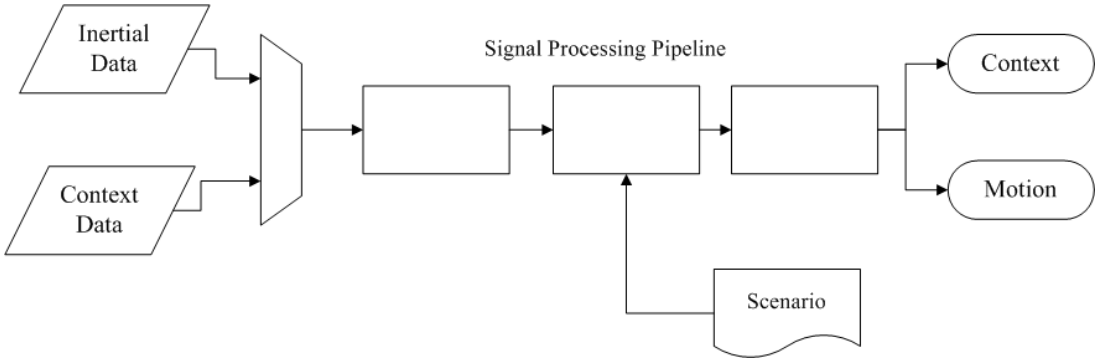


Figure 2.1: High level system description

2.4 System Architecture

We propose here an architecture that is able to provide context guided activity classification, with the capability for real time online operation and multimodal sensor instrumentation. A set of sensors on the user's side provide data to a core classification system. The core system detects a user's context first, and activity classification models are selected based on this. Both the detected context and classified activity are returned to the user, ready to be consumed by 3rd party applications.

To enable such a system, there are a number of integral components: 1) A way to interface with various sensors; 2) A way to interface with the main classification

systems; 3) A way to provide visual guide to the end-user with regards to training and live feedback of classification results; 4) A core system that provides the inference services which would determine a user's context and activity; and 5) A way for domain experts to prescribe scenarios for users.

2.4.1 Client Server Architecture

If we consider that the models are to be built by domain experts, classifications are done by a core system, and the classification results are to be consumed by end-user applications, then this is broken down to a well known Client-Server architecture. An end-user client can handle the instrumentation of sensors, interface with core systems, and provide visual guide. A domain expert client can handle the construction of scenarios, and send them to the server. A server can then implement the required core systems, and provide a way for clients (both end-user and domain expert) to interact with them. Figure 2.2 depicts the architecture of this new system.

This architecture provides context detection and activity classification, where the context information is utilized by an activity classification system, along with inertial sensor data. The end-user client application is used for collecting sensor data and labels from a user, and also for displaying results. A corresponding web service runs on the server, and acts as a gateway between the client and the core system. This provides a means for the client to transmit and access data through a network, in a structured manner. Domain experts can build scenarios, and assign them to individual users through a domain expert client.

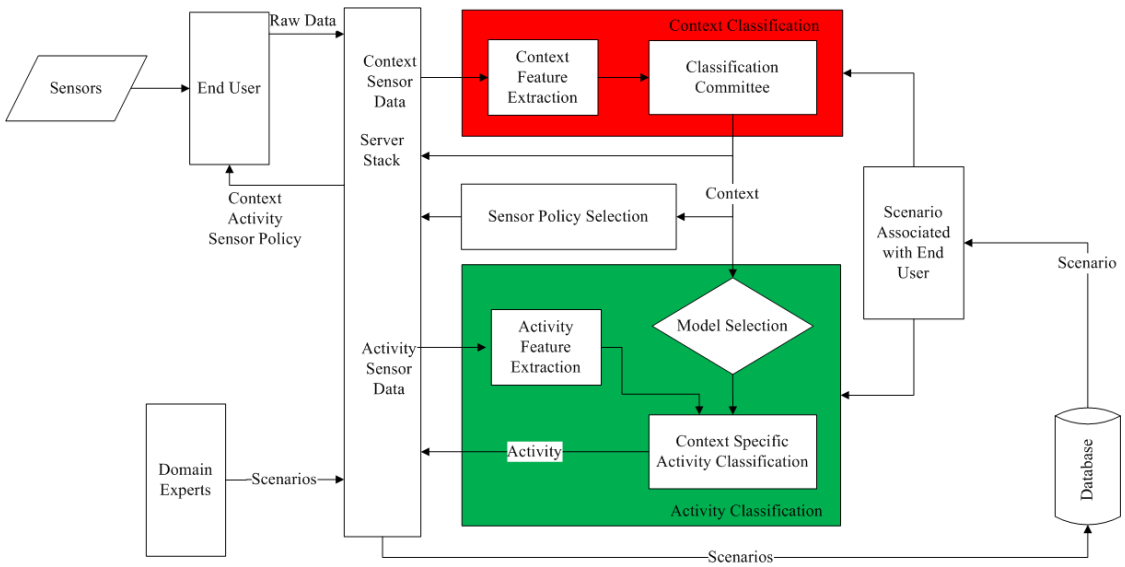


Figure 2.2: System architecture

2.4.2 Object Oriented Architecture

It is important to note that at each step in design and implementation, individual subsystems should be modeled with objects, and the entire system be defined by a set of interfaces and relationships. Each software interface is characterized by their public methods, defined by functionality, expected inputs and expected outputs. By implementing an interface, a class agrees to provide all the methods of that interface [17]. In this way, each subsystem is developed independently without the requirement to reveal its specific implementations, but only that it implements the required interface. This allows any part of our proposed system to be overridden by custom realizations, allowing for rapid prototyping and evaluation of various algorithms.

2.4.3 Message Oriented Architecture

For communication between clients and the server, a message oriented architecture is popular [18]. This decouples the client and server, where both only need to respond to messages they understand, and send messages that the other expects.

Using object oriented concepts and combing with message based client server architecture, we can model the system as depicted in Figure 2.2 with interface models in Figure 2.3 - 2.6.

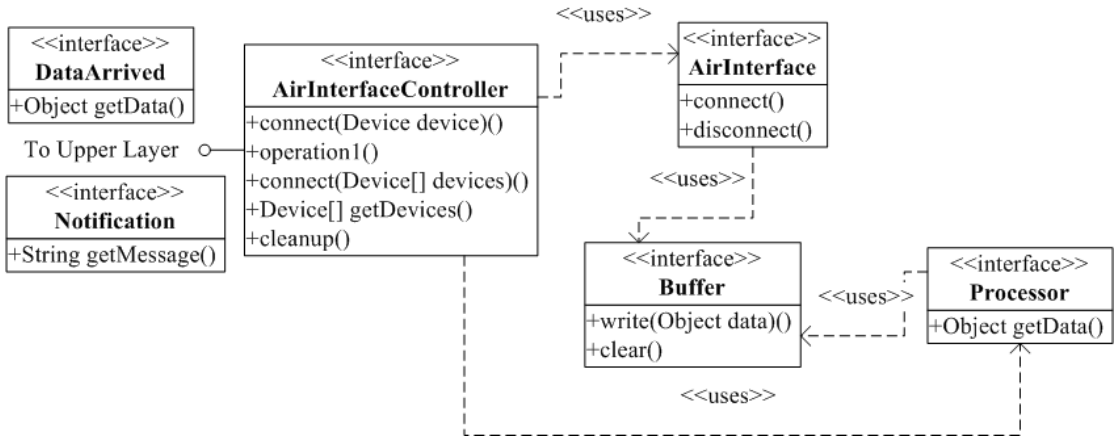


Figure 2.3: End-user client interfaces model

Figure 2.3 describes the end-user client. We see that the SensorData interface provides required abstraction for representing both context and inertial sensor data, and the message client sends messages according to Figure 2.5.

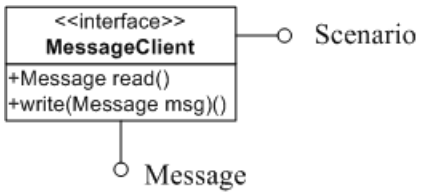


Figure 2.4: Domain expert client interfaces model

Figure 2.4 describes the client for domain experts. A software is given to them that generates scenarios, and these scenarios can then be sent to the server through a message client. Again, the messaging structure is in Figure 2.5.

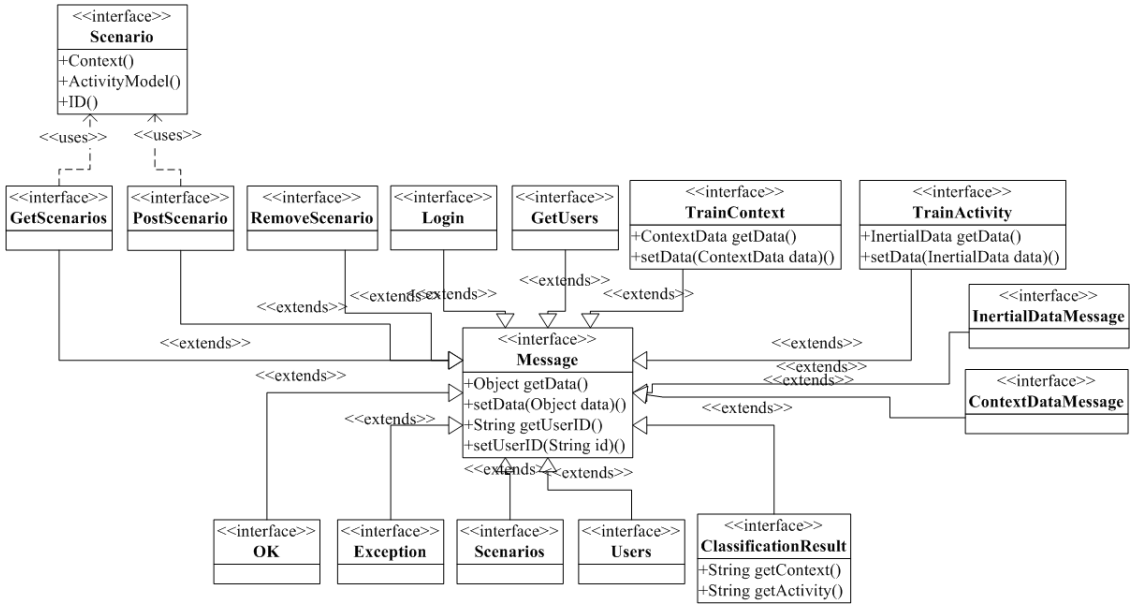


Figure 2.5: Message system interfaces model

The messaging structure is shown in Figure 2.5. Note that all messages extend a base Message interface, which defines the contract for basic functionalities that all messages must have. Top half of the figure shows messages a MessageClient can send,

and bottom half represents the possible server responses. Through this messaging structure, both the end-user and domain expert clients can fully communicate with a server implementing the core systems below.

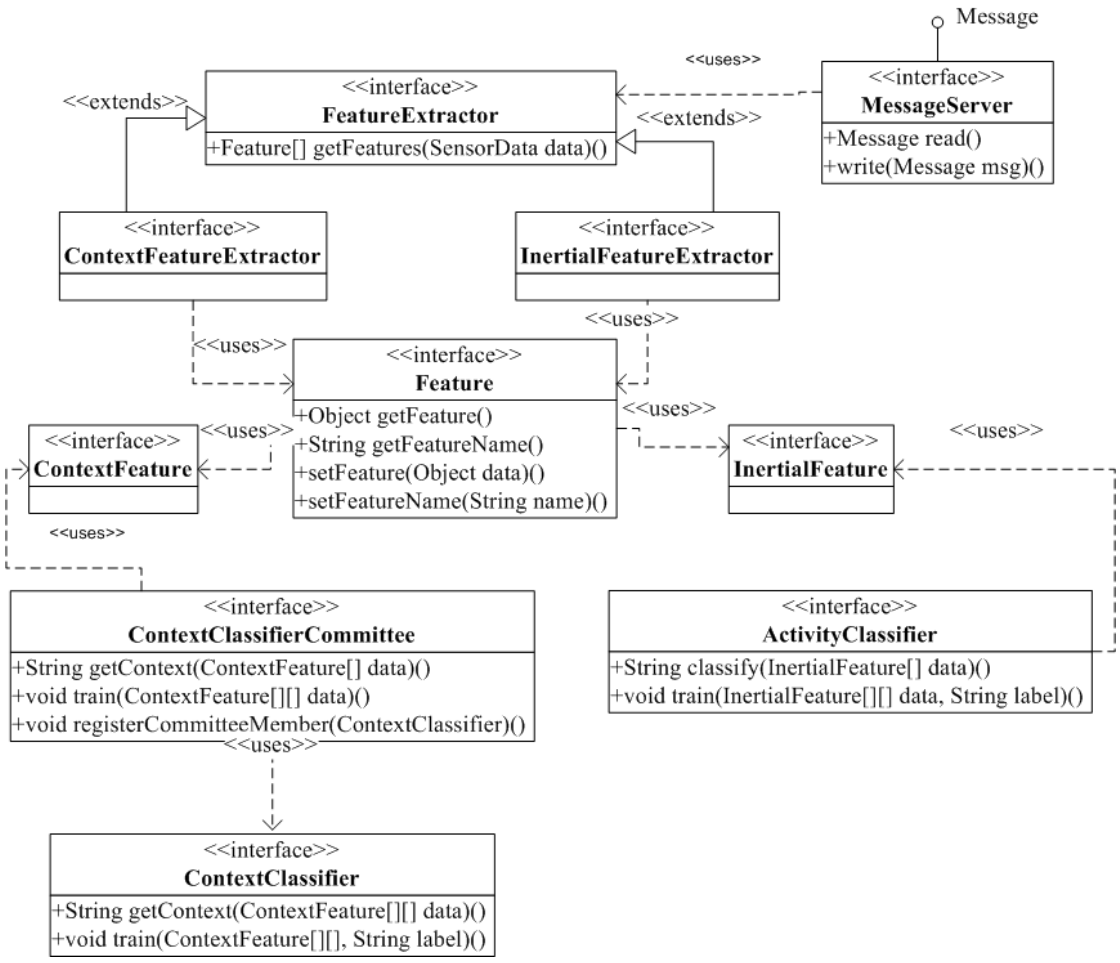


Figure 2.6: Server core systems interfaces model

This set of interfaces describes the core systems. Both context and activity classifiers require features to be extracted by a feature extractor implementing FeatureExtractor. The message server implements MessageServer, and delegate appropriate actions. For

example, if a TrainActivity message is received, then the InertialData received from the message is delegated to InertialFeatureExtractor, and the ActivityClassifier's train() method is called.

As a concrete example of the flexibility using interfaces, consider the ContextClassifier interface, which covers the context classifier. Teams can develop classifiers independently and optimize them according to particular applications. As long as the classifiers provide the getContext method, they can be hot swapped into the system to adjust the classification system behavior without affecting the overall system.

2.4.4 Sensor Instrumentation

One requirement of the architecture on the end-user client is the ability to reliably obtain data from external sensors. By reliably, we mean that the client should be able to: 1) Automatically detect, connect and control the external sensors (start, stop, record) in real time; 2) Know the status of the sensors at all times; 3) Recover from corrupted data, missing data, delay; and 4) Be able to run for an extended amount of time (many hours) without accumulating error.

We propose the following air architecture for instrumenting external sensors (Figure 2.7).

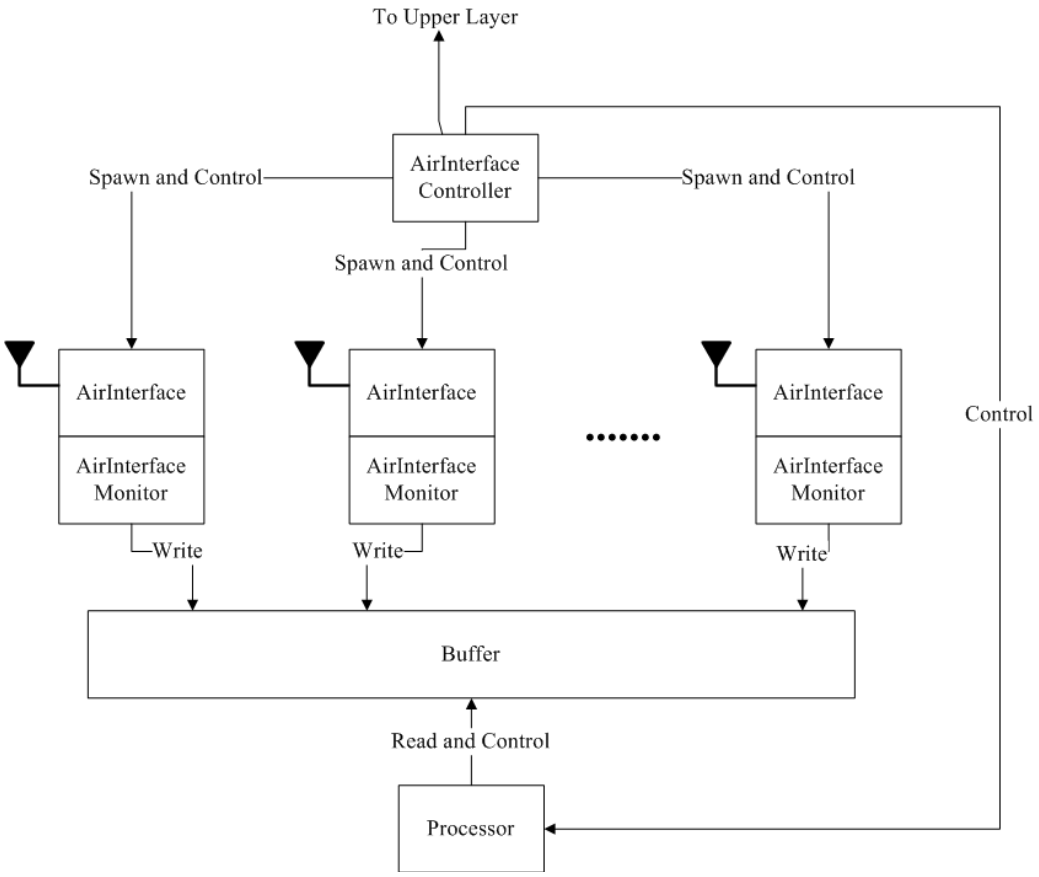


Figure 2.7: Sensor instrumentation architecture

First, the subsystem closest to actual hardware is the AirInterface, its implementation should be as simple as possible, supporting only basic read and write operations required for sensor control and data recording. This is so that the subsystem can execute as fast as possible. Attached to the air interfaces are monitors. There should be one monitor per interface, as the sensors being instrumented can be different, thus requiring different monitoring. It is the monitor's job to track a sensor's state, and both notify upper layer of changes, and take appropriate actions autonomously. For example, if it is detected that a sensor has disconnected, the monitor could notify the

upper layer about the disconnection, while trying to re-establish connection (through the air interface it is attached to).

Each AirInterface obtains data from a stream established to the target device, and from there the data is tagged with a device id, and stored in a central buffer. A processor unit (Processor) runs in parallel to all the air interfaces and processes the buffer. It is the processor's job to synchronize the data from multiple sensors. This is a standard producer-consumer pattern, where the processing unit is decoupled from the recording units through a buffer to ensure that the recording units are not blocked waiting for processing. It is essential, as we cannot have the sensors hang due to insufficient processing speed. Using this buffer, we also have protection against spurious delays. It is also important for the processor to monitor the buffer state to make sure it is not overflowing, as this is an indication that the processing speed cannot keep up with recording speed. In case that the processing speed is not fast enough, measures need to be taken, such as having multiple processors running in parallel, or using memory storage devices to store the data.

Abstracting the underlying sensor instrumentation system is the AirInterface Controller. This controller offers upper layers the ability to initiate connection to sensors, and to obtain synchronized recording data. Figure 2.8 shows the interfaces model of this system.

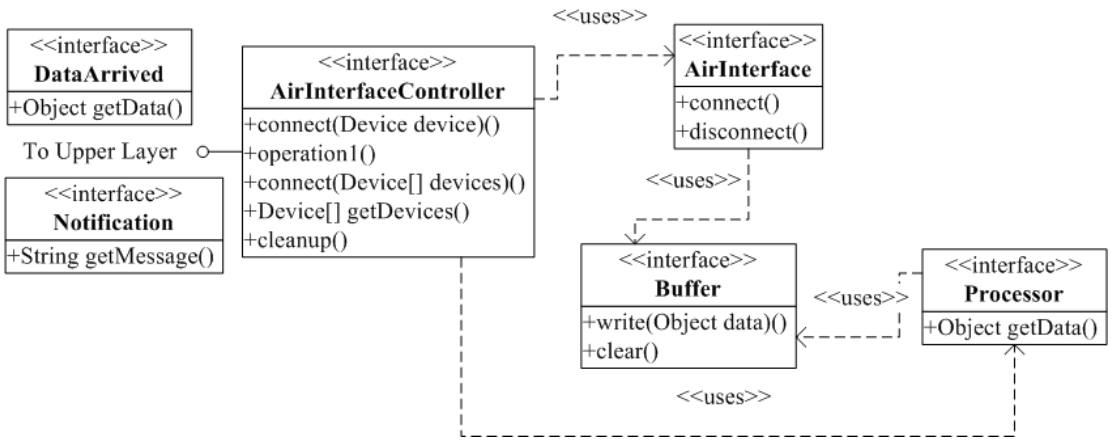


Figure 2.8: Sensor instrumentation interfaces model

Noticeably lacking are specific data related methods on the AirInterface and Processor, as well as the lack of read() on Buffer. To maintain maximum flexibility over a myriad sensor types, the interfaces here are loosely defined, and the four pieces (controller, air interface, buffer, processor) should be implemented as a coherent unit. Only the controller needs an interface for abstracting with upper layers. The only communication to the upper layers is through messages marked by DataArrived, and Notification interfaces. Refer to Section 3.5.1 for a concrete example implementation of the system.

2.5 Context Detection

Definition 1 is designed to capture a large number of situations, so that users with different objectives can define their own sets of useful contexts. They can then identify required characteristics, and select necessary sensors. This generalization

makes classification difficult, as we need to account for a diverse range of data sources such as GPS coordinates, wireless information, audio, and illumination level. For example, consider the following datasets (Table 2.1), demonstrating the result of an experimental system that combines both audio signal processing on sound detected in an environment, along with the Media Access Control (MAC) addresses associated with wireless access points detected in the same environment. Here, three locations were used, and MAC addresses and peak frequency of the audio power spectral density (PSD) were recorded.

Table 2.1: Example datasets

Label	**Wireless MAC address**	**Audio PSD peak frequency**
A	{30:46:9a:06:4d:e0, 00:0c:41:6e:1e:f6}	390.01Hz
B	{2e:25:b3:96:d5:f9, 00:b0:d0:86:bb:f7}	80.91Hz
C	{6e:51:f5:c1:11:00, 00:0c:f1:56:98:ad}	120.19Hz

To separate labels A, B and C in this example, a common method is to find thresholds that divide them, based on the MAC address and PSD peak frequencies. It is clear that the audio peak frequency data for environments can be assembled, and a distinct separation between the labels can be found. The proper treatment of MAC addresses is less clear. It is challenging to represent these identifier values in a feature space, and to define a separation. This difference in data type determines the suitability of various

classifiers. For example, classifiers based on SVM are not suitable for treatment of MAC addresses, whereas a method such as kNN has been used successfully [19].

In order to detect context based on a variety of data sources, there is a need to use multiple classifiers for different features. To this end, we propose the development of a classification committee consisting of n individual classifiers (Figure 2.9). The individual classifiers are trained separately, and after training they can be tested for individual accuracy. A voting weight (α) can be determined for each classifier, proportional to the perceived accuracy. When an unknown class is encountered, the committee performs a linear combination of the individual classifiers, and the context with the highest vote is the output. In interface model 2.6, the committee is represented by ContextClassifierCommittee, and individual classifiers should implement ContextClassifier. They can then be registered by calling registerCommitteeMember() method on the classifier committee implementation.

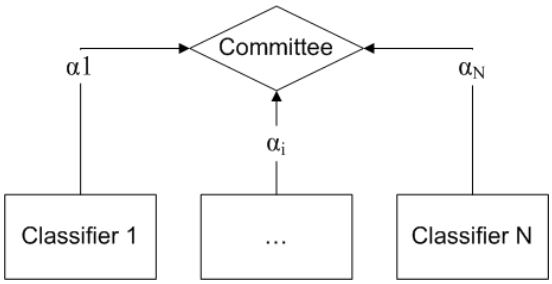


Figure2.9: Context classification committee architecture

This committee approach not only allows fusion of a number of sensors with various data types, but also allows for adaptation of context detection to individuals with

varying habits. For the former, it is easy to see that different classifiers can be selected to compose the committee, depending on inputs. For the latter, suppose a habitual individual exhibits strong patterns in time of day relating to context. The weight of a classifier based on time of day will be adjusted during training so that the habitual subject would have a time classifier with higher vote weighting (compared to a subject that is irregular).

2.6 Context Guided Activity Classification

As described in the previous section, the context is determined through the context classifier committee. This context decision then determines the model/method used for activity classification. We now introduce the concept of a context guided classifier. These classifiers allow us to have specifically optimized models that each focus on the activities of interest, given context. Unlike conventional activity monitoring, there is no single list of comprehensive activities to be built into a monolithic classifier. Instead, a basic set of activities common across all contexts can be chosen, and this set can then be extended or reduced should the need arise for a particular context. These models can be assigned to specific users, giving us the ability to personalize the activities being monitored.

Based on context information, the model selector (Figure 2.2) would select an appropriate activity recognition model from the current scenario, and the activity classifier can then make a classification based on the model. There are a number of

benefits from using this system. First, we can improve classification accuracy and speed due to a simplification of feature space. Then, the system allows scenarios to be determined by investigators, and a person's monitoring program can be modified. Finally, the system gives a user the ability to control his/her privacy. Unlike most other monitoring systems that are always on, a user can decide to only allow monitoring under specific contexts, for specific activities.

2.7 Sensor Control

By having scenarios describing the contexts and activities of interest, we can also optimize sensor sampling rate and selectively enable or disable sensors to reduce energy demand. For example, in contexts where no upper body motions are monitored, the upper body sensors could be disabled or their sampling rate can be reduced. The benefits of this are an overall reduction of power, storage and communication usage.

2.8 Modes of Operation

There are three modes of operation supported by this framework: 1) Construction of models by healthcare professionals such as doctors, registered nurses, personal trainers; 2) Initial training from individual end-users of the classification system for both context and motion; and 3) Live monitoring of the user's context and motion. The first mode is straightforward, where an expert simply logs in to the server, construct a scenario and assign it to a user.

The training scheme required is dictated by underlying classifier implementations, some would require individualized training, while others can function with a generic training set or none at all. If training is required, then once a scenario has been prescribed, an end-user's client would parse this scenario for required parameters, and prompt the user to perform a set of activities under certain contexts to collect the training data. Visual cuos to guide a user through training should be implemented, and the training data can be sent to a server via training messages (TrainContext and TrainActivity). For example, a scenario that monitors walking and running while at the gym would require the user to perform both activities in a gym.

After a scenario has been trained, the end-user client can then go into live mode. Data are collected autonomously in this mode and sent to a server, where a continuous live stream of context and motion classification can be made and returned to the client.

2.9 Example Scenarios

Section 2.1 and 2.2 have defined the concepts of context and scenario, now we present some example usages to show-case the power of this architecture. In the tables, some of the columns such as features and methods will become clearer in subsequent sections.

In primary health care, physicians may often wish to monitor a stroke inpatient's (still in hospital) walking speed and also ensure they are intermittently sitting/standing to

alleviate deterioration in exercise tolerance [3]. The domain expert here is the doctor, and one possible scenario could be Table 2.2.

Once the patient is discharged, physicians may then wish to monitor the patient to make sure that recommended daily activities are performed at home. This could translate to a scenario show in Table 2.3.

Another example is where personal trainers can prescribe personalized training plans for different individuals, including activities of interest and their duration and place (gym, home, office). Here the activity monitoring system can inform the user of his/her training progress, and also track how long the person has stayed inactive (Table 2.4).

Table 2.2: Example scenario 1

Context	Features	Classifier(s)	Activity Model	Features	Classifier	Purpose
Patient room	WiFi	k-nearest-neighbour (kNN)	• Sitting • Standing • Lying down	Accelerometer standard deviation, gravity direction	Naive Bayes	Monitor how long a patient has stayed immobile, assess the risk of bed sores and other problems
Rehabilitation	WiFI	kNN	• Aerobic exercise • Walking Slow • Walking fast • Fall	Acceleration peak magnitude, standard deviation	Naive Bayes	Monitor patient's performance in exercises
Hall way	WiFi, Sound	kNN, AdaBoost	• Standing • Walking fast • Walking slow • Fall	standard deviation	Naive Bayes	Monitor a patient's general physical condition, and detect falls

Table 2.3: Example scenario 2

Context	Features	Classifier(s)	Activity Model	Features	Classifier	Purpose
Home	WiFi, Time of day	kNN	• Aerobic exercise • Walking slow • Walking fast	Acceleration peak magnitude, standard deviation	Naive Bayes	Monitor if the patient is following exercise routine, and also his physical performance

Table 2.4: Example scenario 3

Context	Features	Classifier(s)	Activity Model	Features	Classifier	Purpose
Home	WiFi, Time of day	kNN	• Sitting	standard deviation	Naive Bayes	Monitor if the user is exercising enough
Office	WiFi, Time of day	kNN	• Sitting	standard deviation	Naive Bayes	Monitor if the user is exercising enough
Gym	WiFi, Time of day, Sound	kNN, AdaBoost	• Pushups • Weight lifting • Running	Acceleration peak magnitude, standard deviation	Naive Bayes	Monitor if the user is following the correct routine (amount of time at each activity etc).

Chapter 3

Implementation

3.1 Language and Framework Choices

The first design choice is the language to use. For object oriented designs there are a number of popular languages such as C++, Java and Python. From a development perspective, both Java and Python are much easier to use as they hide away a lot of the complexities that a C++ developer must be mindful of. While Java and Python both provide ways to robustly implement the proposed architecture, there is no direct support for interfaces in Python, and more importantly, development on the mobile platform Android requires Java. While we can implement different parts of the system with different languages, there is no advantage to be had by implementing parts in Python. Also, some areas are computationally intensive, and Java is generally much faster than Python [20]. Finally, having a common language (i.e. Java) lets us use techniques such as object serialization, which provides an easy way to transmit data between client and server (Section 3.2.1).

Section 2.4.1 talked about the need for both an end-user client, and a domain expert client. For the end-user client, an ideal candidate is a mobile application supported by a smartphone for two primary reasons: First, mobile devices are pervasive, which makes the client accessible, and we can leverage services off existing network infrastructure that is available in the residential, workplace, and clinic environments, where the systems reported here are deployed; Second, mobile devices are high performance, so they are able to act not only as user interface platforms, but also as wireless sensor hubs that can log, process and store data from the wearable sensors. The two main contenders here are Apple's iOS and Google's Android. Due to a number of restrictions on iOS such as only being able to interface with Apple approved Bluetooth devices, Android was chosen as the development platform for end-user client.

Domain expert clients are designed to run on stationary computers (laptops, netbooks, desktops), as they are primarily for user management and scenario building. Here Java can be used, and Nokia's Qt library [21] is chosen for Graphical User Interface (GUI) development for the following reasons: 1) It is newer and has a more friendly appearance than Java's native Swing; 2) It is cross-platform; and 3) Qt provides not only a GUI library, but also libraries for audio, video, network etc, which allows for easy extension development later on.

3.2 Server Implementation

The server serves as a gateway between the client and server for real time classification. It also implements the whole automation process.

3.2.1 Object Serialization and Transfer through Network

Section 2.4.3 described the message oriented architecture and how it can be used for context guided activity classification. In this implementation, the server is also a native Java program. This has two advantages, first, the server is able to natively interface with all the other core systems, as they are all developed in Java. Also, messages from the clients can be sent using a technique known as serialization, by which an object is converted into a binary data string that can be transmitted over any stream (network stream in this case). Figure 3.1 demonstrates the process of serialization.

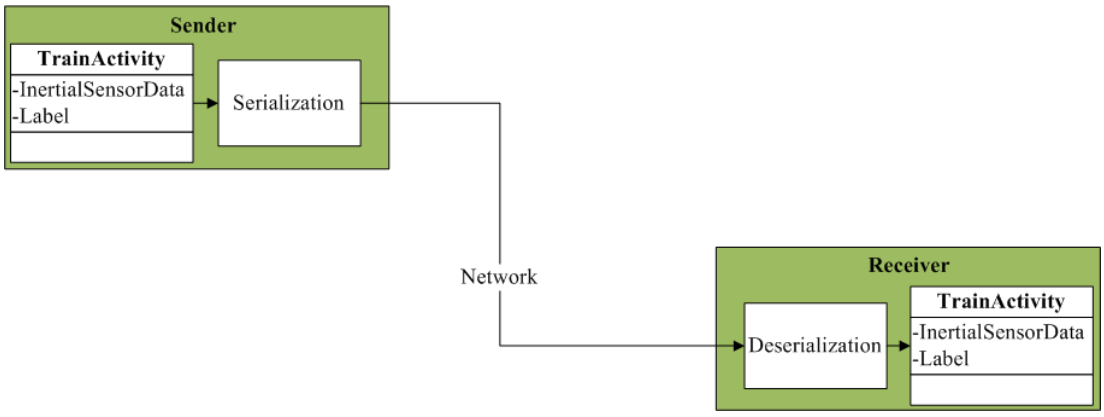


Figure 3.1: Serialization, deserialization process

Here TrainActivity message is constructed at the client (sender) with the InertialSensorData and label filled in. This message object is converted into a data string, transmitted through the network, and a server simply deserializes to obtain the original object exactly, complete with all data members. Without using this technique, developers must come up with their own packing formats to pack objects for transferring. Note that Java has native support for serialization through the Serializable interface [22].

3.2.2 User and Scenario Management

A central concept of the proposed architecture is the ability to prescribe scenarios for individuals by domain experts such as doctors. This requires a user and scenario management system in place on the server. A user logs into the system, and depending on the role can either prescribe a scenario through the domain expert client, or choose to use one of the scenarios through the end-user client. A number of interfaces in Figure 2.5 govern this: Login, GetUsers, GetScenarios etc.

The server implementation uses a flat-file database system (where data are stored in regular files on a hard drive, arranged categorically under a root directory). User login and privilege information are stored in a file containing a serialized java Map<Username, Password> [23], where the passwords are md5 hashed for security. Upon login request, the file is deserialized into the original map, and credentials can be checked.

The domain experts have privileges to view a list of users, and prescribe scenarios for any of them. When a new scenario is posted, the server receives the untrained scenario file and saves it in target user's directory. End-users only have privileges to view a list of scenarios linked to them, and use the scenarios.

3.2.3 Core Classification Components

Both a context classification committee and an activity classifier make up the core system components. These components each have their own section below.

3.2.4 List of Messages

Table 3.1 shows the list of messages implemented, their functions and their responses (also refer to Figure 2.5).

3.3 Context Detection

Context detection is a major part of the work conducted. Using the modular nature of the system, we surveyed a number of classifiers. During this process, we discovered that certain features are not suitable for certain classifiers. This section starts with a survey of classifiers we implemented, then discusses the impact of features on classifier selection, and finally describes a committee based approach that combines a number of classifiers in a personalized manner for realizing context classification.

Table 3.1: Messages implemented

Request	Privilege	Response	Description
Login	User, Expert	OK, Exception	Logs in the user, a dedicated server thread will be created to handle this user's requests. If login unsuccessful, Exception is raised
GetUsers	Expert	Users, Exception	Returns a list of users
GetScenarios	User, Expert	Scenarios, Exception	Returns a list of scenarios Experts can assign a username on the message and that user's scenarios are returned
PostScenario	Expert	OK, Exception	Adds a scenario for a user
RemoveScenario	User, Expert	OK, Exception	Removes a scenario from a user
TrainContext	User	OK, Exception	Trains the context classifier committee with the data posted
TrainActivity	User	OK, Exception	Trains the activity classifier with the data posted
InertialDataMessage	User	ClassificationResult, Exception	Returns a classification based on inertial data
ContextDataMessage	User	ClassificationResult, Exception	Returns a classification based on context data

3.3.1 KNN

The k-nearest-neighbors (kNN) classifier is an instance based learner [24, 25, 26]. It is a lazy learner in that no real work is done when the training sequence is given during the training phase, they are simply stored by the classifier. When an unknown class is encountered, the classifier looks for the k nearest training samples to the unknown class, and a decision is made based on majority vote.

Other than implementation simplicity, another major advantage of kNN is the ability to handle nominal data (discontinuous). This is particularly important for data types like wireless SSID, as we will see in Section 3.3.6. To find the nearest training sample, we need to define a distance function. As there are timestamps and a set of SSIDs and signal strengths per context, we propose a set of custom distance function below.

$$\boldsymbol{t}_{time} - training\ set, pairs\ of\ time\ t, label\ s$$

$$s_{time}(x) - label\ of\ time\ x$$

$$\boldsymbol{t}_{wifi} - training\ set\ containing\ all\ SSIDs$$

$$\boldsymbol{t}_{wifi}(s) - training\ set\ containing\ all\ SSIDs\ given\ context\ s$$

$$\boldsymbol{z}_{wifi}(x, s) - signal\ strength\ of\ SSID\ x, under\ label\ s$$

$$y_{time}, \boldsymbol{y}_{wifi} - observation$$

$$\delta_{time} = \arg\min_{s_{time}(t)} \|t - y_{time}\| \; \forall \; t \in \boldsymbol{t}_{time} \tag{1}$$

$$\boldsymbol{b}_{wifi} = \{\boldsymbol{t}_{wifi} \cap \boldsymbol{y}_{wifi}\} \; - \; set\ of\ common\ SSIDs\ across\ all\ labels \tag{2}$$

$$let\ d(s) = \begin{cases} \min(\|\boldsymbol{z}_{wifi}(b,s) - \boldsymbol{z}_{wifi}(b,y)\| \; \forall \; b \in \; \boldsymbol{b}_{wifi}), & if\ b \in \boldsymbol{t}_{wifi}(s) \\ \infty, & else \end{cases}$$

(3)

$$\delta_{wifi} = \arg\min_{s} d(s) \tag{4}$$

Eq. (1) is a kNN classifier that does matching on time. The distance function here is just a check to see which label has the closest matching time. Eq. (2) first finds a set of common wireless SSIDs between all training data and the unknown class. The class with an SSID whose signal strength is closest to the unknown class is then used as output. In the case of k nearest neighbor, both Eq. (1) and Eq. (4) pick the top k, and a majority vote is performed.

3.3.2 AdaBoost

AdaBoost is a class of boosting method [27, 28, 29]. It is a meta learner, meaning that it is to be used in conjunction with a base learner. The base learner can be any classifier, and is usually straightforward to implement. They can also be very weak: in the binary case, a base learner needs only to outperform chance (50%). By forming multiple weak classifiers and weighting them on their accuracy, AdaBoost can combine the ensemble into a strong classifier. There are a large number of literatures describing the operation and algorithm of a number of AdaBoost variants [27, 28, 29].

For this study, we used the AdaBoost.MH algorithm with a decision stump base learner. AdaBoost.MH is an earlier variant, and one of the most popular [30]. It is an extension of the earliest multiclass AdaBoost.M1 algorithm [27]. A listing of the AdaBoost.MH algorithm can be found in [31]. The decision stump is a modified binary decision stump that handles multiple classes by creating a binary label set, and converting a multi class problem into a binary problem that tests to see if an unknown is of class A, or another class. The implementation simply traverses along each feature space, and picks the point where the dataset can be separated most distinctly. Figure 3.2 demonstrates this using two features.

Each axis on the figure represents a feature of the data, and by plotting feature 2 against feature 1 we obtain a 2D space from which the decision stump algorithm can find the decision region shown in shaded colors. The AdaBoost algorithm runs a large number of these, each time putting more weight on the misclassified data to force the decision stump to adjust its decision regions. The collection of stumps is then summed in a weighted way to form the final classifier.

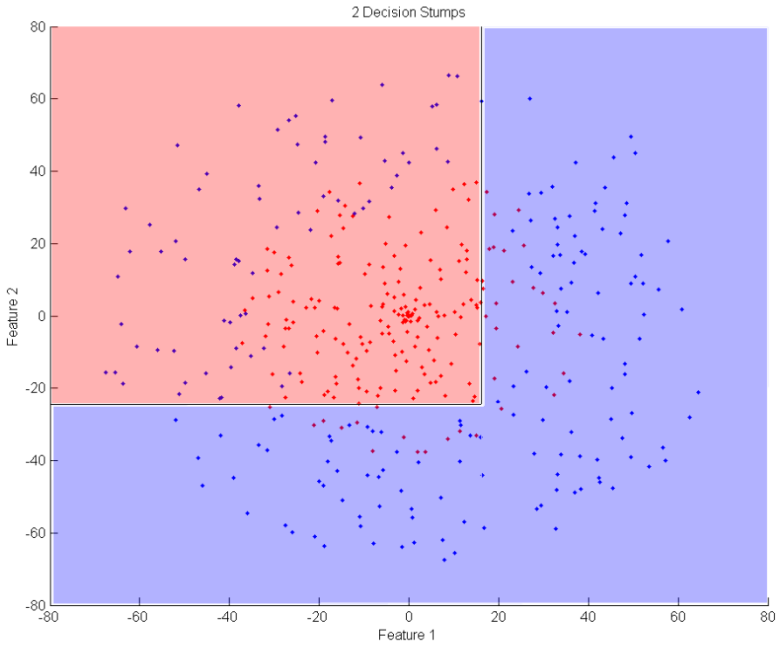


Figure 3.2: Decision stump on binary data

3.3.3 SVM

Support vector machine (SVM) is another popular machine learning algorithm. As a classifier, it finds the support vectors of a training dataset from the training process, and from these support vectors a cut is found that produces the maximum separation of classes on both sides.

Figure 3.3 demonstrates this for a binary case. The support vectors are circled, and the cut (black line) produces the maximum separation to all the support vectors.

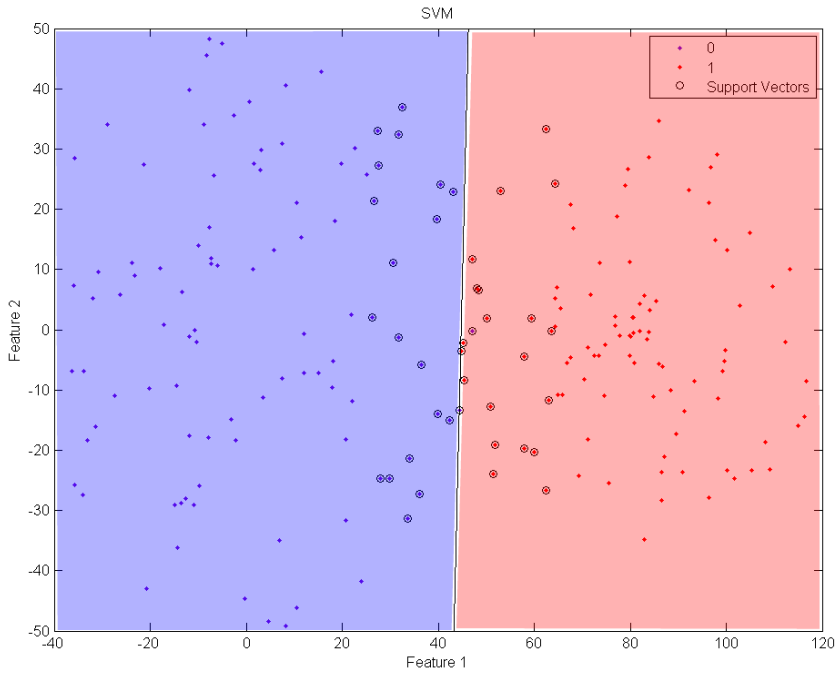


Figure 3.3: SVM on binary data

3.3.4 Decision Table Tree

Decision Table Tree (DTT) is model that we came up with by observing the datasets collected. We observed that wireless information is a strong feature for determining location compared to time and sound. As a result, we designed the tree to first attempt to make a classification based on wireless information, and only fall back to time if the information is not available. Figure 3.4 shows the DTT model.

During training, the DTT model stores all the data in a database. During classification, the DTT first check wireless information using the same distance function as Eq. (4), and the closest result is returned as the classification. However if no wireless

information can be matched, DTT then tries to see if the timestamp overlaps with any time period on record, and return the label as classification. If this also fails, then the DTT returns a label whose time is closest to the unknown timestamp, using Eq. (1).

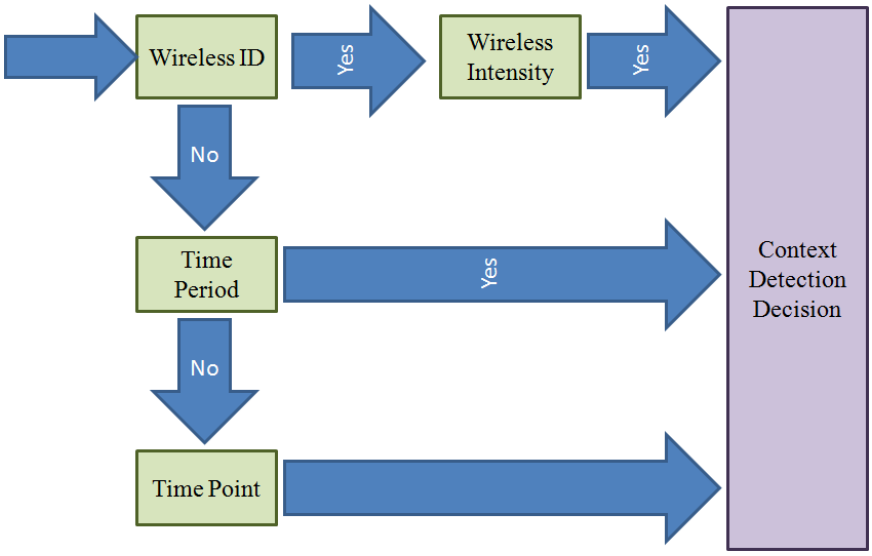


Figure 3.4: DTT model

3.3.5 Artificial Neural Networks

Finally, we also implemented an Artificial Neural Network (ANN). A neural network involves a network of simple processing elements called neurons [32]. These elements can arrange themselves to model a complex behavior, determined by the way connections are made between the processing elements, and the weighting on each connection. A neural network can also be interpreted as an adaptive machine [33], in which it is a massively parallel distributed processor made up of simple processing units. These then have a natural ability for storing experiential knowledge and making it available for use.

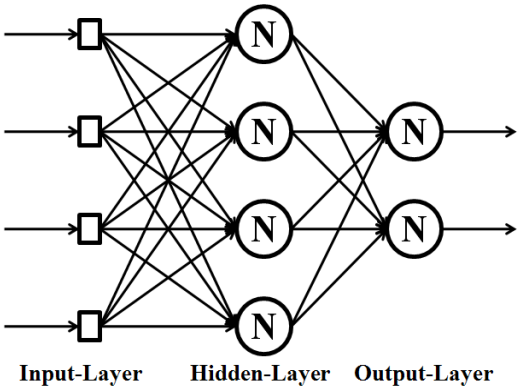


Figure 3.5: ANN with 3 layers

Figure 3.5 shows a 3 layers artificial neural network, with input layer, hidden layer and output layer. There are 4 neurons in the input layer, 4 in the hidden layer, and 2 in the output layer. Each connection has a weight attached, and the output of a layer becomes the input of the next layer. For training, an error back-propagation algorithm is one of the most popular, this algorithm contains a forward path and a backward path through all the layers. The forward path describes the connections and weights of a layer, and is responsible for computing the outputs of that layer. The backward path is used for adjusting the weights and biases for each neuron in the network, based on final output of the current iteration.

3.3.6 Features and Classifier Choice

The classifiers AdaBoost, SVM and ANN work by finding a way to divide the feature space into partitions belonging to different classes. It follows then that they only work well on data that has separable feature space. Classifiers like KNN and DTT work as

long as a custom set of distance functions or rules can be created, making them suitable for some data types, as we will present now.

In our work, we identified time of day, wireless SSID, signal strength, and sound as possible features for identifying context. It is easy to see that two wireless SSIDs are not related to each other, and it is difficult to represent them on a feature space suitable for classifiers. For example, consider the two sets of SSIDs below:

Set 1: [30:46:9a:06:4d:e0, 00:0c:41:6e:1e:f6, 2e:25:b3:96:d5:f9]

Set 2: [00:11:95:4c:7b:57, 00:26:bb:76:b3:85, 62:2a:19:50:35:79]

It is difficult, if not impossible to tell how to plot these on a feature space (such as a 2D space shown in Figure 3.3). We note that this type of data lends itself well for kNN and our own rule based DTT, as we can develop custom rules and distance functions based on sets.

Time and sound features on the other hand are continuous and separable in feature space. This means that we can use classifiers like SVM and AdaBoost without problems.

Based on the discussion of features and the choice of classifiers, there is a need to use separate classifiers to determine context based on different features. In this implementation, the committee is made up of 3 classifiers: kNN (k-nearest neighbors) with time as a feature; kNN with wireless MAC address and signal strength as features;

and AdaBoost with audio peak frequency, peak energy, average power and total energy as features (Table 3.2). These features are extracted from raw sensor data through a java program implementing the ContextFeatureExtractor interface: time is taken from the clock; wireless MAC address and signal strength list is obtained from periodic scans; and audio features are obtained by taking periodic recodings 10 seconds in length, and performing fast Fourier transform.

Table 3.2: Classifiers and features used for committee

Classifier	**Features**
kNN with time	• Timestamp
kNN with wireless	• SSID • Signal strength
AdaBoost	• Peak frequency • Peak energy • Average power • Total energy

In Section 2.5, we proposed a classification committee consisting of individual classifiers. Figure 3.6 shows the particular construction used in this implementation.

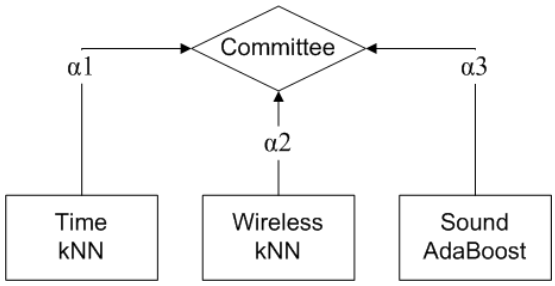


Figure 3.6: Classifier committee

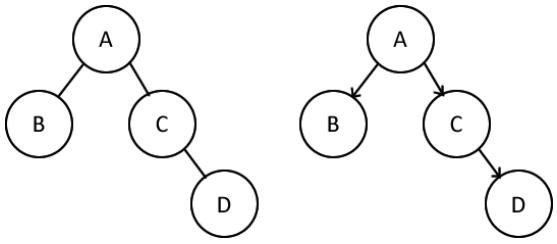
3.4 Activity Classification - Bayesian Networks

This section first introduces all the concepts required for a discussion on our classifier. Sub section 3.4.6 then ties together all the concepts, and presents a coherent formulation for a classification system. Throughout this section, full proofs are given for all theories discussed for completeness.

3.4.1 Graphical Models and Bayesian Networks

3.4.1.1 Theory

A graphical model is a probabilistic device that embeds conditional relationships of random variables. In such a graph (G), random variables are represented by nodes (V), edges (E) represent a relationship between the two variables, and the graph can be written as $G(V, E)$. Figure 3.7.a shows an example of one such graph.



a. Generic graph b. Bayesian network

Figure 3.7: Graph models

For a graph to be a Bayesian Network (BN), the edges (E) must be directed (meaning each edge has a direction which indicates the starting and ending node), and the graph must also satisfy the Markov condition in Definition 4.

Definition 3*:* Given a graph $G(V, E)$ and nodes X, Y. X is the parent of Y if there is an edge from $X \rightarrow Y$, and X is an ancestor of Y if there is a path from $X \rightarrow Y$ (with length > 1, i.e. not an edge). Y is a descendent of X if there is a path from $X \rightarrow Y$ (taking note of the difference between an edge and a path).

Definition 4: Given a graph $G(V, E)$ and a joint probability distribution P of the set $\{V\}$, the graph G satisfies the Markov condition if:

$$P(X \mid A_X, N_X) = P(X \mid A_X), \forall X \in V$$

Where A_X is the parent set of X, and N_X is the non-descendent set. That is, the probability of node X given its parents is conditionally independent to variables not linked to X.

Following the Markov condition we arrive at an important theorem:

Theorem 1*:* If a graph G satisfies the Markov condition, then the joint probability P embedded in the BN is given by the product of all conditional probabilities following Definition 4.

Proof*:* First order the nodes so that if Y is a descendent of X, then Y follows X in the ordering (ordering being the number given to each node, or the order we visit the nodes). E.g. Figure 3.8 can have an ordering $[A_1 \dots A_n]$.

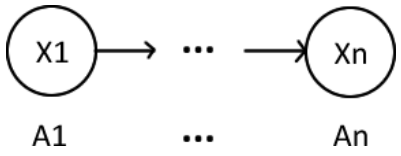


Figure 3.8: Example Bayesian network

From here, the proof is given by induction.

Basis step: Assume A_1 is empty (root of the tree has no parent set).

$$P(X_1) = P(X_1 | A_1)$$

Induction step: Suppose that for $P(X_n \dots . X_1)$ the theorem holds:

$$P(X_N \dots X_1) = P(X_N \mid A_N) \dots P(X_1 | A_1)$$

We show that:

$$P(X_{N+1} \ldots X_1) = P(X_{N+1}|A_{N+1})P(X_N \ldots X_1)$$

First, if any of the $P(X_i|A_i) = 0$, the last equation is 0.

Next, for $P(X_i|A_i) \neq 0$, we have

$$\begin{aligned} P\left(X_{N+1}, X_N \ldots X_1\right) &= P\left(X_{N+1}|X_N \ldots X_1\right)P(X_N \ldots P_1) \\ &= P(X_{N+1}\,|\,A_{N+1})P(X_N \ldots P_1) \\ &= P(X_{N+1}\,|\,A_{N+1})P(X_N\,|\,A_N)\ldots P(X_1\,|\,A_1) \end{aligned}$$

Where line two follows from Definition 4 and line 3 follows from the induction step.

Figure 3.9 demonstrates a canonical example of the Markov condition being satisfied.

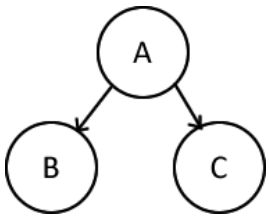


Figure 3.9 Another example of Bayesian network

Markov condition requires that $B \perp C\,|\,A$.

From the graphical model, we have $P(A,B,C) = P(B\,|\,A)P(C\,|\,A)P(A)$

From probability theory, we should have $P(A,B,C) = P(B,C\,|\,A)P(A)$

Combining the above two, we have $P(B,C \mid A) = P(B \mid A)P(C \mid A)$, so $B \perp C \mid A$ and the graph is a BN.

3.4.1.2 Benefits

Using a Bayesian network has a number of benefits. First, it is easy to visualize a graph, and from it the conditional independences between variables. The reverse is also true, where given a set of variables, we can identify the relationships between them, and from there intuitively construct a graphical model that is backed by probabilistic machinery for various computations. Second, a BN is an economical way of representing a complex joint probability distribution involving many variables. Suppose we have variables A, B and C constructed as shown in Figure 3.9. If each variable has L values, then by general probability theory $P(A, B, C)$ requires L^3 space for a lookup table (one L per dimension for each variable). Utilizing the Markov condition, variables B and C have 1 parent each ($m = 1$), and the required space is given by $L_1 \times m \times L_{2\ or\ 3} = L^{m+1}$. Assuming $L_1 = L_2 = L_3$ then in general the space required is nL^{m+1} for the entire graph, where n is the number of nodes in a graph. This is much less than L^n from general probability theory, if the number of parents per node can be restricted (thus small m).

3.4.2 Pearl's Message Passing Algorithm

Pearl's message passing algorithm (MPA) originally appeared in the paper by Judea Pearl in 1982 [34]. When operating on a *tree structured* Bayesian network, it

determines all probability $P(X|\boldsymbol{E})$ exactly, given a set of instantiated nodes $\boldsymbol{E}$ (random variables whose value we have seen).

3.4.2.1 Algorithm

The algorithm can be summarized as follows:

Algorithm 1: Peal's message passing algorithm

Input: A Bayesian network whose model is a tree, and a set of evidence ($\boldsymbol{E}$)
Output: $P(X|\boldsymbol{E})\ \forall\ X \in \boldsymbol{V}$

Initialize:
for $X \in \boldsymbol{V}$
$\lambda(x) = 1$
for each parent (Z) of X
$\lambda_{X\to Z}(z) = 1$
for value r of root R
$P(r|\boldsymbol{e}) = P(r)$
$\pi(r) = P(r)$
for each child C of R
send_π_message($R \to C$)

send_π_message($Z \to X$):
$\pi_{Z\to X}(z) = \pi(z) \prod_{Y\in N_Z\{X\}} \lambda_{Y\to Z}(z)$
$\pi(x) = \sum_z P(x|z)\pi_{Z\to X}(z)$
$\varphi(x|\boldsymbol{e}) = \lambda(x)\pi(x)$
$P(x|\boldsymbol{e}) = normalize(\varphi)$
for each child C of X, $C \notin \boldsymbol{E}$
send_π_message($X \to C$)

send_λ_message($Y \to X$):
$\lambda_{Y\to X} = \sum_y P(y|x)\lambda(y)$
$\lambda(x) = \prod_{U\in N_X} \lambda_{U\to X}(x)$
$\varphi(x|\boldsymbol{e}) = \lambda(x)\pi(x)$
$P(x|\boldsymbol{e}) = normalize(\varphi)$
if X is not root, and parent Z of X is not in **E**

send_λ_message($X \rightarrow Z$)
for each child C of X, $X \neq Y, X \notin \boldsymbol{E}$
send_π_message($X \rightarrow C$)

update_evidence:
for each node $N \in \boldsymbol{E}$
$\lambda(\hat{n}) = 1, \pi(\hat{n}) = 1, P(\hat{n}|\boldsymbol{e}) = 1, set\ all\ else\ 0$
if N is not root and the parent Z of N not in $\boldsymbol{E}$
send_ λ_message($N \rightarrow Z$)
for each child C of N, $C \notin \boldsymbol{E}$
send_π_message($N \rightarrow C$)

Proof:

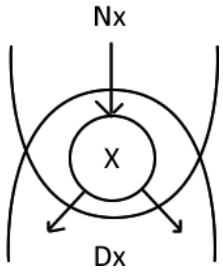


Figure 3.10: Division of graph at node X

Consider Figure 3.10, let $\boldsymbol{D_x}$ be the subset of $\boldsymbol{E}$ containing all members of $\boldsymbol{E}$ that are in the subtree rooted X ($X \in D_X$ if $X \in \boldsymbol{E}$). Let $\boldsymbol{N_X}$ be the subset of $\boldsymbol{E}$ containing non-descendents of X (note that X is a non-descendent of X, and $X \in N_X$ if $X \in \boldsymbol{E}$).

We then have

$$
\begin{aligned}
P(x \mid \boldsymbol{e}) &= P(x \mid \boldsymbol{d_x}, \boldsymbol{n_x}) \\
&= \frac{P(\boldsymbol{d_x}, \boldsymbol{n_x} \mid x) P(x)}{P(\boldsymbol{d_x}, \boldsymbol{n_x})} \\
&= \frac{P(\boldsymbol{d_x} \mid x) P(\boldsymbol{n_x} \mid x) P(x)}{P(\boldsymbol{d_x}, \boldsymbol{n_x})} \\
&= \frac{P(\boldsymbol{d_x} \mid x) P(x \mid \boldsymbol{n_x}) P(\boldsymbol{n_x}) p(x)}{P(\boldsymbol{d_x}, \boldsymbol{n_x}) P(x)} \\
&= \beta P(\boldsymbol{d_x} \mid x) P(x \mid \boldsymbol{n_x})
\end{aligned}
$$

where the 3rd line comes from the fact that $\boldsymbol{d_x}$ and $\boldsymbol{n_x}$ are independent, and β at the end is a factor that contains all the terms not related to x.

From this, we define

$$
\begin{aligned}
\lambda(x) &\propto P(\boldsymbol{d_x} \mid x) \\
\pi(x) &\propto P(x \mid \boldsymbol{n_x})
\end{aligned}
$$

And so

$$
P(x \mid \boldsymbol{e}) \propto \lambda(x) \pi(x)
$$

Looking at $\lambda(x)$, we have:

1. $X \in \boldsymbol{E}$ *and is* $\hat{x}$

   This means that

$$
\lambda(x) \propto P(\boldsymbol{d_x} \mid x) = \begin{cases} 0, for\ x \neq \hat{x} \\ 1, for\ x = \hat{x} \end{cases}
$$

2. $X \notin \boldsymbol{E}$, and is a leaf

   We have

   $$\boldsymbol{d_x} = \{\emptyset\}$$

   $$\lambda(x) \propto P(\emptyset|x) = 1, for\ all\ x$$

3. $X \notin \boldsymbol{E}$, and is not a leaf

   We then consider the case where there are two children, let D_L be the left child and D_R be the right child. And since $X \notin \boldsymbol{E}$, $D_X = D_L \cup D_R$ (Figure 3.11).

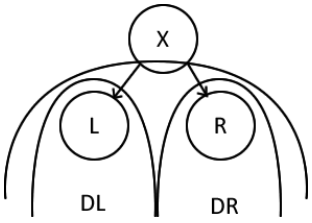


Figure 3.11: Division of subtree into left and right branches

This means that

$$\begin{aligned} P(\boldsymbol{d_x}|x) &= P(\boldsymbol{d_L}, \boldsymbol{d_R}|x) \\ &= P(\boldsymbol{d_L}|x)P(\boldsymbol{d_R}|x) \\ &= \sum_l P(l|x)P(\boldsymbol{d_L}|l)\sum_r P(r|x)P(\boldsymbol{d_R}|r) \\ &= \sum_l P(l|x)\lambda(l)\sum_r P(l|x)\lambda(r) \end{aligned}$$

Where the third line comes from:

$$\begin{aligned} P(\boldsymbol{d_L}, l|x) &= P(\boldsymbol{d_L}|l,x)P(l|x) \\ &= P(\boldsymbol{d_L}|l)P(l|x) \end{aligned}$$

Since the above depends on both l, r and x, we define messages

$$\lambda_{L\to X}(x) = \sum_{l} P(l \mid x)\lambda(l)$$

$$\lambda_{R\to X}(x) = \sum_{r} P(r \mid x)\lambda(r)$$

Looking at $\pi(x)$, we have:

1. $X \in \boldsymbol{E}\ and\ is\ \hat{x}$

   $$\pi(x) \propto P(x|\boldsymbol{n_X}) = \begin{cases} 0, for\ x \neq \hat{x} \\ 1, for\ x = \hat{x} \end{cases}$$

2. $X \notin \boldsymbol{E}$, and is the root

   We have

   $$\boldsymbol{n_X} = \{\emptyset\}$$

   $$P(x|\boldsymbol{n_X}) = P(x|\emptyset) = P(x)$$

3. $X \notin \boldsymbol{E}$, and is not the root

   We then consider X to be the left child of Z, and let T be the right child. $N_X = N_Z \cup D_T$ (the set of non-descendent is now the set of non-descendent of the parent Z, together with the set of descendent of T, since X is only connected to T through its parent).

$$
\begin{aligned}
P(x \mid \boldsymbol{n}_x) &= \sum_z x \mid \boldsymbol{n}_x, z) P(z \mid \boldsymbol{n}_x) \\
&= \sum_z P(x \mid z) P(z \mid \boldsymbol{n}_x) \\
&= \sum_z P(x \mid z) P(z \mid \boldsymbol{n}_z, \boldsymbol{d}_T) \\
&= \sum_z P(x \mid z) \frac{P(\boldsymbol{n}_z, \boldsymbol{d}_T \mid z) P(z)}{P(\boldsymbol{n}_z, \boldsymbol{d}_T)} \\
&= \sum_z P(x \mid z) \frac{\frac{P(z \mid \boldsymbol{n}_z) P(\boldsymbol{n}_z)}{P(z)} P(\boldsymbol{d}_T \mid z) P(z)}{P(\boldsymbol{n}_z, \boldsymbol{d}_T)} \\
&= \gamma \sum_z P(x \mid z) P(z \mid \boldsymbol{n}_z) P(\boldsymbol{d}_T \mid z) \\
&= \gamma \sum_z P(x \mid z) \pi(z) \lambda_{T \to Z}(z)
\end{aligned}
$$

Where the λ message in the last equality comes from

$$
\begin{aligned}
P(\boldsymbol{d}_T \mid z) &= \sum_t P(\boldsymbol{d}_T \mid z, t) P(t \mid z) \\
&= \sum_t P(\boldsymbol{d}_T \mid t) P(t \mid z) \\
&= \lambda_{T \to Z}(z)
\end{aligned}
$$

Since $P(x|\boldsymbol{n}_X)$ depends on values from T and Z, we define message

$$\pi_{Z \to X}(z) = \pi(z) \lambda_{T \to Z}(z)$$

And so

$$P(x \mid \boldsymbol{n}_x) \propto \pi(x) = \sum_z P(x \mid z) \pi_{Z \to X}(z)$$

3.4.2.2 Space and Time Complexity

Let

n = the number of nodes in the tree

l = number of values for a node

k = number of children for a node

The resulting tree would have $n - 1$ edges, and need to store at most l^2 values in a look up table for each pair of nodes (conditional variables), $2l$ values in two separate 1D arrays for the π and λ values, and so the space complexity is given by $O(cl^2)$, where c is a constant.

The number of multiplications required to compute a variable's conditional probability is l to compute the π messages and l^2 to compute the λ messages, l^2 to compute the π values and kl to compute the λ values, giving us a running time complexity of $O(cl^2)$.

3.4.3 Beta Density

Up until now, we have assumed that the conditional probabilities are already given. In classification problems, they often need to be learned from training data. When learning from data, we can either learn from only user training data, or from only expert opinion. Using the former, we obtain estimates of conditional probabilities specific for a user, ignoring any general trends that may be exhibited by the general population. And using the latter, we use trends exhibited by the general population,

and ignore specific input from the user. Here we look at ways to incorporate both our prior belief of a conditional probability (maybe from what we have observed in the general population), and a user's specific behavior (from the user's training data).

Beta density is a probability distribution. It allows us to start with a prior belief about the value of a random variable, and then update this belief with new datasets. To show some of its properties, we will provide a running example.

Suppose we have a binary data transmission channel that randomly assumes one of 100 states numbered $n = 1\ to\ 100$, each with $1/n$ chance of corrupting the transmitted bit. Let X be the outcome from a channel, and F be the random variable associated with n (such that $f = 1$ is associated with the 1st channel and so on). We then have:

$$P(X = corrupted\ |F = f) = f$$

That is, if we know which channel state we are in (thus which f), then the probability of receiving a corrupted bit is f.

Now consider that this is the very first time such a channel is used, then we are more likely to set F to uniform as there are no prior knowledge. Further consider the case where one such channel has been used widely, and we see that the channel's state is centered around $n = 10$ (that is $f = 0.1$). For both of these cases, we would want a different distribution for F, and the Beta density family allows us to do that.

Definition 5: A beta density describes a binary random variable *F* with parameters a and b, and has the form

$$beta(f;a,b)=\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}f^{(a-1)}(1-f)^{(b-1)},0\le f\le 1$$

Where $\Gamma(x)$ is given by:

$$\Gamma(x)=\int_0^\infty t^{(x-1)}e^{(-t)}.dt$$

Going back to the motivating example, note that $beta(f;1,1)$ is uniform, and $beta(f;5,45)$ is a bell curve centered around 0.1. Figure 6 shows both of these.

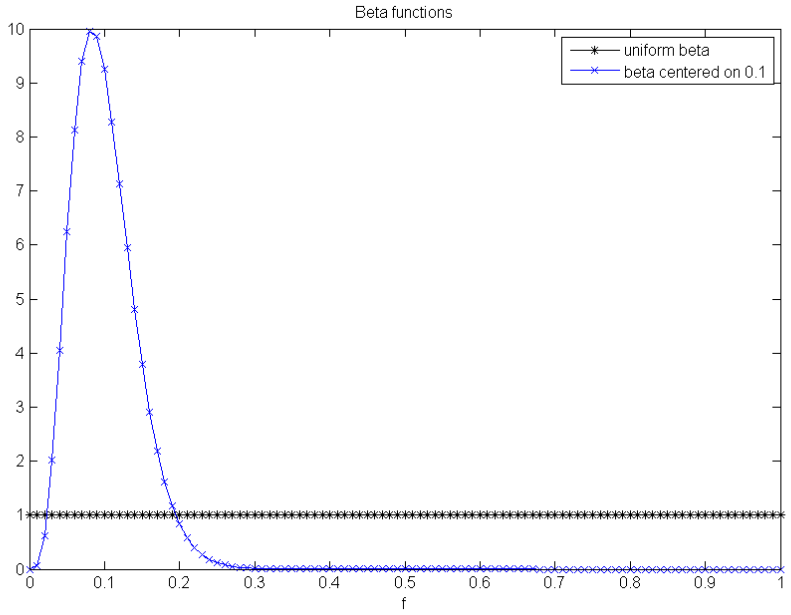


Figure 3.12: Uniform and non-uniform beta densities

Worth noting, is that the beta density centers around $\frac{a}{a+b}$, and has its sharpness associated with $a + b$. Intuitively, $\frac{a}{a+b}$ can be seen as the relative frequency of seeing a occurring in $a + b$ observations, and $a + b$ shows how many samples we have taken (thus how confident we are about the distribution of values around the center).

Now that the beta density is defined, we establish the relationship of F and the probability of error $P(X = 1)$:

Theorem 2*:* Suppose we believe that given beta density,

$$P(X = 1 \mid f) = f$$

Then

$$P(X = 1) = E[F]$$

Proof:

$$\begin{aligned} P(X = 1) &= \int_0^1 P(X = 1 \mid f)\, beta(f; a, b).df \\ &= \int_0^1 f \bullet beta(f; a, b).df \\ &= E[F] \end{aligned}$$

$E[F]$ can then be evaluated through the integral:

$$
\begin{aligned}
E[F] &= \int_0^1 f \bullet beta(f;a,b).df \\
&= \int_0^1 f \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} f^{(a-1)}(1-f)^{(b-1)}.df \\
&= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^1 f^{a}(1-f)^{(b-1)}.df \\
&= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(a+1)\Gamma(b)}{\Gamma(a+b-1+2)} \\
&= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{a\Gamma(a)\Gamma(b)}{(a+b)\Gamma(a+b)} \\
&= \frac{a}{a+b}
\end{aligned}
$$

Where the 2nd gamma function in line 4 comes from the definition of a beta function:

$$
\begin{aligned}
Beta(x,y) &= \int_0^1 t^{(x-1)}(1-t)^{(y-1)}.dt \\
&= \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}
\end{aligned}
$$

And the last line comes from the following result:

$$
\begin{aligned}
\Gamma(x+1) &= \int_0^\infty t^x e^{-t}.dt \\
&= [-t^x e^- t]_0^\infty + x\int_0^\infty t^{x-1} e^{-t}.dt \\
&= x\Gamma(x)
\end{aligned}
$$

The final result is that

$$P(X = 1) = E[F] = \frac{a}{a+b}$$

Intuitively, $E[F]$ can be thought as the prior belief of the probability of X, and we want to start with this prior belief, and update it if the new set of data disagrees. For example, if we believe the channel behaves with beta(f; 1,1), and after 20 transmission there is a 90% error rate, then it is likely this is the wrong estimate for the current channel, and our belief needs to be updated.

Theorem 3: Given a set of independent data $\boldsymbol{t} = \{\boldsymbol{x^1}, \dots, \boldsymbol{x^n}\}$, let $\boldsymbol{k}$ be the number of times $\boldsymbol{x^i} = \mathbf{1}$, let $\boldsymbol{l}$ be the number of times $\boldsymbol{x^i} = \mathbf{0}$. We can update the original beta distribution to

$$beta(f; a+k, b+l)$$

Proof:

$$P(t) = \int_0^1 P(t \mid f)\, beta(f; a, b).df$$

$$= \int_0^1 \prod_{i=1}^{N} P(x^i \mid f)\, beta(f; a, b).df$$

$$= \int_0^1 f^k (1-f)^l \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} f^{(a-1)} (1-f)^{(b-1)}.df$$

$$= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^1 f^{(a+k-1)} (1-f)^{(b+l-1)}.df$$

$$= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \frac{\Gamma(a+k)\Gamma(b+l)}{\Gamma(a+b+k+l)}$$

Where line 2 relies on the data samples being independent from each other. From this, we further obtain

$$beta(f;a,b|\boldsymbol{t}) = \frac{P(\boldsymbol{t}|f)beta(f;a,b)}{P(\boldsymbol{t})}$$

$$= \frac{f^k(1-f)^l beta(f;a,b)}{P(\boldsymbol{t})}$$

$$= \frac{f^k(1-f)^l \dfrac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} f^{(a-1)}(1-f)^{(b-1)}}{\dfrac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \dfrac{\Gamma(a+k)\Gamma(b+l)}{\Gamma(a+b+k+l)}}$$

$$= \frac{\Gamma(a+b+k+l)}{\Gamma(a+k)\Gamma(b+l)} f^{(a+k-1)}(1-f)^{(b+l-1)}$$

$$= beta(f;a+k,b+l)$$

Figure 3.13 shows the result of this update on the running example of binary data transmission.

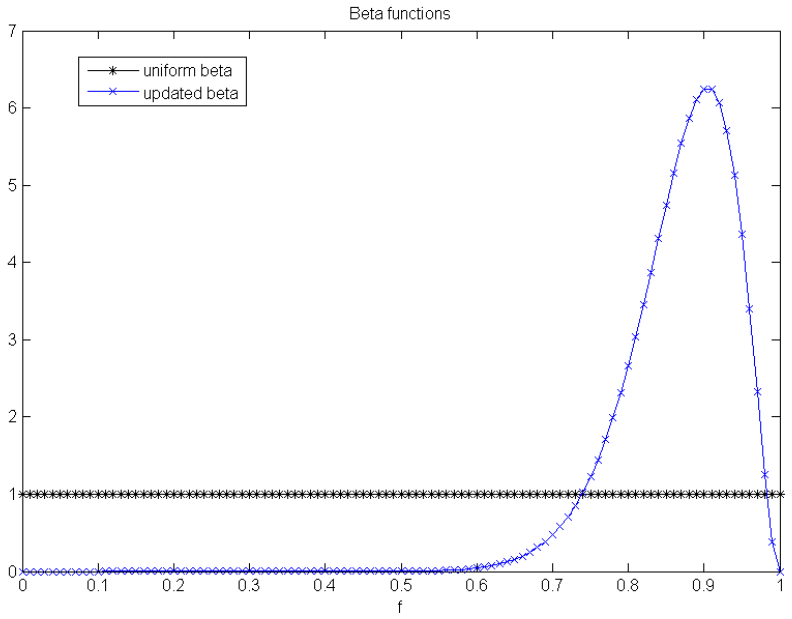


Figure 3.13: Initial uniform beta function and the updated result

3.4.4 Dirichlet Density

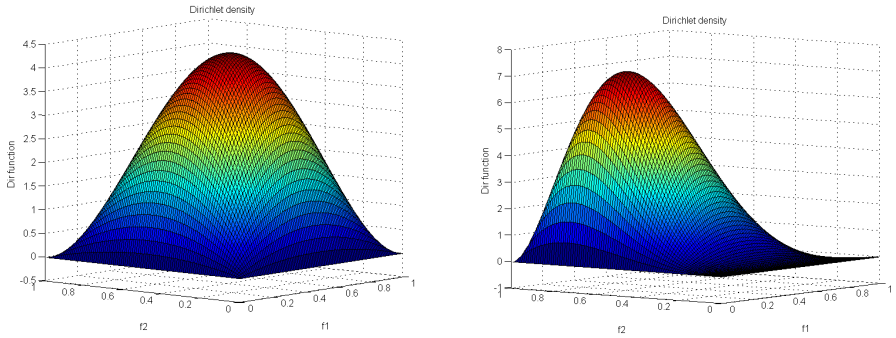
Whereas all the analysis and examples above only apply to binary variables, Dirichlet density applies to multi-nominal variables. The Dirichlet density is a multivariate generalization of the Beta density, and is studied here. For a variable that takes on r values, we have for the Dirichlet distribution with parameters $a_1, \ldots a_r$:

$$dir(f_1,\ldots f_r - 1; a_1,\ldots a_r) = \frac{\Gamma(\sum_i a_i)}{\prod_{i=1}^{r}\Gamma(a_i)} f_1^{(a_1-1)}\ldots f_r^{(a_r-1)}, 0 \le f_i \le 1, \sum_i f_i$$

Note that only $r-1$ number of f are needed, as f_r can be unique determined using:

$$f_r = 1 - \sum_{i=1}^{r-1} f_i$$

Figure 3.14 shows an example of a uniform Dirichlet (a) $Dir(f_1, f_2; 2,2,2)$ and one that tends towards f_2 (b) with $Dir(f_1, f_2; 2,4,2)$.



a. Uniform b.Dirichlet towards f2

Figure 3.14: Dirichlet functions

Following the beta density, Dirichlet density has similar properties regarding expected values and updates:

$$E[F_i] = \frac{a_i}{\sum_{k=1}^{r} a_k}$$

And if

$$P(X = i \mid f_i) = f_i$$

Then

$$P(X = i) = E[F_i]$$

Also

$$Dir(f_1,...,f_{r-1};a_1,...,a_r \mid \boldsymbol{t}) = Dir(f_1,...,f_{r-1};a_1 + l_1,...,a_r + l_r)$$

3.4.5 Augmented Bayesian Networks

An augmented Bayesian network is defined as [35]:

Definition 6: An augmented Bayesian network is a Bayesian network, with the addition of:

1. **For every node X_i in the graph, there is an auxiliary parent F_i and a density function P_{F_i}. Each auxiliary parent is a root, and must only contain an edge to the variable X_i.**

2. **For every node X_i, all values a_i of the parents A_i from the original graph, and f_i of F_i, there is a defined probability distribution of X_i conditioned on a_i and f_i**

For example, Figure 3.15 shows a two node Bayesian network (white) with its augmented construction (auxiliary shaded). F_1 is prior belief (beta density) for variable node X_1 (which has values 1 and 2), while $F_{2,1}$ is the prior belief of node X_2, given $X_1 = 1$. Similarly $F_{2,2}$ is the prior belief of X_2 given $X_1 = 2$.

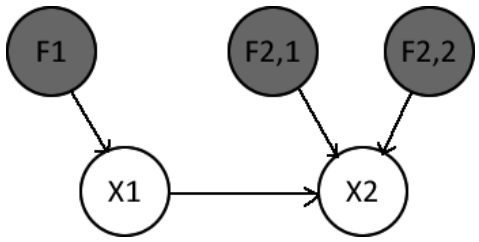


Figure 3.15: Augmented Bayesian network

If we let all the auxiliary node take on beta densities (initialized to uniform), then a list of all functions associated with this augmented BN is listed below (Table 3.3).

3.4.6 Bayesian Network Classifiers

We now have all the pieces required to develop a probabilistic classification system. There are a number of main components for a complete probabilistic classification system: 1) Features measured from raw data; 2) A set of class labels that can be separated by features; 3) A model describing the relationship between class labels and features; 4) A method for estimating the conditional probabilities from raw data; and 5)

An inference engine that queries a model for class label classifications, given an observation of features. All of the theories derived above will be used to implement these components.

Table 3.3: Beta densities associated with the augmented BN

Auxiliary nodes	Uniform beta densities $beta(f_1;1,1)$ $beta(f_{2,1};1,1)$ $beta(f_{2,2};1,1)$
Conditionals in the augmented BN	$P(X_1 = 1 \mid f_1) = f_1$ $P(X_2 = 1 \mid X = 1, f_{2,1}) = f_{2,1}$ $P(X_2 = 1 \mid X = 2, f_{2,2}) = f_{2,2}$
Conditionals in the underlying BN With auxiliary removed	$P(X_1 = 1) = 1/2$ $P(X_2 = 1 \mid X_1 = 1) = 1/2$ $P(X_2 = 1 \mid X_1 = 2) = 1/2$

Throughout this subsection, we use a running example of determining whether a Tennis game has been played, through a number of weather parameters. It is a common example, with raw data taken from the University of California, Irvine (UCI) Machine learning database [36]. The data is summarized in Table 2.

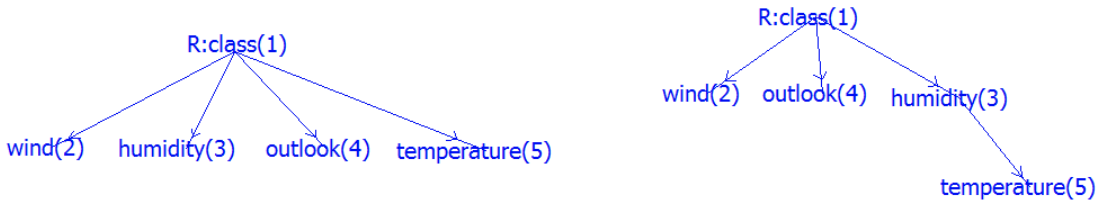
Table 3.4: Example data summary

		Play = Yes	**Play = No**
Outlook	Sunny	2	3
	Overcast	4	0
	Rain	3	2
Temperature	Hot	2	2
	Mild	4	2
	Cool	3	1
Humidity	High	3	4
	Normal	6	1
Wind	Strong	3	3
	Weak	6	2
Play		9	5

Components one and two are straightforward. First, features important for a topic of interest are determined by experts. Here for example weather data such as sunlight, wind strength etc are determined to be important for whether a tennis game goes ahead. Then, the class labels are identified such that the topic of interest is cleanly separated. For example whether the tennis game plays or not (yes, no are then the class labels).

The model we use to describe relationships between features and class labels is a Bayesian network. Each feature is a node on the graph, and the class label is also

represented by a node. The conditional relationship between class node and feature nodes are represented by directional arrows. If there are additional relationships between the features, then they can also be linked by directional arrows. One of the simplest models is the naive Bayes classifier, which assumes dependency between the class label and each of the features, but considers features conditionally independent to each other. Figure 3.16.a shows the running example represented by a naive Bayesian model, while (b) shows one more complex model which introduces dependency amongst the features.



a. Naive Bayes model b. Tree like structure

Figure 3.16: Bayesian models

Once we have a model, dependencies are set by edges and their directions. From this, we can go back to the raw data, and estimate these necessary conditional probabilities. Section 3.4.4 provides a way for us to incorporate both prior beliefs and user training data. Our belief about an individual's behavior can be obtained from a general population, and a generic Dirichlet density can be formed. This prior belief can then be updated through an augmented Bayesian network once we have seen training datasets from the user. In this running example, we may have the match playing and

weather statistics from tennis games across the country. This can serve as a prior belief, if there is location specific data to a particular tennis arena, then we can update our prior belief. Figure 3.17 shows an augmented BN of Figure 3.16.a, and Table 3.5 shows an example of this update by starting with uniform Dirichlet functions.

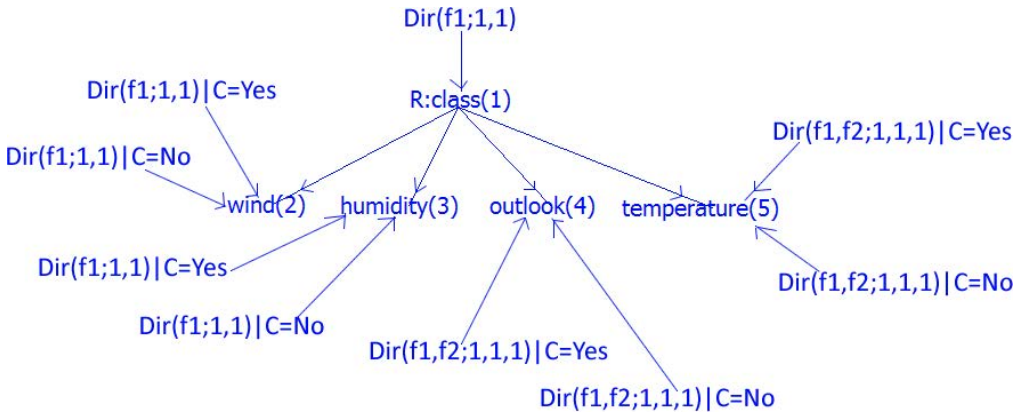


Figure 3.17: Augmented Bayesian network of Figure 3.16.a

Finally, given a model and required conditional probabilities, Pearl's message passing algorithm presented in Section 3.4.2 allows us to infer the probability of our class label being one of its values, given all the observed features. In our example, suppose we observed evidences $\boldsymbol{e}$:

$$\boldsymbol{e} = \{Outlook = Sunny, Temperature = Cool, Humidity = High, Wind = Strong\}$$

then the class probabilities given by Pearl's MPA are

$P(Play = Yes|\boldsymbol{e}) = 0.280$, or 0.205 without using augmented BN (no Dirichlet density update)

$P(Play = No|\boldsymbol{e}) = 0.720$, or 0.795 without using augmented BN (no Dirichlet density update)

Table 3.5 Updated Dirichlet of Figure 3.17

Feature	**Updated Dirichlet**
Wind	$Dir(f_1; 4,7)\|Yes$ $Dir(f_1; 4,3)\|No$
Humidity	$Dir(f_1; 4,7)\|Yes$ $Dir(f_1; 5,2)\|No$
Outlook	$Dir(f_1, f_2; 3,5,4)\|Yes$ $Dir(f_1, f_2; 4,1,3)\|No$
Temperature	$Dir(f_1, f_2; 3,5,4)\|Yes$ $Dir(f_1, f_2; 3,3,2)\|No$
Play	$Dir(f_1; 10,6)$

We see that the results above are consistent with those obtained through probability theory:

$$P(Yes|\boldsymbol{e}) = P(Sunny|Yes)P(High|Yes)P(Cool|Yes)P(Strong|Yes) = 0.0035$$

$$P(No|\boldsymbol{e}) = P(Sunny|No)P(High|No)P(Cool|No)P(Strong|No) = 0.0206$$

After normalization:

$$P(Yes \mid \boldsymbol{e}) = 0.2046$$
$$P(No \mid \boldsymbol{e}) = 0.7954$$

3.4.7 Implementation

Based on theories explored above, we implemented an activity classification system that uses BN for modeling, augmented BN with Dirichlet densities for parameter learning, and Pearl's MPA for inference. By using BN, we enable users to visualize abstract features as nodes, and let them build complex structures modeling an inference problem, based on their domain expertise. By using Dirichlet densities and forming augmented BN for parameter learning, we further enable a user to input their domain expertise in the form of prior knowledge. Finally by using Pearl's MPA, we enable both speed and space efficient inference. The server side component (the main classifier) implementation is described here, and the implementation of the model maker is covered in Section 3.6.

3.4.7.1 Overview

The system's general workflow is described by Figure 3.18. When in training mode, the untrained XML model generated by a domain expert client (Section 3.6) is used to determine the parameters that need learning, and theses parameters are estimated from the training data. The fully trained BN model is then saved as an XML document. In live mode, a previously trained BN model is loaded, and observations are inserted into

the network as evidences. Carrying out Pearl's MPA will update the network to the correct probabilities, and classification can be made.

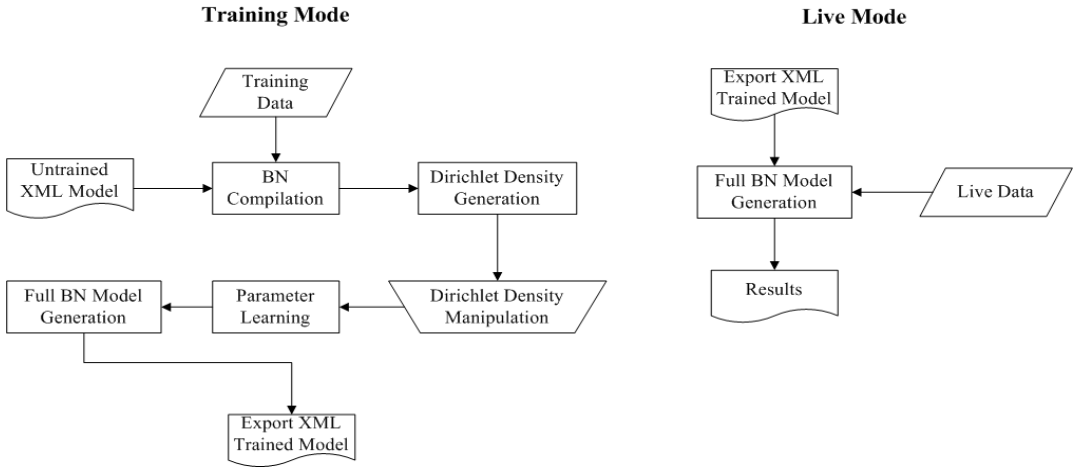


Figure 3.18: BN server side architecture

Figure 3.19 shows the interfaces model.

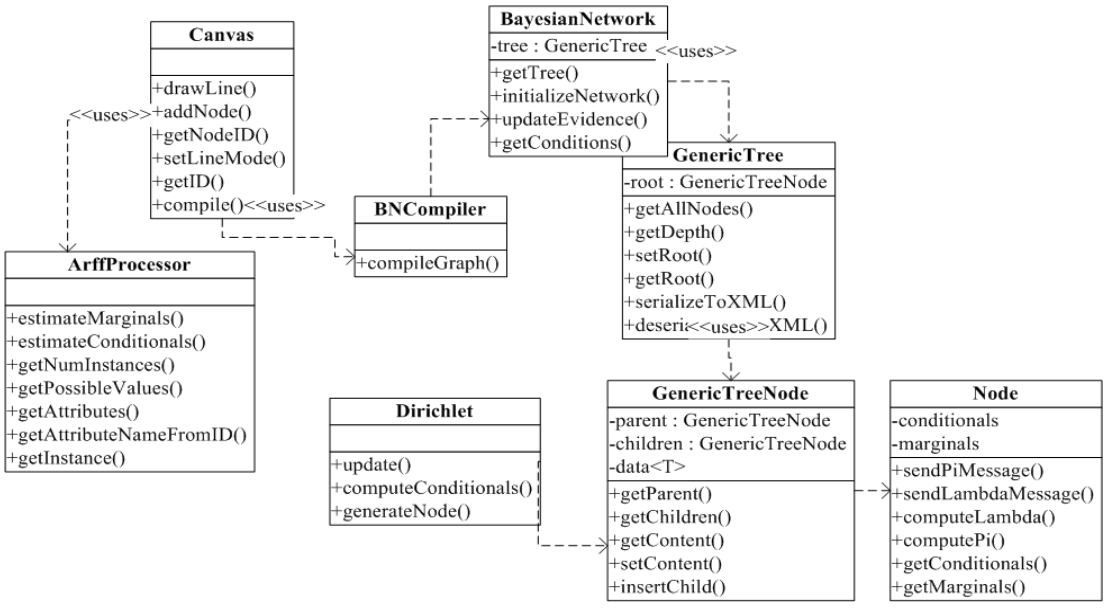


Figure 3.19: BN server side interfaces model

3.4.7.2 Parameter Estimation

When training is conducted, the program needs to finds all conditionals of the graph. This is done by carrying out a breadth first search (BFS) using algorithm listed in algorithm 2.

Algorithm 2: Breadth First Search (BFS)

```
Input: tree
Output: a list of nodes in the order of visit from left to right, then top to
bottom

Initialize queue q, List marked
q.add(root)
mark root as visited (add to marked)
while q is not empty
        x = q.leave
        for each child of x
                if child not marked (not in marked)
                        mark child
                        q.join(child)
return marked
```

Using these conditionals, the program then generates a list of uniform Dirichlet densities, and updates them based on theories developed in Section 3.4.4. First, the counts (l,k...) are determined from the raw data file received. Then, the densities are updated. Finally, actual conditionals of each feature node is determined through $\frac{a_i}{\sum_l a_l}$.

3.4.7.3 Inference

Once the training is complete, inference can be carried out in real time on the server. The end-user client would go into live mode, and stream sensor data to the server. Depending on the BN model prescribed, different features are derived from the data,

and Pearl's MPA is carried out by updating evidences (the observed feature values). Results are then streamed back to the client.

3.4.8 Extension into Activity Classification, Discretization

The extension of a generic BN structural-wise into activity classification is straightforward. There are two types of nodes, one is a class node containing activities of interest, and the other is a feature node, containing feature derived from sensor data. From there, the construction of a BN is no different to that discussed above.

In activity classification, almost all features are continuous. For example, the distribution of standard deviation of acceleration along any particular axis is usually considered Gaussian. This requires that a BN be constructed with continuous variables. The inclusion of continuous variables as feature nodes is non-trivial, and is the topic of this section.

While there are a number of theories extending Bayesian networks into the continuous domain [37,38], they are usually limited in the distributions that can be used (Gaussian, exponential), or the structure of the graph (e.g. must not have continuous parent with discrete child). In this work, we have decided to employ a discretizer, and this section describes our discretizer, and how to take into account a network's structure when discretizing.

3.4.8.1 Equal Distance Discretizer

An equal distance discretizer is the simplest discretizer, but also brings the most flexibility as it assumes no prior knowledge of the data or structure. The discretizer starts by dividing a continuous data set into n number of bins with boundaries $[b_1 \ldots b_{n-1}]$, and data points belonging to each bin is replaced by the bin number instead of their original continuous values. The two edge bins have range $(-\infty, b_1]$ and $[b_{n-1}, \infty)$ respectively. The result is a discrete variable whose value $v \in [1 \ldots n]$.

3.4.8.2 Enhancing the Discretizer Output

A significant drawback with using a discretizer is that if there was not enough data, then the output could contain empty bins. This could also occur if we have training data that was unfortunate enough to have no data in a bin's range, or if there are too many bins.

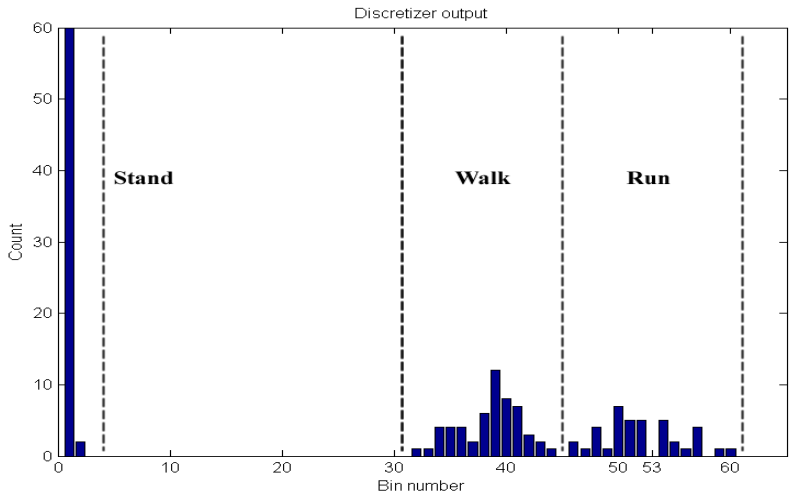


Figure 3.20: Discretizer output

As mentioned before, the distribution being discretized is the features. Figure 3.20 shows the result of discretization on the feature standard deviation in the horizontal axis, and this dataset contains the activities standing, walking and running. It is clear that the blue on the left in low bin number represents the samples for standing, as it generates the least amount of variance on acceleration. The next two roughly Gaussian shaped areas to the right are walking and running respectively.

It is also clear that bin 53 should contain samples relating to running, however there is nothing in the output for that bin. This is because the data set unfortunately does not contain any running sample that falls in that bin. When the BN is trained using this data, it assigns zero probability to bin 53, and any subsequent query on the network using that bin number produces a unknown result. From Figure 3.20, it can also be seen that bin 53 is not very close to the tail, so in actual live environments there is a fair chance that an observation of the feature would fall into that bin.

There are a number of solutions, the easiest of which is to increase training dataset size. The longer a training dataset is, the less likely that a probable bin (for example bin 53 in Figure 3.20) would be blank. However there is no upper bound on how long a training set has to be, and users are not likely to accept long periods of training (for example asking an end-user to run for 10+ minutes is often difficult).

We propose that by interpolating a zero bin with nearby bins, and filling it with generated data, we can get around the problem of empty bins without requiring extra training samples. Algorithm 3 and Figure 3.21 below demonstrate this idea.

Algorithm 3: Interpolation

Input: dataIn - discretizer output (array of integers, index is bin number, element value is count)
Output: dataOut - enhanced discretizer ouput

Copy dataIn to dataCopy

For each zero count bin in dataIn with bin number x
    Find nearest non-empty bin to the left, maximum 3 steps, assign steps taken to steps, assign count to l, assign samples to s
    Find nearest non-empty bin to the right, maximum 3 steps, add steps taken to steps, assign count to r, add samples to s
    new count for bin x is n = floor((l+r)/steps)

    For each unique sample class present in s
        generate m new samples of that class, m = ceiling(count(class)/size(s) * n)
    end for
end for

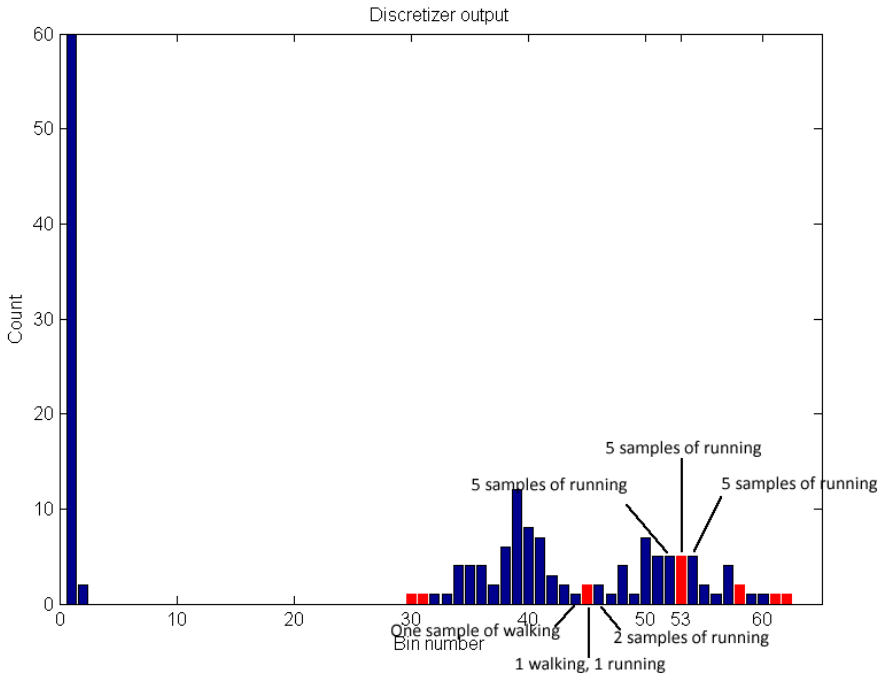


Figure 3.21: Enhanced discretizer output

Algorithm 3 requires that extra samples be generated, and this is also non-trivial in a BN, as it is highly dependent on the structure. Consider the four possible scenarios in a BN in Figure 3.22.

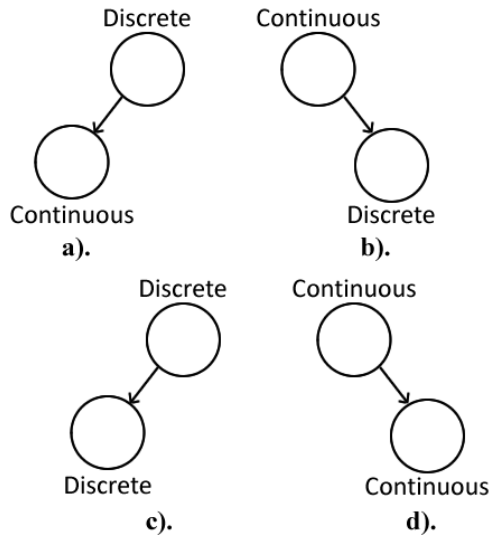


Figure 3.22: Possible node type combinations

In the case a discrete variable is the parent of a continuous variable (a), we generate samples that would fill the empty bin on the continuous variable, whose parent variable values are distributed in proportion to the distribution of neighboring bins (see Figure 3.21 for an example). In the case of a continuous variable followed by a continuous variable (d), both need to be interpolated. Similarly, the case of continuous variable followed by discrete variable (b) can be filled by following case (a), and the case of discrete variable followed by discrete variable does not need to be handled.

3.5 End-User Client

As per Section 2.4.1, the end-user client handles sensor instrumentation, scenario training and then real time classification.

3.5.1 Sensor Instrumentation

Both inertial data and context data need to be collected by the end-user client. Context data includes sound, time and wireless information, which can be obtained directly from an Android device and require no additional sensor instrumentation. This section deals with the instrumentation of external Bluetooth accelerometers for inertial data collection.

3.5.1.1 Hardware

GCDC X6-2mini accelerometers are used in this study. It is a 3-axis accelerometer with the following characteristics:

- ± 2/6g range
- 12/16 bit resolution
- 20 - 320Hz sample rate
- USB and SD card storage

They originally do not have Bluetooth logging capabilities, and so wireless capability is obtained by soldering a RN-42 Bluetooth module [39] onto the serial debugging port of the X6-2mini. The RN-42 chip supports RFCOMM communication, and the

X6-2mini serial port commands can be found in Appendix I. Figure 3.23 shows a picture of the modified accelerometer.

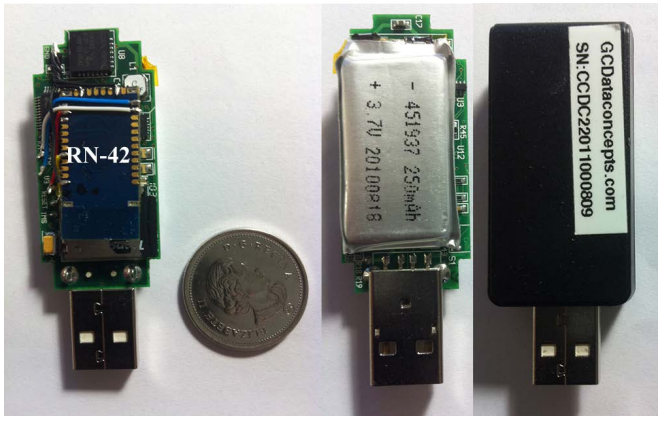


Figure 3.23: Modified X6-2mini

3.5.1.2 Software

Based on the debugging console commands from Appendix I, the implemented AirInterface outlined in Section 2.4.4 supports:

- Data logging
- Battery level check
- Bluetooth transmission on/off

Below is an example listing of the data being received, in the format of (relative time, x, y, z):

0.317,-1189,-3392,2130
0.342,-2745,-5547,1496
0.367,-2720,-6395,1245
0.391,-1886,-6210,1161

The device reports relative timestamp since the start of sensor. Synchronization is carried out by the controller, where a snapshot of all relative timestamps of every sensor is taken at a specific time (t), and subsequent data are tracked with the time t being zero reference. Sensor timing drift is avoided in the controller implementation by re-synchronizing every n seconds. Figure 3.24 is a flow chart depicting the data collection process.

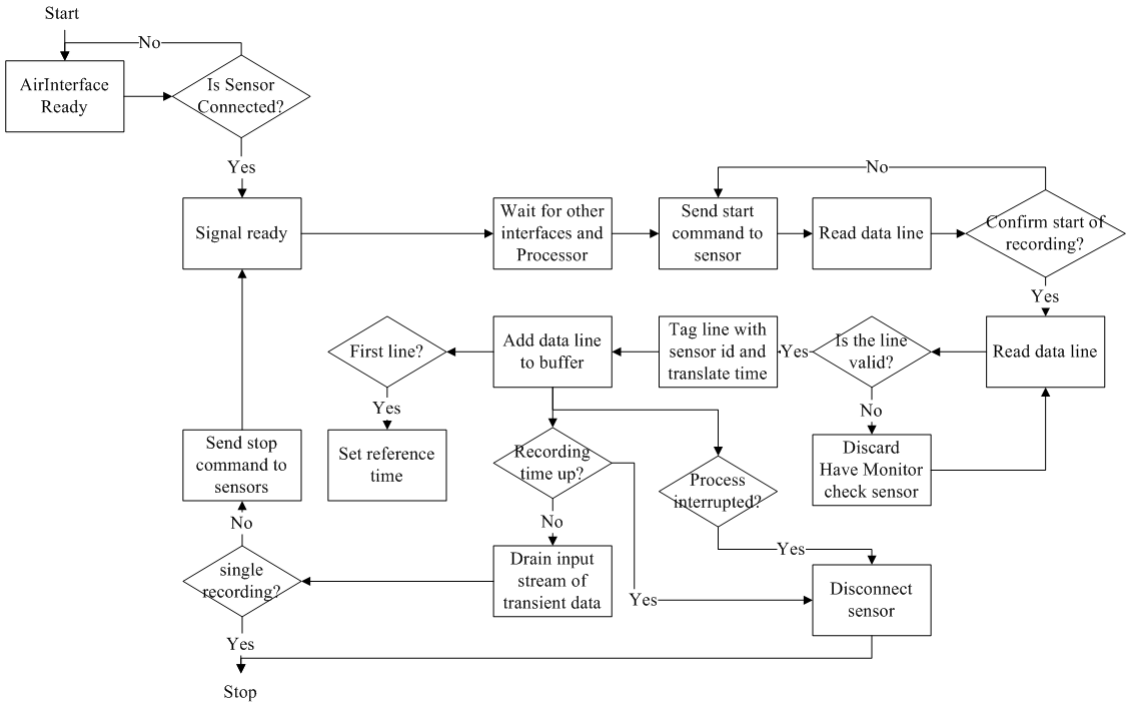


Figure 3.24: Sensor instrumentation system flow chart

Most parts of the process is self explanatory, worth noting are the "Signal ready" and "Wait for other interfaces and Processor" process. To synchronize the controller, multiple AirInterfaces and the processor, our implementation uses a cyclic barrier. A cyclic barrier is a multithreading control construct with the following behaviors:

1. Two operations are supported on a barrier: signal and wait
2. A cyclic barrier is initialized with a count
3. If a process signals the barrier, count decreases by 1
4. If a process waits at the barrier, it is allowed to proceed only after count becomes 0
5. Once all processes waiting at the barrier clears, the count is reset to initialized value (ready to be reused)

On the controller, there is a cyclic barrier initialized to the number of AirInterfaces + 2 (controller and processor). This allows all sub systems to periodically wait for each other to arrive at certain parts of the process before proceeding onwards again.

As individual sensors can send data at up to 320Hz (f), the data processing rate required to achieve buffer stability by the processor unit is:

$$R = 28fn$$

Where f is the sampling frequency of the sensors, n is the number of sensors, and R has unit bytes per second.

To provide as much headroom as possible, the processor implements a lookup table like data structure with insertion time $O(1)$, at the expense of memory footprint $O(nft)$, where n is the number of sensors, f is the sample frequency, and t is the number of seconds between re-sync. At every re-sync, the processor compiles the

table into InertialSensorData and notifies the upper layer with DataArrived. Table 3.6 shows an example constructed lookup table:

Table 3.6: Example lookup table

Time range	Sensor 1 Data String	Sensor 2 Data String
$0 - 1/f$	-1189,-3392,2130	-1886,-6210,1161
$2/f - 3/f$	-2745,-5547,1496	Missing
$4/f - 5/f$	-2720,-6395,1245	-1790,-4317,1308

3.5.1.3 Robustness

We observed a number of issues regarding sensor robustness in our experiments, and present some of them here. Where applicable, solutions are given.

The first problem discovered is that the Bluetooth transmission is not reliable and can deliver wrong control characters. This is especially problematic when the X6-2mini uses single character control commands, as a command character can actually be corrupted to another valid command. For example we have observed that d can be sent as D, which instead of starting the stream, stops it. This problem is dealt with on two levels. First, the AirInterface locks in a loop while trying to start a recording, this means that if a d was mistaken for something else, another d is sent. Second, the AirInterfaceMonitor tracks the incoming data stream, and senses when it has frozen. The AirInterface is not capable of sensing this as it would be blocked on a read

operation due to stream errors, and since no data is coming the read will block indefinitely. On sensing a frozen stream, the monitor restarts any recording that may have been stopped due to erroneous commands, and also unblocks the AirInterface by resetting the stream.

The second problem is corrupted and missing data. As each AirInterface is independent of each other and do not keep track of time (the rationale for a simplest possible design is presented in Section 2.4.4), they are not able to tell if a data line is missing or invalid. The processor however, can detect missing data by looking at empty slots on the buffer, and can detect bad data by doing a regular expression search on the string [40]. The regular expression required for a validation a data line is:

```
\d+.\d+,{1}([-+]?\d+,?){3}
```

Missing and corrupted data points are marked, and are interpolated by the processor. The interpolation method implemented is a simple repeat of the closest valid value.

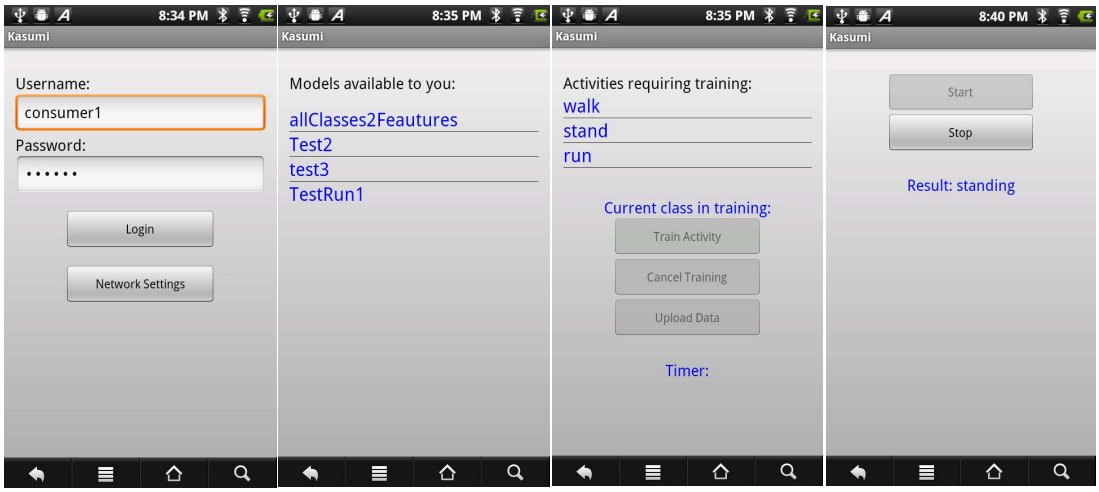
The third problem observed is that re-synchronization can cause brief periods of data loss. Consider a system with 3 sensors connected. If sensor 3 is lagging behind, then at the end of a re-sync cycle while we are waiting for sensor 3, the other sensors are not transmitting. This means that the data in this waiting period does not exist to the client. The problem does not affect our system, as the recording requirements can tolerate such a delay: 1) Training data requires a maximum of 5 minutes and Section 5.1.1 will demonstrate that the time drift is minimum at 5 minutes (thus no re-sync is needed,

circumventing the problem); and 2) Live mode does not require classification resolution up to the second scale, so a few seconds delay does not cause a problem. We would like to note however that this problem could potentially be addressed by using a second buffer to take over the extra data during the waiting time.

Last but not least, there is also the problem of sensors disconnecting. The monitor can detect this by catching socket errors on the stream, and attempt to re-establish connection. In the case of complete sensor failure, the upper layer is notified of disconnection after 3 attempts of reconnection.

3.5.2 Scenario Training

When a person starts the client, he/she is first presented by a login screen (Figure 3.25.a). Once logged in, the client pulls a list of available scenarios, and displays them to the user for selection (Figure 3.25.b). If the selected scenario is not trained, then the client determines which classes (activities) are present (for example running, walking), and guides the user through training each of the activities (Figure 3.25.c). For each activity, a three minute session is recorded, and the data is posted back to the server for training.



a. Login b. Scenario selection c. Training screen d. Live screen

Figure 3.25: End-user client

3.5.3 Live Mode

If the selected scenario has been trained before, then a user is prompted with the screen shown on Figure 3.25.d. Here the user only needs to press Start, and client will automatically instrument the sensors, and send data back to the server every 4 seconds. This means that a classification result would also be available every 4 seconds.

3.6 Domain Expert Client

The domain expert client is responsible for creating scenarios. Recall that a scenario is made up of a number of contexts and activities, each with its own model. A domain expert start by logging into the program as an expert (to gain the privilege of viewing users and creating scenarios), as shown in Figure 3.26.

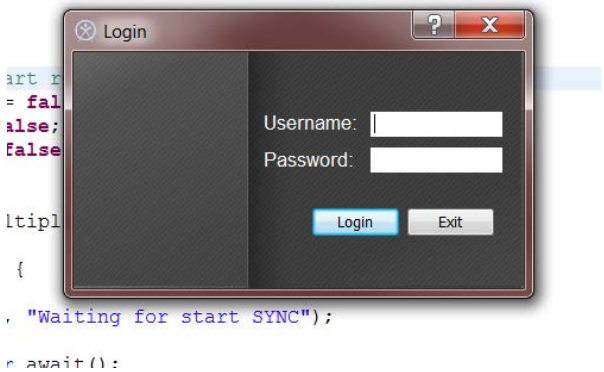


Figure 3.26: Login screen

The expert would then select a user (Figure 3.27), and add a scenario.

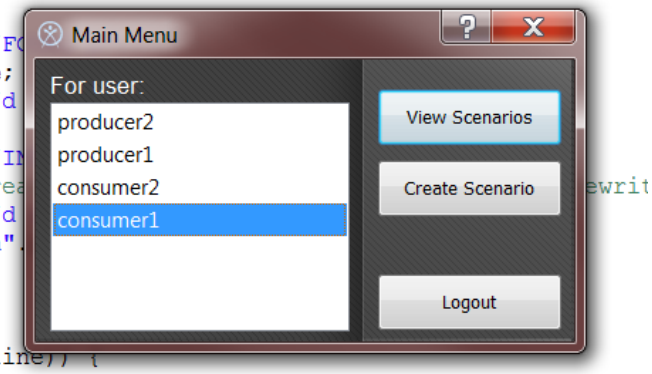


Figure 3.27: User selection

3.6.1 Context Model Generation

Context model generation is straightforward, an expert starts with the context name, followed by selecting a list of prebuilt context features, and finish by pressing the add context button (Figure 3.28).

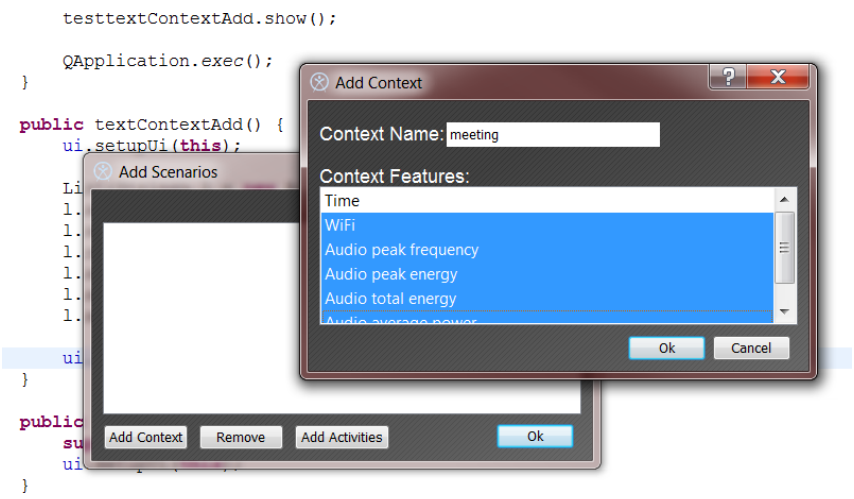
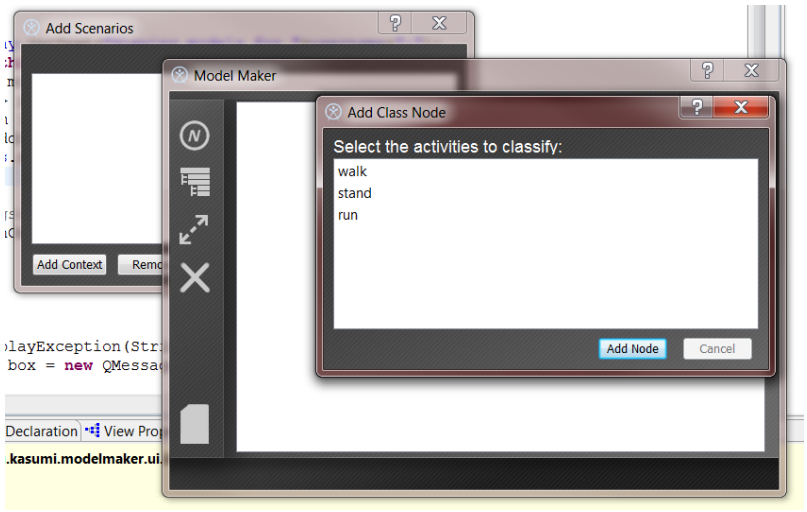


Figure 3.28: Adding context

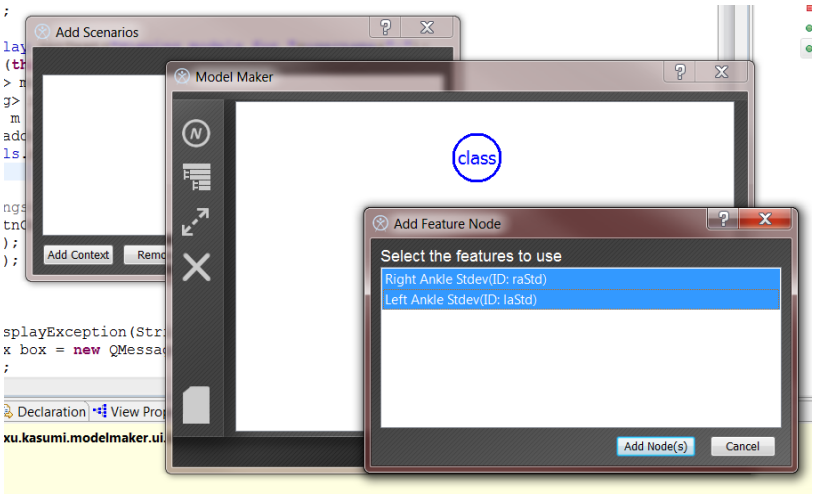
3.6.2 Activity Model Generation

This step requires an intuitive graphical interface that enables an expert to visualize the nodes (features nodes and the class node), and make logical connections that are later translated into conditions in the underlying BN. There is a prebuilt list of sensors and features available to the domain expert client, and an expert would be able to pick from this list.

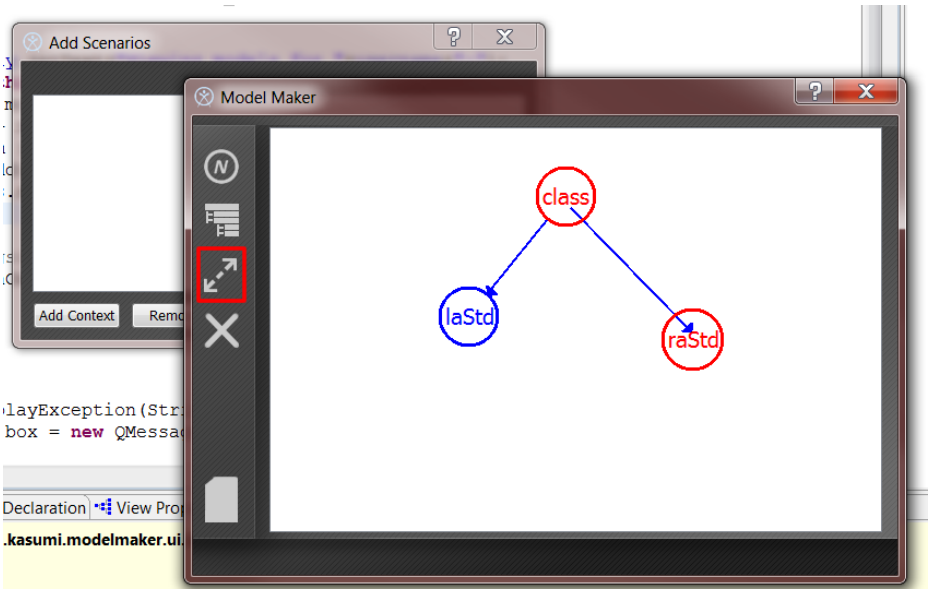
An expert starts by selecting what activities are of interest, and a root node is formed (Figure 3.29.a). From there, additional class nodes and feature nodes can be added (Figure 3.29.b), links can be drawn (Figure 3.29.c) and removed (Figure 3.29.d). Once finished, the model can be compiled and sent to the server (Figure 3.29.e). Because the object graph is constructed using object oriented techniques, it can be saved directly into a XML file through object serialization on the server.



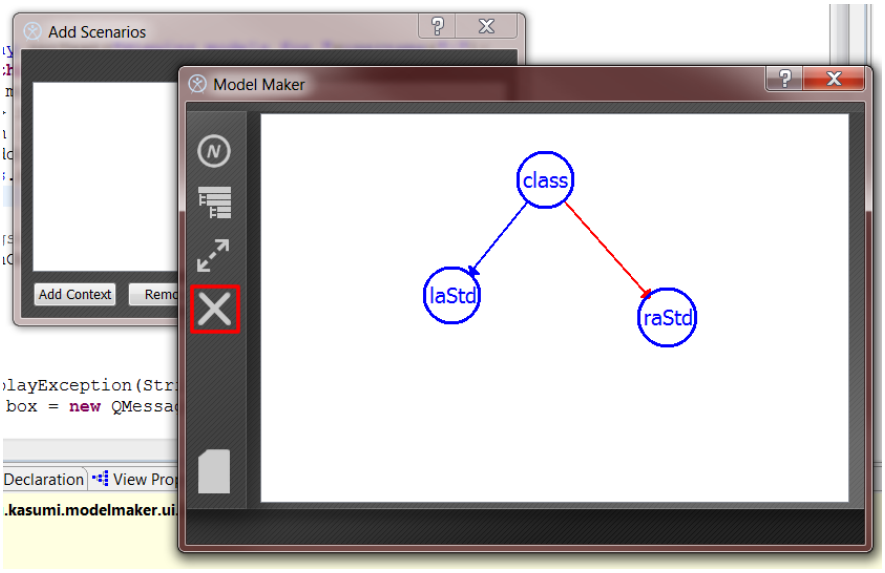
a. Add root



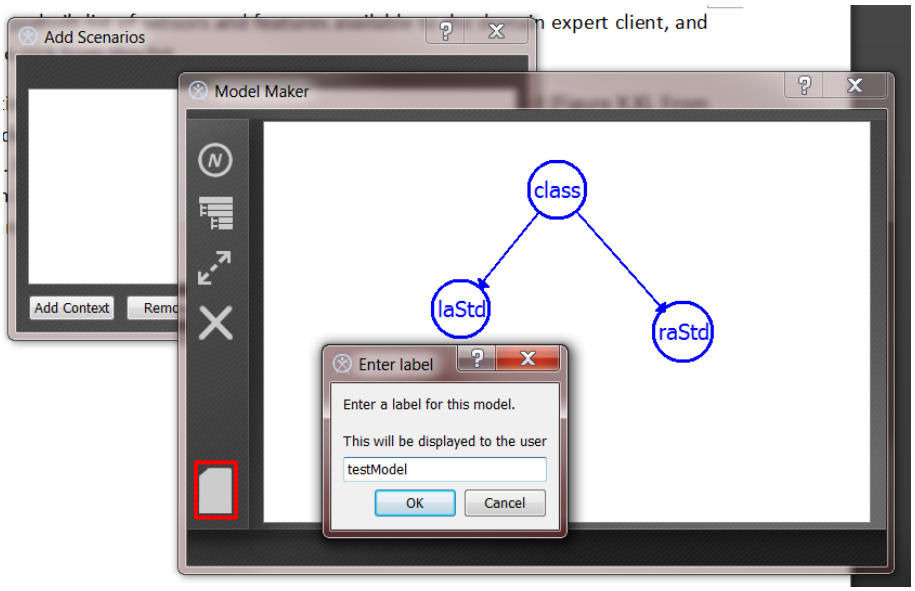
b. Add features



c. Link nodes



d. Delete links or nodes (optional)



e. Compile and submit model

Figure 3.29: Adding activity model

Chapter 4

Data Collection

Apart from the data recorded wirelessly via Bluetooth, some data collection campaigns conducted in the thesis study required offline data recording using the standard X6-2mini. The data logging and labeling method that this thesis developed for supporting these campaigns presents a significant advancement to the available technologies currently used in the community. This chapter aims to provide a description of the problems facing long data collection campaigns, and describe what we developed to address them.

4.1 Problems

Data acquisition not only involves collecting sensor data and corresponding annotations, but also includes post-processing analysis, where all annotations must be matched with corresponding data. Only then are they ready to be used by classifiers.

While many studies in the area of activity classification provide detailed discussions on classifiers and features, they do not address the variety of issues related to data acquisition required for essential system training. Studies and practices have shown a

number of factors affecting data acquisition accuracy, ranging from end users being severely inconvenienced by the equipment they have to carry, to users not being able to record properly or meet the annotation demands using traditional pen and paper approaches [40-43].

Another problem we observed in large measurement campaigns is that many time references for events are recorded based on different clocks (watches, wall clocks etc), depending on where the subject was at the time. These clocks are not synchronized, and can be several minutes apart with each other, compared to the sensor system time. This phenomenon dramatically reduces the effectiveness of the labeling process.

Interestingly, while the process of manually labeling long periods of activity data is a time consuming exercise, given an already labeled dataset it is easy to verify that the labels are correct, as the next Section demonstrates.

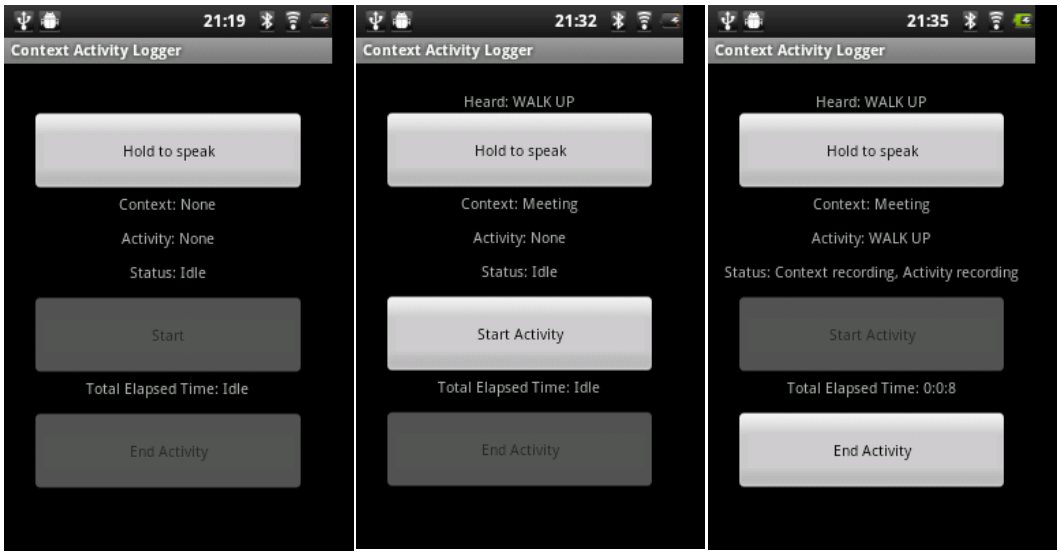
4.2 Context and Activity Data Acquisition and Labeling System

We have designed and implemented a complete data acquisition and processing system that includes voice recognition on an Android based client and a centralized server-hosted labeling tool. Our system is developed against Android SDK version 2.2 and the target device may be any Android smartphone (we used an Archos 32 Internet Tablet, which has support for wireless and audio recording).

The context and activity data acquisition system is displayed in Figure 4.1. While collecting data, a user can hold down the "Hold to Speak" button and speak a recognized context or activity label (a). Once it has been recognized, the "Start" button can be used to either start an activity or context. While recording, the "Status" label shows what are being recorded (b). If there is a context currently being recorded, then an audio recording and a wireless scan is performed periodically. If there is an activity being recorded, then there is a cumulative timer for that activity in the "Total Elapsed Time" display (c). This is useful in tracking activities that takes a short amount of time to complete, but must be repeated multiple times in order to gather enough data. For example, stairs are usually short, so to record walking up stairs for 5 minutes would require multiple attempts, and the total elapsed time can be used to see when 5 minutes of total recording is done.

To perform a data collection run, all sensors have their time synchronized to the phone. For wireless sensors, this is done through wireless control commands from the phone, and for non-interactive sensors this is done by synchronizing the sensors and the phone with a computer. At the beginning of a recording, the user is required to first bundle the sensors together, and shake them repeatedly while indicating to the logger that a synchronization marker is in progress (say to the software "Sync"). Once sync is performed, a user can carry out the data recording, using the application to make many annotations necessary. At the end of the run, we have a toolkit that takes the recorded data (merged into a MATLAB file) and annotation file (generated on the phone as

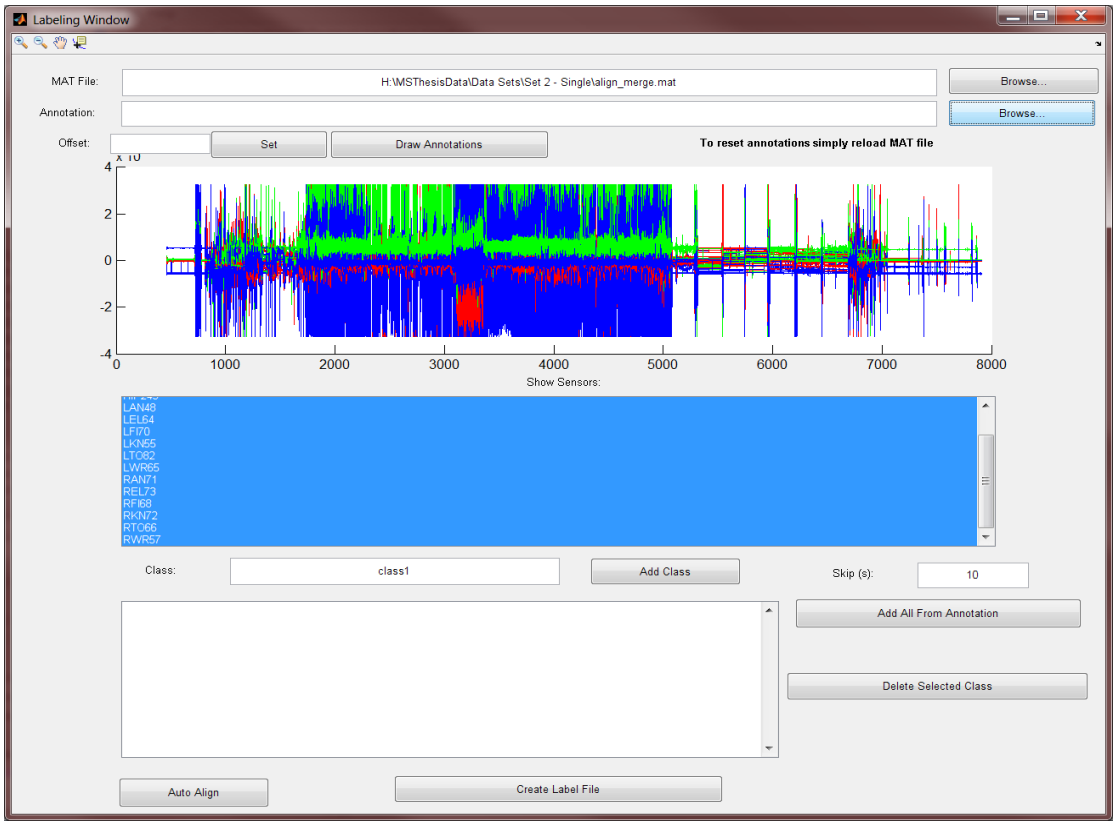
annotation.txt), automatically detect the synchronization time series in the data, and align this to the sync marker from the annotations. From there, the entire sequence of recording is labeled (Figure 4.2.b).



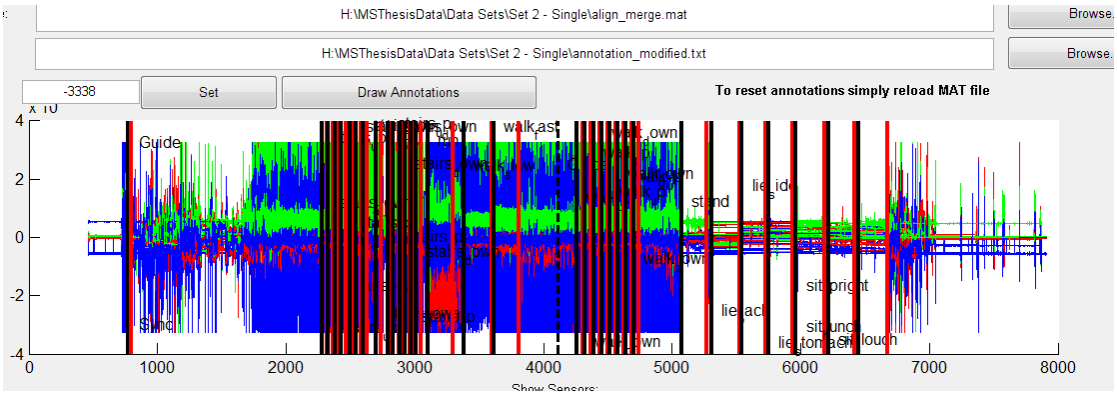
a. Initial screen b. Speak keywords c. Logging in progress

Figure 4.1: Android data collection application

In Figure 4.2.b, black lines are the start of activities, and red lines are the end of activities. We can see from Figure 4.2.a that manually labeling this 1.5 hour long recording would be difficult (let alone a 10+ hour recording). However, to check that the automatic labeling is correct, we can zoom into individual activities (Figure 4.3).



a. Import files



b. Label

Figure 4.2: Automatic labeling

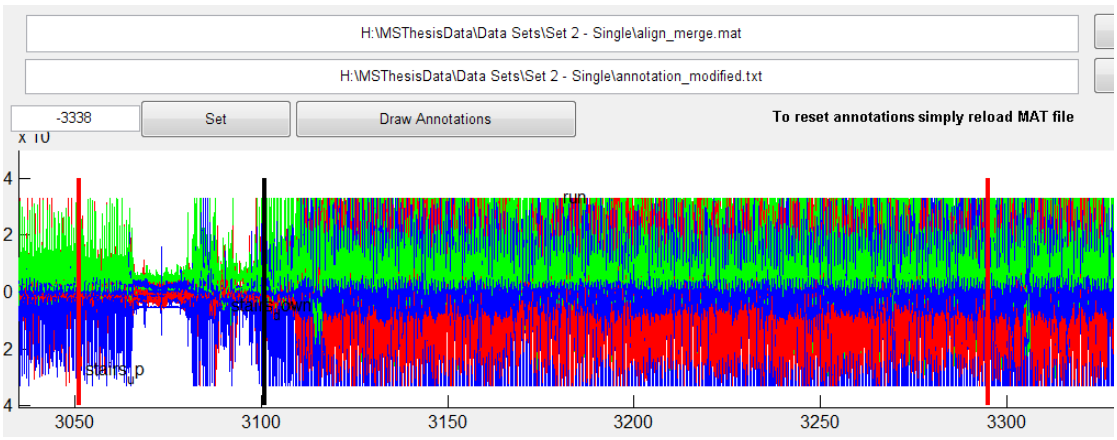


Figure 4.3: Zoomed in waveform with label

Here the transition on the accelerometer waveform between the activities is clear, and the black line is clearly at the correct location. Notice the red lines stop short of when the actual activities end. This is because the person recording the data specifically ended the annotation before he stopped performing the activity.

Now manual effort is only needed for quality checking. Compared to the process of labeling a day-long dataset after the recording is done, this new process drastically reduces the amount of time and effort required for organizing collected data (in our system the labeling is done in three clicks). Using this system, we have a robust means for supporting large campaigns, where users will be given a kit containing an Android (or other smartphone) application and sensors. For activities where the subjects cannot label data with hands (e.g. while in sports), a Bluetooth ear piece can be provided (most current solutions require a second person to do the annotation).

Chapter 5

System Evaluation

System evaluation is done on a number of levels. First, core components are verified individually to make sure that they are working correctly. Then the context guided activity classification system as a whole is evaluated with real life data. This is followed by evaluations of the performance of the activity classification system under different conditions. Finally, limitations of the system are explored.

5.1 Verification

5.1.1 Wireless Sensor Instrumentation

For the wireless sensor instrumentation stack, we are interested in seeing that: 1) The collected data streams are in sync and 2) There are no missing data. Figure 5.1 shows a plot of instrumenting with two accelerometers.

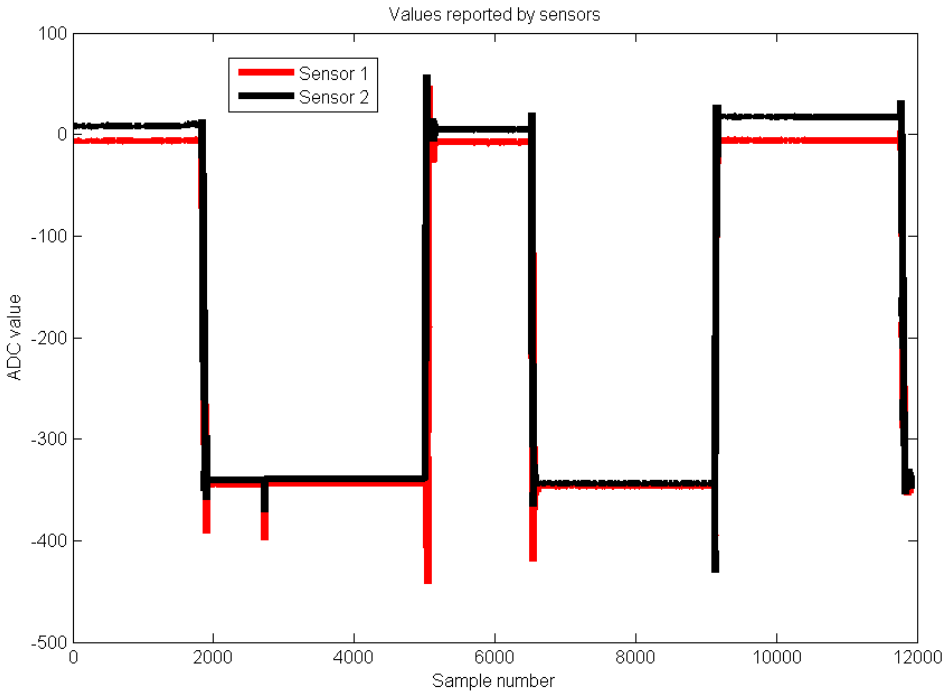


Figure 5.1: Data recorded wirelessly

This is 5 minutes worth of data collected by leaving the sensors flat on a table, and periodically flipping one of the axis (by placing the sensors upside-down). First, we see that the two sensors are clearly in sync. Second, we see that there are no missing data (the lines are bolded for better visual effects, missing and corrupt data show up as holes).

5.1.2 Bayesian Networks Implementation

The implementation of the core BN on the server side is tested individually using a set of test data from the University of California Irvine's Machine Learning Database [36].

Table 5.1 summarizes the datasets, while Table 5.2 compares our results against a well known implementation of Bayesian networks from Waikato Environment for Knowledge Analysis (WEKA) [44].

Table 5.1: UCI dataset description

Dataset	Description
Lens	A toy dataset that describes the type of lens a person wear based on a number of physical factors.
monk-problems [45]	A set of data generated by randomly permuting certain attributes to 0.
Letters	A data set used for testing classifiers on their ability to recognize English letters. Each letter is divided into sub areas and the pixel values are recorded.
post-operative-patient	A number of patients' vital signs were recorded in a hospital, and the patients' destination ward was recorded as the class label (intensive care, general ward, discharged).
solar-flares	A number of solar flares were recorded, and they are classed based on Zurich classes
weather (tennis)	Another toy dataset that is commonly used by a number of text books. This set describes the relationship between the tennis game going ahead and some weather patterns.

Table 5.2: BN results

Dataset	**Instance # (Train/Test)**	**Thesis Toolbox**	**WEKA BayesNet**	**Best Known**
Lens	24/24	95.83%	95.83%	-
monk-problems	124/432	70.92%	71.29%	94% [45]
Letters	2000/18000	69.16%	64.17%	69.16%
post-operative-patient	20/90	76.5%	76.67%	-
solar-flares	50/273	72.11%	72.31%	-
weather (tennis)	14/14	92.85%	92.85%	-

We see that for most cases the performance is comparable between our toolkit and WEKA. This is not surprising as we are also implementing a Bayesian network with the same underlying principles. The minor differences are due to our system using the Dirichlet density functions for parameter learning. For the monk problem, Bayesian networks performed poorly compared to the best known result. This is because the dataset is biased towards rule based classifiers (and from [45] we see that all the top performers are indeed rule based classifiers).

5.2 System Results

5.2.1 Data Collections

We have two collections of results. The first collection contains both context and activity data, and is used to evaluate the entire system. The second collection contains only activity data, and is used to evaluate the performance of the activity classification system under different conditions.

5.2.1.1 Data Collection 1

Table 5.3 lists all scenarios built for this experimental trial.

Data acquisition was performed as follows: three subjects carried an Archos Internet Tablet and six Medical Daily Activity Wireless Network (MDAWN) devices. The tablet supports our Android client, and the MDAWN devices were placed on wrists, waist and ankles. The MDAWNs robustly provide trialaxial accelerometer data [46]. Each subject was asked to record two sets of data. The first set is for training the classifiers, where subjects spent 30 minutes in each context, and performed every required activity under that context for at least 5 minutes. The second set is for testing purposes, and each subject spent over eight hours across the contexts, collecting all data listed.

In total, we collected six full sets of training and testing data. This was sufficient for developing and verifying the overall system, but more data would be needed for any clinical inference and for optimization of features, classifiers and sensor selection.

Table 5.3: Context guided models

	Walking	**Running**	**Walking Upstairs**	**Walking Downstairs**	**Sitting**	**Standing**	**Writing**	**Eating**
Outdoors	X	X	X	X				
Cafeteria	X				X	X		X
Home	X		X	X	X	X		
Class	X				X		X	
Meeting	X				X	X	X	
Bus					X	X		

5.2.1.2 Data Collection 2

The second data collection was designed and carried out by the summer Center for Embedded Networked Sensing (CENS) students, and a full description of their collection methodology can be found in the report [47]. In summary, they used 14 sensors set at 160Hz sampling rate and recorded 14 different activities for 5 minutes each. The list of activities and the sensor placements are reproduced here for completeness.

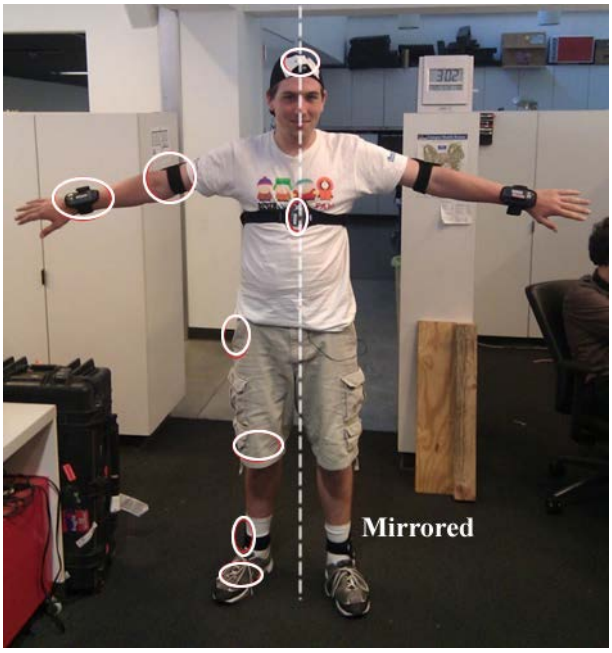


Figure 5.2: Sensor placements

Table 5.4: List of activities

Motion Based	Stationary
Walk slow	Stand
Walk fast	Sit upright
Run	Sit while slouching
Walk up slope	Sit while hunching
Walk down slope	Lying on back
Walk upstairs	Lying on stomach
Walk downstairs	Lying on side

5.2.2 Context Classifiers

Our new context guided classification method includes the classifier committee system, and experimental results directly demonstrate its effectiveness, in the presence of complex classification challenges.

Table 5.5 summarizes the accuracies of the classifier committee and the individual classifiers in the committee in percentage of correctly classified instances.

Table 5.5: Context classifiers

	AdaBoost (%)	**Time kNN (%)**	**Wireless kNN (%)**	**Committee (%)**
Bus	80	59	29	80
Cafeteria	100	35	80	90
Class	80	87	89	95
Meeting	100	73	100	100
Outdoors	85	33	25	85
Home	100	100	100	100

We see that wireless kNN performs with insufficient accuracy for bus and outdoors. In the bus context case, the sensor system detects a large number of wireless access points that have not been incorporated into prior training due to the route of the bus. In the outdoor context case, the system tends to detect access points that belong to one of the contexts at nearby indoor locations. For example, walking near a building causes the context to be classified as that of a context inside the building. Time KNN is also not sufficiently accurate for a number of contexts, and this is due to the varied nature when subjects visit these contexts. For example, subjects visited the cafe and outside at different times of the day. AdaBoost using sound features seems to perform well for all contexts, but we noticed some cases where a bus driving nearby causes a misclassification. Figure 5.3 shows the AdaBoost error percentage vs the number of

iterations. We see that the error of classification reduces steadily as the number of iterations increase.

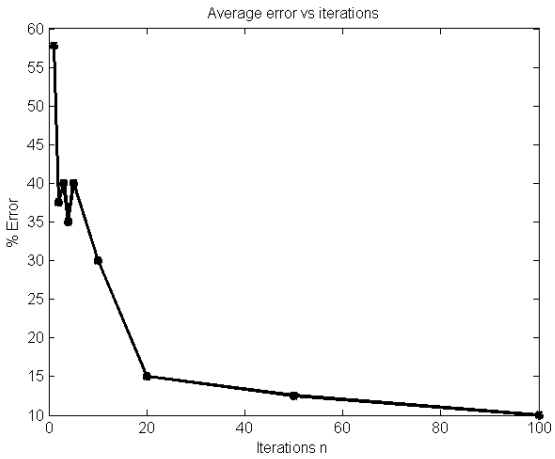


Figure 5.3: AdaBoost error vs iterations

Clearly, this experimental evaluation provides a classification challenge for each individual classifier. However, as shown in Table 5.6, by combining the best of all classifiers, our committee is able to achieve high accuracy for all contexts.

While the final system used the committee shown above, we also surveyed DTT, ANN and SVM. We will present the results for these below. Table 5.6 shows the results for DTT. We tested 108 samples from all 6 contexts.

Table 5.6: Accuracy of DTT

Context	Cafeteria	Class	Outdoors	Meeting	Bus	Home	Overall
Accuracy (%)	100	100	41	77	41	100	90.70

The ANN classifier we implemented was not able to produce a result, due to the disconnected nature of the wireless features.

We also implemented a C-SVM model for context detection, based on libSVM developed by Chang [48]. Table 5.7 shows the results for SVM, this classifier is used only for sound features as wireless features cannot be used on the SVM. We see that the SVM classifier classifies many other classes into outdoors. This caused the outdoors accuracy to be very high where as the other classes have low accuracy.

Table 5.7: Accuracy of SVM

Context	Cafeteria	Class	Outdoors	Meeting	Bus	Home	Overall
Accuracy (%)	67	30	100	79	67	Missing	67.60

5.2.3 Context Guided Activity Classification

A critical benefit of context guided classification is a direct improvement in accuracy for each classifier. This is also demonstrated by experimental results here.

Results are broken down by context. The "Generic" column shows results from a standard classification system with all activities built in. The data in the column are produced using UCLA's Wireless Health Signal Processing Toolkit (WHSPT), a hierarchical Naive Bayes classification tool [49]. The "Specific" column shows accuracy from context guided classifiers. The "Confusion" column shows the top confusion matrix entries.

Table 5.8: Results for bus

Bus					
	Generic (%)	Confusion (%)	Specific (%)	Confusion (%)	Improve (%)
Sitting	95.94	Eating (2.2)	100	None	4
Standing	81.29	Sitting (13.67)	86.33	Sitting (13.66)	6

Table 5.9: Results for outdoors

Outdoors					
	Generic (%)	Confusion (%)	Specific (%)	Confusion (%)	Improve (%)
Walking	99.29	Running (0.71)	99.29	Running (0.71)	0
Running	95.79	Walking (4.21)	95.79	Walking (4.21)	0
Walking downstairs	90.47	Walking upstairs (9.53)	90.47	Walking upstairs (9.53)	0
Walking upstairs	97.3	Walking downstairs (2.7)	97.3	Walking downstairs (2.7)	0

Table 5.10: Results for cafeteria

Cafeteria					
	Generic (%)	Confusion (%)	Specific (%)	Confusion (%)	Improve (%)
Standing	96.91	Eating (2.1)	98.97	Eating (1.03)	2
Walking	84.81	Walking downstairs (13.02)	100	None	17
Eating	1	Sitting (96.5)	1.29	Sitting (98.71)	29
Sitting	100	None	100	None	0

Table 5.11: Results for meeting

Meeting					
	Generic (%)	Confusion (%)	Specific (%)	Confusion (%)	Improve (%)
Sitting	91.67	Writing (5.1)	100	None	9
Walking	97.83	Walking upstairs (2.17)	100	None	2
Writing	2.5	Sitting (97.5)	69.62	Sitting (30.38)	26.84 times
Standing	96.84	Eating (3.16)	100	None	3

Table 5.12: Results for home

Home					
	Generic (%)	Confusion (%)	Specific (%)	Confusion (%)	Improve (%)
Sitting	100	None	100	None	0
Standing	94.12	Eating (5.88)	100	None	6
Walking	98.47	Walking upstairs (0.8)	96.95	Walking upstairs (3.05)	-1
Walking downstairs	100	None	100	None	0
Walking upstairs	96.61	Walking downstairs (3.39)	96.61	Walking downstairs (3.39)	0

Table 5.13: Results for class

Class					
	Generic (%)	Confusion (%)	Specific (%)	Confusion (%)	Improve (%)
Walking	98.56	Walking downstairs (1.44)	100	None	1
Sitting	87.33	Eating (11.76)	71.04	Writing (28.96)	-20
Writing	3.66	Sitting (95.81)	79.41	Sitting (20.59)	20.7 times

We see that in most cases there exists some increase in classification accuracy. This is from the context specific models with reduced number of classes and features. In the case of writing, we are able to see a large increase in accuracy, from being practically impossible (less than 5% accuracy) to decent accuracy (70%). In the case of eating, we are seeing an improvement, but the overall accuracy is still very low. We believe that this is due to inappropriate feature choice, and further testing can be conducted. We note that under the class context, there is a decrease in accuracy for sitting, however the trade off here is that we are now able to classify writing with almost 80% accuracy. By using a different feature or tree structure, we may be able to stop the decrease for sitting.

5.2.4 Classification Speed Increase

Table 5.14 shows the speed increase we are able to achieve. We see that in all cases there is a significant increase in classification speed. This allowed us to construct an online classification system capable of running in real time. To demonstrate the effectiveness with multiple users, the results in the "Specific" column is averaged using time measured from 5 simultaneous users on the classification system. The "Generic" column is obtained again from the WHISPT.

5.2.5 Context Guided Classification Energy Usage

Context guided classification can also offers the capability for selecting optimal sensors and schedules for energy and operating lifetime benefits. This also permits a minimum number of sensor systems to be selected (for user convenience) while maintaining classification accuracy.

Table 5.14: Speed increase using context

	Generic (s)	**Specific (s)**	**Improve (# of times)**
Bus	0.119	0.013	9.2x
Café	0.120	0.019	6.3x
Class	0.122	0.021	5.8x
Meeting	0.127	0.024	5.3x
Outside	0.128	0.022	5.8x
Home	0.119	0.021	5.7x

Based on models constructed, we produced the sensor requirement chart in Table 5.15. Blank cells indicate that a sensor can be safely turned off without affecting the accuracy for a given context. For example, in the case of "Bus" (Table 5.3), only the left waist sensor is required as we are only interested in monitoring two activities, and the waist sensor alone is enough to identify them.

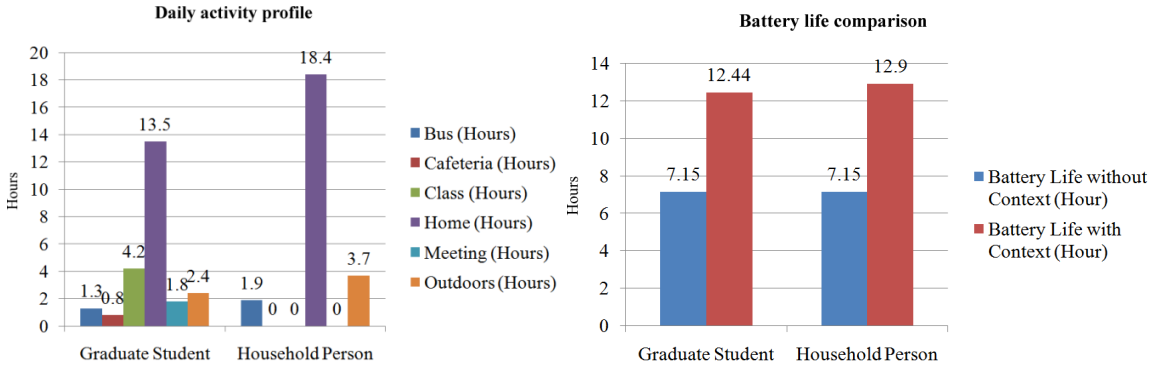
Using this chart, a sensor policy selector can determine which sensors to shut down. To estimate the potential for energy reduction, our analyses are directed to determining the improvement in operation time by adopting sensor activation and sampling schedules, as determined by context. The analysis was performed offline using manual calculations.

To indicate the operating time improvement over a range of subject behaviors, two cases were taken as examples, a graduate student subject and a subject remaining in a residential household. The typical profiles of their daily life are shown in Figure 5.4.a,

and the total operating time using continuous sensor system usage, in comparison to context guided sensor usage, is shown in Figure 5.4.b.

Table 5.15: Sensor requirement

	Left Ankle	Right Ankle	Left Waist	Right Waist	Left Wrist	Right Wrist
Bus			X			
Cafeteria	X	X	X		X	X
Class	X			X		X
Meeting		X	X			X
Outdoors	X	X		X	X	
Home	X	X	X			



a. User profiles b. Battery life comparison

Figure 5.4: User profiles and their battery life comparison

5.2.6 Effect of Discretizer on Bayesian Networks Classification Accuracy

The enhanced discretizer described in Section 3.4.8 affects the training phase of a BN, and thus the classification results. There are two parameters in the discretizer that would affect training: 1) Number of bins used and 2) Whether interpolation is used. In this analysis, 10 datasets were used (from collection 2) with classes walking slow, walking fast, walking up, walking down, running and standing. Figure 5.5 shows the average accuracy achieved with different parameters.

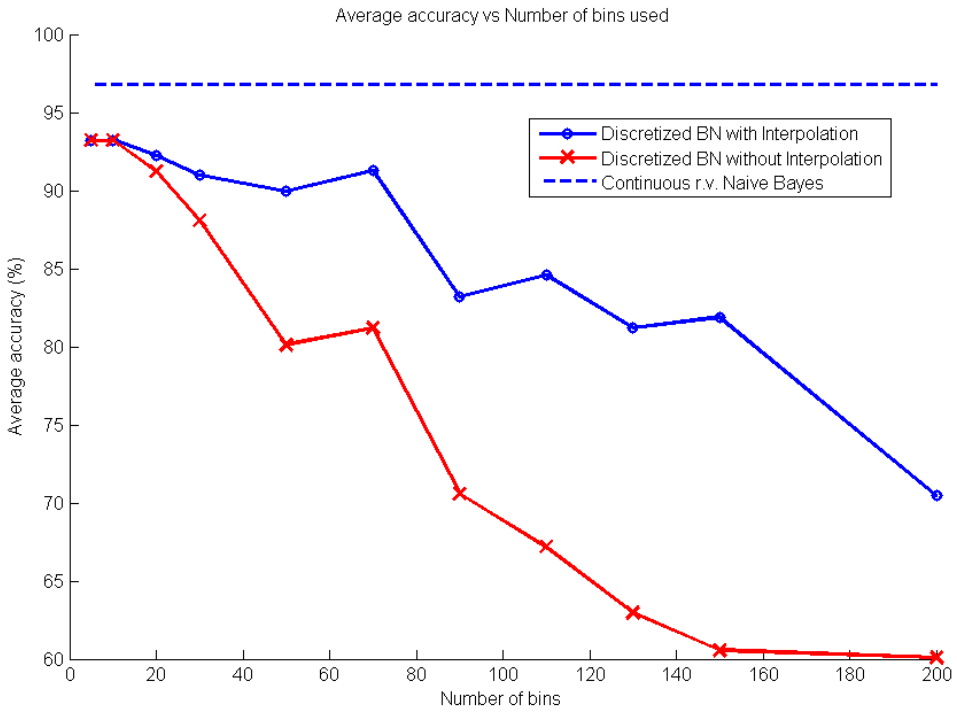


Figure 5.5: Average accuracy vs number of bins used

There are a number of observations. First, the maximum accuracy of the BN is not as high as the Naive Bayes classifier with continuous variables. This is because some observations on the testing set falls in the tails of the distribution, which in the BN are empty bins with zero probability. These observations have small probability in the naive Bayes method through integration of Gaussian, which is enough to make a correct classification.

Second, the accuracy of the BN decreases with an increase of the number of bins used. This is because as more bins are used, there are less number of samples per bin, and more and more bins become empty. At very high bin numbers (such as 200), clusters of bins where each activity is located only have few elements in them, with holes (blank bins) frequently appearing. While interpolation is able to fill in many of them, there are some that cannot be corrected.

Third, the accuracy seems very good at low bin numbers. As long as the distributions of the features for each class are roughly Gaussian, have sharp fall offs (low variance) and do not overlap each other, very low number of bins can capture the clusters very well.

Finally, we see that interpolation plays a significant role when the bin number increases. This is because more and more holes are being created, and without the interpolation to bridge the gap, these blank bins translate into zero probability during training, and subsequently produce unknown results during classification.

5.3 Limitations of Discretized Bayesian Networks

Through the work carried out in this thesis, we note a significant limitation of discretized BN. The discretized BN is more susceptible to feature distribution shifts than a classifier that works with continuous variables (such as Naive Bayes). While both cannot deal with significant shifts very well, the BN is much more sensitive to small shifts. This is because while a small shift translates in to smaller probability on the Naive Bayes classifier (as the probability is computed through an integration of the Gaussian tail), on a discretized BN it translates into some observations falling on an empty bin, which are classified as unknown.

Chapter 6

Conclusion

6.1 Conclusion

Activity monitoring appears as a critical need and valuable source of disease intervention and guidance in healthcare, personal health and wellness promotion, workplace safety, and athletics. In this thesis, we have described the design, implementation, and comprehensive evaluation of a novel end-to-end system that integrates context into activity classification.

On the architecture level, we first presented a refined definition of context, and a classification committee approach for detecting context of diverse forms. We then described how to interface with wireless sensors, and how any current classification system can take advantage of the new context guided architecture through the concept of a context guided classifier. Finally, we described the potential for real time classification through the use of servers and clients that exploit smartphone technologies. The architecture also employs an interface model, consequently providing great flexibility in the rapid implementation and integration of subsystems.

We also presented a realization of the above context guided classification system, where an Android client data collection application was able to solve issues relating to robust data acquisition and large campaign support. For the core system, AdaBoost, kNN and Bayesian network classifiers were all used for context detection and activity classification, demonstrating the inherent system flexibility. This has also demonstrated the important capability our system provides in enabling a matching of classifier systems to applications and the capability for the classification committee to properly combine these for optimization of classification accuracy.

Finally, through a series of experimental field evaluations sampling each of the diverse context examples and activities in multiple episodes by multiple subjects, the critical benefits of this system were demonstrated. First, it was demonstrated that context guided classification has enabled a substantial advance in classification accuracy for many activities including upper body motion. Second, it has been demonstrated that context guided classification offers a computational throughput advance that may be exploited for benefits including the support of real time, high accuracy classification. Finally, it was also demonstrated that the context classification capability can be applied to control the activation and selection of sensors. This benefit will be exploited in the immediate future to enable substantial operating lifetime extension for critical applications.

6.2 Future Work

We end the thesis with some future directions. First, throughout the thesis, we consistently felt the need for a standard activity recording database. Many groups work on the problem of activity classification, but every group uses their own sets of data, and often time little is known about how the data was collected, and how accurate the activities were.

Second, most current researches (including the research behind this thesis) take activities of interest from the list of activities of daily living (ADL). The definition of many activities such as walking, running, grasping are arbitrary, and different groups have different definitions. We believe that by finding a formal framework that could model motions, we can provide a unified way to define activities. This could also be the basis of a different type of classification method.

Lastly, we would like to carry out large clinical trials using the system described in this thesis, and produce some comprehensive results.

Appendix I

GCDC X6-2mini Serial Port Debugging Commands

Command Reference

'c' configuration

's' status

't' yyy-mm-dd hh:mm:ss' - set time

'P' Power off

'x' software reset

'v' Version information

'r/R' turn On/Off data Recording

'd/D' diagnostics On/Off

'+/-' double/halve sample rate

'm/M' microResolutionr On/Off

'u/U' resolUtion 12/16 bits

'g/G' gain 2/6 g

'f/F' filter On

References

[1] B. H. Dobkin, X. Xu., M. Batalin, S. Thomas, and W. J. Kaiser, Accelerometry algorithms for activity recognition. *Stroke*, 2011 (in press).

[2] W.H. Wu, M.A. Batalin, L.K. Au, A.A.T. Bui, and W.J. Kaiser. Context-aware sensing of physiological signals. In *Annual International Conference of the IEEE Engineering in Medicine and Biology Society,* pp. 5271-5275, 2007.

[3] M. F. Gordon. Physical Activity and Exercise Recommendations for Stroke Survivors. *Stroke*, pp. 1230-1240, 2004.

[4] B. Logan and J. Healey. Sensors to Detect the Activities of Daily Living. In *Engineering in Medicine and Biology,* pp. 5362-5365, 2006.

[5] M. Philipose, K.P. Fishkin, M. Perkowitz, D.J. Patterson, D. Fox, H. Kautz, and D. Hahnel. Inferring activities from interactions with objects. *Pervasive Computing*, IEEE, vol. 3, pp. 50-57, 2004.

[6] C. Doukas and I. Maglogiannis. Enabling human status awareness in assistive environments based on advanced sound and motion data classification. In *Proceedings of the 1st ACM international conference on Pervasive Technologies Related to Assistive Environments PETRA 08*, pp. 1, 2008.

[7] D. A. James, N. Davey, and T. Rice. An accelerometer based sensor platform for insitu elite athlete performance analysis. In *Proceedings of IEEE Sensors 2004,* IEEE, vol. 3, pp. 1373-1376, 2004.

[8] S. T. Scott, and L. Jonathan. iLearn on iPhone: Real-time human activity classification on commodity mobile phones. http://www.cs.washington.edu/homes/jlester/publications/UW-CSE-08-04-02.pdf, 2008.

[9] K. Van Laerhoven, A. Schmidt, and H. W. Gellersen. Multi-sensor context aware clothing. In Proceedings Sixth International Symposium on Wearable Computers, pp. 49-56, 2002.

[10] L. Han, S. Jyri, J. Ma, and K. Yu. Research on Context-Aware Mobile Computing. In *22nd International Conference on Advanced Information Networking and Applications Workshops 2008*, pp. 24-30, 2008.

[11] A. Schmidt, M. Beigl, and H. Hans-W. There is more to context than location. *Computers and Graphics,* vol. 23, pp. 893-901, 1999.

[12] D. Siewiorek, A. Smailagic, J. Furukawa, A. Krause, N. Moraveji, K. Reiger, and J. Shaffer. SenSay: a context-aware mobile phone, In *7th IEEE International Symposium on Wearable Computers*, IEEE, 2005.

[13] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, vol. 5, pp. 4-7, 2001.

[14] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger and J. Altmann. Context-awareness on mobile devices: the hydrogen approach. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, pp. 292-302, 2002.

[15] P. Prekop, and M. Burnett. Activities, context and ubiquitous computing. Special Issue on *Ubiquitous Computing Computer Communications*, vol.26, pp. 1168-1176, 2003.

[16] Monash University, "CSE5230 Tutorial: The Naive Bayes Classifier", Online: http://www.csse.monash.edu.au/courseware/cse5230/2004/assets/naivebayesTute.pdf, [August 17 2011].

[17] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, *Addison-Wesley Professional*, 1994.

[18] E. Curry, D. Chambers, and G. Lyons. Extending Message-Oriented Middleware using Interception. In *Proc. of the 3rd Int. Workshop on Distributed Event-Based Systems*, 2004.

[19] Y. C. Cheng, Y. Chawathe, A. LaMarca, and J. Krumm. Accuracy characterization for metropolitan-scale Wi-Fi localization. *ACM*, pp. 233-245, 2005.

[20] D. Nene. "Performance Comparison - C++/Java/Python/Ruby/Jython/JRuby/Groovy", Online: http://blog.dhananjaynene.com/2008/07/performance-comparison-c-java-python-ruby-jython-jruby-groovy/ [August 10, 2011].

[21] Nokia Corporation, "Qt", Online: http://qt.nokia.com/products/ [August 11, 2011].

[22] Oracle Corporation, "Serializable (Java 2 Platform SE v1.4.2)", Online: http://download.oracle.com/javase/1.4.2/docs/api/java/io/Serializable.html [August 11, 2011].

[23] Oracle Corporation, "Map (Java 2 Platform SE v1.4.2)", Online: http://download.oracle.com/javase/1.4.2/docs/api/java/util/Map.html [August 11, 2011].

[24] N. Garcia-Pedrajas and D. Ortiz-Boyer, "Boosting k-nearest neighbor classifier by means of input space projection", *Expert Systems with Applications*, vol. 36, pp. 10570-10582. 2009.

[25] T.K.C. Neo, "A Direct Boosting Algorithm for the K-nearest Neighbor Classifier Via Local Warping of the Distance Metric", Brigham Young University, 2007.

[26] I.H. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems), *Morgan Kaufmann*, 2005.

[27] Y. Freund, R.E. Schapire, and P. Avenue, "A Short Introduction to Boosting", *Society*, vol. 14, 1999, pp. 771-780.

[28] R. Meir, "An introduction to boosting and leveraging," Advanced lectures on machine learning, 2003, p. 118?183.

[29] R.E. Schapire, P. Avenue, and R. A, "The Boosting Approach to Machine Learning An Overview," 2003, pp. 1-23.

[30] R.E. Schapire, P. Avenue, and R. A, "The Boosting Approach to Machine Learning An Overview," 2003, pp. 1-23.

[31] R.E. Schapire and Y. Singer, "Improved Boosting Algorithms Using Confidence-rated Predictions", *Machine Learning*, vol. 37, Dec. 1999, pp. 297-336.

[32] Y. Sun, RNA: Reusable Neuron Architecture for on-chip electrocardiogram classification and machine learning, Master Thesis in University of Pittsburgh, 2010

[33] R. Beale and T. Jackson, Neural Computing - An Introduction, Institute of Physics Publishing, 1990.

[34] J. Pearl, Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach, *Association for Advancement of Artificial Intelligence Conference 82*, 1982.

[35] J. Noble and T. Koski, Bayesian Networks: An Introduction, *John Wiley and Sons*, 2009.

[36] A. Frank and A. Asuncion, "UCI Machine Learning Repository", Online: http://archive.ics.uci.edu/ml, [August 17 2011].

[37] S.L. Lauritzen and F. Jensen, Stable Local Computation with Conditional Gaussian Distributions, *Statistics and Computing*, vol. 11, no. 2, pp. 191-203, 2001.

[38] B.R. Cobb and P.P. Shenoy, Inference in Hybrid Bayesian Networks with Mixtures of Truncated Exponentials, *International Journal of Approximate Reasoning,* vol. 41, 2006.

[39] Roving Networks, "RN-42", Online: http://www.rovingnetworks.com/rn-42.php [August 12, 2011].

[40] J.E.F. Friedl, Mastering Regular Expressions, *O'Reilly Media*, 1997.

[41] S. Pirttikangas, K. Fujinami, and S. Hosio. Experiences on Data Collection Tools for Wearable and Ubiquitous Computing, In *2008 International Symposium on Applications and the Internet,* IEEE, 2008.

[42] S. Patnaik, E. Brunskill, and W. Thies. Evaluating the accuracy of data collection on mobile phones: A study of forms, SMS, and voice. In *2009 International Conference on Information and Communication Technologies and Development ICTD*, pp. 74-84, 2009.

[43] S. J. Lane, N. M. Heddle, E. Arnold, and I. Walker. A review of randomized controlled trials comparing the effectiveness of hand held computers with paper methods for data collection, *BMC medical informatics and decision making*, vol. 6, pp. 23, 2006.

[44] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten, The WEKA data mining software, *ACM SIGKDD Explorations Newsletter*, vol. 11, 2009.

[45] J. Wnek and R. S. Michalski, Comparing Symbolic and Sub symbolic Learning: Three Studies, Machine Learning: A Multistrategy Approach, Vol. 4., R.S. Michalski and G. Tecuci (Eds.), *Morgan Kaufmann*, 1993.

[46] X. Xu, M. A. Batalin, W.J. Kaiser, and B. Dobkin. Robust Hierarchical System for Classification of Complex Human Mobility Characteristics in the Presence of Neurological Disorders, In *Body Sensor Network,* 2011

[47] A.H. Khan, Data Collection Prompt 4, *CENS,* 2011.

[48] C. Chang and C. Lin, "LibSVM: a library for support vector machines," 2001.

[49] J. Y. Xu, Z. Wang, S. Yuwen, G. J. Pottie and W. J. Kaiser, Context Guided Personalized Activity Classification System with Real Time End-to-End Web Implementation, *Wireless Health 2011*, accepted.