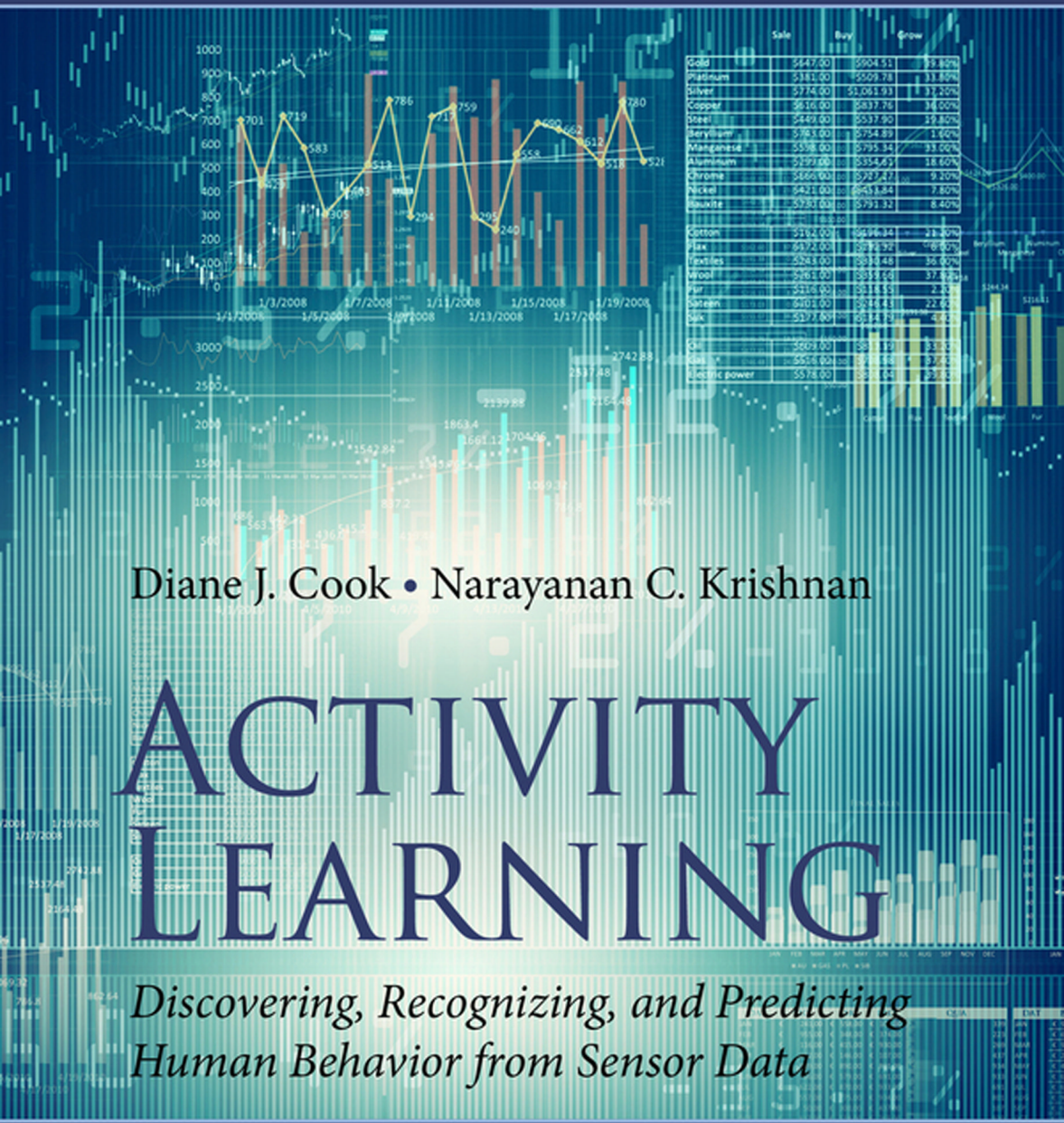
Wiley Series on Parallel and Distributed Computing

Albert Y. Zomaya, Series Editor



Diane J. Cook • Narayanan C. Krishnan

ACTIVITY LEARNING

Discovering, Recognizing, and Predicting Human Behavior from Sensor Data

WILEY

Activity Learning

WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING

Series Editor: Albert Y. Zomaya

A complete list of titles in this series appears at the end of this volume.

Activity Learning

Discovering, Recognizing, and Predicting Human Behavior from Sensor Data

Diane J. Cook
Narayanan C. Krishnan

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data:

Cook, Diane J., 1963-
Activity learning : discovering, recognizing, and predicting human behavior from sensor data / Diane J. Cook, Narayanan C. Krishnan.
pages cm. – (Wiley series on parallel and distributed computing)
Includes bibliographical references and index.
ISBN 978-1-118-89376-0 (hardback)
1. Active learning–Data processing. 2. Detectors–Data processing. 3. Multisensor data fusion. I. Title.
LB1027.23.C665 2015
371.3–dc23

2014039501

Typeset in 10/12pt TimesLTStd by Laserwords Private Limited, Chennai, India.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1 2015

Contents

Preface

Activity learning from sensor data is a research topic that is making its way into many fields including machine learning, pervasive computing, psychology, and sociology. The ability to discover and model activities based on collected sensor data has matured. These methods are now able to handle increasingly complex with increasingly complex situations including unscripted activities, interrupted or interwoven activities, real-time learning, and learning from multiple users or residents. At the same time, there have been advances in the application of the theoretical techniques to challenge real-world problems including health monitoring and home automation.

The goal of this book is to define the notion of an activity model learned from sensor data and to present the key algorithms that form the core of the field. While many current efforts have contributed to the ability to automatically recognize known activities, there are other aspects of activity learning that we want to include in this treatment. These include discovering activity patterns from unlabeled data and predicting when specific activities will occur in the future, as well as mapping sensor event sequences to predefined activity labels. We also want to discuss the challenges that are faced when these theoretical techniques are applied to real-world problems and suggest methods for addressing the challenges.

This book is designed for an interdisciplinary audience who would like to use or design activity learning techniques. As a result, most of the ideas are presented from the ground up, with little assumptions about the background of the reader. Ideally, the book will provide some helpful background and guidance to researchers, undergraduate or graduate students, or practitioners who want to incorporate the ideas into their own work.

The best way to understand activity learning techniques is to look at activity sensor data, play with existing tools, and writing your own code. To help with this process, we provide code for many of the techniques described in this book. A variety of

activity sensor datasets are available for your use as well. All of the related book materials can be found online at http://eecs.wsu.edu/~cook/albook.

We would like to thank the many individuals who helped us with this book. Jacqueline Southwick took the activity photographs that we included and the team of Kyle Elsalhi and Anthony Simmons, who generated the Kinect-based motion history image. Numerous individuals contributed to the data collections we made available with this book and who provided feedback on drafts of the book. These include Larry Holder, Maureen Schmitter-Edgecombe, Aaron Crandall, Brian Thomas, Yasmin Sahaf, Kyle Feuz, Prafulla Dawadi, Ehsan Nazerfard, Bryan Minor, Adriana Seelye, Carolyn Parsey, Jennifer Walker, Alyssa Weakley, Sue Nelson, Thomas Cowger, Selina Akter, and Prasanth Lade. Our editors at Wiley provided guidance throughout the entire process and for that we are grateful. Last but not least, we want to thank our colleagues, friends, and family who were unfailingly supportive of this effort. Diane would like to thank her family, Larry, Abby, and Ryan, who kept her going with ideas, encouragement, and lots of humor. She dedicates this book to them. Narayanan dedicates this book to his family, Krishnan, Geetha, and Karthik for their constant encouragement and support.

Diane J. Cook and
Narayanan C. Krishnan

List of Figures

1

Introduction

"All growth depends upon activity."
—Calvin Coolidge

"From time to time, it is worth wandering around the fuzzy border regions of what you do, if only to remind yourself that no human activity is an island."
—Julian Baggini

"People love chopping wood. In this activity one immediately sees results."
—Albert Einstein

"Nothing is more terrible than activity without insight."
—Thomas Carlyle

"We are what we repeatedly do."
—Aristotle

Learning and understanding observed activities is at the center of many fields of study. An individual's activities affect that individual, those around him, society, and the environment. In the past, theories about behavior and activities were formed based on limited observation. Over the past decade, the maturing of sensor technologies has made it possible to automate activity learning. Sensors have become miniature, low power, low cost, and high capacity. At the same time, we have witnessed tremendous progress in the areas of wireless networks, data processing, and machine learning. These advances have shifted researchers' focus from low level data collection and transmission to high-level information collection, inference, and recognition.

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

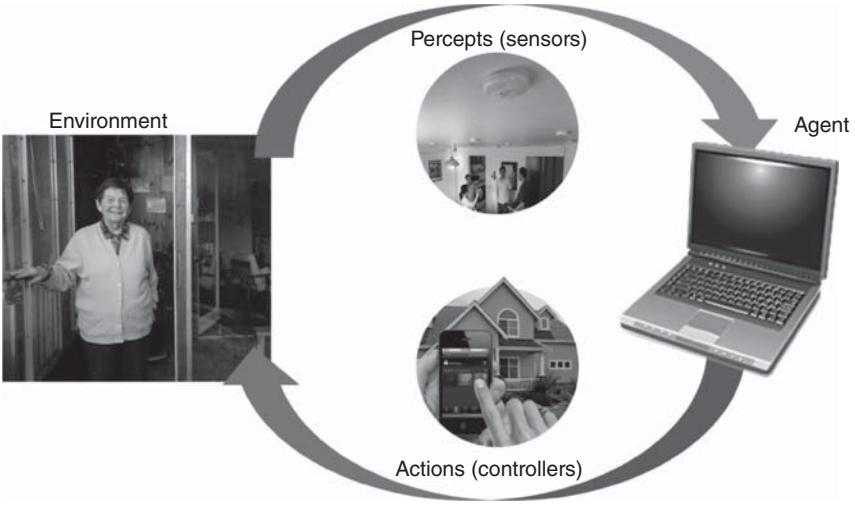


FIGURE 1.1 *The role of activity learning in the design of an intelligent agent.*

Activity learning plays a key role in the design of intelligent agents. Russell and Norvig[1] define an agent as an entity that perceives its environment through sensors and acts upon the environment through actuators. The techniques we describe in this book add intelligence and activity-awareness to this concept of an agent, shown in Figure 1.1. The state of individuals and their environment is perceived by sensors and stored as raw data. The activity learning agent processes this data in order to analyze it, make inferences from it, and describe it in terms of recognizable activities. Activity patterns can then be used to make a more informed decision about actions that can be taken to change the environment in a way that achieves the agent's goal.

Because activity learning technologies have become more robust, automated modeling and tracking of activities have become integral components of numerous solutions to real-world problems. For example, surveillance and security systems try to recognize and predict activities in order to address terrorist threats. Ambient-assisted living exploits activity discovery and monitoring to support independent living for individuals with cognitive and physical impairments. Activity-aware services have transformed ideas such as intelligent meeting rooms, automated homes, and personalized digital assistants from science fiction to everyday reality. Activity learning has also recently made its debut as a core component in products including game consoles and fitness smart phone apps.

As a result of the technology push and application pull, the number and diversity of projects that are designing or utilizing activity learning is exploding. Activity discovery and recognition is found in settings where data can be captured by cameras, body sensors, phones, or buildings. Entire conferences and journals are dedicated to reporting the latest advances in the field. Over the last 3 years, there have been two dozen workshops specifically devoted to activity recognition. The interest and enthusiasm for this topic is still increasing.

The goal of this text is to provide an in-depth look at computational approaches to activity learning from sensor data. Activity learning here refers to several aspects of activity processing. These include features that can be used to represent and model activities from sensor data. Also included are basic techniques for learning activity concepts and recognizing them in offline or streaming mode from sensor data. A companion to the idea of learning activity concepts from labeled data is the idea of automatically discovering activity patterns from unlabeled data, which we describe in detail. The third major component of activity learning that we include is the prediction or forecasting of activities that will occur at a time point in the future.

This book contains a pragmatic introduction to the primary algorithms and approaches to activity learning. Because the emphasis is on computational approaches, we will provide an algorithmic perspective. We detail the methods using pseudocode and provide implementations of the techniques in the source code that accompanies this book. We also illustrate the methods using examples of sensor data collected in real-world settings.

We note that the book is intended to be an algorithmic reference. It is written, where possible, to be independent of any specific sensor device or class of devices. Some of the examples used to illustrate the techniques are based on data collected from a particular type of sensor such as environment sensors, smart phone sensors, or cameras. However, the data features that are used to represent and recognize the data can be extracted and abstracted from a wide variety of low-level data collection sources. Here is a brief overview of the remaining chapters in the book.

Chapter 2 introduces the terminology that will be used throughout the text. The definitions of actions, interactions, and activities are provided and related to each other. A discussion of activity classifications and activity ontology generation is provided.

Chapter 3 provides an overview of sensors that are commonly used for observing and learning activities. In addition to describing sensor types and capabilities, the chapter includes descriptions of high-level features that can be extracted from low-level sensor data.

Chapter 4 introduces machine learning techniques that will be used throughout the remainder of the text. In addition to supervised classification techniques, the chapter also discusses approaches to dimensionality reduction that is valuable when learning concepts from high-dimensional sensor features.

Chapter 5 presents concepts and approaches for activity recognition. Additionally, methods for segmenting data into distinct activities are described together with methods to process streaming data. The chapter also includes performance metrics that can be used to evaluate and compare alternative activity recognition algorithms.

Chapter 6 describes algorithmic techniques for discovering activity patterns from sensor data. These techniques include zero-shot learning, sequence mining, clustering, and topic models. Methods for evaluating the results of activity discovery are summarized.

Chapter 7 covers activity prediction. Activity prediction includes identifying the next activity in a sequence as well as forecasting the next point in time when a particular activity will occur. As with activity recognition and discovery, metrics for evaluating the performance of activity prediction are included.

Chapter 8 discusses activity learning "in the wild." This refers to computational techniques that can be used to address problems that are encountered when activity learning is used in complex, real-world situations. These problems include obtaining sufficient high-quality labeled activity sensor data, transferring data or learned activity models to new settings with new users, mapping sensor sequences to multiple activity labels, and learning activities in the presence of multiple individuals.

Chapter 9 provides an overview of activity learning applications that are currently being investigated. Ongoing work is described that is related to activity recognition for health monitoring and intervention, marketing and activity-aware automation, surveillance and emergency management, describing routine behaviors in terms of activities, and analyzing human dynamics for entire populations.

Chapter 10 provides a conclusion of the book. This chapter poses some grand challenge problems for activity learning as well as uses for activity learning in other fields.

Additional materials including chapter updates, data sets, implementation of algorithms described in the book, and links to other tools and ongoing activity learning research projects are available online. These resources can be found at http://eecs.wsu.edu/∼cook/albook.

2

Activities

2.1 DEFINITIONS

Activity learning is an important concept because it is critical for understanding human behavior as well as designing human-centric technologies. As activity learning projects tend to focus on a specific subset of activity patterns and use a subset of available sensor types, it can be difficult to reach an agreement about the expectations of activity learning and how to compare alternative approaches. Our goal is to provide a unifying treatment of activity learning. As a beginning, we provide definitions for sensor events and activities that we will use as the basis of our discussions throughout the rest of the book.

Sensor Events As Figure 1.1 illustrates, an activity learner receives data from sensors that are used to perceive the state of an individual and/or an environment. We specify input data to an activity learner as a sequence of sensor events. Each sensor event, e, takes the form $e =< t, s, m >$ where t denotes a timestamp, s denotes a sensor ID, and m denotes the sensor message. We define an *activity instance* or *activity occurrence* as a sequence of n sensor events, $< e_1 \ e_2 \ \ldots \ e_n >$. An activity represents the collection of all of its instances. An activity learner may represent an activity as an abstraction of this collection of activity instances.

Types of Activities The terms "action" and "activity" are frequently used interchangeably in activity learning research. In actuality, there is a tremendous diversity of concepts that are classified as activities in the literature. The activity classes that are investigated differ in terms of activity complexity, the type of sensor modality that is typically used to capture the activity, and the computational techniques that are most

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

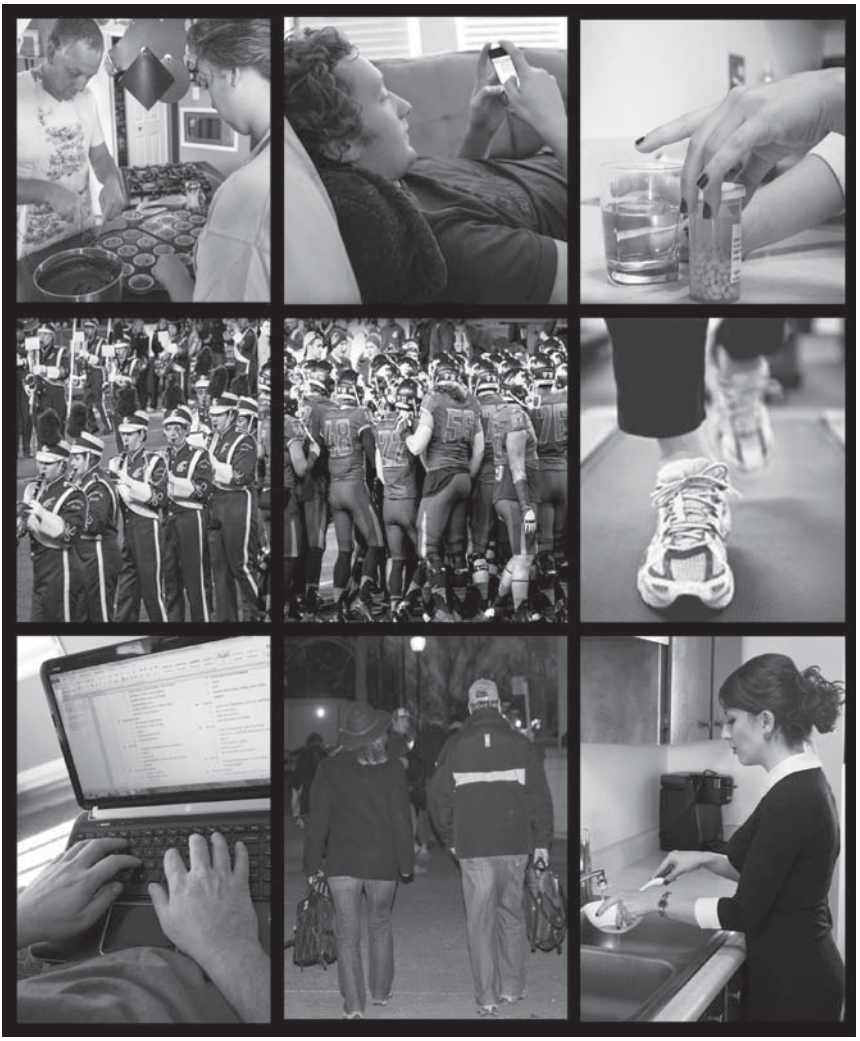


FIGURE 2.1 *Examples of individual group actions and activities that are found in common everyday life settings.*

effective for learning the activity. Figure 2.1 illustrates some of the activities that can be represented and learned using techniques described in this book. While sitting, standing, waving, and walking appear at one end of the spectrum of activities, the other end consists of longer, more complicated tasks such as cooking, marching in formation, and playing a football game.

Throughout this book, we refer to *action* as a simple ambulatory behavior executed by a single person and typically lasting short durations of time. Similarly, we refer to *interaction* as a short, single-movement action that involves multiple individuals.

By contrast, by *activity* we refer to complex behaviors consisting of a sequence of actions that can be performed by a single individual or several individuals interacting with each other. They are typically characterized by longer temporal durations. At the other end of the spectrum are individual states such as human (or animal) postures and poses, environmental state, and object location. While such states are indicative of human behavior, they occur at a single point in time and can be differentiated from actions and activities in this way. Note that these definitions are hierarchical. An action may consist of a sequence of states, while an activity may contain any number of actions. Activities may also be described in terms of environment states or influences that the environment has on the individual performing the activity. Figure 2.2 represents the relationship between these subsets of human behavior.

2.2 CLASSES OF ACTIVITIES

While activity sensor-based datasets are currently available for analysis, the activities that are labeled and learned vary greatly among these datasets. We provide one possible taxonomy of activity classes in Table 2.1. As can be seen in this list, there exist implicit and explicit relationships between activity classes. A model that can be used to represent the activity cooking may also be used to represent the subset cooking breakfast. On the other hand, a cooking breakfast model may not be an effective representative of all cooking tasks. Additionally, some activity classes may be considered functionally different and yet bear a striking resemblance in terms of the sensor events they generate. For example, an individual may fill a glass with water to drink or fill a kettle with water to boil. Such similarities between activities means that additional contextual information, including time of day and the previous activity, need to be considered when learning the activity class. Depending on the sensor modality, some pairs of classes may be virtually indistinguishable. For example, the movements

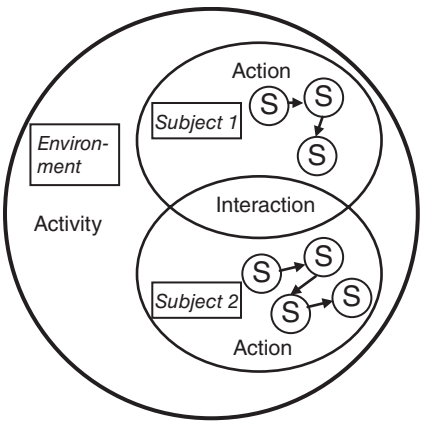


FIGURE 2.2 *Relationship between state (S), action, and activity.*

corresponding to lying down on a rug and falling down may generate identical sensor sequences if only infrared motion sensors are used. In these cases, the activities can only be distinguished by incorporating additional sources of information such as different sensor modalities.

2.3 ADDITIONAL READING

A number of authors have considered ways to categorize classes of activities for modeling, detecting, and recognizing. Candamo et al.[2] and Chaquet et al.[3] distinguish single person activities from multiple person activities and interactions. Borges et al.[4] offer a characterization that breaks activities into gestures, actions, and interactions in a manner similar to the discussion found in this chapter. Krishnan et al.[5] consider methods of learning activity taxonomies directly from sensor data. Chen et al.[6] and Bae[7] employ a method for activity learning that involves hand-constructing an activity taxonomy then using occurrences of the activities to refine the taxonomy. These learned activity ontologies can form a knowledge-driven approach to learning and reasoning about activities[8,9].

In this book, we will refer to examples activities that appear in Figure 2.1 and Table 2.1. However, this is by no means a comprehensive list nor is it the only taxonomical organization of activity classes. Other taxonomies have been generated for use in specific domains. For example, clinicians may evaluate an individual's cognitive and physical health based on their ability to independently complete known activities. As a result, in the health literature activities classes of activities are often categorized as activities of daily living (ADLs)[10] or instrumental activities of daily living (iADLs)[11]. Many resources are available that provide labels, categories, and examples of classes of activities. This includes work by Tapia[12], the Compendium of Physical Activities[13], the American time use survey[14], the Facial Action Coding System[15], the SCARE joint task corpus[16], and the UCF sports dataset[17].

TABLE 2.1 Categories of Routine Activities

Actions

- Walk, run, cycle, jump, sit down, stand up, bend over, throw, dig, kneel, skip
- Lie down, fall down, kneel down, faint
- Ascend stairs, descend stairs
- Open door, open cabinet, pick up item, push item, pull item, carry item, throw item
- Point, wave, talk, make fist, clap, gesture
- Chew, speak, swallow, yawn, nod
- Interactions
  - Shake hands, hug, kiss, hit, chase, wrestle, high five
  - Talk to someone, hand item to someone, throw item to someone

TABLE 2.1 (*Continued*)

Activities

- Clean house
  - Dust, vacuum, sweep, mop
  - Make bed, change sheets
  - Scrub floor, toilet, surface, windows, ceiling fans
  - Clear table, wash dishes, dry dishes
  - Garden, weed, water plants
  - Gather trash, take out trash
  - Organizing items
  - Wash clothes, sort clothes, fold clothes, iron clothes
- Repair home
  - Fix broken appliance
  - Fix floor, wall, ceiling
  - Paint
  - Replace light bulb
  - Replace battery
- Meals
  - Prepare breakfast, lunch, dinner, snack
  - Set table
  - Eat breakfast, lunch, dinner, snack
  - Drink
- Personal hygiene
  - Bathe, shower
  - Brush teeth, floss
  - Comb hair
  - Select outfit, dress
  - Groom
  - Shave, wash face, wash hands
  - Toilet
  - Trim nails, trim hair
- Health maintenance
  - Take medicine, fill medicine dispenser, apply medicine
- Sleep
  - Nighttime sleep
  - Sleep out of bed
- Pet care
  - Feed, water, groom
  - Walk, play, train
  - Clean pet home
- Exercise
  - Lift weights
  - Use treadmill, elliptical, cycle, rower

TABLE 2.1 (*Continued*)

- Calisthenics
- Stretch
- Martial arts
- Dive, golf, swim, skate

- Leisure
  - Play musical instrument
  - Read
  - Sew
  - Watch television, video, play video games
- Travel
  - Enter car, enter bus, exit car, exit bus
  - Drive car, bus
  - Ride in car, bus
  - Ride elevator, escalator
- Social
  - Make phone call, talk on phone
  - Send text, read text, send email, read email
  - Write letters, cards
  - Entertain guests
  - Leave home, enter home
- Work
  - Work at computer, work at desk, work at table
- Group activity
  - Play board game, play card game
  - Play sport against opponent
  - Play sport with team
  - Gather crowd, disperse crowd, move crowd

3

Sensing

Sensors are devices that detect and respond to signals. A sensor converts a physical parameter such as temperature, blood pressure, or humidity into a signal that can be measured electrically and reported as a sensor event. A wide variety of sensors are available for activity learning and monitoring. In recent years, these sensors have also become low cost, wireless, and deployable in real-world, mobile settings. As a result, sensors are ubiquitous and are being integrated into more diverse computational settings including activity learning and monitoring. In order to design a sensor-based approach to activity learning, we need to consider both the types of sensors that will be used and how the data can be preprocessed for integration into data analysis and machine learning algorithms. In this chapter, we review sensors that are commonly used for activity learning and describe the type of data they generate. We also describe features that can be extracted from sensor data and illustrate how they are used on sample activity sensor data.

3.1 SENSORS USED FOR ACTIVITY LEARNING

Sensors differ in type, purpose, output signal, and technical infrastructure. Here, we provide an overview of sensors that are popular for activity learning. We separate the discussion into two categories of sensors: sensors on objects that are not attached to the individual performing an activity (i.e., sensors in the environment) and sensors on objects that are attached to the individual that is performing an activity (i.e., sensors attached to clothes or that are carried by the individual).

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

3.1.1 Sensors in the Environment

Some sensors that monitor activities are not affixed to the individuals performing the activity but are placed in the environment surrounding the individual. These sensors are valuable in passively providing readings without requiring individuals to comply with rules regarding wearing or carrying sensors in prescribed manners. Because they are not customized for each person, environment sensors can monitor activities for a group of individuals but may have difficulty separating movements or actions among individuals that are part of that group. Here we describe sensors that are commonly found installed in and around physical environments.

Passive Infrared (PIR) Sensor PIR sensors, or motion sensors, detect infrared radiation that is emitted by objects in their field of view through multiple slots, as shown in Figure 3.1. If the difference in the detected radiation between the multiple slots of a PIR sensor is greater than a predefined threshold (as would happen when a warm body moves into or out of the range of the sensor), it generates a message. These sensors are often used in security settings where a light may turn on or an alarm may sound when motion is detected. Humans and pets emit infrared radiation, so PIR sensors can be used to detect their movement within their coverage area. Because a passive infrared sensor (PIR) sensor is sensitive to heat-based movement, it operates in dim lighting conditions. On the other hand, it may not sense movement if an object is between the sensor and the moving person. A PIR sensor will sense movement from any object that generates heat, even if the origin is inorganic. As a result, the motion sensor may generate messages if a printer starts printing a large document in the vicinity or if a nearby baseboard heater suddenly turns on.

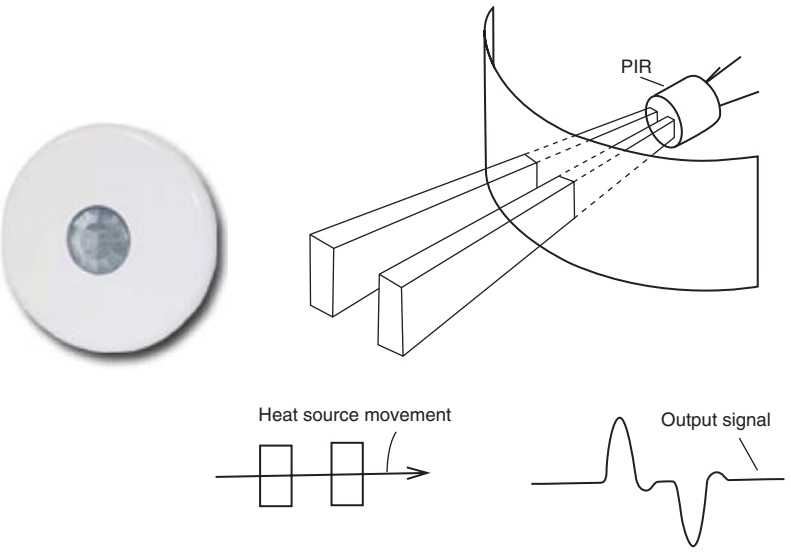


FIGURE 3.1 *PIR infrared sensor packaging (left) and internal architecture (right).*

FIGURE 3.2 *The two components of a magnetic contact switch encased in plastic (left) and a magnetic contact switch acting as a door open/shut sensor (right).*

Magnetic Door Sensor A magnetic contact switch sensor consists of two components: a reed switch and a magnet, as shown in Figure 3.2. When the door shown in Figure 3.2 is closed, the magnet component pulls the metal switch in the second component closed so the electric circuit is complete, thus changing the state of the sensor. The sensor can report this change of state as a sensor event. When the magnet is moved by opening the door, the spring snaps the switch back into the open positive. This cuts off the current and closes the relay, again causing a change in the state of the sensor that can be reported as a sensor event. This feature is useful for detecting if doors, windows, drawers, or cabinets are open or shut, as shown in Figure 3.2.

Temperature Sensor, Light Sensor, Humidity Sensor Additional sensors can be placed in environments to measure ambient temperature, lighting, and humidity. These types of sensors are frequently bundled into one package. For example, the motion sensor shown on the left in Figure 3.1 also contains a light-level sensor in the same unit, while the magnetic door sensor shown in Figure 3.2 also contains a temperature sensor. Such sensors are calibrated to each periodically report their current status (e.g., light level, humidity reading, and temperature reading) or they report their current reading when there is a sufficiently large change in the value from the previous time point.

Vibration Sensors These types of sensors are often attached to items or placed on surfaces in order to detect interaction with the corresponding object. Some sensors are designed to be sensitive both to vibration (dynamic acceleration) and tilt (static acceleration). While they can be useful for generating events when the object they are attached to is handled, they may also generate events when they are accidentally bumped or when nearby surfaces shake.

FIGURE 3.3 *Tactile pressure sensors positioned in a chair (left) and interpreted as a pressure map (right).*

Pressure Sensors There are many types of sensors that measure pressure. For monitoring of activities in a particular environment, tactile sensors are sensitive to touch, force, or pressure. These pressure sensors detect and measure interaction between an individual and a contact surface. For example, Figure 3.3 shows pressure sensors placed on a chair. The force distribution can be mapped as shown in the figure or the combined force can be compared with a threshold value to note that there is an object in contact with the sensor. Pressure sensors can be placed on or under chairs, door mats, floors, and beds to monitor the location and weight distribution of an individual in the space.

Global Positioning System (GPS) Sensors GPS sensors provide valuable location information and can be attached to objects in the environment or on a wearable or carryable device. They rely on communication with at least four global positioning system (GPS) satellites and thus are not usable in all settings. As an example, a smart phone can communicate with cell towers in the region to triangulate the position of the device based on time delay and signal strength in sending the message to and from the towers. Common reporting axes are latitude, longitude, and height. In addition to GPS, other signals including GSM, WiFi, and Bluetooth signals can be employed to provide improved localization, particularly indoors where GPS may not be consistently available.

Radio-Frequency Identification (RFID) Sensors Radio-frequency identification (RFID) employs radio-frequency electromagnetic fields to transfer data. RFID tags act as wireless barcodes that can be attached to objects. Figure 3.4 shows how RFID codes can be "hidden" under a visible object barcode. The RFID reader, or interrogator, queries a tag. A passive tag does not require a power source but collects energy from the interrogating electromagnetic (EM) field and then acts as a passive transponder to modulate the signal. The reader can interpret the signal to determine the tags that are in its area of coverage. As a result, RFID technology can be used to

FIGURE 3.4 *RFID tag positioned beneath visible object barcode.*

track the presence of objects or humans that are wearing a tag and are in close proximity to a reader. More complex tags are designed to be programmable and act as a power source for other sensors that are described in this chapter including light, temperature, and accelerometer sensors. RFID technology can be limited by the range of the reader, which is greater outdoors. For indoor environments, a reader typically needs to be placed in each room where tags will be monitored.

Power Meter Power meters provide readings indicating the amount of electricity that was consumed by a particular building for a unit of time. Electricity typically flows from a utility meter to the building's breaker panel. One method of monitoring usage is thus to clamp a measuring device around the main conductors inside a building's breaker panel. The meter calculates the amount of electricity currently being consumed and can report instantaneous values, accumulated values, or sufficiently changes in consumption.

3.1.2 Sensors on the Body

In addition to placing passive sensors around the environment in which an individual interacts, the individual can also wear, or carry, additional sensors that collect data about their gestures, movement, location, interactions, and actions. These sensors were originally sewn into garments, worn as watches or placed on the body. However, many of these sensors are now also found on smart phones that are routinely carried by individuals as they perform their daily activities. Here we describe some sensors that are commonly used as wearable or carryable mechanisms for monitoring activities.

Accelerometer An accelerometer is a common sensor that is worn or carried by an individual. Acceleration changes can indicate the beginning or ending of a motion (see Figure 3.5). As shown in Figure 3.6, this three-dimensional sensor measures acceleration along the x, y, and z axes. Acceleration is calculated as the change in velocity over time, or $a = \Delta V / \Delta t$. Acceleration can be detected when the person

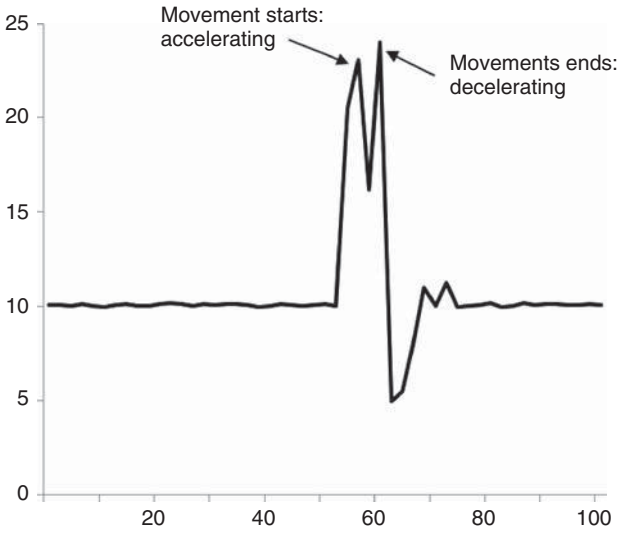


FIGURE 3.5 *Increased acceleration values can indicate a start or stop of a motion.*

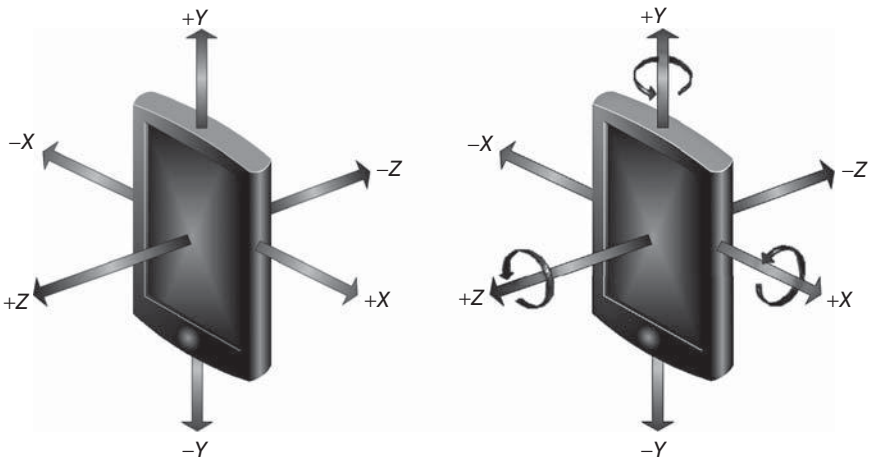


FIGURE 3.6 *A smart phone with a three-axis accelerometer (left) and a three-axis gyroscope (right).*

carrying the device initiates a change in direction or velocity, which makes these sensors ideal for detecting different types of movement.

Gyroscope A gyroscope sensor can enhance the movement detection provided by an accelerometer. In contrast to an accelerometer, the gyroscope measures the change in angular position over time, or angular velocity, calculated as $v = \Delta\theta/\Delta t$. Gyroscopes sometimes experience accumulated error due to drift but are frequently used

in combination with other sensors to provide a more complete model of motion. For example, gyroscope readings are often packaged with accelerometer readings to provide six-dimensional motion data vectors.

Magnetometer This sensor measures the strength of the magnetic field in three dimensions. While it bears some similarity to a compass, it does not always behave the same as a compass because the magnetometer may not always point north when it is influenced by magnetic interference. A magnetometer is valuable for providing some sensor orientation and for detecting and locating metallic objects within its sensing radius. Human activities can then be modeled based on their proximity to these detected objects.

3.2 SAMPLE SENSOR DATASETS

Appendix 1 contains sample sensor data that are captured while humans perform routine activities. There are two tables in the Appendix, one for each of two activities that are performed: Hand Washing and Sweeping. Each table contains sensor messages that are generated while one individual performs the activity with no interruptions. The individual performed the activities in a smart home filled with environmental sensors. These sensors include infrared motion detectors, magnetic door sensors, temperature sensors, water flow and gas flow sensors, a whole-home power meter, and accelerometer-based vibration sensors placed on selected objects in the space. The layout of the instrumented main floor of the smart home and the position of the sensors is shown in Figure 3.7. In addition, the individual wore two accelerometer/gyroscope sensors: one on the arm and one on the hip, as shown in Figure 3.8. Each wearable sensor reports six values: acceleration in the x, y, z directions and rotational velocity around the x, y, and z axes.

In the Hand Washing activity, the participant washes her hands at the kitchen sink using hand soap that is located in a dispenser next to the sink. After her hands are washed, she uses a cloth towel also located in the kitchen to dry her hands. Appendix 1 shows the sensor data that were gathered during the performance of this activity for one participant and in Figures 3.9 and 3.10 the timing of sensor events corresponding to the sensor data are plotted.

A second example activity is plotted in Figures 3.11 and 3.12 for one participant and the corresponding sensor data are provided in Appendix 1. This is a Sweeping activity in which the participant is asked to sweep the kitchen floor and to dust the dining room and living room. All of the needed supplies, including the broom, duster, and dustpan are located in the kitchen closet at the bottom right of the space shown in Figure 3.6.

3.3 FEATURES

In Chapter 4, we will introduce a variety of machine learning algorithms that can be utilized to model human activities. Typically, these algorithms do not process

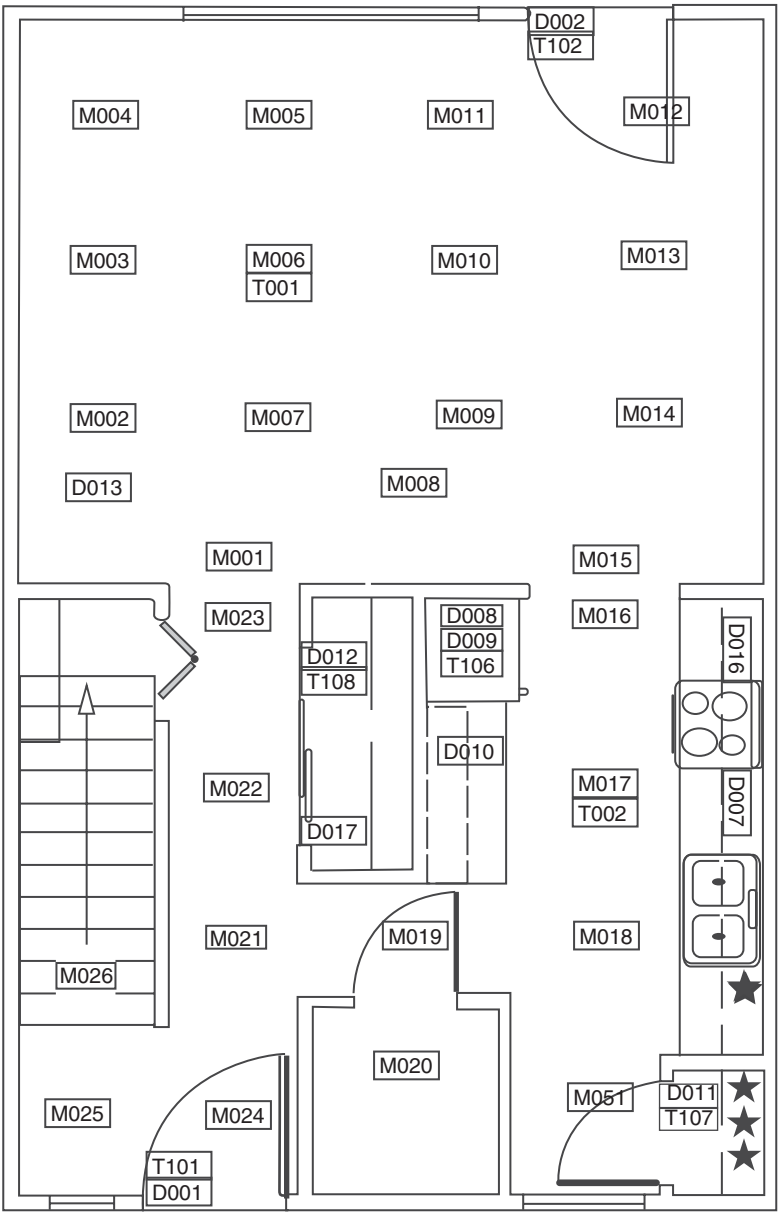


FIGURE 3.7 *Floor plan and sensor layout for the first floor of a smart home. Sensor identifiers starting with "M" are infrared motion sensors, sensors starting with "D" are magnetic door closure sensors, and sensors starting with "T" are ambient temperature sensors. The hand soap object sensor is located near the sink and the additional object sensors are in the kitchen closet. All object sensors are indicated with stars. Additional sensors included in these datasets are* `A001` *(gas usage sensor attached to burner),* `A002` *and* `A003` *(hot and cold water consumption sensors, respectively), and* `P001` *(whole-home power usage meter).*

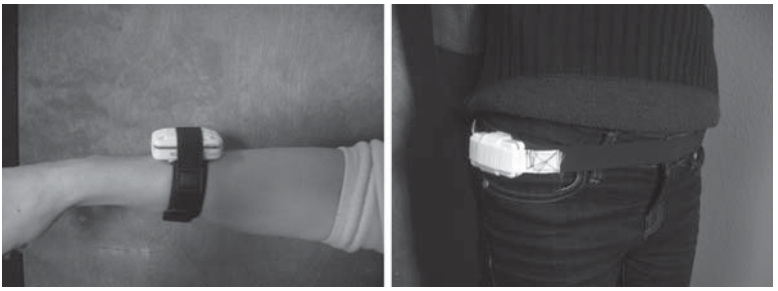


FIGURE 3.8 *Positioning of wearable accelerometers on an individual's dominant arm (left) and hip (right).*

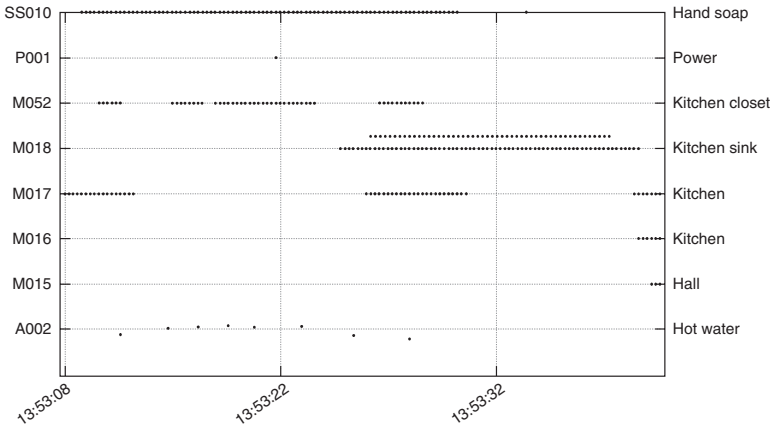


FIGURE 3.9 *Time plot of sensor activity during the Hand Washing activity. The x axis shows the time of day when the sensor message was generated. The y axis shows the identifier (on the left) and functional location (on the right) for each sensor generating messages.*

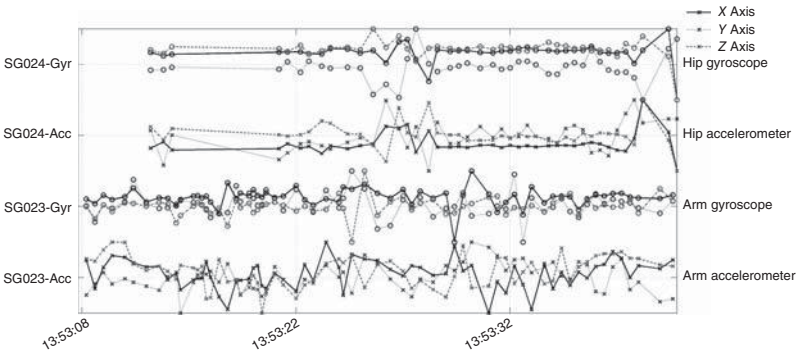


FIGURE 3.10 *Time plot of normalized accelerometer readings during the Hand Washing activity. The x axis shows the time of day when the sensor message was generated. The y axis shows the identifier (on the left) and placement (on the right) for each sensor. There are three lines for each sensor, corresponding to the x, y, and z axes.*

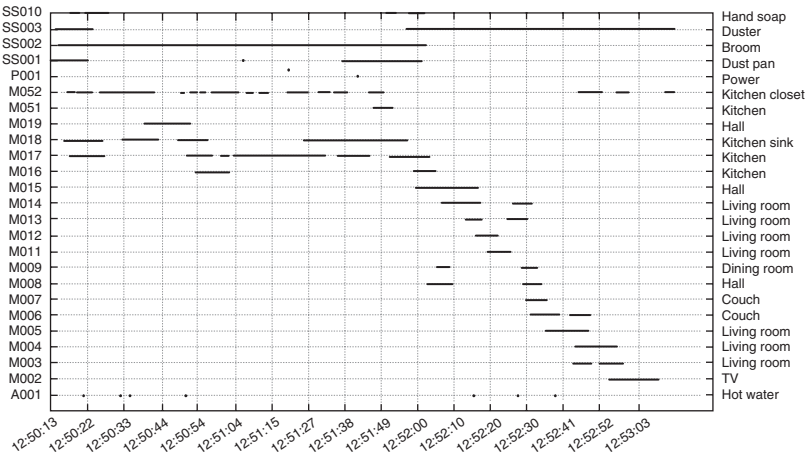


FIGURE 3.11 *Time plot of sensor activity during the Sweeping activity. The x axis shows the time of day when the sensor message was generated. The y axis shows the identifier (on the left) and functional location (on the right) for each sensor generating messages.*

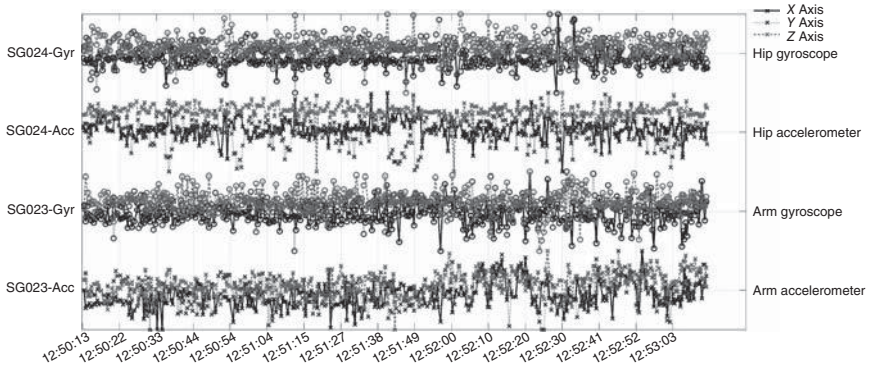


FIGURE 3.12 *Time plot of normalized accelerometer readings during the Sweeping activity. The x axis shows the time of day when the sensor message was generated. The y axis shows the identifier (on the left) and placement (on the right) for each sensor. There are three lines for each sensor, corresponding to the x, y, and z axes.*

raw sensor event data generated by sensors we reviewed in this chapter. Instead, the learned model is often a function of higher-level features that are used to describe the sensor data. As a result, the quality of the learned model varies with the expressiveness of the input feature vector. Because so many varied sensor platforms have been used to model activities, there is a wealth of possible features available to represent an activity. In this discussion, we categorize sensor data features into four groups: (i) Features describing characteristics of the sensor event sequence, (ii) Features describing characteristics of discrete sensor values, (iii) Features describing characteristics of continuous sensor values, and (iv) Activity context.

We will use a subsequence of sensor events from the Sweeping activity example provided in Appendix 1 to illustrate the methods of calculating feature values from sensor data. The corresponding sample of sensor events is shown in Table 3.1.

3.3.1 Sequence Features

We assume in this book that activities are modeled based on a sequence of sensor events. The sequence may represent an entire activity occurrence if the data have been pre-segmented into individual, nonoverlapping activity streams. On the other hand, the sequence may also represent a captured window of time if activities are learned or recognized in real time as sensor events arrive. In either situation, there exist characteristics of the sequence itself that can be useful for modeling activities. We describe such sequence features here.

Time Humans and animals are creatures of habit. As a result, many of our daily activities revolve around specific times. For example, we typically go to sleep in the evening, wake up in the morning, and eat at equally spaced intervals throughout our waking hours. Because individuals follow fairly similar schedules, the notion of time allows us to coordinate group activities such as meetings and sporting events around time units such as the time of the day, the day of the week, the day of the month, and the date. For this reason, the time of the event sequence represents an important feature. A challenge is deciding the precision of time that should be included in the feature vector. The sequence could be represented by the date, the day of the month, the day of the week, and the wall clock time with any level of granularity. For our Sweeping example, we capture time in terms of three granularities: the hour of the day, the number of minutes past midnight on the current day, and the number of milliseconds past midnight on the current day. For sliding-window methods, the focus is on the last sensor event in the sequence and the events before this provide context for the last sensor event. As a result, we capture these time features for the last event in the sequence.

Example 3.1 The time features for our sample activities are:

Hour: 12
Minute: $(12 * 60) + 50 = 770$
Milliseconds: $((((12 * 60) + 50) * 60) + 22) * 1000) + 8739 = 46,230,739$

Sequence Size The size of the sequence can provide insights on the type of activity that is being performed and how it is being performed. For pre-segmented data, the sequence size not only provides an indication of the activity that is being performed but also how much time is being taken to complete the activity.

Example 3.2 As an example, we see that the size of the Sweeping activity is much longer in both duration and number of sensor events than the Hand Washing activity.

TABLE 3.1 Sample of Window Events from the Sweeping Activity

Time	ID	Message
12:50:13.102682	Dustpan	MOVED
12:50:13.149904	HIP	(1019,1921,1520,1850,1800,1898)
12:50:13.367222	ARM	(1584,2402,2318,2040,1838,1736)
12:50:14.128257	HIP	(973,1951,1545,1839,1918,1900)
12:50:14.217971	Duster	MOVED
12:50:14.357487	ARM	(948,1851,1837,1886,1820,2028)
12:50:15.129119	HIP	(1055,1745,1792,1840,1814,1872)
12:50:15.873883	ARM	(926,1867,2119,1751,1853,1863)
12:50:16.134036	HIP	(1047,2137,1487,1819,1594,1992)
12:50:16.235281	Broom	MOVED
12:50:16.36758	ARM	(1055,1677,2182,1705,1613,1862)
12:50:17.001771	M018	ON
12:50:17.11736	HIP	(997,1970,1625,1752,1438,1907)
12:50:17.345406	ARM	(1189,1935,2232,1840,1682,1813)
12:50:18.119112	HIP	(1072,2052,1559,1880,1796,1949)
12:50:18.341054	ARM	(1071,1889,1978,1780,1721,1982)
12:50:18.452039	M017	ON
12:50:18.790212	HandSoap	MOVED
12:50:19.103787	HIP	(1159,2026,1598,1863,1744,1951)
12:50:19.349545	ARM	(1043,1330,1688,1676,1825,1872)
12:50:20.101233	HIP	(986,1966,1573,1838,1727,1921)
12:50:20.334268	ARM	(1007,1674,1700,1702,1868,1916)
12:50:20.741536	HandSoap	STILL
12:50:21.097739	HIP	(981,2074,1527,1853,1724,1930)
12:50:21.342635	ARM	(1208,2346,1888,2217,1234,1483)
12:50:21.895284	Burner	2.8227
12:50:22.090522	HIP	(1149,2003,1701,1857,1615,1949)
12:50:22.339887	ARM	(1134,2594,2088,1894,1845,1872)
12:50:22.412985	HandSoap	MOVED
12:50:22.8739	Dustpan	STILL

If the length of the sliding window is fixed, then measuring the sequence size in terms of duration provides an indication of how much movement and interaction is occurring (or not occurring) during the window. If the time duration of the window is fixed, then the corresponding sequence length serves as this indicator. For our example, we are using a sliding window of length = 30 events, so we add a sequence size feature that represents the time duration, or the time stamp of the last event in the sequence minus the time stamp of the first event in the sequence. As with the *time* feature, the precision of the time duration needs to be determined. We add the sequence duration feature to our vector in terms of the milliseconds that elapsed during the sequence window. For our example, duration is calculated using the smallest measured time unit, milliseconds. The time-based features are summarized in Table 3.2.

$$\text{Duration: } 12:50:22.8739 - 12:50:13.102682 = 46{,}230{,}739 - 46{,}214{,}027 = 16{,}712$$

TABLE 3.2 Window-Based Features for the Sweeping Activity

Window-Based Features			
Hour	12	Milliseconds	46,230,739
Minute	770	Duration	16,712

3.3.2 Discrete Event Features

In some settings, sensors report activity behavior as a well-defined event using a string-based message rather than a numeric value. Many of these sensors also report messages only when events are detected, rather than sampling the state of the sensor at near-constant time intervals. As an example, motion sensors may report a message of "ON" when movement is detected in their field of view, and report "OFF" when movement ceases to be detected. These discrete-value sensor messages can be converted to numeric-value features by mapping string messages onto a set of binary-valued numeric values. Event-based sensors could also emulate sampling-based sensors by reporting their state at constant time intervals. In this case, the motion sensor would report "ON" at every sampling time interval if movement has been detected within the time interval, otherwise it would report "OFF" for the current time interval. As discrete-value, event-based sensors offer convenience in terms of simple representations with minimal reported and stored messages, sensor features can be incorporated that are calculated based on these types of sensors. Here we describe common features used for this purpose.

Bag of Sensors The bag-of-sensors feature is analogous to the bag-of-words model commonly used in text or image mining. In text mining, a document can be represented as a set of words that appear in the document together with the associated word frequencies in the document. The representation is independent of the sequence of words in the document, thus the words and frequencies are thrown together into a "bag." We can use a bag-of-words approach for discrete, event-based sensors by assembling the list of sensors that fall into this category together with the number of times they generate messages in the current window. The number of messages they generate will vary according to the sensitivity of the sensor. For example, the motion sensors used in our Sweeping example are designed to generate an "ON" message when movement is detected corresponding to an individual weighing at least 40 pounds. The sensor generates an "OFF" message when no movement from the individual has been detected for 1.25 seconds. If the sensitivity remains consistent across all possible activities, the bag-of-sensors model can be effective at modeling activities and is appealing in its simplicity.

Note that a visualization of the bag of sensors feature produces a heat map showing how a particular smart environment was utilized over a period of time, as shown in Figure 3.13 (left). For this type of heat map, the pixel intensities or choice of colors indicates the amount of messages or the recency of messages that were generated by the corresponding sensors. The bag of sensors feature is analogous to the motion

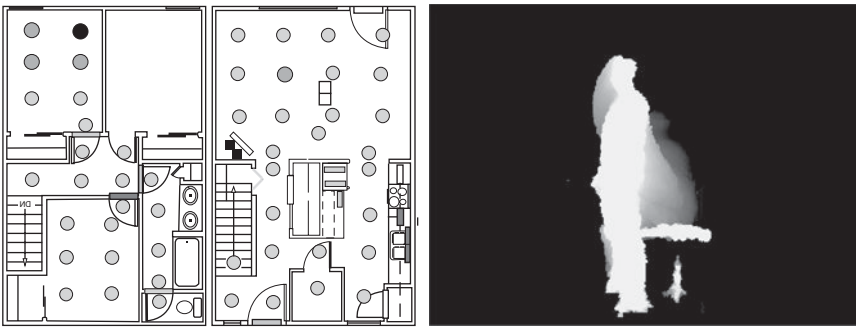


FIGURE 3.13 *Heat map of discrete event sensor usage in a smart home (left, motion sensor locations represented by circles in the image) and motion history image of a sit-to-stand transition movement (right). In both images, the pixel intensities indicate activities occurring in the corresponding region of the space. In the left picture, darker circles indicate areas in which more time was spent. In the right picture, lighter regions indicate more recent movement occurring in the region.*

history images used in vision-based activity recognition. As shown in Figure 3.13 (right), the pixel intensity can be a function of the recency of motion in a sequence of the amount of motion detected in the corresponding region. In both cases, the feature provides a static snapshot of recent activity that can be used to compare with templates for known activities.

Example 3.3 For our Sweeping activity example, we add sensor counts for the event-based sensors, which include vibration sensors (for the dustpan, duster, broom, and hand soap objects), infrared motion sensors (`M017` and `M018`), and the burner sensor. Although the burner sensor reports a numeric value, it is calibrated to generate a message only when the change in burner level exceeds a threshold amount and therefore can be appropriately included in the sensor counts. We do not count sensor events that contain messages corresponding to a lack of activity, such as motion sensor "`OFF`" and object "`STILL`" messages. The additions to our feature vector are listed in Table 3.3. As can be seen from this example, there are many features generated by a bag-of-sensors model and the resulting vector will often be sparse. This situation can be addressed through feature selection or dimensionality reduction techniques, described in Chapter 4.

Elapsed Sensor Time Another feature that can be valuable for discrete, event-based sensors is to calculate the time that has elapsed for each such sensor since it last generated a message. The delay can be calculated based just on events found within the current window or can extend to earlier windows as well. This type of feature provides context for the current activity. In cases where the current window contains more than one activity, it can also provide information on how relevant messages early in the window are to the activity occurring at the end of the window. For example, if an individual took a nap in a chair and a few hours later got

TABLE 3.3 Discrete-Based Features for the Sweeping Activity

Discrete-Based Features			
Sensor Counts		Sensor Elapsed Times	
Dustpan	1	Dustpan	97,712
Duster	1	Duster	86,559
Broom	1	Broom	66,386
Hand soap	2	Hand soap	40,837
`M017` (Kitchen)	2	M017 (Kitchen)	44,219
`M018` (Sink)	1	M018 (Sink)	58,721
Burner	1	Burner	9,786
All other sensors	0	All other sensors	?

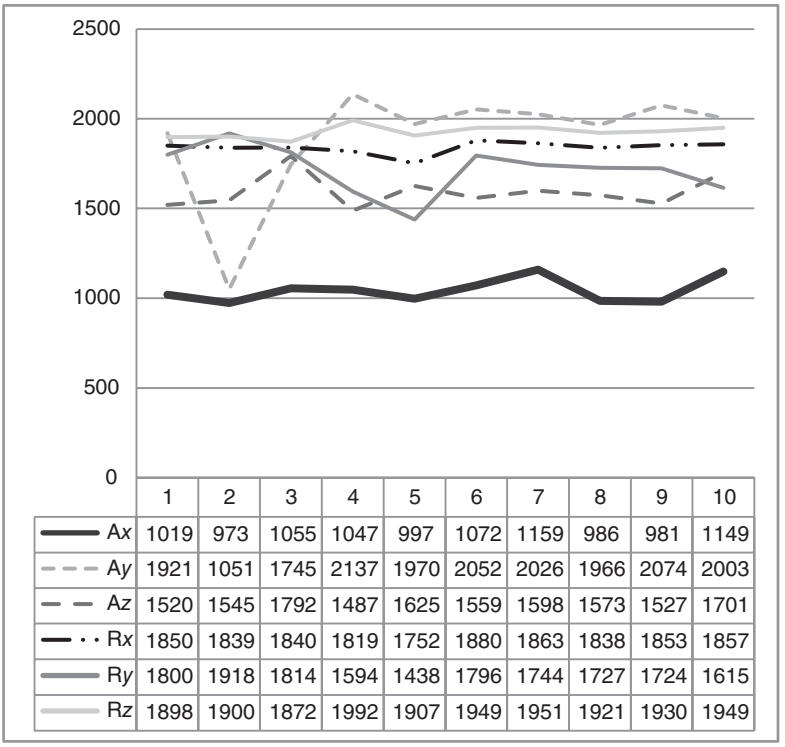
up and left the home, a sequence window could contain sensor messages from both activities. However, the time that elapsed for sensors near the chair will be much greater than those near the front door. The elapsed sensor time feature will capture this relevant information that can be used to more accurately model the Nap and Leave Home activities.

Example 3.4 In our Sweeping example, we add elapsed sensor times for all of the discrete event-based sensors. For those sensors that do not occur within the current window, we can report the values as missing, we can calculate the time since they fired before the current window, or we can use a time value that is greater than the current window duration. Here we report the elapsed sensor times, in milliseconds, only for those event-based sensors that appear in the current window. The elapsed sensor times for all other sensors are reported as missing, denoted by "?". As before, we consider events that indicate activity and not a lack of activity.

3.3.3 Statistical Features

While some features generate messages only when a particular condition is detected, many others report numeric state values at near-constant time intervals. Such data constitutes a sequence of sensor values measured at uniform time intervals, or a time series. As a result, many of the methods used to extract features from time series data can be used in this context. Examples of this type of sensor that are used to monitor activities include accelerometers, gyroscopes, light sensors, temperature sensors, and power usage monitors.

Example 3.5 In our Sweeping activity data, the "`ARM`" and "`HIP`" sensors are accelerometers that reported acceleration and rotational velocity in the x, y, and z axes at uniform time intervals. We use the "`HIP`" accelerometer data to illustrate the feature calculations and add the results to our activity feature vector. Plots of the values for these axes over our sample window are shown in Figure 3.14. Note that sensor data from discrete event sensors can be converted to sampling-based messages by forcing them to periodically report their current state. The state may correspond to



	1	2	3	4	5	6	7	8	9	10
Ax	1019	973	1055	1047	997	1072	1159	986	981	1149
Ay	1921	1051	1745	2137	1970	2052	2026	1966	2074	2003
Az	1520	1545	1792	1487	1625	1559	1598	1573	1527	1701
Rx	1850	1839	1840	1819	1752	1880	1863	1838	1853	1857
Ry	1800	1918	1814	1594	1438	1796	1744	1727	1724	1615
Rz	1898	1900	1872	1992	1907	1949	1951	1921	1930	1949

FIGURE 3.14 *Plot of acceleration (Ax, Ay, Az) and rotational velocity (Rx, Ry, Rz) values for HIP accelerometer from the Sweeping activity data sample.*

the most recent message and persists until an event occurs that generates a different message.

- *Max*, *Min*. These values, as the others listed below, are calculated from the sensor values over the specific window of time. The values are calculated separately for each feature dimension. We let S refer to the set of values for a particular sensor feature and $s_i \in S$ refers to one particular value in the set. In our example, there are six such features being considered. $Max(S)$ thus refers to the maximum of the numbers $s_i \in S$ and $\text{Min}(S)$ refers to the minimum value in the set of numbers. These features can help distinguish between activities that are characterized by different ranges of movements.
- *Sum*, *Mean*. As before, these are computed based on the set of values for a particular sensor feature found in the current window. In these definitions, we let N refer to the number of values found in the set S for the current window.

$$\text{Sum}(S) = \sum_{i=1}^{N} s_i \tag{3.1}$$

$$\text{Mean}(S) = \bar{s} = \frac{\text{Sum}(S)}{N} \tag{3.2}$$

- *Mean Absolute Deviation*, *Median Absolute Deviation*, *Standard Deviation*. While mean absolute deviation (and the related measure of standard deviation) provide a measure of the variability of the sample, in some cases the median absolute deviation provides a more robust measure. This is particularly true in situations where outliers can bias the mean value.

$$\text{MeanAbsDev}\ \ (S) = \frac{1}{N}\sum_{i=1}^{N}|s_i - \bar{s}| \tag{3.3}$$

$$\text{MedAbsDev}(S) = \frac{1}{N}\sum_{i=1}^{N}|s_i - \text{Median}(S)| \tag{3.4}$$

$$\text{StDev}(S) = \sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(s_i - \mu)^2} \tag{3.5}$$

- *Coefficient of Variation*. Another indication of variability in the values is the coefficient of variation. This measure shows the relationship of the standard deviation to the mean of the set of values. This value is typically only computed for features that do not have negative values, as it may not have useful meaning for features with negative values.

$$\text{CV}(S) = \frac{\sigma}{\mu} \tag{3.6}$$

- *Zero Crossings*. Zero crossings are typically calculated only for time series data with a mean of 0. However, zero crossings can be computed for an arbitrary sequence of values to represent the number of times the signal crosses its median. This feature is useful for distinguishing between activities involving rapid and slow movements (e.g., distinguishing Walking from Running).

Example 3.6 In the Ax values for the Hip accelerometer shown in Figure 3.14, this happens between points 2 and 3 ($973.00 < 1034.50 < 1055.00$), points 5 and 6 ($1047.00 > 1034.50 > 997.00$), points 6 and 7 ($997.00 < 1034.50 < 1072.00$), points 8 and 9 ($1159.00 > 1034.50 > 986.00$), and between points 10 and 11 ($981.00 < 1034.50 < 1149.00$), so the number of zero crossings is 5.

$$\text{ZC}(S) = |s_i < \text{Median}(S) < s_{i+1}| + |s_i > \text{Median}(S) > s_{i+1}| \tag{3.7}$$

- *Percentiles*, *Interquartile Range*. If the set of values for a particular feature is sorted in nondecreasing order, then the values that appear at various points along the sorted list provide insights in how the values are distributed over the sequence. A percentile reports a value below which a specified percentage of the feature values fall. Percentiles provide insights on how the values are distributed across the sequence.

Example 3.7 For our sample dataset, we report the 20th, 50th, and 80th percentiles:

$$Pt(S, 20) = 981.01,\ Pt(S, 50) = 1019.01, \text{Pt}(S, 80) = 1072.01$$

Based on these percentiles, the interquartile range can be calculated, which is the difference between the 75th and 25th percentiles. For our Sweeping activity example, the interquartile range is

$$\text{IQ}(S) = 1072.01 - 981.01 = 91.00$$

- *Square Sum of Percentile Observations*. Once the percentiles are defined, the square sum of observations that fall below each percentile (or alternatively, above the percentile) can be reported as a separate feature.

Example 3.8 The corresponding square sum of observations for our example is:

$$\text{SqSumPt}(S, 20) = 1,909,090, \quad \text{SqSumPt}(S, 50) = 4,913,656,$$
$$\text{SqSumPt}(S, 80) = 8,272,047$$

- *Binned Distribution*. This measure represents the fraction of values that fall within equally-spaced bins that span the entire range of sensor values. Binning also provides a mechanism for mapping continuous-valued attributes onto discrete values, by mapping the original value onto a range number or bin number into which the value falls.

Example 3.9 For our example, we consider three equally spaced bins. For k bins, the upper bin margins are defined as shown in Equation 3.8.

$$\bigcup_{j=1}^{k} \frac{j \times (\text{Max}(S) - \text{Min}(S))}{k} \tag{3.8}$$

In our scenario, then, the three bins represent the values ranges [973.00 .. 1035.00], (1035.00 .. 1097.00], and (1097.00 .. 1159.00]. The fraction of values that falls into these bins, or the binned distribution, is 0.5, 0.3, and 0.2.

- *Skewness*. This feature provides an indicator of the degree of asymmetry in the distribution of values for the feature. The greater the skewness, the greater is the lack of symmetry below and above the sample mean. A symmetric dataset will have skewness near 0.

Example 3.10 Applying Equation 3.9 to our sample, we see that the skewness of the data sample is 0.83. This indicates that the data are fairly asymmetric. Looking at the values plotted in Figure 3.14, we observe that in fact about 2/3 of the data points fall below the mean value for , so data are skewed toward lower values.

$$\text{Skewness}(S) = \frac{\frac{1}{N}\sum_{i=1}^{N}(s_i - \mu)^3}{\left(\frac{1}{N}\sum_{i=1}^{N}(s_i - \mu)^2\right)^{\frac{3}{2}}} \tag{3.9}$$

- *Kurtosis*. Similar to skewness, kurtosis provides an indication of the shape of the feature value distribution. While skewness considers the symmetry of the distribution, kurtosis considers the amount of peakedness (or conversely, the amount of flatness) of the distribution toward the mean. Equation 3.10 provides the formula for calculating kurtosis. If the kurtosis is high then the distribution has a distinct peak near the mean and decline quickly with heavy tails. A normal distribution, for example, has a kurtosis of 3, while a uniform distribution has a kurtosis close to 0. In contrast, distributions with low kurtosis have a flat top near the mean.

Example 3.11 We observe as shown in Figure 3.14 that the data are not concentrated around the mean, but have a rather flat distribution. Applying Equation 3.10 to our data, we get a kurtosis of −0.48, which is consistent with this observation.

$$\text{Kurstosis}(S) = \frac{\frac{1}{N}\sum_{i=1}^{N}(s_i - \mu)^4}{\left(\frac{1}{N}\sum_{i=1}^{N}(s_i - \mu)^2\right)^3} - 3 \tag{3.10}$$

- *Correlation*. The amount of correlation that exists between multiple sensors or between the dimensions of a multidimensional sensor, such as a multiple-axis accelerometer, can provide important insights on the type of activity that is being monitored. For example, an accelerometer on a hand may see similar acceleration along all three dimensions if the individual is swimming, but may see little or no acceleration for one axis while the others axes do indicate acceleration for actions such as wiping a table surface. Correlation is measured here between two dimensions at a time, S and V, and can be calculated between all pairs of dimensions.

Example 3.12 Using Equation 3.11, we calculate the correlation between acceleration in the x direction and acceleration in the y direction for our sample data as 0.37. This value indicates that there is a slight, but not strong, positive correlation between the two dimensions and therefore when one changes value the other will tend to change value in the same direction.

$$\text{Corr}(S, V) = \frac{\sum_{i=1}^{N}(s_i - \overline{s})(v_i - \overline{v})}{\sqrt{\sum_{i=1}^{N}(s_i - \overline{s})^2 \sum_{i=1}^{N}(v_i - \overline{v})^2}} \tag{3.11}$$

TABLE 3.4 Continuous-Based Features for Sweeping Activity Hip Accelerometer Data

Hip A*x* Continuous-Based Feature Values for Sweeping Activity					
Max	1,159.00	CV	0.09	SqSumPt(80)	8,272,047
Min	973.00	ZC	5	BD(1)	0.5
Sum	10,438.00	PT(20)	981.01	BD(2)	0.3
Mean	1,043.00	PT(50)	1019.01	BD(3)	0.2
Median	1,033.00	PT(80)	1072.01	Skewness	0.83
MeanAbsDev	19.12	IQ	91.00	Kurtosis	−0.48
MedAbsDev	25.50	SqSumPt(20)	1,909,090	Corr(A*x*,A*y*)	0.37
StDev	66.98	SqSumPt(50)	4,913,656	AC_1	−0.20

- *Autocorrelation*. Just as correlation can be measured between two different dimensions, so also correlation can be calculated for a single dimension but displaced in time. By comparing sensor values with the values that occur at previous time steps, we can observe the amount of persistence in the data, or the tendency for the system to remain in the same state over a period of time. While any time delay can be considered, in Equation 3.12 we compute autocorrelation for a lag of 1, which compares a data point at time i with the data point at time $i + 1$. While an autocorrelation value approaching 1 indicates a high amount of persistence, a value close to 0 indicates near-randomness in the values.

Example 3.13 For the A*x* dimension of the sweeping activity sample, the lag−1 autocorrelation is −0.20. As this is for an accelerometer worn on the hip, the value may indicate many shifts between starting and stopping movement or changes in direction, both of which are common movement patterns while sweeping a floor.

$$\mathrm{AC}_1(S) = \frac{\sum_{i=1}^{N-1}(s_i - \overline{s})(s_{i+1} - \overline{s})}{\sum_{i=1}^{N}(s_i - \overline{s})^2} \tag{3.12}$$

Table 3.4 summarizes the continuous-based sensor features that we extracted from the raw hip accelerometer data for the Sweeping activities. We next turn our attention to features commonly used in signal processing of continuous-valued data sources.

- *Signal Energy*. Signal energy refers to the area between the signal curve and the time axis. For sensor values, this can be computed as the sum of the squared values. Related features are *log energy*, which is the sum of the log squared values, and *signal power*, which is a time-based average of energy.

$$\mathrm{E}(S) = \sum_{i=1}^{N} s_i^2 \tag{3.13}$$

$$\text{Log}E(S) = \sum_{i=1}^{N} \log(s_i^2) \tag{3.14}$$

$$\text{Power}(S) = \frac{1}{N} \sum_{i=1}^{N} s_i^2 \tag{3.15}$$

- *Signal Magnitude Area*. For sensors with multiple dimensions, the sensor values may be summed over all of the dimensions and averaged over time (in our case, over the current window). Equation 3.16 shows how to calculate the signal magnitude area for one such sensor, an accelerometer with x, y, and z dimensions.

$$\text{SMA}(S) \quad = \sum_{i=1}^{N} (|x_i| + |y_i| + |z_i|) \tag{3.16}$$

- *Peak-to-Peak Amplitude*. This value represents the change between the peak (highest value) and trough (lowest value) of the signal. For sensor values, we can compute the difference between the maximum and minimum values of the set.

$$\text{P2PA}(S) = \text{Max}(S) - \text{Min}(S) \tag{3.17}$$

- *Time Between Peaks*. This value represents the time delay between successive occurrences of a maximum value. When processing sensor values that are not strictly sinusoidal signals, special attention must be paid to determine what constitutes a peak. A peak may be a value within a fixed range of the maximum value, or it may be a spike or sudden increase in values. The *number of peaks* in the window for different thresholds of the peak detector may also be included as a separate feature value.

Example 3.14 Consider the data plotted in Figure 3.14. In this case, the values at time points 3, 7, and 10 could be considered peaks. The number of peaks = 3 and time between peaks = $\{4, 3\}$ could be added to our feature vector.

Table 3.5 summarizes the signal-based sensor features that we extracted from the raw hip accelerometer data for the Sweeping activity. These features are commonly extracted from continuous-valued, sampled data sources.

3.3.4 Spectral Features

Many sensors used in activity monitoring are typically viewed as a signal, or a function of varying amplitude over time. The frequency spectrum provides an alternate view of the signal. This view shows how much of the signal lies within each given frequency band over a range of frequencies. A commonly used mathematical operator for transforming the time varying signal into its frequency spectra is the Fourier Transform, which is efficiently implemented using the Fast Fourier Transform (FFT). This transform decomposes the signal into the sum of a number of sine wave frequency components.

TABLE 3.5 Signal-Based Features for Sweeping Activity

Ax Signal-Based Features for Sweeping Activity			
Signal energy	10,935,556.0	P2PA	186.0
Log signal energy	200.5	TBPeaks	3.5
Power	1,093,555.6	NumPeaks	3

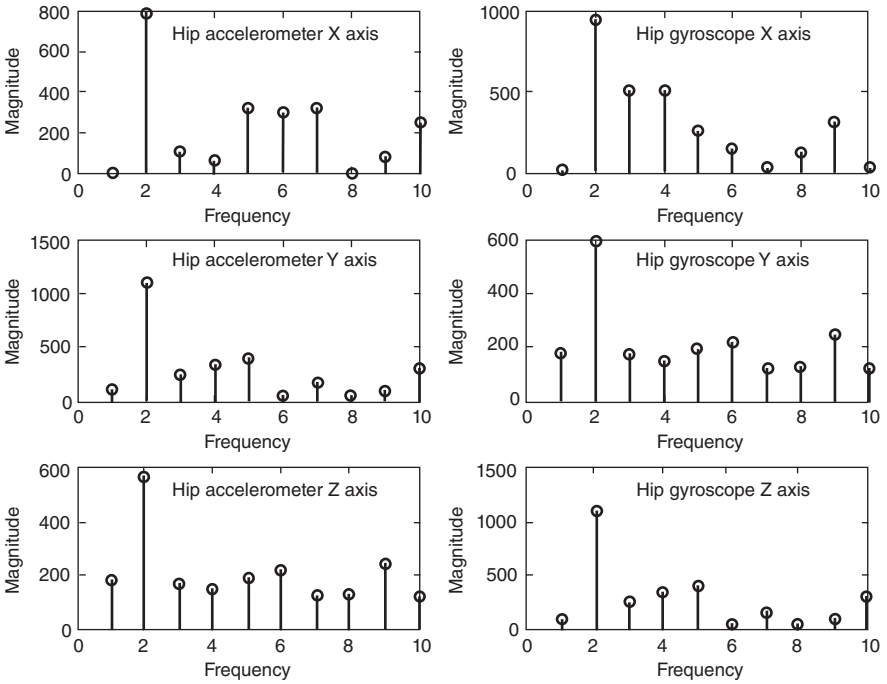


FIGURE 3.15 *Frequency content in the accelerometer and gyroscope signal for the sensor placed on the hip while performing a Sweeping activity.*

Example 3.15 Figure 3.15 illustrates the frequency content for our working example consisting of accelerometer and gyroscope data from the sensor located on an individual's hip. The corresponding spectral feature values are provided in Table 3.6. If we consider a window of 10 sensor readings, the frequency spectrum covers 10 frequencies. Note that converting the signal into its frequency components hides information regarding the frequency changes over time. Therefore, the time series signal is divided into short-duration overlapping windows from which the frequency content is extracted. More expressive features characterizing the signal can be extracted from the frequency spectra. We highlight a few such features here.

Let $\mathcal{F}$ represent the frequency spectrum for the window of sensor readings, s, of length N. Then $\mathcal{F} = \text{FFT}(s)$, which estimates the frequency content in the signal by

applying FFT. Here, $\mathcal{F}(n)$ corresponds to the magnitude of the n^{th} frequency component.

- *Spectral Centroid*: This corresponds to the balancing point or the center of mass of the spectral power distribution. This value is calculated as the weighted mean of the frequency components present in the signal, as shown in Equation 3.18.

$$S_C = \frac{\sum_{i=1}^{N} \mathcal{F}(n) \times n}{\sum_{i=1}^{N} \mathcal{F}(n)} \tag{3.18}$$

- *Spectral Energy*: The equivalent to the energy of the signal as discussed before in the frequency domain is the spectral energy. This value is computed as the sum of the squares of the magnitude of the frequency content, as shown in Equation 3.19.

$$S_E = \sum_{i=1}^{N} \mathcal{F}(n)^2 \tag{3.19}$$

The energy can also be computed from the normalized frequency spectrum, or $\widehat{\mathcal{F}}(n) = \frac{\mathcal{F}(n)}{\sum_{i=1}^{N} \mathcal{F}(n)}$, in which case the equation is modified as shown in Equation 3.20.

$$NS_E = \sum_{i=1}^{N} \widehat{\mathcal{F}}(n)^2 \tag{3.20}$$

- *Spectral Entropy*: This refers to the entropy of the signal in the frequency domain. The frequency content of the signal is normalized before the entropy computation. Let $\widehat{\mathcal{F}}(n) = \frac{\mathcal{F}(n)}{\sum_{i=1}^{N} \mathcal{F}(n)}$ represent the normalized frequency spectrum. Then spectral entropy is calculated as shown in Equation 3.21.

$$S_{EN} = -\sum_{i=1}^{N} \widehat{\mathcal{F}}(n) \times \log(\widehat{\mathcal{F}}(n)) \tag{3.21}$$

As with the other features, we extract the spectral features for our Sweeping activity example. Table 3.6 summarizes these spectral sensor features.

TABLE 3.6 Spectral Features for Sweeping Activity

Spectral Features from Hip A*x* Data for the Sweeping Activity Window of 10 Events			
Spectral centroid	4.9550	Spectral energy	1007864
Spectral entropy	1.8238	Normalized spectral energy	0.2002

3.3.5 Activity Context Features

All of the features that we discussed in this chapter are extracted from a single window of sensor events. However, the context that is defined from events outside the window can be influential as well in modeling and learning the current activity.

- *Previous Activity*. The activity label for the previous window can be very helpful in understanding the activity in the current window. For example, opening the front door often signifies that an individual is returning home if the previous activity was that person leaving home. The difficulty here is knowing the correct activity label for the previous window. The output of an activity recognition algorithm can be used here, although if the label is incorrect than the error will propagate to the feature vector describing the current window. Another approach is to input a probability distribution over possible activity labels for the previous window into the feature vector for the current window.
- *Previous Dominant Sensor*. Another context feature that is helpful is to note the sensor that generated a message most frequently in the previous window. For sampling-based sensors, these could also be the median value of the sensor messages.
- *Weighted Features*. In theory, all of the features extracted from earlier windows can be used as input to the current feature vector. However, the number of such windows may not be fixed and the resulting feature vector will be quite large. An alternative approach is to compute aggregates of the features from previous windows, such as a weighted sum of values. The values can optionally be weighted by a decay factor based on how far back in time the value was observed.

3.4 MULTISENSOR FUSION

Multisensor fusion refers to the synergistic combination of sensory data from multiple sensors to provide more reliable and accurate information. The potential advantages of multisensor fusion are redundancy, complementarity, and reduction in uncertainty of the information. The integration or fusion of redundant information can reduce overall uncertainty and thus serve to increase the accuracy with which the information is perceived by the system. Multiple sensors providing redundant information can also serve to increase reliability in the case of sensor error or failure. For example, if there are multiple wearable accelerometers that are gathering movement data related to activities, multisensor fusion aids in scenarios when some of the accelerometers fail or provide noisy data. The raw sensor data from all the accelerometers can be fused to obtain a more reliable estimate of the movement measurement.

Complementary information from multiple sensors allows features in the environment to be perceived that are difficult or impossible to perceive using just the information from each individual sensor operating separately. For example, in a smart home setting multiple sensors such as motion, temperature, and pressure sensors gather complementary data about a Cooking activity. Motion sensors can provide

data about a human presence in the kitchen area, temperature sensors provide clues to whether the stove is on and pressure or vibration sensors can indicate whether any kitchen objects are being used. While these three sensor classes may independently be weak at characterizing the Cooking activity, fusing them together leads to a stronger model.

Information fusion is worthwhile for reducing uncertainty. Data gathered from sensors can be noisy and incomplete. As a result, activity learning methods that directly utilize raw sensor data are also prone to yield erroneous results. However, when different sensors and methods produce varying levels of errors and each method is reasonably accurate independently, a combination of multiple experts should reduce overall classification error and as a consequence yield more accurate outputs.

Typically, fusion of data/information can be carried out on three levels of abstraction closely connected with the flow of the activity learning process: data, feature, and classifier fusion. Of these, classifier fusion is more popular with the research community, due to sound theoretic principles that support different algorithms.

Data Fusion Multisensor data fusion refers to the stage in the integration process where there is an actual combination of different sources of sensory information into one representational format. Given a group of sensors, a weighted average of the redundant sensor data provides a simple approach to fuse the events generated by different sensors. Let x_1 and x_2 represent measurements from two sensors. The noise variances of the two sensors can be represented as σ_1^2 and σ_2^2, respectively. The fused sensor value that is generated by a weighted average algorithm can be calculated as shown in Equation 3.22. In this equation, $\sigma_3^2 = (\sigma_1^{-2} + \sigma_2^{-2})^{-1}$ represents the variance of the combined estimate.

$$x_3 = \sigma_3^2(\sigma_1^{-2}x_1 + \sigma_2^{-2}x_2) \tag{3.22}$$

While this is a simple approach that employs the noise variances of the sensors as the weights, applying a Kalman filter is more often the preferred approach as it results in estimates for the fused data that are optimal in a statistical sense. Additionally, the Kalman filter does not incur heavy computational expense. Kalman filtering assumes that the sensors can be modeled as a linear system and sensor noise can be modeled as a Gaussian distribution. The algorithm operates using two steps. In the first step, referred to as the prediction step, the Kalman filter estimates the current sensor values along with their uncertainties. Once the actual sensor values are observed, the fused estimate is obtained by the weighted average of the sensor values, with more weights given to the values with higher certainty. Kalman filtering can be performed in real time using the current sensor measurements and the previously-calculated sensor states with their uncertainty values.

Feature Fusion Feature fusion refers to the process of combining features extracted from multiple sensors. The naïve technique for fusing features from multiple sensors is to concatenate the features extracted from each sensor into

a single, longer feature vector. This is an effective technique when the multiple sensors and features extracted from the raw sensor data are independent of each other. When the features extracted from different features are not independent, then dimensionality reduction or feature selection techniques that are discussed in Chapter 4 can be used to fuse the different features.

Classifier Fusion At the third level of sensor fusion, multiple individual classification models are first learned using data from each sensor independently. In the second step, the individual classifier outputs are combined to determine the actual output value. This type of fusion is also referred to as decision fusion or mixture of experts, and has been investigated with a number of proposed approaches. There are two common approaches for fusing output from the classifiers. The goal of the first approach is to find a single best classifier or a group of best classifiers whose outputs are then considered for making the final prediction. The second group of methods perform simple fusion functions. Such functions operate directly on the outputs of the individual classifiers to determine the best combination to make the prediction.

One popular technique, Dynamic Classifier Selection (DCS), extracts the single best classifier that is most likely to produce the correct classification label for a particular data point (such as a sensor event sequence). DCS partitions samples from sensor training data and selects the best classifier for each partition. Given a data point, we first determine the most likely partition to which it belongs and generate its label using only the output of the best classifier for the corresponding partition. The idea of using partition-based DCS is to estimate each classifier's accuracy in a local region of the feature space and generate a final output from the most locally accurate classifier. Different metrics can be employed for partitioning the training samples. These include partitioning samples into a subgroup for which there is agreement between the classifier output and the ground truth and a subgroup for which there is no such agreement and groupings based on the features of the input samples. DCS approaches strongly rely on available sensor event training data and hence can be unduly influenced by noisy data samples. Furthermore, choosing only the locally best classifier will result in loss of additional information that could have been gained from other classifiers for making a more accurate prediction.

Simple fusion functions are the most obvious choice when constructing a multiple classifier system. Common fusion functions are listed below.

- *Simple Voting Strategy*. Voting strategies can be applied to a multiple classifier system assuming that each classifier gives a single class label as output. Voting strategies include unanimous voting in which all classifiers agree on the same class label, simple majority in which more than half the number of classifiers agree on the same class label, and majority voting in which the class label is a label that is most often predicted by the classifiers.
- *Weighted Majority Voting*. If the classifiers in the ensemble do not have identical performance, then classifiers that are more accurate are assigned higher weights when making the final prediction. The accuracy of the classifiers can be used to determine the weights.

- *Highest Average Classification Probability*. When the classifiers also output posterior classification probabilities, a simple strategy is to pick the class label with the highest average probability of classification.

Bayesian methods are also applicable for classifier fusion when classifiers output posterior probabilities. For example, the classification output of multiple models can be combined using a naïve Bayes technique as described in Chapter 4. If $S_1, S_2, \ldots, S_K$ represent the classification output of K models that were independently trained using the data from K sensors, the fused prediction C can be estimated as shown in Equation 3.23.

$$P(C|S_1, \ldots, S_K) = \frac{P(S_1, \ldots, S_K|C)P(C)}{P(S_1, \ldots, S_K)} \tag{3.23}$$

Since classification models are independent of each other, we can rewrite this rule as shown in Equation 3.24.

$$C^* = \text{argmax}_c \left\{ P(C = c) \prod_{k=1}^{K} P(S_k = s_k|C = c) \right\} \tag{3.24}$$

The prior $P(C = c)$ can be estimated by computing the distribution of samples in every class c. The posterior likelihood of a particular output can be computed from the confusion matrix of model S_k as shown in Equation 3.24, where t_C is the number of training instances for which both the class $C = c$ and the classifier output $S_k = s_k$. The term t represents the number of total training instances.

$$P(S_k = s_k|C = c) = \frac{t_c}{t} \tag{3.25}$$

Another approach to classifier fusion is to formulate it as a supervised learning problem. This approach simply treats the outputs of multiple classifiers as input to a second-level classifier, and uses classical supervised learning techniques, such as those described in Chapter 4, to generate the final classification output. The selection of the second-level classifier depends on the distribution of its input, which in this case are probabilities that are skewed toward 0 and 1. As a result, algorithms that assume the data follows a normal distribution are not likely to fare well.

Classifier fusion is useful when activity information is gathered from a redundant set of noisy sensors and data from each sensor alone is sufficient to model the activity. The meta-classifier, a classifier that combines the output of multiple classifiers, exploits the redundancy intrinsic in the sensor network. An example of this situation is a setting where there are multiple accelerometers gathering movement information while an individual performs an activity. Accelerometers can be selected according to their contribution to classification accuracy as assessed during system training.

3.5 ADDITIONAL READING

The activity learning literature is filled with case studies that use environmental sensors including motion, temperature, door usage, and lighting sensors. Similarly, accelerometers, gyroscopes, GPS, and magnetometers are ubiquitous in smart phone-based activity modeling. Because of the inherent range limitations in RFID technology, researchers have experimented with individuals wearing an RFID reader such as the iGlove[18] and the iBracelet[19]. Ongoing research also exists that couples programmability with RFID tags in order to imbue them with additional sensing ability.[20] Zhu et al.[21] and Reddy et al.[22] explore the mining of GPS trajectory data to recognize activities, and Pu et al.[23] have tested wireless signals for action recognition.

We focus this discussion on sensors that are most commonly used for activity learning from sensor data. However, the number and variety of sensors that are developed for activity modeling grow as sensor technology matures and the applications that make use of activity learning become more diverse. Some examples are recent research that utilizes the Doppler effect to sense body movement[24] and research that uses voltage and measurement disaggregation to monitor device usage[25,26]. Bulling et al.[27] investigate a novel direction of eye movement analysis as a modality for learning activities. The examples in Appendix A illustrate the use of special-purpose sensors to monitor gas flow and water flow. Infrared proximity sensors are now found on mobile devices and are gaining traction for gesture recognition. Additional types of sensor modalities emerge each year and they can be incorporated into activity models as well.

While this chapter does not discuss video-based approaches to activity learning in depth, many of the techniques described here can be applied to video processing as well. Some additional features have been designed that are fine-tuned for processing information from CCD, depth, or infrared cameras[2,28–30] and from microphones[31]. While these sensors provide a dense grid of information that is valuable for analyzing the activity, the resulting wealth of data may also require substantial preprocessing in order to extract high-level features for modeling activities.

The discussion in this chapter highlights the many diverse sensors and features that can be employed for learning activities. Effective use of this information can depend on subtle choices of how to tune the sensors and utilize their data. For example, many different sampling rates have been reported for continuous-valued sensors. Yan et al.[32] investigate how the choice of sampling frequency and classification features affects accuracy and energy usage in different ways for each activity class. While we describe a method to convert discrete-event sensors to sampling-based sensors, some time series research, such as the work by Cardinaux et al.[33], converts binary-valued times series data to a start time/stop time representation that approximates the representation of the discrete event sensors. Electricity consumption by wireless sensors has also been investigated by Liu et al.[34], who propose a design that allows sensors to be effectively powered by ambient RF in the environment. Sensor placement is another important factor. Alternative placement locations on the body have been explored by Bao and Intille[35] and by Keally et al.[36], while the effect of placement on

activity modeling has been evaluated by Hoseini-Tabatabaei[37] and by Kunze et al[38]. Environmental sensor selection and placement techniques have also been proposed and evaluated by Philipose et al.[39] and by Cook and Holder[40].

In addition to defining features that can be applied to raw sensor data, hierarchical features can also be defined. As an example, raw sensor data can be clustered using a technique such as density-based clustering or k means clustering. Features can be defined that describe the cluster characteristics[41,42]. By transforming the raw data provided by these sensors into higher-level features, the insights gleaned from a tremendous diversity of sensors can be utilized to create more robust models of human behavior.

Luo et.al.[43] describe the three levels for multisensor fusion and discuss techniques for fusion at each level. Durrant-Whyte and Henderson[44] provide a resource on techniques for fusing raw sensor data and Kalman filtering for data fusion. Ruta and Gabrys[45] provide an overview of methods for combining classifier outputs for multisensor fusion. Kuncheva[46] presents a thorough discussion on the theoretical underpinnings of classifier fusion strategies. Gao et al.[47] and Zappi et al.[48] employ multisensor fusion for combining data from multiple accelerometers for recognizing human locomotion. Hong et al.[49] discuss a methodology for fusing information gathered from uncertain smart home sensor readings using Dempster–Shafer theory.

4

Machine Learning

Many machine learning algorithms have been designed, enhanced, and adapted for activity learning. The types of machine learning models that have been used for activity learning vary almost as greatly as the types of sensors from which data is collected. In this chapter, we provide an overview of the basic supervised machine learning algorithms that are most commonly found at the heart of activity learning algorithms. In addition, as indicated by the number of features described in Chapter 3, sensor data can be very high-dimensional in nature. We also discuss techniques to select features and reduce the dimensionality of the data while maintaining or even enhancing the accuracy of learned activity models.

4.1 SUPERVISED LEARNING FRAMEWORK

Supervised learning algorithms are machine learning algorithms that learn a mapping from input data to a target attribute of the data, which is usually a class label. Typically, the algorithm learns a relationship between the input data attributes (features) and the target attribute by minimizing a loss function that is defined on the pairs of input data and the corresponding target attribute. The discovered relationship is represented in a structure known as a model. Models characterize the hidden relationship between the input data and target attribute and can be used for predicting the target attribute (label), given the values of the input data. There are two main categories of supervised learning algorithms: classification and regression. The difference between these categories lies in the nature of the target attribute. In classification, the target attribute takes on categorical values such as class labels. On the other hand, regression models map the input data to a real-valued target attribute. For example, if we

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

consider the activity learning problems, classification can map sensor features to corresponding activity labels. Regression algorithms can in turn map the sensor features to predicted time points in the future when the activity will next occur.

Many human behavioral modeling challenges can be formulated as supervised learning problems. These form a wide spectrum of activity learning tasks such as mapping sensor data to activity labels (activity recognition), detecting the resident who triggered the sensor events in a multi-resident setting (resident identification), detecting the appliances that are being used in a home based on the current power consumption pattern (appliance usage detection) and modeling the health status of an older adult from sensor data. For the remainder of this chapter, we focus on the single task of activity recognition in order to illustrate the learning algorithms.

The primary focus of this chapter will be on classification algorithms. These algorithms fall into the broad categories of generative and discriminative models. Generative models learn the underlying probabilistic activity model that generates the sensor sequences. Examples of these approaches include naïve Bayes Classifiers, Gaussian Mixture models (GMM) and Hidden Markov models (HMM). On the other hand, discriminative models focus on characterizing the region in the feature space that separates the different activity classes. Approaches in this category include support vector machines (SVMs), decision trees, conditional random fields (CRF), and meta-learners such as boosting and bagging. Some of these techniques are particularly well suited to modeling data sequences. HMM and CRF are examples of these sequential models. Before we discuss these individual algorithms, we suggest steps that will assist the reader to formulate an activity learning problem using a supervised learning framework.

Define and Represent the Input Data and the Target Attribute While the target attribute can be defined based on the application, one has to carefully select the features that will represent the input data. The accuracy of any learned model depends heavily on the features that characterize the input data. For example, the age of an individual will likely not provide insights that are useful for classifying the individual's activity as either Cooking or Watching TV. On the other hand, including a feature that identifies individual's location will be more valuable for this task. Typically, the raw sensor event data is transformed into a number of features (as described in Chapter 3) that are descriptive of the target class. However, one has to be cautious here not to have a too large of a feature vector as this can cause a reduction in the predictive power of the model. We will discuss techniques for reducing the size of a feature vector later in the chapter.

Gather the Training Set The training set refers to the set of examples that are used to train the classification algorithm for learning the model. Therefore, it is imperative for the training set to adequately represent the variations resulting from the real-world use of the model. For example, a cooking model that is trained on data gathered from only microwave usage will not perform well when tested on cooking data gathered from stove usage. From a theoretical perspective, this means that the training data and the data on which the learned model is tested must follow similar distributions. The

values of the target attribute that are associated with the training set (i.e., the ground truth labels) must also be obtained from a reliable oracle such as a human annotator or an expert system.

Choose an Appropriate Classification Algorithm The next step is to choose a classification algorithm that is well suited to the structure of the data. For example, if the data is a sequence, appropriate classifiers are HMMs and CRFs. If causality needs to be modeled, then induction algorithms such as decision trees might be useful.

Determine the Evaluation Metric The standard evaluation metric for a supervised learning problem is its classification accuracy. Sometimes one might be interested in a different metric such as the area under the receiver operating characteristic (ROC) curve when dealing with skewed class distributions. Thus depending on the application, a suitable metric has to be chosen for evaluating the performance of the classification algorithm. Some of these evaluation metrics are discussed in Chapter 5.

Learn the Model and Tune the Related Parameters When all of the required pieces are available, the next step is to learn the model using the training data. Many algorithms are driven by parameters, such as the number of mixture components in a GMM or the penalty parameter of a SVM. These values need to be specified based on prior experience, results from preliminary experiments, or theoretical derivations. The parameters can be further tuned by employing procedures such as multifold cross validation (discussed in Chapter 5) in conjunction with an appropriate evaluation metric.

We now define the notations used for discussing general supervised learning algorithms. The variable X is a D-dimensional random variable that corresponds to the features, or attributes, extracted from the sequence of sensor events. Thus $X =< X^1, X^2, \ldots, X^D >$, where X^D represents the D^{th} feature. The random variable Y corresponds to activity labels and can take a value $k \in \{1, 2, \ldots, K\}$ corresponding to the different activity classes. The labeled training set L of size N consists of pairs $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where x_i represents the i^{th} instance sampled from X and y_i is the i^{th} instance of Y, representing the activity label of x_i. Each $x_i =< x_i^1, x_i^2, \ldots, x_i^D >$ is a D-dimensional vector consisting of values for the D attributes. These notations are illustrated in Table 4.1 with a synthetic dataset consisting of examples drawn from the activities Personal Hygiene, Bed Toilet Transition, Take Medicine, and Other, labeled numerically as 1, 2, 3, and 4, respectively. Here we use integers to represent the attribute values and activity labels for the sake of notational convenience. The features used to represent the dataset are presented as rows and the columns represent the individual data instances. The last row of the table shows the labels associated with a particular data instance. Thus instance x_3 can be described by the feature vector $< 1, 20, 5, 6, 200 >$, which contains specific values for the five attributes. The associated activity label is $y_3 = 3$.

Some of the algorithms that are described in this chapter process temporal data sequences. In this case, we refer to $\{x_1, x_2, \ldots, x_T\}$ as a sequence of data points sampled at time steps $1, 2, \ldots, T$ and x_T as the data instance at the T^{th} time step.

TABLE 4.1 A Synthetic Dataset to Illustrate the Notations Used in This Chapter

X	x_1	x_2	x_3	x_4	x_5	x_6	x_7
Day of the week (X^1)	1	1	1	2	2	2	2
Time of the day in hours (X^2)	2	7	20	6	10	14	21
Bathroom sensor count (X^3)	13	10	5	11	3	12	3
Medicine cabinet sensor count (X^4)	0	2	6	1	4	1	9
Energy in the Hip accelerometer Z axis (X^5)	455	500	200	506	207	521	293
Activity label (Y)	2	1	3	1	4	1	3

As for other types of data, the data point at each time step x_T is sampled from the D-dimensional random variable corresponding to the available features. The activity label sequence is then represented as $\{y_1, y_2, \ldots, y_T\}$, where y_T represents the activity label for the data instance x_T that is observed at the T^{th} time step.

4.2 NAÏVE BAYES CLASSIFIER

A naive Bayes classifier (NBC) is a simple probabilistic classifier created by applying Bayes' theorem. To simplify computation, this classifier utilizes the assumption that data attributes are conditionally independent given the class value (activity label). Consider the problem of estimating the conditional probability distribution of activity labels Y over observed features X (i.e., $P(Y|X)$). By applying Bayes' rule, this conditional probability can be estimated as shown in Equation 4.1.

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{4.1}$$

In this equation, $P(Y|X)$ expresses the likelihood of observing a set of feature values for a given activity label, $P(Y)$ represents the prior probability distribution over class values before observing the evidence, and $P(X)$ represents the evidence, or the probability of observing the particular feature vector in the dataset. Computing the likelihood term involves estimating the combination of parameters. The number of combinations grows exponentially with increasing dimension D of the feature vector, which makes such estimation impractical. The NBC overcomes this problem by imposing the conditional independence assumption on the features while modeling $P(Y|X)$. In particular, each of the features is considered to be conditionally independent of the other, given the activity label. The likelihood can thus be computed as the product of the probability estimates for each particular feature value in the vector, as shown in Equation 4.2.

$$P\left(X^1, X^2, \ldots, X^D|Y\right) = \prod_{d=1}^{D} P(X^d|Y) \tag{4.2}$$

Referring to the example in Table 4.1, the NBC treats the contribution of the value of the Bathroom sensor count attribute as independent of the probability that a data

instance corresponds to Personal Hygiene activity, regardless of the values of all of the other attributes.

The primary goal for an NBC is to learn this conditional probability distribution, $P(X^1, X^2, \ldots, X^D|Y = k)$, for each of the K activities. Once the distributions are computed, given a test point $x =< x^1, x^2, \ldots, x^D\}$, the learned model can be used to output the posterior probability distribution over all activity classes. The posterior probability for class k is given by Equation 4.3.

$$P\left(Y = k|x^1, x^2, \ldots, x^D\right) = \frac{P\left(x^1, x^2, \ldots, x^D|Y = k\right) P\left(Y = k\right)}{\sum_j P\left(x^1, x^2, \ldots, x^D|Y = j\right) P\left(Y = j\right)} \tag{4.3}$$

Taking into account the conditional independence assumption, Equation 4.3 simplifies to the calculation in Equation 4.4.

$$P\left(Y = k|x^1, x^2, \ldots, x^D\right) = \frac{\prod_{d=1}^{D} P\left(x^d|Y = k\right) P\left(Y = k\right)}{\sum_j \prod_{d=1}^{D} P\left(x^d|Y = j\right) P\left(Y = j\right)} \tag{4.4}$$

Given a test instance x, Equation 4.4 shows how to calculate the probability that the class value Y will be any one of the K activity classes. If we are using this equation to perform activity classification, we are only interested in the most likely activity label, which can be obtained by selecting the activity label with the maximum probability. This is the naïve Bayes classification rule shown in Equation 4.5.

$$Y = \text{argmax}_k \frac{\prod_{d=1}^{D} P(x^d|Y = k) P(Y = k)}{\sum_j \prod_{d=1}^{D} P(x^d|Y = j) P(Y = j)} \tag{4.5}$$

As the denominator is independent of the possible values of k, this can be simplified to the calculation shown in Equation 4.6.

$$Y = \text{argmax}_k \prod_{d=1}^{D} P(x^d|Y = k) P(Y = k) \tag{4.6}$$

Modeling Nominal Features We initially consider learning the NBC when the features are categorical or nominal in nature. Consider as an example a feature X^i with M possible discrete values, $\{v_1, v_2, \ldots, v_M\}$. In this case, we need to learn the likelihood probabilities, $P(X^i = v_j|Y = k)$, for each value v_j and class label k. These probabilities can be estimated using maximum likelihood estimates or Bayesian MAP estimates.

The maximum likelihood estimates are simply the frequencies of observed events (events that are found in sample training data) that contain the attribute values of interest. They can be easily estimated as shown in Equation 4.7.

$$P(X^i = v_j|Y = k) = \frac{\#\text{data samples with } X^i = v_j \text{ and } Y = k}{\#\text{data samples with } Y = k} \tag{4.7}$$

When the training dataset does not contain samples for all possible values of an attribute, the numerator in the aforementioned equation will be 0. In this case, the likelihood probability will also be 0. Typically, this is avoided by considering a smooth estimate for the probability. This is accomplished by adding terms to the numerator and denominator of the Equation 4.7, resulting in Equation 4.8.

$$P(X^i = v_j | Y = k) = \frac{(\#\text{data samples with } X^i = v_j \text{ and } Y = k) + l}{(\#\text{data samples with } Y = k) + lM} \tag{4.8}$$

In this equation, l corresponds to the value of the smoothing factor. This is essentially the MAP estimate with Dirichlet priors on the parameters characterizing the probabilities. This process is then repeated for each of the nominal attributes. The sample process can also be applied in order to estimate the prior term, $P(Y = k)$, in which case the calculation would be as shown in Equation 4.9.

$$P(Y = k) = \frac{(\#\text{data samples with label } k) + l}{(\#\text{data samples}) + lK} \tag{4.9}$$

An alternative approach is to consider a uniform distribution, thereby making an assumption that each label is equally likely to occur in the dataset.

Example 4.1 To illustrate the use of an NBC, consider the data instances listed in Table 4.1. The attribute X^1, *Day of the Week*, is an attribute that can have a value of 1, 2, 3, 4, 5, 6, or 7. Let us compute the probability of this attribute taking the value 1, given that the activity is Personal Hygiene ($Y = 1$). The number of Personal Hygiene instances are 3 and among them only 1 has the value of 1 for the *Day of the Week* attribute. Therefore, $P(X^1 = 1|Y = 1) = \frac{1}{3}$. The total number of instances is 7, out of which 3 have been labeled as Personal Hygiene. Hence, the prior probability for the activity Personal Hygiene can be calculated as $P(Y = 1) = \frac{3}{7}$. Similarly, $P(X^1 = 1|Y = 3)$ and $P(Y = 3)$ can be computed as $\frac{1}{3}$ and $\frac{2}{7}$, respectively.

Modeling Continuous Features When the features take on continuous values, we must choose a way to represent the feature distributions $P(X^i|Y)$. The typical approach is to model the continuous attribute X^i, for each activity label j, as a Gaussian distribution $N(\mu_j^i, \sigma_j^{2^i})$ with a well-defined mean μ_j^i and variance $\sigma_j^{2^i}$. Thus the goal here is to estimate the mean and variance of these distributions from the training dataset. These values can be estimated using Equations 4.10 and 4.11 for each continuous attribute X^i and each activity label j.

$$\mu_j^i = E[X^i|Y = j] \tag{4.10}$$

$$\sigma_j^{2^i} = E[(X^i - \mu_j^i)^2|Y = j] \tag{4.11}$$

As was the case for discrete features, the maximum likelihood estimation process can be employed to estimate the feature's mean and variance. Given the

training dataset U, the maximum likelihood estimator for the mean μ_j^i is given by Equation 4.12.

$$\mu_j^i \approx \frac{\sum_{m:y_m=j} x_m^i}{\#\text{data samples with activity label } j} \tag{4.12}$$

Similarly, the maximum likelihood estimator of $\sigma_j^{2^i}$ is given by Equation 4.13.

$$\sigma_j^{2^i} \approx \frac{\sum_{m:y_m=j} (x_m^i - \mu_j^i)^2}{\#\text{data samples with activity label } j} \tag{4.13}$$

Note that the prior probabilities for the discrete activity labels have to be estimated and the approach elaborated in the earlier discussion can be employed for this purpose.

Referring again to Table 4.1, let us compute the parameters of the Gaussian distribution for the numerical attribute attribute X^5, *Energy in the Hip Accelerometer Z Axis*, given that the activity label is Personal Hygiene ($Y = 1$). The mean is calculated as $\mu_1^5 = \frac{500+506+521}{3} = 509$ and the variance as $\sigma_1^{2^5} = \frac{(500-509)^2+(506-509)^2+(521-509)^2}{3} = 78$.

Similarly, the Gaussian distribution parameters for the same attribute and activity label Take Medicine can be computed as $\mu_3^5 = 246.5$ and $\sigma_3^{2^5} = 2162.5$.

Now let us compute the label for a data instance with only attributes $x^1 = 1$ and $x^5 = 250$ using Equation 4.6. The calculations for the attributes are

$$P(Y = 1|x^1 = 1,\ x^5 = 250) = P(x^1 = 1|Y = 1) \times P(x^5 = 250|Y = 1) \times P(Y = 1)$$

$$= \frac{1}{3} \times (8.04 \times 10^{-189}) \times \frac{3}{7} = 1.419 \times 10^{-189}$$

$$P(Y = 3|x^1 = 1, x^5 = 250) = P(x^1 = 1|Y = 3) \times P(x^5 = 250|Y = 3) \times P(Y = 3)$$

$$= \frac{1}{3} \times 0.086 \times \frac{2}{7} = 8.1905 \times 10^{-4}$$

Since $P(Y = 3|x^1 = 1, x^5 = 250) > P(Y = 1|x^1 = 1,\ x^5 = 250)$, the NBC will label this data point as Take Medicine ($Y = 3$).

The NBC provides a probabilistic approach to classification. NBC is a popular approach for two primary reasons – simplicity and the mechanism of calculating explicit hypothesis probabilities. However, one practical difficulty with NBC is the need for the initial knowledge of the different probabilities. When the probabilities are not available in advance, domain knowledge, previously available data or assumptions on the form of the underlying distributions have to be used to determine the

approximations. NBC is often observed to achieve good performance even when the independence assumption is violated for many real world applications.

4.3 GAUSSIAN MIXTURE MODEL

A GMM represents a direct extension of the NBC for continuous attributes that are modeled as Gaussian distributions. Assuming that a continuous-valued attribute follows a Gaussian distribution offers important analytical properties. This assumption also facilitates a simple method for estimating the parameters of the distribution. However, this assumption is not always sufficient for modeling activity datasets. For example, consider the Eating activity that happens on average three times during the day. If one of the attributes is activity start time, then one expects to see a plot similar to the one illustrated in Figure 4.1.

As can be seen from the plot, a single Gaussian is insufficient to model this attribute as the figure clearly indicates the presence of three means. However, a linear superposition of three Gaussians, shown with the dashed-line curve in Figure 4.1, gives a better parameterization of the attribute's distribution. Linear combinations of basic distributions such as Gaussians can themselves be formulated as probability distributions and are referred to as mixture models. Gaussian mixtures are examples of mixture models where the underlying basic distribution is Gaussian. It has been shown that any continuous density can be approximated using a sufficient number

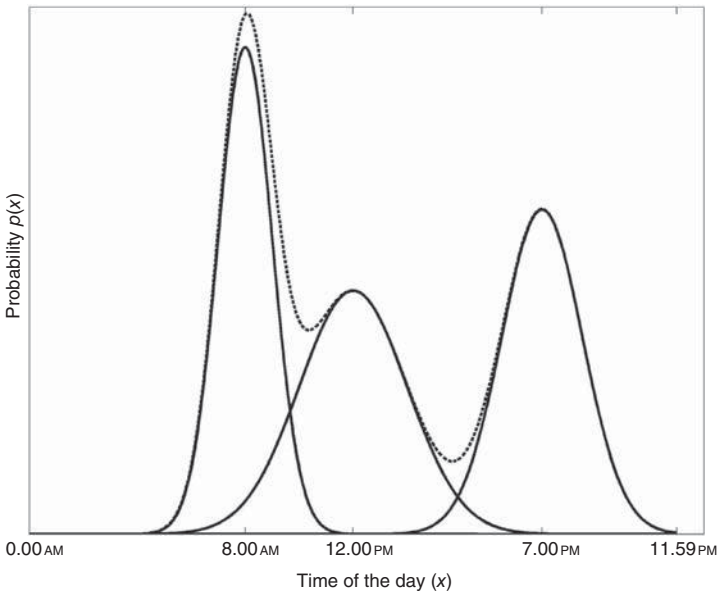


FIGURE 4.1 *Plot illustrating the distribution of the start time of the Eating activity from a synthetic dataset. The three solid curves represent the distribution for breakfast, lunch, and dinner. The dashed curve is the sum of these distributions.*

of Gaussian distributions (defined by appropriate means and variances) and weight factors for the linear combination.

Formally, for a continuous attribute X^i the distribution $P(X^i|Y = k)$ is modeled as a linear combination of M Gaussian densities as shown in Equation 4.14.

$$P(X^i|Y = k) = \sum_{m=1}^{M} \pi_m N(\mu_{km}^i, \sigma_{km}^{2^i}) \tag{4.14}$$

The Gaussian density $N(\mu_{km}^i, \sigma_{km}^{2^i})$ corresponds to the m^{th} mixture component and is parameterized by its corresponding mean μ_{km}^i and variance $\sigma_{km}^{2^i}$. The parameters π_m are called mixing coefficients and follow the constraints

$$\pi_m \geq 0 \text{ and } \sum_{m=1}^{M} \pi_m = 1 \tag{4.15}$$

This approach can be extended to cover all the attributes characterizing the data by considering M multivariate Gaussian distributions with mean μ_{km} and variance σ_{km}^2, $m = 1, 2, \ldots, M$, together with mixing coefficients. Note that both μ_{km} and σ_{km}^2 are D-dimensional vectors. These parameters and the mixing coefficients are estimated using an expectation-maximization procedure.

Expectation maximization (EM) is an iterative algorithm that starts with initial estimates for the mixture components and mixing coefficients. The EM algorithm is sensitive to initialization and so a bad start can lead to a poor result. A simple approach to generate the initial estimates is to randomly assign a data point from the training set as the mean and specify equal variances for all the components. Another approach is to cluster training data using a technique such as k-means clustering. The means and variance can then be estimated directly from the clusters. An alternative approach to initialization is hierarchical clustering. The initialization algorithm updates the estimates using clusters at the top level of the hierarchy and iteratively updates the estimates using clusters at the next lower level in the cluster hierarchy until convergence is detected. Each EM iteration then consists of an Expectation (E) step where each data point is assigned to the nearest mixture component and a Maximization (M) step where the parameters of the mixture components and the mixing coefficients are updated using the modified data assignments. This GMM_EM algorithm is summarized in Figure 4.2.

Most often the number of mixture components is provided by an expert. The number of mixture components is typically determined heuristically based on domain knowledge. GMM can be thought of as the probabilistic counterpart for k-means clustering. In the context of activity recognition, each activity is first modeled as a GMM. Then the NBC classifier as described by Equation 4.6 is then applied to classify a test data point. In the case of the GMM approach, however, the likelihood term found in Equation 4.6 is replaced by the probability derived through the mixture models. Alternatives to EM for estimating mixture model parameters such as Markov chain Monte Carlo, moment matching, and spectral methods can also be considered.

Algorithm **GMM_EM**(x)

// $x = x_1, x_2, .., x_{Nk}$ is the subset L_k of the training data that is marked with activity k
// M is the number of Gaussian mixture components required to describe L_k

for m = 1 .. M

Initialize the multivariate Gaussian mean μ_{km} and variance σ^2_{km}

done

repeat

for i = 1 .. N_k // Expectation step

$p_{im} = N(x_i \mid \mu_{km}, \sigma^2_{km})$ // p_{im} represents probability that data point x_i has been
// sampled from the m[th] mixture component $N(\mu_{km}, \sigma^2_{km})$

$z_i = \underset{m}{\operatorname{argmax}}\, p_{im}$ // z_i is a latent variable representing the mixture
// component to which data point x_i is assigned

done

for m = 1 .. M // Maximization step

A_m = number of data points assigned to m[th] component

$\mu_{km} = \frac{1}{A_m} \sum \{x_i : z_i = m\}$ // Update the mean

$\sigma^2_{km} = \frac{1}{A_m} \sum \{(x_i - \mu_{km})^2 : z_i = m\}$ // Update the variance

$\pi_{km} = \frac{A_m}{N_k}$ // Update the mixing coefficient

done

until convergence

FIGURE 4.2 *The GMM_EM algorithm.*

4.4 HIDDEN MARKOV MODEL

The HMM is a popular tool for modeling data that can be characterized by an underlying process generating a sequence of observations, such as sensor events. HMMs are generative probabilistic models consisting of a hidden variable and an observable variable at each time step, as shown in Figure 4.3. In the context of modeling activities, the hidden variable is the activity and the observable variables are the sensor events or the features extracted from the sensor events. An example of an HMM for an activity sequence is presented in Figure 4.4. Given a sequence of observed sensor events, we cannot directly perceive the state (activity) sequence that produced some or all of these observations. This is because multiple activities can generate similar sensor events. For example, in Figure 4.4 we note that sensor I06 can generate an event while a resident is performing either of the Prepare Meal, Fill Medicine Dispenser, or Watch DVD activities. Thus the activity label cannot be inferred by noticing the I06 event in isolation. Taking into account the sensor events in the sequence leading to the I06 event provides a better context for recognizing the activity. This is where

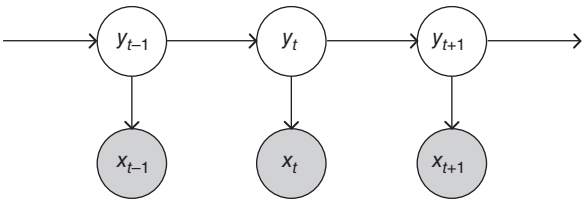


FIGURE 4.3 *Graphical representation of an HMM. The shaded nodes represent the observed variables — the sensor events or the feature vectors from the sensor stream. The white nodes represent the hidden variables, which correspond to the underlying activity labels. Note the direction of the arrow indicating that the underlying hidden states (activities) generate the observations (sensor events).*

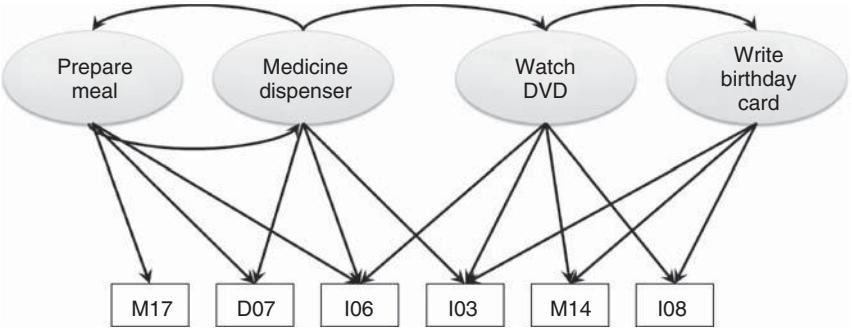


FIGURE 4.4 *A HMM used to recognize four distinct activities.*

the HMM comes in handy. The HMM facilitates computation of the probability that a particular activity model produced the entire sensor sequence, as well as the state (activity) sequence that was most likely to have produced the observations (sensor events).

The HMM is characterized by two dependency assumptions. These are illustrated in Figure 4.3 using directional arrows. First, the Markov assumption dictates that the hidden variable (activity) at time t, namely y_t, depends only on the previous hidden variable, y_{t-1}. In other words, the current activity depends only on the immediately preceding activity. Second, the HMM assumes that the observable variable (sensor event) at time t, namely x_t, depends only on the hidden variable y_t at that particular point in time. With these assumptions, we can specify an HMM using three probability distributions:

- The distribution over initial states, $P(y_1)$, representing the probability with which a activity y occurs at the beginning of the state sequence.
- The transition distribution, $P(y_t|y_{t-1})$, representing the probability of switching from one state at time $t-1$ to another state at the next time step, t. This represents the probability of transitioning between two activities.

- The observation distribution, $P(x_t|y_t)$, indicating the probability that state y_t would generate observation x_t. This represents the probability of a particular activity generating a specific associated sensor event.

Learning the parameters of these distributions corresponds to maximizing the joint probability, $P(X, Y)$, of the paired observation and label sequences in the training data. We can factorize the joint distribution in terms of the three distributions described earlier as shown in Equation 4.16.

$$P(X, Y) = \prod_{t=1}^{T} P(y_t|y_{t-1})P(x_t|y_t) \tag{4.16}$$

In this equation, we denote the distribution over initial states $P(y_1)$ as $P(y_1|y_0)$ in order to simplify notation. The parameters that maximize this joint probability are found by counting the frequencies of occurrence in sample data when dealing with discrete data. Specifically, the initial state probabilities, or probability distribution over activities when the first sensor event is observed, can be computed as the fraction of total data instances that are associated with a particular activity label. The transition probability from activity i to j is determined by the fraction of data instances where activity i is immediately followed by j. This captures the temporal relationship between activities. Lastly, the emission probabilities are calculated by finding the frequency of sensor events as they are observed for each activity.

Example 4.2 Figure 4.4 shows an example HMM that is constructed from sample sensor data and can be used to recognize the activities such as Prepare meal, Fill medicine dispenser, Watch a DVD, or Write a birthday card. In this figure, the circles represent the hidden states (in this case, activities) and the rectangles represent the observables (in this case, sensor events). Edges between hidden states represent the transition between the activities and edges connecting the activities to the sensor IDs represent the emission probabilities of the corresponding observed sensor state.

Inferring which sequence of labels best explains a new sequence of observations can be performed efficiently using the Viterbi algorithm. This dynamic programming technique is commonly used for HMM computations. The labeling problem can be formalized as finding the state sequence $\{y_1, y_2, \ldots, y_T\}$ that is most likely to have generated the given observation sequence $\{x_1, x_2, \ldots, x_T\}$. Let λ denote the HMM that is learned from the training data using the previously-described process. Then λ is essentially a triplet consisting of $\lambda = (\pi, A, B)$, where π represents the vector of initial probabilities, A represents the emission/observation probability where $a_j(x) = P(x|Y = i)$, and B represents the transition probability matrix where $b_{jk} = P(y_t = k|y_{t-1} = j)$.

Furthermore, we define the quantity $\delta_t(j)$ as the highest probability along a single activity sequence path at time t. Equation 4.17 shows how this value is computed by taking into account the first t sensor events and ending in activity j.

$$\delta_t(j) = \max_{y_1, \ldots, y_{t-1}} P(y_1, \ldots, y_t = j, x_1, \ldots, x_t | \lambda) \tag{4.17}$$

By induction Equation 4.18 can be seen to follow.

$$\delta_{t+1}(k) = [\max_j \delta_t(j) b_{jk}] \times a_k(x_{t+1}) \tag{4.18}$$

As we are primarily interested in identifying the activity sequence, we need to keep track of the argument j (activity) that maximized the aforementioned term for each time step t and for each value of k. This is accomplished by introducing another variable, $\psi_t(k)$, that represents this argument. The resulting algorithm to identify the most likely activity sequence is summarized in Figure 4.5.

The above-described approach builds a single HMM where the states correspond to each individual activity. Another approach would be to model each activity in itself as an HMM. This is a popular technique when the activity consists of well-defined substructures. These substructures refer to the hidden states of the HMM. In this case, a variation of the expectation-maximization framework known as the Baum–Welch algorithm is used to learn the parameters of the HMM from training data. This approach results in models $\{\text{HMM}_1, \text{HMM}_2, \ldots, \text{HMM}_K\}$. Given a test sequence, the likelihood of each of these models generating the sequence is computed. The sequence is then assigned the activity label that yields the maximum likelihood.

Algorithm **Viterbi**()

// Initialization
for i = 1 to K do
$\delta_1(i) = \pi_i a_i(x_1)$
$\psi_1(i) = 0$
done

// Recursion
for t = 2 to T do
for j = 1 to K do
$\delta_t(j) = [\max_i \delta_{t-1}(i) b_{ij}] a_j(x_t)$
$\psi_t(j) = \arg\max_{1 \le i \le K} \delta_{t-1}(i) b_{ij}$
done
done

// Termination
$P^* = \max_{1 \le i \le K} \delta_T(k)$
$y_T^* = \arg\max_{1 \le i \le K} \delta_T(k)$

// State (activity) sequence
for t = T-1 down to 1 do
$y_t^* = \psi_{t+1}(y_{t+1}^*)$
done
return

FIGURE 4.5 *The Viterbi algorithm to generate the most likely sequence of activity labels from a sequence of sensor observations.*

4.5 DECISION TREE

Decision tree learning is a popular supervised learning technique for inductive inference. It is used to approximate discrete valued target functions, where the function is represented as a tree. A decision tree classifies instances by traversing down the learned tree from the root node to a leaf node. The algorithm returns the class label associated with the corresponding leaf node. Each internal node of the tree processes the value of one instance attribute and each child branch corresponds to one of the possible values of the attribute.

Example 4.3 An example that illustrates the use of decision trees for activity learning is presented in Figure 4.6. This decision tree classifies a given activity instance into any one of the four activities (Bed Toilet Transition, Personal Hygiene, Take Medicine, Other). For example, the synthetic activity instance (*Bathroom sensor count* = 20, *Time of the day* = 11:00pm, *Medicine cabinet sensor count* = 1) would be processed by traversing the leftmost path of the tree and would therefore be classified as Bed Toilet Transition.

The goal of a decision tree learning algorithm is to infer a decision tree from information found in a set of training examples. The ID3 algorithm is one of the most common algorithms used for this purpose. This algorithm constructs the decision tree in a top-down manner. ID3 evaluates each attribute using a test to determine its ability to classify the training examples. The algorithm selects the best attribute as the root node of the tree. Next, it creates a child node for each possible value of this attribute and the training examples are sorted to the appropriate descendant node. ID3 repeats this process for each node and the associated subset of training examples. Finally when the training examples associated with a descendant node all belong to a single class, the algorithm designates the node as a leaf and marks the node with

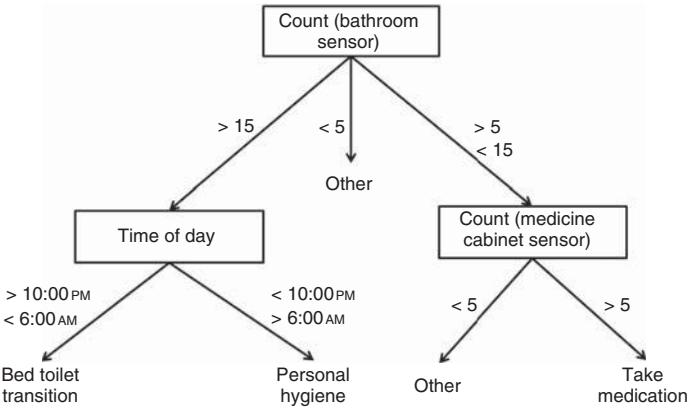


FIGURE 4.6 *A sample decision tree for classifying a data point as Bed Toilet Transition, Personal Hygiene, Take Medicine, or Other.*

```
Algorithm ID3(L, ActivityLabels, Attributes)
  // L is the set of training data points
  // ActivityLabels{1, 2, ..., K} are the class values that will be predicted by the decision tree
  // Attributes{X^1, X^2, .., X^D} is the set of attributes that may be tested by the decision tree

  create Root node for the tree

  if all data points in L belong to one activity a
    return the Root node with activity label = a
  end if

  if Attributes is empty
    a = most commonly-occurring label among training data associated with node
    return Root node tree with activity label = a
  end if

  X^i= attribute with highest information gain
  DecisionAttribute(Root) = X^i
  for each possible value v ∈ X^i
    add branch corresponding to test X^i = v with parent node = Root
    add child node of branch containing L_v, the set of training data L_v ⊆ S with X^i = v

    if L_v = Ø
      a = most commonly-occurring label among training data associated with node
      create Leaf node as child of branch
      assign Label(Leaf) = a
    else
      create Subtree as child of branch
      Subtree = ID3(L_v, ActivityLabels, Attributes - X^i)
    enf if
  done
return Root
```

FIGURE 4.7 *ID3 decision tree algorithm.*

the class label of the associated training examples. The algorithm is summarized in Figure 4.7.

In ID3, the selection of the attribute at every node is performed using a statistical quantitative measure. This measure, *information gain*, quantifies how well a given attribute separates the training example according to the target classification labels (activity labels). Information gain relies on another information theory measure, known as *entropy*. Entropy characterizes the amount of uncertainty in a random variable. Another characterization of entropy is the impurity of an arbitrary collection of examples. If a training set contains examples drawn from K different classes, the entropy of the set is defined as shown in Equation 4.19.

$$\text{Entropy} = -\sum_{k=1}^{K} p_k \log_2 p_k \tag{4.19}$$

In this equation, p_k corresponds to the proportion of data examples in the training set labeled as class k. As an example, consider the training set described in Table 4.1

that contains three examples of Personal Hygiene, one example of Bed Toilet Transition, two examples of Take Medicine and one example of an Other activity. The entropy of this set is

$$E = -\left(\frac{3}{7}\log_2\frac{3}{7}\right) - \left(\frac{1}{7}\log_2\frac{1}{7}\right) - \left(\frac{2}{7}\log_2\frac{2}{7}\right) - \left(\frac{1}{7}\log_2\frac{1}{7}\right) = 1.8424$$

The logarithm is calculated base 2 because entropy is a measure of expected number of bits required for encoding. Information gain measures the expected reduction in entropy caused by partitioning the set of examples using a specific attribute. Let L represent the initial labeled training set and X^i represent the attribute that is used to partition L. Then the information gain $\text{Gain}(L, X^i)$ is defined as shown in Equation 4.20.

$$\text{Gain}(L, X^i) = \text{Entropy}(L) - \sum_{v \in \text{values}(X^i)} \frac{|L_v|}{|L|}\text{Entropy}(L_v) \tag{4.20}$$

In this equation, $\text{values}(X^i)$ represents the set of all possible values for the attribute X^i and L_v represents the subset of L whose elements have a value of v for the attribute X^i. The second term in Equation 4.20 is the expected entropy of the subset of training instances that results from partitioning the set L based on the value of attribute X^i. Information gain refers to the number of bits that is saved when encoding the class label of an arbitrary example of L as a result of knowing the value of the attribute X^i. The ID3 decision tree learning algorithm is summarized in Figure 4.7.

4.6 SUPPORT VECTOR MACHINE

SVM is a popular supervised learning technique that attempts to maximize the predictive accuracy while not overfitting the data. The SVM algorithm models the region in the feature space that separates the data points belonging to two classes. In particular, it searches for a linear function that maximizes the distance to data points in both classes. This technique is grounded in statistical learning theory that provides a framework for learning a predictive function based on data. Specifically, the problem of supervised learning is formulated as a problem of learning a function f from training data that minimizes the expected error on new data.

We describe SVM in the context of binary classification where there are only two classes (positive and negative) to be learned. The positive and negative class examples are marked with the labels 1 and -1, respectively. If we are attempting to distinguish instances of a Hand Washing activity from a Sweeping activity, we can arbitrarily treat Hand Washing as the positive class and Sweeping as the negative class.

Given a set of training examples $\{x_1, x_2, \ldots, x_N\}$ and the corresponding labels $\{y_1, y_2, \ldots, y_N\}$ where $y_i \in \{1, -1\}$, the SVM learns a function $f(x) = w^T x + b$ (where T is the transpose operator) such that if $f(x) > 0$ then x is labeled as 1 (the positive class), otherwise it is labeled as -1 (the negative class). Let us further

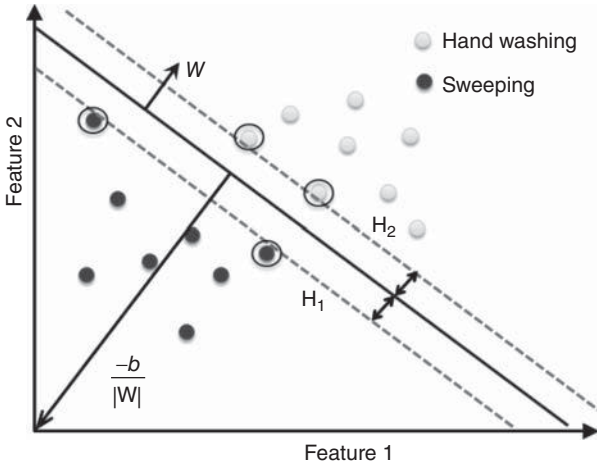


FIGURE 4.8 *Illustration of a SVM for a linearly separable classification problem.*

assume that the two classes are linearly separable. This means that there exists a function f that can classify all the examples correctly.

Figure 4.8 illustrates an artificial scenario involving data points from two activities, Hand Washing and Sweeping. The data points are characterized by two attributes, denoted as feature 1 and feature 2. We use two-dimensional attributes here to illustrate the concept of a SVM. The dashed lines H_1 and H_2 (hyperplanes in higher dimensions) represent the equations $f(x) = 1$ and $f(x) = -1$, respectively. The data points that lie on these lines are referred to as *support vectors*. Figure 4.6 illustrates the points that are support vectors using a concentric black outer circle. The goal of support vector machines is to orientate the separating line in such a way as to be as far as possible from the closest members of both the classes. This separating line is defined by the equation $f(x) = 0$ and is illustrated by the solid line in the figure. This separating line is equidistant from H_1 and H_2. This distance is typically referred to as the *margin*. In order to orientate the separating line to be as far as from the support vectors as possible, we need to maximize the margin. Hence SVM is also referred to as a *maximum margin classifier*.

The learning process involves estimating the values for the variables w and b such that they satisfy conditions on the training data that are summarized in Equation 4.21 and 4.22.

$$w^T x_i + b \geq +1 \text{ for } y_i = +1 \tag{4.21}$$

$$w^T x_i + b \leq -1 \text{ for } y_i = -1 \tag{4.22}$$

These equations can be combined into Equation 4.23.

$$y_i(w^T x_i + b) \geq 0 \quad \forall i \tag{4.23}$$

Simple geometry shows that the margin is $\frac{1}{||w||}$ (where $\|.\|$ refers to the l_2 norm of the vector) and maximizing it subject to the aforementioned constraint is equivalent to the optimization problem expressed in Equation 4.24.

$$\min||w|| \text{ such that } y_i(w^T x_i + b) \geq 0 \quad \forall i \tag{4.24}$$

Minimizing $\|w\|$ is equivalent to minimizing $\frac{1}{2}\|w\|^2$, which results in the quadratic optimization problem shown in Equation 4.25.

$$\min \frac{1}{2}||w||^2 \text{ such that } y_i(w^T x_i + b) \geq 0 \quad \forall i \tag{4.25}$$

Using Lagrange multipliers $\alpha_i \geq 0 \quad \forall i$, this problem can be reformulated as a min–max optimization problem shown in Equation 4.26.

$$L_P : \max_{\alpha_i} \min_{w,b} \frac{1}{2}||w||^2 - \sum_{i=1}^{N} \alpha_i y_i (w^T x_i + b) + \sum_{i=1}^{N} \alpha_i \tag{4.26}$$

This is also referred to as the *primal formulation*. We want to find w and b such that L_P is minimized while finding the appropriate α_i that maximizes L_P. The derivates of L_P with respect to w and b are equated to 0 and the resulting estimates for w are then substituted in L_P to obtain the dual formulation shown in Equation 4.27.

$$L_D : \max_{\alpha_i} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \text{ such that } \alpha_i \geq 0 \forall i, \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{4.27}$$

This is a convex quadratic optimization problem and a QP solver can be used to obtain the optimal values of α_i. The parameters w and b can then be estimated from α_i using Equations 4.28 and 4.29.

$$w = \sum_{i=1}^{N_{SV}} \alpha_i y_i x_i \tag{4.28}$$

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} \left(y_i - \sum_{j=1}^{N_{SV}} \alpha_j y_j x_j \cdot x_i \right) \tag{4.29}$$

In these equations, N_{SV} represents the number of support vectors. The estimate for b is calculated by averaging its value over all of the support vectors. The α_i's for nonsupport vectors are evaluated to zeros.

Nonseparable Case The previous formulation made the assumption that the two classes are linearly separable. This means that there exists a linear function f that can correctly classify the data points belonging to the two classes. However, in most real-world scenarios, this assumption is not valid. Due to either lack of sufficient features to characterize the classes or the inherent similarity between the classes, data points from the two classes may overlap. To handle these situations, SVM slightly

relaxes constraints defined in Equations 4.21 and 4.22 to allow for misclassification of points using a *soft margin*. This is carried out by introducing positive slack variables ζ_i such that the constraints specified in Equations 4.30 through 4.32 are met.

$$w^T x_i + b \geq +1 - \zeta_i \text{ for } y_i = +1 \tag{4.30}$$

$$w^T x_i + b \leq -1 + \zeta_i \text{ for } \quad y_i = -1 \tag{4.31}$$

$$\zeta_i \geq 0 \quad \forall i \tag{4.32}$$

In this soft margin SVM, data points on the wrong side of the margin boundary have a penalty that increases with the distance from it. Figure 4.9 illustrates this scenario. Notice that an instance of the Hand Washing activity is located on the wrong side of the boundary. The soft margin SVM penalizes these points while learning the separating boundary. This penalty is denoted by the slack variable, ζ_i. Note that $\zeta_i = 0$ for data points that are on the correct side of the separating hyperplane. Since the overall goal for SVM is to minimize the number of misclassifications, a reasonable objective function that accounts for the penalty should be generated as shown in Equation 4.33.

$$\min \frac{1}{2}||w||^2 + C\sum_{i=1}^{N} \zeta_i \text{ such that } y_i(w^T x_i + b) - 1 + \zeta_i \geq 0 \text{ and } \zeta_i \geq 0 \quad \forall i \tag{4.33}$$

In this equation, the parameter C controls the tradeoff between the slack variable penalty and the size of the margin. Using Lagrangian multipliers α_i and μ_i and applying the sets of constraints, the primal optimization problem is formulated as shown in Equation 4.34.

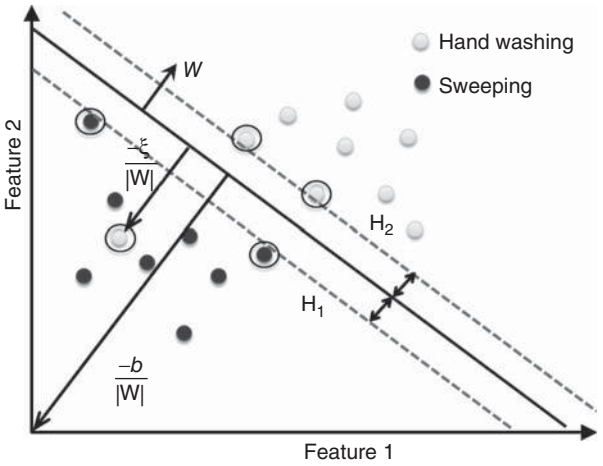


FIGURE 4.9 *Illustration of the soft-margin concept for SVM classification in the nonseparable scenario.*

$$L_P \ : \ \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\zeta_i - \sum_{i=1}^{N}\alpha_i[y_i(w^T x_i + b) - 1 + \zeta_i] - \sum_{i=1}^{N}\mu_i\zeta_i \tag{4.34}$$

Using the same process that was employed for the linearly separable case, the dual formulation of the problem is shown in Equation 4.35.

$$L_D \ : \ \max_{\alpha_i}\left[\sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j x_i \cdot x_j\right] \tag{4.35}$$

$$\text{such that } 0 \leq \alpha_i \leq C \ \forall i \quad \text{and} \quad \sum_{\mathrm{i}=1}^{\mathrm{N}}\alpha_{\mathrm{i}} y_i = 0$$

This is again a quadratic optimization problem whose solution can be obtained using a QP solver. The estimates for w remain the same as in the linear separable case:

$$w = \sum_{i=1}^{N}\alpha_i y_i x_i \tag{4.36}$$

The value of b is determined using Equation 4.29, where the value α_i for the support vectors satisfies the condition $0 \leq \alpha_i \leq C$.

Nonlinear Support Vector Machines Many real world classification problems contain data that is not linearly separable in the original feature space, X. However, mapping the data into a higher dimensional space $\varphi(X)$ might make it linearly separable. In other words, there exists a linear function $f(x) = w^T\varphi(x) + b$ that can classify the data points $\varphi(X)$ correctly in the higher dimensional space. If we refer Equations 4.28 and 4.29 which derive w and b for the linearly separable case, the linear function in the transformed space can be expressed as shown in Equation 4.37.

$$f(x) = \sum_{i=1}^{N_{SV}}\alpha_i y_i \varphi(x_i)\cdot\varphi(x) + \frac{1}{N_{SV}}\sum_{i=1}^{N_{SV}}\left(y_i - \sum_{j=1}^{N_{SV}}\alpha_j y_j \varphi\left(x_j\right)\cdot\varphi(x_i)\right) \tag{4.37}$$

Interestingly, note that the expression only requires the calculation of an inner product between the transformed high dimensional data points and not the actual high dimensional data points themselves. Thus a *kernel* function, k, which can compute the inner product in the high-dimensional space can be used instead, as shown in Equation 4.38.

$$k(x_i, x_j) = \varphi(x_i)\cdot\varphi(x_j) \tag{4.38}$$

Note that the kernel function k is a symmetric function of its arguments, therefore $k(x_i, x_j) = k(x_j, x_i)$. This general principle of replacing the occurrence of the inner products in the higher-dimensional space with a function that estimates the same term in the original space is referred to as the *kernel trick*. The simplest example of a kernel

TABLE 4.2 Sample Kernel Functions

Linear kernel	$x_i^T x_j$
Polynomial kernel	$(x_i^T x_j + c)^d$
Radial basis function kernel	$\exp\left(-\frac{\|\|x_i - x_j\|\|_2^2}{2\sigma^2}\right)$
Hyperbolic tangent kernel	$\tan h(\kappa x_i^T x_j + c)$ for $\kappa > 0$ and $c < 0$

Algorithm **SVM**(S, k, QPSolver)

// S is the set of training data consisting of data points from X and associated labels from Y
// k is the kernal function
// QPSolver solves the quadratic optimization problem

$H_{ij} = y_i y_j k(x_i, x_j)$

// C determines how significantly misclassifications are treated
choose value for penalty parameter C

// Find values for α such that $\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \alpha^T H \alpha$ is maximized

// subject to constraints $\forall i, 0 \le \alpha_i \le C$ and $\sum_{i=1}^{N} \alpha_i y_i = 0$

α = QPSolver(H, C)

determine support vectors by finding indices such that $0 \le \alpha_i \le C$

N_{sv} = number of support vectors

// Calculate the offset parameter

$b = \frac{1}{N_{sv}} \sum_{i=1}^{N_{sv}} (y_i - \sum_{j=1}^{N_{sv}} y_i \alpha_j k(x_i, x_j)$

return

FIGURE 4.10 *The two-class SVM algorithm.*

function is obtained by considering the identity mapping for φ, specifically $\varphi(x) = x$. In this case, the kernel function is $k(x_i, x_j) = x_i^T x_j$, which is typically referred to as the *linear kernel*. There are many forms of kernel functions in common use. We list a few of them in Table 4.2.

The generic two-class SVM algorithm is summarized in Figure 4.10. Using the learned support vectors and offset parameter, the class value of a data point x_i can be determined. To do this, we calculate $\text{sign}(\sum_{i=1}^{N} \alpha_i y_i k(x_i, x_j) + b)$.

Multiclass SVM The multiclass SVM is an extension of the two-class SVM that is capable of assigning labels to data where the number of labels (classes, or in our case activities) is greater than two. The most common method for handling this situation consists of decomposing the multiclass problem into multiple binary class problems.

In the *one versus all* approach, the SVMs are trained by considering data from one of the classes as positive examples, while considering the data from all other classes as negative examples. If there are K classes, this approach results in K SVM models. During testing, the SVM model with the highest margin (or probability) of classification is used to label the test data point. This represents a type of winner-takes-all strategy. Another approach is the *one versus one* method where a separate SVM model is trained to distinguish every pair of classes, resulting in $K(K-1)/2$ SVM models. During testing, every classifier assigns the test instance to one of the two classes and the cumulative votes for each class are maintained. The class with the maximum number of votes is considered to be the winner. This represents a maximum-wins voting strategy. Other methods to handle multiclass SVMs can also be considered, including the directed acyclic graph model and the error-correcting output codes model.

4.7 CONDITIONAL RANDOM FIELD

A CRF is a discriminative probabilistic graphical model that is used for segmenting and labeling sequence data. While it comes in different forms, the form that most closely resembles its generative counterpart, the HMM, and that is commonly used for activity recognition, is known as a *linear chain CRF*. In contrast to classifiers such as naïve Bayes or SVM, a CRF can consider the "context" of a sample (neighboring samples) while classifying a sample. To understand the relevance of CRF for activity recognition, consider the problem where a sequence of (hidden) activities generates a sequence of sensor events that can be observed. As an activity can generate multiple sensor events, it is worthwhile to consider the preceding and succeeding sensor events as the context for an event that has to be classified. These relationships can be modeled using the graphical structure of a CRF.

The linear chain CRF, illustrated in Figure 4.11, still consists of a hidden variable (the activity label) and an observable variable (sensor event) at each time step. In contrast to the HMM model illustrated in Figure 4.3, the arrows on the edges have disappeared in the CRF, making this an undirected graphical model. This means that two connected nodes no longer represent a conditional distribution; instead we refer to the *potential* between two connected nodes. Unlike probability functions, potentials (also called feature functions) are not restricted to a value between 0 and 1. The

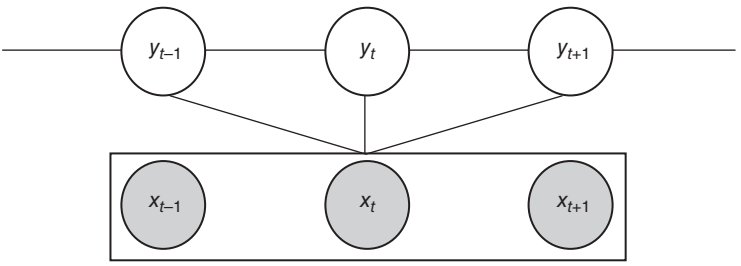


FIGURE 4.11 *Graphical representation of a linear chain CRF.*

potential functions that specify the linear chain CRF are $f_k(y_t, y_{t-1})$ and $g_k(y_t, x_t)$. The f_k function captures the relationship between the label at the current time step and the label at the preceding time step while the g_k function captures the relationship between the current time step label and the observed variables at the current time step. For the sake of simplicity, we use $f_k(y_t, y_{t-1}, x_t)$ to represent both $f_k(y_t, y_{t-1})$ and $g_k(y_t, x_t)$. If there are L such feature functions then a linear chain CRF is essentially the probability distribution $P(y|x)$ that takes the form shown in Equation 4.39.

$$P(y|x) = \frac{1}{Z_x} \prod_{t=1}^{T} \exp\left\{ \sum_{l=1}^{L} \theta_l f_l(y_t, y_{t-1}, x_t) \right\} \tag{4.39}$$

In this equation, Z_x is an instance-specific normalization function, which guarantees that the outcome is the probability shown in Equation 4.40.

$$Z_x = \sum_{y} \prod_{t=1}^{T} \exp\left\{ \sum_{l=1}^{L} \theta_l f_l(y_t, y_{t-1}, x_t) \right\} \tag{4.40}$$

Note that each feature function can depend on observations from any time step. While the argument to function f_k in Equations 4.39 and 4.40 is x_t, this should be interpreted as containing all the components of the global observation sequence x that are needed for computing the features at time t. An interesting observation here is that we do not model the distribution of the observations $P(x)$.

While there are modeling similarities between CRFs and HMMs, note that the HMM's transition probability $P(y_t|y_{t-1})$ and emission probability $P(x_t|y_t)$ have been replaced by the potentials f and g, respectively. The essential difference lies in the way we learn the model parameters. Given a sequence of observations x and corresponding labels y, the HMM learns the parameters by maximizing the joint probability distribution $P(x, y)$. By contrast, the CRF learns the parameters by maximizing the conditional probability distribution $P(y|x)$, which belongs to the family of exponential distributions. An iterative gradient algorithm can learn the model parameters, θ_l. Some particularly successful algorithms include quasi-Newton methods such as BFGS, because they take into account the curvature of the likelihood function. As described in Section 4.3, the Viterbi algorithm can generate activity labels that correspond to an input sequence of observed sensor events given a learned CRF model.

4.8 COMBINING CLASSIFIER MODELS

In the previous sections, we explored different generative and discriminative classification models for activity learning. Classification performance can often be improved by combining multiple models together, instead of using a single model. For example, we may train M different classification models. A combination of the models, such as a voting strategy or a simple average among the models, will decide the winning label for a particular data point. The meta-model that results from combining multiple models is sometimes referred to as a *committee machine*. We will discuss two approaches for combining multiple models in this chapter—Boosting and Bagging.

4.8.1 Boosting

Boosting is a powerful technique that combines multiple base (weak) classifiers to produce a committee (strong classifier) whose performance may be significantly greater than that of any of the base classifiers. Most boosting algorithms consist of learning weak classifiers with respect to a distribution in an iterative manner. The final strong classifier is built by combining the individual weak models in a weighted fashion, where the weights reflect each individual model's accuracy. The boosting algorithm recalculates the data distribution in each iteration, taking into account the model's output in the previous iteration and assigning higher weights to misclassified data points. The result is that in future iterations weak learners are forced to focus more on 'hard' examples, specifically the data points that were misclassified in the previous iterations. Training an activity classifier in this fashion can lead to significant improvements in performance of the final committee over the individual weak models.

Many boosting algorithms have been designed. In this chapter, we focus on a popular algorithm called Adaptive Boosting (AdaBoost). AdaBoost is adaptive in the sense that subsequent built classifiers are tweaked in favor of those instances that were misclassified by previous classifiers. The classifiers are allowed to have a performance that is slightly better than random (i.e., their error rate is smaller than 0.5 for binary classification). Classifiers with an error rate that is higher than what would be expected from a random classifier will be useful even if the performance is weak, since they will have negative coefficients in the final linear combination of classifiers and hence will behave like their inverses. Here we describe the AdaBoost algorithm for a two-class classification problem. However, the algorithm can be easily extended to multiclass problems as well.

Formally, we represent training examples as $\{x_1, x_2, \ldots, x_N\}$ and the corresponding labels as $\{y_1, y_2, \ldots, y_N\}$, where $y_i \in \{1, -1\}$. Each data point is also associated with a weighting parameter w_i^t for iteration t. The initial weighting parameter, w_i^1, is set to $1/N$ for all data points. Let us assume that we are provided with a base or weak classifier that uses the weighted data of iteration t to output a classification model, $h^t(x) \in \{-1, +1\}$. At iteration t, the algorithm learns a new classifier, h^t, using a dataset in which the weighting coefficients are adjusted according to the performance of the previously trained classifier, h^{t-1}. The weight adjustment specifically assigns greater weight to the previously misclassified data points. Finally, when the desired number of base classifiers have been trained, AdaBoost linear combines the weak base classifiers into a final strong classifier, $H(x)$, using coefficients that weight the base classifiers based on their performance.

The AdaBoost algorithm is summarized in Figure 4.12. Given a data point, x, and the set of classifiers $h^1, h^2, \ldots, h^M$ that are generated by the boosting algorithm, the class of x is calculated as shown in Equation 4.41.

$$H(X) = \text{sign}\left(\sum_{t=1}^{M} \alpha^t h^t(x)\right) \tag{4.41}$$

Algorithm **AdaBoost**(S, M, WeakLearner)

// S is the set of training data consisting of data points $\{x_1, x_2, .., x_N\}$ and the corresponding labels
// $\{y_1, y_2, .., y_N\}$, where each $y_i \in \{-1, +1\}$
// M is the maximum number of boosting iterations
// WeakLearner is the base classifier that outputs a classification model every iteration

// Initialize weights for the data

$w_i^1 = \frac{1}{N} \quad \forall i$

for t =1, .., M
  h^t = WeakLearner(S, w^t) // h^t minimizes the weighted error on the training data S
  // Calculate the weighted error of h^t on the training data
  $\varepsilon^T = \sum_{i=1}^{N} w_i^t I(y_i \neq h^t(x_i))$ // I is the indicator function
  if $|\varepsilon^t - 0.5| \leq \theta$ // θ is a predefined threshold
    stop the iterations
  end if
  choose $\alpha^t \in \Re$ // typically $\alpha^t = \frac{1}{2} \text{In}(\frac{1-\varepsilon^t}{\varepsilon^t})$
  for i = 1, .., N // Update the weights
    $w_i^{t+1} = w_i^t \exp(-\alpha^t y_i h^t(x_i))$
  done
  $w_i^{t+1} = \frac{w_i^{t+1}}{\sum_{i=1}^{N} w_i^{t+1}} \quad \forall i$ // Normalize the weights

return $h^1, h^2, .., h^M$

FIGURE 4.12 *The Boosting algorithm to combine multiple classifiers.*

4.8.2 Bagging

Bagging, also referred to as bootstrap aggregating, is another meta-algorithm that combines multiple classification models to improve the predictive accuracy over the individual models. Though it is popularly applied to decision tree methods, it can be coupled with any classifier model. Bagging reduces the variance of the overall model and is useful for avoiding overfitting.

The principle behind a bagging procedure is independent of the classification algorithm. Bagging produces replicates of the original training dataset by sampling with replacement from the dataset. This creates bootstrap samples of equal size to the original dataset. The classification algorithm is then used to train models using each instance of the sampled set. The final model is built by combining the outputs of the individual models. Formally, given the original training set, L with size N, bagging generates M new training sets L^m, each of size $N' \leq N$, by sampling from L uniformly with replacement. Sampling with replacement can result in duplicates of a training example in the new set L^m. When the size of the sampled set L^m is equal to the size of the original training set $N' = N$, on average close to 60% of L^m contains unique examples. Each of the M sets are then used to independently train M classification

```
Algorithm Bagging(S, M, N', h)
  // S is the set of training data
  // M is the number of bagging iterations
  // N' is the size of the sampled set
  // h is the base classifier

  for m = 1, 2, .., M
    sample from S uniformly with replacement to obtain S^m of size N'
    learn model h^m by training h on S^m
  done

return h^1, h^2, .., h^M
```

FIGURE 4.13 *The Bagging algorithm to combine classifiers.*

models, $\{h^1, h^2, \ldots, h^{M\cdot}\}$. Each test data point, x, is then classified by considering the output of the M classification models. Here we mention common approaches for combining the models.

- *Maximum Votes*. Return the activity label with the maximum number of votes accumulated from each of the M models.
- *Highest Probability*. Return the activity label with the highest classification probability across all of the M models.
- *Highest Mean Probability*. Return the activity label with the highest mean probability across all of the M models.

Figure 4.13 summarizes the bagging algorithm. Using this algorithm, the label for a data point x can be calculated by combining the values of the individual models $h^1(x), h^2(x), \ldots h^M(x)$.

4.9 DIMENSIONALITY REDUCTION

As we stated at the beginning of the chapter, the first step in learning an activity model is to carefully select features that exhibit key characteristics of the data. With sensor data the number of features can be quite large. A naive approach to activity learning is to pool all of these available features into one vector that is input to the classifier. The disadvantage here is that some features may be irrelevant or redundant, and will not provide new information that improves the classification accuracy. Some features might even confuse the classifier or cause it to overfit the training data rather than help discriminate various activities. What is worse, due to the "curse of dimensionality," the performance may degrade sharply when more features are added if there is not enough training data to reliably learn all the parameters of the activity models. Therefore, to achieve the best classification performance, the dimensionality of the feature vector should be made as small as possible without losing valuable discriminatory information. In addition, keeping the dimensionality small will reduce the computational cost of training a model. As a result, the activity learning algorithms can more feasibly be implemented and run on lightweight devices such as smart phones.

Following our discussion of supervised learning, dimensionality reduction can be viewed as the process of reducing the number of features under consideration for training a classifier. There are two general approaches for dimensionality reduction: feature selection and extraction. Feature selection approaches try to find a subset of the original features to reduce the dimensions. On the other hand feature extraction transforms data from a high-dimensional space to a space with fewer dimensions in a linear or nonlinear fashion. Note that the feature extraction methods that are presented here are different from the feature extraction technique discussed in Chapter 3 where the goal was to extract statistical properties of raw data. By contrast, feature extraction in this discussion involves defining features that represent the same statistical properties but using a smaller number of individual features. To avoid confusion, we use the term dimensionality reduction to refer to this category of techniques.

Feature Selection Feature selection is a popular approach in machine learning problems where the number of attributes that represent the data is very high (tens of thousands or more). Feature selection can be performed for three main reasons: (i) improving the predictive performance of a classifier, (ii) reducing the number of attributes to be determined, thereby saving computational time, and (iii) understanding the causal relationships between the attributes and predictor variables.

A common approach to feature selection is to rank the attributes using a metric that quantifies the ability of the feature to predict the target class. This metric is referred to as the scoring function and denoted as SF. Given the D attributes $X^1, X^2, \ldots, X^D$, $SF(d)$ represents the score for the d^{th} attribute. By convention, we assume that a high score is indicative of a valuable variable. The feature selection algorithm will sort variables in nonincreasing SF order and then build classifiers using attributes that are progressively of lesser relevance. This approach is categorized as a filter method, since it is a preprocessing step and is independent of the final classifier used to build the predictive model. The scoring function SF can be designed from different metrics. Here we list a few popular metrics that are used for this purpose.

Correlation Coefficient The correlation is computed between a data attribute and the target label. This can be computed as shown in Equation 4.42.

$$SF(X^i) = \frac{\text{cov}(X^i, Y)}{\sqrt{\text{var}(X^i)\text{var}(Y)}} \tag{4.42}$$

In this equation, *cov* represents the covariance function and *var* refers to variance. This metric determines the linear dependencies between a data attribute and the target label.

Predictive Power Another attribute ranking metric is the performance of classifiers that are trained using just one of the available attributes. The classification performance can be quantified in terms of the error rate of the classifier, true positive rate, f score or area under the ROC curve. Attributes that can distinguish the different

target labels have higher predictive power. Note that when a large number of variables result in optimal performance, this ranking metric cannot distinguish between top-ranking attributes. Other metrics such as correlation or cost of computing the attribute have to be employed in such scenarios.

Mutual Information Several feature selection approaches use mutual information, an information theoretic measure, for ranking the data attributes. The mutual information of an attribute measures how much the presence or absence of an attribute contributes to making correct classification decisions. It is defined by the formula found in Equation 4.43.

$$SF(X^i) = \sum_{v \in \text{values}(X^i)} \sum_{y \in \text{values}(Y)} P(X^i = v, Y = y) \frac{\log(P(X^i = v, Y = y))}{P(X^i = v)P(Y = y)} \tag{4.43}$$

The probabilities are estimated using frequency counts for discrete variables. In the case of continuous variables, one can consider discretizing the variables or approximating the densities using nonparametric methods such as Parzen windows. A related feature selection technique is the minimum redundancy maximum relevance (mRMR) approach. In mRMR, the relevance of an attribute is estimated as the mutual information between an attribute and the target label. Similarly, redundancy is estimated as the mutual information between two attributes. The attribute selection problem is then cast as a combinatorial optimization problem for identifying the attributes with maximum relevance and minimum redundancy. Figure 4.14 summarizes the general filter-based approach for feature selection.

Example 4.4 Consider the problem of selecting the most important features for the dataset presented in Table 4.1 when we want to classify the activities Bed Toilet Transition and Take Medicine. We will use the correlation coefficient as the scoring function. The scoring function values for the five features are calculated as 0.17, 0.90, 0.97, 0.95, and 0.97, respectively. If we were to consider only the top four features for

```
Algorithm FeatureSelection(S, D')
  // S is the training data points
  // D' is the reduced number of features
  define the scoring function SF
  for i = 1, 2, .., D
    scores(i) = SF(i)
  done
  sort scores in nonincreasing order
return top D' features
```

FIGURE 4.14 *The filter-based feature selection algorithm.*

learning the activity model, they would be *Time of the day*, *Bathroom sensor count*, *Medicine cabinet sensor count*, and *Energy in the hip accelerometer z axis*.

An alternative approach to feature selection is to score feature subsets instead of individual features. This is referred to as a *wrapper* approach. Wrappers search through the space of possible feature subsets and evaluate each subset by training a classifier on the subset and evaluating the resulting performance. The classifier that is used to evaluate candidate attributes in a wrapper approach is typically the same one that is used to train the final model, although this is not a necessary criterion. Each new attribute subset is used to train a classifier. The trained model is tested on a hold out/validation example set. The classification accuracy on the validation set indicates the score of the subset. Exhaustive search is generically impractical because the number of models is the power set of available features. Early search stopping criteria can be employed to terminate the search. Such criteria include placing a threshold on the classifier performance or imposing a limit on the search run time. Alternative search strategies can also be employed, such as simulated annealing, best first search, or genetic algorithms.

Dimensionality Reduction Dimensionality reduction, as the name suggests, transforms the data in high-dimensional space to a space with fewer dimensions. While there are many linear and nonlinear techniques for reducing the dimensionality of the dataset, we will discuss the main linear technique, principal component analysis. Principal Component Analysis (PCA) projects the data onto a low-dimensional space using an orthogonal transformation in such a way that variance of the projected data is maximized. PCA is a useful technique when we have a large number of attributes that might be correlated with one another. If correlations exist, the data can be represented using fewer attributes by removing these redundancies. PCA achieves this by determining the directions along which there is maximum variation in data. These directions are called *principal components*. The data is then projected onto these principal components, yielding a more succinct representation. The principal components are essentially a linear combination of optimally weighted original data attributes where the PCA algorithm determines the optimal weights for the attributes. We describe the PCA algorithm from the perspective of maximizing the variance in the projected space.

We start with D-dimensional data and want a compact summarization by projecting the data into a D'-dimensional subspace, where $D' << D$. The first principal component indicates a direction in the feature space along which the projections have maximum variance. The second principal component indicates the direction with maximum variance among all directions that are orthogonal to the first principal component. Thus the k^{th} component represents the variance-maximizing direction orthogonal to the previous $k-1$ components. We now describe the procedure to obtain the D' principal components.

The algorithm begins by creating a data matrix X, where each row of the matrix represents a data point and each column represents an attribute. The column mean of this matrix is set to zero by subtracting the mean of X from each row. We then calculate the covariance matrix of the data with zero mean, as shown in Equation 4.44. This is a D-dimensional square matrix, with element (i, j) representing the covariance between attributes X^i and X^j.

$$\text{ccv}(i,j) = \text{cov}(X^i, X^j) \tag{4.44}$$

In the next step, we calculate the Eigen vectors and values for the covariance matrix ccv. Let $\{v_1, v_2, \ldots, v_D\}$ and $\{\lambda_1, \lambda_2, \ldots, \lambda_D\}$ represent the D Eigen values and the vectors of the covariance matrix. Note that these eigenvectors are unit eigenvectors (their magnitude is 1) and are of D dimensions, representing the weights of the original D attributes. The eigenvectors represent the direction of the data along which there is certain variance whose magnitude is represented by the corresponding eigenvalues. By selecting the D' eigenvalues of maximum value, we are essentially selecting the D' directions that exhibit maximum variance. These D' eigenvectors are the principal components. While there is no deterministic approach to estimate the value of D', heuristics such as the setting D' to number of eigenvalues required to capture 90% of variance can be used. The algorithm is summarized in Figure 4.15.

Example 4.5 As an example, let's perform PCA on the data presented in Table 4.1. Note that PCA does not take into account the target label of the data points. The data matrix after removing the mean from every row is:

Algorithm **PCA**(X, D')
// X is the data matrix of size NxD, N=number of data points, D=dimension of feature space
// D' is the number of dimensions in the lower-dimensional space

for j = 1 .. D // De-mean the data
  for i = 1 .. N
    $X(i,j) = X(i,j) - \mu_j$ // μ_j is the mean of the j^{th} attribute
  done
done

$ccv\ (i,\ j) = \frac{1}{N} \sum_{k=1}^{N} X\ (k,i)\ X\ (k,j)$ // Compute the covariance matrix

compute the eigenvalues $\lambda_1, .., \lambda_D$ and sort in nonincreasing order

compute the eigvenvectors $v_1, .., v_D$ and sort in nonincreasing order

// Determine the transformation / projection matrix using the
// D' eigenvectors corresponding to the top D' eigenvalues
$T = [v_1, .., v_{D'}]$

$X' = XT$ // Project data into the lower dimensional space

return X'

FIGURE 4.15 *The PCA dimensionality reduction algorithm.*

0.5714	−9.4286	4.8571	−3.2857	71.8571
0.5714	−4.4286	1.8571	−1.2857	116.8571
0.5714	8.5714	−3.1429	2.7143	−183.1429
−0.4286	−5.4286	2.8571	−2.2857	122.8571
−0.4286	−1.4286	−5.1429	0.7143	−176.1429
−0.4286	2.5714	3.8571	−2.2857	137.8571
−0.4286	9.5714	−5.1429	5.7143	−90.1429

The covariance matrix for this set of data points is:

0.2857	−0.8810	0.5952	−0.3095	0.9286
−0.8810	51.9524	−21.4048	20.0238	−614.7381
0.5952	−21.4048	18.8095	−12.5476	565.6429
−0.3095	20.0238	−12.5476	10.5714	−353.3810
0.9286	−614.7381	565.6429	−353.3810	20,935.1429

Table 4.3 lists the nonincreasing order of eigenvalues and the corresponding eigenvectors obtained by the Eigen decomposition of the covariance matrix. Each row in this table presents the eigenvalue and the corresponding eigenvector. From the table, we can observe that the first eigenvalue is significantly higher than the other eigenvalues. In fact, it appears that close to 99% of the variance in the data is captured by the top three eigenvalues. Let us utilize this result and project the data into a three-dimensional space using the eigenvectors of the top three eigenvalues. This will transform the data with mean 0 to:

72.2535	−7.9709	1.2904
116.9494	−0.5444	−1.6282
−183.3534	2.5916	2.6353
123.017	−1.6865	−0.8406
−176.0863	−6.8192	−1.1266
137.7947	6.2548	1.9945
−90.5749	8.1747	−2.3248

An added advantage of reducing the dimension of the data is that we can more easily visualize the lower-dimensional dataset. We plot the resulting three-dimensional data for our activity learning example in Figure 4.16.

TABLE 4.3 Eigenvalues and Eigenvectors for Activity Learning Example

	EigVal	X^1	X^2	X^3	X^4	X^5
EigVec1	20,974.50	0	−0.0294	0.027	−0.0169	0.9991
EigVec2	37.64	−0.0262	0.9431	−0.1573	0.2892	0.0369
EigVec3	3.77	0.1016	0.2954	0.7959	−0.5182	−0.0216
EigVec4	0.73	0.3992	−0.139	0.4948	0.7592	−0.0047
EigVec5	0.13	0.9108	0.0552	−0.3102	−0.2667	0.0055

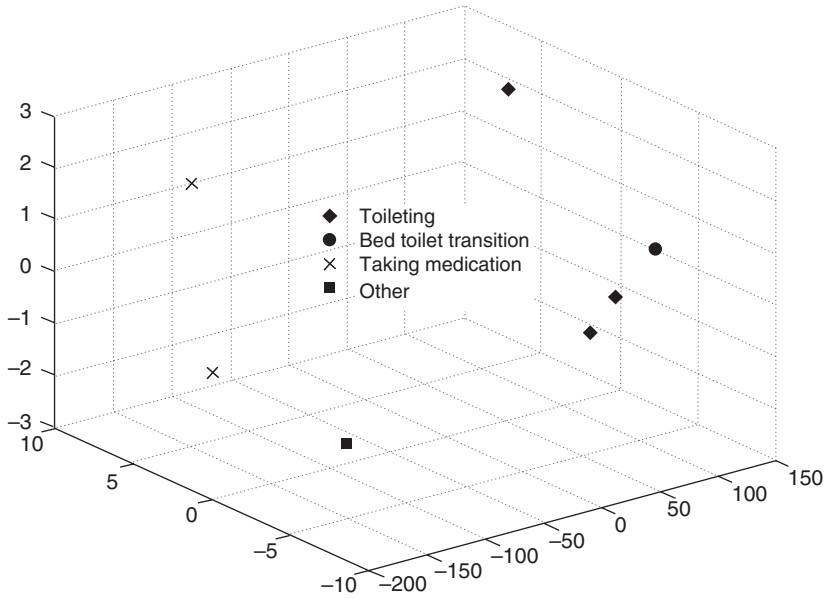


FIGURE 4.16 *Plot of the data points projected onto a three-dimensional subspace using PCA. The axes in the figure represent the three principal components.*

4.10 ADDITIONAL READING

There are many good sources that give a detailed description of the algorithms discussed in this chapter such as Duda et al.[50], Mitchell[51], and Bishop[52]. Duda et al.[50] provide an excellent reference to learn more about NBCs. In addition to providing a detailed description, Mitchell[51] illustrates NBCs in the context of document classification task. Duda et al.[50] and Bishop[52] provide a thorough treatment of mixture models and the EM algorithm. The EM algorithm was first proposed by Dempster et al.[53] as an alternative to maximum likelihood estimation. GMMs were first used by Reynolds et al.[54] for the task of speaker identification. Bilmes[55] provides a detailed tutorial of the EM algorithm and its application to parameter estimation for GMMs and HMMs. The three classic problems associated with hidden Markov models are discussed by Rabiner[56].

In addition to traditional supervised learning algorithms, human activities have also been represented as symbol sequences and modeled as context free grammars[57–63]. The grammar rules are typically supplied by an expert or extracted in an unsupervised fashion from a large amount of sample data. While the approach can be more complex than supervised concept learning, the grammars do offer a semantic richness and ability to model hierarchical activities, which may be difficult to capture with other representations and supervised learning algorithms.

Guyon et al.[64] provide an excellent introduction to the problem of feature selection with pointers to literature that describe different commonly used algorithms. A more thorough discussion on various feature selection techniques and appropriate methodologies for validation is present by Liu and Motoda[65]. Feature selection has been widely used for determining the appropriate set of features that should be extracted to effectively characterize wearable sensor data for activity recognition[66–68]. Lee et al.[69] discuss nonlinear dimensionality reduction techniques such as isomap, locally linear embedding and multidimensional scaling, along with variants of principal component analysis such as kernel PCA and probabilistic PCA at great length. Singular value decomposition is also used for reducing the representation space of a dataset and is compared with PCA by Wall et al.[70]

5

Activity Recognition

The field of activity recognition (AR) is concerned with the question of how to label activities from a sensor-based perception of the environment. The problem of AR is to map a sequence of sensor events, $x =< e_1\ e_2\ \ldots\ e_n >$, onto a value from a set of predefined activity labels, $a \in A$. AR can be viewed as a type of supervised machine learning problem. An AR algorithm learns a function that maps a feature vector, X, describing a particular sensor event sequence onto an activity label, $h : X \rightarrow A$. In this supervised machine learning problem, the classes are the activity labels and the sensor events are represented by features combined into the input vector X of D dimensions. AR can use the learned function to recognize, or label, occurrences of the learned activity.

AR faces challenges that make it unique among supervised machine learning problems. The sequential nature of the input data, the ambiguous partitioning of data into distinct data points, and the common overlapping of activity classes mean that additional data processing must be performed in order to accomplish the goal of recognizing activities. As Figure 5.1 shows, the steps involved in AR include collecting and preprocessing sensor data, then dividing it into subsequences that are converted to feature representations. The final feature vectors are either labeled by an oracle and provided as training data to learn an activity model or are sent to an already-trained classifier to identify the activity label. In Chapter 3, we described the data collection and preprocessing steps and discussed how raw sensor data can be represented as a set of high-level features. In Chapter 4, we described methods for supervised learning that map such a feature vector, X, onto class values.

In this chapter, we describe the remaining steps involved in the AR process. When humans perform activities, they do so fluidly, in such a way that consecutive activities blur into each other rather than being clearly delineated with sufficiently-large gaps between activities. They can also perform several activities in parallel or interweave

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

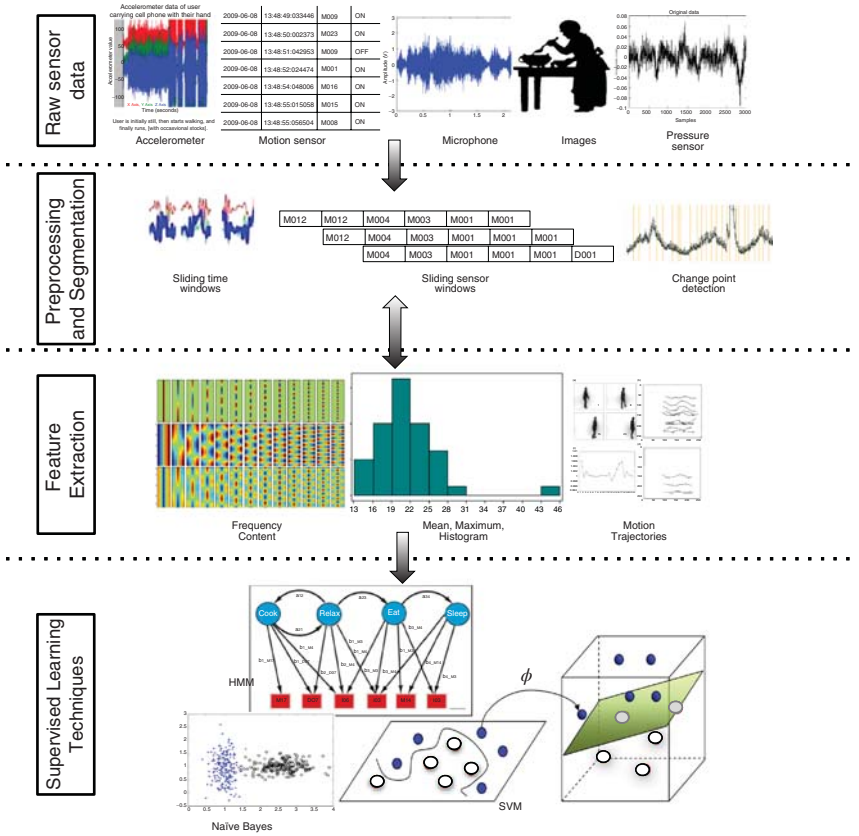


FIGURE 5.1 *The AR process includes the stages of raw sensor data collection, sensor preprocessing and segmentation, feature extraction and selection, classifier training, and data classification.*

them for maximum efficiency. As a result, the continuous stream of sensor-based information needs to be separated into pieces that can more clearly be distinguished by a trained classifier. Several approaches can be taken to address this issue of sensor event segmentation for AR. We introduce some of these methods, describe how they fit into the AR process, and discuss the related problem of activity spotting. We also overview methods for evaluating the performance of the resulting AR algorithms.

5.1 ACTIVITY SEGMENTATION

An activity recognition algorithm, AR, needs to recognize activities when they occur in unscripted settings, at the time that an individual (or a group) performs the activity. To achieve this goal, AR must map the state of the user and the corresponding environment at the current point in time to an activity label. The first task then is to select the

sensor events that comprise the current activity and define the corresponding context. The choice of segmentation or subsequence selection technique is critical because no classifier can produce meaningful results if the input features do not have discriminatory power. We consider two alternative approaches for making this selection: event segmentation and window sliding.

Event segmentation is a two-step process. In the first step, the sequence of streaming sensor events is separated into nonoverlapping subsequences or partitions. Each subsequence should be internally homogeneous and represent a single activity. AR can map each separate sequence to a corresponding activity label. The result of segmenting the data is a complete partitioning of the sensor events, as formalized in Definition 5.1.

Definition 5.1 *Given a sequence of n sensor events* $S = \langle e_1 \ \ldots \ e_n \rangle$, *an* event segmentation *partitions the sequence into a set of* x *subsequences* $P = \langle S_1, \ \ldots \ , S_x \rangle$, *such that each* $S_i \subseteq S$ *and the order of the elements in* S_i *is preserved. In addition, the set of subsequences is nonempty, nonoverlapping, and* $\bigcup_{i=1}^{i=x} S_i = S$.

Activity segmentation thus refers to the problem of segmenting a continuous stream of data into discrete meaningful units of activity execution. Each of these pieces, or segments, can be classified as an activity. For example, in Figure 5.2 the input sequence of sensor events is partitioned into the nonoverlapping subsequences

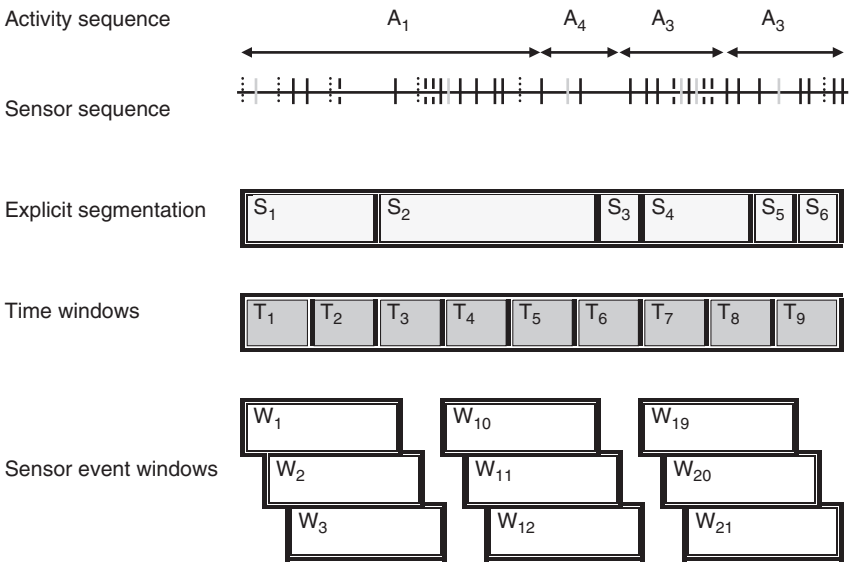


FIGURE 5.2 *Illustration of alternative approaches to processing sensor data streams for AR. Sensor event types and timings are depicted by the vertical lines, where the line type indicates the type, location, and value of the sensor message. The sensor windows are obtained using explicit segmentation, sliding window extraction with a fixed window time duration, or sliding window extraction with a fixed window length (number of sensor events).*

S_1.. S_6, each of which can then be mapped to an activity label. These subsequences represent activity segments and are defined by their corresponding start and end times in the sensor sequence. As we can see in Figure 5.1, the segments may not always align perfectly with activity boundaries and this needs to be taken into consideration when evaluating the entire AR process.

There are two classes of approaches that are common for event segmentation. The first relies on supervised machine learning, in which sample data is provided along with ground truth labels from which appropriate segment boundaries can be learned. The second utilizes features of the data itself, without supervised guidance, to identify activity boundaries in the event sequence data.

The first approach to sensor data segmentation that we will discuss relies on information about the activities that are modeled and are known to occur in the data sequence. Classifier-based and rule-based approaches can be designed to identify activity boundaries based on the known activity information.

Classifier-Based Segmentation When a supervised approach is employed for activity segmentation, machine learning algorithms are trained to recognize activity beginnings and endings (activity boundaries or breakpoints) or changes between activities (transitions) based on labeled examples. The sequence of sensor events between the activity beginning and ending, or between detected transitions, is partitioned into a separate sequence and used for activity labeling. Here we describe types of data samples that could be used to train a classifier to recognize activity boundaries.

- Examples of activity starts and stops. To employ this approach, a sufficient number of start and stop instances must be provided for each activity class. Thus for a set of A possible activities, $2A$ models need to be learned.
- Examples of activity transitions. The switches from one activity to another, or activity transitions, themselves can be explicitly modeled. The input to this type of learning problem consists of sensor events at the end of one activity combined with sensor events at the beginning of the next together with any events that occur between the two activities. In order to use this learning approach to activity segmentation, a model needs to be learned for every pair of activities, resulting in A^2 models. In addition, this approach runs into problems when sensor events may be observed that do not correspond to a known activity. In such situations, the transition may not be recognized because the individual is transitioning to or from an unknown activity. In other cases, the transition itself may be a separate activity that recurs and could be modeled. For example, transitioning to a Leave Home activity may itself comprise several steps such as collecting car keys, gathering supplies for an outing, and turning out lights. This could be labeled as Preparing to Leave Home and may be a predictable behavior that itself is recognizable. In these situations, the transition data could actually contain more sensor data than the activities that precede and follow the transition.

Another alternative is to treat transition recognition as a binary class problem, where the set of all possible transitions represents one class and all within-activity

sequences represents the second class. While only one model needs to be learned in this case, this is a much more complex problem that will be difficult to learn if the number of possible activities is large.

Unsupervised approaches to activity transition have also been successful. One well-known approach to this is change point detection. We will discuss this later when we review unsupervised methods for activity segmentation.

- Examples of activity occurrences. The idea here is that if each activity is modeled and a data point does not sufficiently follow any of the learned activity patterns, then it must represent a transition between the known activities. A one-class classifier can be trained on each known activity and used to "reject" points that do not fit into the known activities, in which case they represent transitions. A one-class classifier distinguishes examples of the target class from all other possible data points. With this method, A unique models must be learned in total. Alternatively, the sequence can be rejected if none of the activity models recognize the sequence with sufficient confidence. As with the activity transition method, a situation may occur where not all possible activities are modeled. In this situation, the rejected data points may represent other unknown activities and not simply transitions between known activities.

Rule-Based Segmentation An alternative method to locate activity boundaries is to identify points in the sensor event sequence when the data is better supported by two or more activities than by a single contiguous activity (which indicates a change of activities at that point in the sequence). A common method to detect this change is to construct rules that look for particular types of change in the sensor events. This approach is based on the assumption that activities tend to be clustered around particular types of sensor events or other easily detectable sensor features such as locations, times, or objects. From the perspective of sensor events, this means that the same types of sensor readings will occur each time a particular activity is performed. For each sensor type (in the case of discrete event sensors), the set of associated activities can be determined and stored. In the case of sampling-based sensors, the sensor values can be partitioned into ranges that occur when the activity is performed. The association of activities for each sensor event type or value range can be determined from domain knowledge or can be determined based on sample data.

As summarized in Figure 5.3, the sensor-activity mappings, stored in the vector MSA, can be used to detect activity boundaries. When neighboring sensor events $< e_{i-1},\ e_i >$ do not have any associated activities in common, this indicates that an activity boundary has been reached and a transition is occurring. In this case, event e_{i-1} is marked as the end of one activity and event e_i is marked as the beginning of the next activity. In addition to determining the activities associated with sensor events, other feature similarities (such as time of day) can be determined as well and used to identify activity boundaries.

The rule-based segmentation algorithm provides a fairly conservative approach to activity boundary detection for segmentation. Consider the sensor event streams

```
Algorithm RuleBasedSegment(S, MSA)
  // S is an input sequence of sensor events <e1 e2 .. en>
  // MSA contains the set of activities associated with each sensor event type or value range
  Boundaries = {}
  begin = 1
  Boundaries.append(begin)    // Indicate the beginning of the sequence is an activity boundary

  i = 1
  while i<n do
     i = i +1
     SAbegin = MSA(ebegin)       // Find the activities associated with the boundary sensor event
     SAi = MSA(ei)               // Find the activities associated with the current sensor event
     if SAbegin ∩ SAi = ∅
        end = i – 1
        Boundaries.append(end)
        begin = i
        Boundaries.append(begin)
     enf if
  done

  Boundaries.append(i)           // Indicate the end of the sequence is an activity boundary

  return Boundaries
```

FIGURE 5.3 *Rule-based activity segmentation algorithm.*

provided in Appendix A. Notice that there are some sensor types that are associated with both of the activities.

Example 5.1 Consider sensor "M017", which appears in both the Hand Washing and the Sweeping activities. In contrast, "WATER" appears only in the Hand Washing activity and "BURNER" appears only in the Sweeping activity. The MSAs for these sensors are thus MSA(M017)= {handwashing, Sweeping}, MSA(WATER)={HandWashing}, and MSA(BURNER)= {Sweeping}. If the following sensor sequence appears in the data stream:

```
10:00:00.00000  M017  ON
10:00:00.00000  M017  ON
10:01:00.00000  M017  ON
10:02:00.00000  WATER  ON
10:03:00.00000  BURNER  ON
10:04:00.00000  M017  ON
10:05:00.00000  M017  ON
10:06:00.00000  M017  ON
```

then a boundary will be detected between the WATER and BURNER events because their associated activities do not overlap. Note that some false negatives can result from this approach. If an M017 event occurred between the WATER and BURNER entries then no boundary would be detected because M017 belongs to both activities. This problem can be addressed by considering the probabilities of relationships

between sensor events and activities as well as considering a subsequence of events before or after the candidate activity boundary.

5.2 SLIDING WINDOWS

The second approach for handling streaming data is to divide the entire sequence of sensor events into a set of time ordered, possibly overlapping subsequences, or sliding windows. The windows follow the formalism given in Definition 5.2.

Definition 5.2 *Given a sequence of n sensor events* $S = \langle e_1, \ldots, e_n \rangle$, event windowing *identifies a set of x windows,* $P = \langle S_1, \ldots, S_x \rangle$,, *with window sizes* $\{w_1, \ldots, w_n\}$, *such that each* S_i *is an ordered subsequence of S. The set of windows is ordered, nonempty, possibly overlapping, and* $\bigcup_{i=x}^{i=x} S_i \supseteq S$. *Window* S_i *can thus be represented by the sequence* $< e_i, e_{i+w_i} >$.

Using a sliding window algorithm, the last (most recent) sensor event in each window S_i is mapped to an activity label by an AR algorithm based on the learned mapping $h: S \rightarrow A$. The sequence of sensor events in the window provides a context for making an informed mapping. Because the windows are ordered, each window can be mapped to an activity label as it occurs, which makes this a valuable approach when labeling activities in real time from streaming data. This technique offers a simpler approach to learn the activity models during the training phase over the explicit segmentation approach. Furthermore, it reduces the computational complexity of AR over the explicit segmentation process. This AR technique can also be used to facilitate *activity spotting*, which is the process of locating instances of a particular activity from a continuous data stream in which the activity is mixed with background noise and irrelevant actions.

There still remains a number of decisions to make with a sliding windowing approach. First, window sizes need to be determined based on the appropriateness for the context and the type of activities that will be recognized. Second, events may need to be weighted within a window based on their relevance to the current context.

5.2.1 Time Based Windowing

One approach to handling sliding windows is to divide the entire sequence of sensor events into equal-size time intervals as illustrated in Figure 5.1 by the subsequences denoted as $T_1, T_2, \ldots T_9$. This is referred to as *timestamp-based sliding windows*. In this approach, the timestamp of the sensor event is an important parameter that is used in describing each sensor event and generating activity feature vectors.

Using this approach, parameter w_i refers to a time duration. This is a reasonable approach when dealing with data obtained from sensors that sample their state at constant time intervals. In such a scenario, every window is guaranteed to contain a fixed amount of data. This is a common approach when using accelerometers and gyroscopes, for example, where data is sampled at a constant rate from the sensors. However, one has to deal with the problem of selecting the optimal length of the time

interval. If a very small interval is chosen, there is a possibility that it will not contain any relevant activity information for making any useful decision. If the time interval is too wide, then information pertaining to multiple activities can be embedded into it and the activity that dominates the time interval will have a greater influence in the classification decision. This problem manifests itself when dealing with sensors that do not have a constant sampling rate. In the current context of motion and door sensor events, it is very likely that some time intervals do not have any sensor events in them (e.g., T_6 in Figure 5.2). One approach that can be taken when the optimal window size is unknown is employ an ensemble of classifiers, as described in Chapter 4, each of which is trained for a different window size.

5.2.2 Size Based Windowing

The second approach to defining window sizes is to divide the sequence into windows containing an equal number of sensor events. This is commonly referred to as *bursty*, *fixed-size*, or *sequence-based sliding windows*. The bursty approach is valuable when data arrives asynchronously, as occurs with discrete-event sensors or when external events are included in the analysis, thus the arrival rate of the data is not constant. Using this approach, parameter w_i refers to a number of sensor events. This is illustrated in Figure 5.2 by the subsequences denoted as W_1, W_2, … , W_{21}. Even if the size of the window (defined by number of sensor events in the window) is fixed, these windows may actually vary in their time duration. This is appropriate considering that during the performance of highly-mobile activities, multiple motion sensors could be triggered, while during more sedentary activities or silent periods, there will few, if any, sensor events. The sensor events preceding the last event in a window define the context for the last event.

Like the previous approach, this method also has some inherent drawbacks. For example, consider the sequence W_{11} in Figure 5.2. The last sensor event of this window corresponds to the beginning sensor event of activity A_3. It is possible that there exists a significant time lag between this event and its preceding sensor event. The relevance of all the sensor events in this window to the last event in the window might be minimal if the time lag is large. As a result, treating all the sensor events with equal importance may result in loss of recognition effectiveness. Note also that if all of the sensors provide constant-time sampling, the time based windowing and size based windowing approaches will yield the same results.

In the presence of multiple residents, sensor firings from two different activities performed by the different residents will be grouped into a single window, thereby introducing conflicting influences for the classification of the last sensor event. While by itself this approach may not be intuitively alluring, we will show that events can be weighted within the window to account for the relationship between the sensor events. This type of windowing approach does offer computational advantages over the explicit segmentation process and can perform in real time because it does not require knowledge of future sensor events to classify past or current sensor events.

The value of the window size parameter, w, may depend on the context in which AR is performed. The value can be derived through an empirical process by studying

the effect of the different values of w on the performance of the recognition system. Among the different factors that influence the value for w is the average number of sensor events that span the duration of alternative activities. At one end of the spectrum are activities such as Leave Home that may be defined by rapid firing of a small set of environment sensors, while at the other extreme is the activity Sleep that continues for hours but typically results in an occasional firing of one or two sensors due to minimal resident movement during this time. Ideally, the size of a sensor event window should be large enough to define the context of the last sensor event. Heuristics such as the average length of the longest recognizable activity can be used to bound the window size.

5.2.3 Weighting Events Within a Window

Once the sensor window S_i is defined, the next step is to transform this window into a feature vector that captures relevant activity information content as described in Chapter 3. However, one of the problems associated with fixed-size windowing is that windows could contain sensor events that are widely spread apart in time. An illustration of this problem is presented in Figure 5.4. These are examples of a sequence of discrete-event sensor events collected by environment sensors in a smart home. Notice the time stamp of the last two events in the sequence in Figure 5.4. There is a gap of nearly one and a half hours between these sensor events. All the sensor events that define the context of the last event within this window have occurred in the "distant" past. In the absence of any weighting scheme, the feature vector may be biased toward the inclusion of misleading information. Even though the sensor event corresponding to the end of the Personal Hygiene activity occurred in the past, it would have an equal influence on defining the context of the event corresponding to Enter Home. To overcome this problem, a time-based weighting scheme can be incorporated to take into account the relative temporal distance between the sensors.

W2009-07-19	10:18:59.406	LivingRoom	ON	Personal_Hygiene
2009-07-19	10:19:00.406	Bathroom	OFF	Personal_Hygiene
2009-07-19	10:19:03.015	OtherRoom	OFF	Relax
2009-07-19	10:19:03.703	LivingRoom	OFF	Relax
2009-07-19	10:19:07.984	LivingRoom	ON	Relax
2009-07-19	10:19:11.921	LivingRoom	OFF	Relax
2009-07-19	10:19:13.203	OtherRoom	ON	Relax
2009-07-19	10:19:14.609	Kitchen	ON	Relax
2009-07-19	10:19:17.890	OtherRoom	OFF	Relax
2009-07-19	10:19:18.890	Kitchen	OFF	Relax
2009-07-19	10:19:24.781	FrontMotion	ON	Leave_Home
2009-07-19	10:19:28.796	FrontMotion	OFF	Leave_Home
2009-07-19	10:19:31.109	FrontDoor	CLOSE	Leave_Home
2009-07-19	12:05:13.296	FrontDoor	OPEN	Enter_Home

FIGURE 5.4 *Illustration of time dependency in a sliding window of sensor events.*

When the sliding window is a constant size, it is possible for two sensor events that are spread apart in time to be part of the same window. In order to reduce the influence of such sensor events on deciding the activity label for the most recent sensor event, a time-based weighting factor can be applied to each event in the window based on its relative time to the last event in the window.

Let $\{t_{i-w}, \ldots, t_i\}$ represent the time stamps of the sensor events in window S_i. For each sensor event e_j, the difference between the time stamp of e_j and the time stamp of e_i, the last event in the window, is computed. The contribution, or weight, of sensor event e_j can be computed using an exponential function as shown in Equation 5.1.

$$C(i,j) = \exp\left(-\varkappa^{(t_i - t_j)}\right) \tag{5.1}$$

Features that are based on a simple count of the sensor events within a window can now be replaced by a sum of the time-based contributions of each sensor event within the window. Features which sum the values of sensor events can also employ this type of weighting approach to adjust the influence of each sensor event on the feature vector. The value of $\varkappa$ determines the rate of decay of the influence. Figure 5.5 shows the effect of the choice of $\varkappa$ on the rate of decay. If $\varkappa > 1$, then only sensor events that are temporally close to the last event contribute to the feature vector. When $0 < \varkappa < 1$, the feature vector is under the influence of a temporally wider range of sensor events. When $\varkappa = 0$, the temporal distance has no influence on the feature vector, making it a simple count of the different sensor events.

Similarly, in situations when the sensor event corresponds to the transition between two activities (or in other settings when multiple activities are performed by more than

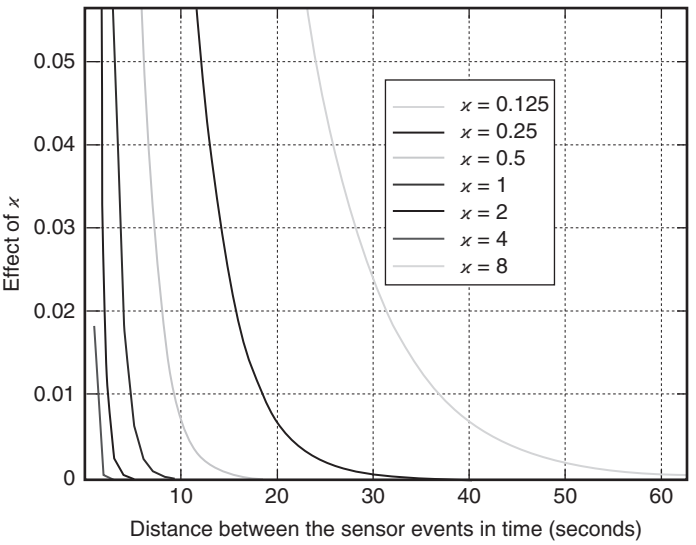


FIGURE 5.5 *Effect of $\varkappa$ on weights.*

2009-07-23	19:59:58.093	Bathroom	ON	Personal_Hygiene
2009-07-23	20:00:02.390	Bathroom	OFF	Personal_Hygiene
2009-07-23	20:00:04.078	Bathroom	ON	Personal_Hygiene
2009-07-23	20:00:08.000	LivingRoom	ON	Relax
2009-07-23	20:00:08.640	OtherRoom	ON	Relax
2009-07-23	20:00:09.343	LivingRoom	OFF	Relax
2009-07-23	20:00:12.296	Kitchen	ON	Relax
2009-07-23	20:00:25.140	LivingRoom	OFF	Relax
2009-07-23	20:00:27.187	FrontMotion	ON	Leave_Home
2009-07-23	20:00:27.437	Kitchen	OFF	Leave_Home
2009-07-23	20:00:30.140	FrontMotion	OFF	Leave_Home
2009-07-23	20:00:32.046	FrontMotion	ON	Leave_Home
2009-07-23	20:00:36.062	FrontMotion	OFF	Leave_Home
2009-07-23	20:00:39.343	FrontMotion	ON	Leave_Home
2009-07-23	20:00:43.671	FrontMotion	OFF	Leave_Home
2009-07-23	20:00:46.265	FrontDoor	CLOSE	Leave_Home

FIGURE 5.6 *Illustration of sensor dependency in a sliding window of sensor events.*

one resident in parallel), the events occurring in the window might not be related to the sensor event under consideration. An example of this situation is illustrated in Figure 5.6. This particular sequence of sensor events from a smart home testbed represents the transition from the Personal Hygiene activity to the Leave Home activity. Notice that all the initial sensor events in the window come from a bathroom in the home, whereas the second set of sensor events is from an unrelated functional area of the apartment, namely the area near the front door. While this certainly defines the context of the activity, since the sensors from a particular activity dominate the window, the chances for a wrong conclusion about the last sensor event of the window are higher. This problem can be addressed by defining a weighting scheme based on the mutual information (MI) between the sensors.

The MI measure reduces the influence of sensor events within the window that do not typically occur within the same time frame as the last sensor event in the window. In the context of environmental sensors, motion sensors that are geographically distant from the most recent sensor location will receive less weight than those that are close. MI is typically defined as the quantity that measures the mutual dependence of two random variables. For sensor-based AR, each individual sensor is considered to be a random variable. The MI or dependence between two sensors is then defined as the chance of these two sensors occurring successively in the entire sensor stream. If S_i and S_j are two sensors, then the MI between them, $MI(i,j)$, is defined as

$$MI(i,j) = \frac{1}{N}\sum_{k=1}^{N-1} \delta\left(s_k, S_i\right) \delta\left(s_{k+1}, S_j\right) \tag{5.2}$$

where

$$\delta(s_k, S_i) = \begin{cases} 0 & \text{if } s_k \neq S_i \\ 1 & \text{if } s_k = S_i \end{cases} \tag{5.3}$$

The summed term thus takes a value of 1 when the current sensor is S_i and the subsequent sensor is S_j. If two sensors are adjacent to each other, such that triggering one sensor most likely results in also triggering the other sensor, then the MI between these two sensors will be high. Similarly, if the sensors are far apart such that they do not often occur together, then the MI between them will be low. Note that the computation of MI using this bi-gram model depends on the order in which sensor events occur. The MI matrix is typically computed offline using sample data, with or without activity labels, from the set of sensors that will be employed for AR.

Example 5.2 Consider the event sequence shown in Figure 5.6. There are 16 events or 15 pairs of successive events (N = 15). The sequence {`Bathroom, Livingroom`} appears one time and {`LivingRoom, Bathroom`} does not appear at all, so `MI(Bathroom, LivingRoom)=1/15`. In contrast, `MI(Bathroom, FontDoor)=0` and `MI(LivingRoom, Kitchen)=2/15`.

Once computed, the MI matrix can then be used to weight the influence of sensor events in a window while constructing the feature vector. Similar to time-based weighting, each event in the window is weighted with respect to the last event in the window. Thus instead of computing feature values based on the count of different sensor events or an aggregation of the sensor values, it is the aggregation of the contribution of every weighted sensor event based on MI that can is included in the feature vector representing the data in the sliding window.

While Equations 5.2 and 5.3 are based on the occurrence of events from a particular sensor, a similar approach can be used to weight sensor events with values that do not commonly occur in the same window with other sensor values. By weighting sensors according to mutual information, the impact of sensor events can be reduced when their occurrence is due to noise, to interweaving of activities, or to activities of multiple individuals who are being detected by the same sensors.

Example 5.3 Figure 5.7 shows intensity-coded MI scores between different sensors located in a smart home. It is evident from the figure that each of the sensors is functionally very distinct from the others. Furthermore, the relatively strong diagonal elements indicate the higher chance of sensor generating multiple similar messages in a row rather than transitioning to different sensors. Because each of these sensors is triggered by human motion, the MI values for this testbed indicate that the resident tends to remain in one location more than moving around. A few other observations can be made from Figure 5.7. For example, consider the similarities between sensors 5 and 6. These two sensors correspond to the *Front door* and *Kitchen* sensors that are geographically close to each other. As a result, when the resident enters the testbed, the kitchen sensor is likely to sense motion as well as the front door motion sensor. However, the *Kitchen cabinet* sensors (#7) do not get triggered. Another subtle observation is the relatively high similarity between the *Medicine cabinet* sensor (#13), the *Kitchen* sensor, and the *Kitchen cabinet* sensors (#7). The resident of this home stores medicine in the kitchen cabinet. When the resident retrieves medicine each day all of the related kitchen sensors have a relatively high likelihood of firing.

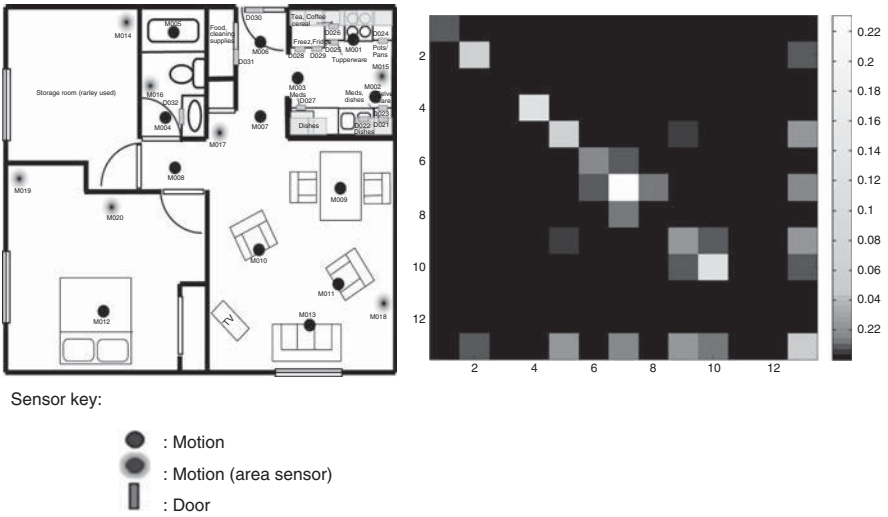


FIGURE 5.7 *Intensity-coded MI between every pair of motion sensors (right) in a smart home (left).*

5.2.4 Dynamic Window Sizes

The previously described approaches employ a fixed window size for computing the feature vectors. An important challenge with this approach is identifying the optimal window size. Various heuristics such as the mean length of the activities and sampling frequency of the sensors can be employed to determine the window size. For fixed-sized windows, lengths between 5 and 30 are common. For time-based windows containing events from high-frequency sampled sensors, common sizes are between 1 and 6 seconds. An alternative approach, however, is to automatically derive the window size from the data itself.

A probabilistic method can be used to derive an appropriate window size for each sensor event that is being classified with an activity label. Assuming that there are M activity labels, a set of candidate window sizes is defined, $\{w_1, w_2, \ldots, w_L\}$, where w_1 corresponds to the minimum window size that is observed in the data for any activity, $w_1 = \min\{ws(A_1), ws(A_2), \ldots, ws(A_M)\}$, w_L is the median window size observed for all of the activities in A, $w_L = \text{median}\{ws(A_1), ws(A_2), \ldots, ws(A_M)\}$, and the remaining candidate window sizes are obtained by dividing the $[w_1, w_L]$ interval into equal-length bins. After generating candidate window sizes, the most likely window size for an activity A_m can be calculated as

$$w^* = \text{argmax}_{w_l}\{P(w_l|A_m)\} \tag{5.4}$$

Note that we can also estimate the probability of an activity A_m being associated with sensor s_i as $P(A_m|s_i)$. Thus, if the most recent sensor event is generated by sensor

s_i, the most likely activity $A*$ associated with the sensor is calculated as

$$A^* = \text{argmax}_{A_m}\{P(A_m|s_i)\} \tag{5.5}$$

To consider both of these influences in selecting an optimal size, we can combine Equations 5.4 and 5.5. The optimal window size for events generated by sensor s_i can be determined by combining the two equations according to the factorization in Equation 5.6.

$$w^* = \text{argmax}_{w_l} P(w_l|s_i) = \text{argmax}_{w_l}[P(w_l|A_m) * P(A_m|s_i)] \tag{5.6}$$

Each of the probability vectors can be estimated using available sample data. The window sizes for the sensor events that are used to train AR classifiers and to label new events are computed using the probabilities estimated from the sample data.

A similar approach can be taken to dynamically compute time-based window sizes. As with size-based windows, time windows can be defined for every sensor event. For a sensor event s_i, a time window consists of all sensor events that fall within a fixed time duration starting from the timestamp of event s_i. The sensor events within a particular time window are aggregated to form features using the same process that was adopted for length-based windows. The time window can be labeled using the activity associated with the last sensor event in the time window.

Example 5.4 As an example, consider a dataset that is formed by combining the events in Figures 5.4 and 5.6. For this dataset, the window sizes for the activities that occur are

Personal hygiene	{2, 3}	Relax	{5, 8}
Leave home	{3, 8}	Enter home	{1}

The minimum window size is 1 and the median size is 3. We consider the candidate window sizes 1 and 3. Consider that the most recent sensor event is generated by a *Front door* sensor. We see from the data that P(LeaveHome | FrontDoor) = 0.67 and P(EnterHome | FrontDoor) = 0.33, so the most likely activity given this sensor event is Leave Home. For the Leave Home activity, the only window sizes that are observed are sizes 3 and 8, each with probability 0.5. The probability of window size 1 for Leave Home is 0.0, so the most likely window size of 3 would be used to determine the context for this activity.

5.3 UNSUPERVISED SEGMENTATION

Although activity-centered segmentation allows activity breakpoints to be easily detected, it relies on knowledge of the predefined activities and sufficient training

data to learn models of activities and their boundaries. It is more difficult to leverage activity-centered approaches to detect breakpoints within activities that are highly variable and do not appear in the vocabulary of predefined tasks. This is a limitation, because such free-form tasks are very common in routine behavior. Here we describe two alternative approaches, change point detection and piecewise representation, that can be used to identify potential activity boundaries without explicit guidance from a supervised learning algorithm.

Change Point Detection An alternative method for activity segmentation is to detect activity breakpoints or transitions based on characteristics of the data without being given explicitly labeled examples of activity breakpoints or transitions. We have already shown a number of approaches that exist for segmenting time series data. However, in some cases we want to (i) identify likely activity boundaries in real time as the data is observed and (ii) perform the detection without explicit training a classifier on known activities or known activity transitions.

To accomplish this, we can borrow from time series analysis to perform change point detection, which is the problem of detecting abrupt changes in time series data. Such abrupt changes may represent transitions that occur when individuals complete one activity and begin another activity. To perform change point detection, probability distributions of the feature descriptions can be estimated based on the data that has been observed since the previous candidate change point. In order to ensure that change points are detected in real time, the algorithm only tries to identify the most recent change or activity boundary. Sensor data from the current point in time back to the most recent change point is referred to as a run, r. If t sensor events or time units have occurred since the most recent detected change point, then there are t possible runs of different lengths that could be associated with the recent data. If the algorithm is applied after each new sensor event is received, then there are only two possibilities to consider: either the new event is a member of the current run and the run length increases by one, or a change point occurred, the run ended, and the new sensor event is part of a new run. Let r_t refer to the length of the most recent run. We can then calculate the probability of r_t given the previous t sensor events.

$$P\left(e_{1:t+1}|e_{1:t}\right) = \sum_{r_t} P\left(e_{t+1}|r_t \& e_{1:t}\right) P\left(r_t|e_{1:t}\right) \tag{5.7}$$

We can compute the probability of the run length after event t given the corresponding sequence of sensor events using Bayes rule.

$$P\left(r_t|e_{1:t}\right) = \frac{P\left(r_t \& e_{1:t}\right)}{P\left(e_{1:t}\right)} \tag{5.8}$$

The numerator can then be recursively computed based on the previous run length and corresponding sensor events.

$$
\begin{aligned}
&= \sum_{r_{t-1}} P\left(r_t \,\&\, r_{t-1} \,\&\, e_t\right) \\
P\left(r_t \,\&\, e_{1:t}\right) &= \sum_{r_{t-1}} P\left(r_t \,\&\, e_t | r_{t-1} \,\&\, e_{1:t-1}\right) P\left(r_{t-1} \,\&\, e_{1:t-1}\right) \\
&= \sum_{r_{t-1}} P\left(r_t | r_{t-1}\right) P\left(e_t | r_{t-1} \,\&\, e_{1:t-1}\right) P\left(r_{t-1} \,\&\, e_{1:t-1}\right)
\end{aligned}
\tag{5.9}
$$

The algorithm to compute the probabilities does not require storing all of the intermediate probability values but can use the probabilities estimated up through the previous sensor event to compute the probabilities for the current sensor event. The process starts at the previous change point, so $P(r_0 = 0) = 1$. The term $P(e_t | r_{t-1} \,\&\, e_{1:t-1})$ represents the probability that the most recent sensor event belongs to each of the possible runs identified through the previous data point. Calculating this probability relies on knowing the type of data distribution. We let e_t^r represent the data point that is associated with run r_t and π_t^r represent the probability of data point e_t^r based on the run through t and the appropriate data distribution.

The probability of r_t given the previous value has two cases depending on whether a change point just occurred or whether a change point did not occur and the run length grew by one. To compute these two cases, we first compute the hazard function H. The hazard function is used to represent a failure rate, which is estimated here as the number of events that end a run (or segment) to the total number of events, based on sample data.

$$
P\left(r_t | \, r_{t-1}\right) = \begin{cases} H\left(r_{t-1} + 1\right) & \text{if } \; r_t = 0 \\ 1 - H\left(r_{t-1} + 1\right) & \text{if } \; r_t = r_{t-1} + 1 \end{cases}
\tag{5.10}
$$

The change point detection algorithm is summarized in Figure 5.8. As the pseudocode shows, a change point can be detected and the corresponding activity boundary made after looking ahead only one sensor event. The parameters of the predictive distribution associated with a current run are indicated by X_t^r. Simple distributions such as the marginal predictive distribution can be used when the exact distribution of the data is unknown.

For detecting activity transitions, the sequence data is constructed as follows. The current state of the environment (including the sensors and the residents or the individual wearing sensors) is viewed as a single data point. This gives us a multidimensional data point where the number of dimensions corresponds to the number of features describing the state. The current state is then updated at every time step, or after every sensor event, to yield the sequence data. The probability of a run ending before the current time point or sensor event is computed and compared with the probability of the run extending to include the current data point. If the probability of the run ending is larger, then the activity boundary is noted before the most recent event and the process repeats with the new run starting at the most recent sensor event.

Algorithm **DetectChangePoint**()

```
Boundaries = {}
begin = 1
Boundaries.append(begin)  // Indicate the beginning of the sequence is an activity boundary
P(r_0 = 0) = 1            // The initial run length is 0
X_1^0 = X_prior           // Initialize the parameters of the data distribution model based on priors
i = 1
while data do
    i = i + 1
                          // Calculate the probability that the run length grows by 1
    P(r_i = r_{i+1} + 1 & e_{1:i}) = P(r_{i-1} & e_{1:i-1}) x π_i^r x (1 - H(r_{i-1} + 1))
                          // Calculate the probability that a change point occurred
    P(r_i = 0 & e_{1:i}) = Σ_{r_{i-1}} P(r_{i-1} & e_{1:i-1}) x π_i^r x H(r_{i-1})
                          // Update the data evidence probability
    P(e_{1:i}) = Σ_{r_i} P(r_i & e_{1:i})
                          // Determine the run length distribution
    P(r_i | e_{1:i}) = P(r_i & e_{1:i}) / P(e_{1:i})
                          // Update the distribution statistics
    X_{i+1}^{r+1} = X_prior
                          // Predict the next data point
    P(e_{i+1} | e_{1:i}) = Σ_{r_i} P(e_{i+1} | e_i^r & r_i) x P(r_i | e_{1:i})|
    if P(r_i = 0) > P(r_i = r_{i+1} + 1 )
        end = i - 1
        Boundaries.append(end)
        begin = i
        Boundaries.append(begin)
    end if
done
return Boundaries
```

FIGURE 5.8 *Online change point detection algorithm.*

Piecewise Representation The idea of piecewise approximation originates from time series analysis, in which the entire series cannot be accurately represented by a single simple function yet it can be segmented into pieces that can be individually represented by such functions. For an unsupervised approach to segmentation, we can try to model each segment with a simple regression or classifier model. Supervised versions of piecewise approximation can use recognition accuracy based on pre-trained activity models to determine the value of segment choice.

Given a method of representing each segment and an error function to determine the fit between the function and the data, we can employ several different techniques to efficiently search through the space of possible segment choices. A *top down* approach starts by viewing the entire sequence as a segment and chooses a point in the

sequence (if one exists) where splitting the sequence into two smaller segments results in decreased error over representing the segments separately. The process repeats until performance is not improved by further splitting. A *bottom up* method starts by considering each event as a separate segment and merges neighboring segments until the performance does not improve with further merging. These techniques are similar to hierarchical clustering techniques that have been applied to nonsequential data. A third *sliding window* technique starts at the beginning of the input sequence with a segment of size one and grows the segment until the performance does not improve with further extensions of the segment. The process then repeats starting with the first data point not included in the most recent segment.

These techniques are effective for a variety of types of data with or without the use of trained activity models. However, they are designed to be applied in offline mode to the entire sequence of available sensor event data. A hybrid method can be used to combine these approaches in a semi-online approach. Using the hybrid algorithm, a subsequence is stored in a buffer that is long enough to store approximately five typical segments. The bottom-up technique is used to segment the data in the buffer and the left-most segment boundary is recorded. The data corresponding to the segment is removed and the next data points in the sequence are added to the buffer to maintain a constant-size buffer, then the process is repeated. This hybrid technique provides more of a "look ahead" then is used by a pure sliding window and thus the segment quality can be improved. Unlike a pure bottom up method, however, this hybrid method allows the data to be processed in near real time. Figure 5.9 provides pseudocode for the hybrid segmentation algorithm.

5.4 MEASURING PERFORMANCE

Once a sequence of sensor events is partitioned into overlapping windows or nonoverlapping windows (segments), the AR algorithm uses one of the classifiers described in Chapter 4 to map each window's subsequence of sensor events s_i to an activity label A_i based on the learned mapping $f : S \rightarrow A$. In order to compare alternative AR algorithms and to estimate the expected performance of the recognition for future data, we want to be able to determine the performance of the algorithm on already available data as well as data we may collect in the future.

Example 5.5 To illustrate the performance evaluation process, consider an example scenario in which we train a Decision Stump classifier to distinguish our Hand Washing activity from the Sweeping activity as described in Chapter 3. As mentioned in Chapter 3, a decision stump classifier is a one-level decision tree. An attribute is selected to be queried as the root of the tree and its children are leaf nodes, which provide an activity classification. To simplify the recognition scenario, we utilize a fixed sliding window of 10 sensor events with only two types of features: the time duration of the window and the sensor counts (bag of sensors) for each window. Due to the choice of features, we only consider the discrete event sensors. This includes the motion sensors, door sensors, item sensors, and utilization of water and the stove

```
Algorithm HybridSegment(Buffer, BufferSize)

  // BufferSize is specified as 5 times the size of a typical activity segment
  Boundaries = {}
  begin = 1
  Boundaries.append(begin)    // Indicate the beginning of the sequence is an activity boundary

  while data do
     error = ModelError(Buffer)
     Segment = BottomUp(Buffer, error)
     end = begin + Segment[1].size
     Boundaries.append(end)
     begin = end + 1
     Boundaries.append(begin)
     Buffer.remove(Segment[1].size)          // Remove the first segment from the buffer
     Buffer.append(data, Segment[1].size)    // Add the next Segment[1].size events to the buffer
  done

  return Boundaries


Algorithm BottomUp(Seq, MaxError)
  Segment = {}                 // Segment contains the size of each segment in the input sequence

  for i = 1 to Seq.size - 1
     S.begin = i
     S.end = i
     S.size = 1
     Segment.append(S)
  done

  for i = 1 to length(Segment) - 1
     MergeCost[i] = ModelError(Merge(Segment[i], Segment[i+1]))
  done

  while min(MergeCost) < MaxError
     i = min(MergeCost)
     S[i] = Merge(Segment[i], Segment[i+1])
     Segment.remove(S[i+1])
     if i < length(Segment)
        MergeCost[i] = ModelError(Merge(Segment[i], Segment[i+1]))
     end if

     if i >1
        MergeCost[i-1] = ModelError(Merge(Segment[i-1], Segment[i]))
     end if
  done

  return segment
```

FIGURE 5.9 *Hybrid bottom-up/sliding-window segmentation algorithm.*

burner. Note that with a window size of 10 the Hand Washing activity contains 10 data points while the sweeping activity contains 44 data points.

A first step at evaluating the performance of an AR algorithm is to generate a table that summarizes how the activities were classified, typically referred to as the confusion matrix. Each row of the matrix represents data points and their actual activity

TABLE 5.1 Example Confusion Matrix

	Classified as 1	Classified as 0
True 1	a (TP)	b (FN)
True 0	c (FP)	d (TN)

label, while each column of the matrix represents the data points and the activity label they were assigned by the classifier. As shown in Table 5.1, each cell $[i, j]$ provides the number of data points from activity class i that were categorized as activity j by the classifier. Notice that the cells along the diagonal indicate the correct classifications. A classifier that provides correct labels for each data point would generate zero values for every cell except those along the diagonal. Thus the values represented by "a" and "d" represent the number of correctly classified data points for this two-class problem while the values represented by "b" and "c" represent the number of incorrectly classified data points.

Example 5.6 Table 5.2 summarizes the results from our activity example. Mistakes are made in labeling data points from both classes. The sweeping class has fewer errors, which could be due to its larger number of training data points. To shed more light on the performance of the recognition algorithm, we next introduce a range of metrics that could further evaluate the algorithm.

While Table 5.2 shows the confusion matrix that results from learning two activity classes, the techniques can be applied to any number of activity classes. Table 5.3 shows the confusion matrix that results from applying a naive Bayes learner to the Hand Washing and Sweeping activities as well as a third activity, Cooking. The Cooking class contains 32 examples that represent data collected while an individual

TABLE 5.2 Confusion Matrix Using Decision Stump for Recognizing Hand Washing and Sweeping Activities

		Classifier Label Hand Washing	Sweeping
Activity Label	Hand Washing	6	4
	Sweeping	3	41

TABLE 5.3 Confusion Matrix Using Naive Bayes for Recognizing Hand Washing, Sweeping, and Cooking Activities

		Classifier Label Hand Washing	Sweeping	Cooking
Activity Label	Hand Washing	9	1	0
	Sweeping	0	44	0
	Cooking	0	0	32

cooked a bowl of soup. Note that even though the number of classes has increased, the naive Bayes classifier actually classifies more data points correctly than the decision stump did for two activity classes.

Many performance metrics have been introduced to evaluate classifier algorithms. These are useful for evaluating AR algorithms. However, the AR problem has some unique characteristics that are not captured by traditional metrics so additional measures need to be considered as well. In this section, we provide an overview of both, using our Hand Washing versus Sweeping problem to illustrate the metrics.

5.4.1 Classifier-Based Activity Recognition Performance Metrics

Accuracy The most common metric for evaluating classifier performance is accuracy. Accuracy is calculated as the ratio of correctly-classified data points to total data points. Referring to the example confusion matrix in Table 5.1, we notice that accuracy can be calculated from the matrix by summing the values along the diagonal and dividing by the sum of all of the values in the matrix, or $(a + d)/(a + b + c + d)$.

$$\text{Accuracy} = \frac{\#\text{correctly classified activities}}{\#\text{total activities}} \tag{5.11}$$

Accuracy is a measure that can be applied to multi-class problems as well as binary-class problems. This will be necessary for most AR algorithms (just the "clean house" category in Table 2.1 alone lists 23 separate activities to be recognized). However, accuracy does not tell the whole story, because it does not provide insights on the source of the error or the distribution of error among the different activity classes.

Example 5.7 For our AR example in Table 5.2, we observe that the accuracy is 0.87. However, the accuracy for the individual classes varies quite a bit (the accuracy for Hand Washing is 0.60 and the accuracy for Sweeping is 0.93). The companion to accuracy is *Error Rate*, which is computed as 1 - Accuracy.

Sensitivity, Specificity Accuracy and Error Rate give a high-level idea about the recognizer's performance. However, in cases where there is a nonuniform distribution among the activity classes, these measures can be deceiving. Accuracy and Error rate are ineffective for evaluating classifier performance in a class-imbalanced dataset because they consider different types of classification errors as equally important. For example, if the dataset consists of 5% Eat activities and 95% Sleep activities, a random prediction of all of the test instances being Sleep will yield an accuracy of 0.95. However, in this case the classifier did not correctly recognize any of the Eat activities. In order to provide a comprehensive assessment of AR, we need to either consider metrics that can report the performance of AR on two classes separately or not let the effect of class imbalance be reflected in the metric. Sensitivity, specificity, and g-mean are useful for reporting recognizer performance in such cases of class imbalance and for providing insight on recognition performance for individual classes. Sensitivity, also referred to as the *true positive rate* or *true activity rate* (*TP Rate*), refers to the

portion of a class of interest (the positive class) that was recognized correctly. Referring to the example confusion matrix in Table 5.1, we see that the true positive rate is $a/(a+b)$. For a problem with more than two activity classes, the true positive rate would be calculated in a similar manner.

Example 5.8 Using the example in Table 5.3, if Hand Washing is the positive class then the true positive rate would be calculated as the number of Hand Washing examples classified as Hand Washing divided by the total number of Hand Washing examples, or $9/10 = 0.9$.

$$\text{Sensitivity} = \text{TP Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.12}$$

Similarly, the *false positive rate* (*FP Rate*) can be reported as the ratio of negative examples (examples not in the class of interest) classified as positive to the total number of negative examples. The false positive rate for the confusion matrix in Table 5.1 is computed as $c/(c+d)$.

$$\text{FP Rate} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{5.13}$$

In contrast, specificity refers to the portion of the negative class examples (examples not in the class of interest) that were recognized correctly.

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{5.14}$$

G-mean G-mean utilizes the Sensitivity and Specificity measures of the performance of the AR algorithm both in terms of the ratio of positive accuracy (Sensitivity) and the ratio of negative accuracy (Specificity). G-mean thus provides activity class-sensitive measure of the recognition performance. The sensitivity, specificity, and g-mean scores can be calculated separately for each class to see how well the recognition algorithm handles each activity. They can also be combined by averaging the scores over each class, multiplied by the class size to provide a weighted average and thus an overall performance measure for the algorithm. Note that the weighted average sensitivity and specificity scores yields the same result as the accuracy calculation.

$$\text{G-mean} = \sqrt{\frac{\text{TP}}{\text{TP} + \text{FN}} \times \frac{\text{TN}}{\text{TN} + \text{FP}}} = \sqrt{\text{Sensitivity} \times \text{Specificity}} \tag{5.15}$$

Precision, Recall, f-measure Like sensitivity and specificity, precision and recall provide class-sensitive measures. Precision is calculated as the ratio of true positive data points to total points classified as positive, while recall is calculated as the ratio of true positive data points to the total points that are true (should be classified as true).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.16}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.17}$$

While precision and recall can be used to evaluate AR performance, they are traditionally used to evaluate the performance of retrieval approaches, such as document retrieval algorithms. In the context of activity learning, these can be useful measures if the goal is not to just label a particular data sequence with an activity name but to search through a set of sensor event sequences to retrieve all of the sequences in which a particular activity occurs. In this case, precision can be used to determine the proportion of identified sequences that are relevant (actually contain the activity of interest). This can be used, for example, to determine on which days a caregiver visited to provide physical therapy for a smart home resident. Similarly, recall can be used in this situation to calculate the fraction of relevant sequences that are actually identified and retrieved by the algorithm. In these cases, TP is measured as the number of relevant identified sequences. The precision denominator, TP + FP, is the total number of identified sequences, and the recall denominator, TP + FN, indicates the number of relevant sequences that exist. For AR evaluation, recall can be viewed as synonymous with sensitivity or TP rate.

Finally, f-measure (also referred to as f-score or $f1$ score) provides a way to combine precision and recall as a measure of the overall effectiveness of activity classification.f-measure is calculated as a ratio of the weighted importance of recall or precision. The weight coefficient in Equation 5.7, β, is typically set to 1. Table 5.4 summarizes these performance metrics for our example AR problem.

$$\text{f-measure} = \frac{(1+\beta)^2 \times \frac{\text{TP}}{\text{TP+FN}} \times \frac{\text{TP}}{\text{TP+FN}}}{\beta^2 \times \frac{\text{TP}}{\text{TP+FN}} + \frac{\text{TP}}{\text{TP+FN}}} = \frac{(1+\beta)^2 \times \text{Recall} \times \text{Precision}}{\beta^2 \times \text{Rcall} + \text{Precision}} \tag{5.18}$$

Kappa Statistic While the performance metrics we have described so far give an indication of the recognition algorithm's performance, the values may be difficult to interpret. If two competing algorithms are being considered they can be compared using these metrics. If one algorithm is being considered, it is useful to compare

TABLE 5.4 Traditional Classifier-Based Performance Evaluation for Decision Stump Algorithm Applied to Hand Washing (H) and Sweeping (S) Activities

Performance Metric and Value			
Accuracy	0.87	G-mean (H)	0.56
Error rate	0.13	G-mean (S)	0.56
Sensitivity (H)	0.60	Precision (S)	0.67
Sensitivity (S)	0.93	Precision (H)	0.91
FP Rate (H)	0.07	F-measure (S)	0.55
FP Rate (S)	0.40	F-measure (H)	0.55
AUC-ROC	0.83	AUC-PRC	0.88

it to a baseline. The Kappa statistic, or Cohen's kappa coefficient, provides a way to compare the algorithm to one that assigns activity labels based on chance. The Kappa statistic is traditionally used to assess inter-rater reliability, or the degree to which two raters agree on the class value for a data point. When evaluating an AR algorithm, the two raters represent the algorithm being evaluated and ground truth (the actual activity labels). The Kappa statistic can take on a value between -1 and 1, where a value of 1 indicates perfect agreement, a value of 0 indicates the agreement is equal to chance, and a value of -1 indicates perfect disagreement. $P(A)$ is the probability of agreement between the AR and ground truth (the accuracy of the algorithm) and $P(E)$ is the probability that the accuracy is due to chance.

$$\kappa = \frac{P(A) - P(E)}{P(E)} \tag{5.19}$$

$P(E)$ can be estimated using a number of methods. Here we estimate it by computing the distribution of class labels assigned by the algorithm as well as the actual class label distribution.

Example 5.9 Thus $P(E) = P(H|Alg) \times P(H) + P(S|Alg) \times P(S) = 8/54 \times 10/54 + 46/54 \times 44/54 = 0.72$ for our example scenario. The κ value can then be computed as $(0.87 - 0.72)/(1.00 - 0.72) = 0.54$.

In addition to using the Kappa statistic to evaluate the performance of a classifier, it is also a useful statistic for evaluating the reliability of labeled activity data. If multiple individuals examine sensor data in order to annotate the data with ground truth labels, the consistency of the labels can be determined between the annotators by calculating $P(A)$, the proportion of data points where the annotators agree on the label, and $P(E)$, the proportion of data points where the annotators would agree if labels were randomly assigned.

Receiver Operating Characteristic Curve (ROC) *Receiver operating characteristic curve* (ROC)-based assessment facilitates explicit analysis of the trade-off between true positive and false positive rates. This is carried out by plotting a two-dimensional graph with the false positive rate on the x-axis and the true positive rate on the y-axis. An AR algorithm produces a (TP_Rate, FP_Rate) pair that corresponds to a single point in the ROC space, as shown in Figure 5.6. One recognition algorithm can generally be considered as superior to another if its point is closer to the (0,1) coordinate (the upper left corner) than the other. If the algorithm generates a probability or confidence value and uses a threshold to decide whether the data sample belongs to the activity class, the threshold value can be varied to generate a set of points in the ROC space. This set of points generates an ROC curve, as shown by the solid-line curve in Figure 5.10.

To assess the overall performance of an AR algorithm, we can look at the *Area Under the ROC curve*, or AUC. In general, we want the false positive rate to be low and the true positive rate to be high. This means that the closer to 1 the AUC value is,

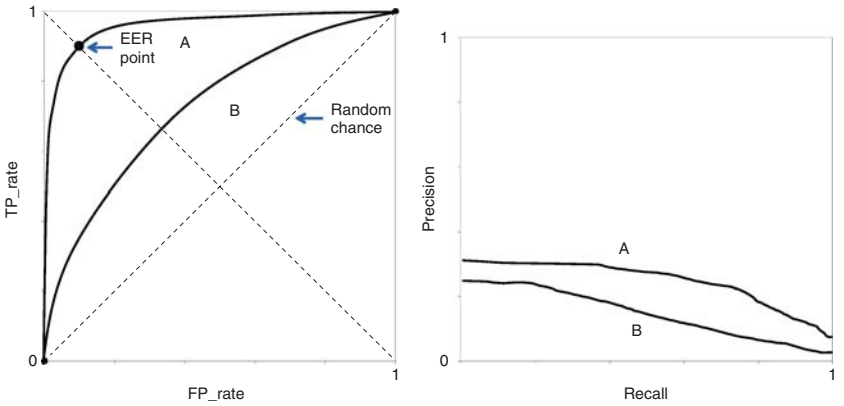


FIGURE 5.10 *Example ROC curve (left) and PRC (right).*

the stronger is the recognition algorithm. In Figure 5.10, the algorithm that generated curve A would be considered stronger than the algorithm that generated curve B. Another useful measure that can be derived from the ROC curve is the*Equal Error Rate* (EER), which is the point where the false positive rate and the false negative rate are equal. This point, which is illustrated in the ROC curve in Figure 5.6, is kept small by a strong recognition algorithm.

Precision-Recall Curve (PRC) A precision-recall curve (PRC) can also be generated and used to compare alternative AR algorithms. The PR curve plots precision rate as a function of recall rate. While optimal algorithm performance for an ROC curve is indicated by points in the upper left of the space, optimal performance in the PR space is near the upper right. As with the ROC, the area under a PRC can be computed to compare two algorithms and attempt to optimize AR performance. Note that the PR curves in Figure 5.10 (right) correspond to the same algorithms that generated the ROC curves in Figure 5.10 (left). In both cases, Algorithm A yields a larger area under the curve and could be interpreted as outperforming Algorithm B. It should be noted that an algorithm that performs best using the ROC metric may not perform best using the PRC and vice versa. However, the metrics can be considered separately to better understand the nature of algorithm performance. The PRC in particular provides insightful analysis when the class distribution is highly skewed, which often happens when modeling and recognizing activities.

5.4.2 Event-Based Activity Recognition Performance Metrics

AR can be viewed as a classification task. However, there are aspects of AR that are not like many other classification problems. First, the sequential nature of the sensor events means that the data points are not independent. In fact, they are related in time, in space, and in function. Second, the distinction between the class labels is not always clear. Some activities are very similar in function and purpose and thus

the sensor events are also similar. Finally, if activities are not pre-segmented, then any given window of sensor events could represent a transition between activities rather than just one activity. As a result, additional performance metrics are needed to better understand possible mis-matches between the labels that are generated by an AR algorithm and the ground truth activity labels for a given data point.

Using event-based evaluation relies on considering entire sensor event segments at a time. In contrast to the pre-segmented data, a segment here is a subsequence within the input data where the AR label and the ground truth label remain constant. Figure 5.11 shows an artificial sequence of sensor events that are labeled with the Hand Washing and Sweeping activities. Segment boundaries are indicated with vertical lines. With this approach to performance evaluation, each segment is analyzed to determine if it is correctly classified (the AR label matches the ground truth) or not. By considering one activity at a time, this can be viewed as a binary classification problem. For example, in Figure 5.9 the activity being considered, or positive class, is Hand Washing. A TP occurs when both rows show Hand Washing, a TN occurs when both rows show Sweeping, and the other situations indicate an FP or FN. The false positive and false negative errors are further divided into the subcategories described here to better understand the type of error that is occurring.

Overfill A false positive that occurs at the start or end of a segment that is partially matched is considered an overfill. This could occur at a segment boundary, for example, when the recognition algorithm is viewing a transition and has not received enough information to successfully detect the new activity. This is a possible explanation for the first overfill in Figure 5.11. The second overfill is found at the end of an activity and may occur when the AR anticipates a new activity (perhaps based on the time of day) before it actually occurs.

Insertion A false positive that represents an inserted positive label between two negative segments is considered an insertion. As seen in the example, the false positive segment due to insertion is not an extension of a true positive that is simply too long (as with overfill), but is an actual incorrectly-inserted positive label.

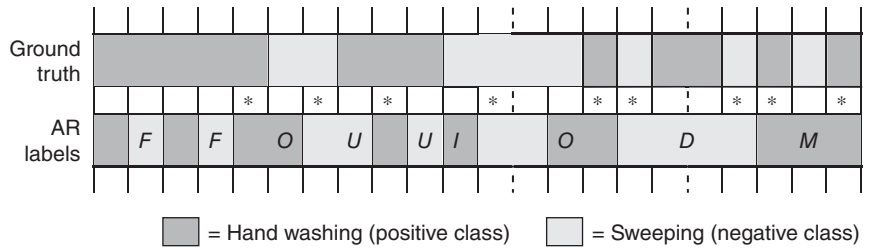


FIGURE 5.11 *Event-based evaluation of Hand Washing AR. The vertical solid lines indicate the performance segment boundaries. The top row shows the ground truth labeling of hand washing or sweeping and the bottom row shows the labels generated by an AR algorithm. Each segment mismatch is labeled with the type of error that is represented, either an overfill (O), underfill (U), insertion (I), deletion (D), merge (M), or fragmenting (F).*

Merge A merge is a false positive portion of a segment. The surrounding sensor events are correctly labeled with the positive class but the change in the middle of the segment to a different activity was not sensed by the AR algorithm.

Underfill Each of the previous three error classes has a corresponding type that can be used for analyzing false negatives. In the same way that an overfill error corresponds to a positive segment that extends too far, an underfill error occurs when a positive segment is too short because the beginning or ending of the segment is incorrectly labeled as the negative class. In our example, the fourth occurrence of a Hand Washing-labeled segment is too short, which results in an underfill error on either side of the segment.

Fragmenting A fragmenting false negative occurs between two true positives in a single consecutive positive segment.

Deletion The deletion error occurs when a false negative occurs between two true negatives. In our example, the AR algorithm combines sensor events into a single negative example because it failed to detect a switch to the positive class in the middle of the subsequence.

Once the correctly-labeled and incorrectly-labeled segments are identified and characterized, they can be used to generate rates or percentages for correct labels (C) and for each type of error for the sensor event sequence.

Example 5.10 Our example in Figure 5.9 has 22 sensor events total, so the O, U, F, and D rates are $2/22 = 0.090$ while the I and M rates are $1/22 = 0.045$. The error rates are fairly constant for each type of error in this artificial example. In realistic cases, they can provide insights on whether the errors are due to slow transitions to new activities, failure to detect short, quick activities, or other types of problems with AR.

Timeliness A related sequence-based performance measure is the timeliness of AR. Timeliness applies to a particular occurrence of an activity and is measured as the relative point in time for the activity at which the AR algorithm correctly provides the correct label and does not switch to an inaccurate label for the remainder of the activity.

Example 5.11 In Figure 5.11, there are 11 distinct activities that are observed and the point in time at which a correct label is generated and not changed is indicated by a star ($*$). As can be seen in this figure, the first activity is not correctly predicted and held stable unless the fifth event of the sequence, so the timeliness is 0.2. In contrast, the last activity is correctly labeled from its onset, so the timeliness is 1.0. An AR algorithm may need sufficient context to recognize the current activity, and the corresponding delay in outputting a correct label is captured by this performance metric.

5.4.3 Experimental Frameworks for Evaluating Activity Recognition

The next question is how to choose data on which the supervised learning algorithm will be tested. Because we want the learned model to generalize beyond the data it has already seen and correctly classify new data points that it has not previously seen, the model is usually trained on one set of data and tested on a separate, "holdout" set of data. In the context of AR, the question is how to select subsets of available data for training and testing. Here we describe two techniques that are commonly applied to this process.

The first method is k-fold cross validation. This method is effective when the amount of labeled data is limited because it allows all of the data points to play a role both in training and testing the learned model. With this approach, the set of data points is split into k nonoverlapping subsets. The model is trained and tested k times and the performance is averaged over the k iterations. On each iteration, one of the k partitions is held out for testing and the other $k - 1$ partitions are used to train the model. The choice of k varies, common choices are three-fold (this is particularly common when there are few data points available because each partition should ideally have at least 30 data points) and 10-fold cross validation. The results reported in the confusion matrices of Table 5.2 and 5.3 and the performance evaluation of Table 5.4 are based on a three-fold cross validation of the decision stump algorithm for the Hand Washing and Sweeping activities.

While cross validation is a popular validation technique for machine learning algorithms, its use is trickier when applied to sequential data. This is because the long contiguous sequence must be separated into individual data points and subsets of data points. As a result, some of the context is lost that may be captured by some algorithms when training the model. When activities are pre-segmented, the segments can represent individual data points. When a sliding window approach is used, individual windows represent the data points. Some alternative selections are to separate the datasets by time boundaries such as hours, days, or weeks in order to retain much of the sequential relationships. This method also allows the AR algorithm to be tested for its ability to generalize to new days, weeks, or months. When the recognition algorithm is trained to generalize over multiple users or physical settings, the individual users or environments can be treated as separate data points for testing. Finally, leave-one-out testing can be used to train the data on contiguous sequential data and test it on held-out data from the end of the sequence. The length of the training sequence can iteratively increase so that eventually all of the available data is used for both training and testing.

While most methods choose a holdout set for testing either through random selection or at the end of a sequence, holdout selection can also be performed strategically to demonstrate the generalizability of the AR algorithm over selected dimensions. For example, an algorithm can be trained over multiple individual people or homes, selecting one person or home as the holdout set. Similarly, an entire activity can be held out to determine how the algorithm performs on a previously-unseen activity class.

Additionally, a need may arise to compare the performance of two alternative AR algorithms. If minimizing error rate (or conversely, maximizing accuracy) of activity

classification is considered the performance metric of interest, then two alternative approaches f^1 and f^2 can be compared by, for example, looking at the mean error found through cross validation. However, because cross validation testing is performed only on a subset of the possible data points in the space, this comparison may not be a strong indicator of how the two approaches will compare in general.

Statistical tests such as the Student's *t*-test can be used to determine whether a set of performance differences between f^1 and f^2, such as generated through k-fold cross validation, is significant. To perform this computation from cross-validation results let $f_1^1, f_2^1, \ldots, f_k^1$ represent the set of results for $f^1, f_1^2, f_2^2, \ldots, f_k^2$ represent the set of values for f^2, and $d_1, d_2, \ldots, d_k$ represent the differences between the results (i.e., $d_i = f_i^2 - f_i^2$). We are then trying to determine if the mean $\overline{f^1}$ is significantly different from the mean $\overline{f^2}$. This calculation assumes that the values for f^1, f^2, and d follows a Student's distribution, which approximates a normal distribution when k becomes large. We first reduce the difference to a zero-mean, unit-variable variable t as a function of the difference mean, number of folds, and difference variance σ_d^2 as shown in Equation 5.20.

$$t = \frac{\overline{d}}{\sqrt{\sigma_d^2/k}} \tag{5.20}$$

If $t < -z$ or $t > z$, where z represents a confidence limit, then the difference in performance between the two approaches can be termed significant at the corresponding confidence level. The values of z are determined by the Student's distribution and the number of folds k (or, corresponding, the degrees of freedom $k - 1$). For example, if $k = 10$ and we want to reject the hypothesis that there is no difference in performance between the approaches with probability $p < 0.05$ (a common threshold used to report statistical significance), then $z = 2.262$.

5.5 ADDITIONAL READING

Ali and Aggarwal[71] learn activity breakpoints from video data using a supervised learning technique that is provided with activity begin and end frames for each activity class. Ho and Intille[72] and Feuz et al.[73] use supervised learning to recognize transitions between specific activity pairs. Iqbal and Bailey[74] use a multilayer perceptron to recognize and categorize breakpoints between any type of computer-related task. Niu and Abdel-Mottaleb[75] describe a method of activity segmentation that relies on rejecting sequences not clearly belonging to any one activity. Somewhat related is the idea of co-segmentation of event sequences introduced by Duchenne et al.[76] Co-segmentation is applied to two sequences that are known to contain the same activity of interest. The fact that they share the activity facilitates the process of identifying the activity boundaries within both sequences. The idea of employing an ensemble of classifiers to recognize activities from different window sizes is introduced by Zheng et al.[77]. Varying the size of a sliding window based on activity likelihood and relevant sensors has been explored by Krishnan and Cook[78] and by Okeyo et al[79]. A thorough

treatment of one-class classifiers and their uses for outlier and anomaly detection is provided by Khan and Madden[80].

Hong and Nugent[81] introduce an activity segmentation method that learns the relationship between activities, sensors, and locations, and then uses changes in these parameters to identify activity boundaries. In addition, Gu et al.[82,83] propose the idea of identifying activity boundaries based on the difference between accumulated sensor relevance. In their case, the sensors were object sensors and weights were calculated based on the discriminatory relevance of each object to each activity. Yamasaki[84] takes a similar approach to identifying activity boundaries based on differences in activity signatures from video data. This approach pinpoints activity boundaries based on maximizing the difference between feature vectors describing successive frames in the video. Keogh et al.[85] propose the idea of combining sliding windows with bottom-up segmentation in their SWAB algorithm to provide real-time segmentation with improved performance over a pure sliding window approach.

The Bayesian online change point detection is based on work by Adams and MacKay[86]. An alternative approach to online change point detection is to directly estimate the ratio of the probability density functions before and after candidate change points. Direct density ratio estimation is useful when the individual probability distributions are unknown[87]. Techniques to perform this include kernel mean matching, logistic regression, Kullback-Leibler importance estimation, unconstrained least-squares importance fitting, and relative unconstrained least-squares importance fitting (RuLSIF).[88] Other change point detection variations were introduced by Guenterberg et al.[89], who identify an activity boundary when the differences in spectral energy (defined in Chapter 3) before and after the boundary point are greater than a threshold value. Feuz et al.[90] use change point detection to identify activity boundaries as a time to provide prompt-based interventions. Xie et al.[91] introduce a new variation on change point detection that scales well to high-dimensional data. Avci and Passerini[92] segment data based on naturally-occurring gaps in the sensor event patterns and then use an estimate of the expected number of types of gaps in the data to improve AR.

A number of survey articles exist which provide an excellent overview of AR algorithms for different classes of sensors and activities. Many of the articles detail the algorithms and summarize achieved recognition performance on simulated and real-world datasets[30,93–101]. AR using NBCs has been explored by Cook[102] using smart home sensor data and by Bao et al.[35] and Ravi et al.[103] for accelerometer data. Popular methods for AR also include hidden Markov models[104–107] and conditional random fields[41,102,108]. Other have found SVMs[109] and decision trees[35] to be effective. Many methods have been explored which combine these underlying learning algorithms, including boosting and other ensemble methods[103,110,111].

While AR focuses on labeling activities, other work focuses on evaluating the quality of the activity that was performed[112–114]. Ogris et al.[115] and Amft[116] describe the problem of activity spotting and propose alternative approaches using data from wearable sensors.

Ward et al.[117] and Bulling et al.[95] introduce a number of performance metrics that fall into the categories of time-based and event-based evaluation of activity recognition algorithms. Reiss et al.[96] discuss the effectiveness of hold-one-activity-out evaluation of AR algorithms for generalizability assessment. The notion of the timeliness of AR was introduced by Ross and Kelleher[118].

Di Eugenio and Class[119] provide a good introduction to the kappa statistic and alternative ways to calculate $P(E)$ in calculating the kappa statistic. Davis and Goadrich[120] provide a useful discussion of the relationship between ROC and PRC curves and how to compute the area under the curves. Additional methods for evaluating and comparing the performance of supervised learning algorithms are presented by Cohen[121] and by Dietterich[122]. Nuzzo[123] offers an interesting discussion of the limits of p values for determining statistical significance and the need to also consider effect sizes (in the discussion of this chapter, the effect size would be the amount of difference in performance between alternative AR algorithms).

6

Activity Discovery

The most common approach to reasoning about human activities from sensor data is to identify activities that are important or interesting to track, then model and recognize occurrences of those activities. However, modeling all of human behavior in a recognition-based approach faces a number of challenges and limitations. First, in order to model and recognize activities a large amount of sensor data must be available that is pre-labeled with the actual activities they represent (the "ground truth" labels). In most real-world everyday settings, such pre-labeled data are not readily available.

Second, the time that is spent on commonly-tracked activities is a fraction of an individual's total behavioral routine. The US Bureau of Labor Statistics alone reports 462 activities that people perform in their daily lives and this number is likely much larger worldwide when we consider the unique interests of individual people and the diversity of activities that accompanies different cultures. Activities are often tracked if they are needed for a particular automation task or if they are used to indicate the status of a condition such as functional health. For example, when the activities listed in the Lawton Instrumental Activities of Daily Living Scale[11] are tracked by time spent, the breakdown for Americans follows the chart shown in Figure 6.1. Note that the largest category in this chart is "Other," which represents the time spent in activities that do not fit into this scale. Even if all of the activities listed in Table 2.1 are tracked, they will still not account for time spent transitioning between activities and performing unique activities that are not found on this list. As Aristotle said, "We are what we repeatedly do."[124] Modeling and tracking only preselected activities therefore ignores the important insights that other activities can provide on routine behavior and activity context for individuals.

Third, the accuracy of activity recognition can be improved by combining it with activity discovery. If the activities shown in Figure 6.1 are modeled by an activity

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

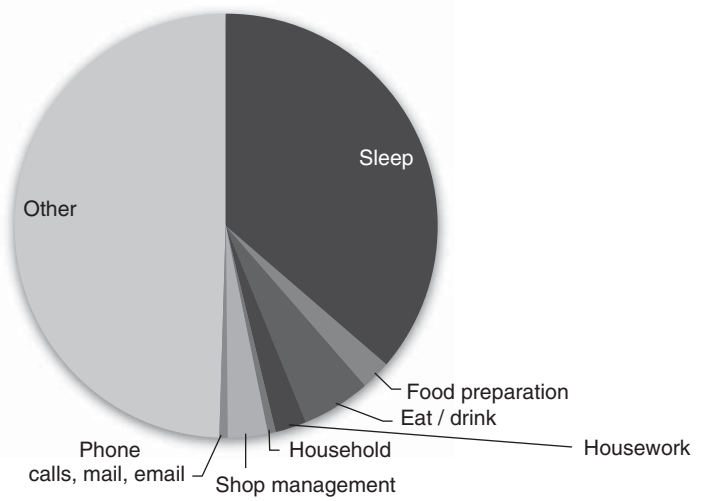


FIGURE 6.1 *Time spent by Americans as reported by the Bureau of Labor Statistics*[14].

recognition algorithm, the algorithm will be affected by the large Other class that skews the class distributions. The problem of modeling classes with a skewed class distribution has vexed researchers for over a decade. Techniques have been explored to deal with the problem including resampling data points to balance the distribution and cost-sensitive learning to apply greater weight to classes with a small number of examples. The situation described here is particularly problematic because the Other category represents a potentially boundless combination of diverse activities and is therefore difficult to model accurately. Activity recognition can be paired with activity discovery in these types of situations. Activities can be discovered from within the Other class that serve to break the class into smaller pieces and track the discovered patterns together with predefined activity classes.

In this chapter, we will consider several different approaches to discovering activities from sensor data. The first method, zero-shot learning, discovers new activity classes to learn whose labels do not appear in training data. The remaining methods, including frequency and compression-based sequence mining, clustering, and topic models, employ unsupervised methods to discover potential activity classes from unlabeled sample sensor data.

6.1 ZERO-SHOT LEARNING

The first type of activity discovery method we will discuss actually builds upon an existing activity recognition algorithm. In zero-shot learning, the goal is to enhance an existing classifier $h : X \rightarrow A$ that maps sensor features onto activity labels so that it can recognize an expanded set of activity labels A', where $A \subset A'$. Thus, this method can be used to map features onto activity labels for activity classes that do not have any available training data.

Zero-shot learning relies on the assumption that high-level features, F, can be used to describe activities and that a value is available for each of the features, for each of the activities in A'. Examples of such high-level features for activity modeling may include the following:

- Takes place in the kitchen
- Utilizes tools from cabinet x
- Occurs between 8:00 am and 10:00 am

The zero shot learner thus maps the original, low-level sensor features onto these high-level features, $h_1{:}X \rightarrow F$ and maps high-level features onto activity labels, $h_2{:}F \rightarrow A$. The first mapping, h_1, can be learned by a classifier given training examples. The second mapping, h_2, needs to be provided by an expert or external information source. Figure 6.2 summarizes the attribute values we used for the example activities of Hand Washing and Sweeping as well as two other activities for which training data are not given, Cooking and Sleeping.

To learn a mapping from raw sensor features to high-level features, a separate classifier is learned for each feature. The input to the classifier is all of the available training data. Data points corresponding to activities which are associated with a given feature $f \in F$ represent instances of the positive class for the feature and all of the remaining data points represent instances of the negative class.

In some cases, particularly when $|A'| >> |A|$, the training data may contain either all positive or all negative examples for a particular feature. In Figure 6.2, for example, all of the activity classes for which training data are available (Hand Washing and Sweeping) utilize the *Kitchen* sensor feature. On the other hand, neither of these activities utilize the *Bedroom* sensor feature. In this situation, a one-class classifier can be trained to recognize the positive (or negative) examples from the training data and to reject all other data points (which thus belong to the opposite class).

Once h_1 and h_2 are both available, they can be used to recognize any of the activities in A'. First, if a trained activity classifier cannot recognize a data point with sufficient confidence, then a vector of high-level feature values can be generated using each of the learned h_1 mappings. A classifier such as the nearest neighbor algorithm can then be used to determine which activity is the best fit for the feature vector given the activity-feature matrix as shown in Figure 6.2. Note that this method can even be used to discriminate between two activities where no training data are available for either activity. In the case of our example, the presence of water and burner sensors

	Kitchen	Bedroom	Water	Burner	Soap	Broom
Hand washing	1	0	1	0	1	0
Sweeping	1	0	0	0	0	1
Cooking	1	0	1	1	0	0
Sleeping	0	1	0	0	0	0

FIGURE 6.2 *Example activity-attribute matrix. Each row represents a distinct activity and each column represents a separate high-level feature. The contents of the matrix indicate whether the feature is associated with the activity* ($value = 1$) or not ($value = 0$).

would indicate that the activity is more likely to be Cooking than Sleeping, whereas if neither the water nor the burner is used but the bedroom sensor fires then it is more likely to be the Sleeping activity.

Zero-shot learning offers a mechanism for learning a large set of activity classes even from a fairly constrained set of training examples. The method does face difficulties, however, if multiple discriminatory features do not appear in the training examples, because learning classifiers for these features will be challenging. In such situations, even a small amount of training data for the unknown classes will be needed in order to discriminate between them.

6.2 SEQUENCE MINING

Because activities are being discovered from sequential sensor event data, an obvious initial approach to discovering activity patterns is to mine sample sensor event data for common, or frequent, recurring sequence patterns. Sequential pattern mining is commonly used to identify common progressions of purchasing patterns and searches for recurring patterns in ordered sequences of symbols.

To apply a sequence mining approach to activity discovery, we define a vocabulary of symbols that are used to describe each of the possible sensor events. Sensors that generate discrete-valued messages (such as the "ON" and "OFF" messages generated by motion sensors in our sample data) can easily be represented as symbols by concatenating the sensor identifier that generated the event together with the event message.

Example 6.1 To see how this works, consider that the first sensor event in Appendix 1:

13:53:08.061116	M017	ON

can be represented as the symbol *M017ON*. In the case of sensor events that generate numeric values, the numeric values can be partitioned into disjoint intervals or bins. The sensor name and the appropriate interval can then be merged into a symbol that represents the numeric sensor event. In the case of our sensor event data sample, the hip accelerometer generates continuous-valued messages. The string *AccelHip((500.0, 125.5],(2200.0,2300.0],(1805.0,1852.0])* is one such symbol that can be used to represent a range of values for the accelerometer placed on the hip.

A sequence mining algorithm performs a search through the space of candidate sequences to identify interesting patterns. A pattern here consists of a sequence definition and all of its occurrences in the data. Each candidate sequence pattern is evaluated according to a predefined criterion and the top candidates are reported as the discovered activities. Alternative search algorithms can be employed but the most common is a breadth-first search. In this approach, all unique symbols are sorted by the evaluation measure and stored in an open list. These initial symbols represent candidate

sequence patterns of length 1. Each of the candidates (or a selected number if the open list grows too large to store and process) is expanded to create new candidate patterns. Candidate expansion thus assumes that pointers that indicate where the pattern instances occur in the original data are maintained. New candidate patterns are formed by merging an occurrence of the pattern in the original data with the symbol that occurs before or after the pattern in the data, effectively creating a pattern with length one greater than the previous (parent) pattern. In this way, the algorithm searches through increasingly-long sets of candidate patterns until the set of unique pattern candidates is exhausted or another stopping criteria is met.

There are a few challenges that are faced when sequence mining is used for activity discovery. The first is the computational expense of the algorithm. The space of possible sequence patterns is exponential in the length of the input data. As a result, bounds need to be placed on the search algorithm to constrain its run time. One bound that can be used is to place a threshold on the total number of candidates that are evaluated. Using this bound, once the limit is reached no new candidates are generated and the best results found so far are reported.

A second bound can be enforced by using a beam search, a type of greedy approach in which a bound is placed on the size of the open list. When a candidate pattern is removed from the open list and expanded, its children are placed on the open list only if there is room. If there is not enough room, the children (and other candidates) with the lowest frequency are removed so that the open list size (the size of the beam) does not exceed the predefined size.

The second challenge is determining the best evaluation measure for candidate patterns. Here we describe two alternative criteria: frequency and compression.

6.2.1 Frequency-Based Sequence Mining

One criterion that is commonly used in sequence pattern mining is frequency, or the number of times the sequence pattern appears in the sample data. Using this evaluation criterion, all unique symbols that occur in the data are sorted by frequency and stored in an open list. When each of the candidates is expanded to create child candidate patterns, those children are evaluated. Constraints can be imposed on the type of patterns that are discovered, such as the monotone and anti-monotone properties, that reduce the amount of computation that is needed. The generated patterns with the greatest overall frequency are reported. This algorithm is summarized in Figure 6.3.

Example 6.2 Consider the sample sequence that is formed by joining the two sensor event subsequences shown in Figures 5.4 and 5.6. To simplify the sequence processing, we denote each unique type of sensor event by a two-letter symbol indicating the location in the home where the event took place (`LR` = LivingRoom, `BR` = Bedroom, `OR` = Otherroom, `KI` = Kitchen, `FM` = FrontMotion, `FD` = FrontDoor, and `Ba` = Bathroom). The resulting sequence is thus

```
LR BR OR LR LR LR OR K OR K FM FM FD FD Ba Ba Ba LR OR LR K LR FM
K FM FM FM FM FM FD
```

```
Algorithm FrequentSequenceMining(Sequence, BeamWidth, MaxBest, Limit)
  // BeamWidth is threshold on size of ChildList
  // MaxBest is threshold on number of best patterns that will be reported
  // Limit is threshold on the number of candidate patterns that are evaluated
  ParentList = {}
  ChildList = {}
  BestList = {}
  ProcessedSeqs = 0

  // Create an activity sequence from each unique sensor event and its instances;
  // insert the resulting sequence patterns in ParentList
  while ProcessedSeqs <= Limit and ParentList is not empty
    Parent = RemoveHead(ParentList)
    Extend each instance of parent in both directions
    Group the extended instances into Child activity patterns
    for each Child
      if Frequency(Sequence, Child) >=  Frequency(Sequence, Parent)
        if NotDuplicate(Child, ChildList)
          Insert Child in ChildList in sorted order by Freqency(Sequence, Child)
        end if

        if Length(ChildList) > BeamWidth
          Remove the activity pattern at the end of ChildList
        end if
      end if
    done

    Insert Parent in BestList in order by frequency
    ProcessedSubs = ProcessedSubs + 1
    if Length(BestList) > MaxBest
      Remove the activity pattern at the end of BestList
    end if

    Switch ParentList and ChildList
  done

return BestList
```

FIGURE 6.3 *Frequency-based sequence mining algorithm.*

Figure 6.4 shows the candidate patterns that are generated at each step of the algorithm. After each step, we only keep the pattern with the highest frequency and discard the rest. The algorithm halts when the highest-frequency patterns only have one occurrence. The best patterns in each step are highlighted in bold. This example highlights the fact that many of the discovered patterns reflect time spent in one location (in this case, the region near the front door). The discovered activity here is consistent with a pre-labeled activity, namely Leave home.

6.2.2 Compression-Based Sequence Mining

While many sequence mining algorithms rely on frequency for the evaluation measure, the measure does not always yield the best activity patterns. This is primarily because shorter patterns will necessarily yield greater frequency in the data. In order

Step					
Patterns	1	2	3	4	5
	FM (8)	**FM FM (5)**	**FM FM FM (3)**	**FM FM FM FM (2)**	**FM FM FM FM FM (1)**
	LR (7)	K FM (2)	K FM FM (2)	K FM FM FM (1)	K FM FM FM FM (1)
	OR (4)	FM FD (2)	FM FM FD (1)	FM FM FM FD (1)	FM FM FM FM FD (1)
	Ki (4)				
	FD (3)				
	BR (1)				

FIGURE 6.4 *Highest-value patterns found by frequency mining.*

to discover sufficiently abstract and complex activities, the sequence pattern must be longer than just a few sensor events. An alternative approach then is to search for a sequence pattern that best compresses the input dataset, using compression-based sequence mining.

Compression-based discovery is based on the minimum description length (MDL) principle, which states that the best theory to describe a set of data is the theory that minimizes the description length of the entire dataset. Description length calculation is based on the model of a local computer, the encoder, sending a description of a concept to a remote computer, the decoder. The local computer must encode the concept (in this case, a sequence of sensor events) as a string of bits that can be sent to the remote computer, which decodes the bit string to restore the original concept. The concept's description length is the number of bits in the string. An evaluation heuristic based on the MDL principle thus assumes that the best pattern is one that minimizes the description length of the original dataset when it is compressed using the pattern definition.

To compute the MDL-based value of a candidate pattern, each occurrence of the pattern can be replaced by a single event labeled with the pattern identifier. As a result, the description length of a pattern P given the input data D is calculated as $DL\,(P) + DL\,(D|P)$, where $DL\,(P)$ is the description length of the pattern definition and $DL\,(D|P)$ is the description length of the dataset compressed using the pattern definition. Description length is calculated in general as the number of bits required to minimally encode the dataset. We estimate description length as the number of sensor events that comprise the dataset. As a result, the mining algorithm seeks a pattern P that maximally compresses the data or maximizes the value of

$$\text{Compression} = \frac{DL\,(D)}{DL\,(P) + DL\,(D|P)} \tag{6.1}$$

As before, the initial state of the search algorithm is the set of pattern candidates consisting of all uniquely labeled sensor features. The entire dataset is scanned to create initial patterns of length one. After this first iteration, the whole dataset

does not need to be scanned again. Instead, patterns discovered in the previous iteration are extended as before to include sensor events occurring immediately before or after each occurrence of the pattern in the sample data. A pruning heuristic can be employed to remove patterns from consideration if the newly extended child pattern yields a value that is less than the value of its parent pattern. However, this is just a heuristic since the Apriori property does not necessarily hold when using a compression evaluation measure.

Note that with compression-based mining, any knowledge that can be encoded into the pattern definition can be used to further compress the sequence data and thus improve the value of a discovered activity pattern. For example, the periodicity of a candidate pattern can be taken into account when computing compression. Periodicity is a common attribute in activities, given the emphasis individuals place on organizing their time. In the work environment, meetings are held weekly to provide updates on the status of a project. School is in session from 8:00 am to 3:00 pm Monday through Friday, and recreational sporting teams play each weekend.

Because candidate pattern descriptions are stored together with pointers to the occurrences of the pattern in the sample data, periodicity detection algorithms can be applied to the occurrence start times to determine the period length and predictability. If the periodicity is sufficiently predictable, the sequence pattern periodicity can be encoded into the pattern definition by adding bits representing the periodicity of the pattern (i.e., the amount of time that elapses between occurrences of the pattern). As a result, the pattern occurrences do not need to be replaced with a pattern identifier because their occurrences can be reconstructed from the pattern definition. While the pattern description will be larger, the sensor event sequence will be smaller after compression and thus periodic sequences will be preferred over those that occur in a more sporadic manner.

Another challenge to consider with sequence pattern mining for activity discovery is yielding the desired level of complexity in the activity descriptions. Depending on the purpose of activity learning, the desired activities may be simple movements with short length/duration or complex activities that can themselves be described in terms of these shorter components. Ideally, then, a hierarchy of activities would be discovered that decompose into other activity patterns and collectively represent a majority of the original data.

This goal can be achieved by applying compression-based activity discovery in an iterative manner. Once the search terminates and the sequence mining algorithm reports the best patterns that were found, the sensor event data can be compressed using the best pattern. The compression procedure replaces all instances of the pattern by single event descriptors, which represent the pattern definition. The discovery process can then be invoked again on the compressed data. This procedure can be repeated a user-specified number of times. Alternatively, the search and compression process can be set to repeat until no new patterns can be found that compress the data.

Because human behavioral patterns rarely occur exactly the same way twice, an edit distance measure can be employed to determine if a sensor sequence is an acceptable variation of a current pattern, and thus should be considered an occurrence of the

pattern. This allowance provides a mechanism for finding fewer patterns that abstract over slight variations in how activities are performed.

An edit distance measure is thus needed to determine the fit of a variation to a pattern definition. One such measure is the Damerau–Levenshtein measure. This measure counts the minimum number of operations needed to transform one sequence, x, to be equivalent to another, x'. In the case of the Damerau–Levenshtein distance, the allowable transformation operators include changing one symbol name to another (recall that for sensor data, each symbol represents a sensor event), addition/deletion of a symbol, and transposition of two symbols. A sensor event sequence can be considered sufficiently similar to another if the edit distance is less than a threshold value times the size of the longer sequence. The edit distance is computed in time $O(|x| \times |x'|)$.

As an example, Figure 6.5 depicts a dataset where the unique sensor events are represented by varying circle colors. The algorithm discovers four instances of the pattern P in the data that are sufficiently similar to the pattern definition. The resulting compressed dataset is shown as well as the pattern P' that is found in the new compressed dataset. The hierarchical compression-based discovery algorithm is summarized in Figure 6.6.

Example 6.3 Using our previous example, we can calculate the description length of the original sequence as 30 bits (assuming an equal-length encoding of 1 bit for each sensor event). Note that because patterns of length 1 do not compress the input sequence, patterns are ordered by frequency in the first step. The top patterns after step 1 are shown at the top of Figure 6.5. In step 2, the sequence `FM FM` is generated, which appears five times in the input data (although only three nonoverlapping occurrences exist). If the data were compressed using this sequence each nonoverlapping occurrence of `FM FM` would be replaced by a single symbol. However, two extra bits would be needed to describe the length-2 pattern. This effectively reduces the description length from 30 bits to $(30 - (3 * (2 - 1)) + 2) = 29$ bits, for a

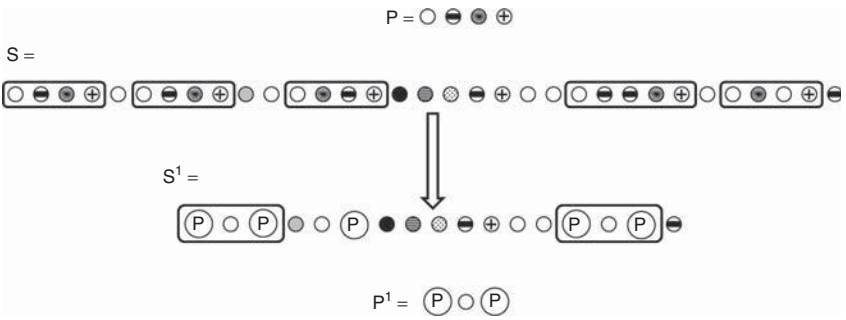


FIGURE 6.5 *Example of the hierarchical discovery algorithm applied to an input sequence S. Each unique sensor event symbol is represented by a unique circle pattern in the diagram. A sequence pattern (P) is identified and used to compress the dataset into a smaller sequence* S^1. *A new best pattern* (P^1) *is found in the second iteration of the hierarchical discovery algorithm.*

```
Algorithm CompressionSequenceMining(Sequence, BeamWidth, MaxBest, Limit,
                                    MaxIterations)
  // BeamWidth is a threshold on size of ChildList
  // MaxBest is a threshold on number of best patterns that will be reported
  // Limit is a threshold on the number of candidate patterns that are evaluated
  // MaxIterations is a threshold on number of discover-and-compress cycles to perform
  Iterations = 0
  // Identify best pattern and compress using pattern until exceed limit or
  // no further compression is possible
  while Iterations <= MaxIterations and BestCompression > 0
    ParentList = {}
    ChildList = {}
    BestList = {}
    ProcessedSeqs = 0
    // Create an activity sequence from each unique sensor event and its instances;
    // insert the resulting sequence patterns in ParentList
    while ProcessedSeqs <= Limit and ParentList is not empty
      Parent = RemoveHead(ParentList)
      Extend each instance of parent in both directions
      Group the extended instances into Child activity patterns
      for each Child
        if Iterations = 1 then
          Value(Child) = Frequency(Sequence, Child)
        else Value(Child) = Compression(Sequence, Child)
        end if
        if NotDuplicate(Child, ChildList)
          Insert Child in ChildList in sorted order by Value(Child)
        end if
        if Length(ChildList) > BeamWidth then
          Remove the activity pattern at the end of ChildList
        end if
        ProcessedSubs = ProcessedSubs +1
        Insert Parent in BestList in order by Value(Parent)
        if Length(BestList) > MaxBest
          Remove the activity pattern at the end of BestList
        end if
      done
      Switch ParentList and ChildList
    done
    // Replace each occurence of the best pattern found this iteration with a single node
    Sequence = Compress(Sequence, Head(BestList))
    Iterations = Iterations + 1
  done
return BestList
```

FIGURE 6.6 *Hierarchical compression-based sequence mining algorithm.*

compression value of 1.03. This is the only step-2 pattern that results in a positive compression, so this is the only pattern that is expanded. There are two step-3 patterns that appear more than once: the patterns `K FM FM` and `FM FM FD` appear two times each. Each of these appears two times, so they are valued at a compression of $(29 - (2 * (3 - 1)) + 3) = 28$ bits, for a compression value of 1.04 bits. None of the expansions of these patterns appears more than once and thus will not result in a positive compression, so the algorithm finishes.

Unlike the frequency-mining algorithm, the compression-based mining approach can generate a global sorted list of discovered patterns. For this example, the best overall pattern is `K FM FM`. We can then compress the input data using this pattern by replacing each occurrence of the pattern with a single symbol (in this case, the symbol P1). The compressed sequence is

```
LR BR OR LR LR LR OR K OR P1 FD FD Ba Ba Ba LR OR LR K LR FM P1
FM FM FM FD.
```

The entire discovery process can be repeated on this smaller sequence. No patterns exist in the new sequence that further compress the sequence. If such a pattern did exist, it would be added to the set of discovered activity patterns. Some of the discovered patterns may contain other pattern names in their description. The resulting output may therefore be a hierarchy of activity patterns, described at different levels of complexity.

Notice that the algorithm described here produces a lossy type of compression because differences between occurrences of an activity pattern may be allowed. Lossless compression techniques could also be employed, although the description length of the sequence will increase in order to encode the differences between the pattern definition and each occurrence of the pattern in the sensor event sequence.

6.3 CLUSTERING

A common method of trying to find structure in unlabeled data is to group it into clusters. For activity discovery, these clusters may represent distinct activity classes. Ideally, then, the clusters reflect properties of the activities that cause sensor event sequences from the same activity to bear a stronger resemblance to each other than they do to the remaining sensor event sequences. Clustering sequence and time series data does introduce challenges that are not typically faced in other data analysis tasks, but there are algorithms that have shown to be effective under certain constraints.

In order to apply clustering techniques to sensor event sequences for the purpose of activity discovery, three main choices must be made. The first is selecting the data points to cluster, the second is choosing an effective clustering algorithm, and the third is identifying an appropriate similarity metric to use within the algorithm. Once the clusters are formed, the data points can be fed into a classifier in order to recognize future occurrences of the discovered activity in the same way that predefined

activities are learned and recognized. The cluster centroid, or average of the points in the cluster, can be used to describe the discovered activity.

As our goal is to find a one-to-one mapping between the final clusters and the activities that the data represents, each data point should ideally represent one occurrence of an activity. This means that any of the segmentation methods described in Chapter 5 can be applied to a dataset and the resulting segments, or features describing the segment sensor events, can be clustered to derive the corresponding activities. Similarly, a window with fixed length q can slide along the dataset, dividing the data into length-q segments. Clustering methods applied to these constant-length data sequences are known as *q-gram* clustering methods.

There are a number of well-tested clustering methods that can be applied to activity data. One of the most commonly-used techniques is the *k means* algorithm. This algorithm, like many clustering algorithms, assumes that the number of desired clusters is specified a priori. This number is k, which in our case represents the number of activities that are exhibited by the data. The algorithm initially chooses k random points from the data sets to form k separate clusters. For each remaining data point, the distance from the point to the k clusters is calculated and the point is assigned to the closest cluster.

On every subsequent iteration of the clustering algorithm, a new cluster center is computed by averaging the points in the cluster. The distance from each data point to each cluster is then reevaluated and the point is moved to a different cluster if the distance to the new cluster center is smaller than the distance to its previous cluster center. This process continues until a predefined convergence criterion is met.

Other clustering algorithms are available that do not make assumptions about the number of clusters, but do rely on other convergence criteria. For example, density-based methods identify clusters of arbitrary shape and quantity rely on a density threshold to determine the cluster boundaries. Hierarchical clustering methods do not rely on either assumption but provide cluster groupings at different levels of precision by either dividing the dataset into levels of ever-increasing cluster precision (top-down or divisive hierarchical clustering) or merging individual data points into larger, more complex groups (bottom-up or agglomerative clustering).

One component that all of these methods have in common is that they all rely upon a function that determines the similarity, or conversely the distance, between two data points. While Euclidean distance is sometimes used for its simplicity, it can often mislead the clustering algorithm because every attribute is given equal weight and distances may not be normalized. In addition, it does not necessarily take into account the sequential nature of the data. Here are alternative distance metrics that can be considered.

Kullback–Leibler Distance This formula can be used to compare two sensor event sequences that are represented as two probability distributions, $P1$ and $P2$. The calculation is asymmetric, because it represents the information that is potentially lost when $P2$ is used to approximate $P1$. The value can be used as a distance measure between two data points or a cluster and a data point and is calculated as shown in Equation 6.2. In this equation, s is a single event or feature in the data sequence, taken

from a vocabulary Θ. The probability distributions can take into account sequence ordering or can estimate feature values based on a bag of feature representation of the data.

$$\text{distance}_{KL}(P1||P2) = \sum_{s\in\Theta} \ln\left(\frac{P1(s)}{P2(s)}\right) P1(s) \tag{6.2}$$

Mutual Information In Chapter 5, we used the notion of mutual information to weight sensor events within a window based on their relevance to the current activity. In this case, the mutual information can be computed between two data points, or sensor event sequences, to determine their distance. MI($S1$, $S2$) can be calculated based on the marginal probability functions of $S1$ and $S2$ as well as the joint probability function, as shown in Equation 6.3.

$$MI(S1, S2) = \sum_{s1\in S1} \sum_{s2\in S2} P(s1, s2) \log\left(\frac{P(s1, s2)}{P(s1)P(s2)}\right) \tag{6.3}$$

Edit Distance The notion of edit distance was introduced in Section 6.3 as a way of allowing variation between an activity pattern definition and individual occurrences of the pattern. Edit distance, or the minimum number of operations needed to transform one sequence to be equivalent to another, can also provide a measure of distance between data points to use in clustering the data points. The Damerau–Levenshtein distance, described in the previous section, provides one specific formula for calculating edit distance.

Dynamic Time Warping Dynamic time warping is a measure that is commonly used when comparing two temporal sequences. Because sequences with similar content may occur with a variety of different timings or speeds, the times of the occurrences need to be aligned in order to determine if the time series are in fact occurrences of the same pattern. Dynamic time warping calculates an optimal match between two given sequences where the only allowable change is to "warp" the sequences nonlinearly in the time dimension. Sequence alignment is often used in practice to determine the sequence match. The match may allow for a combination of accelerations and decelerations in one sequence that facilitates its alignment with the second sequence. In addition to determining the ideal alignment, dynamic time warping also generates the cost, in number of warping operations, associated with aligning the temporal sequences.

6.4 TOPIC MODELS

The goal of discovering activity patterns from sample sensor event sequences bears similarity to other discovery tasks. In particular, text mining shares a similar goal of discovering the main themes that pervade a large corpus of documents. This goal led to the design of *topic models* and approaches to discover these themes from a collection of information. The idea behind topic models is to represent a document

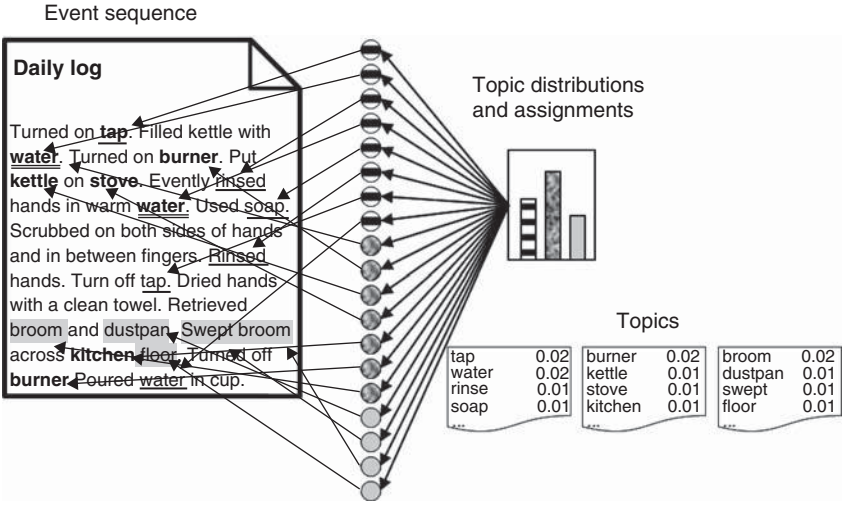


FIGURE 6.7 *Illustration of the topic model process. Each sensor event sequence is a mixture of topics (high-level features), each topic is a distribution over low-level sensor features, each sensor feature is drawn from one of the topics.*

as a mixture of topics, each of which has an associated probability distribution over related words. If a document is viewed as a "bag of words," then a new document can be created by choosing a topic mixture, then for each topic generating a set of words based on the associated multinomial probability distribution. Figure 6.7 shows an example of a document that can be organized into a set of topics, each with a probability distribution over words as defined by the relative frequency of the words found in treatments of the associated topics.

Applying this to activity learning, each activity sequence can be considered as a mixture of activities and each activity has an associated multinomial distribution over sensor events. As defined in Equation 6.4, we can then compute the probability of an event e occurring in a sequence s assuming there are a fixed number, A, of activities.

$$P(e|s) = \sum_{a=1}^{A} P(e|a)P(a|s) \tag{6.4}$$

Ultimately, we want to learn the activities, or topics, given a set of activity sequences (akin to documents in text mining) and the low-level sensor features that are generated by the activity sequences. The activity sensor event sequences are observed but the underlying activity structure, including the activities, the per-sequence activity distributions, and the per-sequence per-event activity assignments are *hidden*. Topic modeling uses the observed sequences to infer the hidden structure, in a sense turning the sequence generation process on its tail.

One way to learn the hidden structure is using a process such as Latent Dirichlet Allocation (LDA). Intuitively, we first fix the number, A, of activity categories to

learn and start by making random guesses of what the activities are and how they connect to events and sequences. The initial guesses are not likely to be correct so they are incrementally improved to make them more internally consistent. Each piece of observed evidence can be analyzed and events can be reassigned to topics guided by the formula in Equation 6.4. During this process, events will gradually become more common for activities in which they are already common. In addition, activities will become more common in sequences where they are already common. The model becomes more consistent as activities focus on specific sequences and events. LDA thus trades off two goals. First, allocate events to as few activities as possible. Second, for each activity, assign a high probability to as few events as possible. Trading off these goals results in groups of tightly co-occurring sensor events.

Formally, LDA places a Dirichlet prior with parameter α, or $P(\theta_s|\alpha)$, on the sequence-activity distributions, $P(a|\theta_s)$. Fitting the model consists of finding parameters α for the Dirichlet distribution and parameters β for the activity-event distributions, $P(e|a,\beta)$, that maximize the likelihood, L, of the events for sequences $s = 1, \ldots, S$, as shown in Equation 6.5. In this equation, each sequence s consists of the words e_n^s, for a total of N_s words. The inner sum is marginalized over activity a and the integral is marginalized over activity activations θ_s.

$$L(\alpha,\beta) = \prod_{s=1}^{S} \int P(\theta_s|\alpha) \left(\prod_{n=1}^{Ns} \sum_{a=1}^{A} P\left(e_n^s|a,\beta\right) P(a|\theta_s) \right) d\theta_s \tag{6.5}$$

Topic modeling provides an elegant way to learn activity classes directly from sample data. Unlike zero-shot learning, the set of known activities and their relationship to the sequences does not need to be provided a priori, although the number of activities does need to be provided. Unlike the sequence mining approaches, topic models do not take into account ordering information but view the activities as a bag of features. They do provide a fairly scalable approach to analyzing large datasets of sensor sequences to discover underlying structure and possible classes of activities.

6.5 MEASURING PERFORMANCE

Evaluating the performance of unsupervised discovery methods is not as straightforward as for activity recognition, because there is no single "correct" outcome against which the performance of the algorithm can be measured. However, there are several desirable characteristics of activity discovery and the performance of the algorithms can be evaluated along these dimensions. Here we highlight some alternative performance metrics and describe the algorithm characteristics that the metrics evaluate.

6.5.1 Expert Evaluation

A common method of determining the value of discovered activities is to ask experts who are familiar with the data to evaluate the activities. In most cases, the individual

performing the activities can play the role of the expert. Their feedback can indicate how reflective the discovered patterns are of their normal behavioral routine.

Predefined Activity Alignment The second approach to determine the value of discovered activities is to determine how well they align with predefined activities. In cases where ground truth activity labels are available, discovered patterns can be matched with known activities to determine how well they align. The alignment can be determined using any similarity measure such as inverse edit distance. The expert evaluation and activity alignment measures provide insight on how well the discovered activities reflect an intuitive description of the activities that occur. These measures are not ideal when performed in isolation, however, because the unsupervised discovery methods may actually discover meaningful patterns in the data that the expert does not detect, either because the expert is unaware of this structure in the data or the patterns are not captured by predefined activity labels. A similar approach is to generate synthetic sensor data in which known activities are embedded. Evaluating the performance of the discovery algorithm then amounts to ascertaining whether the discovery algorithm finds the known activities in the artificial data.

Predictive Stability A desired characteristic of supervised learning algorithms, such as those used for activity recognition, is that they correctly label not only the training data but also data that has not yet been observed. Similarly, a valuable characteristic of discovery algorithms is that they discover patterns that not only describe the available data but that reflect underlying patterns inherent in other similar, unseen data. This desired stability of the discovered activities can be evaluated using a leave-one-out analysis, where the held-out data may reflect sensor data for a time frame or for an individual that has not yet been observed. For example, a discovery algorithm, D, can be run on $n - 1$ days of data and the data for day n is held out. Once the patterns are discovered, recognition algorithms can be trained to recognize occurrences of the activity pattern. The expected recognition performance can be determined using cross validation on the training data. The stability of the patterns can then be determined by testing the learned models on data from data n. The difference between recognition performance on day-n data and cross validation performance can be reported as the stability of the discovered patterns and can be used to evaluate the strength of the activity discovery algorithm.

Compressive Stability The stability of discovered activity patterns can also be analyzed in a manner that is independent of activity recognition. For example, the sequence mining algorithms described in this chapter introduce pattern evaluation measures that are to evaluate candidate patterns, including frequency and compression. These measures can be applied to both training data and testing data, normalized for data size, and compared to determine the stability of discovered patterns. Another measure that is common in language model discovery that can be employed here is *perplexity*. Given a discovered activity pattern, d, perplexity can be used to determine how well it predicts test sensor data $x_1, \ldots, x_N$ that is drawn from

the same probability distribution as the training data. The perplexity of d is defined as shown in Equation 6.6.

$$\text{Perplexity}(d) = 2^{-\sum_{i=1}^{N} \frac{1}{N} \log_2 d(x_i)} \tag{6.6}$$

The exponent in Equation 6.6 represents the average number of bits needed to represent any given test sensor event x_i using an optimal encoding based on the discovered pattern d. Low-perplexity models better compress the data because fewer bits are needed to represent each sensor event. As with the other stability measures, the inverse difference between perplexity of the training data and perplexity on held-out test data can represent the stability of the discovered activities.

Cluster Evaluation Clustering is a mature area within data mining. As a result, several evaluation measures have been introduced and applied. These metrics assess the similarity or cohesiveness of individual clusters and the dissimilarity between different clusters. One such approach is to sum the pairwise similarity within each cluster weighted by the size of the corresponding cluster. Another is the Dunn index, which computes the ratio between the minimum intercluster distance to the maximum intra-cluster distance. The Dunn index calculation, shown in Equation 6.7 for n clusters, relies upon a function that calculates the distance between clusters i and j, $d(i,j)$, and between points within cluster k, $d'(k)$.

$$D = \min_{1 \le i \le n} \left\{ \min_{1 \le j \le n, i \ne j} \left\{ \frac{d(i,j)}{\max_{1 \le k \le n} d'(k)} \right\} \right\} \tag{6.7}$$

Another popular measure is the Davies–Bouldin index. This index computes the ratio of the aggregated within-cluster distances to the between-cluster separation, as shown in Equation 6.8. Smaller ratios indicate better clusters because they are compact and well separated from each other.

$$DB = \frac{1}{n} \sum_{i=1}^{n} \max_{i \ne j} \left\{ \frac{d'(i) + d'(j)}{d(i,j)} \right\} \tag{6.8}$$

Leave-One-Activity-Out In the case of zero-shot learning, the goal is to learn models for activities that do not have explicit training data. Evaluating these approaches thus consists of holding out samples for one activity class from the training data and using the samples to test the ability to learn the "unknown" class. Variations on this include leave-two-activities-out, in which multiple activity classes are held out from the training data in order to determine how effectively the algorithm can discriminate between activity classes when no data are available for either of the classes.

Boost Activity Recognition Activity recognition is typically applied to a fairly small set of commonly-occurring activity classes. As a result, the recognition algorithms face a challenge of highly-skewed class distributions, with the Other class being the dominating activity. Consider an activity recognition model that is trained to distinguish the Other class from the Phone Call class shown in Figure 6.1. Among

these two classes, Other makes up 99% of the samples. As a result, a majority classifier that labels each of the data points as Other will result in very impressive 99% accuracy. Ignoring the Other class in order to even the distribution is not an ideal solution either, because the algorithm must have a mechanism in place for rejecting points from the Other class if they are not explicitly modeled. Explicitly modeling the Other class faces problems apart from size, because this class actually represents the union of a number of distinct activities, thus the class is complex and difficult to model.

A solution to this problem is to discover the activities that comprise the Other class. Instances of the discovered activities, A_D, can be labeled and used to train and activity recognition algorithm to identify occurrences of the discovered classes in the same way that instances of the predefined activities, A_P, are used to train the algorithm to recognize their occurrences. The set of learned activities is thus $A = A_P \cup A_D$. As a result, the resulting class sizes tend to be more uniform and represent a greater diversity of activities that are being performed. In this case, traditional activity recognition performance metrics can be applied to the set A of activities to determine how well they are recognized as a whole. The performance can also be compared between activities in A_P and activities in A_D to see if the discovered activities are as easily learned and recognized as the predefined activities.

6.6 ADDITIONAL READING

The idea of zero-shot learning to learn models for classes that are missing from the training data was introduced by Palatucci et al.[125] for use in image processing applications and was later adapted for use in activity learning by Cheng et al.[126] The task of most effectively learning the high-level features has been further investigated by Russakovsky and Fei-Fei[127].

Frequency-based sequence mining was introduced by Agrawal and Srikant in the GSP algorithm[128] and was enhanced to improve algorithm performance by FreeSpan[129], PrefixSpan[130], SPADE[131], and dSPADE[132]. The Minimum Description Length theory was introduced by Rissanen[133] and has been employed for compression-based discovery by Cook et al.[134–136]. Elfeky et al.[137] provide an overview of methods to efficiently detect periodicity in time series databases that can be adapted for detecting periodicity in sensor event mining algorithms.

Data clustering is a heavily explored area. Clustering sequential data presents unique challenges. These challenges have been explored by researchers including Ukkonen[138], who designed string matching techniques using q-grams. Kullback and Leibler[139] introduced the notion of comparing probability distributions for a distance measure. The Damerau–Levenshtein distance measure[140] is one of the most popular edit distance measures for comparing sequences. However, Zhou et al.[141] improve this traditional measure to account for the time span of the sequence. Yang and Wang[142] also improve on traditional edit measures to not only maximize the global alignment of the two sequences but also to detect possible local alignments using probabilistic suffix trees. Alon et al.[143] successfully used dynamic warping to locate instances of

particular gestures in video data. Cluster evaluation measures have been defined by Dunn[144] and by Davies and Bouldin[145], among others.

Topic models and LDA has been extensively explored in the document mining community by researchers including Blei et al.[146] More recently, Huynh et al.[147] have utilized the method to discover activity patterns from wearable sensor data and Varadarajan et al.[148] have applied it to identify recurrent activity sequences from video data. The pairing of activity recognition and activity discovery was proposed by Cook et al.[134].

Other techniques have been explored for discovery of concepts and features in activity data that are not described here. One is the application of deep learning to activity discovery and modeling, as explored by Baccouche et al.[149] and for utilized discovery of relevant features by Coates et al.[150] Another is the idea of mining an outside source such as web-based activity descriptions to generate information needed to discovery and recognition activities from sensor data. This idea was introduced by Wyatt et al.[151] and was tested with activity models based on object sensors.

7

Activity Prediction

In addition to discovering and recognizing activities from sensor data, activity learning includes predicting activities. Activity recognition considers past or current sensor data in order to map the data to an activity label. In contrast, activity prediction (AP) algorithms hypothesize information about activities that will occur in the future. Two specific prediction questions are addressed in this chapter:

1. Given the sequence of events (or activities) that has been observed up to the current point in time, can we predict what event (or activity) will occur next (*sequence prediction*)?
2. For any given activity, can we predict when in the future it will occur again (*activity forecasting*)?

Activity predictions are valuable for providing activity-aware services in the home or using mobile phones, tablets, phablets, and other devices. For example, anticipating a resident's activities facilitates home automation. An activity-aware sensor network can use predictions of upcoming activities to automate the home in preparation for the activities (e.g., warm up the house before the resident returns home). By combining AP with activity recognition, AP can also be used to provide automated prompting. If the time window when the activity normally occurs passes and the activity recognizer did not detect that the activity was performed, a prompt can be issued and repeated until the user interacts with the environment or the activity has been performed by the user and detected by the activity recognizer.

In this chapter, we will introduce a compression-based approach to perform sequence prediction. Moreover, we will consider three alternative methods for predicting the timing of activity occurrences, which include time series-based

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

activity forecasting, probabilistic graph-based activity prediction, and induction of activity timing rules.

7.1 ACTIVITY SEQUENCE PREDICTION

Activity prediction is the cornerstone of activity-aware service algorithms. Part of this problem is sequential prediction, or using an observed sequence of events to predict the next event to occur. Symbolic sequence prediction is based on information theoretic ideas for designing lossless compression strategies. While we illustrate the algorithm using an example sequence of sensor events, the algorithm can be used to predict any symbol from a fixed vocabulary, including predicting navigation trails, locations within a home, or upcoming activities. Sequence prediction has also been applied to challenges outside the realm of activity learning, including biological sequence analysis, speech modeling, text analysis, and music generation. For any sequence of symbols that can be modeled as a stochastic process, the sequence prediction algorithm we describe here will use variable-order Markov models to optimally predict the next symbol.

Consider a sequence of events being generated by an arbitrary process, represented by the stochastic process $X = \{x_i\}$. We can then state the sequential prediction problem as follows. Given a sequence of symbols $\{x_1, \ x_2, \ \dots \ , x_i\}$, what is the next symbol in the sequence, x_{i+1}? Well-investigated text compression methods have established that good compression algorithms are also good predictors. In particular, a prediction model with an order that grows at a rate approximating the source's entropy rate is an optimal predictor. Text compression algorithms commonly make use of an incremental parsing approach to processing information. This incremental parsing feature is also useful for online predictors and forms the basis of one approach to sequence prediction.

Consider a stochastic sequence $x_1^n = x_1, x_2, \ \dots \ , x_n$. At time t, the predictor will output the next likely symbol x_t based on the observed history, or the sequence of input symbols $x_1^{t-1} = x_1, x_2, .., x_{t-1}$, while minimizing the prediction errors over the entire sequence. Theoretically, an optimal predictor must belong to the set of all possible finite state machines and it has been shown that Markov predictors perform as well as any finite state machine. Markov predictors maintain a set of relative frequency counts for the symbols that are observed at different contexts in the sequence, thereby extracting the sequence's inherent pattern. Markov predictors use these counts to generate a posterior probability distribution for predicting the next symbol. Furthermore, a Markov predictor whose order grows with the number of symbols in the input sequence attains optimal predictability faster than a predictor with a fixed Markov order. As a result, the order of the model must grow at a rate that lets the predictor satisfy two conflicting conditions. It must grow rapidly enough to reach a high order of predictability and slowly enough to gather sufficient information about the relative frequency counts at each order of the model to reflect the model's true nature.

The LZ78 data compression algorithm is an incremental text parsing algorithm that introduces such a method for gradually changing the Markov order at the appropriate rate. This algorithm is a modeling scheme that sequentially calculates empirical

```
Algorithm LZ78StringParsing(x)
  // x is sequence of observed symbols x_1, .., x_N
  // Initialization
  w = null
  Dictionary = null
  i = 1
  while i <= N
    v = Append(w, x_i)
    if v ∈ Dictionary
      w = v
    else
      Insert v in Dictionary
      w = null
      Increment the frequency for every prefix in Dictionary of phrase v
    end if
    i = i + 1
  done
return
```

FIGURE 7.1 *LZ78 string parsing algorithm.*

probabilities for each subsequence, or phrase, of the data, with the added advantage that the generated probabilities reflect phrase contexts observed from the beginning of the parsed sequence to the current symbol.

LZ78 creates and maintains a dictionary of substrings to use for compression. The algorithm parses an input string $x_1, x_2, \ldots, x_N$ into $c(N)$ substrings $w_1, w_2, \ldots, w_{c(N)}$ such that for all $j > 0$, the prefix of the substring w_j (all but the last character of w_j) is equal to some w_i for $1 < i < j$. This prefix property ensures that the dictionary of parsed substrings and their relative frequency counts can be maintained efficiently in a multiway tree structure called a trie.

As LZ78 is a compression algorithm, it consists of both an encoder and a decoder. When we are performing prediction, we do not need to reconstruct the parsed sequence so we do not rely on encoding or decoding phrases. Instead, we must simply construct an algorithm that breaks the input sequence (string) of events or states into phrases. The algorithm for parsing and processing an input symbol sequence is shown in Figure 7.1.

Example 7.1 Consider a string, $x^n = aaababbbbbaabccddcbaaaa$, that represents a sequence of locations visited by a resident in a home, where $a =$ Kitchen, $b =$ LivingRoom, $c =$ Bathroom, and $d =$ Bedroom. The algorithm can be applied to this sequence to determine the most likely location that will be visited next. An LZ78 parsing of this string would create a trie as shown in Figure 7.2 and would yield the phrases a, aa, b, ab, bb, bba, abc, c, d, dc, ba, and aaa. The parsing algorithm maintains frequency counts for all prefixes, or contexts, within the phrases w_i. For example, the context a occurs five times (at the beginning of the phrases a, aa, ab, abc, and aaa) and the context bb occurs two times (at the beginning of

the phrases *bb* and *bba*). As LZ78 parses the sequence, successively larger phrases accumulate in the dictionary. As a result, the algorithm gathers the predictability of successively higher-order Markov models, eventually attaining the universal model's predictability.

The drawback of LZ78 is its slow convergence rate to optimal predictability. This means that the algorithm must process numerous input symbols to reliably perform sequential prediction. However, LZ78 does not exploit all of the available information. For example, context information that crosses phrase boundaries is lost. Using our example string, the fourth symbol (*b*) and fifth and sixth symbols (*ab*) form separate phrases. Had they not been split, LZ78 would have found the phrase *bab*, creating a larger context for prediction. Unfortunately, the algorithm processes one phrase at a time without looking back. As the number of symbols in an input sequence grows, the amount of information that is lost across phrase boundaries increases rapidly.

This slow convergence rate can be partially addressed by keeping track of all possible contexts within a given phrase. In order to recapture information lost across phrase boundaries, an alternative sliding window approach can be considered. The Active LeZi, or ALZ, approach to sequence prediction maintains a variable-length window of previously seen symbols. Throughout the ALZ algorithm, the length of the window is set to k, the length of the longest phrase that has been parsed so far by LZ78. As a result, this enhanced algorithm allows LZ78 to construct an approximation to an order $k - 1$ Markov model. Our example trie shown in Figure 7.2 has a depth of three, which corresponds to the length of the longest phrase and indicates a Markov order of two.

Within the sliding window ALZ gathers statistics on all possible contexts, as shown in Figure 7.3. By capturing information about contexts that would cross

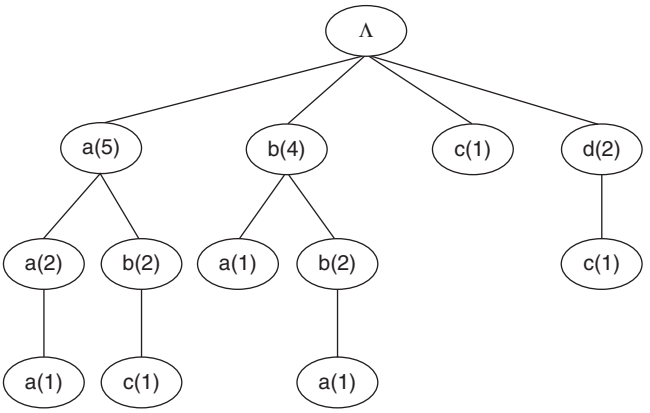


FIGURE 7.2 *The trie formed by the LZ78 parsing of the location sequence aaababbbbbaabccddcbaaaa. The numbers in parentheses indicate the frequency of the corresponding phrase encountered within the sequence as it is parsed.*

```
Algorithm ALZStringParsing(x)
  // x is sequence of observed symbols x_1, .., x_N

  // Initialization
  w = null
  Dictionary = null
  window = null
  MaxLength = 0

  i = 1
  while i <= N
    v = Append(w, x_i)
    if v ∈ Dictionary
      w = v
    else
      Insert v in Dictionary
      if (Length(v) > MaxLength)
        MaxLength = Length(v)
      end if
      w = null
    end if

    Add v to window
    if (Length(window) > MaxLength)
      Delete the first symbol in window
    end if

    // Increment frequency for related contexts
    Increment the frequency for every prefix in Dictionary that includes v
    i = i + 1
  done
return
```

FIGURE 7.3 *ALZ string parsing algorithm.*

phrase boundaries in the original LZ78, we can build a closer approximation to an order- k Markov model and converge more quickly to optimal predictability.

Example 7.2 Figure 7.4 shows the trie that is formed by an ALZ parsing of the input sequence *aaababbbbbaabccddcbaaaa*. This is a more complete model than the one shown in Figure 7.2 and it represents an order-MaxLength-1 Markov model.

ALZ's trie and corresponding frequency counts are managed at the same time as the input sequence is parsed. The more complex case occurs when the input sequence both rapidly increases maximum phrase length (and consequently the ALZ window size) and yet grows slowly enough that the most possible subphrases in the ALZ window add new nodes to the trie. Given that the maximum LZ phrase length increases by at most one for each new phrase, the most complex case occurs when each new LZ phrase is one symbol longer than the previous phrase. In the worst case, each subphrase in the ALZ window that includes the subsequent symbols adds a new node to the trie. This means that at order k, the trie increases by k^2 nodes before the model

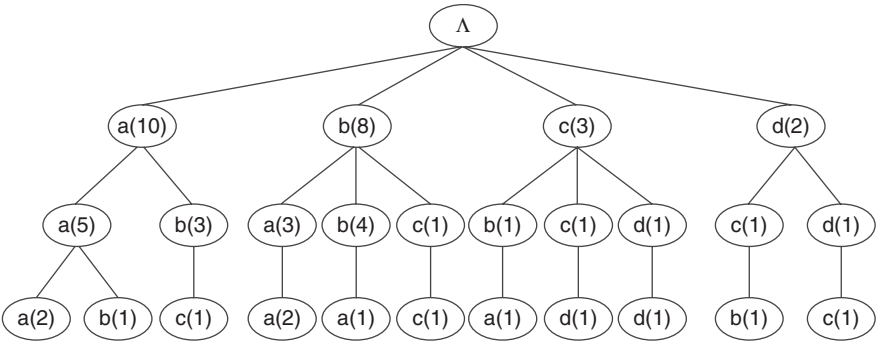


FIGURE 7.4 *The trie formed by the ALZ parsing of the location sequence* aaababbbbb aabccddcbaaaa.

transitions to order $k + 1$. Therefore, the number of nodes generated in the trie by the time the model attains order k is $O(k^3) = O(n^{3/2})$, because $k = O(\sqrt{n})$. In practice, however, the order tends to grow much more slowly because the worst case is interspersed with intervals of shorter phrases. The limiting case brings the space requirement to $O(n)$ and the time complexity to $O(n^{3/2})$.

Ultimately, we want to use the model to generate a probability distribution over the possible symbols that will occur next in the sequence. Given the distribution the sequence prediction algorithm outputs the symbol with the highest probability as the most likely event (or activity, or location, or state) to occur next.

To achieve better rates of convergence to optimal predictability, the predictor must lock on to the minimum possible set of states that the sequence represents. For sequential prediction, this is possible by using a mixture of all possible order models (i.e., phrase sizes) to assign a probability estimate to the next symbol. Multiple order models can be incorporated using the Prediction by Partial Match (PPM) family of predictors. PPM algorithms generate and fuse Markov models from multiple orders to build a probability distribution. As described earlier, ALZ builds an order- k Markov model. The PPM strategy then gathers information from Markov models of orders 1 through k to assign each possible next symbol its corresponding probability value.

Example 7.3 ALZ's window represents the set of contexts that can be used to compute the probability of the next symbol. Our example uses the last phrase *aaa* (which is also the current ALZ window). In this phrase, the contexts that can be used are all suffixes of the phrase, except the window itself (i.e., *a*, *aa*, and the null context).

Consider the case of computing the probability that the next symbol is *a*. We compute the probability separately using each available context order and combine the results to generate the prediction probability for a particular symbol. The trie in Figure 7.4 has a depth of 3 so for any particular prediction task we can consider contexts of order 0, 1, and 2. Following the leftmost path in Figure 7.4, we see that the symbol *a* occurs two out of the five times that the context phrase *aa* appears, the other

cases producing two null (end of phrase) outcomes and one b outcome. Therefore, the probability of encountering a at the context aa is 2 in 5, and we now escape to the order-1 context (that is, switch to the model with the next smaller order) with probability 2 in 5. This corresponds to the probability that the outcome is null, which forms the context for the next lower length phrase. At the order-1 context, we see the symbol a five out of the 10 times that we see the a context, and of the remaining cases, we see two null outcomes. Therefore, we predict the symbol a at the order-1 context with probability 5 in 10 and escape to the order-0 model with probability 2 in 10. Using the order 0 model, we see the symbol a 10 times out of the 23 symbols processed so far, so we predict a with probability 10 in 23 at the order-0 context. As a consequence, we compute the combined probability of seeing a, representing the Kitchen location in our example, as the next symbol as

$$\frac{2}{5} + \frac{2}{5}\left\{\frac{5}{10} + \frac{2}{10}\left(\frac{10}{23}\right)\right\} = 0.635.$$

We make a few observations about this approach for assigning probabilities. First, we note that it solves the zero-frequency problem. In the earlier example, if we chose only the longest context to calculate probabilities, it would have returned a zero probability for the symbol c, whereas lower-order models reveal that the probability should be non-zero. Second, the blending strategy assigns greater weight to higher-order models when calculating probabilities if it finds the symbol being considered in that context, while it suppresses the lower-order models owing to the null context escape probability. This strategy is consistent with the advisability of making the most informed decision.

The blending strategy considers nodes at every level in the trie from order 1 through $k - 1$ for an order-k Markov model, which results in a worst-case run time of $O(k^2)$. The prediction run time is thus bounded by $O(n)$. This assumes that a fixed-sized alphabet is used to represent what is being predicted (sensor events, ranges of sensor values, locations, states, or activities), which implies that the prediction algorithm can search for the child of any node in the trie in constant time. By efficiently maintaining pointers only to nodes in the tree that are likely to be expanded, the algorithm can be further improved to perform predictions in $O(k) = O(\sqrt{n})$ time. Additional information can be stored in the trie along with frequencies as needed. For example, elapsed time between observed symbols can be stored to predict not only the next event that will occur but also how much time will pass before the event occurs.

7.2 ACTIVITY FORECASTING

We now consider the task of predicting at what time in the future a particular activity will occur. To accomplish this task, we can borrow ideas from time series analysis. A time series is a sequence of random variable values over time, $x_1, x_2, x_3, \ldots, x_t$. Forecasting has been used in time series analysis to predict future values of a target variable given observed past and current values. Given the observed sequence through

time t, the forecaster will output values x_{t+1}, x_{t+2}, x_{t+3}, ... In the case of activity forecasting, the random variable of interest is X_a, the amount of time that will elapse until the next occurrence of activity $a \in A$. Given labeled training data, the values of X_a for past points in time can be calculated.

There are two assumptions that are typically made when performing forecasting. The first is that the data have a natural temporal ordering. The second is that the forecasted variable has numeric values. In the case of AP, both of these assumptions hold. We assume that the input to the prediction problem is a time-ordered sequence of sensor events in which the temporal ordering is enforced. The output of the prediction algorithm is an estimated number of time units until the targeted activity will occur, so the output can be represented using a numeric variable.

As with activity recognition, historic training data must be available in which each sensor event is tagged with the number of time units that elapsed between the sensor event and the next occurrence of the target activity.

Example 7.4 As an example, consider the labeled data shown in Figure 5.4. The values of variable $X_{\text{EnterHome}}$, the amount of time that will elapse until the next occurrence of the Enter Home activity, for each of these sensor events is shown in Figure 7.5. This example shows a forecasted value for each sensor event. Most time series analysis techniques assume that the observations are made at equal-length time periods. The forecasted values can be interpolated to provide values at equal intervals, as needed.

When performing activity prediction, the output of the forecast algorithm is the time that will elapse until the next occurrence of the targeted activity. Standard forecasting techniques can be used for this task. These include calculating a moving average or average value within a sliding window over the data. The moving average can be combined with an autoregressive function that generates a forecasted value based on a weighted function of past observed values, called an autoregressive

2009-07-19	10:18:59.406001	LivingRoom	ON	6060.594
2009-07-19	10:19:00.406001	Bathroom	OFF	6059.594
2009-07-19	10:19:03.015001	OtherRoom	OFF	6056.985
2009-07-19	10:19:03.703001	LivingRoom	OFF	6056.297
2009-07-19	10:19:07.984001	LivingRoom	ON	6052.016
2009-07-19	10:19:11.921001	LivingRoom	OFF	6048.079
2009-07-19	10:19:13.203001	OtherRoom	ON	6046.797
2009-07-19	10:19:14.609001	Kitchen	ON	6045.391
2009-07-19	10:19:17.890001	OtherRoom	OFF	6042.110
2009-07-19	10:19:18.890001	Kitchen	OFF	6041.110
2009-07-19	10:19:24.781001	FrontMotion	ON	6035.219
2009-07-19	10:19:28.796001	FrontMotion	OFF	6031.204
2009-07-19	10:19:31.109001	FrontDoor	CLOSE	6028.891
2009-07-19	12:05:13.296001	FrontDoor	OPEN	0.000

FIGURE 7.5 *Forecasted time elapsed values for Enter Home activity.*

moving average, or ARMA, model. In the case where the underlying process that generates the values changes over time, the function can be applied to value differences over time. The learned function can be integrated to map it back to the original data, resulting in an autoregressive integrated moving average, or ARIMA, approach.

One limitation of these standard techniques is that the function cannot always adequately model complex processes with nonlinear interactions between the features, as often occurs when learning activity models. Forecasting the numeric activity timing variables can thus be framed as a supervised learning problem where the classifier maps a feature vector onto the corresponding time unit value. As a result, any machine learning algorithm, such as regression, which maps input features onto continuous-valued output can be used. Such techniques include backpropagation and recurrent neural networks and support vector machines, described in Chapter 4. However, in addition to the feature values (described in Chapter 3) that represent the state at the current time t_i, additional features must be included. These features are called the "lag" values and represent the value of the output, or class variable, at previous time units $t_{i-L}, \ldots, t_{i-1}$. In the case of activity prediction, the lag values represent the forecasted activity elapsed time for the L past sensor events, where L represents the maximum lag.

A supervised learning algorithm must thus be selected that can handle numeric or discrete input values and that maps the values onto an output numeric value. Because AP must be updated after each sensor event, an algorithm should be selected that can update its model without a large computational cost. The SVM learning algorithm described in Chapter 4 is appropriate for such a problem. Another algorithm that fits these constraints with a lower computational cost is a regression tree. Here we describe how a regression tree is learned and then explain how it can be used as the backbone of an AP algorithm. Since this is a forecasting-based AP algorithm, we will refer to it as FAP.

Regression trees, like decision trees, are created in a top-down method. Moreover, like traditional decision trees, the tree is traversed from the root node to a leaf node. Each internal node queries one feature of the data point and a path is traversed based on the outcome of the query. When a leaf node is reached in a regression tree, information contained in the leaf node is used to generate a value for the data point.

Example 7.5 Figure 7.6 shows a sample regression tree that was automatically constructed from sensor data representing the predicted number of seconds that will elapse until the next occurrence of a Bed Toilet Transition activity.

While the structure of a regression tree is similar to a decision tree, the split criteria at each internal node and the value computation at the leaf nodes are different from a traditional decision tree. Most decision tree algorithms choose an attribute at each split point that maximizes information gain, as described in Chapter 4. In contrast, a regression tree selects an attribute based on maximizing the reduction in error. For a regression tree, the standard deviation in class values that result from splitting the dataset for a particular attribute is an estimate of the error for the corresponding

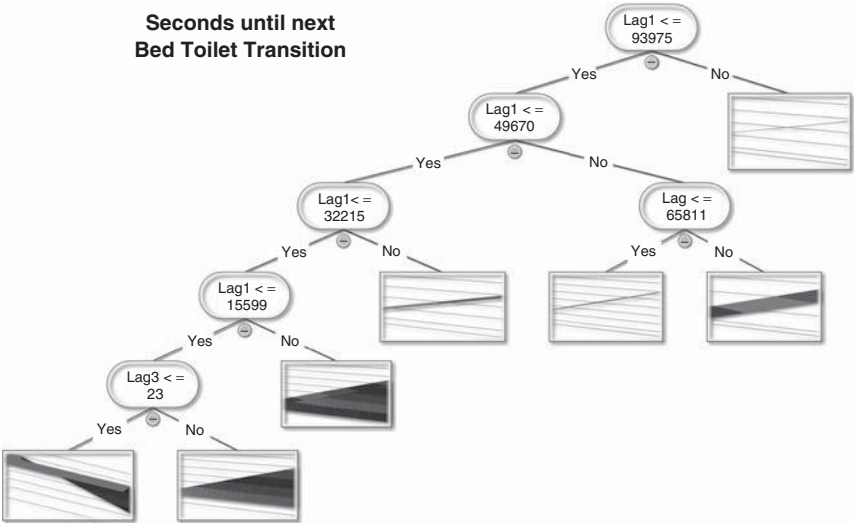


FIGURE 7.6 *A regression tree to output expected number of seconds until the next occurrence of a Bed Toilet Transition activity. Each internal node queries the value of a particular variable and each node contains a multivariate linear regression function. Each of the regression functions generates a prediction value as a function of the lags and other input feature values. The features include lag variables, or values of the predicted variable at previous time units. Lag1 indicates the prediction value at time t – 1 and Lag3 indicates the prediction value at time t – 3.*

subset of data points. Letting T represent the set of training examples, or labeled sensor events, the regression tree selects an attribute that maximizes gain as defined in Equation 7.1.

$$\text{Gain}(T, A) = \sigma(T) - \sum_{v \in \text{Values}(A)} \frac{|T_v|}{|T|} \times \sigma(T_v) \tag{7.1}$$

The attribute that maximizes gain (reduces the expected error) is chosen for querying and splitting at the internal node. Splitting terminates when all of the attributes have appeared along the current path or the error for the remaining subset of data points (the standard deviation of class values for the data points) is below a threshold fraction of the error in the original training dataset. Unlike traditional decision trees, each leaf node represents a linear regression of the class values for the data points that are contained in that leaf node. The multivariate linear model is constructed using linear regression. The model takes the form $w_0 + w_1a_1 + w_2a_2 + \ldots + w_ka_k$, where $a_1, a_2, \ldots, a_k$ are attribute values and the weights $w_1, w_2, \ldots, w_k$ are calculated using standard regression. Instead of using all of the attributes in the linear model, only those that are queried along the path to the leaf node are included in the linear model.

The model is then updated by removing as many attributes (variables) as possible. Using a greedy search, the attributes are removed if the removal does not affect the accuracy of the model. While most of the features described in Chapter 3 are numeric

and thus easily fit into this model, discrete attributes with v possible values can be incorporated by transforming them into a set of $v - 1$ binary attributes.

Like traditional decision trees, regression trees can be pruned to improve overall predictive performance. In order to perform pruning, linear models must be built for each internal node as well as each leaf. The regressed model takes the form $w_0 + w_1a_1 + w_2a_2 + \ldots + w_ka_k$, where $a_1, a_2, \ldots, a_k$ only represent values for attributes found below the current node. Once the tree is fully constructed, pruning is performed in a bottom-up manner along each path in the tree and the linear models at each node can be used to compare the error that would result from turning the internal node into a leaf with the error resulting from leaving the branch untouched.

When a new data point is processed, the tree is traversed from the root down to the appropriate leaf node. The resulting linear model at the leaf node can be evaluated to yield the prediction value. However, the value can be improved through a smoothing process, which is particularly valuable if the leaf node was constructed from only a few data points. Smoothing combines the leaf's model with the model for each internal node along the path from the leaf node to the root. At each internal node S, the corresponding class value, $C(S)$, is calculated as shown in Equation 7.2.

$$C(S) = \frac{n_i \times C(S_i) + k \times M(S)}{n_i + k} \tag{7.2}$$

where S_i and n_i refer to the node's i^{th} branch that was traversed for this data point and the number of training points that followed that branch, respectively. $M(S)$ represents the value that is calculated by the linear model at node S and k is a smoothing constant.

In order to use a regression tree as an activity forecaster, additional L features must be added to the feature vector. These represent the L lag values, or previous class values. Other features, called time indexes, are also generated. The time index provides an indication of where the data point falls along the timeline from the beginning of the data sequence to the current point in time. This is useful if the data points do not occur at equal time intervals, as was the case for the data shown in Figure 5.4. This timestamp feature can be calculated as shown in Equation 7.3.

$$\text{index}(e_i) = \frac{\text{time}(e_i) - \text{time}(e_1)}{\text{time}(e_n) - \text{time}(e_1)} \tag{7.3}$$

where n represents the number of data points in the entire sequence being processed and time (e_i) is a numeric representation of the time at which sensor event i occurred.

7.3 PROBABILISTIC GRAPH-BASED ACTIVITY PREDICTION

As we discussed in Chapter 4, probabilistic graphs such as naive Bayes graphs, hidden Markov models (HMMs), and conditional random fields can be used to calculate a probability distribution over activity labels given observed information such as sensor events. They can also be used for sequential AP by maintaining and updating a

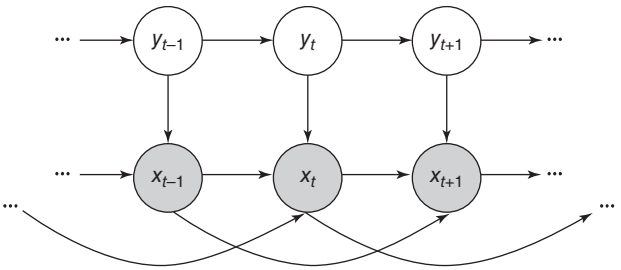


FIGURE 7.7 *An order 2 AR-HMM.*

probability distribution over the labels of activities and their corresponding likelihood of occurring next in the sequence.

As is shown in Figure 4.3, a HMM is a specific type of probabilistic graph that is well suited to incorporating temporal relationships between activities as well as evidence relationships between activities and sensor readings in its calculation of probability distributions. Note that in using an HMM, the joint probability distribution calculation shown in Equation 4.4.1 indicates that the activity transition probabilities $P(y_t|y_{t-1})$ are conditioned only on the previous activity. In addition, the activity evidence distributions $P(x_t|y_t)$ are conditioned only on the current activity. As a result, these traditional HMMs may not adequately capture long-term correlations between sensor events. This could affect the performance of activity prediction, particularly when activities are interwoven or performed in parallel.

For activity prediction, we consider a variation of an HMM that is known as an autoregressive HMM (AR-HMM). An autoregressive model considers the influence of past sensor events on the current sensor events and values, and the order (number of past events considered) is generally fixed. Figure 7.7 shows an example HMM with order 2. If all of the model variables had full observability, the conditional probability table (CPT) values could be estimated from sample data. However, in the AR-HMM some of the variables are not directly observed. These are the hidden variables y_i that represent the activity classes. As a result, we can use an expectation maximization (EM) algorithm to find a locally optimal maximum likelihood estimate of the parameters.

The idea behind EM is that if we know values of the variables for each node in the HMM learning the CPTs would be easy. As a result, in the E step we compute the expected value of each node using an inference algorithm and then treat the expected values as though the corresponding distribution was actually observed. In the M step, we maximize the parameter likelihood from the corresponding expected counts.

Example 7.6 Using our running example, if we consider the value of y_t as Eat and the value of y_{t-1} as Sleep, we can compute

$$P(y_t = \text{Eat}|y_{t-1} = \text{Sleep}) = \frac{EN(y_t = \text{Eat}, y_{t-1} = \text{Sleep})}{EN(y_{t-1} = \text{Sleep})},$$

where $EN(y)$ represents the expected number of times activity y occurs in the dataset given the current guess for the parameter values. Given the expected activity counts, we maximize the parameters and re-compute the expected counts in a repetitive process.

A similar process is applied to estimate the emission probabilities. Continuing our example, we can compute the probability that x_t refers to a particular sensor "M02" being fired at time t as

$$P(x_t = M02|y_t = Eat, x_{t-1} = M07) = \frac{EN(x_t = M02, y_t = Eat, x_{t-1} = M07)}{EN(y_t = Eat, x_{t-1} = M07)}.$$

As seen in Figure 7.7, each observable state in an AR-HMM has extra links from earlier observations. As a result, we need to calculate observation transition probabilities in the same way HMMs calculate transition probabilities for hidden states. The learning process thus requires the update of three sets of probabilities throughout the network: the hidden state transition probabilities $P(y_t|y_{t-1})$, the observation state transition probabilities $P(x_t|x_{t-1})$, and the emission probabilities $P(x_t|y_t, \ x_{t-1})$.

In this discussion, we have used HMMs and AR-HMMs to perform sequential prediction. However, HMMs can be used to model time as well. As a result, they can predict not only the next activity (or location, or sensor event) that will occur but also how many time units into the future it will occur. This is known as HMMs with explicit state duration. In these graphs, the probability of transitioning from a state to itself is modeled by a separate duration distribution. The likelihood of an activity occurring at a given time point in the future is thus dependent upon the current and previous activities, the current and previous observed sensor events, and the probability of the current activity having a duration that is one time unit greater than the current amount of time that has already elapsed for the activity.

7.4 RULE-BASED ACTIVITY TIMING PREDICTION

One approach for predicting the timing of activities is to learn logical rules that describe when an activity is typically initiated as a function of other events that are automatically detected from sensor events and values. Activities that are part of an individual's regular routine are usually initiated based on wall clock time or based on activity context.

Example 7.7 As an example of the first pattern, one activity of interest may be Pick Up Grandchildren From School, which occurs every Tuesday afternoon at 3:00pm. Another activity to track might be Taking Medicine while eating breakfast, which is an example of the second type of pattern.

A rule-based activity predictor, or RAP, learns patterns for each activity, *PA*, as a function of another reference activity, *RA*, with which it is correlated. RAP models

the relative time offset between initiation of *RA* and *PA* as a Gaussian distribution with a corresponding mean and standard deviation. The specific format for an activity pattern is thus:

<activity> [<relative_activity> <mean (s)> <standard_deviation (s)>]+

where the "+" means one or more relative activities.

Possible relative activities for *PA* include all other activities the individual performs, combined with periodic clock-based activities. The clock-based activities include the start of each year, month, day, week, and hour, as well as the start of each specific month of the year (January, February, … , December), each specific day of the week (Sunday, … , Saturday), and calendar-based events of interest (birthdays, holidays). This way, activity timings can be learned both for activities that occur at regular times and activities whose occurrence is relative to another activity. Each activity timing rule is represented by the name of the prompted activity *PA*, the name of the relative activity *RA*, the mean time delay in seconds between *RA* and *PA*, and the time standard deviation in seconds.

Example 7.8 If the Pick Up Grandchildren activity takes place every Tuesday around 2:40pm (+/−5 minutes), the associated prediction rule would be represented as:

Pick_up_grandchildren Activity_Tuesday 52800 300

If the individual picked up their grandchildren every Tuesday and Thursday around 2:40pm, then the rule would be:

Pick_up_grandchildren Activity_Tuesday 52800 300 *Activity_Thursday* 52800 300

On the other hand, if the individual takes medicine about ten minutes (+/−5 minutes) after breakfast begins each morning, then the corresponding rule would be:

Take_Medicine Eat_Breakfast 600 300

For each activity *PA*, RAP can learn a timing rule using a two-step process: consider timing patterns that are based on a single relative activity, and then consider timing patterns that are based on multiple relative activities. All of these possibilities are evaluated and the highest-ranked pattern is chosen for the prediction rule. First, consider the method for evaluating patterns based on a single relative activity. RAP must select a relative activity other than *PA* from among the activities the individual performs, along with the clock-based activities described earlier. An ideal relative activity *RA* is one that always occurs before each instance of the activity *PA* and always at the same (ideally small) time before *PA*. Therefore, the score for a relative activity *RA* should increase proportional to the number of times it co-occurs with *PA*, should decrease proportional to the variance in the time delay between each *RA* and

PA, and should decrease proportional to the absolute time delay between each *RA* and *PA*. Each potential relative activity *RA* is thus evaluated according to three properties: (i) The likelihood that activity *PA* occurs after each activity *RA*, (ii) The confidence in the distribution of the occurrence times of *PA* relative to *RA*, and (iii) The mean delay between *RA* and *PA*.

Property 1 is essentially the probability that *RA* occurs before each instance of *PA*. We estimate this probability from sample data. Given m instances of relative activity *RA* in the sensor data and n instances of activity *PA* occurring between two consecutive *RA*s, we estimate the occurrence likelihood as n/m. This forms the first factor of our overall rule score P, shown in Equation 7.4, for *RA* as the relative activity for *PA*. Property 2 measures the variance in the delay between the two activities. Again, we want minimal variance, so this factor will be in the denominator of Equation 7.4. There are two contributions to the variance in the delays. The first contribution is the actual variance in the distribution of the delays between each co-occurrence of *RA* and *PA*. For all such occurrences of *PA* preceded by *RA*, the rule-based learner models the time delay between the two activities as a Gaussian and computes the corresponding mean μ and standard deviation σ for these delays. The standard error $\sigma/\sqrt{n}$ can be used as an estimate of the confidence (smaller the better) that *PA* follows μ time units after *RA*. This comprises the second factor in P below, which decreases P based on increased distribution error. The second contribution to the delay error involves the $(m - n)$ occurrences of *RA* that are not followed by an occurrence of *PA*. This contribution to the distribution error can be estimated as the standard error based on a variance of one and a sample size of $(m - n)$. This comprises the third factor in P below, which decreases P based on increased distribution error due to the absence of a *PA* after *RA*. Property 3 prefers a smaller mean delay time μ. Therefore, the fourth factor $1/\sqrt{\mu}$ is included in P below, which decreases P as the mean delay increases. Combining all these factors, we arrive at the following predictability measure P shown in Equation 7.4.

$$P = \left(\frac{n}{m}\right)\left(\frac{1}{\sigma/\sqrt{n}}\right)\left(\frac{1}{\sqrt{m-n}}\right)\left(\frac{1}{\sqrt{\mu}}\right) \tag{7.4}$$

This predictability measure estimates the correlation between the two activities and thus represents how well *RA* plays the role of the relative activity for *PA*. If $m = 0$ or $n = 0$, we set $P = 0$. If $\sigma = 0$, we set $\sigma = 1$. If $\mu = 0$, we set $\mu = 1$. If $(m - n) = 0$, we set $(m - n) = 1$. The relative activity with the highest P value, along with its associated mean and standard deviation, are output as the activity's timing rule. If two relative activities have the same P value, we prefer the one with the smaller mean.

The second step in the process is to consider rules where the prompt activity *PA* occurs relative to several other relative activities, not just one. While timing patterns can be learned as a function of multiple relative activities, considering all subsets of activities as potential patterns is not computationally tractable. However, RAP can consider patterns that involve subsets of selected clock or calendar-based events such as the months of the year (January, February, … , December), the days of the week (Sunday, Monday, … , Saturday), and the hours of the day, since many activities

occur relative to specific sets of months, days or hours (e.g., leaving for work at 7am Monday through Friday). Additional relative activities can be included for this, such as month-of-year, day-of-week, and hour-of-day, where their predictability P values are computed as the sum of the above-average P values of each individual month, day, or hour within the set. If one of these multiple relative activity patterns wins out over all the others, then the output pattern consists of all the individual month, day, or hour relative activities whose frequency is in the upper half of the range of normalized frequencies.

Example 7.9 Using our example of leaving for work at 7am Monday through Friday, the day-of-week relative activity would be considered by summing the above-average P values for each individual day of the week. RAP would detect that the frequencies for Sunday and Saturday are low, and these days are thus not included. Therefore, the final activity timing pattern would look as follows (assuming 7am $+/-$ 15 minutes):

Leave_for_Work *Activity_Monday* 25200 900 *Activity_Tuesday* 25200 900
Activity_Wednesday 25200 900 *Activity_Thursday* 25200 900
Activity_Friday 25200 900

A feature of the rule-based approach is that the expected timing of an activity is learned together with the typical timing variance. If the prediction rules are used in the context of an application such as activity prompting or home automation, for example, the application software will need to monitor the current time, the activity context as determined by an activity recognizer, and the activity timing rules that were learned. When an action such as turning on a device or issue a prompt is warranted, it can be sent to the appropriate device, such as a touch-screen computer located in the home or a mobile device.

7.5 MEASURING PERFORMANCE

We summarize some methods for prediction evaluation here.

Leave n-at-the-End-Out Testing Many of the performance measures that were introduced in Chapter 5 can be used to evaluate the performance of sequential prediction algorithms as well. One important distinction, however, is the fact that unlike many standard machine learning datasets, the data points used to train and test sequential prediction algorithms are not independent. As a result, the evaluation method cannot randomly select data points to hold out for testing because the held out data points are part of a sequence and therefore play a role in the definition of other data points as well. Instead, data at the end of a sequence can be held out for evaluation. In this way, the data leading up to the held out set can be used for training and is not interrupted by data that was extracted for the hold out set. If one point is evaluated

at a time then the last data point is held out for testing. However, multiple points at the end can be held out to determine how effectively the algorithm can predict sensor events, user locations, or activities that occur multiple points into the time horizon.

While data can be effectively held out from the end of the sequence to evaluate a prediction algorithm, the evaluation method should be careful in selecting data for training. If the evaluation procedure is repeated in order to allow every data point to be used as a test instance, and the data leading up to the test instance is used for training the model, then the amount of training data will vary for each test point. As a result, a sliding window evaluation should be used in which a fixed number of data points m are used in one iteration of the evaluation process. If horizon n of future data points are predicted, then the first $m - n$ data points are used to train the model and the remaining n points are used for testing. The window can then move one position over in the sequence of available data and the process can be repeated. The method can be repeated for multiple window sizes as needed to show the sensitivity of the performance to alternative choices of window size.

Example 7.10 We can train the model on 10 symbols, test on the last symbol, and repeat the process 13 times in order to eventually process the entire sequence. If we employ the sequential prediction approach described in section 7.1, then the correct symbol will be output 5 out of the 13 iterations, resulting in an accuracy of 0.38. The performance is fairly low for this example because there are a very small number of data points on which to train the algorithm.

Jaccard Index Another method to evaluate the accuracy of sequential predictions is the Jaccard index. The Jaccard index, or Jaccard similarity coefficient, compares a set of labels generated by the model with the corresponding set of ground truth labels, as shown in Equation 7.5.

$$\text{Jaccard}\left(\text{Predicted}_{e\in\text{Events}}, \text{Actual}_{e\in\text{Events}}\right) = \frac{|\text{Predicted} \cap \text{Actual}|}{|\text{Predicted} \cup \text{Actual}|} \tag{7.5}$$

Note that this measure corresponds to the calculation of classification accuracy as shown in Equation 5.11. The Jaccard index for our sequential prediction example is thus 5/13, the same as the accuracy measure. However, the numerator and denominator of the ratio in Equation 7.5 can be replaced by distance measures that calculate how close the corresponding symbols are in the sequence.

Mean Absolute Error (MAE), Mean Squared Error (MSE) If a forecasting technique is used to estimate when an activity will occur, then the typical supervised learning notions of correct, incorrect, true positive, and false positive do not directly apply. Instead, we want to know how close the predicted value is to the actual value, and we want to accumulate these differences over every test point. Mean absolute error directly measures this quantity. As shown in Equation 7.6, the absolute value of the difference between the predicted and actual value is summed and normalized

over each of the data points. A well-known alternative to mean absolute error (MAE) is mean squared error (MSE), computed in Equation 7.7.

$$\text{MAE} = \frac{\sum_{e=1}^{\#\text{events}} |\text{predicted}(e) - \text{actual}(e)|}{\#\text{events}} \tag{7.6}$$

$$\text{MSE} = \frac{\sum_{e=1}^{\#\text{events}} (\text{predicted}(e) - \text{actual}(e))^2}{\#\text{events}} \tag{7.7}$$

Mean Signed Difference (MSD) Both the mean absolute error and the mean squared error disregard the direction of the error (predicting lower or higher than the actual value of the activity timing). The mean signed difference, shown in Equation 7.8, takes the direction into account in the aggregated error calculation.

$$\text{MSD} = \frac{\sum_{e=1}^{\#\text{events}} \text{predicted}(e) - \text{actual}(e)}{\#\text{events}} \tag{7.8}$$

Root Mean Squared Error (RMSE), Normalized Root Mean Squared Error (NRMSE) Yet another common measure of prediction error is the RMSE, shown in Equation 7.9. Like the earlier measures, this formula aggregates the difference between predicted and actual error and squares each difference to remove the sign factor. The square root is computed of the final estimate to offset the scaling factor of squaring the individual differences.

$$\text{RMSE} = \sqrt{\frac{\sum_{e=1}^{\#\text{events}} (\text{predicted}(e) - \text{actual}(e))^2}{\#\text{events}}} \tag{7.9}$$

Normalized Root Mean Squared Error (NRMSE) A difficulty with the measures that have been summarized so far is that the values are sensitive to the unit size of the predicted value. In the case of activity forecasting, a prediction that is one minute longer than the actual time (i.e., predicting a Bed Toilet Transition occurring in 6 minutes rather than 5) yields a smaller MAE, MSE, MSD, or RMSE value than if the prediction were made in seconds (i.e., the prediction was 360 seconds instead of 300, yielding an error of 60). Normalized versions of the error measures thus facilitate more direct comparison of error between different datasets and aids in interpreting the error measures. Two common methods are to normalize the error to the range of the observed data or normalize to the mean of the observed data. These formulas for the normalized root mean squared error are given in Equations 7.10 and 7.11.

$$\text{NRMSE} = \frac{\text{RMSE}}{\text{Max}\,(\text{Actual}) - \text{Min}\,(\text{Actual})} \tag{7.10}$$

$$\text{NRMSE} = \frac{\text{RMSE}}{\text{Mean}\,(\text{Actual})} \tag{7.11}$$

Pearson's Correlation Coefficient (r) Correlation, often measured as a correlation coefficient, indicates both the strength and the direction of a relationship between two variables. A number of different coefficient calculations have been used for different situations. The most popular is the Pearson product-moment correlation coefficient, which is obtained by computing the ratio between the covariance of the two variables to the product of their standard deviations. This is shown in Equation 3.11. When evaluating forecasting algorithms, n represents the number of evaluated data points and the correlation coefficient is used to estimate the correlation between the predicted values output by a learned model and the actual observed values. The correlation coefficient is $+1$ in the case of a perfect linear relationship and -1 in the case of an inverse relationship. A coefficient of 0 indicates that there is no linear relationship between the two sets of values. Calculating the correlation coefficient makes it easier to intuitively evaluate the scale-free performance of a forecasting algorithm because the coefficient range is $r = [-1 \ \ldots \ +1]$, regardless of the scale of the variable that is being predicted. In the case of activity prediction, the correlation coefficient is calculated as shown in Equation 7.12.

$$r = \frac{\text{Combined}}{\sqrt{\text{Predicted}} \times \sqrt{\text{Actual}}} \tag{7.12}$$

where the numerator represents the covariance of the predicted and actual values and is calculated as $\text{Combined} = (\Sigma_{e=1}^{n}(predicted(e) - \overline{predicted}) \times (actual(e) - \overline{actual}))/(n-1)$. The denominator represents the product of the standard deviations for the predicted values and the actual values, calculated as $\text{Predicted} = (\Sigma_{e=1}^{n}(predicted(e) - \overline{predicted})^2)/(n-1)$ and $\text{Actual} = (\Sigma_{e=1}^{n}(actual(e) - \overline{actual})^2)/(n-1)$.

Example 7.11 Table 7.1 summarizes the forecasting performance metrics for an example AP task. In this example, 80,000 data points were used to train a model to predict the number of seconds that would elapse before a Bed Toilet Transition activity was performed. In addition to utilizing the bag of sensors, sensor counts, and time of day features that are described in Chapter 3, additional forecasting features that index the predicted value at the previous three time units were included. The regression tree that was learned from this data is shown in Figure 7.6. Figure 7.8 shows the 1-step-ahead predicted values and the actual values for the Bed Toilet Transition times based on this learned model over a period of one week.

After the tree is learned, it is used to generate predicted values for 20 data points that were held out from training. Table 7.1 shows the actual and predicted values for the data points, along with the associated error for each data point. As can be seen, the tree tends to under-predict the time until the next occurrence of the activity, although there are occasionally very small values output by the tree. Table 7.2 summarizes the forecasting performance metrics for this example AP problem.

TABLE 7.1 Actual and Predicted Elapsed Time Until the Next Occurrence of a Bed Toilet Transition Activity. The Associated Error is Shown for Each Data Point

Actual	Predicted	Error
37,779	37,680.582	−98.418
37,770	37,674.573	−95.427
31,165	37,665.561	6,500.561
31,159	31,049.914	−109.086
31,147	31,043.903	−103.097
31,076	31,034.769	−41.231
31,016	30,963.587	−52.413
31,010	30,903.436	−106.564
30,999	30,897.452	−101.550
30,976	30,886.450	−89.550
30,967	30,863.393	−103.607
27,725	30,854.374	3,129.374
27,631	27,603.967	−27.033
27,627	27,509.727	−117.273
27,143	27,507.139	364.139
27,139	27,021.923	−117.077
27,115	27,017.915	−97.085
27,108	26,994.065	−113.935
27,100	26,987.048	−112.952
27,090	26,979.038	−110.962

7.6 ADDITIONAL READING

The LZ78 compression algorithms were introduced by Ziv and Lempel[152] and were adapted to address the problem of sequential prediction by Feder et al.[153]. Bhattacharya and Das[154] investigated methods to improve the convergence rate for the sequential prediction algorithm and Gopalratnam and Cook[155] incorporated a sliding window to improve predictive accuracy. Begleiter et al.[156] compare several alternative variable-order models for sequence prediction.

Quinlan introduced the idea of a regression tree with the M5 algorithm[157]. Decision trees are appealing because of the ability to interpret the learned model and also because training time is typically shorter than for neural network and SVM approaches. In addition, Utgoff[158] proposed methods to allow incremental induction of decision trees. This allows the activity model to be refined as new training data becomes available, without a need to store all of the training data and learn the model from scratch with each new batch of training data. Recurrent neural networks have

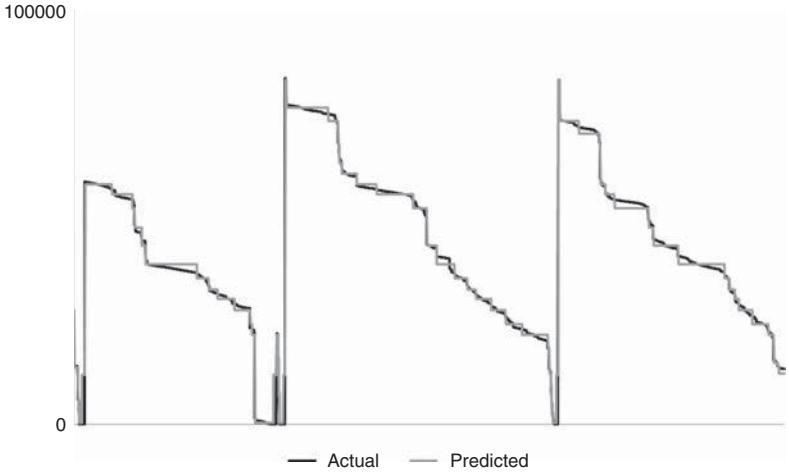


FIGURE 7.8 *Actual and predicted class values over a three day period based on a regression tree model. In this graph, the x-axis represents the time of day (from the beginning of day one to the end of day three) and the y-axis represents the number of seconds (predicted or actual) that will elapse before the next occurrence of the target activity, Bed Toilet Transition. The class variable is the number of seconds that will elapse before the next occurrence of the target activity.*

TABLE 7.2 Forecasting-Based Performance Evaluation for Regression Tree Algorithm Applied to Bed Toilet Transition Activity Prediction

Performance Metric and Value	
MAE	579.5667
MSE	2,617,237.0000
MSD	419.8407
RMSE	1,617.7880
NRMSE-Range	0.1514
NRMSE-Mean	0.0539
r	0.8943

been investigated by Mahmoud et al.[159] to predict occupancy regions in a home. The original ARIMA model was introduced by Box and Jenkins[160].

Krumm and Horvitz[161] applied Bayesian updating to infer the next location for drivers based on grid locations. The notion of explicit state duration HMMs has been explored by Dewar et al.[162], while autoregressive HMMs have been extensively used for applications such as speech processing[163] and visual tracking[164]. The idea of rule-based activity timings was investigated by Cook and Holder[165] and was used to generate automated activity initiation reminders.

8

Activity Learning in the Wild

Discovering, recognizing, and predicting activities are all challenging problems because of the complex nature of sensor data and the variances found in human behavior. What can possibly make activity learning harder? Performing activity learning in real-world, unconstrained settings. In this chapter, we provide techniques for addressing some of the real-world complexities that arise when activity learning is released into the wild.

8.1 COLLECTING ANNOTATED SENSOR DATA

A primary challenge that researchers face is obtaining a sufficient amount of labeled data to adequately train the machine learning algorithms. Many activity learning algorithms, including activity recognition and activity prediction, rely upon the availability of training data, or sensor data that both reflects realistic conditions and is accurately labeled with the corresponding activity labels. If sensor data is collected in a laboratory setting then annotation is straightforward because subjects are performing the activities that were specified by experimenters. In the wild, however, behavior is not pre-scripted and sensor data must be labeled during or after the sensor data is generated. Here we look at three approaches to the problem of collecting annotated sensor data. First, we highlight points to consider when annotation is performed manually. Second, we introduce methods for indirectly obtaining labeled data from other related sources of information. Finally, we describe techniques for synthetically generating labeled data in order to boost the performance of supervised learning algorithms.

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

Manual Annotation of Data The most common method of collecting annotated sensor data is to label it manually. This is so common that tools have been generated, such as the VARS video annotation tool shown in Figure 8.1, to facilitate labeling. This is an example of post-processing annotation in which labels are provided after the data is generated. Real-time data annotation in laboratory settings can be assisted by tools such as the real-time annotation tool (RAT), shown in Figure 8.2, which allows experimenters to mark activities that are currently being observed. The tool stores the activity labels and other notes in a relational database together with the raw sensor data in order to facilitate analysis. Real-time data annotation outside the laboratory can be accomplished by keeping a device such as a smart phone nearby and indicating the activities that are being performed at the time they are performed. This approach reduces the chance of error due to memory lapses, but disrupts the routine itself and thus can add noise to the data collection. Real-time annotation can be performed by an observer rather than the subject, although the positioning of the annotator is tricky because they need to be close enough to accurately observe the activity but far enough to not disrupt the behavior. On the other hand, labeling collected data can be very laborious. Researchers have reported that labeling discrete event sensor data can take as much as ten minutes for every hour of generated data and labeling dense sampled data can require two orders of magnitude more time than performing the activity itself.

Another consideration is the selection of an annotator. In any manifestation, manual labeling is susceptible to labeling mistakes. If the annotation is performed by an individual other than the one who performed the activities, mislabeling may also occur due to subjective interpretation of the situation. One way to improve the consistency and accuracy of the labeling is to obtain labels from multiple annotators. If the

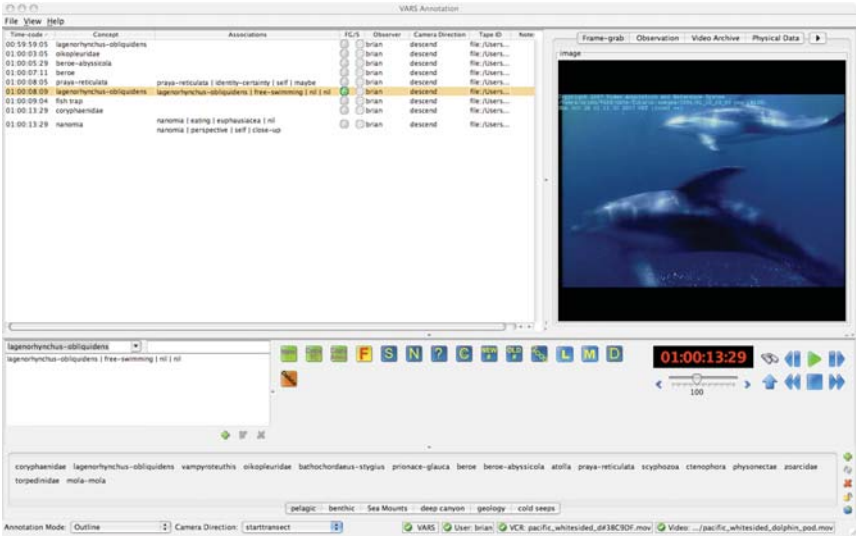


FIGURE 8.1 *The VARS video annotation tool allows a researcher to label aspects of an activity scene while watching the corresponding video.*

FIGURE 8.2 *An experimenter observes a participant performing everyday activities via web cameras and provides activity labels remotely using the RAT.*

data is presented in an easy to understand manner, crowd sourcing can even be used to generate a large number of labels for available data. For reliable annotators, majority voting can be used to resolve the discrepancies, otherwise techniques need to consider the reliability of the source when combining labels from multiple annotators. In these cases, the level of agreement between annotators is typically reported using the Kappa statistic shown in Equation 5.19. The inter-annotator agreement may vary by activity class but the statistic does take into account the agreement occurs by chance and offers an indication of the consistency and reliability of the labels. In general, such labeling is considered successful when the Kappa value is greater than 0.6.

Indirect Annotation from External Sources Another method to obtain the labeled data that is needed for activity learning is to glean information from other sources. Individuals may not want to spend time examining raw sensor data to categorize it or to push a button indicating the activity they are performing when they are performing the activity. On the other hand, many individuals are eager to post information about their daily routines using social media.

Example 8.1 Figure 8.3 shows some example tweets that reference activities. As the entries indicate, these postings provide information about the individuals' routines and activities. While tweets T1, T4, and T5 refer to activities that were performed in the past, tweet T3 describes a current activity and tweet T2 mentions an activity that is expected to occur in the future. As a result, this ubiquitous type of social media can be used to label past, current, and anticipated sensor data with the corresponding activity label.

One challenge in using an external source of information such as social media posts is in trying to make sense of the information and correctly associating it with the

T1: I just made this entire cake from scratch as a project for my English class! #proud
T2: I am going to try to make roast duck for the first time tonight for my family - wish me luck
T3: Watching Gravity, it is keeping me up past midnight #GoodMovie
T4: Funny how fast you can get up in the morning once you realize you overslept - I got ready in five minutes and made it to my 8am class on time
T5: Biking with Larry, we made it up the Lewiston grade this morning #biking #riders

FIGURE 8.3 *Example tweets that indicate an individual's activities.*

appropriate activity. In an ideal setting, external annotation of data works as follows. An individual posts a message. The content of the message is parsed to extract (i) the activity that is mentioned, (ii) who is performing it, and (iii) when. Given these three parameters, the sensor data that are collected at the posted time for the referenced individual can be labeled with the described activity. For example, given tweet T4 in Figure 8.3 the sensor data at 8am for the poster can be labeled as Attend Class and sensor data up to 7:55am can be labeled as Sleeping.

The second challenge for this approach to annotating data is preserving privacy. Even though individuals intend to share text updates describing their routines and interesting events, they may not be aware that the updates are used to label sensor data with their activities and to model activities for future recognition. This area of research is critical for activity learning and is largely unexplored.

Using this external information is ripe with technological challenges as well. First, not all posts contain easily-discernible activity references. However, there is content in the post that can be used to identify the activity that is described. To do this, a media-based activity recognition algorithm can be trained to recognize the activity category of a posting given a training set of postings that are clearly labeled. Because of inherent differences in written expressions between individuals, a "local" classifier would be trained for each individual.

The next obstacle to using social media for labeling activity data is that while there exists a wealth of social media data on the Internet, the data for any one user is sparse. To overcome this obstacle, the media-based classifiers can leverage data that are available from friends of the user. Recall from the discussion in Chapter 4 that boosting can combine multiple classifiers from different individuals to produce a stronger ensemble classifier. For our labeling task, we will harness the insights of a set of local classifiers, one for each individual, to create a "collaborative" classifier. Building on the assumption that friends and colleagues will tend to refer to similar events and will describe activities using similar wording, we include in a person's collaborative ensemble the local classifiers for the person and for all others that are connected to the person in a social network.

Figure 8.4 shows a portion of a social network that can be used in collaborative boosting of a media-based activity classifier for one individual, notated in the figure as user u_1. In the collaborative algorithm, u_1's local model is enhanced by considering not only its own classification of a data point but also the voting-based classification

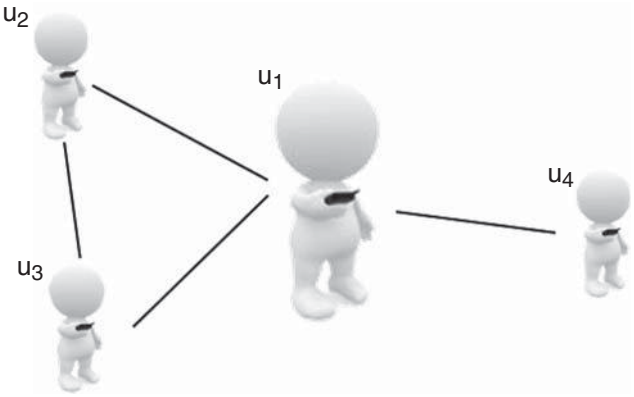


FIGURE 8.4 *Illustration of a social network used to train an activity classifier for user* u_1.

of the same data point by all of u_1's connections (in this case, by the classifiers for u_2, u_3, and u_4). Each connection's vote is weighted by the strength of the connection, which can be measured based on the distance in the network or the frequency of connection between the two individuals.

As with the AdaBoost algorithm, the local classifier is updated by using a dataset in which the weights of the data points are adjusted according to its own performance (internal boosting) as well as by the discrepancy between its label for a data point and the voting-based labeling of the user's friends (collaborative boosting). In addition, each user can "borrow" training data that are available for his/her friends and try classifying that data. If the data of the local classifier's output differs from that of the lender's output for a borrowed data point, then the correct label can be used to further train the individual's local classifier.

Achieving Balance in Labeled Data If training data are collected in a laboratory setting, the experimenter completely controls the conditions and so the method of performing the activity, the timing, and the environment are ideal for analyzing and labeling the sensor data. In these settings, the participants usually perform each of the activities so that training data are also well balanced between the activity classes. In the wild, these conditions do not hold. The data are not only complex and noisy, but also typically very imbalanced. While individuals may spend 1/3 of their time sleeping (as indicated in Figure 6.1), they may only exercise 20 minutes a day that comprises 1/72 of collected data. If the data are collected over just a few days, the Exercise activity class (the minority class) may be underrepresented in comparison with the Sleep class (the majority class) although both are important to learn. The problem becomes more apparent when learning activities that do not occur regularly, such as guests visiting on a holiday. In these cases, there may not be sufficient samples to adequately learn the smaller, less frequent activity classes.

The goal of many supervised learning algorithms is to optimize recognition accuracy for the entire dataset, which means performance on a minority class may

be overlooked. In fact, a simple classifier that labels all data points as members of the majority class will achieve a 96% recognition accuracy for the Exercise/Sleep problem, despite incorrectly classifying all of the Exercise samples. If activities are being learned in order to monitor and promote healthy lifestyles, then identifying Exercise occurrences is more critical than achieving optimal overall accuracy for the majority class.

The imbalanced class distribution problem has vexed researchers for over a decade and thus has received focused attention. Two common techniques to address this learning situation are cost-sensitive learning and resampling. Cost-sensitive learning methods counter the underlying assumption that all errors are equal by introducing customized costs for misclassifying data points. By assigning a sufficiently high cost to minority data points, the algorithm may devote the necessary attention to these points and subsequently learn an effective class model. Similar to the confusion matrix shown in Table 5.1, a cost matrix needs to be meticulously generated. For class imbalance situations, the positive (minority) class is often of most interest. As a result, the cost for false negatives is typically greater than the cost for false positives ($C_{FN} > C_{FP}$).

There are many ways to integrate cost sensitivity into individual classifier types. One approach that is independent of the choice of classifier technique is to integrate cost sensitivity into a boosting method. Recall that the boosting method, described in Chapter 4, weights data points based on their classification accuracy in the previous iteration. The individual classifiers from each iteration are then combined to generate a final ensemble classifier. For cost-sensitive learning, the initial weights (w_i^1) for each data point are determined based on the cost matrix rather than set as $1/N$ for N data points.

A popular approach to imbalanced class distributions is to resample or modify the dataset in a way that balances the class distribution. Determining the ideal class distribution is an open problem and in most cases it is handled empirically. Naive resampling methods include oversampling the minority class by duplicating existing data points and undersampling the majority class by removing chosen data points. One caution with this approach is that random oversampling and undersampling increases the possibility of overfitting and discarding useful information from the data, respectively.

An alternative way of oversampling is to synthetically generate new minority class samples. The synthetic minority oversampling technique (SMOTE), creates artificial data based on similarities between minority data points that exist in the training data. To do this, the k-nearest neighbors are first identified for each minority data point x_i. The k-nearest neighbors are defined as the k elements in the minority class of the training data who have the smallest Euclidian distance to x_i in the multidimensional feature space. Next, a point is randomly chosen along the line connecting one of the nearest neighbors $\hat{x}_i$ to x_i and is added as a new synthetic example of the minority class. Figure 8.5 shows an example of the SMOTE data generation process for a two-dimensional feature space.

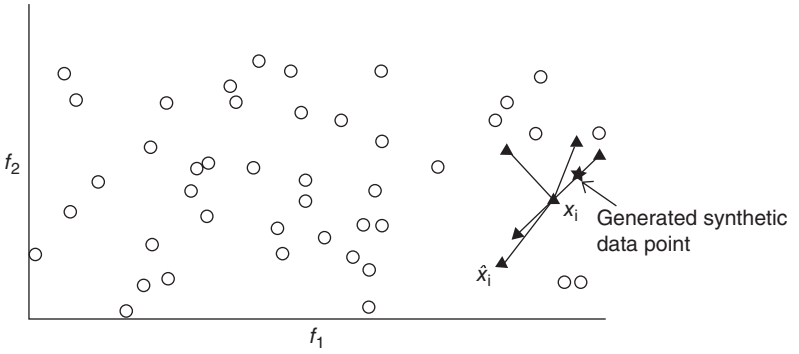


FIGURE 8.5 *Example of SMOTE synthetic data point creation for* k = 5 *nearest neighbors of point* x_i.

Active Learning In many situations, some labeled training data are available but they are too scarce to learn reliable activity models. In situations where more training data are needed but obtaining correct labels is expensive, active learning techniques can be used to select data for labeling that is most beneficial to the activity learning process. Active learning algorithms select an informative unlabeled data point and ask an oracle for the label of the instance. For activity learning, the oracle may be the individual whose activities are being modeled or may be an external annotator. The oracle provides a label for the data point and then the newly-labeled data are added to the training dataset. The value of an unlabeled data point for oracle querying may be based on factors such as the classifier's uncertainty of the point's label, disagreement between multiple classifiers on the data point, the margin distance to the class boundary, or the anticipated improvement in classifier accuracy for the entire dataset based on the label for the data point.

In order to learn a wide spectrum of activity classes for a large number of individuals, an enormous amount of training data is needed. Instead of querying an oracle for the label of one specific sensor sequence, it would be advantageous to pose a more general type of situation to the oracle that encompasses a set of similar sequences. In addition to getting more labeled data from a generalized query, such a general question may be easier for an individual to answer than trying to recollect the activity that occurred at a specific place and time.

For the generalized active learning (GAL) approach, we assume that a function exists to measure the informativeness of a data point. For activity learning, the informativeness may be indicated by the dissimilarity of the data point to data that are already labeled or the frequency with which similar data points occur in the unlabeled space. As before, the input data are denoted by x and the class label is pulled from a set y. The function that measures the informativeness of x is denoted by $\Phi(x)$ and the most informative data point is denoted by $x*$. The similarity between each pair of data points in the labeled dataset L is computed and stored in a proximity matrix M.

As with most active learning algorithms, GAL first trains a classifier C on the available labeled training data, L. GAL then determines the unlabeled data point that

represents the most informative instance $x*$. This algorithm measures instance informativeness using a density weighted metric. In particular, it measures the similarity of each data point x to other unlabeled instances and the dissimilarity of x to labeled instances. Optimizing the combination of these two factors results in Equation 8.1.

$$x^* = \arg\max_x \left[(1-\alpha) \times \Phi(x) + \alpha \times \frac{|L| \sum_{u=1}^{|U|} M[x, x_u]}{|U| \sum_{l=1}^{|L|} M[x, x_l]} \right] \tag{8.1}$$

In Equation 8.1, parameter α balances the contribution of density in comparison with the base informativeness measure, $\Phi(x)$. The proximity matrix, M, stores the similarity between each two points in the dataset. The second term shows the contribution of density with respect to both labeled and unlabeled data. The first term, $\Phi(x)$, is a base informativeness measure. One type of base informativeness measure that we can use is a measure of entropy, as shown in Equation 8.2, where y_i ranges over all possible label values.

$$\Phi(x) = -\sum_i P_\theta(y_i|x) \log P_\theta(y_i|x) \tag{8.2}$$

Next, the nearest neighbors of $x*$ are identified along with its strangers (data points that are not near in the multidimensional space). The stranger set is obtained by randomly selecting points from the rest of the unlabeled data. The nearest neighbors are temporarily assigned a label of 1 while strangers are temporarily assigned a label of 0. A rule induction classifier such as a decision tree (described in Chapter 4) or an inductive logic programming classifier is employed to distinguish between the 0 and 1 classes. In addition to distinguishing neighbors from strangers, the resulting induced model also identifies the most discriminating features of the nearest neighbor data points. The learned set of rules is presented to the oracle in order to obtain an activity label for the entire set. The label l and confidence c that are returned from the oracle is used to update the training dataset. The process is repeated until a maximum number of queries are posed or sufficient activity recognition accuracy improvement is achieved. The algorithm is summarized in Figure 8.6.

Example 8.2 An example query that can be generated by GAL when learning models for activities Hand Washing, Sweeping, and Cooking as described in Chapter 5 is:

> *"What is the activity label if the duration is 1–12 minutes and 78%-100% of the sensors are bathroom sensors and 1%-2% of the sensors are bedroom sensors and the water is on?"*

This query focuses on four out of the original thirty features (a reduction of 87%) and aggregates three of the 40 instances into a single query. A variety of measures can be employed to estimate the informativeness of a data point and to induce the rules that will comprise oracle queries. The goal of this approach is to construct compact, easy-to-understand queries that an oracle can easily answer and that label an entire

```
Algorithm GAL(L, U, M)
  // L is the set of labeled data
  // U is the set of unlabeled data
  // M is the proximity matrix

  while stopping criteria is not met
    Train classifier C using L
    Identify x*                          // Select most informative instance
    Identify N_x*                        // Identify nearest neighbors
    Sample {U - N_x*} to get E_x*        // Identify strangers
    Δ = N+_x* ∪ E-_x*                    // Create dataset labeled with 1s and 0s
    Learn rule set R from Δ              // Induce rules distinguishing neighbors from strangers
    c, l = AskOracle(R)                  // Ask oracle to provide activity label for neighbor set rule

    for all x_j ∈ Δ                      // Add each neighbor if the data point is covered by rule

      if x_j is covered by some r_k ∈ R_and c > θ     // and oracle's confidence is sufficient
        Add x_j to L with label l
      end if
    done
  done
return
```

FIGURE 8.6 *GAL generalized active learning algorithm.*

subset of data points. Active learning can also be used to adapt a generalized model of activities to a new setting or individual by asking the user to label only those situations that are not covered completely or accurately by the existing model.

Note that while techniques such as area under the ROC curve, or AUC, are common for evaluating the performance of supervised learning algorithms including activity recognition algorithms, evaluating the performance of active learning needs to also consider the number of data points that need to be labeled by an oracle in order to achieve desired classifier performance. One such measure is the area under the learning curve, or ALC. An algorithm's learning curve plots classifier accuracy on holdout data as a function of the number of labeled training examples that were provided to the algorithm, as shown in Figure 8.7. The goal of an active learning algorithm is to order the points on the x axis such that accuracy grows as quickly as possible. The closer the ALC is to 1, the more effective the active learning algorithm is at identifying the points that allow the classifier to quickly converge on the optimal activity model.

A technique that is similar to active learning in flavor is experience sampling. This methodology interrupts individuals at key moments in time to ask them to stop their routine and make notes of the current experience in real time. The idea has a close equivalent in psychology and clinical psychopharmacology, in which subjects report behavior, symptoms, and effects in close to real time over the course of a study. This can be used to provide real-time manual annotation of sensor data with a smaller amount of disruption than continual noting of daily activities. If machine learning techniques are used to determine the most critical times for sampling then the number of interruptions required to obtain necessary labeled data can be minimized.

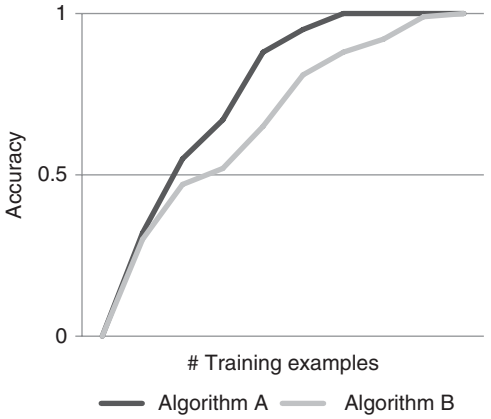


FIGURE 8.7 *Learning curves for two algorithms, A and B. Algorithm A converges to optimal performance more quickly and thus has a larger ALC value than Algorithm B.*

Semi-Supervised Learning In addition to seeking labels for activity sensor sequences, activity learning algorithms should make maximal use of the data that are available, both labeled and unlabeled. Semi-supervised learning algorithms are used in settings where a large amount of unlabeled data is available together with a smaller amount of labeled training data. A common semi-supervised technique is self training. Using self training, a supervised learning algorithm trains the activity classifier with the available labeled data. The learned classifier generates labels for the unlabeled data. Each iteration, a portion of the unlabeled data (typically those that the classifier labels with the highest confidence) and the generated labels are integrated into the training data and used to retrain the classifier.

Semi-supervised methods have found to be particularly valuable when there is far more unlabeled data than labeled data and when points that are close in the space tend to have the same class labels. There is a danger with semi-supervised learning that classification mistakes may be reinforced through the self-training process. In practice, however, semi-supervised learning has been effective at generating stronger activity classifiers that yield improved performance on the available labeled training data.

8.2 TRANSFER LEARNING

Activity learning algorithms require substantial amounts of data. When supervised algorithms are used to model activity classes, the collected data also needs to be labeled with the corresponding activity classes. At the same time, the activity learning algorithms should ideally work under very diverse circumstances. Such diversity is introduced when the algorithms need to be used for activity learning with new users, new sensors, new environments, and activities, or when changes are explicitly or implicitly made to a current setting (e.g., when wearable sensors suffer from on-body displacement). Training data may not always be available for every possible

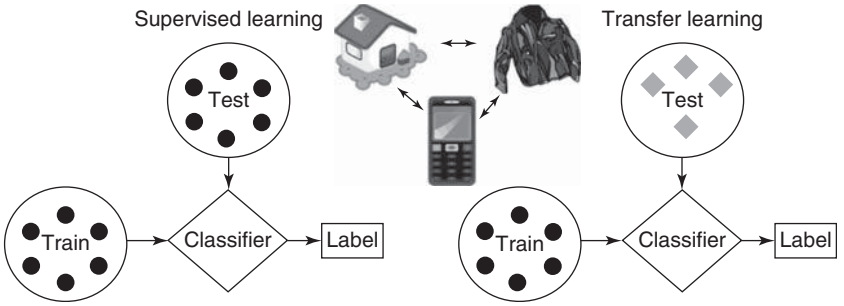


FIGURE 8.8 *Depiction of difference between supervised learning and transfer learning approaches to activity modeling.*

setting, so researchers need to design methods to identify and utilize subtle connections between the settings via techniques such as transfer learning.

The ability to identify deep, subtle connections, what we term transfer learning, is a hallmark of human intelligence. Transfer learning extends what has been learned in one context to new contexts. Transfer learning maps raw data or learned information from one domain, the source, to another domain, the target. Here we defined a domain D as a two-tuple $(X,\ P(X))$. $\mathcal{X}$ is the feature space of D and $P(X)$ is the corresponding data marginal distribution where $X = \{x_1, \ldots, x_n\} \in \mathcal{X}$. A task T is a two-tuple $(Y,\ f(\,))$ for a given domain D. Y is the label space of D and $f(\,)$ is an objective predictive function for D. The function $f(\,)$ is sometimes written as a conditional probability distribution $P(y|x)$. The function $f(\,)$ is not necessarily provided but can be learned from the training data.

Consider the problem of activity recognition using motion sensors. The domain is defined by an n-dimensional feature space, which may consist of n sensor counts (or sensor values) within a given time window and a marginal probability distribution over all possible sensor counts (or values). The task is composed of a label space Y consisting of the set of labels for the activities of interest and a conditional probability distribution consisting of the probability of assigning label $y_i \in Y$ given the observed data point $x \in X$. Figure 8.8 illustrates the difference between using supervised learning for activity modeling in contrast with using transfer learning methods. While the supervised learning algorithm trains and tests a model on the same type of data, the transfer learning algorithm trains the model on a different type of data than the data that are used for testing.

We can now formally define transfer learning using these terms. Given a set of source domains $DS = D_{s_1}, \ldots, D_{s_n}$ where $n > 0$, a target domain D_t, a set of source tasks $TS = T_{s_1}, \ldots, T_{s_n}$ where $T_{s_i} \in TS$ corresponds to $D_{s_i} \in DS$, and a target task T_t which corresponds to D_t, transfer learning helps improve the learning of the target predictive function $f_t(\,)$ in D_t where $D_t \notin DS$ and $T_t \notin TS$. This definition of transfer learning is broad and encompasses a large number of different transfer learning scenarios. The source domains can differ from the target domain by having a different feature space, a different distribution of instances in the feature space, or both. The

source tasks can differ from the target task by having a different label space, a different predictive function for labels in that label space, or both. The source data can differ from the target data by having a different domain, a different task, or both. However, all transfer learning problems rely on the basic assumption that there exists some relationship between the source and target, which allows for the successful transfer of knowledge from the source to the target.

To further illustrate the variety of problems that fall under the scope of transfer-based activity learning, we provide illustrative scenarios that are focused on transfer learning for activity recognition. Not all of these scenarios can be addressed by current transfer learning methods. The first scenario represents a typical transfer learning problem solvable using developed techniques. The second scenario represents a more challenging situation that pushes the boundaries of current transfer learning methods. The third scenario requires a transfer of knowledge across such a large difference between source and target datasets. Current techniques only scratch the surface of what is required to make such a knowledge transfer successful.

Scenario 1 In a home that has been equipped with multiple motion and temperature sensors, an activity recognition algorithm has been trained using months of annotated labels to provide the ground truth for activities that occur in the home. A transfer learning algorithm should be able to reuse the labeled data to perform activity recognition in a new setting. Such transfer will save months of man-hours annotating data for the new home. However, the new home has a different layout as well as a different resident and different sensor locations than the first home.

Scenario 2 An individual with Parkinson's disease visits his neurosurgeon twice a year to get an updated assessment of his gait, tremor, and cognitive health. The medical staff perform some gait measurements and simulated activities in their office space to determine the effectiveness of the prescribed medication, but want to determine if the observed improvement is reflected in the activities the patient performs in his own home. A learning algorithm will need to be able to transfer information between different physical settings, as well as time of day, sensors used, and scope of the activities.

Scenario 3 A researcher is interested in studying the Cooking activity patterns of college students living in university dorms in the United States. The research study has to be conducted using the smart phone of the student as the sensing mechanism. The Cooking activity of these students typically consists of heating up a frozen snack from the refrigerator in the microwave oven. In order to build the machine learning models for recognizing these activity patterns, the researcher has access to Cooking activities for a group of grandmothers living in India. This dataset was collected using smart home environmental sensors embedded in the kitchen and the Cooking activity itself was very elaborate. Thus the learning algorithm is now faced with changes in the data at many layers; namely, differences in the sensing mechanisms, cultural changes, age-related differences, different location settings, and finally differences in the activity labels. This transfer learning from one setting to another diverse setting

is most challenging and requires significant progress in transfer learning domain to even attempt to solve the problem.

These scenarios illustrate different types of transfer that should be possible using machine learning methods for activity recognition and extended to methods for activity prediction and discovery as well. Transfer learning targets differences between the source and target domains that commonly take the form of differences in the feature representation or the label space. When activity differences occur between times, people, sensor devices, or sensor sampling rates, they can also result in differences in the marginal probability distribution of the data points and/or the classification function.

There is a wealth of research in the area of transfer learning. Instead of describing all of the algorithms that can be used for activity transfer learning, we highlight a few algorithms that illustrate the diversity of approaches that can be taken as well as the diversity of types of transfer that may occur.

Instance Transfer Transferring data points, or instances, between the source and target domains is intuitively appealing. Because the data distribution may be different between the source and target domains, the source domain data cannot be reused directly. While parts of the source data may therefore be harmful to the learning process in the target domain, some of the data can be used in combination with a small amount of labeled target domain data. The TrAdaBoost algorithm is an extension of AdaBoost that facilitates this type of instance transfer.

TrAdaBoost iteratively re-weights the source data to reduce the effect of "bad" source data while encouraging greater contribution from the "good" source data. Each iteration, TrAdaBoost trains a base classifier on the weighted source and target data but calculates error only on the target data. Like AdaBoost, TrAdaBoost increases the weights for misclassified target data points and decreases weights for correctly classified target data points. For the source data, however, TrAdaBoost uses a Weighted Majority Algorithm to *decrease* the weights of misclassified data points by a constant factor while preserving the weights of correctly classified source data points. The source data points that are consistently misclassified will receive weights that converge to zero halfway through the iterative process. They will not be used in the final classifier's output because they do not transfer well to the target domain. The TrAdaBoost algorithm is summarized in Figure 8.9.

TrAdaBoost is useful when the primary difference between the source and target datasets is the data distribution. This would occur, for example, when a large amount of labeled wearable or smart phone sensor data is collected for one user but only a small amount of labeled data is available for the second user. On the other hand, the Weighted Majority method of adjusting weights for the source data can face difficulties in practice. If there is a large amount of source data, then many iterations are needed to learn discernible differences between the weights of useful and conflicting source domain instances. At the same time, some source instances may have the weights reduced too quickly. This can happen in situations where the source instances are also quite representative of the target domain but are difficult for the classifier to learn. As a result, these difficult data points quickly cease to influence the final

Algorithm **TrAdaBoost** (S, T, M, WeakLearner)

// S is the source training data consisting of data points $\{x_1, x_2, .., x_L\}$
// and the corresponding labels $\{y_1, y_2, .., y_L\}$
// T is the target training data consisting of data points $\{x_{L+1}, x_{L+2}, .., x_{L+N}\}$
// and the corresponding labels $\{y_{L+1}, y_{L+2}, .., y_{L+N}\}$
// M is the maximum number of boosting iterations
// WeakLearner is the base classisier that outputs a classification model every iteration

$w_i^1 = \frac{1}{L+N} \quad \forall i$ // Initialize weights for the data

$\beta = \frac{1}{1 + \sqrt{\frac{2\ln(L)}{M}}}$

for t = 1, .., M
  h^t = WeakLearner(S, T, w^t) // h^t minimizes the weighted error on complete (source and
  // target) dataset S+T
  // Calculate the weighted error of h^t on the target training data
  $\varepsilon^T = \sum_{i=L+1}^{N} w_i^t \, I(y_i \neq h^t(x_i))$ // I is the indicator function
  if $|\varepsilon^t - 0.5| \leq \theta$ // θ is a predefined threshold
    stop the iterations
  end if
  choose $\alpha^t \in \Re$ // typically $\alpha^t = \frac{1}{2}\ln(\frac{1-\varepsilon^t}{\varepsilon^t})$
  for i = 1, .., L // Update the source weights using Weighted Majority
    if $h^t(x_i) \neq y_i$
      $w_i^{t+1} = \beta \times w_i^t$
    end if
  for i = L+1, .., N // Update the target weights as usual
    $w_i^{t+1} = w_i^t \exp(-\alpha^t \, y_i \, h^t(x_i))$
  done
  $w_i^{t+1} = \frac{w_i^{t+1}}{\sum_{i=1}^{L+N} w_i^{t+1}} \quad \forall i$ // Normalize the weights
done
return LinearModel(h^1, h^2, .., h^M)

FIGURE 8.9 *The TrAdaBoost algorithm to transfer instances from the source to target domains.*

boosted classifier. Alternative weight adjustment schemes can be investigated to help alleviate these difficulties.

8.2.1 Instance and Label Transfer

In most instance transfer algorithms, the data distribution is different between the source and target domains. In response, many of the transfer learning algorithms

employ a type of weighting algorithm where the weight of a data point is reflective of its relevance to the target domain, determined by its similarity to the target data and classifier. TrAdaBoost implicitly determines these weights based on classifier importance. Another approach is to explicitly determine the weights using other data mining techniques. For example, an external source such as web page descriptions can be mined to determine the similarity between two activities. The similarity measures can then be used to transfer labeled data between different activity classes. In this situation not only do the instance data distributions differ between the source and target domains, but the label space also differs between the two domains.

The Cross-domain activity recognition (CDAR) algorithm, learns a similarity function between different activities by mining web documents describing the corresponding activities and computing the similarity of the corresponding text features. The similarities are then used to weight training data for transfer from the source to target activity classes. In this learning situation, each source domain $A^i \in A_S$ contains activities $A^i = \{a^i_1, \ldots, a^i_{L_i}\}$, the target domain contains activity $A_T = \{a_1, \ldots, a_{L_T}\}$, and the two sets of activities do not overlap, $A_S \cap A_T = \emptyset$. While there is a sufficient amount of labeled data for the source domain, no labeled training data are available for the target domain. The two domains do have a common feature space.

External information about known activities is readily available on the web, provided that the activity class has a representative label such as Cook, Shower, or Jog. A search site can be used to retrieve web documents on each activity class in the source and target domains. The web documents are then processed to assemble a bag-of-words description, which is a vector containing the frequency of each word in the page. The vector is then refined to replace each word's frequency value with a corresponding term frequency, inverse document frequency (tf-idf) value, as shown in Equation 8.3. The calculation estimates the value of the word in distinguishing the activity from others because it appears often in the description of one particular activity and not in the description of others. The first component of the equation calculates the term frequency, which is the relative frequency of word w in a retrieved document u. The second component calculates the log of the inverse document frequency, which is a measure of how rare word w is in all of the retrieved documents. An object that is used in Cooking activities, such as "bowl", will have a relatively high tf-idf value for the Cooking activity because it will appear in related web pages but not likely appear in the pages describing Showering or Jogging. While common stop words such as "the" and "you" will appear often on Cooking web pages, they will also appear frequently in the other activity web pages and thus receive a lower tf-idf score.

$$\text{tfidf}_{u,w} = \frac{\text{freq}(u, w)}{\sum_v \text{freq}(u, v)} \times \log \frac{|\{\text{doc}_u\}|}{|\{\text{doc}_u : w \in \text{doc}_u\}|} \tag{8.3}$$

Example 8.3 Table 8.1 provides the frequency, tf, idf, and tfidf values for 15 words selected from WikiHow pages describing the activities Cook, Shower, and Jog, shown in Figure 8.10. Based on the tfidf vectors that are created for each activity, the similarity between each pair of activities using the Kullback–Leibler (KL) measure

TABLE 8.1 Term Values Calculated from Web Pages Describing Cook, Shower, and Jog

Doc	Value	Body	Bowl	Brush	Clean	Hair	Heat	Mile	Muscle
Cook	freq	0	5	1	2	0	35	0	2
	tf	.00	.02	.00	.01	.00	.16	.00	.01
	idf	.18	.48	.18	.18	.48	.18	.48	.00
	tfidf	.0000	.0096	.0000	.0018	.0000	.0288	.0000	.0001
Shower	freq	9	0	7	12	31	0	0	1
	tf	.04	.00	.03	.05	.12	.00	.00	.00
	idf	.18	.48	.18	.18	.48	.18	.48	.00
	tfidf	.0072	.0000	.0054	.0090	.0576	.0000	.0000	.0000
Jog	freq	2	0	0	0	0	1	6	1
	tf	.02	.00	.00	.00	.00	.01	.07	.01
	idf	.18	.48	.18	.18	.48	.18	.48	.00
	tfidf	.0036	.0000	.0000	.0000	.0000	.0036	.0336	.0001

Doc	Value	Run	Stove	Temperature	Tower	Water	Work	You
Cook	freq	0	3	20	1	27	10	119
	tf	.00	.01	.09	.00	.12	.04	.53
	idf	.18	.48	.18	.18	.00	.00	.00
	tfidf	.0000	.0048	.0162	.0000	.0000	.0000	.0000
Shower	freq	4	0	3	6	18	4	156
	tf	.02	.00	.01	.02	.07	.02	.62
	idf	.18	.48	.18	.18	.00	.00	.00
	tfidf	.0036	.0000	.0018	.0036	.0000	.0000	.0000
Jog	freq	25	0	0	0	2	4	42
	tf	.30	.00	.00	.00	.02	.05	.51
	idf	.18	.48	.18	.18	.00	.00	.00
	tfidf	.0540	.0000	.0000	.0000	.0000	.0000	.0000

4 Ways to **Cook** - wikiHow

How to **Cook. Cooking** has come a long way from when our ancestors roasted wild game and local vegetation over an open fire. We've discovered an infinite ...

4 Ways to Take a **Shower** - wikiHow

How to Take a **Shower. Showering** is an activity that millions of people make a part of their daily routine. It's a fast, effective and refreshing way to get clean.

How to Start **Jogging:** 7 Steps (with Pictures) - wikiHow

How to Start **Jogging.** Running can be a great way to get into shape and stay fit. It is by far one of the simplest (and cheapest) sports available, can be done ...

FIGURE 8.10 *Sample WikiHow pages for the activities Cook, Shower, and Jog.*

defined in Equation 6.2. Because KL measures are asymmetric, the sum $KL(a_i, a_j) + KL(a_j, a_i)$ can be computed for each activity pair. The KL similarity measures for Cook, Shower, and Jog based on our sample web pages and selected words are shown in Table 8.2.

A target activity can be learned by reusing instances that were collected for source activities. Each of the instances receives a weight that reflects its similarity to the target activity class and is input to a supervised learning algorithm that is modified to make effective use of the instance weights. Many supervised learning algorithms have versions that use such weights, including naive Bayes, decision trees, and supper vector machine classifiers. The CDAR algorithm is summarized in Figure 8.11. Note that the performance of this type of activity transfer algorithm will be influenced by the number of web pages that are extracted and the choice of similarity measure. It

TABLE 8.2 Kullback–Leibler Similarity Measures for Sample Activity Pairs

(Cook, Shower)	0.0188
(Cook, Jog)	0.0228
(Shower, Jog)	0.0604

Algorithm **CDAR**(S, T)

// S is the set of source domains, each source A^i consists of activities $A^i = \{a_1^i, .., a_{L_i}^i\}$

// and data points $\{x_1^i, .., x_{n_i}^i\}$ with corresponding labels $\{y_1^i, .., y_{n_i}^i\}$

// T is the target domain consisting of activity $A_T = \{a_1, .., a_{L_T}\}$ with no available training data

for each activity $a \in A_S, A_T$ // Create vectors for all activity classes in source and target
  Extract a set of web pages U related to a
  for each web page $u \in U$
    calculate frequencies for u based on web page
  done
  create tf-idf vector D_a
done

for each activity pair (a_i, a_j)
  calculate $sim(a_i, a_j) = KL(D_i, D_j)$ // Use Kullback-Divergence to calculate activity similarity
done

for each activity $a \in A_S$
  for each data point $x_k^a \in a$
    assign $weight(x_k^a) = sim(a, A_T)$
  done
done

learn activity model for T using all weighted labeled training data

return

FIGURE 8.11 *The CDAR algorithm to transfer instances between activity labels.*

will also be influenced by the number of activities in the source domains that have a similarity to or relationship with the target activity.

8.2.2 Feature Transfer with No Co-occurrence Data

In the feature transfer approach, information can be transferred between domains that exhibit distinct characteristics and therefore utilize different feature spaces. This class of transfer learning algorithms is also referred to as *heterogeneous transfer learning*. In a multi-home transfer learning scenario, for example, learned activity models are transferred between a collection of actual physical source sensing spaces and a physical target space. In the context of the multi-home transfer learning algorithm, physical home, and sensor layout information is transferred between smart homes that have been equipped with ambient sensors. However, the approach can be applied to feature transfer for any type of sensor space. Because the sources and target are constrained to work on the same task we simplify the notation and denote the source space as S consisting of N individual sources $S_1, \ldots, S_N$ and the single target space as T. Any physical aspect of the space including the number and type of sensors as well as the residents and their behavioral patterns can be different. The goal of the algorithm is to recognize activities in the target space T using no target labeled data, some available unlabeled target data, and a large collection of labeled source data.

A_S refers to the union of activities from all of the source spaces and A_T refers to the set of target activities. Similarly, S_S and S_T refer to the set of source and target sensors, respectively. In order to map activities from the source to target spaces, we need to find a mapping $F(S_S) = S_T$ that maps the source sensor network to the target sensor network, keeping in mind that the two sets of sensors may have different locations and properties. Once $\mathcal{F}$ is identified, it can be used together with spatial and temporal features to identify the activity mapping function $F(A_S) = A_T$.

Different houses have different floor plans and different sensor layouts, which results in a different feature space. While one home may have a single front door connected to a kitchen which is used for entry and exit, another may have multiple exterior doors such as one connected to a hallway and another connected to a living room, both of which are used for entering and leaving the home. In order to map sensors from the source to target, we can introduce meta-features, which are features that describe properties of the actual features. In the home scenario, each sensor can be described using its location (as is shown in Figure 5.6). Each sensor event that is collected in the source and target domains can be mapped to the meta-feature descriptions. At this point, all of the domains use a common vocabulary and the learned models in the source space can be used to recognize activities occurring in a target space. Because an abstract set of feature descriptions is used, data can be combined from multiple sources to provide even more information to map to the target space.

The multi-home transfer learning algorithm presumes that a mapping is given from each low-level feature to a meta-feature. Alternatively, this mapping can be determined automatically. If labeled target data are available, finding the mapping can be viewed as an optimization task where the goal is to optimize activity recognition performance on the target data given mapped information from the source domain. If unlabeled data are available, activity discovery methods described in Chapter 6 can

be used to identify classes of activities in the data. Mappings from the source to target sensors and mappings from the source to target activity classes can be identified simultaneously as a type of expectation maximization (EM) problem. The process depends on the assumption that similar sensors will be used in similar activities and similar activities will have similar sensors as well as other similar characteristics including frequency, location, and time of day.

8.2.3 Informed Feature Transfer with Co-occurrence Data

When the same data point is found in the source and target domains, albeit with different feature representations, transfer learning problems can be addressed using multi-view learning algorithms. Multi-view learning algorithms represent instances using multiple distinct feature sets or views. The relationship between views can be used to align the feature spaces using methods such as canonical correlation analysis, manifold alignment, or manifold co-regularization. Alternatively, multiple classifiers can be trained for each view and the labels can be propagated between views using Co-Training or Co-EM techniques.

As a heterogeneous transfer technique, multi-view learning applies knowledge learned from a previous task to a new, related task. Multi-view transfer learning algorithms can be categorized as *supervised* or *unsupervised* depending on whether labeled data are available in the source space (view 1) and as *informed* or *uninformed* learning depending on whether labeled data are available in the target space (view 2). Figure 8.12 illustrates the difference between informed supervised transfer learning and uninformed supervised transfer learning for multi-view learning. In this figure, the patterned bands represent labeled activities while the light gray bands represent unlabeled activities.

Co-training and Co-EM represent two informed supervised learning techniques, which can also be viewed as semi-supervised multi-view learning techniques. In Co-Training, a small amount of labeled data in each view is used to train two classifiers, one for each view. These classifiers then assign labels to a subset of the unlabeled data. The newly-labeled instances are added to the set of labeled data and the process

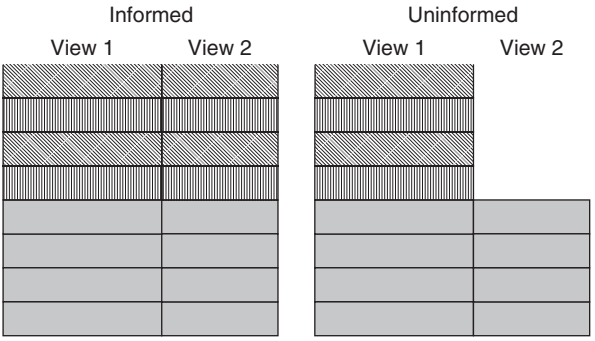


FIGURE 8.12 *Multi-view transfer learning.*

```
Algorithm Co-Training(L, U)
  // L is the set of labeled training examples
  // U is the set of unlabeled training examples
  create a set U' ⊆ U of u examples
  for i = 0 to k
    use L to train classifier h1 for view 1
    use L to train classifier h2 for view 2
    label most confident p positive examples and n negative examples from U' using h1
    label most confident p positive examples and n negative examples from U' using h2
    add the self-labeled examples to L
    replenish U' using 2p + 2n examples from U
  done
return
```

FIGURE 8.13 *The Co-Training algorithm.*

```
Algorithm Co-EM(L, U)
  // L is the set of labeled training examples
  // U is the set of unlabeled training examples
  use L to train classifier h1 for view 1
  create set U1 by using h1 to label U
  while h1 and h1have not converged
    use L∪U1 to train classifier h2 for view 2
    create set U2 by using h2 to label U
    use L∪U2 to train classifier h1 for view 1
    create set U1 by using h1to label U
  done
return
```

FIGURE 8.14 *The Co-EM algorithm.*

is repeated. The Co-Training algorithm is summarized in Figure 8.13. This algorithm is specified for a binary classification task but easily extends to k-ary classification problems by allowing each classifier to label n positive examples for each class instead of labeling p positive examples and n negative examples.

Co-EM is a variant of Co-Training that has been shown to perform better in some situations. Unlike Co-Training, Co-EM labels the entire set of unlabeled data at every iteration. The Co-EM algorithm is summarized in Figure 8.14. Convergence can be measured in the algorithms as the number of labels that change at each iteration. Alternatively, a fixed number of iterations can be specified.

8.2.4 Uninformed Feature Transfer with Co-occurrence Data Using a Teacher–Learner Model

An alternative approach for transferring information across domains with different feature spaces is to allow the source domain to act like a teacher, which finds

opportunities to coach the learner (the target domain) when activities occur that belong to a known activity class. This form of transfer is particularly appealing for activity learning because learners from two different feature spaces will have opportunities to observe activities in parallel as they occur. Each such situation is a learning opportunity. The teacher–student learning paradigm is a type of multi-view learning in which multiple classifiers are viewing the same data points from different perspectives. For activity learning, diversity in perspectives naturally arises when different sensor modalities are used. As Figure 8.5 shows, sensors on the phone, on clothes, and in the building may all collect information as an individual performs an activity, such as Taking Medicine as shown in Figure 2.2, in their home. Because of the different sensor capabilities and data representations, however, each sensor modality will generate a different view of what taking medicine looks like.

In order for a teacher–learner model to be applicable, two requirements must be met. First, an existing classifier (the teacher) must already be trained in the source domain. Second, the teacher must operate simultaneously with a classifier in the target domain (the learner) to provide training for the student. When these requirements are met, the data points in the source and target domains are linked. The i^{th} data point in the source feature space j is the same is the i^{th} data point in the target feature space k, or $x_i^j = x_i^k$. In this learning approach, the target classifier need not initially have any labeled data. Each time the teacher (source) and student (target) simultaneously view a new activity (data point), the teacher informs the student of the activity label that is generated by the source activity classifier. The student represents the same activity using its own feature representation and labels it using the activity class provided by the teacher. The teacher–student algorithm is summarized in Figure 8.15.

As an example, a kitchen cabinet containing a medicine dispenser may be equipped with vibration sensors, accelerometers, RFID sensors, and/or a depth camera. A classifier is trained using these sensors to recognize when medicine is retrieved from the cabinet. This classifier becomes the teacher. Next, several wearable accelerometers are attached to the person who opens and shuts the cabinet. A new classifier, the student, needs to be trained using the wearable sensors. When the individual opens or shuts the cabinet, the teacher labels the activity according to its classification model. This label is given to the student which can then be used combined with the observed sensor values to form a labeled training data point. The transfer occurs in real time without the need to supply other manually-labeled data.

```
Algorithm Teacher-Learner(L, U)

  // L is the set of labeled training examples in view 1
  // U is the set of unlabeled training examples

  Use L to train classifier h1 for view 1
  Create set U1 by using h1 to label U
  Use U1 to train classifier h2 for view 2

return
```

FIGURE 8.15 *The Teacher–Learner algorithm.*

The teacher–student model presents a new perspective on multi-view transfer learning and also introduces new challenges. One challenge is that the accuracy of the student classifier may be bounded by the accuracy of the teacher. An open question is under what conditions the student can outperform the teacher. If the student receives externally-labeled data then it can also bring that information back to the teacher and raise the accuracy of both models, transitioning from a teacher/student relationship to a colleague relationship.

8.2.5 Uninformed Feature Transfer with Co-occurrence Data Using Feature Space Alignment

The teacher–student model is valuable when the teacher can operate in a side-by-side manner with the student learner. When such learning environments are not available, heterogeneous transfer learning can still be accomplished by mapping both the source and target domains to a common, or at least highly similar, feature space. One method to generate this mapping is to apply a dimensionality reduction technique to both spaces, then align the reduced spaces through manipulations that include scaling, translating, and rotating the spaces.

We note that the dimensionality of both the source and target domains can be high and may differ from each other. In some cases, however, the domains actually exhibit a limited number of degrees of freedom because they have a low intrinsic dimensionality. As we discussed in Chapter 4, principal component analysis can be used to project the original data onto a lower-dimensional feature space. Viewing each lower-dimensional feature space as a shape, a technique known as Procrustes analysis is used to scale, rotate, and translate the two shapes into optimal alignment. Once they are optimally aligned, data that are available in the source domain can be mapped onto the lower-dimensional translated space and provided as input to a supervised learning algorithm. New data in the target domain can also be mapped onto the same space and input to the learned activity classifier in order to generate a corresponding activity label.

Manifold Alignment has been proposed as a technique for transferring knowledge between two different views without requiring any labeled data. The algorithm assumes that the data from both views share a common latent manifold, which exists in a lower-dimensional subspace. The basic idea is that the two feature spaces can be projected onto a lower-dimensional subspace and the pairing between views can then be used to optimally align the subspace projections onto the latent manifold. A classifier can then be trained using projected data from the source view and tested on projected data from the target view. The details are shown in Figure 8.16.

8.3 MULTI-LABEL LEARNING

In a traditional multi-class activity learning setting, a single data point is associated with only one activity label. However, in situations where an individual performs concurrent activities or multiple residents perform different activities at the same time, a

```
Algorithm ManifoldAlign(L, U1, U2)
  // L is the set of labeled training examples in view 1
  // U1 and U2 are paired unlabeled training examples, one for each view
  X, EV = PCA(U1)                  // X is low-dimensional embedding of U1
  Y = PCA(U2)                      // Y is low-dimensional embedding of U2

  // apply Procrustes analysis to find optimal alignment of X and Y
  // translate projections X, XU, Y, YU so they are centered about the origin
  U∑V^T = SVD(Y^T X)               // compute the singular value decomposition of Y^T X
  Q = UV^T
  k = trace(Σ)/trace(Y^T Y)
  Y' = kYQ          // optimal mapping that minimizes ||X-Y'||F where ||.||F is Frobenius norm
  Project L onto low-dimensional embedding using EV
  Train classifier on projected L
  Test classifier on Y'
return
```

FIGURE 8.16 *The Manifold Alignment algorithm.*

sequence of sensor events may correspond to multiple activities. In these cases, activity learning involves modeling a data points as instances of multiple classes. When multiple class values are assigned to each data point, the problem is referred to as *multi-label learning*. For activity recognition, the problem now is to map a sequence of sensor events, $x = < e_1\ e_2..e_n >$, onto a subset of values from the complete set of predefined activity labels, $V \subseteq A$.

Example 8.4 Figure 8.17 illustrates the distribution of multi-label activity sensor data instances from an actual two-resident smart home. As the figure shows, many data instances are associated with two or more activity labels. Table 8.3 lists some combinations of activities in this dataset. The combinations commonly occur in residential settings. For example, there are 120 data instances in which one resident is working in the bedroom, while the second resident is sleeping.

The multi-label problem can also arise due to limitations in the selection of features that are extracted from a sliding window of sensor events. In some cases where the feature representation aggregates or abstracts detailed sensor event information, two sets of identical features may in fact correspond to different activity labels. One might view multi-class learning as a special case of multi-label learning, where even though there are multiple classes, each data point is mapped to only one label. The goal of a multi-label learner is to learn all the label associations of the training dataset and accurately predict all the labels associated with a test data point. Because of the increasingly prevalent need for multiple-label learning in areas such as document classification, music information retrieval, and object recognition from images, this learning setting has witnessed the development of many new algorithms in the past decade.

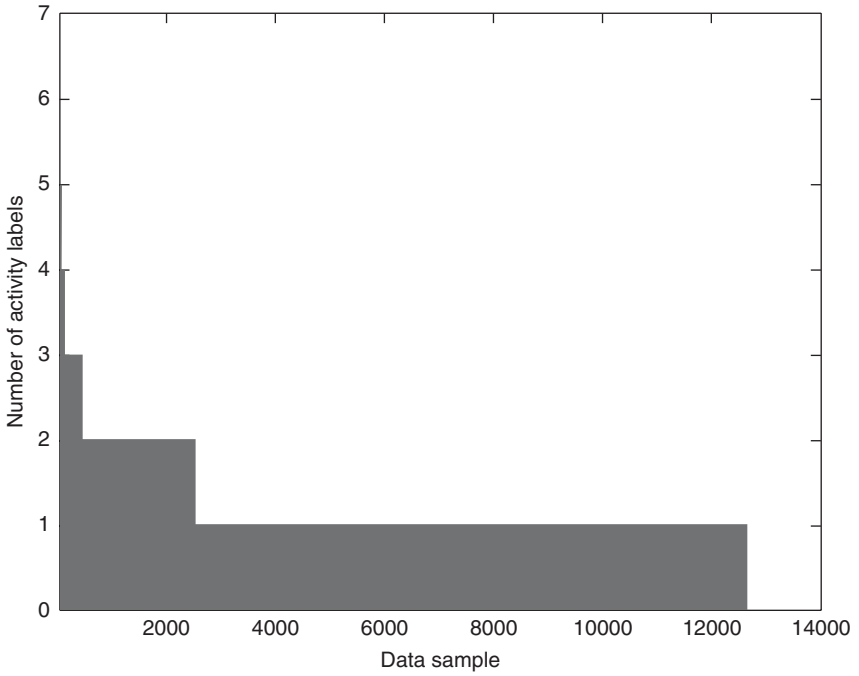


FIGURE 8.17 *The number of distinct activity labels associated with data points sampled from a two-resident smart home.*

TABLE 8.3 Example of Multi-Label Instances in a Multi-Resident Activity Learning Dataset

Combination of Activity Labels	Number of Instances
R1 Sleep, Work Bedroom2	120
Cook, Work Living Room	41
Personal Hygiene, Bathe, Work Bedroom2	13
Personal Hygiene, Wash Dishes	13
R1 Sleep, Personal Hygiene, R2 Sleep in Bed, Work Bedroom2	10
R1 Sleep, Personal Hygiene, Bathe, Cook, Eat, Bed Toilet Transition, R2 Sleep	1

We group these algorithms into two classes based on the principle idea driving the algorithm: *problem transformation* and *label dependency exploitation.* The focus of *problem transformation* approaches is to transform the multi-label learning problem into well-established learning strategies such as multiple binary classifications or multi-class learning. As the name suggests, *label dependency exploitation* approaches exploit the correlations between the labels and data to derive new "label" vectors that are then learned using supervised learning methods. We denote the number of training samples, the dimensionality of the data and the number of labels as N, D, and

Q, respectively. We refer to the i^{th} data point as $x_i \in R^D$ and the corresponding label encoding as $y_i \in \{0,1\}^D$. In this discussion, $y_{iq} = 1$ if $x_i \in q^{\text{th}}$ class and is 0 otherwise. Let $X = [x_1, .., x_N] \in R^D \times N$ represent the data matrix and $Y = [y_1, .., y_N] \in \{0,1\}^D \times N$ represent the corresponding label matrix.

8.3.1 Problem Transformation

A simple problem transformation is to force the multi-label learning problem into a multi-class learning problem by considering only one label (chosen at random) for each multi-label instance while discarding the other labels. Another approach is to completely discard all of the multi-label instances from the dataset. These two approaches alter the multi-label problem to a great extent, resulting in significant information loss. Discarding the multiple labels associated with an instance or instances is not practical for learning problems which only have a small number of available training samples. Another approach, termed *label power set* (*LP*), creates a new label for every unique combination of multiple labels. This results in a large number of labels with very few examples representing a single label. When the initial number of labels Q is itself very high, this approach results in a multi-class learning problem with an extremely skewed class distribution, adding complexity to the learner.

A popular problem transformation method that neither discards the instances and labels nor generates new labels is the *binary relevance* approach. The original dataset is transformed into Q datasets, where the q^{th} dataset contains all of the examples from the original dataset and is trained to recognize only one of the activity labels. Each data point in the q^{th} dataset is assigned a label 1 if the set of original data point labels contain activity q and 0 otherwise. A binary classification model is then learned using each of the Q datasets. This is similar to a one-versus-all multi-class learning setting. This learning process results in Q classification models, one for each activity. The activity labels associated with the classification models that return a positive output for a test data point are the predicted labels for the test data point. This algorithm is summarized in Figure 8.18.

Example 8.5 To illustrate this algorithm, consider the multi-label activity learning dataset presented in Table 8.4. This dataset is decomposed into four datasets as defined by the four columns. Each column defines one of the activities that may be associated with the data instance. In order to learn the binary classification model for the activity R1 Sleep (referring to Resident 1 is sleeping), instances 1, 2, 3, and 5 are considered positive examples, while instances 4 and 6 constitute the negative examples. To learn the Bed Toilet Transition model, only instance 4 acts as a positive example, while the rest of the instances are identified as the negative examples. This process is repeated for activities R2 Work and R2 Sleep.

While this is a simple approach, the number of binary models to be learned increases as the number of labels increase. Furthermore, transforming the multi-label

Algorithm **BinaryRelevance**(X, Y)

// X is the set of data points $\{x_1, .., x_n\}$, where each $x_i \in R^D$
// Y is the set of activity labels for each data point $\{y_1, .., y_n\}$, where each $y_i \in \{0,1\}^Q$

for q = 1 to Q
  if $y_{iq} = 1$
    $y_i' = 1$
  else $y_i' = 0$
  $D_q = [X, Y']$
  Learn model H_q using any supervised classification algorithm
done
for each data point x
  if $H_q(x) = 1$
    q is output activity label for x
return

FIGURE 8.18 *The binary relevance multi-label learning algorithm.*

TABLE 8.4 Binary Relevance Approach for Multi-Label Learning

Data instance	R1 Sleep	Bed Toilet Transition	R2 Work	R2 Sleep
1	X			
2	X		X	
3	X			X
4		X		X
5	X			
6				X

problem into binary classification problems in this fashion can result in class imbalance in the dataset, posing an additional problem for the learner.

8.3.2 Label Dependency Exploitation

Label dependency exploitation methods transform the data and/or label space (generally into a lower dimensional space) by exploiting the sparsity and dependency between the labels, thus reducing the computational complexity of the learning process. The primary difference between approaches in this category and problem transformation methods is the different label space that is used for learning. Problem transformation approaches retain the original label space, or at best consider a subset of the space while training the classifiers. Label dependency exploitation methods transform the data and/or label space by taking advantage of the label sparsity and inter-dependency. The learning approaches in this category consist of encoding and decoding phases. During the encoding phase, the original label space is transformed and traditional supervised learning approaches are employed to learn the mapping

between the data and transformed label space. In the decoding phase, the labels for a test data point are decoded from the output of the learners.

The *principal label space transformation* (*PLST*) is one of the first approaches that transforms the label space by exploiting the correlations between the labels. *PLST* captures the key correlations between labels using a linear subspace, or *flat*, within the original high-dimensional space. The method only involves a simple linear encoding of the labels and linear decoding of the predictions, both easily computed from a singular value decomposition of label matrix, Y', representing the label vectors. By capturing the key correlations, *PLST* is capable of reducing the number of binary classifiers that have to be learned without compromising the prediction accuracy.

PLST first shifts each label vector y to $z = y - \overline{y}$, where $\overline{y}$ is the estimated mean of the set of training label vectors. PLST then linearly maps z to the code vector t using the transformation matrix $V : t = Vz$. PLST only considers a matrix V with orthogonal rows. The algorithm uses a regression function, r, to map the input data points x to the label code vector t. The goal of PLST is to find a transformation matrix V that minimizes the prediction error as well as the encoding error, defined by Equation 8.4.

$$\text{Error} = ||r(X) - ZV^T||^2 + ||Z - ZV^T V||^2 \tag{8.4}$$

In Equation 8.4, $r(X)$ contains $r(x_i)^T$ as rows and Z contains z_i^T as rows. The matrix ZV^T contains the code vector t_i^T as rows. The first term in the error summation represents the prediction error of the regression model that maps input data to the code vectors. The second term corresponds to the encoding error. The prediction labels of a test data point are obtained by performing round-based decoding on the output of the regression function, as expressed in Equation 8.5.

$$-\widetilde{y} = \text{round}(V^T r(x) + \overline{y}) \tag{8.5}$$

PLST determines the optimal $K \times Q$ linear transformation matrix, V, by applying singular value decomposition to Z and identifying the K right singular vectors, v_k, that correspond to the top K largest singular values. These K singular vectors corresponding to the top K eigenvalues that represent the principal directions of the label vector space with maximum variation. The PLST algorithm is summarized in Figure 8.19.

Example 8.6 Lets compute the label code vectors for the sample multi-label dataset provided in Table 8.4. The labels for the 6 data points ($\{y_i\}_{i=1}^6$) can be constructed using binary notation, resulting in $\{(1,0,0,0), (1,0,1,0), (1,0,0,1), (0,1,0,1), (1,0,0,0), (0,0,0,1)\}$. The mean of this vector of labels is (0.67, 0.17, 0.17, 0.50). The vectors ($\{z_i\}_{i=1}^6$) are then calculated as ($\{0.33, -0.17, -0.17, -0.50), (0.33, -0.17, 0.83, -0.50), (0.33, -0.17, -0.17, 0.50), (-0.67, 0.83, -0.17, 0.50), (0.33, -0.17, -0.17, -0.50), (-0.67, -0.17, -0.17, 0.50)\}$. The eigenvalues (in decreasing order) obtained from the singular value decomposition of $Z^T Z$ are $\{2.95, 0.76, 0.50, 0.30)\}$. Setting $K = 2$, the transformation matrix obtained by selecting the two rows of B corresponding to the top two eigenvalues is $\text{V} = \{(-0.61, 0.33, 0.00, 0.72), (0.37, -0.49, -0.58, 0.65)\}^{\text{T}}$. The transformed label vector ($\{t_i\}_{i=1}^6$) that results from

Algorithm **PrincipalLabelSpaceTransform**(X, Y)

// X is the set of data points $\{x_1, .., x_n\}$, where each $x_i \in R^D$
// Y is the set of activity labels for each data point $\{y_1, .., y_n\}$, where each $y_i \in \{0,1\}^Q$

$\overline{y} = mean(Y)$

for i = 1 to N

$z_i = y_i - \overline{y}$

done

$Z = [z_1,...,z_N]^T$

// Perform singular value decomposition. Let Σ represent the matrix with the eigenvalues placed
// in decreasing value along its diagonal.
// Let V contain the top K rows of B.

$A\Sigma B = SVD(Z^T Z)$

transform $\{(x_i, y_i)\}_{i=1}^N$ to $\{(x_i, t_i)\}_{i=1}^N$ where $t_i = Vz_i$

learn the regression model $r : X \rightarrow \{t_i\}_{i=1}^N$

for each data point x_i

label vector $\widetilde{y} = round(V^T r(x) + \overline{y})$

return

FIGURE 8.19 *The principal label space transform multi-label learning algorithm.*

using the aforementioned transformation matrix is $\{(-0.62, 0.03), (-0.62, -0.55), (0.10, 0.57), (1.04, -0.29), (0.71, 0.20)\}$. This vector is used as the output for training the regression function.

The primary drawback of using PLST is that it only captures correlations between the labels. The algorithm can be further improved by considering correlations between data and labels as well. The main advantage of this approach for multi-label learning is its efficient decoding scheme that only uses a round-off function.

Another approach that exploits the dependency between labels and data to encode the label vector is *multi-label output codes using canonical correlation analysis* (*MOCCA*). In this approach, label dependency is characterized as the most predictable directions in the label space. These dependencies are extracted as the canonical output variates using canonical correlation analysis. The canonical variates represent the directions in which there is maximum correlation between the labels and the data. The final label code vector encodes these most-predictable variates as well as the original labels. Predictions for the codeword define a graphical model of labels as Bernoulli potentials (from binary classifiers on the original labels) and Gaussian potentials (from regression on the canonical variates). The decoding is performed using a variational inference technique known as mean-field approximation, which offers a tractable mechanism for estimating the output labels.

Canonical correlation analysis (CCA) is a tool that is often used to model the correlations between two sets of multidimensional variables. CCA's goal is to discover a lower-dimensional space in which the two sets of variables are maximally correlated. This is a useful approach for multi-label classification where one view is derived from the data and the other view is derived from the class labels. Specifically, CCA finds a pair of projection directions, $w_x \in R^D$ and $w_y \in R^Q$, such that the correlation between the pair of projected variables, $w_x^T x_i$ and $w_y^T y_i$, is maximized. This maximization problem can be expressed using Equation 8.5.

$$\max_{w_x \in R^D \text{ and } w_y \in R^Q} w_x^T X Y^T w_y \tag{8.5}$$

subject to the constraints

$$w_x^T X X^T w_x = 1$$

$$w_y^T Y Y^T w_y = 1$$

By formulating the Lagrangian of the aforementioned convex optimization problem, the Kaursh–Kunh–Tucker conditions lead to the eigenproblems on w_x and w_y given in Equation 8.6.

$$X^T Y (Y^T Y)^{-1} Y^T X w_x = \lambda X^T X w_x \tag{8.6}$$

$$Y^T X (X^T X)^{-1} X^T Y w_y = \lambda Y^T Y w_y$$

This generalized eigenproblem results in multiple pairs of eigenvectors, $\{(w_x^k, w_y^k)\}_{k=1}^K$, that successively maximize the correlation between the resulting canonical input and output variates. The canonical output variates, $\{(w_x^k)^T Y\}_{k=1}^K$, are referred to as the *most predictable variates* and are used to encode the label vector. In MOCCA, the label code vector is defined as shown in Equation 8.7.

$$t_i = (y_{i1}, y_{i2}, \ldots, y_{iQ}, (w_x^1)^T y_i, (w_x^2)^T y_i, \ldots, (w_x^K)^T y_i) \tag{8.7}$$

Note that the transformed label code contains the original label vector as a subcomponent. MOCCA includes this component based on a principle from information theory that the message contains high information about itself. MOCCA then learns Q binary classifiers to predict the subcomponent of the transformed label code corresponding to the original label vector. MOCCA also learns K regressors for predicting the canonical output variates.

For each instance x, the classification and regression models are used to obtain a predictive distribution on the codeword, t. Each classifier predicts the Bernoulli distribution $\varphi_q(y_q) = P_q^{y_q}(1 - P_q)^{1-y_q}$, where P_q is the probability that the test sample is marked with label q. Similarly each regression model, $r^k(x)$, predicts a Gaussian

distribution, $\psi_k(y)$, for the canonical output variate $w_y^{kT} y$, as given in Equation 8.8 where α^k is the variance that can estimated using a cross validation approach.

$$\psi_k(y) \propto \exp - \frac{(w_y^{kT} y - r^k(x))^2}{2\sigma^{k2}} \tag{8.8}$$

To perform the decoding step, a mean field approximation to $P(\widetilde{y})$ is applied. In this process, $P(\widetilde{y})$ is fully factorized as $P(\widetilde{y}) = \prod_{q=1}^{Q} \widetilde{P}_q(\widetilde{y}_q)$ to allow tractable inference over labels. The Kullback–Leibler divergence between $P(\widetilde{y})$ and its approximation $\widetilde{P}(\widetilde{y})$ is minimized. This results in the fixed point formulation shown in Equation 8.9 for updating the distribution.

$$\widetilde{P}_q(y_q) \leftarrow \frac{1}{z_j} \exp \left\{ \lambda \log \varphi_q \left(y_q \right) + \sum_{k=1}^{K} E_{y \approx \widetilde{P}} \left[\frac{\log \psi_k (y)}{y_q} \right] \right\} \tag{8.9}$$

In Equation 8.9, Z_j is a normalization constant. The factors $\widetilde{P}_q$ are directly considered as the probability with which a test samples x can be marked with the class label q. The MOCCA algorithm is summarized in Figure 8.20.

MOCCA is able to take advantage of dependencies between the labels and data to create predictable code vectors. However, the number of models to be learned

Algorithm **CCAOutputCodes**(X, Y)

// X is the set of data points $\{x_1, .., x_n\}$, where each $x_i \in R^D$
// Y is the set of activity labels for each data point $\{y_1, .., y_n\}$, where each $y_i \in \{0,1\}^Q$

// Encoding phase

$\{(w_x^k, w_y^k)\}_{k=1}^{K} = CCA(\{x_i, y_i)_{i=1}^{N})$

// Compute the transformed label code vector

for i = 1 to N

$t_i = (y_{i1}, y_{i2}, .., y_{iQ}, (w_x^1)^T y_i, (w_x^2)^T y_i, .., (w_x^K)^T y_i)$

done

for q = 1 to Q

learn the binary classifier using $\{(x_i, y_{iq})\}_{i=1}^{N}$

done

for k = 1 to K

learn the regressors (r) using $\{(x_i, (w_x^k)^T y_i)\}_{i=1}^{N}$

done

// Decoding phase

$\{\widetilde{P}_q(y_q)\}_{q=1}^{Q}$ = MeanFieldApproximation(n,$\{P_q\}_{q=1}^{Q}$,$\{r^k(x)$,$\sigma^k\}_{i=1}^{K}$)

use $\{\widetilde{P}_q(y_q)\}_{q=1}^{Q}$ as the predictive distribution over y_1,..,y_Q

return

FIGURE 8.20 *The multi-label output code algorithm using canonical correlation analysis.*

increases beyond the original number of labels. Another drawback of this approach is the computational complexity of the decoding phase. This can be reduced to some extent by imposing additional constraints on the transformation matrix and using a round-based decoding scheme similar to the one employed in PLST.

8.3.3 Evaluating the Performance of Multi-Label Learning Algorithms

Multi-label learning requires different metrics than those used in traditional single label classification. Let $\hat{y}$ represent the vector of predicted labels for a test data point, x. We present here some of the commonly employed metrics for evaluating the performance of multi-label learning algorithms.

Hamming Loss Hamming loss evaluates the number of instance-label pairs that have been misclassified. Hamming loss is derived from information theoretic measure—Hamming distance between two strings of equal length that computes the number of positions at which the corresponding symbols are different. Instead of comparing two strings, Hamming loss performs an XOR operation on the predicted label and true label vectors. The average Hamming loss on the test dataset is given by the formula in Equation 8.10. Here $||.||_1$ refers to the l_1 norm and $\oplus$ refers to the XOR operation. The smaller the value, the better the performance. This is one of the most prevalent multi-label performance criteria.

$$HL(X) = \frac{1}{N_T}\sum_{i=1}^{N_T}\frac{||\hat{y}_i \oplus y_i||_1}{Q} \tag{8.10}$$

Subset Accuracy Subset accuracy measures the accuracy of the predictions on all the labels. A prediction is considered to be correct only if all the labels for the data point are predicted accurately. Subset accuracy can be computed as shown in Equation 8.11.

$$SA(X) = \frac{1}{N_T}\sum_{i=1}^{N_T}\delta(y_i, \hat{y}_i) \tag{8.11}$$

In this equation, $\delta(y_i, \hat{y}_i)$ is 1 if $y_{iq} = \hat{y}_{iq}\ \forall q$. In a sense, this is a more pessimistic measure compared to Hamming loss. Larger subset accuracy values are better as it indicates that the model is more successful in accurately predicting all the labels associated with a data point.

MacroF1 MacroF1 averages the F1 score on the predictions of different labels. The F1 score for a label (F_q for the q^{th} label) can be computed using Equation 5.18 provided in Chapter 5. The formula for computing MacroF1 measure is shown in Equation 8.12.

$$F_{MA} = \frac{1}{Q}\sum_{q=1}^{Q}F_q \tag{8.12}$$

This can be further simplified in the current context as the label vector $y_i \in \{0,1\}^Q$, resulting in Equation 8.13. Larger values of the MacroF1 measure indicate better performance of the model.

$$F_{MA} = \frac{1}{Q}\sum_{q=1}^{Q} \frac{2\sum_{i=1}^{N_T} y_{iq}\widehat{y}_{iq}}{\sum_{i=1}^{N_T} y_{iq} + \sum_{i=1}^{N_T} \widehat{y}_{iq}} \tag{8.13}$$

MicroF1 MicroF1 calculates the F1 measure on the predictions of different labels as a whole.

$$F_{MI} = \frac{2\sum_{i=1}^{N_T} ||y_i \cap \widehat{y}_i||_1}{\sum_{i=1}^{N_T} ||y_i||_1 + \sum_{i=1}^{N_T} ||\widehat{y}_i||_1} \tag{8.14}$$

We compute the true positives and false negatives for different labels as a whole and then compute the overall F1 measure according to Equation 5.18. As is the case with the MacroF1 measure, larger values of the MicroF1 measure indicate better performance of the model.

8.4 ACTIVITY LEARNING FOR MULTIPLE INDIVIDUALS

Activity learning is certainly a difficult task when it is "in the wild". The complexity of the task increases exponentially, however, when it is performed for multiple users concurrently. This is because if each activity is modeled separately and each combination of behaviors from multiple individuals is treated as a separate activity class, then the number of activities to learn grows exponentially with the number of users. Therefore, methods must be devised to handle multiple-individual situations. These range from explicitly detecting group activity, to splitting sensor information into multiple streams so that each individual's behavior can be analyzed separately.

8.4.1 Learning Group Activities

One approach to learning activity models in the presence of multiple activity performers is to explicitly detect and model multiple-person situations. As stated earlier, the number of situations can become too large if every possible scenario is modeled. However, we may choose to identify multiple-person activities because of their value in understanding and tracking social interactions and group behaviors.

Note that the approaches to learning group activities are different depending on the sensor devices that are used. In the case of wearable sensors, each individual generates a distinct stream of sensor data. There may not be an issue with identifying the person associated with the data stream, but the streams need to be fused and analyzed as a whole to identify the group behavior. On the other hand, if one stream of data is generated for the entire group (as is the case if video or environmental sensors are

used), then the data are already fused and either needs to be handled as a whole or separated into multiple streams before analyzing the group activity.

Learning models of group activities is an interesting endeavor because there are different dynamics in group activities than in single-person activities. In addition, models of group activities may need to abstract over the specific number of individuals that are in the group. Consider as examples the Marching Band and Huddle activities found in the second row of Figure 2.2. These activities have distinctive behavioral patterns that can be easily recognized, yet do not encompass only a fixed number of participants. On the other hand, some group activities, such as Machine Band and Business Meeting, do have particular roles that users play in the activity. In a football game, an easily recognizable role is "kicker". In a business meeting, a distinctive role is "moderator". One approach to learning models for such activities, then, is to introduce new features to the model that specify these roles. Recognizing the occurrence of these roles is a key for discovering, recognizing, predicting, and tracking the group activity.

The group activity recognition (GAR) algorithm, is designed to generalize across varying numbers of individuals in the group and varying numbers of roles. The key to the algorithm is to first detect group activity roles and then include these roles in the feature vector that is used to learn the group activity model. GAR is designed initially for cases where each individual is associated with a separate data stream, which facilitates the process of identifying individual roles within the group. Variable f_t^i thus refers to the feature vector extracted for sensor data generated by individual i at time t.

In the first step of GAR, group roles are detected by clustering sensor data collected from multiple instances of the activity class. The hypothesis is that feature vectors for individuals playing similar roles will themselves be similar. A mixture of Gaussians is then used to model the set of roles that correspond to the activity class. The Gaussian mixture models (GMM) is constructed by extracting feature vectors for each user's sensor data during the activity, clustering the vectors, and estimating the GMM parameters using expectation maximization.

The Gaussian distributions can be used to identify the probability that an individual i plays role R_m^n at time t using Equation 8.15, where $m \in M$ roles and $n \in N$ activities. In this equation, the term $P(f_t^i|R_m^n)$ can be estimated using the Gaussian density calculation shown in Equation 4.16.

$$P(R_m^n|f_t^i) = \frac{P(f_t^i|R_m^n)P(R_m^n)}{P(f_t^i)} = \frac{P(f_t^i|R_m^n)P(R_m^n)}{\sum_{x,y} P(f_t^i|R_y^x)} \tag{8.15}$$

In GAR's second step, a group activity model is trained on labeled data, where the data are represented as the set of role feature vectors that were identified. Two additional features are also captured. The first indicates whether any user exists that plays a specific role R_m^n and is calculated as $\max_i(P(f_t^i|R_m^n))$. If no individual is likely play this role then the value approaches 0, otherwise the value approaches 1. The second feature represents the contribution of individuals in the group playing role R_m^n to the total number of individuals involved in the group activity. The values are

computed as $\frac{\sum_i P(f_t^i | R_m^n)}{I}$, where I indicates the number of individuals in the group. In a business meeting with eight people, for example, there is typically only one moderator at a time. While the person moderating the meeting may rotate among the individuals, the ratio will remain close to 1/8 throughout the activity. Once the feature vector is extracted then the supervised learning algorithms described in Chapter 4 can be used to model the group activities themselves.

One limitation of this approach is that GAR assumes all of the individuals that are associated with a group activity are known. In order to use GAR, the individuals comprising such a group need to be identified. To do this, we can make an assumption that the group members move together as a compact, coherent unit (or cluster) for a period of time. This type of unit is referred to as a pedestrian *flock*.

The flock detection (FD) algorithm, , makes use of sensors that are associated with each individual. Similarity measures are calculated for each pair of sensor streams and clustering is applied to the resulting matrix of similarity measures to identify flocks, or spatio-temporal-based clusters of individuals. Three specific factors that can be included in the pair-wise similarity computation are overlap in movement behavior, windowed acceleration cross-correlation, windowed relative heading cross correlation, and time since last turn.

Overlap in Movement Behavior (OMB) This similarity measure determines the amount of overlap that occurs in the activity labels for two individuals, i and j. This is motivated by the hypothesis that individuals in flocks will mimic each other's behavior or perform the same behavior simultaneously.

$$OMB(i,j) = \frac{\sum_{t=0}^{T} A_{i_t} = A_{j_t}}{T} \tag{8.16}$$

Windowed Acceleration Cross Correlation (WACC) This measure is based on the hypothesis that similar movement patterns result in correlation of the related acceleration values. While the cross-correlation measures introduced in Chapter 3 could be used to measure this, noise may be introduced because differences may exist in sensor orientation and individual trajectories. An alternative, then, is to consider the variance V of the signal magnitude. In addition, the trajectory may be shifted in time between flock members, so the correlation is computed for a set of possible lag values, l. The definition of lag variables is described in Chapter 7 for use in activity prediction.

$$\text{WACC}(i,j) = \max_{l \in [-1,1]}(\text{corr}(V_i, V_j)) \tag{8.17}$$

Windowed Relative Heading Cross Correlation (WHCC) Like the WACC measure, WHCC determines the correlation between two sensor devices, this time based on the sensed orientation, or heading, H, for each individual.

$$\text{WHCC}(i,j) = \max_{l \in [-1,1]}(\text{corr}(H_i, H_j)) \tag{8.18}$$

Time Since Last Turn (TSLT) TSLT considers the time a navigational turn was made, measured as a significant change in heading. A value K is computed for each time point t, which is 0 when a turn is detected, otherwise it copies the previous value plus a small adjustment factor (1/50 seconds has been used in practice).

$$\text{TSLT}(i,j) = \sum_{t=0}^{T} \|K_{i_t} - K_{j_t}\| \tag{8.19}$$

While these techniques are useful when sensor streams are available for each user, they require significant adaptation for situations when video or ambient sensors are utilized. However, techniques can be introduced that operate specifically in single-sensor-stream situations. Once again, features are added to the activity model that provide explicit indications of the presence of multiple individuals and whether they are interacting or acting together in group (or flock) behavior. These features may include area dwell time and area transitions.

Area Dwell Time (ADT) In the same way that accelerometer cross-correlation combined with location information will provide an indication of individuals moving together (as a flock), so time spent in spatial areas within an environment can indicate group activities. If individuals are interacting face-to-face then they will be in the same region of the environment. Dwell time can be calculated separately for each spatial region within the environment. In many cases, different types of interactions and group activities can be distinguished based on the spatial areas in which they occur. For example, a group may cook a meal together in the kitchen or watch television together in the living room. The number of sensor firings in each room (or sensor values) can be analyzed, as described in Chapter 3, to also give an indication of the number of individuals that are in the group. This is based on the hypothesis that the number of sensor events (or the sensor values) will increase proportionately with the number of individuals that are in the space.

Area Transitions (AT) AT is computed as the number of transitions that occur for each pair of spatial regions within an environment. Just as the TSLT computation provides a measure of how "in sync" the navigation patterns are for multiple individuals with separate sensor devices, so AT also provides an indication of the extent to which multiple people are interacting within a space. Typically, if individuals are in multiple spaces within an environment, the AT measure will be higher. Thus, AT provides an indication of the number of people within an environment as well as the extent to which they are exhibiting flock, or group, behavior.

8.4.2 Train on One/Test on Multiple

A second approach we will consider for activity learning in the presence of multiple users is a train on one, test on multiple (TOTM) approach. In these approaches, we assume there are multiple training datasets and each separate dataset corresponds to one individual user performing their part of a group activity. As a result, the approach

is most amenable in situations for which there is a separate sensor data source for each individual person. Given these individual training sequences, the goal of TOTM is to learn an activity model that can map each observation from any user to the correct activity label.

Just as multiple types of probabilistic graphs have been successful for single-user activity recognition, so too these graphs can be enhanced to handle the multiuser case. The first model that we explore for learning activities from individual streams is a coupled hidden Markov model, or CHMM. Figure 4.3 provides an example hidden Markov model HMM for representing a sequence of activities $\{y_1, \ldots, y_N\}$ as hidden nodes in the probabilistic graph and the corresponding sensor events $\{x_1, \ldots, x_N\}$ as observable nodes. In the CHMM, shown in Figure 8.21, there is one chain, or sequence, of activity labels and one sequence of observables for each user. Each user's HMM represents the sequence of steps that is performed by that user and is connected to the other users' HMMs by connecting a hidden state at time t from one HMM to a hidden state at time $t + 1$ for any other user's HMM, as well as to its own HMM next hidden state.

$$P(X, Y) = \prod_{t=1}^{T} P(y_t^a | y_{t-1}^a) P(y_t^b | y_{t-1}^b) P(y_t^a | y_{t-1}^b) P(y_t^b | y_{t-1}^a) P(x_t^a | y_t^a) P(x_t^b | y_t^b) \quad (8.20)$$

The Viterbi algorithm summarized in Figure 4.5 can be adapted to generate a sequence of group activity labels from a sequence of observed sensor events. The

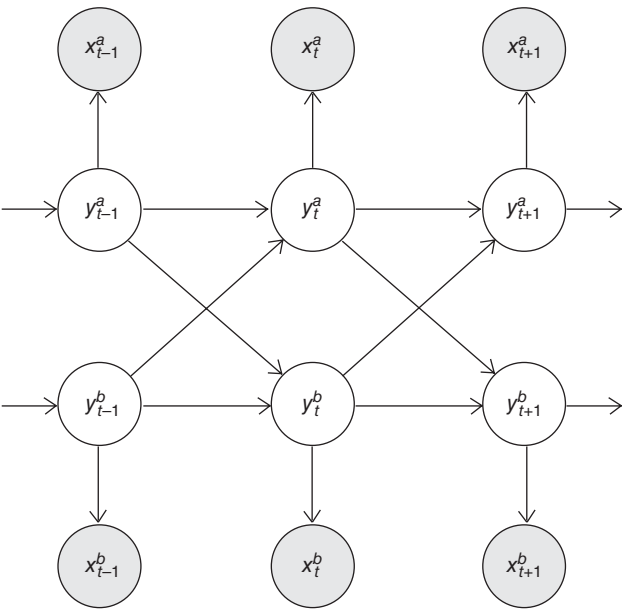


FIGURE 8.21 *Graphical representation of CHMM model for two users, denoted a and b.*

change to the algorithm consists of replacing the calculations from Equation 4.18 with those found in Equation 8.7. In this equation, X refers the input sequence of observable sensor events and Y refers to the two-dimensional stream of activities, Y^a and Y^b. The observation sequence in this case does not need to be separated into each user. Instead, the model is tested and used on data fused from multiple users. The output of the CHMM will provide an indication of which single-person or group activity is being performed by each individual in the space for each time point in the input sequence.

The second probabilistic graph that we will adapt to the multiuser situation is a conditional random field. Recall that while HMMs are generative models, conditional random fields are discriminative models and thus are popular for tasks such as activity recognition. Figure 4.11 shows a basic linear-chain conditional random fields (CRF). For multiuser activity learning, the basic linear-chain CRF shown in Figure 4.11 can be modified, or factored, as highlighted in Figure 8.22 to include connections at each time point between the different users and between the users and the observed sensor events. To use the CRF algorithms defined in Chapter 4, the potential functions now need to be enhanced to consider previous activities of each user as well as the current observed sensor event. These can be specified as $f_k(y_t^a, y_t^b, y_{t-1}^a, y_{t-1}^b)$ and $g_k(y_t^a, y_t^b, x_t)$. The instance-specific normalization function Z_x is also modified to encompass the new potential function. Inference for the factored CRF can be performed using the Viterbi algorithm as described in Section 4.3.

8.4.3 Separating Event Streams

The approaches considered so far look at one source of data that represents combined user behaviors in order to model, analyze and label the activities that are occurring within the data. The opposite approach can also be taken in which the combined data are separated into single-user streams and then analyzed using conventional single-person activity learning algorithms. The problem boils down to accurately

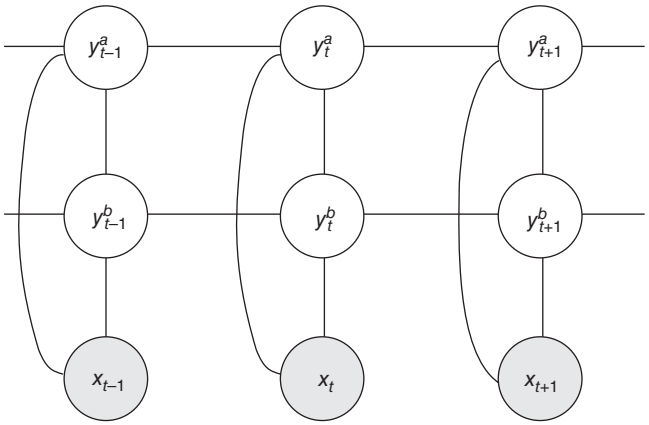


FIGURE 8.22 *Factored CRF representing a sequence of activities for two users, a and b.*

separating the data streams. This type of source separation problem has long been referenced in signal processing terms as the *cocktail party problem*. In this problem, a number of people are talking simultaneously during a cocktail party. In order to listen to just one person, an individual must be able to separate or filter out sources of noise other than the speaker of interest.

In Chapter 5, we discussed one way of focusing on one user at a time. Specifically, as a sliding window moves through the sequence of sensor events, the goal is to label the most recent event in the window with the corresponding activity (and user) that caused the sensor event or value to be generated. This problem can be addressed by defining a weighting scheme based on the mutual information between the sensors. The weights allowed the most relevant sensor events to play a more significant role in labeling the activity and user. This type of noise resistance plays a more critical role in multi-person scenarios than in single-person scenarios. The mutual information (MI) of two sensor events can be determined by their frequency of co-occurring within windows of sample training data. The resulting MI values provide an indication of the sensor events that likely belong to the same activity and/or person. Events with lower MI values can be removed or weighted lightly to reduce their influence on activity modeling and labeling.

An alternate approach is to apply temporal clustering to the data stream to find the best separation of data sources. In this approach, we assume that while the parameters that define a particular activity are known (they can be learned from labeled training data), the number of activities that are occurring at any given point in time are unknown and the mapping of a particular sensor event to an activity is unknown. Each occurrence of an activity is represented by a Markov renewal process (MRP), which is known to generate a sequence of sensor events with the Markov property, as shown in Equation 8.21. In this equation, τ_{n+1} represents the difference in time between t_{n+1} and t_n.

$$\begin{aligned} &P(\tau_{n+1} \leq t, x_{n+1} = j | (x_1, t_1), \ \ldots \ , (x_n = i, t_n)) \\ &= P(\tau_{n+1} \leq t, x_{n+1} = j | x_n = i) \quad \forall n \geq 1, t \geq 0 \end{aligned} \tag{8.21}$$

Activity occurrences are known to be time limited. Each associated process has a start time (birth) and an end time (death). The overall system is a set of such time-limited MRPs together with a Poisson process that generates clutter noise. As a result, the system is referred to as a multiple Markov renewal process, or MMRP. An algorithm to separate the MMRP into multiple streams consists of searching through a space of potential clusterings and evaluating each potential separation based on its likelihood given the data.

Many potential clustering techniques could be used to separate sensor events into non-overlapping subsets. For any candidate clustering the likelihood that the entire dataset is explained by a particular set of K clusters is shown in Equation 8.22, where $P_{\text{MRP}}(k)$ represents the likelihood that the sensor event sequence in cluster k is generated by a single MRP and $P_{\text{NOISE}}(h)$ represents the likelihood that a single sensor

event was generated by noise from noise model h.

$$\text{likelihood} = \prod_{k=1}^{K} P_{\text{MRP}}(k) \prod_{h=1}^{H} P_{\text{NOISE}}(h) \tag{8.22}$$

We divide the right-hand side of Equation 8.22 by the likelihood that all of the sensor events are generated by a single noise process. This results in the likelihood ratio shown in Equation 8.23 that needs to be maximized by the MMRP algorithm. In this equation, $P_{\text{NOISE}}(k)$ is used to express the joint likelihood of all events in cluster k being generated by the one noise model.

$$L = \prod_{k=1}^{K} \frac{P_{\text{MRP}}(k)}{P_{\text{NOISE}}(k)} \tag{8.23}$$

The overall likelihood ratio, L, indicates the relative likelihood that the sensor dataset was generated by the corresponding set of activity clusters and noise, as opposed to the case where all of the data is noise. The likelihood ratio for a single cluster k is shown in Equation 8.24. In addition to model parameters f that define the cluster activities, each activity occurrence in the cluster, $x_{k,i}$, is also governed by the likelihood P_b of a point in time when it is born, the likelihood P_d of a point in time when it dies, and a clutter noise component P_c.

$$\frac{P_{\text{MRP}}(k)}{P_{\text{NOISE}}(k)} = \frac{P_b(x_{k,1}) P_d(x_{k,n}) \prod_{i=2}^{n_k} f_{x_{k,i-1}}(x_{k,i}, t_{k,i} - t_{k,i-1})}{\prod_{i=1}^{n_k} P_c(x_{k,i})} \tag{8.24}$$

Equation 8.10 can be used to evaluate a candidate set of clusters. However, the number of combinations of such clusters is exponential. The MMRP algorithm simplifies the potentially exhaustive search by transforming the clustering problem to a network flow problem. The maximum flow problem finds a maximum-valued flow through a single source, single sink flow network (represented as a graph) where each edge in the graph indicates the maximum flow that can travel along the edge.

For MMRP problem, each node in the flow network represents a single sensor event. The flow component of each edge in this case represents the negative log cost of the corresponding portion of the cluster solution. This is calculated as the sum of the costs, a, for the related activity birth, death, clutter, and transition (to a new state and time within the activity model), as shown in Equation 8.25.

$$-\log(L) = \sum_{k=1}^{K} a_b(x_{k,1}) + a_d(x_{k,n}) + \sum_{i=2}^{n_k} a_t(x_{k,i-1}, x_{k,i}, t_{k,i}, -t_{k,i-1}) + \sum_{i=1}^{n_k} a_c(x_{k,i}) \tag{8.25}$$

Figure 8.23 provides an example of a flow network for the MMRP problem. The network has a single source, s, a single sink, t, and three vertices corresponding to three observed sensor events. Each edge is labeled with the corresponding costs and

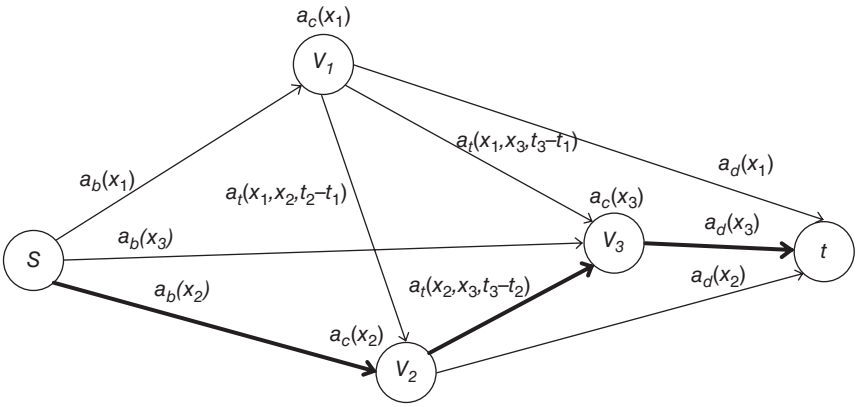


FIGURE 8.23 *Flow network for three sensor events with one cluster (path) highlighted.*

a unit capacity is defined for each of the edges in the network. Each path, such as the one highlighted in Figure 8.23, corresponds to a single cluster. Each vertex can be included in only one path and vertices not included in any of the final paths are considered noisy events.

Because each edge is labeled with a cost rather than a flow capacity, generating a set of clusters is equivalent to minimizing, rather than maximizing, the flow through the network. The Edmonds–Karp algorithm is known to solve this problem in time $O(VE^2)$. Here V is the number of vertices (in our case, sensor events) and E is the number of edges.

8.4.4 Tracking Multiple Users

One huge aid in separating sensor streams according to users is to track the users themselves. The computer vision and robotics communities have long investigated multiple track targeting techniques. An approach that has been tested in those communities as well as activity tracking is particle filters. Particle filters allow a population of possible navigational paths to be sampled and evaluated at once and their likelihood to be estimated using Bayesian techniques.

To discuss particle filters for user tracking, we first have to introduce the notion of Bayes filters to estimate the state of a dynamic multiuser system based on noisy sensor data. For user tracking, the state can represent user locations and activities and it is represented by a probability distribution, or belief, over the possible states at a particular time step, $p(X_t = x_t)$. Using information from sensor events, the Bayes filter updates the belief at each time step. The approach is simplified through using of the Markov assumption, as is found when updating probability distributions for HMMs (as discussed in Chapter 4). The simultaneous tracking and activity recognition (STAR) algorithm is described here in the context of discrete-event sensors to track individuals in a smart environment but can be adapted to process values from continuously-sampled sensors as well.

STAR estimates the state $x_t = \{x_t^1, \ldots, x_t^M\}$ for M users at time t using sensor events collected up to the current time, $z_{1:t}$. The probability distributions are updated after each sensor event as shown in Equation 4.18 and the Viterbi algorithm. Because multiple users are involved, the computation may grow more complex as indicated in Equation 8.20, although the additional relationships can be ignored if independence between users is assumed. However, we still need to be able to track M users simultaneously and thus need to associate, or map, each sensor event to a particular user. This is where particle filters step in.

For each time step, we want to find the best assignment of sensor events to users and correspondingly update the belief, or state of each occupant. Because the assignments of sensor events to users are not provided, STAR must estimate the posterior distribution over both user state and sensor assignments. As there may be multiple sensor events or readings at time t, the corresponding set of sensor values is represented as $\{e_t^1, \ldots, e_t^E\}$. A sensor assignment matrix θ_t is maintained, where entry $\theta_t(i,j) = 1$ if event e_t^i is mapped to user j and 0 otherwise. Equation 4.18 is thus modified to include the event mapping, following Equation 8.26.

$$P(X_{1:t}, \theta_{1:t} | z_{1:t}) = P(X_{1:t} | \theta_{1:t}, z_{1:t}) P(\theta_{1:t} | z_{1:t}) \tag{8.26}$$

Each of STAR's particles maintains the state of all users, the sensor event assignments and the weight of the particle, $s_t^j = \{x_t^j, \theta_{1:t}^j, w_t^j\}$. The state distribution x_t^j is updated using the Bayes filter, the assignment θ_t^j is updated using particle filter and the marginal distribution of the assignment is approximated using a set of N weighted particles as shown in Equation 8.27.

$$p(\theta_{1:t} | z_{1:t}) \approx \sum_{j=1}^{N} w_t^j \delta(\theta_{1:t}^j, \theta_{1:t}) \tag{8.27}$$

In Equation 8.14, w_t^j represents the importance weight of particle j. The value of $\delta(x,y)$ is 1 if $x = y$ and is 0 otherwise. The marginal distribution of the belief, or state values, can thus be computed as shown in Equation Error! Reference source not found.

$$\begin{aligned} p(X_t | z_{1:t}) &= \sum_{\theta_{1:t}} p(X_t | \theta_{1:t}, z_{1:t}) p(\theta_{1:t} | z_{1:t}) \\ &\approx \sum_{\theta_{1:t}} p(X_t | \theta_{1:t}, z_{1:t}) \sum_{j=1}^{N} w_t^j \delta(\theta_{1:t}^j, \theta_{1:t}) \\ &= \sum_{j=1}^{N} w_t^j p(X_t | \theta_{1:t}^j, z_{1:t}) \end{aligned} \tag{8.28}$$

In the particle filter, information up to time step $t-1$ is used to predict the next state for particle j. The importance sampling is repeated N times at each time step to generate a full sample set S_t. The algorithm is summarized in Figure 8.24.

Algorithm **STAR**()

// Initialization

$x_1 = \{x_1^1, .., x_1^M\}$ // Determine initial user locations using prior information

for i = 1 to N do

// Draw with replacement a random sample from previous time step according to weights

$s_{t-1}^j = RandomSample(S_{t-1}, w_{t-1}^j)$

// Sample a possible sensor assignment matrix

$\theta_t^j = AssignSensorEvents(p(\theta_t | x_{t-1}^j))$

// Update state of each user

$$p(X_t = x \mid z_{1:t}, \theta_{1:t}^j) = \frac{p(z_t \mid X_t = x, \theta_t^j) p(X_t = x \mid z_{1:t-1}, \theta_{1:t-1}^j)}{\sum_x p(z_t \mid X_t = x, \theta_t^j) p(X_t = x \mid z_{1:t-1}, \theta_{1:t}^j)}$$
$$\propto p(z_t \mid X_t = x, \theta_t^j) p(X_t = x \mid z_{1:t-1}, \theta_{1:t-1}^j)$$

// Importance sampling

// Weight new sample proportionate to likeihood of posteriors of each user state

$w_t^j = \eta \sum_x p(z_t \mid X_t = x, \theta_t^j) p(X_t = x \mid z_{1:t-1}, \theta_{1:t}^j)$ // η is a normalizing constant

done

return

FIGURE 8.24 *The STAR particle filter-based algorithm for tracking multiple users.*

The role of the AssignSensorEvents function that is called in Figure 8.24 is to generate a possible assignment of sensor events to users for the new sample. This function should incorporate information that is available about the users, sensors, and environment being used. Choosing an assignment that is impossible will result in a particle that has zero weight and thus wastes computational time in the algorithm. Such a situation can occur when spatial or temporal constraints are violated, such as assignment sensors from two different rooms to the same user at the same time. Instead, assignments can be chosen based on the probability of such assignments in labeled sample data or based on rules that are customized for the particular tracking environment and task. Ongoing techniques for identifying user locations based on biometrics can improve the selection of possible assignments as well-smart floors, wearable RFID, door sensors, cameras, and other measures provide improved probabilistic information on where individuals are located in a space. In addition, behaviometrics, or identification based on known behavioral patterns for each individual, can also supplement the assignment choices.

8.5 ADDITIONAL READING

A number of tools have been designed to aid with annotation of video and sensor data. The Anvil tool[166] shown in Figure 8.1 facilitates analysis of video and audio data, outputting tags to XML files. Another such tool is Elan[167], which is geared toward use in speech and gesture research. Annotation times have been reported by

researchers including Banziger[168] and Szewcyk et al.[169]. An explanation of the Kappa statistic for use in inter-rater consistency is provided by Fleiss et al.[170]. Song et al.[171] used social media and network analysis to associate social media posts with activities for classification purposes.

He and Garcia[172] overview work in learning from imbalanced data. In addition to sampling and cost-sensitive learning, they describe kernel-based learning methods that have also been explored for their role in handling imbalanced data. Kernel-based learning methods make the minority class samples more separable from the majority class by mapping the data to a high dimensional feature space. The kernel classifier construction algorithm proposed by Hong et al.[173] is based on orthogonal forward selection and a regularized orthogonal weighted least squares estimator. Wu et al.[174] propose a kernel-boundary alignment algorithm for adjusting the SVM class boundary by modifying the kernel function's matrix according to the imbalanced data distribution. Another kernel modification technique is the *k*-category proximal support vector machine proposed by Fung et al.[175]. This method transforms the soft-margin maximization paradigm into a simple system of *k*-linear equations for either linear or nonlinear classifiers. The SMOTE sampling method was created by Chawla et al.[176] and the boosting approaches to cost-sensitive learning were introduced by Sun et al.[177]. Das et al.[178] offer an alternative sampling method that maintains the original probability distribution of the minority class and therefore may be better suited to handle class imbalances for sensor data that reflect human behavior.

The idea of forming generalized queries in order to maximize active learning benefit while reducing the number of oracle queries was introduced by Rashidi and Cook[179]. Alemdar et al.[180] and Rebetez et al.[181] have examined alternative measures for selecting data points to query in active learning. Stikic et al.[182] evaluated both semi-supervised learning and active learning for their role in generating labeled training data. Tapia et al.[183] experimented with experience sampling for activity labeling. Moskowitz and Young[184] provide a more in-depth explanation and assessment of ecological momentary assessment. Zhu and Goldberg[185] provide an excellent overview of semi-supervised learning and Chinaei[186] explores the topic specifically for support vector machine classification. Several researchers have investigated methods of employing crowd sourcing for activity labeling. These include Lasecki et al.[187], Hwang and Lee[188], and Zhao et al.[189].

The field of transfer learning is quite large and varied. Pan and Yang[190] provide an overview of the area as a whole, Xu et al.[191] describe a range of approaches that have been explored for multi-view learning, and Feuz et al.[192] survey transfer learning techniques that have been applied to activity recognition. In the field of machine learning, transfer learning is studied under a variety of different names including learning to learn, life-long learning, knowledge transfer, inductive transfer, context-sensitive learning, and metalearning[193–197]. It is also closely related to several other areas of machine learning such as self-taught learning, multi-task learning, domain adaptation, and co-variate shift.

Transfer learning has been applied to activity learning from video data[198–204], wearable sensor data[8,205,206], smart phone data[207,208], and ambient sensor data[209–211]. The type of information that has been mapped between the source

and target for activity learning includes transfer of data points as illustrated in the TrAdaBoost algorithm[203,212,213], through activity similarity calculation performed by the CDAR algorithm[209], and using other weighting methods pursued by Hachiya et al.[205] as well as Lam et al.[200]. Another type of transfer discussed in this chapter is transfer of feature representation, as shown through the multi-home transfer approaches[210,211]. Liu et al.[201] as well as Krumm and Rouhana[214] adopt a similar approach by defining an abstract feature vocabulary that represents a generalization of the features found in the source and target spaces. Blum and Mitchell[215] propose co-training to propagate labels between the source and target views. The Co-EM approach is introduced by Nigam et al.[216] for this task.

Approaches to multi-view learning are surveyed by Sun[217]. The relationship between the source and target views can be aligned using a variety of techniques. As examples, Kakade and Foster[218] explore canonical correlation analysis, Wang et al.[219] introduce manifold alignment and Sindhwani and Rosenberg[220] use manifold co-regularization. The teacher-learner approach to multi-view learning was introduced by Kurz et al.[206] and was further explored by Roggen et al.[221] for use in introducing new sensors in a dynamic fashion to the activity recognition process. Wang and Mahadevan[219] and Shi and Yu[222] have applied dimensionality reduction and data projection calculation to accomplish multi-view transfer learning for heterogeneous feature spaces. Wang and Mahadevan make use of the Procrustes analysis described in this chapter to align the source and target feature spaces. Two other types of transfer outlined by Pan and Yang are transfer of model parameters and transfer of relational knowledge. Parameter transfer has been explored by Cao et al.[223] who model the source and target tasks using GMM which share a prior distribution and by van Kasteren et al.[210] who learn hyperparameter priors for a HMM from both source and target domains. Yang et al.[204] actually learn a delta function from the source to target decision functions using a support vector machine. Lastly, Pan and Yang explore the idea of identifying underlying relationship in the data and transferring this knowledge from the source to target domains[190].

A comprehensive overview of approaches for multi-label learning is presented by Tsoumakas and Katakis[224] and Zhang Zhou[225]. Tai and Lin[226] introduced the idea of transforming label vectors using principal components and Zhang and Schneider[227] introduced use of CCA for transforming the label vector. Sun et al.[228] provide an in-depth analysis of CCA with application to multi-label learning and Chen and Lin[229] extend the idea of using feature-aware label space transformation by relaxing some of the constraints applied to traditional CCA.

Activity learning for and in the presence of multiple individuals has drawn increased interest in recent years. Pentland[230–232] engineered some of the first approaches to specifically detecting interactions between individuals, performing computational social science using sensors and reasoning. Hung et al.[233] have evaluated the ability of traditional activity learning methods to recognize actions that occur in social settings such as speaking and laughing. Other researchers who specifically focused on detecting group activities include Hirano and Maekawa[234] who train models on group activities using group roles as discriminating features. Petersen et al.[235] and also introduce new features for activity models, in this case to

specifically detect the presence of visitors in smart homes. Kjaergaard et al.[236,237] Gordon et al.[238], and Lu and Chiang[239] fuse information from multiple sensors in order to model and track potential group activities. Ryoo and Matthies[240] adopt a first-person perspective in which they model and recognize interactions between external entities and the observer.

The idea to modify probabilistic graphs such as HMMs, DBNs, and CRFs in order to perform multiuser activity recognition has been evaluated by a number of research groups including Wang et al.[241], Wu et al.[242], Tolstikov et al.[243], Hu and Yang[244], Oliver et al.[245], and Chiang et al.[246]. Gu et al.[247] also consider the problem of training on one user at a time but testing in multiuser situations. However, they take an alternate approach based on looking for differences in feature frequencies between possible classes of activities. The maximal flow approach to clustering data streams was introduced by Stowell and Plumbley[248]. Ye et al.[249] take a different approach to separating sensor streams. Their KCAR algorithm maintains an ontology of activities and associates sensor events with the most likely corresponding activities. These mappings are used to partition a single sensor stream into fragments, each of which is one activity that could be performed in parallel with the other activities.

Wilson and Atkeson[250] and Choi and Savarese[251] have explored the use of particle filters, belief propagation, and branch-and-bound search to simultaneously track individuals and identify their activities in multi-resident settings. These tracking-based approaches are boosted by work in biometrics including smart floors[252,253], smart doorways[254], and smart doormats[255]. Lu et al.[256] introduced the notion of switching between sensor modalities to reduce ambiguity. Kwapisz et al.[207] identify users through sensor found in ubiquitous smart phones and Cornelius and Kotz[257] use accelerometer patterns to identify the current user. In contrast, Crandall et al.[258] identify individuals through behaviometrics instead of biometrics. This approach learns behavioral patterns that represent distinct signatures for individuals residing in a particular environment.

9

Applications of Activity Learning

Up to this point, we have focused in this book on the techniques that can be used to model, discover, track, and predict actions and activities of interest. Now we take a closer look at why these techniques might be used. Specifically, we provide a survey of some general areas in which activity learning has been applied. Although the potential uses of activity learning and activity-aware services are unbounded, we focus our attention on a few domains of current interest. These include health, transportation, marketing, emergency management, home automation, security, and behavioral analysis. We also provide an overview of methods that can be used to express activities and activity occurrences in understandable terms using visualization and text diary descriptions.

9.1 HEALTH

According to the UK's Secretary of State for Health, the possibilities of using sensor data and activity learning for health monitoring and intervention are "extraordinary." The need for these capabilities is increasing dramatically, primarily due to the aging of the population. The estimated number of individuals over the age of 85 is expected to triple by 2050 and currently 50% of adults in this age category need assistance with everyday activities. Because more individuals are living longer with chronic diseases and a shortage will emerge in the care workforce, we must consider innovative health care options if we are to provide quality care to our aging population.

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

In addition to assisting older adults, there are many other large segments of the population that need health monitoring and assistance that can be boosted using activity learning techniques. Approximately 1% of children today have an autism spectrum disorder (ASD), which represents the fastest-growing development disability. Based on findings on the United States, over 7% of the population has had post-traumatic stress disorder (PTSD) at some point in their lives. These conditions have a dramatic impact on an individual's quality of life. The associated health care costs are also significant: in the United States, the annual cost of caring for individuals with dementia is $600 billion. Health care costs are approximately $126 billion annually for ASD and $42 billion for PTSD. These numbers do not account for the time and resources that friends and family contribute to taking care of loved ones with chronic conditions, often at the expense of their own time, potential income, and health.

Activity Learning and Health Assessment Activity learning plays an integral part in understanding the relationship between behavior and health. In the same way that DNA is comprised of millions of nucleotides, an individual's behavioral signature is comprised of millions of individual sensor events. Activity learning can thus play an integral role in "mapping the behavior genome." In fact, change in behavior is a common manifestation of many conditions including dementia, ASD, and PTSD, and is used by clinicians as a way to diagnose such conditions. In clinical settings, manual diagnosis of chronic conditions can be time intensive and can require multiple pieces of information including laboratory results and test scores. Access to clinical expertise can be difficult for individuals living in remote locations and the monetary expense of a medical diagnosis can be prohibitive, thus making reliable alternatives to traditional medical diagnosis valuable.

Activity learning and analysis has already helped improve the understanding of the relationship between behavior and health and laid the foundation for automated health assessment. For example, behavioral changes in smart environments have been shown to correlate with the onset of dementia. These changes include the speed with which daily activities are completed, the manner in which activities are performed, the time of day in which activities are initiated, walking speed in a familiar environment, and overall activity level. ASD has been associated with specific actions in young children, including hand flapping and body rocking. Recognition of these actions using pervasive sensors is valuable in diagnosing the condition and early detection greatly improves the effectiveness of medical treatments. Similarly, PTSD is known to have an effect on sleep amount and quality, which again can be recognized and analyzed using activity modeling and recognition techniques. By utilizing activity discovery, additional routine behaviors can be detected that are also correlated with known medical conditions. Some machine learning techniques have been introduced that can even categorize individuals into health categories (e.g., healthy, mild cognitive impairment, or dementia) based on features extracted based on individuals performing activities in sensed environments.

Health Interventions Activity monitoring can provide information to individuals and their caregivers on behavioral routines and potentially on health status. However,

FIGURE 9.1 *The COACH hand-washing assistance system (a)* Activity forecasting-based prompts can be delivered to a computer, television (b), or mobile device (c). *Source: Reproduced by permission of Alex Mihailidis.*

they can also be used to provide activity-aware interventions that improve quality of life and allow individuals to function independently. For example, sensors can be used to analyze movements and provide feedback to individuals who are undergoing physical therapy.

At a higher level, activity learning also plays a key role in helping individuals with memory issues to initiate and complete critical activities of daily living without relying on caregiver assistance. As an example, the COACH system assists individuals with dementia to wash their hands in a correct and complete manner. The system relies on a single video camera that tracks both hands as well as the towel, as shown in Figure 9.1. Extracted features are fed into a partially observable Markov decision process (POMDP) that models the steps involved in the hand washing process. Example actions that can be generated by the POMDP include prompts for the user to initiate the current plan step or automated calls for human assistance.

In addition, the activity prediction techniques described in Chapter 7 can be utilized to prompt individuals to initiate activities once they have performed the activity enough times for it to be an established part of their routine. The prompted tasks can be routine activities that need to be maintained or new activities that are introduced in order to achieve a healthier lifestyle. The advantage of an activity-aware approach to prompting is that no prompting rules need to be written and customized by the user, caregiver, or expert. In addition, a data-driven approach to prompting is sensitive to contexts in which the activity is a natural fit because the individual has previously performed the activity in such a context. The activity prompter can be coupled with an activity recognition algorithm to determine if the prompted activity was in fact initiated by the user.

In order to determine if a prompt should be issued, the activity prompter needs two pieces of information: the current date/time and occurrences of recent activities as detected by an activity recognition algorithm. The information is fed into an activity prediction or forecasting algorithm. A prompt can be delivered in advance of the time the activity should be performed if the individual wants a reminder, or once the predicted time of the activity has passed and the activity has not been detected. As shown in Figure 9.1, the prompt can be delivered to stationary or mobile devices and repeated until either the activity is detected, a fixed number of repeat iterations have occurred, or the user interacts with the prompting system.

The effectiveness of prompting systems has been documented in the literature. In addition, evidence has been provided that presenting performed activities to an individual can aid in memory performance and recollection of the performed tasks. A common therapy for individuals with memory problems is to note daily activities in a memory notebook, which can then be referenced to recall the events that occurred throughout the day and the planned activities that still need to be completed. The SenseCam digital camera helps automate this process by taking pictures at rates up to 10 frames/second, then turning the sequences of pictures into an abbreviated video.

9.2 ACTIVITY-AWARE SERVICES

Many companies are realizing that effective software goods and services are those that are aware of the user's context. Unlike software that requires excessive configuration or applications that provide redundant and unwanted information at inopportune moments, activity-aware services close the loop shown in Figure 1.1 by using senses and reasoning to determine the user's current activity context, then selecting a piece of information or action to take that is beneficial given the overall goal and the current situation. An increasing number of applications offer context awareness. For example, Square Register allows retail store clerks to retrieve information on individuals who enter their store and customize their experience accordingly. Some navigation applications draw upon knowledge of the user's current location, preferences, and navigation history to suggest fast routes and stops along the way that will capture their interest. Providing activity-awareness allows software designers to raise the bar on customer insights and customized applications.

One area that has made effective use of activity awareness is pervasive gaming. Pervasive games allow users to interact with the game and other players seamlessly in both virtual and real worlds where the game technology is invisible. Activity learning is valuable in such games where detecting particular actions and tasks is important to assign points, advance the storyline, and customize the game environment. As an example, the "Them and Us" game, shown in Figure 9.2, detects and facilitates social interactions within the game. "Them and Us" pulls from everyday social settings to lay out the game, including weddings, conferences, dates, classes, and playgrounds. The game monitors the movements of the players and their automatically-detected interactions to monitor how players relate to each other and work together within the game. The more a player participates and collaborates, the more points they can score.

Another application that is starting to make use of activity awareness is home automation. Users of smart home technology envision a day in which their house will be not only aware but anticipatory of their activities and automate the home accordingly. The temperature should adjust based on an individual's sleep/wake and leave/return routines, the coffee should be brewing with the resident wakes up, and their favorite music should be playing when they come home after work. Concerns about sustainability are growing as we see that current consumption of energy is up by 200% since 1949. To address these concerns, home automation can be used to

FIGURE 9.2 *A "Them and Us" game replayed inside the after-game visualization interface. Source: Reproduced by permission of Alan Chamberlin.*

reduce consumption resources are not needed to support current activities or to time resource consumption when renewable energy is available.

The MavHome smart home project provides one example of an activity-aware automation strategy. The MavHome automation goal is to minimize the number of manual interactions that occur in the home by automating the interactions without introducing unnecessary device on/off changes that need to be reversed by the resident. To do this, sequential prediction algorithms as described in Chapter 7 identify the likely activities and interactions that will occur in the near future. A hierarchical POMDP is constructed based on observed sensor events and the learned corresponding resident activities. As shown in Figure 9.3, nodes at higher levels of the hierarchical model represent activities that are combinations of activities or events that comprise the lower levels. A reinforcement learning algorithm is applied to the POMDP to identify an automation strategy that achieves the minimum-manual-interaction goal. If the resident does reverse an automation step, the corresponding "no act" step is added to the POMDP so that the home eventually learns when a beneficial automation step should be taken and when automation should be avoided in order to not be an annoyance.

9.3 SECURITY AND EMERGENCY MANAGEMENT

Another application of activity learning that is garnering attention is activity-aware security and emergency management. Activity-aware security systems utilize information about recognized activities to improve security risk detection and improve the

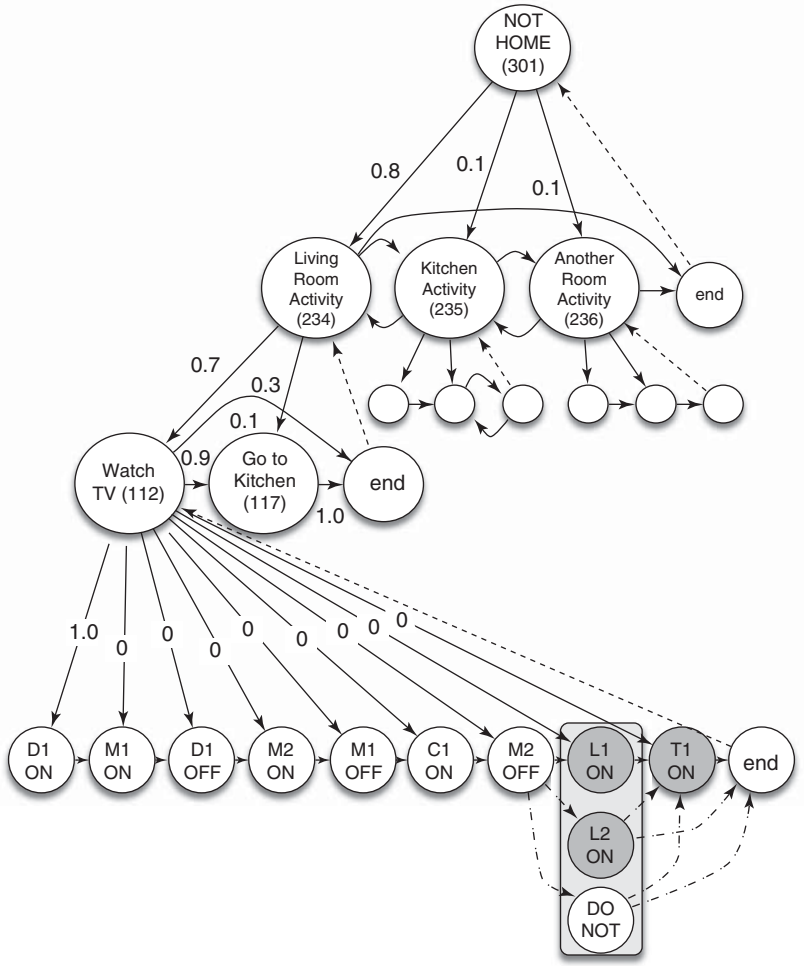


FIGURE 9.3 *Example hierarchical model learned from MavHome sensor data.*

effectiveness of security decisions. If the goal of a system is to monitor the safety of a physical site, then activity learning provides valuable knowledge about the types of activities that are expected to occur, the times they normally occur, the people who perform the activities and the manner in which they are performed. If no activity model supports the current sensor events at the site, then appropriate action should be taken.

One example of activity-aware security surveillance is a Markov Chain Monte Carlo (MCMC) algorithm that analyzes video to identify low-level interactions between human and vehicles. Probabilistic reasoning is used to jointly track humans and vehicles and to classify short videos with labels such as enter/exit vehicle, approach vehicle, open vehicle door, sit inside vehicle. These automatically-recognized actions occur in a sequence that is then pieced together to reassemble

an entire scene. Scene reconstruction can then be used to monitor an area such as a parking lot or to assist military operations in detecting possible car bomb plants.

Another critical application of activity monitoring is emergency management. Tracking activities of individuals and group in a building or region can facilitate efficient evacuation when emergency situations arise. Another use that has been explored is coordination of activities among firefighters. Firefighters work as a team but split into subgroups in order to work on tasks in parallel. Individuals within a subgroup may be in close proximity, but those in other groups may be outside of visual range that makes it difficult to coordinate activities. Automatic recognition and tracking of group activities and movement patterns is beneficial for training teams in emergency management and for helping groups coordinate their actions in complex mission situations.

9.4 ACTIVITY RECONSTRUCTION, EXPRESSION AND VISUALIZATION

As we have demonstrated in this chapter, modeling and learning activities opens up a range of software programs, interventions, diagnostic tools, and services that are activity aware. While many of these tools can be automated, humans also want to be aware of activities that are being performed. Techniques need to be developed that generate an understandable interpretation of sensor-based activity data, that reconstruct a scene or time period of interest based on known activities, and that provide insight on the relationship between activities and factors such as health and resource utilization.

A key to interpreting sensor data and activity recognition results is visualization. Here we provide an overview of alternative activity visualization, reconstruction, and description techniques that provide an effective interface for interpreting sensor data and visualizing sensor-detected activities. We first consider visualization of raw data, then visualization of activity timings and characteristics and finally methods of charting time-based activity occurrences and trends.

Visualizing Activity-Based Sensor Data Learning activity models is valuable because activity labels provide us with a rich vocabulary to express the behavior of individuals and groups. However, visualizing even raw sensor data can provide insights on activities and parameters that influence activities. Figure 9.4 shows an example visualization of accelerometer data collected via watch-based recorders. The sensors are worn by two macaques in this study. The height of each part in all of the charts indicates the physical activity intensity level reflective of accelerometer-based motion. Each line in charts A and B contain a visualization of one day of data, where daytime hours are indicated by white bars at the top. Panels C and D display activity by hour, averaged over the 35 days. Differences are apparent even at an initial glance between the individual plotted in panels A and C and the individual plotted in panels B and D. The relationship between time of data and activity intensity is also clear. The side-by-side charts allow inspection of aggregated behavioral parameters along with

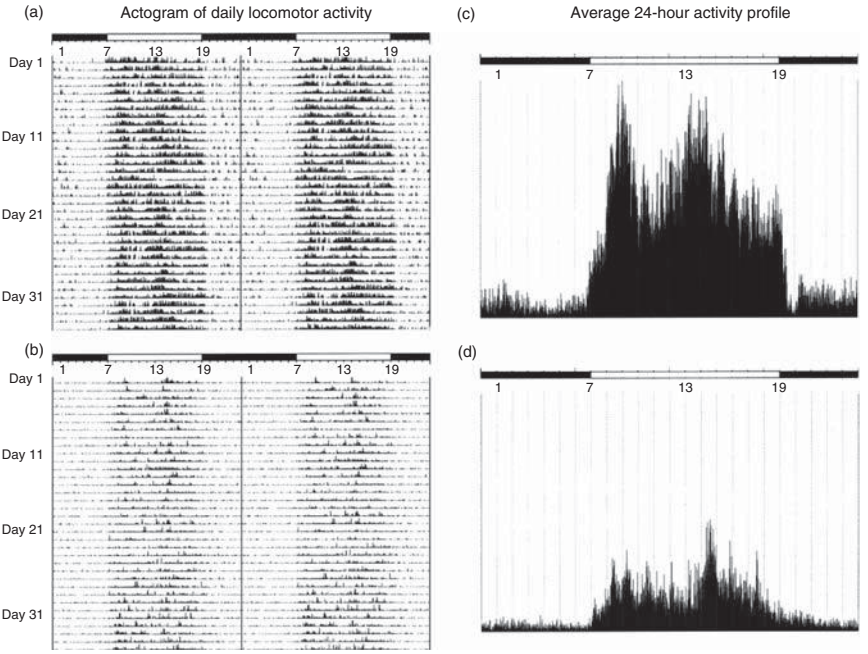


FIGURE 9.4 *Activity profiles based on visualization of accelerometer data. Source: Copyright© Elsevier, 2015.*

detailed daily analysis. In addition, visualizing multiple individuals on one screen highlights behavioral differences between individuals.

The charts shown in Figure 9.4 highlight features of behavior using time-based sensor values. When discrete-event sensors are used, a similar visualization can be created based on the frequency, or density, of sensor events. The result is an activity density map, such as the ones shown in Figure 9.5. In an activity density map, different colors or color intensities are used to represent varying levels of motion sensor density. The density is computed as the number of motion sensors events that are generated during a specific hour divided by the time that is spent in the sensed environment during that hour. The darker the color, the greater the number of sensor events (indicative of more movement and activity during that time). Each row (along the y axis) represents the hours in a single day from midnight at the bottom to 11:00pm on the top. Each column represents a distinct day. Figure 9.5 shows examples of month-long density maps generated from three different homes. The map on the left shows a very regular routine each day. Time spent sleeping is indicated by the lighter-colored rows at the bottom of the map and the light-colored columns around days 13–15 indicate time spent out of the home. In the middle plot the individual is highly active even while sleeping. In contrast, the plot on the right indicates very little activity during the day as well as at night.

Once activities are identified, visualization techniques are again useful to understand the nature of the activities. Figure 9.6 shows a radial bar chart that displays

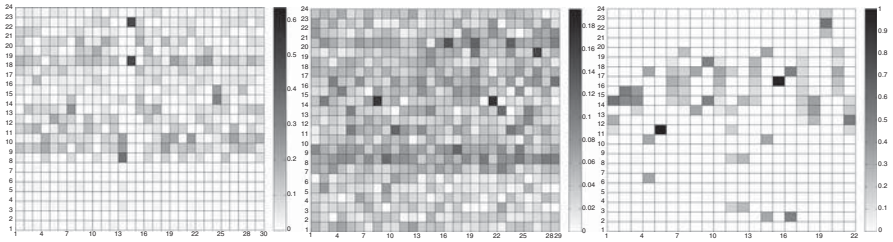


FIGURE 9.5 *Sample activity density maps.*

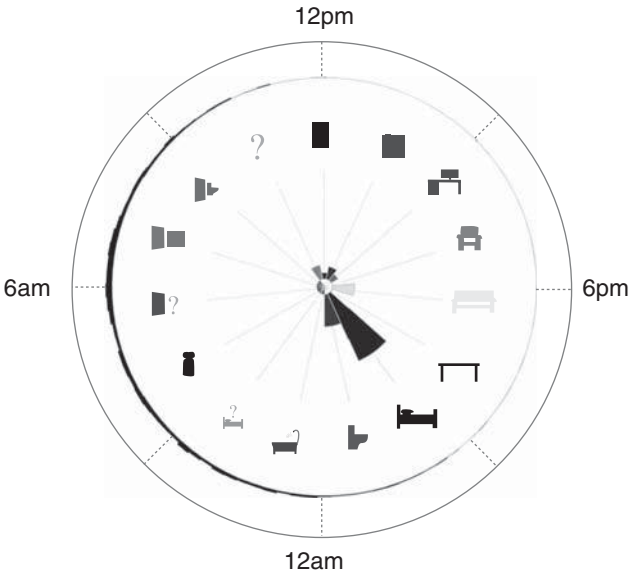


FIGURE 9.6 *Radial bar chart for the Bed Toilet Transition activity.*

features of the activity including the location and time of day distributions over occurrences of the activity. In this particular chart layout, bars on the inside of the circle point to locations in the home (the 12:00 position is the front door, followed in a clockwise direction by the kitchen, work area, lounge chair, living room, dining room, bedroom, bathroom, shower, other room, medicine cabinet, other exterior door, kitchen door, bathroom door, and other location). The radial chart presents the clearest information when there is a natural similarity-based ordering among the locations in the chart. The icons in this chart are generated by computing maximal information coefficient distances between each location and then performing a least-cost traversal of the similarity-based graph. Bars on the outside of the radial bar chart indicate the times during the day when the activity is performed. For example, Figure 9.6 shows the radial bar chart for Bed Toilet Transition activities as they occur for one individual in a particular home.

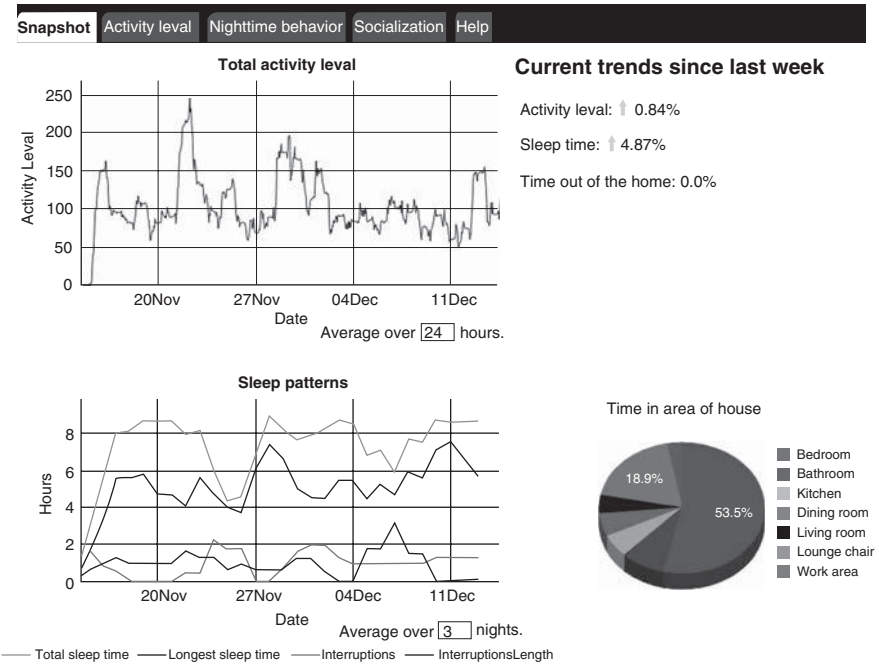


FIGURE 9.7 *Visualization of activity density, activity times, and activity trends.*

An alternative type of activity information display is shown in Figure 9.7. In this chart the activity level, similar to the density graph in Figure 9.5, is shown together with time spent in detected activities of interest (in this case, Sleep). An additional feature of this visualization is that in addition to displaying parameters of the activity occurrences, behavioral trends can also be visualized. By comparing changes in time spent on individual activities that are automatically detected from sensor data, trend directions and amounts of changes can be calculated and displayed. In this case, the individual experienced a 0.84% increase in overall activity density in the environment, sleep increased by 4.87%, and time outside the home remained constant.

Finally, Figure 9.8 shows a visualization called an activity storyline. In this visualization, the activities that are performed and locations that are visited are displayed along a daily timeline. In this particular example, smart phone sensors are used to collect information about an individual's movement and activity patterns, which in turn are categorized and displayed on the phone itself. As with the other visualization strategies, this snapshot provides a quick, easy to interpret overview of the activity types, times, durations, and locations and thereby offers tremendous insights on the individual's behavioral routine.

Generating Text Descriptions of Sensed Activities An alternative to graphically displaying sensed activities is to generate a textual description of a person's daily routine based on the time and location of sensed activities. Such a notion of a

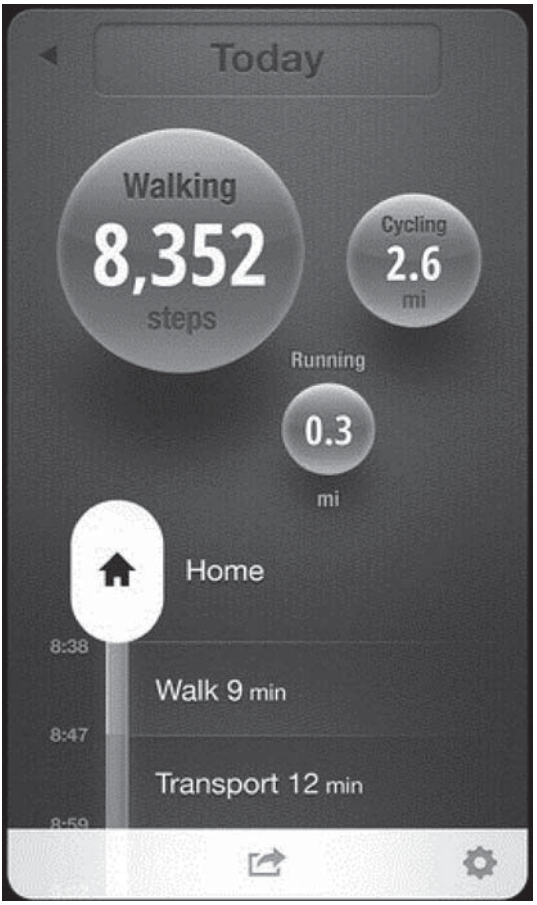


FIGURE 9.8 *Visualization of an activity daily diary.*

"daily activity diary" has been developed for location-based storylines. The textual diary provides a succinct description of a routine that is appropriate for providing summaries without loading the reader down with the need to interpret each sensor event that occurs throughout the day. These summaries would be valuable for caregivers monitoring a loved one or for security companies monitoring the activities that occur within and surrounding a building.

Before translating activity occurrences and descriptions to readable text, the daily diary must first determine what level of detail to provide. When descriptions are based purely on a person's location, for example, a text entry could be made in the diary each time the latitude/longitude position of the individual changes. Sensor data needs to first be processed to filter out sensor events that are not important for mentioning (e.g., current sensor battery levels) as well as activities or locations that would be considered a transition between activities, as described in Section 5.1. Next, sensor

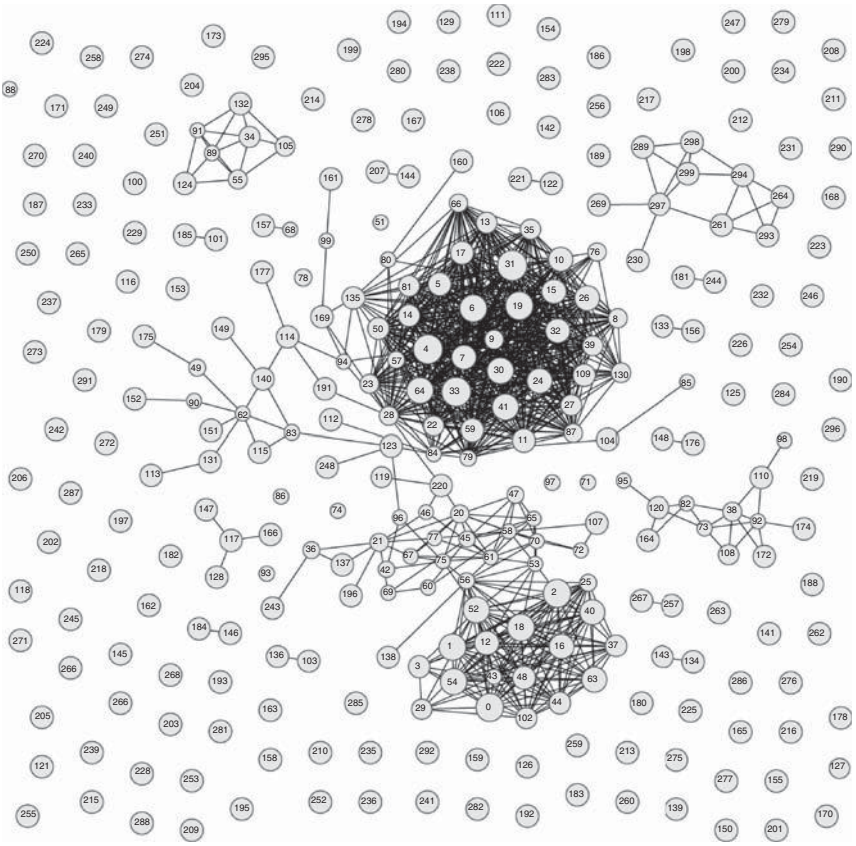


FIGURE 9.9 *Wi-Fi cluster co-occurrence plot. Source: Copyright© Elsevier, 2015.*

locations or activity occurrences are clustered to identify the more general location or more abstract activity type that encompasses a significant portion of the person's time. For example, activities Cook Dinner, Set Table, Eat, and Wash Dinner Dishes could be clustered based on spatio-temporal features and generalized to the activity label Eat Dinner. For example, Figure 9.9 shows a plot of location clusters in which clusters are connected by an edge if they are both sensed at close points in time. The largest group of densely connected vertices in the figure is associated with home and the next largest is associated with work. The cluster occupancy is averaged for separate time periods from sample data to visualize the average behavior for an individual and use the corresponding locations, or associated activities, as terms in the diary descriptions.

Next, statistical machine translation techniques are used to generate English-language descriptions of sensor event clusters or activity occurrences. To do this, sensor data clusters, labeled with the corresponding activity names, can be provided as training data with sample textual descriptions of the data. For example, the sensor data provided in Figure 5.4 could be mapped to the textual description:

Sue got dressed at 10:18am. Sue left home at 10.19am and returned 2 hours later.

This approach to generating activity diaries assumes that each sentence contains at least one reference to the time and the sensor event or activity that it describes. It also assumes that each sentence is an independent description that does not reference information from other sentences. The text-based activity diaries can be coupled with visual summaries described earlier in this chapter to provide a comprehensive description of behavioral routines based on sensor data.

9.5 ANALYZING HUMAN DYNAMICS

Scientists spend an extreme amount of time trying to understand the nature of human behavior. Psychologists, anthropologists, and sociologists pose and attempt to answer difficult questions about individual and collective human behavior. Most of the analysis techniques rely on data collected from survey information. As an example, the American Time Use Survey asked thousands of American residents to recall how they spent every minute of a day. The results are plotted in Figure 9.10. The problem is that these data collections are based on self-report and limited observation, which introduces a great deal of noise and bias into the data.

With our ability to now ubiquitously sense and learn activities that are performed by individuals and groups, these questions can be answered in a more complete and definitive manner. Here we highlight four studies that address specific questions about human behavior that are answered using sensing and activity learning techniques described in this book.

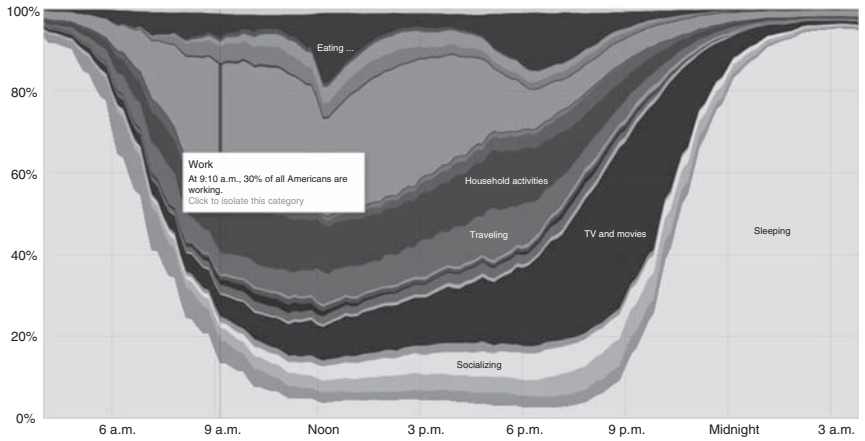


FIGURE 9.10 *Visualization of how surveyed Americans spent their time in 2008[259]. Source: Reproduced by permission of IOP Publishing.*

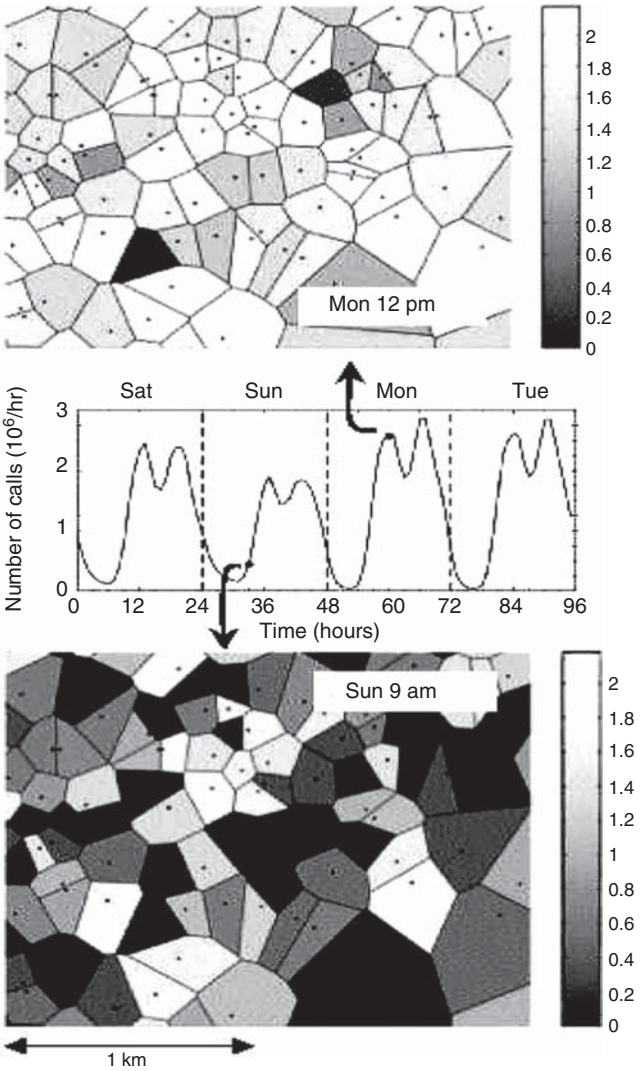


FIGURE 9.11 *Mobility patterns grouped by cell tower region for two time periods. The bars on the right correspond to the number of calls per hour on a log scale.*

Understanding Human Mobility at a Population Level The first study we look at exploits mobile phone use data to understand at a city level how humans move and act throughout a typical week. Figure 9.11 illustrates how patterns of human location within a city and their social connections change throughout the day and week. The top plot in the figure shows activity to be centered around offices during work days (top picture) and for less activity to be observed, with a different spatial pattern, on the weekends (bottom picture).

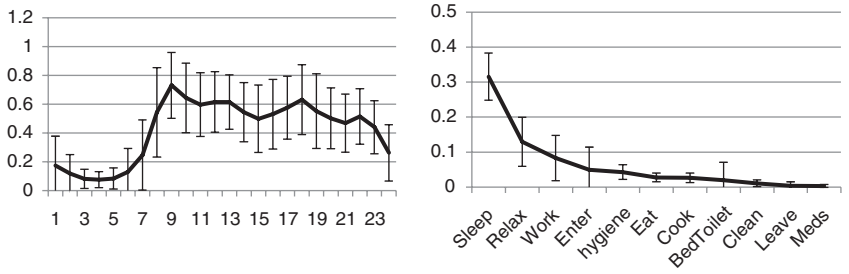


FIGURE 9.12 *Plot of relative activity level as a function of the hour of the day (left) and relative activity duration as a function of the activity class (right).*

This study also analyzes individual activity patterns based on mobile phone data. Because average mobile phone use varies between people, the users are first grouped based on total number of calls. The probability density function of time between calls is measured and found to be exponential with a heavy tail. The theoretical model of the "Phone call" activity frequency is also found to be quite similar with distributions for other activity frequencies, including sending and receiving email.

Analyzing how Human Spend Their Time Using Activity Learning Techniques The second study we include identifies how activity level varies throughout the day and how individuals spend their time for a group of individuals. Both questions are answered by applying activity learning and recognition algorithms to data collected in smart home environments. Figure 9.12 shows the results of these analyses for 18 smart homes. As the plots indicate, there is a clear pattern for the entire group in which activity levels are low in the early hours of the day but then increase, peaking at mid-morning, midafternoon, and early evening. The exact activity levels vary quite a bit across the population, which may be due to mobility differences and due to sensor granularity within the home.

In contrast, the variance across the population for time devoted to various activities is quite a bit smaller. As the graph shows, the most time is dedicated to sleep while other activities receive less time such as taking medicine (which is typically quick) and cleaning the home (which may not happen as often as other activities). Larger variances exist for the enter activity (which takes into account time spent outside the home) and for bed toilet transitions, which do vary dramatically by age, health, and sleep quality. Overall, the sensed activity distribution is somewhat similar to the distribution that was created based on survey data shown in Figure 9.9. Automating the data collection and analysis from sensor information and activity learning algorithms, however, facilitates scaling such analyses to much larger groups and numbers of activities. The exponential decay in activity time is reflective of many other phenomena found in nature, including the phone use frequency mentioned earlier but also including purchasing patterns and population distributions.

Modeling the Connection Between Human Behavior and Social Relationships In this study, the connection between social relationships and activities was investigated using both survey data and sensed activity data. In this case, activity

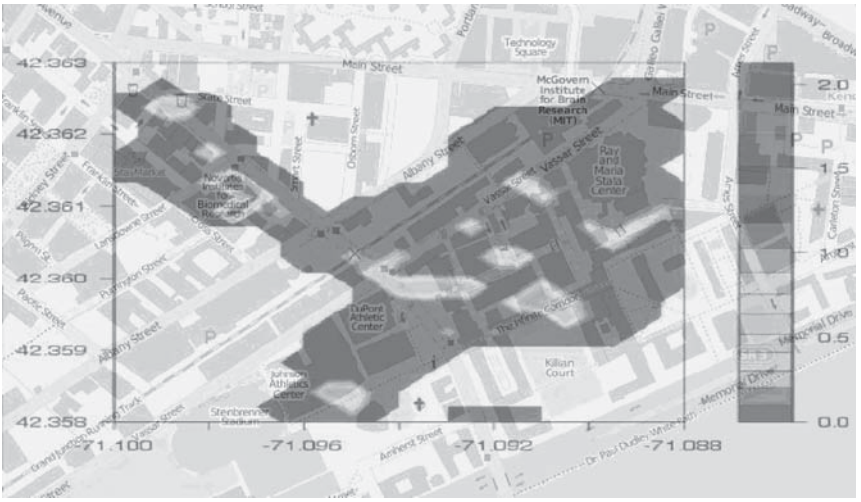


FIGURE 9.13 *The meaningfulness of a place is indicated by friendships that are made there. Source: Reproduced by permission of Alex Pentland.*

patterns for individuals living in a college dorm were analyzed. Survey information captured information about perceived relationships with others in the dorm and mobile phone sensors recorded location-based activity every 6 minutes.

Many of the sensor-based findings aligned well with social theory and with survey results. In particular, self-reported friends have a higher correlation with participation in selected events. Pairs of individuals who exercised together had a greater likelihood of being friends. The study also found that the probability that any two individuals would become friends varied with the locations on campus where they spent time together. Figure 9.13 shows a plot of this finding, which indicates that some locations have greater significance for forming social relationships than others.

Predicting Social Relationships Based on Human Activity and Mobility Finally, the last study we highlight analyzes the relationship between human mobility-based behavior and social interactions. The insights based on mobility-based activities and phone call-based social interactions using mobile phone records. Like the previous study, mobile phone data suggests that for any given pair of individuals, the more similar their mobility activity patterns are the higher the chance is for a close social network tie between the individuals. The observed connection is actually used here as a basis for predicted social network links based on user activity and mobility information.

9.6 ADDITIONAL READING

Activity learning and tracking have moved to the forefront of technologies for health care. Statistics related to the aging of the population, ASD, and PTSD are available

from a variety of reports and web sites[260–264]. The link between activity performance and health status has been recognized by many researchers including Farias et al.[265], Covington[266], and Friedman[267]. Kaye et al.[268,269] and Wadley et al.[270] have detected correlation between variances in computer use and cognitive health and between walking speed and cognitive health. Aharony et al.[271] investigate the relationship between sensed social behavior, financial status, and decision making. Moturu et al.[272] link sensed social behavior with sleep patterns and self-perceived mood. Saito et al.[273] predict BMI based on social behavior. Dawadi et al.[113,114,274] have generalized these findings to evidence of correlation between daily behavioral routines and cognitive and physical health status. These changes have been detected using sensors placed in smart homes to monitor routine without disrupting it. Albinali et al.[275] and Westeyn et al.[276] have used action recognition to detect signs of ASD in young children. Patterson and Singh[277] similarly analyze and recognize gestures to identify cases of cerebral palsy in infants.

Activity-aware interventions are recently being designed and tested for their ability to provide customized assistance to maintain and improve health. These include the work of Hodges et al.[278] on the SenseCam project, the automated prompting work of Mihailidis et al. for hygiene activities[279–280], the activity prediction-based prompting for general activities described by Holder and Cook[165], and the many commercially available products that monitor and encourage quality sleep and exercise[281–285].

O'Reilly provides an insightful look at context-awareness and the impact it has on the design of effective applications[286]. Horvitz and Krumm[287] provide details on automatic routine services that suggest diversions based on a user's navigational context. Chen et al.[288] investigate the relationship between activities and resource consumption and Youngblood et al.[289] propose activity-sensitive methods for automating interactions with a physical environment. An introduction to pervasive computing and its use for pervasive gaming is provided by Chamberlain et al.[290]

Activity-aware security is an area of investigation that has just started to receive attention. The computer vision-based monitoring of human-vehicle interactions was proposed by Ryoo et al.[291] Feese et al.[292] introduce the use of group movement patterns and action monitoring to guide firefighters in emergency situations.

Cook et al. generated radial basis graphs to describe activities.[5] The maximal information coefficient, introduced by Reshef et al.[293], is based on the idea that if a relationship exists between two variables then a grid can be drawn on the scatter plot of the two variables that partitions the data to encapsulate the relationship. The largest possible mutual information of pairs of locations is computed for grids of varying resolution, the values are normalized, and the largest values for any resolution grid are used as the MIC value. The daily timeline of activity visualization is generated by the Moves app at moves-app.com[294]. Activity density maps have been designed used by Wang et al.[295] to analyze health status and trends for older adults in a retirement community. Visualization of circadian activity was performed by Haley et al.[296] Frank et al.[297] investigated the automatic generation and have demonstrated the ideas using data from the Nokia mobile Data Challenge.

With the ubiquitous availability of sensors on mobile devices as well as environmental sensors, a number of researchers have pursued an activity learning approach

to understanding human behavior and social dynamics. Candia et al.[298] design a theoretical model of human mobility and social connections based on mobile phone data and Cook et al.[299] analyze the distribution of time based on tracked activities in smart home environments. Dong et al.[232] investigate the connection between human behavior and social relationships using mobile phone data and Wang et al.[300] using mobile phone data to address the question of how individual mobility patterns impact the formation of social networks.

10

The Future of Activity Learning

Activity learning has made tremendous strides. Techniques for modeling, recognizing, discovery, and predicting activities from sensor data were virtually unmentioned until the past ten years. Since that time, researchers have been able to provide a foundation for these capabilities and test them in real-world settings. Thanks to the advent of ubiquitous sensing, activity-aware applications are now deployed and are commercially available to anybody with an Internet connection or a mobile device. Sensors of all shapes and sizes are combining forces with machine learning techniques to translate our physical world into a digital world. Here we put forward some challenges that activity learning can tackle next as well as ideas for how researchers can utilize these systems once they are fully operational.

Learning Activity Vocabularies When activity-based sensor data are collected from multiple sources, differences begin to emerge not only because of differences in users and physical settings, but also because the activity labels and annotation styles vary from one data source to another. Lists of routine human activities, such as those described in Chapter 2, have been published. However, even these contain inconsistencies in activity terms and descriptions. Activities are the vocabulary that we can use to express human behavior. An activity vocabulary standard can be defined and made available to the research community, which would facilitate sharing of common datasets and comparing alternative learning techniques. However, another interesting direction is to automate the construction of an activity taxonomy from sensor data. Such a taxonomy or ontology can provide a way to not only list activity names but relate them to each other as abstractions, variations, synonyms, and antonyms.

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

Automating the ontology formation also provides a way to relate activity labels with corresponding sensor event sequences.

Sensor Selection As Chapter 3 shows, a wealth of sensors of varying types, precision, cost, and size are available for capturing activity information. What is not always well understood is the sensors that are needed to fully capture and model a particular activity. A great density of sensors and sensor types provides fine-grained information on exactly where, when, and how activities are performed. On the other hand, the addition of sensors imposes more cost, more energy consumption, and more data processing requirements. The challenge of sensor selection will be to identify or rank the sensor types, locations, sampling rates, and other parameters that are needed to optimize the quality of the learned activity while minimizing the number of sensors and amount of sensor data that needs to be captured.

Fusing Sensor Data with Other Sources of Information Advances in pervasive computing, sensors, and wireless networking make it possible to tap these sources of information to learn models of activity in a nonintrusive manner. However, other sources of information are also available to help describe activities. Dictionaries provide definitions of activities. Web pages and literary sources allude to activities or reference them by name in the context of a larger plot. Social networks capture dense information on human actions, interactions, and locations on a regular basis. Activity learning algorithms can make greater use of information sources outside sensor events to learn models of activities and to tie external sources of knowledge with the information that is automatically gleaned from sensors. By harnessing the power of these rich data sources, we can not only better model and recognize activities but we can also observe how well automatically-sensed actions and activities align with people's perception of human behavior.

Inferring Intent from Action Sensors provide information about the concrete, the observed actions in a physical world. What cannot be directly observed, however, is a person's intent, beliefs, or goals. Researchers postulate that these can be inferred from action and can be determined even more accurate from the person's context derived from observing a lot of activities over a longer period of time. An interesting challenge would be to derive BDIs, or models of belief-desire-intention, from sensor data and learned activities. This would provide new insights on human psychology. It can also be used to provide services that move from context-awareness and activity-awareness to intent-awareness and makes the services even more sensitive to the needs of the individual who is being served. For example, an activity-aware home may recognize that Bob performs aerobic exercises each morning. If the home understands that Bob's intent is to reduce body fat for a healthier lifestyle then the home will be able to supply intent-aware suggestions and services, enhancing the activity-aware services that are currently available. For example, the house might also recognize when Bob is eating and recommend that he swap the butter he currently puts on his toast with preserves, in order to help Bob achieve his overall goal.

Identifying Critical Events Identifying anomalies that indicate critical events is a huge challenge. Once a rich set of activities is learned, the learned models provide a basis for detecting outliers or situations that do not fit the normal activity situation. Research is needed to determine which of these situations is life-threatening, which is safety endangering, which are system malfunctions, and which are simply part of the complex fabric of an individual's life. A system that can detect and prevent a potential drowning in a swimming pool is invaluable. A system that calls for emergency assistance when a dog jumps into the pool to retrieve a toy, however, is a significant annoyance. Not all anomalous events are critical events and research can provide ways to map detected anomalies into appropriate categories and corresponding reactions.

Discovering Insights on Human Behavior Now that activity learning is becoming more mature and therefore more robust, the field is open for researchers to understand human behavior on a non-invasive manner. Researchers have long questioned the ecological validity of studying psychology and sociology based on laboratory tests and self-report information. Because activities can be learned and sensed in everyday environments without changes to normal routines, theories that have been espoused by these communities can be confirmed or rejected based on automatically observed behavior. We particularly look forward to researchers proposing new theories of behavior from automatically learned activities, some of which may be surprising or inconsistent with the models that have been postulated in the past from more limited information.

Scaling Activity Learning As research has been directed toward core activity learning techniques and has done so with increasingly successful results over the last decade, it is now time to consider scaling activity learning. The notion of scalability here can take on multiple dimensions. First, it refers to the ability of activity learning systems to maintain a desired level of efficiency and functionality as the number of individuals, activities, sensors, and settings increases by orders of magnitude. Generally, an increase in a system dimension adds capability to the system while incurring associated overheads. These capabilities and overheads can be measured by price tags, human time and attention, computation and communication power, storage capacity, accessibility, responsiveness, or other valuable resource usage. An activity learning algorithm that is scalable provides a rate of increase in capability that is greater than the rate of increase in overhead.

Not all activity learning applications are large scale or require large-scale resources. However, the ability to scale activity learning does open up ideas for large-scale use of the models. The Internet of Things (IOT) implies that every tagged object could be part of a very large-scale connected system across the globe. While current activity learners track individuals and analyze their activity patterns, future activity learning systems can use IOT and other pervasive computing technologies to scale learn and analyze activity performance and patterns for neighborhoods, communities, or even countries.

Privacy Preserving Activity Learning Throughout this book we have emphasized the need for quality and quantity of data from which activities can be learned. Clearly, what is a treasure trove for researchers can be viewed as an invasion of privacy for some people and a potential security risk for others. Data can be anonymized by removing names, addresses, and other discriminating features. It can be encrypted, password protected, and shared only in the aggregate. However, lessons learned in machine learning and data mining teach us that personal information can be retrieved from information that at the surface appears general or innocuous. In her testimony to the Department of Homeland Security, Latanya Sweeney[301] reported that she was able to retrieve an individual's private information from "anonymized" medical data using only a birth date, gender, and zip code. In fact, she stated that 87% of the US population is uniquely identified by these three fields. In order to make guarantees of security and privacy, activity learning researchers need to be aware that their "cleaned" data may not necessarily be anonymous data. We also need to investigate what inferences can be made from learned activity models that are beneficial for others and what can potentially be used to harm others. Understanding these issues in greater depth will allow us to better reason about how data can be securely collected, stored, analyzed, shared, and used.

What if Activity Learning Works? Advances in activity learning techniques rely on advances in machine learning and data mining as well as advances in wireless networks, sensors, and embedded systems that facilitate data collection. Activity learning is not an island unto itself. We rely on maturing in other areas to boost our own field. Similarly, successes in activity learning can be used to improve technologies in other areas. For example, since we know that gestures and actions accompany speech, action recognition algorithms can be used to improve speech recognition in everyday settings.

Activity learning can also boost the performance of sensor networks. Specifically, networks might make use of learned activity models and predicted activity occurrences to only send events that are inconsistent with the predicted day's schedule. As a result, battery life is saved, network traffic is reduced, and less new data are collected, which needs to be stored and analyzed.

There is a rich opportunity here for researchers to explore grand challenge problems in other fields of interest and see how activity learning techniques can progress these other fields. Pick your favorite activity learning grand challenge or application and get started.

Appendix

A.1 SAMPLE ACTIVITY DATA

Sensor Data Collected While Individual Performs a Hand Washing Activity

Time	Sensor ID	Sensor Message
13:53:08.061116	M017	ON
13:53:08.411453	ARM	(1541,1157,2252,1774,1805,1852)
13:53:08.429769	HIP	(721,2117,1638,1894,1732,2047)
13:53:09.239997	HANDSOAP	MOVED
13:53:09.221860	ARM	(2141,1153,2107,1839,1838,1897)
13:53:09.932730	HIP	(973,1867,1701,1786,1620,1703)
13:53:10.426353	HIP	(1156,1979,1663,1960,1754,1931)
13:53:10.734417	ARM	(2136,1139,2135,1836,1831,1898)
13:53:11.037931	WATER	0.122072
13:53:11.222313	ARM	(2135,1130,2121,1838,1830,1896)
13:53:11.451506	HIP	(982,2059,1623,1834,1802,1887)
13:53:11.890754	M017	OFF
13:53:12.433339	HIP	(1027,1857,1548,1819,1805,1872)
13:53:12.534139	ARM	(974,1741,1877,1805,1782,1894)

(*continued*)

Sensor Data Collected While Individual Performs a Hand Washing Activity

Time	Sensor ID	Sensor Message
13:53:13.413594	HIP	(679,575,2058,1762,1868,1693)
13:53:13.510662	WATER	0.263559
13:53:13.891037	ARM	(964,1555,1977,1884,1843,1862)
13:53:13.942329	HIP	(1130,1724,1496,1875,1623,1894)
13:53:14.387867	ARM	(1010,1801,1923,1908,1842,1923)
13:53:14.443325	HIP	(893,2219,1571,1748,1936,1831)
13:53:15.385983	ARM	(1505,1822,1047,1883,1767,1788)
13:53:15.931949	HIP	(932,2164,1625,1827,1865,1871)
13:53:16.033640	WATER	0.29059
13:53:16.381560	ARM	(476,1568,2228,1207,1723,1823)
13:53:16.427890	HIP	(953,2205,1624,1838,1821,1889
13:53:17.916500	HIP	(969,2183,1606,1838,1822,1898)
13:53:17.983717	ARM	(945,1556,1626,1794,2037,1953)
13:53:18.372664	ARM	(957,1157,1185,2005,1853,1832)
13:53:18.430836	HIP	(964,2193,1641,1836,1831,1893)
13:53:18.483483	WATER	0.320005
13:53:19.373567	ARM	(1319,1364,1751,2134,1892,1744)
13:53:19.907770	HIP	(984,2182,1637,1838,1823,1898)
13:53:20.426566	HIP	(959,2178,1635,1838,1828,1895)
13:53:20.486367	ARM	(792,1113,621,1847,1818,1864)
13:53:20.965551	WATER	0.278715
13:53:21.371230	ARM	(1162,2060,1778,1831,1891,1930)
13:53:21.616903	HIP	(978,1889,1577,1835,1810,1892)
13:53:22.176705	POWER	2375
13:53:22.370082	ARM	(643,1259,1061,2296,1993,1774)
13:53:22.608580	HIP	(981,1981,1577,1837,1787,1906)
13:53:23.104906	HIP	(994,1993,1594,1828,1866,1869)
13:53:23.351881	ARM	(948,1542,1615,1539,1732,1776)
13:53:23.438208	WATER	0.3002
13:53:24.103194	HIP	(999,1951,1644,1851,1816,1896)
13:53:24.361415	ARM	(1337,1725,2246,1663,1782,2085)
13:53:25.005343	HIP	(977,1990,1582,1849,1824,1898)

Sensor Data Collected While Individual Performs a Hand Washing Activity

Time	Sensor ID	Sensor Message
13:53:25.353191	ARM	(1709,1214,2353,2161,2210,1230)
13:53:25.586322	M018	ON
13:53:25.903958	WATER	0.102502
13:53:26.344369	ARM	(1531,2180,2076,2003,1536,1697)
13:53:26.473571	M017	ON
13:53:26.496878	HIP	(1141,2233,1423,1787,1701,1898)
13:53:27.329380	ARM	(1085,1317,1884,2145,1875,2032)
13:53:27.487384	HIP	(1155,1959,1588,1891,1884,1892)
13:53:28.095657	HIP	(951,2080,1564,1803,2099,1839)
13:53:28.365902	ARM	(1329,1663,1596,2074,1900,1907)
13:53:28.44001	WATER	0.0313677
13:53:29.206744	HIP	(989,2106,1588,1846,1793,1901)
13:53:29.323795	ARM	(1147,2099,1101,2010,1774,1720)
13:53:30.321485	ARM	(1225,2213,2648,1922,1958,1857)
13:53:30.489946	HIP	(986,2020,1543,1843,1854,1890)
13:53:30.864281	HANDSOAP	STILL
13:53:31.01971	M017	OFF
13:53:31.326011	ARM	(1500,2685,1435,1968,1714,2017)
13:53:31.466857	HIP	(1004,2017,1546,1833,1841,1888)
13:53:32.321271	ARM	(586,1943,2281,1649,1766,1707)
13:53:32.469952	HIP	(1002,2013,1560,1840,1819,1897)
13:53:33.330513	ARM	(834,2327,1860,1165,1383,1806)
13:53:33.476859	HIP	(999,2089,1566,1858,1867,1883)
13:53:34.163966	POWER	2387
13:53:34.312662	ARM	(1442,2048,1502,1631,1736,1759)
13:53:34.454890	HIP	(999,2011,1542,1843,1777,1919)
13:53:35.080271	HIP	(1001,1988,1529,1839,1774,1908)
13:53:35.312974	ARM	(1187,1762,2574,1892,1888,2022)
13:53:36.301057	ARM	(1193,2220,2016,1362,1670,1703)
13:53:36.441309	HIP	(1011,2063,1586,1844,1837,1898)
13:53:37.298464	ARM	(1358,1265,2341,1932,1943,1826)

(*continued*)

Sensor Data Collected While Individual Performs a Hand Washing Activity

Time	Sensor ID	Sensor Message
13:53:37.444395	HIP	(1012,1945,1588,1834,1864,1885)
13:53:38.295640	ARM	(1794,2054,2401,1989,1883,1842)
13:53:38.435514	HIP	(967,2060,1580,1833,1788,1860)
13:53:39.283410	ARM	(1035,1331,2273,1857,1841,1974)
13:53:39.443636	HIP	(1050,2237,1520,1785,1748,1895)
13:53:39.988195	M017	ON
13:53:40.004498	M016	ON
13:53:40.006357	M018	OFF
13:53:40.279996	ARM	(1296,938,2073,1794,1718,2163)
13:53:40.36503	M015	ON
13:53:40.462104	HIP	(1099,2126,1546,1939,1957,1859)
13:53:41.107873	M014	ON
13:53:41.134085	M017	OFF
13:54:24.361415	ARM	(1337,1725,2246,1663,1782,2085)
13:54:25.005343	HIP	(977,1990,1582,1849,1824,1898)
13:54:25.353191	ARM	(1709,1214,2353,2161,2210,1230)
13:54:25.903958	WATER	0.102502
13:54:26.344369	ARM	(1531,2180,2076,2003,1536,1697)
13:54:26.473571	M017	ON

A second activity that is captured is that of sweeping. The individual gathers sweeping supplies including a broom, duster and dustpan from the kitchen closet. The supplies are used to sweep the kitchen floor and dust surfaces in the dining room and living room. The following table shows the sensor data that is collected during this activity.

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:50:13.102682	SS001	MOVED
12:50:13.149904	SG024	(1019,1921,1520,1850,1800,1898)
12:50:13.367222	SG023	(1584,2402,2318,2040,1838,1736)
12:50:14.128257	SG024	(973,1951,1545,1839,1918,1900)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:50:14.217971	SS003	MOVED
12:50:14.357487	SG023	(948,1851,1837,1886,1820,2028)
12:50:15.129119	SG024	(1055,1745,1792,1840,1814,1872)
12:50:15.873883	SG023	(926,1867,2119,1751,1853,1863)
12:50:16.134036	SG024	(1047,2137,1487,1819,1594,1992)
12:50:16.235281	SS002	MOVED
12:50:16.36758	SG023	(1055,1677,2182,1705,1613,1862)
12:50:17.001771	M018	ON
12:50:17.11736	SG024	(997,1970,1625,1752,1438,1907)
12:50:17.345406	SG023	(1189,1935,2232,1840,1682,1813)
12:50:18.119112	SG024	(1072,2052,1559,1880,1796,1949)
12:50:18.341054	SG023	(1071,1889,1978,1780,1721,1982)
12:50:18.452039	M017	ON
12:50:18.790212	SS010	MOVED
12:50:19.103787	SG024	(1159,2026,1598,1863,1744,1951)
12:50:19.349545	SG023	(1043,1330,1688,1676,1825,1872)
12:50:20.101233	SG024	(986,1966,1573,1838,1727,1921)
12:50:20.334268	SG023	(1007,1674,1700,1702,1868,1916)
12:50:20.741536	SS010	STILL
12:50:21.097739	SG024	(981,2074,1527,1853,1724,1930)
12:50:21.342635	SG023	(1208,2346,1888,2217,1234,1483)
12:50:21.895284	BURNER	2.8227
12:50:22.090522	SG024	(1149,2003,1701,1857,1615,1949)
12:50:22.339887	SG023	(1134,2594,2088,1894,1845,1872)
12:50:22.412985	SS010	MOVED
12:50:22.8739	SS001	STILL
12:50:23.103554	SG024	(900,1872,1612,1865,1882,1860)
12:50:23.325268	SG023	(1151,2326,2160,1792,1798,1878)
12:50:23.980949	SS003	STILL
12:50:24.085518	SG024	(931,1905,1697,1856,1851,1875)
12:50:24.439162	SG023	(1157,2437,2191,1926,1881,1840)

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:50:25.077601	SG024	(969,1914,1633,1845,1832,1947)
12:50:25.329807	SG023	(603,1626,1846,1872,1815,1943)
12:50:26.073557	SG024	(1041,1631,1742,1873,1960,1866)
12:50:26.311588	SG023	(467,1435,2252,1610,1958,1892)
12:50:27.021357	M017	OFF
12:50:27.039859	M018	OFF
12:50:27.092388	SG024	(884,1759,1883,1905,1943,1823)
12:50:27.312776	SG023	(953,2037,2179,1962,1864,1919)
12:50:28.072648	SG024	(982,1845,1581,1829,1824,1949)
12:50:28.261877	SS010	STILL
12:50:28.300587	SG023	(1366,2129,1983,1665,2015,1891)
12:50:29.073019	SG024	(905,1821,1662,1776,1793,1940)
12:50:29.304551	SG023	(435,1649,1860,1699,2014,1937)
12:50:30.075277	SG024	(1028,1801,1623,1819,1770,1892)
12:50:30.310112	SG023	(747,669,1725,1757,1632,2020)
12:50:31.064673	SG024	(986,1913,1582,1843,1856,1906)
12:50:31.290254	SG023	(725,14,2022,1721,1691,1903)
12:50:31.841793	BURNER	2.85582
12:50:32.055235	SG024	(992,1738,1801,1827,1850,1907)
12:50:32.408175	SG023	(670,1269,1487,2042,1388,2031)
12:50:32.6823	M018	ON
12:50:33.049097	SG024	(924,2032,1592,1856,1775,1894)
12:50:33.278794	SG023	(17,1172,2346,1834,1675,2079)
12:50:34.046108	SG024	(979,1935,1579,1870,1881,1881)
12:50:34.316605	SG023	(1199,2198,2222,1799,1971,1853)
12:50:34.355906	BURNER	2.82254
12:50:35.040684	SG024	(1058,1846,1535,1888,1846,1899)
12:50:35.27368	SG023	(1828,2227,1892,1987,1960,2029)
12:50:36.043684	SG024	(1044,1597,1640,1675,1857,1905)
12:50:36.265537	SG023	(1072,2072,2477,1769,1879,2011)
12:50:37.056012	SG024	(1210,1432,1905,1787,1854,1921)
12:50:37.266938	SG023	(1579,1851,2605,1998,1798,1775)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:50:38.036181	SG024	(974,1674,1743,1868,1890,1863)
12:50:38.26824	SG023	(1022,1987,2038,1621,1910,1735)
12:50:38.283261	M051	OFF
12:50:39.033099	SG024	(884,2066,1785,1909,1517,1872)
12:50:39.329853	M019	ON
12:50:39.762776	SG023	(1392,768,1986,1693,1738,2157)
12:50:40.020762	SG024	(1013,1849,1587,1805,1790,1870)
12:50:40.27761	SG023	(1526,1538,2145,1802,1741,1606)
12:50:41.02102	SG024	(861,1750,1730,1907,1897,1907)
12:50:41.384632	SG023	(1259,1315,2292,1748,1854,2319)
12:50:42.02552	SG024	(1009,1837,1604,1796,1840,1867)
12:50:42.242443	SG023	(1420,794,2219,1847,2043,1831)
12:50:43.014653	SG024	(999,1876,1597,1843,1808,1890)
12:50:43.256349	SG023	(1483,1384,2126,1869,1790,1612)
12:50:43.764732	M018	OFF
12:50:44.127062	SG024	(1046,1995,1529,1869,1941,1884)
12:50:44.242536	SG023	(1457,914,1881,1851,1719,2240)
12:50:45.024533	SG024	(891,1932,1900,1815,1898,1894)
12:50:45.23364	SG023	(1657,1558,2352,1810,2003,2072)
12:50:46.002748	SG024	(1044,1922,1521,1824,1831,1878)
12:50:46.2668	SG023	(1819,1254,2022,1953,1793,1728)
12:50:47.225258	SG023	(1765,1021,2066,1716,1835,1620)
12:50:47.501071	SG024	(928,1902,1584,1861,1981,1826)
12:50:48.238949	SG023	(1199,1030,2046,1776,1767,1909)
12:50:48.487187	SG024	(897,1815,1354,1866,1873,1844)
12:50:49.214601	SG023	(1534,1130,1892,1749,1834,1459)
12:50:49.494279	SG024	(1076,1887,1485,1843,1773,1928)
12:50:50.225629	M018	ON
12:50:50.245519	T005	20.5
12:50:50.262268	SG023	(1827,738,2135,1921,1843,1652)
12:50:50.509589	SG024	(1036,1737,1541,1832,1694,1919)

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:50:51.235981	SG023	(716,1010,2161,2018,2054,1726)
12:50:51.493873	SG024	(1038,2038,1436,1807,1881,1880)
12:50:51.610834	BURNER	2.84301
12:50:51.870885	M017	ON
12:50:52.205444	SG023	(1434,2771,1579,2084,1763,1953)
12:50:52.488461	SG024	(712,1849,1914,1761,1755,1940)
12:50:52.643684	M019	OFF
12:50:53.225553	SG023	(1288,2658,1971,1978,1635,1854)
12:50:53.469181	SG024	(1140,1855,1643,1680,1739,1805)
12:50:54.202976	SG023	(1188,2658,2158,1896,1809,1864)
12:50:54.467669	SG024	(1043,1991,1587,1815,1972,1907)
12:50:54.896864	M016	ON
12:50:55.35083	SG023	(316,1589,2065,1536,1814,1945)
12:50:55.483143	SG024	(1009,1622,1814,1852,1776,1902)
12:50:56.198372	SG023	(912,2060,1797,1913,1604,2152)
12:50:56.458544	SG024	(1163,1463,1609,1831,1846,1901)
12:50:56.774596	M018	OFF
12:50:57.19153	SG023	(1322,1974,2297,1891,2006,1956)
12:50:57.455761	SG024	(1153,1477,1622,1844,1895,1853)
12:50:58.021719	M017	OFF
12:50:58.210705	SG023	(1829,2553,1991,1919,1924,1848)
12:50:58.455453	SG024	(965,1851,1780,1877,1927,1881)
12:50:59.180823	SG023	(947,1787,2010,1752,1875,1923)
12:50:59.456617	SG024	(906,1979,1706,1846,1799,1872)
12:51:00.178204	SG023	(938,1960,2249,1812,2075,1898)
12:51:00.463521	SG024	(908,1952,1947,1828,1798,1857)
12:51:00.900278	M017	ON
12:51:01.174687	SG023	(875,1917,2437,1799,1922,1758)
12:51:01.441858	SG024	(1029,1891,1506,1810,1705,1954)
12:51:02.174299	SG023	(1473,2146,2127,1672,1886,1804)
12:51:02.434144	SG024	(922,1912,1766,1827,1817,1917)
12:51:03.163076	SG023	(1362,2256,2085,1770,1906,1714)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:51:03.288928	M017	OFF
12:51:03.459142	SG024	(942,1808,1762,1817,1794,1915)
12:51:04.130247	M016	OFF
12:51:04.199881	SG023	(1036,1838,1945,1754,1896,1786)
12:51:04.429023	SG024	(983,1959,1465,1907,1778,1894)
12:51:04.503612	M017	ON
12:51:05.176608	SG023	(1094,224,2022,1693,1827,1674)
12:51:05.422281	SG024	(1047,1713,1715,1811,1827,1860)
12:51:06.154021	SG023	(540,442,1910,1747,1711,1740)
12:51:06.425596	SG024	(951,1678,1808,1873,1886,1792)
12:51:06.981236	POWER	1832
12:51:07.152281	SG023	(714,2419,2006,2117,1558,2181)
12:51:07.435855	SG024	(1012,1852,1832,1703,1764,1968)
12:51:08.143708	SG023	(979,2600,1850,1526,1763,1622)
12:51:08.414795	SG024	(951,1903,1514,1834,1809,1947)
12:51:08.914897	SG024	(1107,1765,1540,1822,1798,1914)
12:51:09.160782	SG023	(1302,2369,2111,1874,1807,1758)
12:51:10.13908	SG023	(1134,1947,1959,1766,1847,1991)
12:51:10.428903	SG024	(1091,1693,1629,1845,1807,1894)
12:51:11.133432	SG023	(1049,2235,2015,1770,1904,1766)
12:51:11.402919	SG024	(1107,1667,1662,1836,1838,1868)
12:51:12.132326	SG023	(893,1979,2519,1825,1921,1840)
12:51:12.397958	SG024	(932,2150,1767,1787,1654,1808)
12:51:13.13401	SG023	(1921,1224,2238,1548,1627,2182)
12:51:13.394112	SG024	(1064,1980,1548,1789,1664,1948))
12:51:14.125642	SG023	(1449,1342,2179,1839,1840,2324)
12:51:14.406158	SG024	(964,1949,1636,1827,1824,1878)
12:51:15.121459	SG023	(1507,1548,2324,1967,1900,2327)
12:51:15.382876	SG024	(1062,1939,1680,1834,1964,1900)
12:51:16.128527	SG023	(562,916,1638,1852,1906,1622)
12:51:16.381411	SG024	(948,2216,1560,1837,1846,1883)

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:51:17.225709	SG023	(1359,1352,2195,1619,1896,1739)
12:51:17.376611	SG024	(883,2074,1824,1793,1672,1963)
12:51:18.107301	SG023	(1503,1895,2361,2020,1916,2125)
12:51:18.386369	SG024	(1034,1832,1619,1852,1835,1914)
12:51:19.110443	SG023	(1115,1501,1504,2099,1881,2223)
12:51:19.371315	SG024	(897,2144,1556,1892,1804,1825)
12:51:20.098942	SG023	(872,1809,2122,1719,1760,2221)
12:51:20.363866	SG024	(1008,1852,1573,1803,1794,1935)
12:51:21.009062	POWER	1817
12:51:21.099004	SG023	(1399,1254,1880,1766,1847,1567)
12:51:21.374107	SG024	(987,1795,1616,1826,1847,1891)
12:51:22.095113	SG023	(1826,1152,1719,2065,1798,2066)
12:51:22.362273	SG024	(953,1844,1658,1835,1874,1912)
12:51:23.088291	SG023	(1476,1624,2605,1633,1831,1414)
12:51:23.375772	SG024	(825,1686,1843,1814,1517,1921)
12:51:24.313694	SG023	(90,1333,1803,1768,1699,1934)
12:51:24.37567	SG024	(998,1857,1595,1832,1885,1866)
12:51:25.081034	SG023	(973,2208,1728,2162,1583,2239)
12:51:25.346117	SG024	(1005,1965,1531,1843,1867,1935)
12:51:25.91867	M018	ON
12:51:26.090662	SG023	(1026,1842,2236,1917,1816,2105)
12:51:26.360582	SG024	(994,1767,1744,1825,1882,1906)
12:51:27.077397	SG023	(1088,1550,2731,1672,1944,1813)
12:51:27.345018	SG024	(848,1866,1931,1902,1741,1906)
12:51:28.074122	SG023	(929,2294,2259,1990,1776,2088)
12:51:28.340392	SG024	(1032,1899,1610,1839,1795,1890)
12:51:29.067173	SG023	(384,1393,1811,1727,1976,1937)
12:51:29.350419	SG024	(1082,1787,1873,1897,1866,1928)
12:51:30.062208	SG023	(1087,1754,2076,1839,2066,1998)
12:51:30.330812	SG024	(998,1866,1733,1865,1853,1870)
12:51:31.077879	SG023	(670,1251,2298,1741,1879,1965)
12:51:31.324345	SG024	(1009,1590,1634,1831,1794,1892)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:51:32.055785	SG023	(1486,2258,1959,2034,1918,2350)
12:51:32.366104	SG024	(998,1903,1890,1879,1866,1879)
12:51:32.513554	M017	OFF
12:51:33.050537	SG023	(896,1855,2366,1755,1873,1859)
12:51:33.336708	SG024	(1101,1739,1612,1864,1909,1878)
12:51:34.043553	SG023	(802,1707,1482,1821,1312,2159)
12:51:34.309783	SG024	(1078,1922,1480,1790,1502,1973)
12:51:35.318368	SG024	(1010,1814,1774,1831,1859,1900)
12:51:35.551276	SG023	(1615,625,2343,1813,1851,1757)
12:51:36.288103	M017	ON
12:51:36.31913	SG024	(1400,2238,1812,1900,1788,1895)
12:51:36.537747	SG023	(1166,1560,2306,1551,1599,1843)
12:51:37.0474	SG023	(1982,1504,2499,1795,1737,1877)
12:51:37.305369	SG024	(1035,1929,1517,1834,1774,1907)
12:51:37.983376	SS001	MOVED
12:51:38.061858	SG023	(1256,1251,2345,1729,1854,2064)
12:51:38.313264	SG024	(1021,1907,1580,1844,1847,1897)
12:51:39.041053	SG023	(1853,2761,2039,1744,1862,1963)
12:51:39.291345	SG024	(957,2009,1601,1866,1859,1912)
12:51:40.021228	SG023	(1012,2294,2357,1851,1827,1887)
12:51:40.284949	SG024	(1039,1878,1472,1810,1710,1928)
12:51:41.148851	SG023	(1751,2259,1877,2467,1505,1976)
12:51:41.314479	SG024	(1411,1862,1891,1895,1681,2052)
12:51:42.041452	POWER	1807
12:51:42.095218	SG023	(1931,936,2197,1856,1883,1802)
12:51:42.278838	SG024	(1108,1626,1678,1830,1820,1911)
12:51:43.017297	SG023	(1986,1765,2486,1888,2013,1847)
12:51:43.285772	SG024	(1095,1539,1580,1817,1807,1907)
12:51:44.013655	SG023	(1617,1027,1843,1981,1749,1847)
12:51:44.127388	SS002	STILL
12:51:44.233186	SS002	MOVED

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:51:44.323562	SG024	(1225,1513,1616,1822,1804,1917)
12:51:45.008775	SG023	(1854,1235,1864,2256,2051,2009)
12:51:45.266559	SG024	(1073,1629,1662,1842,1833,1883)
12:51:45.576156	M017	OFF
12:51:46.006972	SG023	(1785,1404,1776,1892,1977,1864)
12:51:46.263199	SG024	(1028,1586,1585,1901,1879,1888)
12:51:47.111514	M051	ON
12:51:47.278632	SG024	(1112,1749,1689,1845,1814,1936)
12:51:47.503353	SG023	(1663,1162,1673,1835,1858,1896)
12:51:48.003632	SG023	(1599,1173,1763,1854,1839,1895)
12:51:48.266343	SG024	(1015,1827,1599,1833,1808,1891)
12:51:49.256484	SG024	(1140,2259,1919,1736,1824,2047)
12:51:49.487314	SG023	(1677,929,2131,1645,1986,1828)
12:51:50.272369	SG024	(1128,1486,1524,1850,1807,1886)
12:51:50.499137	SG023	(1344,1511,1708,2136,1962,1977)
12:51:51.054932	SS010	MOVED
12:51:51.248526	SG024	(1128,1588,1650,1822,1931,1792)
12:51:51.485558	SG023	(1625,1194,1727,1987,2013,1780)
12:51:51.790467	M017	ON
12:51:52.259451	SG024	(972,1970,1613,1852,1812,1922)
12:51:52.479607	SG023	(1048,2425,2131,2141,1686,1795)
12:51:52.586872	M051	OFF
12:51:53.033222	SS010	STILL
12:51:53.243974	SG024	(1003,1873,1589,1834,1815,1902)
12:51:53.508002	SG023	(1065,2066,2009,1960,1824,1858)
12:51:54.247584	SG024	(993,1851,1578,1851,1836,1881)
12:51:54.480738	SG023	(1475,2131,2217,1855,1798,1937)
12:51:55.228119	SG024	(993,1947,1609,1834,1847,1898)
12:51:55.460648	SG023	(1198,1758,2564,1949,2011,2309)
12:51:55.948749	SS002	MOVED
12:51:56.501583	SG023	(1853,1459,2496,2115,1971,1805)
12:51:56.95919	SG023	(911,1910,2162,1849,1906,1974)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:51:57.08999	SG024	(996,2083,1515,1938,2085,1796)
12:51:57.171061	M018	OFF
12:51:57.455376	SG023	(331,2362,2423,1125,1763,1632)
12:51:57.480745	SS003	MOVED
12:51:58.01167	SS010	MOVED
12:51:58.085978	SG024	(931,2041,1621,1841,1630,1928)
12:51:58.458655	SG023	(1324,2875,2453,2489,1917,1826)
12:51:58.706651	M016	ON
12:51:59.089929	SG024	(760,1903,1312,1758,1889,1812)
12:51:59.452562	M015	ON
12:51:59.494309	SG023	(1492,1490,3096,2087,1668,1914)
12:52:00.087492	SG024	(1125,1917,1464,1853,1788,1900)
12:52:00.449336	SG023	(699,3270,2505,1930,1941,2346)
12:52:00.557576	SS001	STILL
12:52:01.085493	SG024	(923,1894,1303,1933,1867,1912)
12:52:01.452298	SG023	(1583,2555,2292,1866,1954,1842)
12:52:01.685244	SS010	STILL
12:52:01.799029	SS002	STILL
12:52:02.075131	M008	ON
12:52:02.090358	SG024	(1055,1965,1275,1878,1604,2053)
12:52:02.442863	SG023	(1715,2391,2441,1818,1835,1996)
12:52:02.461554	M017	OFF
12:52:03.071154	SG024	(983,2027,1833,1797,1766,1907)
12:52:03.445668	SG023	(1480,2665,2249,1999,1734,2121)
12:52:04.065556	SG024	(957,1975,1497,1791,1687,1949)
12:52:04.194288	M016	OFF
12:52:04.431763	SG023	(958,949,1787,1604,1891,2173)
12:52:04.592832	M009	ON
12:52:04.931277	SG023	(999,1166,1796,1842,1719,1835)
12:52:05.058684	SG024	(971,2053,1788,1834,1658,1881)
12:52:05.427128	SG023	(1172,2063,1768,1422,2258,1746)

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:52:06.054524	SG024	(992,1765,1742,1794,1730,1897)
12:52:06.304371	M014	ON
12:52:06.422208	SG023	(1470,2563,2715,1663,1923,1635)
12:52:07.051292	SG024	(1055,1833,1556,1878,1872,1932)
12:52:07.41329	SG023	(2290,2564,2788,2018,1785,1915)
12:52:08.042582	SG024	(1017,1969,1673,1798,1894,1871)
12:52:08.292017	M009	OFF
12:52:08.415032	SG023	(2632,2852,2365,1663,1698,1953)
12:52:08.989256	M008	OFF
12:52:09.039674	SG024	(1022,1968,1589,1840,1797,1902)
12:52:09.40642	SG023	(1453,2263,2583,2168,1732,1880)
12:52:10.038467	SG024	(1082,1815,1518,1795,1823,2012)
12:52:10.422051	SG023	(1775,1098,2071,1998,1879,1824)
12:52:11.042378	SG024	(1061,1760,1663,1814,1866,1821)
12:52:11.406101	SG023	(2488,2201,2387,1919,1565,1741)
12:52:12.027539	SG024	(995,2020,1802,1864,1740,1859)
12:52:12.407109	SG023	(2314,2485,2347,1954,2146,1769)
12:52:13.027395	SG024	(991,2115,1578,1886,1765,1887)
12:52:13.082596	M013	ON
12:52:13.39379	SG023	(2394,2966,2075,1541,1564,1892)
12:52:14.04242	SG024	(948,2123,1667,1860,1724,1939)
12:52:14.40596	SG023	(2804,3617,1920,1615,1977,1912)
12:52:15.030923	SG024	(1018,1860,1736,1820,1835,1911)
12:52:15.404709	SG023	(2381,2645,2399,1994,1769,1847)
12:52:15.645982	BURNER	2.82254
12:52:16.02657	SG024	(884,1936,1620,1943,1784,1979)
12:52:16.046891	M012	ON
12:52:16.386827	SG023	(1953,182,2153,1957,2067,1591)
12:52:16.414116	M015	OFF
12:52:17.037345	SG024	(1066,1970,1513,1828,1590,1949)
12:52:17.398987	SG023	(1310,2833,2550,1703,1906,1510)
12:52:17.429858	M014	OFF

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:52:17.467905	M013	OFF
12:52:18.131726	SG024	(1101,1678,1658,1728,1714,1907)
12:52:18.375879	SG023	(2066,2917,2495,2007,1755,1854)
12:52:19.326193	M011	ON
12:52:19.377863	SG023	(2658,3054,2276,2039,1815,1913)
12:52:19.505964	SG024	(990,1858,1647,1833,1897,1897)
12:52:20.381277	SG023	(2285,2696,2196,1684,1584,1939)
12:52:20.738875	SG024	(951,1935,1719,1828,1800,1936)
12:52:21.010784	SG024	(1103,1824,1723,1830,1795,1911)
12:52:21.361639	SG023	(2304,2127,2384,1577,1838,2039)
12:52:22.37875	SG023	(2017,2610,2429,2735,1694,2409)
12:52:22.428486	M012	OFF
12:52:22.501176	SG024	(1309,1818,1421,1904,1660,1877)
12:52:23.356798	SG023	(1168,1496,2366,1806,1744,2034)
12:52:23.499615	SG024	(934,1645,1754,1801,1771,1994)
12:52:24.351336	SG023	(924,1631,2302,1774,1691,1860)
12:52:24.485291	SG024	(946,1983,1447,1782,1633,1945)
12:52:25.349275	SG023	(675,395,2048,1783,1490,935)
12:52:25.416492	M013	ON
12:52:25.47689	SG024	(889,2226,1633,1860,1651,1888)
12:52:26.023408	M011	OFF
12:52:26.358676	SG023	(1861,18,1738,1871,1887,1286)
12:52:26.49021	SG024	(1143,1745,1513,1883,1963,1790)
12:52:26.87429	M014	ON
12:52:27.339116	SG023	(2207,2170,3173,2590,1913,1522)
12:52:27.495627	SG024	(1124,2285,1692,1812,1991,1887)
12:52:28.031882	BURNER	2.85247
12:52:28.33979	SG023	(1247,959,2416,2028,1931,1642)
12:52:28.46798	SG024	(1047,1879,1499,1826,1875,1849)
12:52:28.988908	M009	ON
12:52:29.326138	SG023	(1368,1709,2095,1800,1968,2021)

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:52:29.376079	M008	ON
12:52:29.479725	SG024	(1041,1850,1568,1629,1719,1856)
12:52:30.170618	M007	ON
12:52:30.330385	SG023	(1019,1672,2306,1695,1989,2039)
12:52:30.453829	SG024	(899,1556,1041,1761,1855,1810)
12:52:30.773439	M013	OFF
12:52:31.145957	M006	ON
12:52:31.326063	SG023	(432,1616,1669,1796,2101,2216)
12:52:31.458844	SG024	(1050,1973,1495,1893,1797,1904)
12:52:31.534146	M014	OFF
12:52:32.44616	SG024	(948,1892,1545,1854,1715,1930)
12:52:32.823622	SG023	(1667,1377,2762,1837,2247,2257)
12:52:33.397764	M009	OFF
12:52:33.451156	SG024	(847,1908,1713,1829,1803,1882)
12:52:33.841357	SG023	(1350,1095,1504,1230,2197,2306)
12:52:34.312475	SG023	(1237,1430,2249,2107,1897,1937)
12:52:34.441164	SG024	(982,1981,1518,2093,2093,1791)
12:52:34.550964	M008	OFF
12:52:35.31027	SG023	(2017,612,2434,1789,1838,2228)
12:52:35.442897	SG024	(1074,1849,1773,1911,1763,1959)
12:52:35.558734	M005	ON
12:52:35.573569	M007	OFF
12:52:36.311571	SG023	(976,1285,2118,2158,2201,2101)
12:52:36.450724	SG024	(957,1975,1570,1842,1815,1881)
12:52:37.308765	SG023	(816,2308,2076,1885,1774,1718)
12:52:37.44497	SG024	(1005,1876,1546,1829,1831,1894)
12:52:38.006046	BURNER	2.83207
12:52:38.300657	SG023	(723,1370,2521,1757,1590,2001)
12:52:38.4295	SG024	(1008,1795,1539,1817,1675,1978)
12:52:39.288592	SG023	(1157,1186,2293,1820,1601,1611)
12:52:39.42975	SG024	(1025,1918,1548,1784,1739,1898)
12:52:39.492972	M006	OFF

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:52:40.286397	SG023	(1355,3071,2469,1829,2007,1839)
12:52:40.417349	SG024	(1051,1655,1295,1980,1647,1937)
12:52:41.310924	SG023	(2413,2628,2518,2012,1838,1801)
12:52:41.432209	SG024	(1076,1928,1485,1857,1856,1907)
12:52:42.442085	SG023	(1813,2440,2497,1893,1645,1770)
12:52:42.460075	SG024	(723,2094,1840,1886,1913,1831)
12:52:42.747785	M006	ON
12:52:43.282372	SG023	(1347,2335,2123,1755,1808,1893)
12:52:43.407015	SG024	(1008,2266,1528,1762,1736,1857)
12:52:43.592031	M003	ON
12:52:44.280571	SG023	(1680,2927,2507,1562,2190,1819)
12:52:44.401324	M004	ON
12:52:44.423522	SG024	(742,2203,1981,1825,1916,1823)
12:52:45.298306	SG023	(458,3279,2093,1733,1784,1738)
12:52:45.407327	SG024	(1070,2070,1473,1766,1734,1933)
12:52:46.267412	SG023	(1369,2874,2327,1970,1763,1830)
12:52:46.394774	SG024	(913,1993,1609,1828,1788,1921)
12:52:47.262079	SG023	(1749,3797,2837,2052,1821,1731)
12:52:47.397942	SG024	(868,2165,1821,1891,1768,1915)
12:52:48.260983	SG023	(2295,2606,2655,1386,1872,1875)
12:52:48.390858	SG024	(994,1979,1548,1840,1822,1913)
12:52:48.743538	M005	OFF
12:52:48.764928	M006	OFF
12:52:48.778634	M003	OFF
12:52:49.258773	SG023	(1944,1806,2292,1767,1644,2229)
12:52:49.381293	SG024	(1200,2079,1617,1832,1703,1967)
12:52:50.37373	SG023	(2354,2572,2743,2055,1683,1815)
12:52:50.751139	SG023	(2027,2762,2399,2193,1711,1741)
12:52:51.245415	SG023	(2083,2765,2293,1388,2070,2075)
12:52:51.346045	M003	ON
12:52:51.380606	SG024	(1178,1777,1494,1908,1948,1865)

(*continued*)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:52:52.242926	SG023	(1694,1187,2483,1628,1810,2054
12:52:52.375861	SG024	(941,1839,1669,1842,1648,1897
12:52:53.239922	SG023	(689,2464,2430,1615,1830,1148)
12:52:53.374768	SG024	(832,2082,1711,1744,1837,1941)
12:52:54.234284	SG023	(3145,3190,2459,1696,1975,1972)
12:52:54.329416	M002	ON
12:52:54.383917	SG024	(903,2071,1649,1872,1652,1929)
12:52:55.366815	SG023	(1903,1964,2190,1908,1755,1898)
12:52:55.737073	SG023	(1679,2173,2776,1823,2052,2162)
12:52:56.230601	SG023	(1445,2816,2472,1697,2063,2069)
12:52:56.374147	SG024	(964,1995,1591,1869,1885,1874)
12:52:56.595402	M004	OFF
12:52:57.233027	SG023	(1719,2517,2598,1554,1732,1924)
12:52:57.356799	SG024	(943,2104,1677,1794,1796,1936)
12:52:58.225558	SG023	(1429,1815,2227,1950,1719,1669)
12:52:58.348098	SG024	(1106,1779,1589,1827,1836,1894)
12:52:58.952807	M003	OFF
12:52:59.219818	SG023	(979,1682,2648,2035,1842,1827)
12:52:59.34435	SG024	(886,1880,1630,1844,1933,1909)
12:53:00.217299	SG023	(1060,2467,2048,1951,1809,2144)
12:53:00.342954	SG024	(1027,1785,1658,1778,1743,1951)
12:53:01.206178	SG023	(1709,2307,2439,1842,1863,1801)
12:53:01.344737	SG024	(1014,1774,1746,1854,1854,1877)
12:53:02.206985	SG023	(1227,1671,2034,1805,1570,1935)
12:53:02.337979	SG024	(1009,2081,1540,1815,1843,1829)
12:53:03.200074	SG023	(1710,1299,2214,1910,1816,1838)
12:53:03.33001	SG024	(1124,1593,1793,1824,1978,1885)
12:53:04.197216	SG023	(1052,2144,2236,1687,1673,1798)
12:53:04.323212	SG024	(944,1732,1817,1827,1730,1863)
12:53:05.189522	SG023	(1179,1854,1965,1850,1821,1905)
12:53:05.321309	SG024	(1051,1775,1637,1827,1861,1923)
12:53:06.196276	SG023	(938,1600,2766,1923,1896,1576)

Sensor Data Collected While Individual Performs a Sweeping Activity

Time	Sensor ID	Sensor Message
12:53:06.344547	SG024	(1084,1833,1689,1788,1836,1803)
12:53:07.309256	SG023	(1515,2221,2843,1533,1887,1983)
12:53:07.684585	SG023	(2432,2104,2886,1360,1692,1930)
12:53:08.179187	SG023	(2069,1761,3039,1705,1682,1854)
12:53:08.309847	SG024	(888,1870,1888,1854,1758,1942)
12:53:09.175384	SG023	(1963,1431,2694,1903,1865,1940)
12:53:09.311664	SG024	(986,1930,1528,1846,1759,1925)
12:53:10.179166	SG023	(1498,1727,2755,1722,1972,2043)
12:53:10.231976	M002	OFF
12:53:10.32943	SG024	(1035,1865,1515,1815,1861,1852)
12:53:11.171072	SG023	(1786,2186,2933,1853,1871,1859)
12:53:11.305767	SG024	(995,1970,1560,1794,1662,1959)
12:53:12.186223	SG023	(1810,2762,2636,1670,1774,1839)
12:53:12.295038	SG024	(1018,2037,1542,1831,1773,1932)
12:53:13.177505	SG023	(1258,2494,2367,2550,1914,1954)
12:53:13.307114	SG024	(995,1883,1479,1823,1787,1907)
12:53:14.162811	SG023	(1906,3130,2164,1747,1863,1768)
12:53:14.286245	SG024	(908,1779,1784,1789,1721,1960)
12:53:14.815529	M001	ON

Bibliography

1. Russell, S. J. & Norvig, P. *Artificial Intelligence: A Modern Approach (third edition).* (Prentice Hall, 2010).
2. Candamo, J., Shreve, M., Goldgof, D., Sapper, D. & Kasturi, R. Understanding transit scenes: A survey on human behavior recognition algorithms. *IEEE Trans. Intell. Transp. Syst.* 11, 206–224 (2010).
3. Chaquet, J. M., Carmona, E. J. & Fernandez-Caballero, A. A survey of video datasets for human action and activity recognition. *Comput. Vis. Image Underst.* 117, 633–659 (2013).
4. Borges, P. V. K., Conci, N. & Cavallaro, A. Video-based human behavior understanding: A survey. *IEEE Trans. Circuits Syst. Video Technol.* 23(11), 1993–2008 (2013).
5. Cook, D. J., Krishnan, N. & Wemlinger, Z. Learning a taxonomy of predefined and discovered activity patterns. *J. Ambient Intell. Smart Environ.* 5(6), 621–637 (2013).
6. Chen, L., Nugent, C. & Okeyo, G. An ontology-based hybrid approach to activity modeling for smart homes. *IEEE Trans. Human-Machine Syst.* 44, 92–105 (2014).
7. Bae, I.-H. An ontology-based approach to ADL recognition in smart homes. *Futur. Gener. Comput. Syst.* 33, 32–41 (2014).
8. Blanke, U. & Schiele, B. Remember and transfer what you have learned - recognizing composite activities based on activity spotting. *Int. Symp. Wearable Comput.* 1–8 (2010).
9. Riboni, D. & Bettini, C. OWL 2 modeling and reasoning with complex human activities. *Pervasive Mob. Comput.* 7, 379–395 (2011).
10. Shelkey, M. & Wallace, M. Katz index of independence in activities of daily living. *J. Gerontol. Nurs.* 25, 8–9 (1999).
11. Graf, C. The Lawton instrumental activities of daily living scale. *Am. J. Nurs.* 108, 52–62 (2008).
12. Tapia, E. M. Using machine learning for real-time activity recognition and estimation of energy expenditure. Dissertation, Massachusetts Institute of Technology. (2008).
13. Ainsworth, B. E. *et al.* Compendium of physical activities: A second update of codes and MET values. *Med. Sci. Sports Exerc.* 43, 1575–1581 (2011).

14. Bureau of Labor Statistics. American time use survey - Activity coding lexicons. *United States Dep. Labor* (2013). At <http://bls.gov/tus/lexicons.htm>
15. Ekman, P., Friesen, W. V & Hager, J. C. *Facial Action Coding System: The Manual*. (Research Nexus eBook, 2002).
16. Stoia, L., Shockley, D. M., Byron, D. K. & Fosler-Lussier, E. Noun phrase generation for situated dialogs. *Int. Nat. Lang. Gener. Conf.* 81–88 (2006).
17. Rodriguez, M., Ahmed, J. & Shah, M. Action MACH: A statio-temporal maximum average correlation height filter for action recognition. *IEEE Conf. Comput. Vis. Pattern Recognit.* (2008).
18. Smith, J. R. *et al.* RFID-based techniques for human-activity detection. *Commun. ACM* 48, 39–44 (2005).
19. Buettner, M., Prasad, R., Philipose, M. & Wetherall, D. Recognizing daily activities with RFID-based sensors. *Int. Conf. Ubiquitous Comput.* 51–60 (2009).
20. Sample, A. P., Waters, B. H., Wisdom, S. T. & Smith, J. R. Enabling seamless wireless power delivery in dynamic environments. *Proc. IEEE* 101, 1343–1358 (2013).
21. Zhu, Y., Zheng, V. W. & Yang, Q. Activity recognition from trajectory data. *Computer with Spatial Trajectories* (Zheng, Y. & Zhou, X.) 179–212 (Springer, 2011).
22. Reddy, S. *et al.* Using mobile phones to determine transportation modes. *ACM Trans. Sens. Networks* 6, 13:1–13:27 (2010).
23. Pu, Q., Gupta, S., Gollakota, S. & Patel, S. Whole-home gesture recognition using wireless signals. *ACM MOBICOM* 485–486 (2013).
24. Gupta, S., Morris, D., Patel, S. N. & Tan, D. SoundWave: Using the Doppler effect to sense gestures. in *SIGCHI Conf. Hum. Factors Comput. Syst.* 1911–1914 (2012).
25. Larson, E. *et al.* Disaggregated water sensing from a single, pressure-based sensor. *Pervasive Mob. Comput.* 8, 82–102 (2012).
26. Froehlich, J. *et al.* Disaggregated end-use energy sensing for the smart grid. *IEEE Pervasive Comput.* 10, 28–39 (2011).
27. Bulling, A., Ward, J. A., Gellersen, H. & Troster, G. Eye movement analysis for activity recognition using electrooculography. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 741–753 (2011).
28. Gill, T., Keller, J. M., Anderson, D. T. & Luke, R. H. A system for change detection and human recognition in voxel space using the Microsoft Kinect sensor. *IEEE Appl. Imag. Pattern Recognit. Work.* 1–8 (2011).
29. Hongeng, S., Nevatia, R. & Bremond, F. Video-based event recognition: Activity representation and probabilistic recognition methods. *Comput. Vis. Image Underst.* 96, 129–162 (2004).
30. Aggarwal, J. K. & Ryoo, M. S. Human activity analysis: A review. *ACM Comput. Surv.* 43, 1–47 (2011).
31. Moncrieff, S., Venkatesh, S., West, G. & Greenhill, S. Multi-modal emotive computing in a smart house environment. *Pervasive Mob. Comput.* 3, 79–94 (2007).
32. Zhixian, Y., Vigneshwaran, S., Chakraborty, D., Misra, A. & Aberer, K. Energy-efficient continuous activity recognition on mobile phones: An activity-adaptive approach. *Int. Symp. Wearable Comput.* 17–24 (2012).
33. Cardinaux, F., Brownsell, S., Hawley, M. & Bradley, D. Modelling of behavioural patterns for abnormality detection in the context of lifestyle reassurance. *Prog. Pattern Recognition, Image Anal. Appl.* 5197, 243–251 (2008).

34. Liu, V. *et al.* Ambient backscatter: Wireless communication out of thin air. *ACM SIGCOMM* 39–50 (2013).
35. Bao, L. & Intille, S. Activity recognition from user annotated acceleration data. *Pervasive* 1–17 (2004).
36. Keally, M., Zhou, G., Xing, G., Wu, J. & Pyles, A. PBN: Towards practical activity recognition using smartphone-based body sensor networks. *ACM Conf. Embed. Networked Sens. Syst.* 246–259 (2011).
37. Hoseini-Tabatabaei, S. A., Gluhak, A. & Tafazolli, R. A survey on smartphone-based systems for opportunistic user context recognition. *ACM Comput. Surv*. 45, 27:1–27:51 (2013).
38. Kunze, K. Compensating for on-body placement effects in activity recognition. Dissertation, Universitat Passau, (2011).
39. Philipose, M. *et al.* Inferring activities from interactions with objects. *IEEE Pervasive Comput.* 3, 50–57 (2004).
40. Cook, D. J. & Holder, L. B. Sensor selection to support practical use of health-monitoring smart environments. *Data Min. Knowl. Discov.* 10, 1–13 (2011).
41. Blanke, U. *et al.* All for one or one for all? Combining heterogeneous features for activity spotting. *IEEE Int. Conf. Pervasive Comput. Commun. Work.* 18–24 (2010).
42. Zhang, M. & Sawchuk, A. A. Motion primitive-based human activity recognition using a bag-of-features approach. *ACM SIGHIT Int. Heal. Informatics Symp.* 631–640 (2012).
43. Luo, R. C., Chih-Chen, Y. & Kuo-Lan, S. Multisensor fusion and integration: Approaches, applications, and future research directions. *IEEE Sens. J.* 2, 107–119 (2002).
44. Durrant-Whyte, H. & Henderson, T. C. Multisensor data fusion. in *Handbook of Robotics* (Siciliano, B. & Khatib, O.) 585–610 (Springer, 2008).
45. Ruta, D. & Gabrys, B. An overview of classifier fusion methods. *J. Comput. Inf. Syst.* 7, 1–10 (2000).
46. Kuncheva, L. I. A theoretical study on six classifier fusion strategies. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 281–286 (2002).
47. Gao, L., Bourke, A. K. & Nelson, J. A system for activity recognition using multi-sensor fusion. *IEEE Eng. Med. Biol. Soc. Conf.* 7869–7872 (2011).
48. Zappi, P. *et al.* Activity recognition from on-body sensors: Accuracy-power trade-off by dynamic sensor selection. *Eur. Conf. Wirel. Sens. Networks* 17–33 (2008).
49. Hong, X. *et al.* Fusion of sensor data for activity recognition in smart homes. *Pervasive Mob. Comput.* 5, 236–252 (2009).
50. Duda, R. O., Hart, P. E. & Stork, D. G. *Pattern Classification*. (John Wiley and Sons, 2000).
51. Mitchell, T. *Machine Learning*. (McGraw-Hill, 1997).
52. Bishop, C. *Pattern Recognition and Machine Learning*. (Springer, 2006).
53. Dempster, A. P., Laird, N. M. & Rubin, D. B. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Stat. Soc.* 39, 1–38 (1977).
54. Reynolds, D. A. & Rose, R. C. Robust text-independent speaker identification using Gaussian mixture speaker models. *IEEE Trans. Speech Audio Process.* 3, 72–83 (1995).
55. Bilmes, J. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. (1997).
56. Rabiner, L. R., Wilpon, J. G. & Juan, B. H. A segmental k-means training procedure for connected word recognition. *AT&T Tech. J.* 65, 21–31 (1986).

57. Kitani, K. M., Sato, Y. & Sugimoto, A. Recovering the basic structure of human activities from a video-based symbol string. *IEEE Work. Motion Video Comput.* (2007).
58. Hu, B., Wang, W. & Jin, H. Human interaction recognition based on transformation of spatial semantics. *IEEE Signal Process. Lett.* 19, 139–142 (2012).
59. Ivanov, Y. A. & Bobick, A. Recognition of visual activities and interactions by stochastic parsing. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 852–872 (2000).
60. Minnen, D., Essa, I. & Starner, T. Expectation grammars: Leveraging high-level expectations for activity recognition. *IEEE Int. Conf. Comput. Vis. Pattern Recognit.* 626–632 (2003).
61. Moore, D. & Essa, I. Recognizing multitasked activities using stochastic context-free grammar. *Natl. Conf. Artif. Intell.* 770–776 (2001).
62. Hamid, R. *et al.* A novel sequence representation for unsupervised analysis of human activities. *J. Artif. Intell.* 173, 1221–1244 (2009).
63. Zhang, Z., Tan, T. & Huang, K. An extended grammar system for learning and recognizing complex visual events. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 240–255 (2011).
64. Guyon, I. & Elisseeff, A. An introduction to variable and feature selection. *J. Mach. Learn. Res.* 3, 1157–1182 (2003).
65. Liu, H. & Motada, H. *Computational Methods for Feature Selection*. (CRC Chapman and Hall, 2007).
66. Huynh, T. & Schiele, B. Analyzing features for activity recognition. *Jt. Conf. Smart Objects Ambient Intell.* 159–163 (2005).
67. Zhang, M. & Sawchuk, A. A. A feature selection-based framework for human activity recognition using wearable multimodal sensors. *Int. Conf. Body Area Networks* 92–98 (2011).
68. Pirttikangas, P., Fujinami, K. & Nakajima, T. Feature selection and activity recognition from wearable sensors. *Ubiquitous Comput. Syst. Lect. Notes Comput. Sci.* 4239, 516–527 (2006).
69. Lee, J. A. & Verleysen, M. *Nonlinear Dimensionality Reduction*. (Springer, 2007).
70. Wall, M. E., Rechtsteiner, A. & Rocha, L. M. in *A Pract. Approach to Microarray Data Anal.* 91–109 (2003).
71. Ali, A. & Aggarwal, J. K. Segmentation and recognition of continuous human activity. *IEEE Work. Detect. Recognit. events video* 28–35 (2001).
72. Ho, J. & Intille, S. S. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. *SIGCHI Conf. Hum. Factors Comput. Syst.* 909–918 (2005).
73. Feuz, K., Cook, D. J., Rosasco, C., Robertson, K. & Schmitter-Edgecombe, M. Automated detection of activity transitions for prompting. *IEEE Trans. Human-Machine Syst*. (2015).
74. Iqbal, S. T. & Bailey, B. P. Understanding and developing models for detecting and differentiating breakpoints during interactive tasks. *SIGCHI Conf. Hum. Factors Comput. Syst.* 697–706 (2007).
75. Niu, F. & Abdel-Mottaleb, M. HMM-based segmentation and recognition of human activities from video sequences. *IEEE Int. Conf. Multimed. ExpoICME* 804–807 (2005).
76. Duchenne, O., Laptev, I., Sivic, J., Bach, F. & Ponce, J. Automatic annotation of human activities in video. *Int. Conf. Comput. Vis.* 1491–1498 (2009).
77. Zheng, Y., Wong, W.-K., Guan, X. & Trost, S. Physical activity recognition from accelerometer data using a multi-scale ensemble method. *Innov. Appl. Artif. Intell. Conf.* 1575–1581 (2013).

78. Krishnan, N. & Cook, D. J. Activity recognition on streaming sensor data. *Pervasive Mob. Comput.* 10, 138–154 (2014).
79. Okeyo, G., Chen, L., Wang, H. & Sterritt, R. Dynamic sensor data segmentation for real-time knowledge-driven activity recognition. *Pervasive Mob. Comput.* 10, 155–172 (2014).
80. Khan, S. S. & Madden, M. G. A survey of recent trends in one class classification. *Irish Conf. Artif. Intellignece Cogn. Sci.* 188–197 (2010).
81. Hong, X. & Nugent, C. D. Segmenting sensor data for activity monitoring in smart environments. *Pers. Ubiquitous Comput.* 17, 545–559 (2013).
82. Gu, T., Chen, S., Tao, X. & Lu, J. An unsupervised approach to activity recognition and segmentation based on object-use fingerprints. *Data Knowl. Eng.* 69, 533–544 (2010).
83. Palmes, P., Pung, H. K., Gu, T., Xue, W. & Chen, S. Object relevance weight pattern mining for activity recognition and segmentation. *Pervasive Mob. Comput.* 6, 43–57 (2010).
84. Yamasaki, T. & Aizawa, K. Motion segmentation and retrieval for 3D video based on modified shape distribution. *EURASIP J. Appl. Signal Processing* 2007, 211–211 (2007).
85. Keogh, E., Chu, S., Hart, D. & Pazzani, M. An online algorithm for segmenting time series. *IEEE Int. Conf. Data Min.* 289–296 (2001).
86. Adams, R. P. & MacKay, D. J. *Bayesian online changepoint detection*. Technical Report, University of Cambridge, Cambridge, UK, 2007.
87. Yoshinobu, K. & Sugiyama, M. Change-point detection in time-series data by direct density-ratio estimation. *SIAM Int. Conf. Data Min.* 389–400 (2009).
88. Liu, S., Yamada, M., Collier, N. & Sugiyama, M. Change-point detection in time-series data by relative density-ratio estimation. *Neural Networks* 43, 72–83 (2013).
89. Guenterberg, E., Ostadabbas, S., Ghasemzadeh, H. & Jafari, R. An automatic segmentation technique in body sensor networks based on signal energy. *Int. Conf. Body Area Networks* 21 (2009).
90. Feuz, K., Cook, D. J., Rosasco, C., Robertson, K., & Schmitter-Edgecombe, M. Automated detection of activity transitions for prompting. *IEEE Transactions on Human-Machine Systems* (2015).
91. Xie, Y., Huang, J. & Willett, R. Change-point detection for high-dimensional time series with missing data. *IEEE J. Sel. Top. Signal Process.* 7, 12–27 (2013).
92. Avci, U. & Passerini, A. Improving activity recognition by segmental pattern mining. *IEEE Trans. Knowl. Data Eng.* 26, 889–902 (2014).
93. Chen, L., Hoey, J., Nugent, C. D., Cook, D. J. & Yu, Z. Sensor-based activity recognition. *IEEE Trans. Syst. Man, Cybern. Part C Appl. Rev*. 42, 790–808 (2012).
94. Ke, S.-R. *et al.* A Review on Video-Based Human Activity Recognition. *Computers* 2, 88–131 (2013).
95. Bulling, A., Blanke, U. & Schiele, B. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Comput. Surv.* 46, 107–140 (2014).
96. Reiss, A., Stricker, D. & Hendeby, G. Towards robust activity recognition for everyday life: Methods and evaluation. *Pervasive Comput. Technol. Healthc.* 25–32 (2013).
97. Vishwakarma, S. & Agrawal, A. A survey on activity recognition and behavior understanding in video surveillance. *Vis. Comput.* 29, 983–1009 (2013).
98. Vail, D., Veloso, M. & Lafferty, J. Conditional random fields for activity recognition. *AAMAS* (2007).
99. Lara, O. & Labrador, M. A. A survey on human activity recognition using wearable sensors. *IEEE Commun. Surv. Tutorials* 15, 1192–1209 (2013).
100. Chen, L. & Khalil, I. in *Act. Recognit. Pervasive Intell. Environ.* (Chen, L., Nugent, C. D., Biswas, J. & Hoey, J.) 1–31 (Atlantis Ambient and Pervasive Intelligence, 2011).

101. Tuaraga, P., Chellappa, R., Subrahmanian, V. S., Udrea, O. & Turaga, P. Machine recognition of human activities: A survey. *IEEE Trans. Circuits Syst. Video Technol.* 18, 1473–1488 (2008).
102. Cook, D. Learning setting-generalized activity models for smart spaces. *IEEE Intell. Syst.* 27, 32–38 (2012).
103. Ravi, N., Dandekar, N., Mysore, P. & Littman, M. L. Activity recognition from accelerometer data. in *Innov. Appl. Artif. Intell.* 1541–1546 (2005).
104. Ward, J. A., Lukowicz, P., Troster, G. & Starner, T. E. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE Trans. Pattern Anal. Mach. Intell.* 28, 1553–1567 (2006).
105. Singla, G., Cook, D. J. & Schmitter-Edgecombe, M. Recognizing independent and joint activities among multiple residents in smart environments. *Ambient Intell. Humaniz. Comput. J.* 1, 57–63 (2010).
106. Lester, J., Choudhury, T., Kern, N., Borriello, G. & Hannaford, B. A hybrid discriminative/generative approach for modeling human activities. in *Int. Jt. Conf. Artif. Intell.* 766–772 (2005).
107. Amft, O. & Troster, G. On-body sensing solutions for automatic dietary monitoring. *IEEE Pervasive Comput.* 8, 62–70 (2009).
108. Van Kasteren, T., Noulas, A., Englebienne, G. & Krose, B. Accurate activity recognition in a home setting. in *ACM Conf. Ubiquitous Comput.* 1–9 (2008).
109. Bulling, A., Ward, J. A. & Gellersen, H. Multimodal recognition of reading activity in transit using body-worn sensors. *ACM Trans. Appl. Percept.* 9, 2:1–2:21 (2012).
110. Wang, S., Pentney, W., Popescu, A. M., Choudhury, T. & Philipose, M. Common sense based joint training of human activity recognizers. in *Int. Jt. Conf. Artif. Intell.* 2237–2242 (2007).
111. Lester, J., Choudhury, T. & Borriello, G. A practical approach to recognizing physical activities. in *Int. Conf. Pervasive Comput.* 1–16 (2006).
112. Taati, B., Snoek, J. & Mihailidis, A. Video analysis for identifying human operation difficulties and faucet usability assessment. *Neurocomputing* 100, 163–169 (2013).
113. Dawadi, P., Cook, D. J., Schmitter-Edgecombe, M. & Parsey, C. Automated assessment of cognitive health using smart home technologies. *Technol. Heal. Care* 21, 323–343 (2013).
114. Dawadi, P., Cook, D. J. & Schmitter-Edgecombe, M. Automated cognitive health assessment using smart home monitoring of complex tasks. *IEEE Trans. Syst. Man, Cybern. Part B* 43, 1302–1313 (2013).
115. Ogris, G., Stiefmeier, T., Lukowicz, P. & Troster, G. Using a complex multi-modal on-body sensor system for activity spotting. in *Int. Symp. wearable Comput.* 55–62 (2012).
116. Amft, O. Self-taught learning for activity spotting in on-body motion sensor data. in *Int. Symp. wearable Comput.* 83–86 (2011).
117. Ward, J. A., Lukowicz, P. & Gellersen, H. W. Performance metrics for activity recognition. *ACM Trans. Intell. Syst. Technol*. 2, 6:1–6:23 (2011).
118. Ross, R. J. & Kelleher, J. Accuracy and timeliness in ML based activity recognition. in *AAAI Work. Plan, Act. Intent Recognit.* 39–46 (2013).
119. Di Eugenio, B. & Glass, M. The kappa statistic: A second look. *Comput. Linguist.* 30, 95–101 (2004).

120. Davis, J. & Goadrich, M. The relationship between precision-recall and ROC curves. in *Int. Conf. Mach. Learn.* 233–240 (2006).
121. Cohen, P. R. *Empirical Methods for Artificial Intelligence*. (A Bradford Book, 1995).
122. Dietterich, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Comput.* 10, 1895–1923 (1998).
123. Nuzzo, R. Scientific method: Statistical errors. *Nature* 506, 150–152 (2014).
124. Durant, W. *The Story of Philosophy: The Lives and Opinions of the World's Greatest Philosophers*. (Simon and Schuster, 1926).
125. Palatucci, M., Pomerleau, D., Hinton, G. & Mitchell, T. Zero-shot learning with semantic output codes. in *Annu. Conf. Neural Inf. Process. Syst.* 1410–1418 (2009).
126. Cheng, H.-T., Griss, M., Davis, P., Li, J. & You, D. Towards zero-shot learning for human activity recognition using semantic attribute sequence model. in *ACM Int. Jt. Conf. Pervasive Ubiquitous Comput.* 355–358 (2013).
127. Russakovsky, O. & Fei-Fei, L. Attribute learning in large-scale datasets. in *Parts Attrib. Work. Eur. Conf. Comput. Vis.* 1–14 (2010).
128. Agrawal, R. & Srikant, R. Mining sequential patterns. in *Proc. Int. Conf. Data Eng.* 3–14 (1995).
129. Han, J. *et al.* FreeSpan: Frequent pattern-projected sequential pattern mining. in *ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* 355–359 (2000).
130. Pei, J. *et al.* Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Trans. Knowl. Data Eng.* 16, 1424–1440 (2004).
131. Zaki, M. J. Efficient enumeration of frequent sequences. in *ACM Int. Conf. Inf. Knowl. Manag.* 68–75 (1998).
132. Gouda, K. & Hassaan, M. Mining sequential patterns in dense databases. *Int. J. Database Manag. Syst.* 3, 179–194 (2011).
133. Rissanen, J. *Stochastic Complexity in Statistical Inquiry*. (World Scientific Publishing Company, 1989).
134. Cook, D. J., Krishnan, N. & Rashidi, P. Activity discovery and activity recognition: A new partnership. *IEEE Trans. Syst. Man, Cybern. Part B* 43, 820–828 (2013).
135. Heierman, E. O. & Cook, D. J. Improving home automation by discovering regularly occurring device usage patterns. in *IEEE Int. Conf. Data Min.* 537–540 (2003).
136. Rashidi, P., Cook, D., Holder, L. & Schmitter-Edgecombe, M. Discovering activities to recognize and track in a smart environment. *IEEE Trans. Knowl. Data Eng.* 23, 527–539 (2011).
137. Elfeky, M. G., Aref, W. G. & Elmagarmid, A. K. Periodicity detection in time series databases. *IEEE Trans. Knowl. Data Eng.* 17, 875–887 (2005).
138. Ukkonen, E. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.* 92, 191–202 (1992).
139. Kullback, S. & Leibler, R. A. On information and sufficiency. *Ann. Math. Stat.* 22, 79–86 (1951).
140. Levenshtein, V. I. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* 10, 707–710 (1966).
141. Zhou, H. Y. & Shan, J. Time sequence clustering based on edit distance. *Appl. Mech. Mater.* 401–403, 1428–1431 (2013).
142. Yang, J. & Wang, W. CLUSEQ: Efficient and effective sequence clustering. in *Int. Conf. Data Eng.* 101–112 (2003).

143. Alon, J., Athitsos, V., Yuan, Q. & Sclaroff, S. A unified framework for gesture recognition and spatiotemporal gesture segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* 31, 1685–1699 (2008).
144. Dunn, D. Well separated clusters and optimal fuzzy partitions. *J. Cybern.* 4, 95–104 (1974).
145. Davies, D. L. & Bouldin, D. W. A cluster separation measure. *IEEE Trans. Pattern Anal. Mach. Intell.* 1, 224–227 (1979).
146. Blei, D., Ng, A. & Jordan, M. Latent Dirichlet allocation. *J. Mach. Learn. Res.* 3, 993–1022 (2003).
147. Huynh, T., Fritz, M. & Schiele, B. Discovery of activity patterns using topic models. *in Int. Conf. Ubiquitous Comput.* 10–19 (2008).
148. Varadarajan, J., Emonet, R. & Odobez, J.-M. A sequential topic model for mining recurrent activities from long term video logs. *Int. J. Comput. Vis.* 103, 100–126 (2013).
149. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C. & Baskurt, A. Sequential deep learning for human action recognition. in *Int. Conf. Hum. Behav. Underst.* 29–39 (2011).
150. Coates, A., Karpathy, A. & Ng, A. Y. Emergence of object-selective features in unsupervised feature learning. in *Int. Conf. Neural Inf. Process. Syst.* 1–9 (2012).
151. Wyatt, D., Philipose, M. & Choudhury, T. Unsupervised activity recognition using automatically mined common sense. in *Natl. Conf. Artif. Intell.* 21–27 (2005).
152. Ziv, J. & Lempel, A. A compression of individual sequences via variable rate coding. *IEEE Trans. Inf. Theory* IT-24, 530–536 (1978).
153. Feder, M., Merhav, N. & Gutman, M. Universal prediction of individual sequences. *IEEE Trans. Inf. Theory* 38, 1258–1270 (1992).
154. Bhattacharya, A. & Das, S. K. LeZi-Update: An Information-theoretic approach for personal mobility tracking in PCS networks. *Wirel. Networks* 8, 121–135 (2002).
155. Gopalratnam, K. & Cook, D. J. Online sequential prediction via incremental parsing: The Active LeZi Algorithm. *IEEE Intell. Syst.* 22, (2007).
156. Begleiter, R., El-Yaniv, R. & Yona, G. On prediction using variable order Markov models. *J. Artif. Intell. Res.* 22, 385–421 (2004).
157. Quinlan, J. R. Learning with continuous classes. in *Aust. Jt. Conf. Artif. Intell.* 343–348 (1992).
158. Utgoff, P. E. Incremental induction of decision trees. *Mach. Learn.* 4, 161–186 (1989).
159. Mahmoud, S., Lotfi, A. & Langensiepen, C. Behavioural pattern identification and prediction in intelligent environments. *Appl. Soft Comput.* 13, 1813–1822 (2013).
160. Box, G. & Jenkins, G. *Time Series Analysis*. (Holden-Day, 1970).
161. Krumm, J. & Horvitz, E. Predestination: inferring destinations from partial trajectories. in *UbiComp 2006 Eighth Int. Conf. Ubiquitous Comput.* 243–260 (2006).
162. Dewar, M., Wiggins, C. & Wood, F. Inference in hidden Markov models with explicit state duration distributions. *IEEE Signal Process. Lett.* 19, 235–238 (2012).
163. Kenny, P., Lennig, M. & Mermelstein, P. A linear predictive HMM for vector-valued observations with applications to speech recognition. *IEEE Trans. Acoust. Speech Signal Process.* 38, 220–225 (1990).
164. Park, D. W., Kwon, J. & Lee, K. M. Robust visual tracking using autoregressive hidden Markov model. in *IEEE Conf. Comput. Vis. Pattern Recognit.* 1964–1971 (2012).
165. Cook, D. J. & Holder, L. Automated activity-aware prompting for activity initiation. *Gerontechnology* 11, 534–544 (2013).
166. Kipp, M. Multimedia annotation, querying and analysis in ANVIL. in *Multimedia Information Extraction* (Maybury, M.) 351–367 (Wiley, 2012).

167. Sloetjes, H. & Wittenburg, P. Annotation by category - ELAN and ISO DCR. in *Int. Conf. Lang. Resour. Eval.* 816–820 (2008).
168. Banziger, T. & Scherer, K. R. in *Bluepr. Affect. Comput. A Sourceb*. (Scherer, K. R., Banziger, T. & Roesch, E. B.) 271–294 (Oxford University Press, 2010).
169. Szewcyzk, S., Dwan, K., Minor, B., Swedlove, B. & Cook, D. J. Annotating smart environment sensor data for activity learning. *Technol. Heal. Care* 17, 161–169 (2009).
170. Fleiss, J. L., Levin, B. & Paik, M. C. The measurement of interrater agreement. *Stat. Methods Rates Proportions* 2, 212–236 (1981).
171. Song, Y., Lu, Z., Leung, C. W. & Yang, Q. Collaborative boosting for activity classification in microblogs. in *ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.* 482–490 (2013).
172. He, H. & Garcia, E. Learning from imbalanced data. *IEEE Trans. Knowl. Data Eng.* 21, 1263–1284 (2009).
173. Hong, X., Chen, S. & Harris, C. A kernel-based two-class classifier for imbalanced data sets. *IEEE Trans. Neural Networks* 18, 28–41 (2007).
174. Wu, G. & Chang, E. KBA: Kernel boundary alignment considering imbalanced data distribution. *IEEE Trans. Knowl. Data Eng.* 17, 786–795 (2005).
175. Fung, G. & Mangasarian, O. Multicategory proximal support vector machine classifiers. *Mach. Learn.* 59, 77–97 (2005).
176. Chawla, N., Bowyer, K., Hall, L. & Kegelmery, W. P. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* 16, 321–357 (2002).
177. Sun, Y., Kamel, M. S., Wong, A. K. C. & Wang, Y. Cost-sensitive boosting for classification of imbalanced data. *Pattern Recognit.* 40, 3358–3378 (2007).
178. Das, B., Krishnan, N. & Cook, D. J. RACOG and wRACOG: Two probabilistic oversampling methods. *IEEE Trans. Knowl. Data Eng.* (2015).
179. Rashidi, P. & Cook, D. J. Ask me better questions: Active learning queries based on rule induction. in *Int. Conf. Knowl. Discov. Data Min.* 904–912 (2011).
180. Alemdar, H., van Kasteren, T. L. M. & Ersoy, C. Using active learning to allow activity recognition on a large scale. *Lect. Notes Comput. Sci.* 7040, 105–114 (2011).
181. Rebetez, J., Satizabal, H. F. & Perez-Uribe, A. Reducing user intervention in incremental activity recognition for assistive technologies. in *Int. Symp. Wearable Comput.* 29–32 (2013).
182. Stikic, M., Larlus, D., Ebert, S. & Shiele, B. Weakly supervised recognition of daily life activities with wearable sensors. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, (2011).
183. Munguia-Tapia, E., Intille, S. S. & Larson, K. Activity recognition in the home using simple and ubiquitous sensors. in *Pervasive* 158–175 (2004).
184. Moskowitz, D. S. & Young, S. N. Ecological momentary assessment: What it is and why it is a method of the future in clinical psychopharmacology. *J. Psychiatry Neurosci.* 31, 13–20 (2006).
185. Zhu, X. & Goldberg, A. B. *Introduction to Semi-Supervised Learning*. (Morgan and Claypool, 2009).
186. Chinaei, L. *Active learning with semi-supervised support vector machines*. Dissertation, University of Waterloo (2007).
187. Lasecki, W. S., Song, Y. C., Kautz, H. & Bigham, J. P. Real-time crowd labeling for deployable activity recognition. *ACM Conf. Comput. Support. Coop. Work Soc. Comput.* (2013).
188. Hwang, K. & Lee, S.-Y. Environmental audio scene and activity recognition through mobile-based crowdsourcing. *IEEE Trans. Consum. Electron.* 58, 700–705 (2012).

189. Zhao, L., Sukthankar, G. & Sukthankar, R. Robust active learning using crowdsourced annotations for activity recognition. *AAAI Work. Hum. Comput.* 74–79 (2011).
190. Pan, S. J. & Yang, Q. A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* 22, 1345–1359 (2010).
191. Xu, C., Tao, D. & Xu, C. A survey on multi-view learning. *Comput. Res. Repos.* **abs/1304.5**, (2013).
192. Cook, D. J., Feuz, K. & Krishnan, N. Transfer learning for activity recognition: A survey. *Knowl. Inf. Syst.* 36, 537–556 (2013).
193. Arnold, A., Nallapati, R. & Cohen, W. W. A comparative study of methods for transductive transfer leraning. *Int. Conf. Data Min.* (2007).
194. Elkan, C. The foundations of cost-sensitive learning. *Int. Jt. Conf. Artif. Intell.* 973–978 (2011).
195. Thrun, S. *Explanation-Based Neural Network Learning: A Lifelong Learning Approach*. (Kluwer, 1996).
196. Thrun, S. & Pratt, L. *Learning To Learn*. (Kluwer Academic Publishers, 1998).
197. Vilalta, R. & Drissi, Y. A prospective view and survey of meta-learning. *Artif. Intell. Rev.* 18, 77–95 (2002).
198. Duan, L., Xu, D., Tsang, I. & Luo, J. Visual event recognition in videos by learning from web data. *IEEE Trans. Pattern Anal. Mach. Intell.* 34, 1667–1680 (2012).
199. Farhadi, A. & Tabrizi, M. Learning to recognize activities from the wrong view point. *Lect. Notes Comput. Sci.* 5302, 154–166 (2008).
200. Lam, A., Roy-Chowdhury, A. & Shelton, C. Interactive event search through transfer learning. *Lect. Notes Comput. Sci.* 6494, 157–170 (2011).
201. Liu, J., Shah, M., Kuipers, B. & Savarese, S. Cross-view action recognition via view knowledge transfer. *IEEE Conf. Comput. Vis. Pattern Recognit.* 3209–3216 (2011).
202. Wei, B. & Pal, C. Heterogeneous transfer learning with RBMs. *AAAI Conf. Artif. Intell.* 531–536 (2011).
203. Nater, F., Tommasi, T., Van Gool, L. & Caputo, B. in *Interact. Multimodal Inf. Manag*. (Bourlard, H. & Popescu-Belis, A.) 1–17 (EPFL Press, 2013).
204. Yang, J., Yan, R. & Hauptmann, A. G. Cross-domain video concept detection using adaptive SVMs. *Int. Conf. Multimed.* 188–197 (2007).
205. Hachiya, H., Sugiyama, M. & Ueda, N. Importance-weighted least-squares probabilistic classifier for covariate shift adaptation with application to human activity recognition. *Neurocomputing* 80, 93–101 (2012).
206. Kurz, M. *et al.* Real-time transfer and evaluation of activity recognition capabilities in an opportunistic system. *Int. Conf. Adapt. Self-Adaptive Syst. Appl.* 73–78 (2011).
207. Kwapisz, J., Weiss, G. & Moore, S. Activity recognition using cell phone accelerometers. *Int. Work. Knowl. Discov. from Sens. Data* 10–18 (2010).
208. Zhao, Z., Chen, Y., Liu, J., Shen, Z. & Liu, M. Cross-people mobile-phone based activity recognition. *Int. Jt. Conf. Artif. Intell.* 2545–2550 (2011).
209. Hu, D. H., Zheng, V. W. & Yang, Q. Cross-domain activity recognition via transfer learning. *Pervasive Mob. Comput.* 7, 344–358 (2011).
210. Van Kasteren, T., Englebiennce, G. & Krose, B. Transferring knowledge of activity recognition across sensor networks. *Int. Conf. Pervasive Comput.* 6030, 283–300 (2010).
211. Rashidi, P. & Cook, D. J. Activity knowledge transfer in smart environments. *Pervasive Mob. Comput.* 331, (2011).

212. Dai, W., Yang, Q., Xue, G. R. & Yu, Y. Boosting for transfer learning. *Int. Conf. Mach. Learn.* 193–200 (2007).
213. Al-Stouhi, S. & Reddy, C. K. Adaptive boosting for transfer learning using dynamic updates. *Eur. Conf. Mach. Learn.* 60–75 (2011).
214. Krumm, J. & Rouhana, D. Placer: Semantic place labels from diary data. *Int. Jt. Conf. Pervasive Ubiquitous Comput.* 163–172 (2013).
215. Blum, A. & Mitchell, T. Combining labeled and unlabeled data with co-training. *Annu. Conf. Comput. Learn. Theory* 92–100 (1998).
216. Nigam, K. & Ghani, R. Analyzing the effectiveness and applicability of co-training. *Int. Conf. Inf. Knowl. Manag.* 86–93 (2000).
217. Sun, S. A survey of multi-view machine learning. *Neural Comput. Appl.* 23, 2031–2038 (2013).
218. Kakade, S. M. & Foster, D. P. in *Learn. Theory* 82–96 (Springer, 2007).
219. Wang, C. & Mahadevan, S. Manifold alignment using procrustes analysis. *Int. Conf. Mach. Learn.* 1120–1127 (2008).
220. Sindhwani, V. & Rosenberg, D. S. An RKHS for multi-view learning and manifold co-regularization. *Int. Conf. Mach. Learn.* 967–983 (2008).
221. Roggen, D., Frster, K., Calatroni, A. & Trster, G. The adARC pattern analysis architecture for adaptive human activity recognition systems. *J. Ambient Intell. Humaniz. Comput.* 4, 169–186 (2013).
222. Shi, X. & Yu, P. Dimensionality reduction on heterogeneous feature space. *Int. Conf. Data Min.* 635–644 (2012).
223. Cao, L., Liu, Z. & Huang, T. Cross-dataset action detection. *IEEE Conf. Comput. Vis. Pattern Recognit.* 1998–2005 (2010).
224. Tsoumakas, G. & Katakis, I. Multi-label classification: An overview. *Int. J. Data Warehous. Min.* 3, 1–13 (2007).
225. Zhang, M. & Zhou, Z. A review of multi-label learning algorithms. *IEEE Trans. Knowl.* 31, 1–42 (2013).
226. Tai, F. & Lin, H.-T. Multilabel classification with principal label space transformation. *Neural Comput.* 24, 2508–2542 (2012).
227. Zhang, Y. & Schneider, J. Multi-label output codes using canonical correlation analysis. *AI Stat.* (2011).
228. Sun, L., Ji, S. & Ye, J. Canonical correlation analysis for multilabel classification: A least-squares formulation, extensions, and analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 194–200 (2011).
229. Chen, Y.-N. & Lin, H.-T. Feature-aware label space dimension reduction for multi-label classification. *Adv. Neural Inf. Process. Syst.* 25, 1538–1546 (2012).
230. Laibowitz, M., Gips, J., Aylward, R., Pentland, A. & Paradiso, J. A. A sensor network for social dynamics. *Proc. Int. Conf. Inf. Process. Sens. Networks* 483–491 (2006).
231. Lazer, D. *et al.* Life in the network: The coming age of computational social science. *Science (80)*. 323, 721–723 (2009).
232. Dong, W., Lepri, B. & Pentland, A. Modeling the co-evolution of behavior and social relationships using mobile phone data. *Mob. Ubiquitous Multimed.* 134–143 (2011).
233. Hung, H., Englebienne, G. & Kools, J. Classifying social actions with a single accelerometer. *ACM Int. Jt. Conf. Pervasive Ubiquitous Comput.* 207–210 (2013).
234. Hirano, T. & Maekawa, T. A hybrid unsupervised/supervised model for group activity recognition. *Int. Symp. wearable Comput.* 21–24 (2013).

235. Petersen, J., Larimer, N., Kaye, J. A., Pavel, M. & Hayes, T. L. SVM to detect the presence of visitors in a smart home environment. *Int. Conf. IEEE Eng. Med. Biol. Soc.* 5850–5853 (2012).
236. Kjaergaard, M. B. Studying sensing-based systems: Scaling to human crowds in the real world. *IEEE Comput.* 17, 80–84 (2013).
237. Kjaergaard, M. B., Wirz, M., Roggen, D. & Troster, G. Detecting pedestrian flocks by fusion of multi-modal sensors in mobile phones. *ACM Conf. Ubiquitous Comput.* 240–249 (2012).
238. Gordon, D., Hanne, J.-H., Berchtold, M., Shirehjini, A. A. N. & Beigl, M. Towards collaborative group activity recognition using mobile devices. *Mob. Networks Appl.* 18, 326–340 (2013).
239. Lu, C.-H. & Chiang, Y.-T. Interaction-enabled multi-user model learning for a home environment using ambient sensors. *IEEE J. Biomed. Heal. Informatics* (2015).
240. Ryoo, M. S. & Matthies, L. First-person activity recognition: What are they doing to me? in *IEEE Conf. Comput. Vis. Pattern Recognit.* 2730–2737 (2013).
241. Wang, L., Gu, T., Tao, X., Chen, H. & Lu, J. Multi-user activity recognition in a smart home. *Atl. Ambient Pervasive Intell.* 4, 59–81 (2011).
242. Wu, T., Lian, C. & Hsu, J. Y. Joint recognition of multiple concurrent activities using factorial conditional random fields. *AAAI Work. Plan, Act. Intent Recognit.* (2007).
243. Tolstikov, A., Phus, C., Biswas, J. & Huang, W. Multiple people activity recognition using MHT over DBN. *Int. Conf. Smart Homes Heal. Telemat.* 313–318 (2011).
244. Hu, D. H. & Yang, Q. CIGAR: Concurrent and interleaving goal and activity recognition. *Natl. Conf. Artif. Intell.* 1363–1368 (2008).
245. Oliver, N., Rosario, B. & Pentland, A. A Bayesian computer vision system for modeling human interactions. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 831–843 (2000).
246. Chiang, Y.-T., Hsu, K.-C., Lu, C.-H. & Fu, L.-C. Interaction models for multiple-resident activity recognition in a smart home. *Proc. Int. Conf. Intell. Robot. Syst.* 3753–3758 (2010).
247. Gu, T., Wang, L., Chen, H., Tao, X. & Lu, J. Recognizing multiuser activities using wireless body sensor networks. *IEEE Trans. Mob. Comput.* 10, 1618–1631 (2011).
248. Stowell, D. & Plumbley, M. D. Segregating event streams and noise with a Markov renewal process model. *J. Mach. Learn. Res.* 14, 2213–2238 (2013).
249. Ye, J., Stevenson, G. & Dobson, S. KCAR: A knowledge-driven approach for concurrent activity recognition. *Pervasive Mob. Comput*. Online at http://www.sciencedirect.com/science/article/pii/S1574119214000297. DOI: 10.1016/j.pmcj.2014.02.003 (2015).
250. Wilson, D. & Atkeson, C. Simultaneous tracking and activity recognition (STAR) using many anonymous, binary sensors. *Pervasive* 62–79 (2005).
251. Choi, W. & Savarese, S. A unified framework for multi-target tracking and collective activity recognition. *Eur. Conf. Comput. Vis.* 215–230 (2012).
252. Branzel, A. *et al.* GravitySpace: Tracking users and their poses in a smart room using a 2D pressure-sensing floor. *ACM SIGCHI Conf. Hum. Factors Comput. Syst.* 725–734 (2013).
253. Sousa, M., Techmer, A., Steinhage, A., Lauterbach, C. & Lukowicz, P. Human tracking and identification using a sensitive floor and wearable accelerometers. *IEEE Int. Conf. Pervasive Comput. Commun.* 165–170 (2013).
254. Hnat, T. W., Griffiths, E., Dawson, R. & Whitehouse, K. Doorjamb: Unobtrusive room-level tracking of people in homes using doorway sensors. *ACM Conf. Embed. Networked Sens. Syst.* 309–322 (2012).

255. Ranjan, J., Yao, Y. & Whitehouse, K. An RF doormat for tracking people's room locations. *ACM Int. Jt. Conf. Pervasive Ubiquitous Comput.* 797–800 (2013).
256. Lu, C.-H., Wu, C.-L. & Fu, L.-C. A reciprocal and extensible architecture for multiple-target tracking in a smart home. *IEEE Trans. Syst. Man, Cybern. Part C Appl. Rev.* 41, 120–129 (2011).
257. Cornelius, C. & Kotz, D. Recognizing whether sensors are on the same body. *Pervasive Mob. Comput.* 8, 822–836 (2012).
258. Crandall, A. & Cook, D. J. Coping with multiple residents in a smart environment. *J. Ambient Intell. Smart Environ.* 1, 323–334 (2009).
259. Carter, S., Cox, A., Quealy, K. & Schoenfeld, A. How different groups spend their day. *New York Times* (2009).
260. Vincent, G. & Velkoff, V. The next four decades - the older population in the United States: 2010 to 2050. (2010).
261. Alzheimer's Association. Alzheimer's disease: facts and figures. *Alzheimer's Dement.* 8, 131–168 (2012).
262. UCSF Center for California Health Workforce Studies. An aging U.S. population and the health care workforce: Factors affecting the need for geriatric care workers. (2006). at <http://futurehealth.ucsf.edu/Public/Publications-and-Resources/Content.aspx?topic=_An_Aging_U_S_Population_and_the_Health_Care_Workforce_Factors_Affecting_the_Need_for_Geriatric_Care_Workers>
263. Centers for Disease Control and Prevention. How many children have autism? *Natl. Cent. Birth Defects Dev. Disabil*. (2014). at <http://www.cdc.gov/ncbddd/features/counting-autism.html>
264. US Department of Veterans Affairs. How common is PTSD? *PTSD Natl. Cent. PTSD* (2014). at <http://www.ptsd.va.gov/public/PTSD-overview/basics/how-common-is-ptsd.asp>
265. Farias, S. *et al.* MCI is associated with deficits in everyday functioning. *Alzheimer Dis. Assoc. Disord.* 20, 217–223 (2006).
266. Covington, O. "Virtual nurse" helps Forsyth Medical Center track diabetes. *Bus. J.* (2013). at <http://www.bizjournals.com/triad/news/2013/05/20/forsyth-medical-center-using-virtual.html>
267. Friedman, M. J. PTSD history and overview. *US Dep. Veterans Aff*. (2014). at <http://www.ptsd.va.gov/professional/PTSD-overview/ptsd-overview.asp>
268. Kaye, J. *et al.* Unobtrusive measurement of daily computer use to detect mild cognitive impairment. *Alzheimer's Dement.* 10, 10–17 (2014).
269. Dodge, H. H., Mattek, N. C., Austin, D., Hayes, T. L. & Kaye, J. A. In-home walking speeds and variability trajectories associated with mild cognitive impairment. *Neurology* 78, 1946–1952 (2012).
270. Wadley, V., Okonkwo, O., Crowe, M. & Ross-Meadows, L. A. Mild cognitive impairment and everyday function: Evidence of reduced speed in performing instrumental activities of daily living. *Am. J. Geriatr. Psychiatry* 16, 416–424 (2007).
271. Aharony, N., Pan, W., Ip, C., Khayal, I. & Pentland, A. Social fMRI: Investigating and shaping social mechanisms in the real world. *Pervasive Mob. Comput.* 7, 643–659 (2011).
272. Moturu, S., Khayal, I., Aharony, N., Pan, W. & Pentland, A. Using social sensing to understand the links between sleep, mood and sociability. *IEEE Int. Conf. Soc. Comput.* 208–214 (2011).
273. Oloritun, R. O. *et al.* Change in BMI accurately predicted by social exposure to acquaintances. *PLoS One* 8, (2013).

274. Dawadi, P., Cook, D. J. & Schmitter-Edgecombe, M. *Longitudinal functional assessment of older adults using smart home sensor data*. *IEEE J. Biomed. Heal*. Informatics (2015).
275. Albinali, F., Goodwin, M. S. & Intille, S. S. Recognizing stereotypical motor movements in the laboratory and classroom: A case study with children on the autism spectrum. *Int. Conf. Ubiquitous Comput.* 71–80 (2009).
276. Westeyn, T., Vadas, K., Bian, X., Starner, T. & Abowd, G. Recognizing mimicked autistic self-stimulatory behaviors using HMMs. *IEEE Int. Symp. Wearable Comput.* 164–169 (2005).
277. Singh, M. & Patterson, D. J. Involuntary gesture recognition for predicting cerebral palsy in high-risk infants. *Int. Symp. Wearable Comput.* 1–8 (2010).
278. Hodges, S. *et al.* SenseCam: A retrospective memory aid. *Interational Conf. Ubiquitous Comput.* 177–193 (2006).
279. Hoey, J., Von Bertoldi, A., Craig, T., Poupart, P. & Mihailidis, A. Automated handwashing assistance for persons with dementia using video and a partially observable Markov decision process. *Comput. Vis. Image Underst.* 114, 503–519 (2010).
280. Hoey, J., Monk, A. & Mihailidis, A. People, sensors, decisions: Customizable and adaptive technologies for assistance in healthcare. *ACM Trans. Interact. Intell. Syst.* 2, (2012).
281. Fittle. Engaging people in healthy lifestyles. (2014). at <http://fittle.org/>
282. Sleep Cycle. Waking up made easy. (2014). at <http://www.sleepcycle.com/>
283. Moves. Activity tracking without gadgets. (2014). at <http://www.moves-app.com/>
284. Jawbone. The UP system. (2014). at <https://jawbone.com/up>
285. Endomondo. Track your workouts, challenge your friends, analyze your training. (2014). at <http://www.endomondo.com>
286. O'Reilly, T. Context aware programming. *O'Reilly Radar* (2013). at <http://radar.oreilly.com/2013/07/context-aware-programming.html>
287. Horvitz, E. & Krumm, J. Some help on the way: Opportunistic routing under uncertainty. *ACM Conf. Ubiquitous Comput.* 371–380 (2012).
288. Chen, C., Cook, D. J. & Crandall, A. The user side of sustainability: Modeling behavior and energy usage in the home. *Pervasive Mob. Comput.* 9, 161–175 (2013).
289. Youngblood, G. M. & Cook, D. J. Data mining for hierarchical model creation. *IEEE Trans. Syst. Man, Cybern. Part C* 37, 1–12 (2007).
290. Chamberlain, A., Martinez-Reyes, F., Jacobs, R., Watkins, M. & Shackford, R. Them and us: An indoor pervasive gaming experience. *Entertain. Gaming* 4, 1–9 (2013).
291. Ryoo, M. S., Lee, J. T. & Aggarwal, J. K. Video scene analysis of interactions between humans and vehicles using event context. in *ACM Int. Conf. Image Video Retr*. 462–469 (2010).
292. Feese, S. *et al.* Sensing group proximity dynamics of firefighting teams using smartphones. *Int. Symp. Wearable Comput.* 97–104 (2013).
293. Reshef, D. N. *et al.* Detecting novel associations in large data sets. *Science (80)*. 334, 1518–1523 (2011).
294. Moves. Moves: Activity tracker for iPhone and Android. www.moves-app.com (2013).
295. Wang, S., Skubic, M. & Zhu, Y. Activity density map visualization and dissimilarity comparison for eldercare monitoring. *IEEE Trans. Inf. Technol. Biomed.* 16, 607–614 (2012).
296. Haley, G. E. *et al.* Circadian activity associated with spatial learning and memory in aging rhesus monkeys. *Exp. Neurol.* 217, 55–62 (2009).

297. Frank, J., Mannor, S. & Precup, D. Generating storylines from sensor data. *Pervasive Mob. Comput.* 9, 838–847 (2013).
298. Candia, J. *et al.* Uncovering individual and collective human dynamics from mobile phone records. *J. Phys. a Math. Theor.* 41y 1–11 (2008).
299. Cook, D. J., Crandall, A., Thomas, B. & Krishnan, N. CASAS: A smart home in a box. *IEEE Comput.* 46(7), 62–69 (2012).
300. Wang, D., Pedreschi, D., Song, C., Giannotti, F. & Barabasi, A.-L. Human mobility, social ties, and link prediction. *ACM SIGKDD Int. Conf. Knowl. Discov. Data Min* 1100–1108 (2011).
301. Sweeney, L. Privacy technologies for homeland security. *nnnnn* (2005). at <http://www.dhs.gov/xlibrary/assets/privacy/privacy_advcom_06-2005_testimony_sweeney.pdf>

Index

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data, First Edition. Diane J. Cook and Narayanan C. Krishnan.

WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING
Series Editor: Albert Y. Zomaya

Parallel and Distributed Simulation Systems / Richard Fujimoto

Mobile Processing in Distributed and Open Environments / Peter Sapaty

Introduction to Parallel Algorithms / C. Xavier and S. S. Iyengar

Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences / Albert Y. Zomaya, Fikret Ercal, and Stephan Olariu (*Editors*)

Parallel and Distributed Computing: A Survey of Models, Paradigms, and Approaches / Claudia Leopold

Fundamentals of Distributed Object Systems: A CORBA Perspective / Zahir Tari and Omran Bukhres

Pipelined Processor Farms: Structured Design for Embedded Parallel Systems / Martin Fleury and Andrew Downton

Handbook of Wireless Networks and Mobile Computing / Ivan Stojmenović *(Editor)*

Internet-Based Workflow Management: Toward a Semantic Web / Dan C. Marinescu

Parallel Computing on Heterogeneous Networks / Alexey L. Lastovetsky

Performance Evaluation and Characterization of Parallel and Distributed Computing Tools / Salim Hariri and Manish Parashar

Distributed Computing: Fundamentals, Simulations, and Advanced Topics, *Second Edition* / Hagit Attiya and Jennifer Welch

Smart Environments: Technology, Protocols, and Applications / Diane Cook and Sajal Das

Fundamentals of Computer Organization and Architecture / Mostafa Abd-El-Barr and Hesham El-Rewini

Advanced Computer Architecture and Parallel Processing / Hesham El-Rewini and Mostafa Abd-El-Barr

UPC: Distributed Shared Memory Programming / Tarek El-Ghazawi, William Carlson, Thomas Sterling, and Katherine Yelick

Handbook of Sensor Networks: Algorithms and Architectures / Ivan Stojmenović *(Editor)*

Parallel Metaheuristics: A New Class of Algorithms / Enrique Alba (*Editor*)

Design and Analysis of Distributed Algorithms / Nicola Santoro

Task Scheduling for Parallel Systems / Oliver Sinnen

Computing for Numerical Methods Using Visual C++ / Shaharuddin Salleh, Albert Y. Zomaya, and Sakhinah A. Bakar

Architecture-Independent Programming for Wireless Sensor Networks / Amol B. Bakshi and Viktor K. Prasanna

High-Performance Parallel Database Processing and Grid Databases / David Taniar, Clement Leung, Wenny Rahayu, and Sushant Goel

Algorithms and Protocols for Wireless and Mobile Ad Hoc Networks / Azzedine Boukerche (*Editor*)

Algorithms and Protocols for Wireless Sensor Networks / Azzedine Boukerche (*Editor*)

Optimization Techniques for Solving Complex Problems / Enrique Alba, Christian Blum, Pedro Isasi, Coromoto León, and Juan Antonio Gómez (*Editors*)

Emerging Wireless LANs, Wireless PANs, and Wireless MANs: IEEE 802.11, IEEE 802.15, IEEE 802.16 Wireless Standard Family / Yang Xiao and Yi Pan (*Editors*)

High-Performance Heterogeneous Computing / Alexey L. Lastovetsky and Jack Dongarra

Mobile Intelligence / Laurence T. Yang, Augustinus Borgy Waluyo, Jianhua Ma, Ling Tan, and Bala Srinivasan (*Editors*)

Research in Mobile Intelligence / Laurence T. Yang *(Editor)*

Advanced Computational Infrastructures for Parallel and Distributed Adaptive Applicatons / Manish Parashar and Xiaolin Li *(Editors)*

Market-Oriented Grid and Utility Computing / Rajkumar Buyya and Kris Bubendorfer *(Editors)*

Cloud Computing Principles and Paradigms / Rajkumar Buyya, James Broberg, and Andrzej Goscinski (*Editors*)

Algorithms and Parallel Computing / Fayez Gebali

Energy-Efficient Distributed Computing Systems / Albert Y. Zomaya and Young Choon Lee *(Editors)*

Scalable Computing and Communications: Theory and Practice / Samee U. Khan, Lizhe Wang, and Albert Y. Zomaya (*Editors*)

The DATA Bonanza: Improving Knowledge Discovery in Science, Engineering, and Business / Malcolm Atkinson, Rob Baxter, Michelle Galea, Mark Parsons, Peter Brezany, Oscar Corcho, Jano van Hemert, and David Snelling *(Editors)*

Large Scale Network-Centric Distributed Systems / Hamid Sarbazi-Azad and Albert Y. Zomaya *(Editors)*

Verification of Communication Protocols in Web Services: Model-Checking Service Compositions / Zahir Tari, Peter Bertok, and Anshuman Mukherjee

High-Performance Computing on Complex Environments / Emmanuel Jeannot and Julius Žilinskas (*Editors*)

Advanced Content Delivery, Streaming, and Cloud Services / Mukaddim Pathan, Ramesh K. Sitaraman, and Dom Robinson *(Editors)*

Large-Scale Distributed Systems and Energy Efficiency / Jean-Marc Pierson (*Editor*)

Activity Learning: Discovering, Recognizing, and Predicting Human Behavior from Sensor Data / Diane J. Cook and Narayanan C. Krishnan

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.