HIERARCHICAL DISTRIBUTED CLASSIFICATION IN WIRELESS SENSOR NETWORKS

by

Ji Xu

BEng, Beijing University of Post and Telecommunications, 1998

a Thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY

Spring 2010

APPROVAL

Name: Ji Xu

Degree: Master of Science

Title of Thesis: Hierarchical Distributed Classification in Wireless Sensor Networks

Examining Committee: Miss. Anne Lavergne
Chair

Dr. Jian Pei, Senior Supervisor

Dr. Alexandra Fedorova, Supervisor

Dr. Jie Liang, SFU Examiner

Date Approved: July 22th, 2009

Declaration of
Partial Copyright Licence

Abstract

Wireless sensor networks promise an unprecedented opportunity to monitor the physical environment via inexpensive wireless device. Classification based on observations from distributed sensor nodes is an important application in wireless sensor networks. Developing an efficient classification method encounters many challenges in wireless sensor net works, such as accuracy and energy consumption.

In this thesis, we propose a novel hierarchical distributed classification approach, in which the sensor nodes locally build the classifier and send to parent, who combines the received classifiers by generating pseudo data training data distribution is studied. The simulation results show that our approach obtains a high accuracy and saves much energy.

Acknowledgments

This thesis is the result of the inspiring and thoughtful guidance and supervision of my senior supervisor, Professor Jian Pei at School of Computing Science, Simon Fraser University. I would like to thank him for his great help during my study at SFU. I would also like to thank Professor Alexandra Fedorova and Professor Jie Liang for serving in my thesis committee, and Miss Anne Lavergne for chairing it.

Thanks go to the members of Professor Pei's research group for the series of inspiring presentations and discussions. I am also grateful for the good time and friendship I enjoyed while I worked in the research group.

I would also like to express my gratitude to my collaborator, Mr. Xu Cheng, for all his encouragement and support during this time.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Wireless Sensor Networks

Wireless sensor networks (WSNs) have gained worldwide attention in recent years, particularly with the proliferation in Micro-Electro-Mechanical Systems (MEMS) technology which has facilitated the development of smart sensors [51]. A sensor network is composed of a large number of sensor nodes that are densely deployed either inside the phenomenon or very close to it. The position of sensor nodes need not be engineered or redetermined. The recent advances in transceiver and embedded hardware designs have made massive production of inexpensive wireless sensors possible [1]. These sensors are small, with limited processing and computing resources, and inexpensive compared to traditional sensors. These sensor nodes can sense, measure, and gather information from the environment and, based on some local decision process, can form a network with each node storing, processing and relaying the sensed data, often to a base station or user for further computation.

Smart sensor nodes are low power devices equipped with one or more sensors, a processor, memory, a power supply, a radio, and an actuator. A variety of mechanical, thermal, biological, chemical, optical, and magnetic sensors may be attached to the sensor node to measure properties of the environment. Since sensor nodes have limited memory and are typically deployed in difficult-to-access locations, a radio is implemented for wireless communication to transfer the data to a base station (e.g., a laptop, a personal hand held device, or an access point to a fixed infrastructure). Battery is the main power source in a sensor node. Secondary power supply that harvests power from the environment such as solar panels may be added to the node depending on the appropriateness of the environment

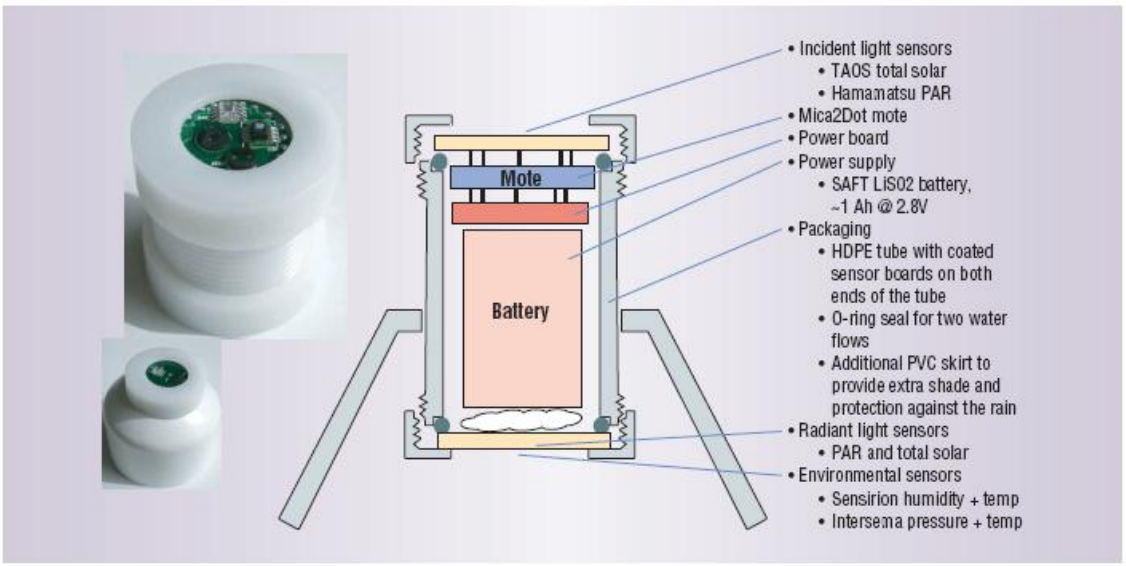

Figure 1.1: A example of sensor node (courtesy of [16])

where the sensor will be deployed. Depending on the application and the type of sensors used, actuators may be incorporated in the sensors. Figure 1.1 shows an example of a sensor node.

A WSN typically has little or no infrastructure. It consists of a number of sensor nodes (few tens to thousands) working together to monitor a region to obtain data about the environment. There are two types of WSNs: structured and unstructured. An unstructured WSN is one that contains a dense collection of sensor nodes. Sensor nodes may be deployed in an ad hoc manner into the field. Once deployed, the network is left unattended to perform monitoring and reporting functions. In an unstructured WSN, network maintenance such as managing connectivity and detecting failures is difficult since there are so many nodes. In a structured WSN, all or some of the sensor nodes are deployed in a pre-planned manner. The advantage of a structured network is that fewer nodes can be deployed with lower network maintenance and management cost. Fewer nodes can be deployed now since nodes are placed

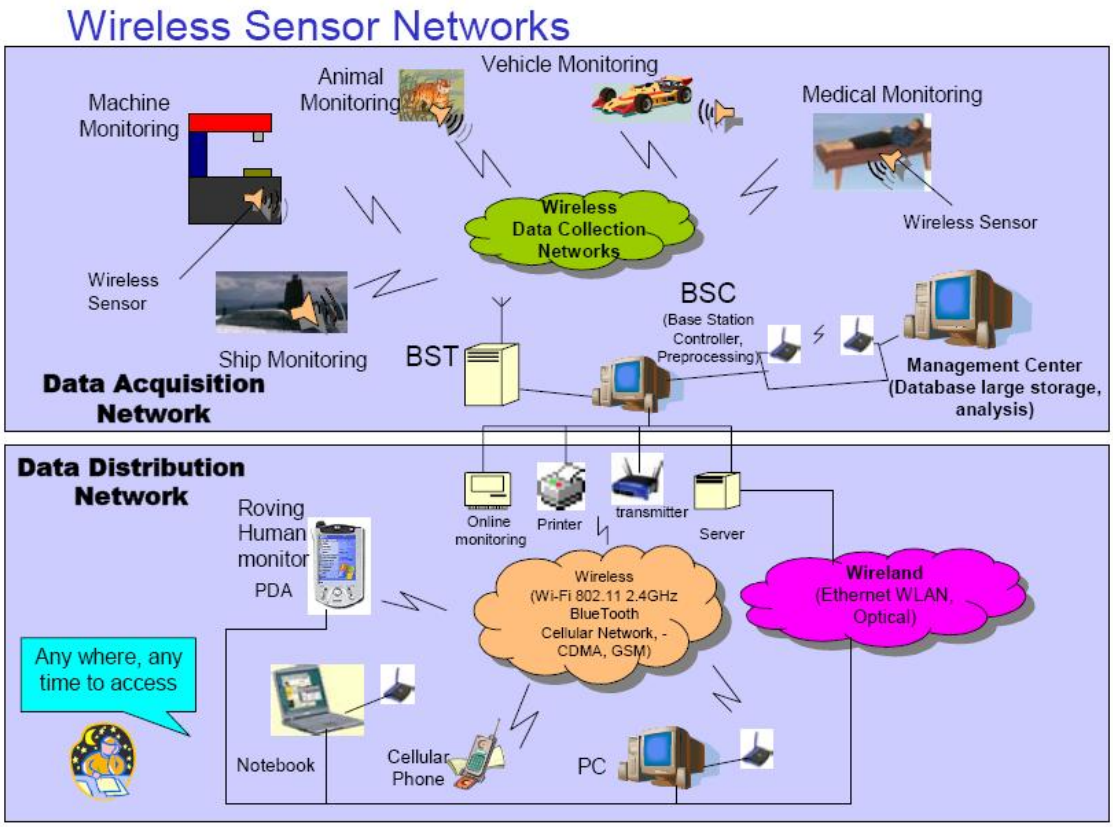

Figure 1.2: Wireless Sensor Networks

at specific locations to provide coverage while ad hoc deployment can have uncovered regions. Figure 1.2 shows an example of wireless sensor networks.

Wireless sensor networks can be used in many applications, such as environment surveillance, manufacturing management, business asset administration, automation in transportation and health-care industry, to provide us with real-time information about the environment and allow us to interact with the environment as needed. WSNs have great potential for many applications in scenarios such as military target tracking and surveillance [24, 50], natural disaster relief [11], biomedical health monitoring [22, 35], and hazardous environment exploration and seismic sensing [48]. In military target tracking and surveillance, a WSN can assist in intrusion detection and identification. Specific examples include spatially-correlated and coordinated troop and tank movements. Regarding natural disasters, sensor nodes can sense and detect the environment to forecast disasters before they occur. In

biomedical applications, surgical implants of sensors can help monitor a patient's health. For seismic sensing, ad hoc deployment of sensors along the volcanic area can detect the development of earthquakes and eruptions [51]. Given the huge amount of sensed data, classifying them becomes a critical task in many of these applications.

As an example, for wildlife monitoring, the sensor nodes continuously sense the physical phenomena such as temperature, humidity and sunlight, and meanwhile may also count the number of animals. The number reflects the suitability of the current environment for the animals, for example, if it is greater than a threshold, the environment is classified as suitable, and otherwise not. After learning the relation between the physical phenomena and the classes from such *training data*, we may later determine the suitability of the inquired environment from the external source. These inquires are the *unseen data* which only have the physical phenomena but do not have the class label. The sensor data collection and object count have been extensively studied in the literature, e.g., the Great Duck Island Project [37], yet efficient classification for wireless sensor networks has not been well addressed.

Unlike traditional networks, a WSN has its own design and resource constraints. Resource constraints include a limited amount of energy, short communication range, low bandwidth, and limited processing and storage in each node. Design constraints are application dependent and are based on the monitored environment. The environment plays a key role in determining the size of the network, the deployment scheme, and the network topology. The size of the network varies with the monitored environment. For indoor environments, fewer nodes are required to form a network in a limited space whereas outdoor environments may require more nodes to cover a larger area. An ad hoc deployment is preferred over pre-planned deployment when the environment is inaccessible by humans or when the network is composed of hundreds to thousands of nodes. Obstructions in the environment can also limit communication between nodes, which in turn affects the network connectivity (or topology).

Energy conservation in a WSN maximizes network lifetime and is addressed through efficient reliable wireless communication, intelligent sensor placement to achieve adequate coverage, security and efficient storage management, and through data aggregation and data compression. The above approaches aim to satisfy both the energy constraint and provide quality of service for the application. For reliable communication, services such as congestion control, active buffer monitoring, acknowledgements, and packet-loss recovery are

necessary to guarantee reliable packet delivery. Communication strength is dependent on the placement of sensor nodes. Sparse sensor placement may result in long-range transmission and higher energy usage while dense sensor placement may result in short-range transmission and less energy consumption. Coverage is interrelated to sensor placement. The total number of sensors in the network and their placement determine the degree of network coverage. Depending on the application, a higher degree of coverage may be required to increase the accuracy of the sensed data.

WSN applications can be classified into two categories: monitoring and tracking [51]. Monitoring applications include indoor/outdoor environmental monitoring, health and wellness monitoring, power monitoring, inventory location monitoring, factory and process automation, and seismic and structural monitoring. Tracking applications include tracking objects, animals, humans, and vehicles. Research in WSNs aims to meet the above constraints by introducing new design concepts, creating or improving existing protocols, building new applications, and developing new algorithms.

1.2 Data Mining on WSNs

Data Mining based on Wireless Sensor Network data has been an issue over the last years. While WSN deliver data that might not have been accessible before, mining WSN data has to face the limitations of WSNs and the special characteristics of WSN data [31]. A first task is deciding about the data processing which is distributed in the system. It might be completely done at the sink[1] (here all measured data is delivered to the sink) or it might be done completely within the network. Both extremes define the range of possible distributions. Generally, data processing within the network might reduce communication costs because not all captured sensor data but the result of data processing has to be delivered to the sink. However, in settings where the result of data processing is intended to be used within the network (e.g, adaption of network behavior to the environment), even the costs for node-to-sink delivery might be avoided.

Classification is an important problem in the field of data mining. The starting point of classification is a training set of example records, where each record consists of several attributes $(\mathbf{x}, y)$. These attributes may be either continuous, of an ordered domain, or

[1] In the WSN literature, the sink is the network component that gathers all sensor measurements, and is usually considered to benefit from higher computational resources than the sensor nodes

categorical. One of the attributes, called the classifying attribute Y, determines the class which a record belongs to. The classification problem is to construct a model that defines a function $h : \{\mathbf{x}\} \Rightarrow \{y\}$ which is called a classifier. The goal is to classify of future yet unclassified records based on h. In WSNs the construction of a classifier might be necessary if a WSN is intended to classify phenomena in its sensing environment. The classification task could be an innovative application where no or little information about the classifier is available or an existing classifier may not be parameterizable for its application in a new environment.

Several models for classification exist, for example neural networks, statistical models, decision trees and genetic models [46]. Among these, decision trees can be constructed relatively fast and are simple. Since the operations that are needed to implement a decision tree are relatively simple, it is well fitted to the limited computing power of sensor nodes. Furthermore, decision trees obtain similar and sometimes better accuracy compared to other classification models.

1.3 Motivation and Contribution

Classification is typically done in two steps: first, a classifier is constructed to summarize a set of predetermined classes, by learning from a set of training data; then, the classifier is used to determine the classes of newly-arrived data. Within this framework, there have been significant efforts in improving its speed and accuracy, most of which assume centralized storage and computation. The wireless sensor networks however pose a series of new challenges, particularly for the first step. First, the number of sensor nodes is huge, but each of them has only limited storage that can hardly accommodate all the training data of the whole network. Second, the sensor nodes are generally powered by non-replenishable batteries, and energy efficiency is thus of paramount importance, which makes the straightforward solution of sending all training data to powerful base station quite inefficient [19, 27]. In short, conventional centralized solution is not directly applicable in this new type of network environment.

To address the above challenges, in this thesis, we present a novel distributed solution for classification in energy-constrained sensor networks. Our approach hierarchically organizes the sensor nodes and performs classification in a localized and iterative manner, utilizing a decision tree method. Starting from each leaf node, a classifier is built based on its local

training data. An upstream node, upon receiving the classifiers from its children, will use them and its own data to build a new classifier. These local classifiers will be iteratively enhanced from bottom to top and finally reach the base station, making a global classifier for all the data distributed across the sensor nodes. Since only the classifiers will be forwarded upstream, the energy consumption for transmission can be significantly reduced.

The key difficulty here however lies in training a new classifier from a mix of downstream classifiers and the local training dataset, which cannot be directly accomplished by existing learning algorithms that work on dataset only. We address this problem by generating a *pseudo training dataset* from each downstream classifier. We develop a smart generation algorithm, which ensures that the pseudo data closely reflects the characteristics of the original local data. We also introduce a control parameter that adaptively balances the recovery quality and data size. Through extensive simulations, we demonstrate that our approach maintains high classification accuracy, with very low storage and communication overhead.

We also notice that, in practice, the data distribution across different sensor nodes is not necessarily homogeneous. For example, depending on the location, the data sensed by one node may always have low temperature and low humidity, and the data sensed by another node at a different location may always have high temperature and high humidity. We show strong evidence that such heterogeneity can easily lead to mis-classification, and we propose an enhanced ID3 algorithm to mitigate its impact. To the best of our knowledge, our study is the first solution addressing this issue.

1.4 Organization of the Thesis

The remainder of the thesis is structured as follows: In Chapter 2, we present an overview of related work. In Chapter 3, we describe our approach. The experimental results are shown in Chapter 4. Finally, Chapter 5 concludes the thesis and discusses some directions for future research.

Chapter 2

Related Work

In this section, we will give an overview of the work previously done on the topics related to our study. In more specific, we will discuss the following: (1) wireless sensor networks techniques,(2) data gathering in WSNs,(3) the classification problem and general approach,(4) ensemble methods.

2.1 Wireless Sensor Networks Techniques: An Overview

To enable wireless sensor applications using sensor technologies, each sensor node is an individual system. In order to support different application software on a sensor system, development of new platforms, operating systems, and storage schemes are needed. The communication protocols, which enable communication between the application and sensors, also enable communication between the sensor nodes. Services are developed to enhance the application and to improve system performance and network efficiency. From application requirements and network management perspectives, it is important that sensor nodes are capable of self-organizing themselves. That is, the sensor nodes can organize themselves into a network and subsequently are able to control and manage themselves efficiently. As sensor nodes are limited in power, processing capacity, storage, new communication protocols and management services are needed to fulfil these requirements. In this section, some techniques of wireless sensor networks will be reviewed briefly.

2.1.1 Embedded Network Technology

WSNs merge a wide range of information technologies that span hardware, systems software, networking, and programming methodologies [16].

Microprocessors, power, and storage

The hardware of a sensor network node consists of a microprocessor, data storage, sensors, analog-to-digital converters (ADCs), a data transceiver, controllers that tie the pieces together, and an energy source. Recently, a new operating point has emerged that suits all these components. As semiconductor circuits become smaller, they consume less power for a given clock frequency and fit in a smaller area. In simple microcontrollers, miniaturization increases efficiency rather than adding functionality, allowing them to operate near one milliwatt while running at about 10 MHz. Most of the circuits can be powered off, so the standby power can be about one microwatt. If such a device is active 1 percent of the time, its average power consumption is just a few microwatts. This scale of power can be obtained in many ways. Solar cells generate about 10 milliwatts per square centimeter outdoors and 10 to 100 microwatts per square centimeter indoors. Mechanical sources of energy, such as the vibration of windows and air conditioning ducts, can generate about 100 microwatts [24]. A typical cubic-centimeter battery stores about 1,000 milliamp-hours, so centimeter-scale devices can run almost indefinitely in many environments. However, low-power microprocessors have limited storage, typically less than 10 Kbytes of RAM for data and less than 100 Kbytes of ROM for program storage-or about 10,000 times less storage capacity than a PC has. This limited amount of memory consumes most of the chip area and much of the power budget. Designers typically incorporate larger amounts of flash storage, perhaps a megabyte, on a separate chip.

Microsensors

Sensors give these nodes their eyes and ears. Many materials change their electrical characteristics when subjected to varying environmental conditions. Sensors are manufactured so these changes are predictable over a certain range. For example, a thermistor is a variable resistor that changes smoothly with temperature. An ADC converts the voltage drop into a binary number that a microcontroller can store or process. Photocells and fog detectors work similarly, but they consist of finely interleaved combs separated by a material that

uses incident photons or moisture to change resistance. Many more sophisticated structures have been developed to detect other phenomena [30]. These structures consume a few milliwatts and only need to be turned on a fraction of the time. Extremely efficient ADCs have been developed so that the sensor subsystem has an energy profile similar to the processor. Microelectromechanical systems (MEMS) can sense a wide variety of physical phenomena cheaply and efficiently. Researchers can use the processes for etching transistors on silicon to carve out tiny mechanical structures, such as a microscopic springboard within an open cavity. Gravitational forces or acceleration can deflect this cantilevered mass, causing powerful internal forces that cause changes in material properties or delicate alignments, which can be amplified and digitized. Manufacturers used the first major commercial MEMS sensors, the accelerometer, to trigger automotive airbag release. Whereas high-precision piezoelectric accelerometers cost hundreds of dollars, MEMS provided sufficient precision for a few dollars. Once the devices entered mass production, they could ride the CMOS technology growth of modern chips to become increasingly accurate while remaining inexpensive. A wide variety of MEMS devices can sense various forces, chemical concentrations, and environmental factors.

Microradios

For some time now, manufacturers have added sensors to many appliances, vehicles, and gadgets. The breakthrough comes from communicating sensor readings to other devices, translating the physical world into information-bits-that the devices can transport, store, and process. Radio components can now be manufactured using conventional CMOS technology, enabling wireless entry devices, pagers, walkie-talkies, cell phones, and wireless local area networks for mobile laptops. However, the amount of energy required to communicate wirelessly increases rapidly with distance. Obstructions such as people or walls and interference further attenuate the signal. Wireless LANs and cell phones consume hundreds of milliwatts and rely on a powerful infrastructure. WSN radios consume about 20 milliwatts, and their range typically is measured in tens of meters. For small devices to cover long distances, the network must route the information hop by hop through nodes. Even so, communication remains one of the most energy-consuming operations, with each bit costing as much energy as about 1,000 instructions. Thus, WSNs process data within the network wherever possible [27, 49].

2.1.2 Systems Challenge

Bridging the gap between the hardware technology's raw potential and the broad range of applications presents a system challenge. The network must allocate limited hardware to multiple concurrent activities, such as sampling sensors, processing, and streaming data. The potential interconnections between devices must be discovered and information routed effectively from where it is produced to where it is used. There must also be a means of programming the ensemble.

TinyOS

Conventional operating systems such as Unix run well on a 32-bit microprocessor at 50 to 100 MHz, with several megabytes of RAM and a gigabyte or more of secondary storage. Today, this can be achieved in a handheld device that runs for several hours on a single charge [4]. A more typical operating point for WSNs is one year on a pair of AA batteries with a small fraction of these resources. Further, this application focuses on structured interaction with the physical world, rather than on complex human interactivity. The developers of the open source TinyOS tailored it for this application. The TinyOS provides a framework for assembling application-specific systems that can handle substantial concurrency within limited physical resources. The software components and the underlying operating system support specific eventdriven functionality. The lowest-level components abstract the physical hardware and deliver physical interrupts as sanitized, asynchronous events. Each component handles certain events and signals actions in other components, but they never consume processor cycles while waiting for future events. Each application includes only the components it requires. For example, a small stack of components process sensor readings. The lowest components handle the ADC to obtain raw readings, whereas the higher ones filter and distill data streams for the application. The network involves a more complex stack in which the lower levels deal with acquiring the radio channel, framing data streams into packets that receiving nodes can recognize. These components also perform error coding and channel scheduling, as well as detecting the arrival of incoming packets and processing them into input buffers. Higher levels deal with buffer management, authentication, and multiplexing the network across application components. A typical top-level application might receive and process a stream of filtered sensor readings, then deliver important notifications to the network. A second component would receive such notification

messages, maintain a routing structure, and retransmit them along the next hop in a route to a data collection gateway.

Network sensor platforms

Berkeley motes and TinyOS are widely used for exploring system issues and deploying pilot applications. The microcontroller provides a modest amount of RAM and program storage, and contains an internal ADC. A simple frequency-agile radio with roughly the bandwidth of a modem provides the connectivity that developers can use to construct a network. Off-chip flash memory provides storage to hold both the program while it transfers through the network and the data buffering beyond the on-chip RAM. Several sensor boards have been designed for this platform. The Intel iMote is a recent integrated design that uses a commercial chip with a powerful ARM microprocessor, storage, and radio integrated into a single package. The radio implements the Bluetooth standard, which is becoming widely used in laptops and cell phones. The radio operates at higher bandwidth and has a sophisticated frequency-hopping protocol. Normally, the ARM processor would be devoted to managing the Bluetooth radio and transferring packets to and from a serial port. In the iMote, however, TinyOS runs directly on the ARM processor, providing a stand-alone system that services various sensors and routes, processes high-level information streams, and manages power consumption. Most of the lower-level TinyOS components are implemented directly in hardware. These include an extremely low-power ADC and a very efficient radio developed in the Smart Dust project (http:// robotics.eecs.berkeley.edu/ pister/SmartDust/). The entire design occupies only 5 square millimeters. It is estimated that at 1 percent active, the chip could run a hundred years on the energy stored in a pair of AA batteries. On the other end of the spectrum of in situ nodes lie 32-bit processor-based devices such as Stargate, which run traditional operating systems such as Linux that are equipped with longer-range radios such as IEEE 802.11 or with cell phone modems. This node class will play a critical role in most deployed systems. At a minimum, these nodes will act as gateway points both to retrieve data off the sensor network and to monitor, configure, and task the system. In more sophisticated heterogeneous systems, these nodes will be distributed more widely and used as points of aggregation, data storage, data fusion, and hosts for higher-end sensors. Because they are so much more energy intensive, these nodes operate along with a large battery and some form of recharge such as a solar panel. Alternatively, when wall outlets are available, the nodes can draw power from them.

2.1.3 Self-organized Networks

Wireless communication and instrumentation have long been associated with remote sensing from satellites and missile telemetryprime examples of wireless links. A network consists of many nodes, each with multiple links connecting to other nodes. Information moves hop by hop along a route from the point of production to the point of use. In a wired network like the Internet, each router connects to a specific set of other routers, forming a routing graph. In WSNs, each node has a radio that provides a set of communication links to nearby nodes. By exchanging information, nodes can discover their neighbors and perform a distributed algorithm to determine how to route data according to the applications needs. Although physical placement primarily determines connectivity, variables such as obstructions, interference, environmental factors, antenna orientation, and mobility make determining connectivity a priori difficult. Instead, the network discovers and adapts to whatever connectivity is present [25].

Connectivity

The networking capability of WSNs is built up in layers [2]. The lowest layer controls the physical radio device. Radios are by nature a broadcast medium: When one node transmits, a collection of others can receive the signal unless it is garbled by other transmissions at the same time. To avoid contending for the radio channel, the link layer listens on the channel and transmits only when the channel is clear. It transmits a structured series of bits that form a packet encoded in the radio signal. When not transmitting, nodes sample the channel and scan for a special symbol at the start of a packet that also lets the receiver align itself with the senders time. The packet layer manages buffers, schedules packets onto the radio, detects or even corrects errors, handles packet losses, and dispatches packets to system or application components.

Dissemination and data collection

Developers use this basic communication capacity to implement protocols that let the collection of nodes transport and process information and coordinate their activities. A basic capability in such networks involves disseminating information over many nodes. This can be achieved by a flooding protocol in which a root node broadcasts a packet with some identifying information. Receiving nodes retransmit the packet so that more distant nodes can

receive it. However, a node can receive different versions of the same message from several neighboring nodes, so the network uses the identifying information to detect and suppress duplicates. Flooding protocols use various techniques to avoid contention and minimize redundant transmissions. The network uses dissemination to issue commands, convey alarms, and configure and task the network. But it also uses dissemination to establish routes. Each packet identifies the transmitter and its distance from the root. To form a distributed tree, nodes record the identity of a node closer to the root. The network can use this reverse communication tree for data collection by routing data back to the root or for data aggregation by processing data at each level of the tree. The root can be a gateway to a more powerful network or an aggregation point within the sensor network, as determined by some higher-level task. Often, tree formation and data collection are interwoven. Data can begin following up the tree as soon as a parent node is discovered. Nodes may learn of potential parents by overhearing data messages. The network continually collects statistics to reinforce the best routes. These communication patterns differ significantly from those found on the Internet, where many client computers open connections to named servers and transfer large streams of data back and forth. In sensor networks, communication is usually performed in the aggregate, and participants are identified by attributes such as physical location or sensor value range. This style of routing has been formulated as directed diffusion, a process in which nodes express interest in data by attribute. The nodes flood interest outward to form a routing gradient, and they collect data up the gradient by reinforcing the associated subtree [34]. Reliability also follows a different pattern. Increasingly, sensor networks will deploy disruptiontolerant networking approaches in which they transfer bundles of data reliably, hop by hop, in contrast to the Internet, which sets up an end-toend connection using byte or packet matching between the original source and destination to determine reliability. The DTN model better suits the variable connectivity that results from dynamic environments and the need to duty-cycle.

2.1.4 Conserving Power and Bandwidth

Communication, usually the most energyintensive operation a node performs, must contend for a share of limited bandwidth. The network stack attempts to minimize energy usage, either by eliminating communication or by turning off the radio when no communication needs to occur. Several approaches are possible. For example, nodes could process data locally and only communicate when they detect an interesting event. This approach would be

employed in an intelligent alarm system or an environmental monitoring system that focuses data collection on time or areas of interest. In many cases, crude low-power sensors trigger higher-powered sensor devices, such as cameras. Performing aggregation within the network can reduce communication. For example, an application might need to determine the average temperature at shaded nodes in a certain geographic region. Selecting the subset of readings of interest could be performed at the trees leaves, routing their aggregation as data upward, so that each node transmits at most a single packet to provide a statistical summary of its subtree. More sophisticated aggregation could involve detecting distributed regions of interest. Compression and scheduling also can conserve energy at lower layers. Some protocol overhead is associated with data communication to maintain routing structures, manage contention, and enhance reliability. Sensor networks can avoid explicit protocol messages by piggybacking control information on data messages and by overhearing packets destined for other nodes [1]. They can use prescheduled time to reduce contention and the time the radio remains live. This can be coordinated with the high-level application behavior by, for example, periodic low-rate data sampling. Alternatively, the network could implement energy conservation generically within lower layers by, for example, time division multiple access. In the spatial dimension, the network can assign specific responsibilities to certain nodes, such as retransmission or aggregation. Finally, the network can reject uninteresting packets by turning off the radio after receiving only a portion. However, because these many optimizations can be mutually conflicting, a rich and growing body of literature employs different combinations of techniques under different application and platform assumptions.

2.1.5 Privacy

Dense instrumentation, real-time access, and in-network processing make a qualitative difference in our ability to perceive what is happening throughout large physical structures. In environmental monitoring and condition- based maintenance, the purpose of data collection, the parties responsible for using the data, and the scope of dissemination are clear. The situation becomes much less clear in more casual settings in which more general human activity occurs, such as the home, the workplace, a transportation terminal, or a shopping venue. In these cases, many potentially interested parties can have varying uses for the data. More detailed sensingsuch as occupancy, motion, and even physiological statefurther amplifies concerns over proper use and dissemination. Indeed, image data, such as that obtained by the surveillance cameras in pervasive use today, can be viewed as an extremely

powerful sensor, but network access and automated analysis are limited [51]. These social factors are an inherent concern with sensor network technology. Fortunately, this area has become an active focus of research while the technology is still in its early stages.

2.2 Data Gathering in WSNs

As a type of newly emerged network, WSN has many special features comparing with traditional networks such as Internet, wireless mesh network and wireless mobile ad hoc network. First of all, a sensor node after deployed is expected to work for days, weeks or even years without further interventions. Since it is powered by the attached battery, high efficient energy utilization is necessary, which is different from Internet as well as wireless mesh and mobile ad hoc network, where either constant power sources are available or the expected lifetime is several order of magnitude lower than it is for WSNs.

Although a sensor node is expected to work through a long time, it is often not required to work all the time, i.e., it senses ambient environment, processes and transmits the collected data, and then waits for a while until the next sensing-processing-transmitting cycle. To support fault tolerance, a location is often covered by several sensor nodes. To avoid duplicate sensing, while one node is performing the sensing-processing-transmitting cycle, other nodes are also in the waiting state. In these cases, energy consumption can be further reduced by letting the waiting nodes turn to *dormant* state, where most of the components (wireless radio, sensing component, processing unit) in a sensor node are turned off, and when the next cycle comes, turned back to the normal (*active*) state again. Define *duty-cycle* as the ratio between active period and the full active/dormant period. A low duty-cycle WSN clearly enjoys a much longer lifetime for operation [34].

Another special feature related to energy consumption is to control the transmission range of a sensor node. Previous researches have shown that one of the major energy costs in a sensor node comes from the wireless communication, where the main factor increases with the 2 to 6 power of the transmission distance [4]. As a result, the transmission range of a sensor node is often considered adjustable and may be dynamically adjusted to achieve better performance and lower energy consumption.

In a data gathering application, sensors are often deployed at locations specified by the application requirement to collect sensing data. The collected data is then forwarded to a central base station for further processing. Traditionally, these sensors are connected by

wires which are used for data transmission and power supply. However, the wired approach is found demanding great efforts for deployment and maintenance. To avoid disturbing the ambient environment, the deployment of the wires has to be carefully designed. And a breakdown in any wire may cause the whole network out of service and enormous time and efforts may be taken to find out and replace the broken line. In addition, the sensing environment itself may make the wired deployment and its maintenance very difficult, if not impossible. For example, the environment near a volcano or a wildfire scene, where the hot gases and steams can damage a wire easily. Indeed, even in a less harsh environment like wild habitat or a building, the threats from rodents are still critical and make the protection of wires much more difficult than that of sensors. All these issues make wireless sensor network a pleasant choice as it emerges with technology advances.

On the other hand, although many research efforts have been done on WSNs, and quite a few prototype or preliminary systems have been deployed, data collection in WSNs is still in its early stage and its special features call for novel approaches and solutions different from other applications. For example, a common work pattern in most of other applications, such as target tracking, is that sensing data or information is locally processed and stored at some nodes and may be queried later by some other nodes. Data collection, nevertheless, requires all sensing data be correctly and accurately collected and forwarded to the base station, since the processing of the data needs global knowledge and is much more complex than that in other applications like target tracking. Thus the major traffic in data collection is the reported data from each sensor to the base station. Such "many-to-one" traffic patterns, if not carefully handled, will cause high unbalance and inefficiency of energy consumption in the whole network.

There have been numerous works on data gathering in wireless sensor networks. For example, LEACH [27] and PEGASIS [34] attempt to make the data collection task energy efficient. There are also a few related works about classification in this context [10, 2, 25], but they have focused on detecting or tracking objects, while not giving detailed design of classifiers. To the best of our knowledge, our work is the first addressing energy-efficient distributed classification for wireless sensor networks, particularly with heterogeneous data distribution across the sensor nodes.

2.3 The Classification Problem and General Approach

Classification, which is the task of assigning objects to one of several predefined categories, is a pervasive problem that encompasses many diverse applications [46]. Data classification is a two-step process [26]. In the first step, a model is built describing a predetermined set of data classes or concepts. The model is constructed by analyzing database tuples described by attributes. Each tuple is assumed to belong to a predefined class, as determined by one of the attributes, called the class label attribute. In the context of classification, data tuples are also referred to as *samples*, or *objects*. The data tuples analyzed to build the model collectively form the *training data set*. The individual tuples making up the training set are referred to as *training samples* and are randomly selected from the sample population. Since the class label of each training sample is provided, this step is also known as *supervised learning*. Typically, the learned model is represented in the form of classification rules, decision trees, or mathematical formulae.

In the second step, the model is used for classification. First, the predictive accuracy of the model (or classifier) is estimated. The *holdout method* is a simple technique that uses a *test set* of class-labeled samples. These samples are randomly selected and are independent of the training samples. The *accuracy* of a model on a given test set is the percentage of test set samples that are correctly classified by the model. For each test sample, the known class label is compared with the learned model's class prediction for that sample. Note that if the accuracy of the model is estimated based on the training data set, this estimate can be optimistic since the learned model tends to *overfit* the data. Therefore, a test set is used. If the accuracy of the model is considered acceptable, the model can be used to classify future data tuples of objects for which the class label is not known.

The input data for a classification model is a collection of records. Each sample is characterized by a tuple $(\mathbf{x}, y)$, where $\mathbf{x}$ is the attribute set and y is a special attribute, designated as the class label.

In the following, we will generally review some solutions of the classification problem.

2.3.1 Decision tree

A decision tree has three types of nodes: (1) A root node that has no incoming edges and zero or more outgoing edges; (2) Internal nodes, each of which has exactly one incoming edge and two or more outgoing edges; (3) Leaf or terminal nodes, each of which has exactly one

incoming edge and no outgoing edges. In a decision tree, each leaf node is assigned a class label. The non-terminal nodes, which include the root and other internal nodes, contain attribute test conditions to separate records that have different characteristics. Classifying a test record is straightforward once a decision tree has been constructed. Starting from the root node, the test condition is applied to the record and follow the appropriate branch based on the outcome of the test. This will lead to either another internal node, for which a new test condition is applied, or a leaf node. The class label associated with the leaf node is then assigned to the record.

In principle, there are exponentially many decision trees that can be constructed from a given set of attributes. While some of the trees are more accurate than others, finding the optimal tree is inefficient and costly because of the exponential size of the search space. Nevertheless, efficient algorithms have been developed to induce a reasonably accurate, albeit suboptimal,decision tree in a reasonable amount of time. These algorithms usually employ a greedy strategy that grows a decision tree by making a series of locally optimal decisions about which attribute to use for partitioning the data.

A learning algorithm for inducing decision tree must address two issues: (1) How should the training records be split? Each recursive step of the tree-growing process must select an attribute test condition to divide the records into smaller subsets. To implement this step, the algorithm must provide a method for specifying the test condition for different attribute types as well as an objective measure for evaluating the goodness of each test condition. (2) How should the splitting procedure stop? A stopping condition is needed to terminate the tree-growing process. A possible strategy is to continue expanding a node until either all the records in a node belong to the same class or all the records have identical attribute values. Although both conditions are sufficient to stop any decision tree induction algorithm, other criteria can be imposed to allow the tree-growing procedure to terminate earlier.

In Section 3.2, we will discuss more details of decision tree.

2.3.2 Bayesian Classifiers

In many applications the relationship between the attribute set and the class variable is non-deterministic. In other words, the class label of a test record cannot be predicted certainly even though its attribute set is identical to some of the training examples. This situation may arise because of noisy data or the presence of certain confounding factors affecting classification but are not included in the analysis. The bayesian classifier model,

which is based on the *Bayes theorem*, is an approach for modeling probabilistic relationships between the attribute set and the class variable. Let X be a data sample whose class label is unknown, and H be a hypothesis, such as that the data sample X belongs to a specified class C. For classification problems, we want to determine $P(H|X)$, the probability that the hypothesis H holds given the observed data sample X. Since $P(H)$ is the prior probability of H and $P(H|X)$ is the posterior probability of H conditioned on X, the Bayes theorem provides a way of calculating the posterior probability. Bayes theorem can be written as $P(H|X) = \frac{P(X|H)P(H)}{P(X)}$. A naïve Bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label y. We only have to estimate the conditional probability of each X_i, given the class variable Y. To classify a test record, the naïve Bayes classifier computes the posterior probability for each class Y by:

$$P(Y|X) = \frac{P(Y)\prod_{i=1}^{d} P(X_i|Y)}{P(X)}. \tag{2.1}$$

The naïve Bayesian classifier makes the independence assumption which simplifies computation. However, in practice, dependencies can exist between variables. Bayesian belief networks specify joint conditional probability distributions. They allow class conditional independencies to be defined between subsets of variables, and provide a graphical model of causal relationships, in which learning can be performed. These networks are also known as belief networks.

2.3.3 Artificial Neural Networks

The study of artificial neural networks (ANN) was inspired by attempts to simulate biological neural systems [46]. The human brain consists primarily of nerve cells called *neurons*, linked together with other neurons via strands of fiber called *axons*. Axons are used to transmit nerve impulses from one neuron to another whenever the neurons are stimulated. A neuron is connected to the axons of other neurons via *dendrites*, which are extensions from the cell body of the neuron. The contact point between a dendrite and an axon is called a *synapse*. Neurologists have discovered that a human brain learns by changing the strength of the synaptic connection between neurons upon repeated stimulation by the same impulse. Roughly speaking, a neural network is a set of connected input/output units where each connection has a weight associated with it.

Analogous to human brain structure, an ANN is composed of an inter-connected assembly of nodes and directed links. *Perceptron*, a simple neural network architecture, consists of two types of nodes: input nodes, which are used to represent the input attributes, and an output node, which is used to represent the model output. The nodes in a neural network architecture are commonly known as neurons or units. In a perceptron, each input node is connected via a weighted link to the output node. The weighted link is used emulate the strength of synaptic connection between neurons. As in biological neural systems, training a perceptron model amounts to adapting the weights of the links until they fit the input-output relationships of the underlying data.

An artificial neural network has a more complex structure than that of a perceptron model. The network may contain several intermediary layers between its input and output layers. Such intermediary layers are called hidden layers and the nodes emgbedded in these layers are called hidden nodes. In a feed-forward neural network, the nodes in one layer are connected only to the nodes in the next layer. The perceptron is a single-layer, feed-forward neural network because it has only one layer of nodes, the output layer, that performs complex mathematical operations. In a recurrent neural network, the links may connect nodes within the same layer or nodes from one layer to the previous layers. The network also may use types of activation functions other than the sign function. these activation functions allow the hidden and output nodes to produce output values that are nonlinear in their input parameters. These additional complexities allow multilayer neural networks to model more complex relationships between the input and output variables.

The backpropagation algorithm performs learning on a multilayer feed-forward neural network, and learns by iteratively processing a set of training samples, comparing the network's prediction for each sample with the actual known class label. For each training sample, the weights are modified so as to minimize the mean squared error between the network's prediction and the actual class. These modifications are made in the "backwards" direction, that is, from the output layer, through each hidden layer down to the first hidden layer (hence the name *backpropagation*). Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops.

2.3.4 Support Vector Machine (SVM)

A classification technique that has received considerable attention is support vector machine (SVM) [46]. This technique has its roots in statistical learning theory and has shown promising empirical results in many practical applications, from handwritten digit recognition to text categorization. SVM also works very well with high-dimensional data and avoids the curse of dimensionality problem. Another unique aspect of this approach is that it represents the decision boundary using a subset of the training examples, know as the *support vectors*. Decision boundaries with large margins tend to have better generalization errors than those with small margins. Intuitively, if the margin is small, then any slight perturbations to the decision boundary can have quite a signification impact on its classification. Classifiers that produce decision boundaries with small margins are therefore more susceptible to model overfitting and tend to generalize poorly on previously unseen examples. The capacity of a linear model is inversely related to its margin. The models with small margins have higher capacities because they are more flexible and can fit many training sets, unlike models with large margins. Therefore, it is desirable to design linear classifiers that maximize the margins of their decision boundaries in order to ensure that their worst-case generalization errors are minimized. A linear SVM is a classifier that searches for a hyperplane with the largest margin, that is, finds a hyperplane separating the positive and negative samples. The decision boundary of a linear classifier can be written in the following form:

$$\mathbf{w} \cdot x + b = 0, \tag{2.2}$$

where $\mathbf{w}$ and b are parameters of the model. We can rescale the parameters $\mathbf{w}$ and b of the decision boundary so that the two parallel hyperhplanes b_{i1} and b_{i2} can be expressed as follows:

$$b_{i1} : \mathbf{w} \cdot x + b = 1, \tag{2.3}$$

$$b_{i2} : \mathbf{w} \cdot x + b = -1. \tag{2.4}$$

The margin of the decision boundary is given by the distance between these two hyperplanes. For the data sets that have nonlinear decision boundaries, we transform the data from its original coordinate space $\mathbf{x}$ into a new space $\Phi(\mathbf{x})$ so that a linear decision boundary can be used to separate the instances in the transformed space.

2.3.5 Other Classification Methods

- **k-Nearest Neighbor Classifier**
  Nearest neighbor classifiers are based on learning by analogy. The training samples are described by n-dimensional numeric attributes. Each sample represents a point in an n-dimensional space. In this way, all of the training samples are stored in an n-dimensional pattern space. When given an unknown sample, a k-nearest neighbor classifier searches the pattern space for the k training samples that are closest to the unknown sample. These k training samples are the k "nearest neighbors" of the unknown sample. "Closeness" is defined in terms of Euclidean distance, where the Euclidean distance between two points, $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ is $d(X, Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. The unknown sample is assigned the most common class among its k nearest neighbors. When $k = 1$, the unknown sample is assigned the class of the training sample that is closest to it in the pattern space.

- **Case-Based Reasoning**
  Case-based reasoning(CBR) classifiers are instanced-based. The samples or "cases" stored by CBR are complex symbolic descriptions. Business applications of CBR include problem resolution for customer service help desks, for example, where cases describe product-related diagnostic problems. CBR has also been applied to areas such as engineering and law, where cases are either technical designs or legal rulings, respectively. When given a new case to classify, a case-based reasoner will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. if no identical case is found, then the case-based reasoner will search for training cases having components that are similar to those of the new case. Conceptually, these training cases may be considered as neighbors of the new case. The case-based reasoner tries to combine the solutions of the neighboring training cases in order to propose a solution for the new case.

2.4 Ensemble Method

Learning algorithms try to find a hypothesis in a given space H of hypotheses and in many cases if we have sufficient data they can find the optimal one for a given problem. But in real cases we have only limited data sets and sometimes only few examples are available

[29]. In these cases the learning algorithm can find different hypotheses that appear equally accurate with respect to the available training data, and although we can sometimes select among them the simplest or the one with the lowest capacity, we can avoid the problem combining them to get a good approximation of the unknown true hypothesis. Thus, there is a growing realization that combinations of classifiers can be more effective than single classifiers. Why rely on the best single classifier, when a more reliable and accurate result can be obtained from a combination of several? This essentially is the rationale behind the idea of multiple classifier systems.

Ensemble methods are to construct a set of base classifiers and take a majority voting on predictions in classification. The main idea is to learn an ensemble of classifiers from subsets of data, and combine predictions from all these classifiers in order to achieve high classification accuracies. The method can significantly improve the accuracy of prediction, because if base classifiers are independent, then ensemble makes a wrong prediction only if more than half of the base classifiers are wrong. For example, suppose there are two classes and each base classifier has an error rate of 35%, and we use 25 base classifiers, and then the error rate will be

$$\sum_{i=3}^{25} \binom{25}{i} \times 0.35^i \times 0.65^{25-i} = 0.06.$$

A logic view of the ensemble method is presented in Figure 2.1.

Ensemble classifiers have been used for classification in both centralized and distributed data as follows. For centralized massive data, base-level classifiers are generated by applying different learning algorithms with heterogeneous models [39], or a single learning algorithm to different versions of the given data. For manipulating the data set, three methods can be used: random sampling with replacement in bagging [8], re-weighting of the mis-classified training examples in boosting [21], and generating small bites of the data by importance sampling based on the quality of classifiers built so far [6]. There are many classic methods for centralized classification, such as bagging [8], boosting [20] and AdaBoost [21]. Random forest [7] is another advanced tool, which combines tree predictors, such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. Random inputs and random features produce good results in the classification. The detail will be described as follow:

- **Bagging**

  Probably the most well-known sampling approach is that exemplified by bagging [8].

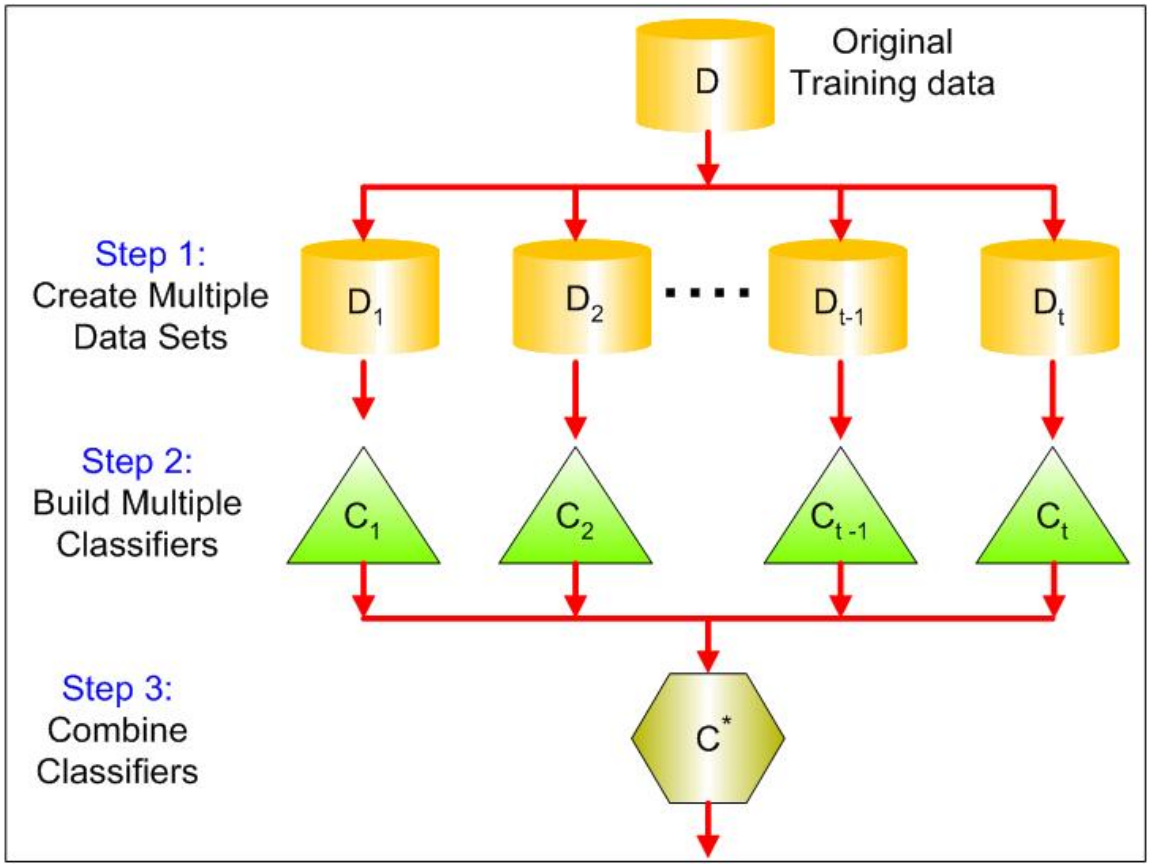

Figure 2.1: Ensemble Classifiers (extracted from [46])

Given a training set, bagging generates multiple bootstrapped training sets and calls the base model learning algorithm with each of them to yield a set of base models. Given a training set of size t, bootstrapping generates a new training set by repeatedly (t times) selecting one of the t examples at random, where all of them have equal probability of being selected. Some training examples may not be selected at all and others may be selected multiple times. A bagged ensemble classifies a new example by having each of its base models classify the example and returning the class that receives the maximum number of votes. The hope is that the base models generated from the different bootstrapped training sets disagree often enough that the ensemble performs better than the base models. Breiman [8] made the important observation that instability (responsiveness to changes in the training data) is a prerequisite for bagging to be effective. A committee of classifiers that all agree in all circumstances will give identical performance to any of its members in isolation. If there is too little data, the gains achieved via a bagged ensemble cannot compensate for the decrease in accuracy of individual models, each of which now sees an even smaller training set. On the other end, if the data set is extremely large and computation time is not an issue, even a single flexible classifier can be quite adequate.

- **Boosting**

  Another method that uses different subsets of training data with a single learning method is the boosting approach [20]. It assigns weights to the training instances, and these weight values are changed depending upon how well the associated training instance is learned by the classifier; the weights for misclassified instances are increased. Thus, re-sampling occurs based on how well the training samples are classified by the previous model. Since the training set for one model depends on the previous model, boosting requires sequential runs and thus is not readily adapted to a parallel environment. After several cycles, the prediction is performed by taking a weighted vote of the predictions of each classifier, with the weights being proportional to each classifiers accuracy on its training set.

- **AdaBoost**

  AdaBoost is a practical version of the boosting approach [21]. There are two ways that Adaboost can use these weights to construct a new training set to give to the

base learning algorithm. In boosting by sampling, examples are drawn with replacement with probability proportional to their weights. The second method, boosting by weighting, can be used with base learning algorithms that can accept a weighted training set directly. With such algorithms, the entire training set (with associated weights) is given to the base-learning algorithm.

- **Random Forests**
  Random forest is a class of ensemble methods specifically designed for decision tree classifiers [46]. It combines the predictions made by multiple decision trees, where each tree is generated based on the values of an independent set of random vectors. Each decision tree uses a random vector that is generated from some fixed probability distribution. A random vector can be incorporated into the tree-growing process in many ways. The random vectors are generated from a fixed probability distribution. Bagging using decision trees is a special case of random forests, where randomness is injected into the model-building process by randomly choosing N samples, with replacement, from the original training set. Bagging also uses the same uniform probability distribution to generate its bootstrapped samples throughout the entire model-building process. The "strength" of a set of classifiers refers to the average performance of the classifiers. As the trees become more correlated or the strength of the ensemble decreases, the generalization error bound tends to increase. Randomization helps to reduce the correlation among decision trees so that the generalization error of the ensemble can be improved.

For distributed data classification using the ensemble method, several techniques have been proposed [33, 14].

For naturally distributed data, several techniques have been proposed as follows. Lazarevic *et al.* [32] give a distributed version of boosting algorithm, which efficiently integrates local classifiers learned over distributed homogeneous databases. In the proposed method, the classifiers are first learned from disjoint datasets at each boosting round and then exchanged amongst the sites. The exchanged classifiers are then combined, and finally, their weighted voting ensemble is constructed on each disjoint data set. The ensemble that is applied to an unseen test set represents an ensemble of ensembles built locally on all distributed sites. The performance of ensembles is used to update the probabilities of drawing the data samples in succeeding boosting iterations.

Tsoumakas *et al.* [47] present a framework for constructing a global predictive model from local classifiers that do not require moving raw data around, achieves high predictive accuracy and scales up efficiently with respect to large numbers of distributed data sets. It can be broken down into the following phases: Local Learning and Classifier Combination. In local learning step, an available classification learning algorithm is used to train a local predictive model from all instances of that node. All models apart from the local one will be combined to form a global classifier.

Chawla *et al.* [15] present a distributed approach to pasting small bites, which uniformly votes hundreds or thousands of classifiers built on all distributed data sites. In this approach, the authors divide a data set into T disjoint subsets, and assign each disjoint subset to a different processor. On each of the disjoint partitions, they follow the Breiman's approach of pasting small votes. They randomly sample with replacement, as they can load the entire disjoint subset of data in a processor's memory. Thus, the number of disjoint partitions can be dictated by the amount of memory available on each processor. They combine the predictions of all the classifiers by majority vote. Again, using the above framework of memory requirement, if they break up the data set into T disjoint subsets, the memory requirement will decrease by a factor of $\frac{1}{T}$, which is substantial. One can essentially divide a data set into subsets easily managed by the computer's main memory.

Distributed classification in peer-to-peer networks is also studied [36]. The authors propose an ensemble approach, in which each peer builds its local classifier on the local data and then combines all the classifiers by plurality voting. These distributed solutions however are not customized for sensor networks. In particular, they do not consider the limit of memory size, nor the energy consumption, which are two critical concerns with respect to battery-powered sensor nodes. Directly applying these distributed classifications into our application scenario will thus result in poor performance, as demonstrated through our simulations.

2.5 Summary

In this chapter, we first presented some techniques of wireless sensor networks. We reviewed some classification algorithms and introduced a few ensemble methods. And also described some of the key issues for data gathering in wireless sensor networks.

Chapter 3

Hierarchical Distributed Classification

In this chapter, we first present the system overview, and then introduce our classification approach in detail, showing how to build the decision tree, how to generate the pseudo data, and how to build the classifier hierarchically. We discuss the accuracy and energy consumption afterwards.

3.1 System Overview

We consider N sensor nodes $n_1, n_2, \ldots, n_N$ distributed in a field. Each node covers an area of the field and is responsible for collecting data within the area. The data reporting follows a spanning tree rooted at the base station n_0. The routing protocol design for forming the spanning tree is out of the scope of this thesis, and there are indeed numerous solutions in the literature [19, 49].

Each sensor node n_i first collects its local training data D_i. If node n_i is a leaf node, it builds a classifier C_i by a learning algorithm $\aleph$, which we will illustrate in Section 3.3. The node then sends C_i to its parent node, say n_j. We use a decision tree T to represent the classifier[1], which, compared to the original data, is of a much smaller size. From this perspective, it can be viewed as a data compression scheme. Compared to conventional

[1]We will use "classifier" and "decision tree" interchangeably throughout this thesis provided the context is clear.

compression schemes for sensor data collection, however, the decision tree preserves finer details of the distribution information of the data, thus facilitating distributed classification, as will be elaborated later.

An upstream node n_j, upon receiving the classifiers from its children, combines the children's classifiers with its local training data D_j to build an enhanced classifier C_j. These local classifiers will be iteratively enhanced from bottom up and finally reach the base station, making a global classifier for all the data distributed across the sensors. Since only the classifiers will be forwarded upstream, the energy consumption for transmission can be significantly reduced. The sensor nodes may continuously sense new data and forward to upstream.

The challenges here lie in training the enhanced classifier from a mix of downstream classifiers and the local dataset, which cannot be directly accomplished by existing training algorithms that work on datasets only. To address this problem, a pseudo training dataset will be generated from each downstream classifier. That is, for each child node n_i, node n_j will generate a set of pseudo training data D'_i from the classifier C_i, and then combine all these data with its own training data to build the enhanced classifier. Obviously, the pseudo data, recovered from classifiers, should closely reflect the characteristics of the original local data, in particular, the distribution of different classes and the attribute values. The size of the pseudo data is also an important concern given that a sensor node generally has a limited memory. We will address these detailed issues in Section 3.4.

For ease exposition, we list the major notations in Table 3.1. Also note that, in this paper, we do not consider issues like node failure or packet loss, which have been extensively addressed in the literature [19, 27, 4].

Notation	**Explanation**
N	the number of sensor nodes, excluding the base station
n_i	sensor node ($i = 1, 2, ..., N$)
n_0	the base station
D_i	the training data collected by node n_i
D'_i	the pseudo data generated from classifier C_i
C_i	the local classifier built by node $n_i, i = 1, 2, ..., N$
C_0	the global classifier built at the base station

Table 3.1: List of notations

3.2 Decision Tree Overview

Decision tree [26] is one of the most important models for classification, and also serves as the foundation for our hierarchical distributed classification. A decision tree is a flow-chart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and leaf nodes represent classes or class distributions. The topmost node in a tree is the root node. The decision tree consists of nodes that form a rooted tree, meaning it is a directed tree with a node called the root that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called an internal or test node. All other nodes are called leaves (also known as terminal nodes or decision nodes). A decision tree is a mapping from observations about an item to conclusions about its class label. A node in the tree is a test of some attribute, and a branch is a possible value of the attribute. To perform classification, we can start from the root, test the attribute, and move down to a tree branch. Figure 3.1 shows a decision tree example. For instance, it indicates that if the temperature is medium and the sunlight is weak, the environment is classified as suitable (Yes) for animals.

In a decision tree, each internal node splits the instance space into two or more subspaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case each test considers a single attribute, such that the instance space is partitioned according to the attribute's value. In the case of numeric attributes the condition refers to a range. Each leaf is assigned to one class representing the most appropriate class label. Alternatively, the leaf may hold a probability vector indicating the probability of the certain class label. Instances are classified by navigating the tree from the root of the tree down to a leaf, according to the outcome of the tests along the path.

Naturally, decision-makers prefer less complex decision trees, since they may be considered more comprehensive. Furthermore, according to Breiman *et al.* [9] the tree complexity has a crucial effect on its accuracy performance. The tree complexity is explicitly controlled by the stopping criteria used and the pruning method employed. Usually, the tree complexity is measured by one of the following metrics:

- the total number of nodes;
- total number of leaves;
- tree depth;

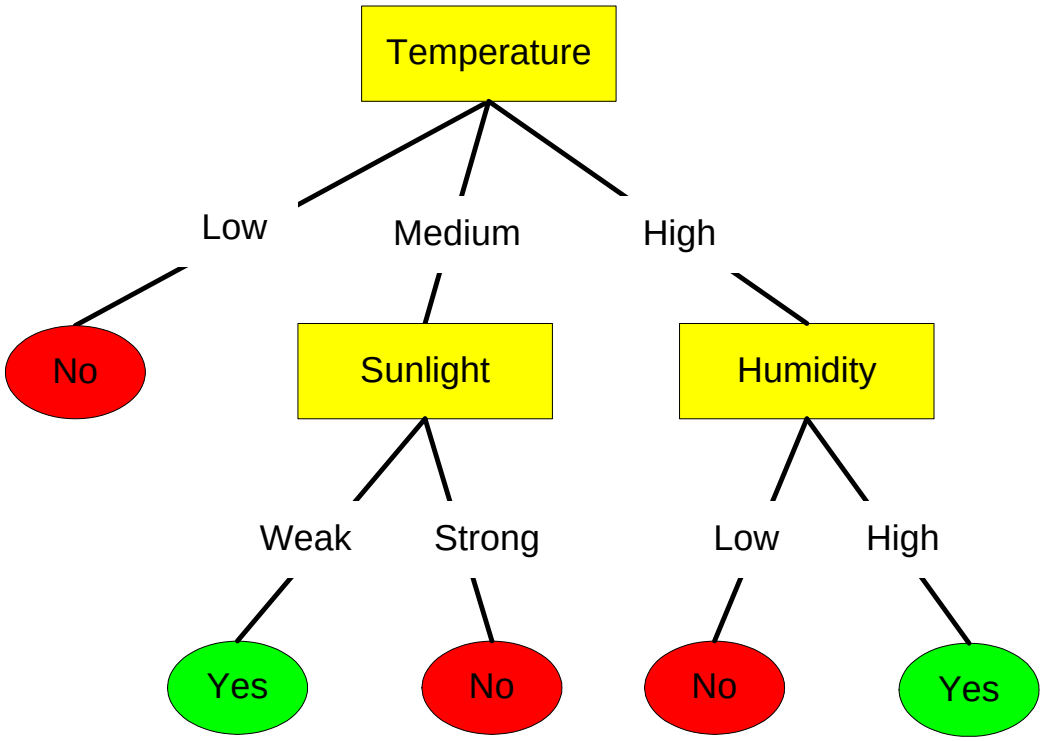

Figure 3.1: An example of a decision tree

- number of attributes used.

Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part, and taking the leaf's class prediction as the class value [44].

3.2.1 Splitting Function

In most of the cases the discrete splitting functions are univariate. Univariate means that an internal node is split according to the value of a single attribute. Consequently, the algorithm searches for the best attribute upon which to split. There are various univariate criteria. These criteria can be characterized in different ways, such as according to the origin of the measure: information theory, dependence, and distance. The following sections describe the most common criteria in the literature.

- Information Gain

  Information gain uses the entropy measure as the split function [26]. To choose the best attribute at the current node, the measure calculates the information gain using entropy. The entropy can be shown as follows

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i, \tag{3.1}$$

  where S is the dataset, c is the number of classes and p_i is the proportion of each class. The information gain is then calculated as

$$Gain(S, A) = Entropy(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Entropy(S_v), \tag{3.2}$$

  where $V(A)$ is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

- Gini Index

  The Gini index has been used in various works that measures the divergence between the probability distributions of the target attribute's values [46]. The Gini index is defined as

$$Gini(S) = 1 - \sum_{i=1}^{c} p_i^2, \tag{3.3}$$

  where S is the dataset, c is the number of classes and p_i is the proportion of each class. Consequently, the evaluation criteria for selecting the attribute A is defined as

$$GiniGain(S, A) = Gini(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} Gini(S_v), \tag{3.4}$$

  where $V(A)$ is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v.

- Twoing

  Breiman *et al.* [9] point out that the Gini index may encounter problems when the domain of the target attribute is relatively wide. In this case they suggest using binary criterion called twoing criterion. Twoing groups classes into two superclasses so that the classification problem considered as a two-class problem, the greatest decrease in node impurity is realized. This criterion is defined as

$$twoing(S) = \frac{1}{4} \frac{|S_L|}{|S|} \frac{|S_R|}{|S|} \sum_{v \in V(A)} \left| \frac{|S_{L_v}|}{|S_L|} - \frac{|S_{R_v}|}{|S_R|} \right|^2, \tag{3.5}$$

where S is the dataset, S_L is the dataset in the left subtree and S_R is the dataset in the right subtree. $V(A)$ is the set of all possible values for attribute A, S_{L_v} is the subset of S_L for which attribute A has value v, and S_{R_v} is the subset of S_R for which attribute A has value v. When the target attribute is binary the Gini and twoing criteria are equivalent. For multiclass problems, the twoing criteria prefers attributes with evenly divided splits.

- Gain Ratio
  Quinlan [43] proposeed the gain ratio measure that normalizes the information gain as follows:
  $$GainRatio(S, A) = \frac{InformationGain(S, A)}{Entropy(S)}. \tag{3.6}$$
  Note that this ratio is not defined when the denominator is zero. Also the ratio may tend to favor attributes for which the denominator is very small. Consequently, there are two stages to deal with this situation. First, the information gain is calculated for all attributes. Then, taking into consideration only attributes that have performed at least as good as the average information gain, the attribute that has obtained the best ratio gain is selected. Quinlan [42] showed that the gain ratio tends to outperform simple information gain criteria both from the accuracy aspect as well as from the classifier complexity aspect.

- Likelihood Ratio Chi-Squared Statistics
  The likelihood ratio is defined as [3]:
  $$G^2(S, A) = 2 \ln 2 |S| InformationGain(S, A). \tag{3.7}$$
  This ratio is useful for measuring the statistical significance of the information gain criterion.

3.2.2 Pruning Techniques

Employing tight stopping criteria tends to create small and under-fitted decision trees. On the other hand, using loose stopping criteria tends to generate large decision trees that are over-fitted to the training set. The pruning methods originally suggested by Breiman *et al.* [9] were developed for solving this dilemma. According to this methodology a loosely stopping criterion is used, letting the decision tree to overfit the training set. Then the overfitted

tree is cut back into a smaller tree by removing sub branches that are not contributing to the generalization accuracy. It has been shown in various studies that employing pruning methods can improve the general performance of a decision tree especially in noisy domains.

Another key motivation of pruning is "trading accuracy for simplicity" as presented by Bratko and Bohanec [5]. When the goal is to produce a sufficiently accurate compact concept description, pruning is highly useful. Within this process the initial decision tree is seen as a completely accurate one. Thus, the accuracy of a pruned decision tree indicates how close it is to the initial tree. There are various techniques for pruning decision trees. Most of them perform top down or bottom up traversal of the nodes. A node is pruned if this operation improves a certain criterion.

- Cost-Complexity Pruning

  Breiman *et al.*s pruning method [9], cost complexity pruning (also known as weakest link pruning or error complexity pruning) proceeds in two stages. In the first stage, a sequence of increasingly smaller trees $T_0, T_1, \ldots, T_k$ are built on the training data where T_0 is the original tree before pruning and T_k has only the root node of tree T_0. In the second stage, one of these trees is chosen as the pruned tree, based on its classification accuracy on a *pruning set*. A pruning set is a portion of the training data that is set aside exclusively for pruning alone, based on its generalization error estimation. Let α be the cost complexity per leaf, the cost complexity risk of a tree T with root t is defined as:

  $$R_\alpha(\{t\}) = R(t) + \alpha \tag{3.8}$$

  and

  $$R_\alpha(T_t) = R(T_t) + \alpha|leaf(T_t)|, \tag{3.9}$$

  where $R(T)$ is the resubstitution risk estimate of T, T_t is the subtree of T which rooted at node t and $|leaf(T)|$ is the number of leaf nodes of tree T. The critical value of each node is

  $$g(t) = \frac{R(t) - R(T_t)}{(|leaf(T_t| - 1)}. \tag{3.10}$$

  Each node has a critical value and the subtree rooted at t to $\{t\}$ will be pruned when $\alpha \geq g(t)$.

  In the second phase, the generalization error of each pruned tree $T_0, T_1, \ldots, T_k$ is estimated. The best pruned tree is then selected. If the given dataset is large enough,

the authors suggested to break it into training set and pruning set. The trees are constructed using the training set and evaluated on the pruning set. On the other hand, if the given dataset is not large enough, they proposed to use the cross-validation methodology, despite the computational complexity implications.

- Pessimistic Error Pruning (PEP)
  Quinlan's pessimistic pruning [43] avoids the need of pruning set or cross validation, and uses the pessimistic statistical correlation test instead. The basic idea is that the error ratio estimated using the training set is not reliable enough. Instead, a more realistic measure known as continuity correction for binomial distribution should be used
  $$\varepsilon'(T,S) = \varepsilon(T,S) + \frac{|leaves(T)|}{2|S|}, \tag{3.11}$$
  where $\varepsilon(T,S)$ indicates the error rate of the tree T over the sample S and $|leaves(T)|$ denotes the number of leaves in T. However, this correction still produces an optimistic-error rate. Consequently, Quinlan suggested pruning an internal node if its error rate is within one standard error from a reference tree, namely
  $$\varepsilon'(pruned(T,t),S) \leq \varepsilon'(T,S) + \sqrt{\frac{\varepsilon'(T,S)(1-\varepsilon'(T,S))}{|S|}} \tag{3.12}$$
  The last condition is based on statistical confidence interval for proportions. Usually, the last condition is used such that T refers to a sub-tree whose root is the internal node t and S denote the portion of the training set that refer to the node t. The pessimistic pruning procedure performs top-down traversing over the internal nodes. If an internal node is pruned then all its descendants are removed from the pruning process, resulting in a relatively fast pruning.

- Error-based Pruning
  Error-based pruning is an improvement of the pessimistic pruning. It is implemented in the well-known C4.5 algorithm. As in pessimistic pruning the error rate is estimated using the upper bound of the statistical confidence interval for proportions
  $$\varepsilon_{EBP}(pruned(T,t),S) = \varepsilon(T,S) + Z_\alpha \sqrt{\frac{\varepsilon(T,S)(1-\varepsilon(T,S))}{|S|}}, \tag{3.13}$$
  where $\varepsilon(T,S)$ denotes the misclassification rate of the tree T on the training set S. Z is the inverse of the standard normal cumulative distribution and is the desired

significance level. Let $subtree(T, t)$ denote the subtree rooted by the node t. Let $maxchild(T, t)$ denote the most frequent child node of t (that is, most of the instances in S reach this particular child) and let S_t denote all instances in S that reach the node t. The procedure performs bottom-up traversal over all nodes and compares the following values:

$$\varepsilon_{EBP}(subtree(T, t), S_t) \tag{3.14}$$

$$\varepsilon_{EBP}(pruned(subtree(T, t), t), S_t) \tag{3.15}$$

$$\varepsilon_{EBP}(subtree(T, maxchild(T, t)), S_{maxchild(T,t)}). \tag{3.16}$$

According to the lowest value the procedure either leaves the tree as is, prunes the node t, or replaces the node t with the subtree rooted by $maxchild(T, t)$.

3.2.3 Decision Trees Algorithms

- ID3

  Quinlan [40] proposed the ID3 algorithm. It is considered as a very simple decision tree algorithm. ID3 uses information gain as the splitting criteria. The growth stops when all instances belong to a single value of the target feature or when the best information gain is not greater than zero. ID3 does not apply any pruning procedures. Nor does it handle numeric attributes neither missing values. The algorithm then creates a node for each possible attribute value, and partitions the training data into descendant nodes. There are three conditions to stop the recursion: (1) all samples at a given node belong to the same class, (2) no attribute remains for further partitioning, and (3) there is no sample at the node.

- C4.5

  C4.5 is an evolution of ID3, presented by the same author [43]. It uses gain ratio as the splitting criteria. The splitting ceases when the number of instances to be splitted is below a certain threshold. Error-based pruning is performed after the growing phase. C4.5 is capable of handle numeric attributes. It can induce from a training set that incorporates missing values by using corrected gain ratio criteria.

- CART

CART stands for classification and regression trees. It was developed by Breiman *et al.* [9] and is characterized by the fact that it constructs binary trees, that is each internal node has exactly two outgoing edges. The spits are selected using the twoing criteria and the obtained tree is pruned by cost-complexity pruning. When provided CART can consider misclassification costs in the tree induction. It also enables users to provide prior probability distribution.

An important feature of CART is its ability to generate regression trees. Regression trees are trees where their leaves predict a real number and not a class. In the case of regression CART looks for splits that minimize the prediction squared error (the least-squared deviation). The prediction in each leaf is determined based on the weighted mean for the node.

- CHAID

  The acronym CHAID stands for Chi-squared Automatic Interaction Detector. It is A tree classification methods originally proposed by Kass [28]. For each input attribute a_i, CHAID finds the pair of values that is least significantly different with respect to the target attribute. The significant difference is measured by the p value obtained from a statistical test. The statistical test used depends on the type of target attribute. If the target attribute is continuous, an F test is used, if it is nominal, then a Pearson chi-squared test is used, if it is ordinal, then a likelihood-ratio test is used.

  For each selected pair CHAID checks if the p value obtained is greater than a certain merge threshold. If the answer is positive it merges the values and searches for an additional potential pair to be merged. The process is repeated until no significant pairs are found.

  It then selects the best input attribute to be used for splitting the current node, such that each child node is made of a group of homogeneous values of the selected attribute. Note that no splitis performed if the adjusted p value of the best input attribute is not less than certain split threshold. This procedure stops also when one of the following conditions is fulfilled. Maximum tree depth is reached. Minimum number of cases in node for being a parent is reached, so it can not be split any further. Minimum number of cases in node for being a child node is reached. CHAID handles missing values by treating them all as a single valid category. CHAID does not perform pruning [44].

3.2.4 Decision Trees for Large Datasets

With the recent growth in the amount of data collected by information systems there is a need for decision trees that can handle large datasets [44].

Catlett [12] examined two methods for efficiently growing decision trees from a large database by reducing the computation complexity required for induction. However, the Catlett method requires that all data will be loaded into the main memory before induction. Namely, the largest dataset that can be induced is bounded by the memory size.

Chan and Stolfo [13] suggested partitioning the datasets into several disjoin datasets, such that each dataset is loaded separately into the memory and used to induce a decision tree. The decision trees are then combined to create a single classifier. However, the experimental results indicate that partition may reduce the classification performance, meaning that the classification accuracy of the combined decision trees is not as good as the accuracy of a single decision tree induced from the entire dataset.

Mehta *et al.* [38] have proposed SLIQ an algorithm that does not require loading the entire dataset into the main memory. Instead it uses secondary memory (disk) A certain instance is not necessarily resident in main memory all the time. SLIQ creates a single decision tree from the entire dataset. However, this method also has upper limit for the largest dataset that can be processed because it uses a data structure that scales with the dataset size and this data structure is required to be resident in main memory all the time.

Shafer *et al.* [45] presented a similar solution called SPRINT. This algorithm induces decision trees relatively quickly and removes all of the memory restrictions from decision tree induction. SPRINT scales any impurity based split criteria for large datasets.

Gehrke et al. [23] introduced RainForest; a unifying framework for decision tree classifiers that are capable of scaling any specific algorithms from the literature (including C4.5, CART, and CHAID). In addition to its generality, RainForest improves SPRINT on a factor of three. In contrast to SPRINT, however, RainForest requires a certain minimum amount of main memory, proportional to the set of distinct values in a column of the input relation. However, this requirement is considered modest and reasonable.

3.3 Building Decision Trees

In our system, the decision trees are built by the widely-used ID3 algorithm [41]. The basic ID3 algorithm, however, does not keep the information about the attribute distribution and

the amount of the original training data, which makes generating pseudo data quite difficult from a classifier.

To solve this problem, we let each leaf node also record the counts of the classes (i.e., the number of positive and negative labels in this application). Therefore, we can have the knowledge about the amount of the samples for building each branch of the decision tree, which further enable the distribution of the generated pseudo data close to the original.

Moreover, in the basic ID3 algorithm, if all the samples belong to the same class, the recursion stops (e.g., when temperature is low in Figure 3.1). Hence, the information of other attributes will be missing, which can cause problems with heterogeneous data distribution across different sensor nodes.

For example, if all the training data sensed by a sensor node are below 10 degree in temperature and below 20% in humidity, and the class labels are all negative. Using the basic ID3 algorithm, only one attribute, say temperature, will appear in the decision tree, and the information of humidity will be completely missing. This will likely lead to a pseudo data generated with humidity uniformly distributed from 0% to 99%, which is clearly not the case for the original data.

Therefore, for the stop condition of the recursion, we eliminate the first one (referred to Section 3.2.3) in our enhanced ID3 algorithm. The new condition thus becomes when no attribute remains for further partitioning or when there is no sample.

In Algorithm 1 and Algorithm 2 below, we describe the enhanced ID3 algorithm for building the decision tree. Note that a brief illustration of the key steps of the basic ID3 is in Section 3.2, and more details can be found in Quinlan's work [41].

Algorithm 1 LearningAlgorithm (D)

Require: training dataset D
call EnhancedID3 ($root$, D, 0)
Return root

3.4 Generating Pseudo data

The pseudo data generation is one of the most important steps in our framework. A critical challenge here is to generate data that is as close to the original data as possible. In particular, the distribution of each attribute should closely resemble that of the original

Algorithm 2 EnhancedID3 (p, D, d)

Require: pointer to the decision tree p, training dataset D, depth of the decision tree node d
  if number of data tuples in $D = 0$ **then**
    Return
  end if
  if d = number of attributes of training data **then**
    get the most common class label in $D \rightarrow$ attribute of p
    record the counts of the class labels for p
    Return
  end if
  get the best attribute $\rightarrow$ *target_attribute*
  for all *target_attribute* value branches **do**
    add a child p'
    set the value range of the branch
    partition $D \rightarrow D'$ satisfying the value range
    call EnhancedID3 (p', D', $d + 1$)
  end for

data.

Another issue is how many pseudo data tuples should be generated. Intuitively, the less data we generate from the child nodes, the less weight they have. Since no data from one node is considered less important than those from another, we should generate the same amount of pseudo data as the original data. Therefore, a sensor node close to the base station has to generate a huge amount of pseudo data, i.e., the same amount of the original data at all its descendants (not only the immediate children). This is often impossible given the limited memory of embedded sensor nodes. To this end, we introduce a *preservation factor*, ranging from 0 to 1 (the base station always has a factor of 1), to control the amount of the generated pseudo data.

Algorithm 3 summarizes our method to generate the pseudo data. For illustration, suppose the decision tree's leaf node represents a rule of when temperature is between 10 and 20, humidity is between 20 and 40, sunlight is *normal*, and there are 5 positive class labels and 45 negative ones. As such, the class label is negative. Assuming the preservation factor is set to 0.8, we then randomly generate $(5 + 45) \times 0.8 = 40$ data tuples that satisfy the attribute requirement. It follows that each data tuples has a probability $5/50 = 10\%$ to be assigned a class label as positive and 90% to be negative.

Algorithm 3 GeneratePseudoData (C, a)

Require: decision tree received from one child C, preservation factor, a
  for all leaf nodes *node* of decision tree C **do**
    get rule R of *node*
    get class label counts $\rightarrow c_1, c_2, \ldots, c_L$
    randomly generate $a \cdot \sum_{i=1}^{L} c_i$ data satisfying R
    assign class label $l_k (k = 1, \ldots, L)$ to the data with probability as the proportion of the class label $l_k / \sum_{i=1}^{L} c_i$
    add these data to pseudo data set D'
  end for
  RETURN pseudo data set D'

The original data is partitioned to each decision tree leaf node, and each set of generated data resembles part of the original data. Therefore, the combined pseudo data will largely reflect the characteristics the original training data. We will closely examine the effectiveness of our method and also the impact of the preservation factor in Section 4.

3.5 Hierarchical Classification

As mentioned above, we let the sensor nodes in the network be organized by a spanning tree, as illustrated in Figure 3.2. A leaf node builds the decision tree with the local sensed training data and sends the decision tree to the parent. The intermediate node periodically checks if there is any new classifier from children. If yes, it will generate a set of pseudo data for each new classifier, and combines them with its local data. There are two situations here. (1) If the node has never built any classifier, which indicates that it has never received any classifier from its children, it will combine the generated pseudo data (from the classifier it just received) with its local sensed data and performs the learning algorithm. (2) If the node has once built a classifier, the previous received classifiers may have already been discarded (due to the memory constraint). The node will then generate a set of pseudo data for its local classifier with the preservation factor being 1, and combines it with the other pseudo datasets to build the new decision tree.

The base station will build the global classifier. Initially, it waits until receiving all the classifiers from its children, and then generates pseudo data for each and combines them to build the decision tree. Since the base station in general is a more powerful node, it can store all the classifiers from its immediate children. Therefore, whenever one of them

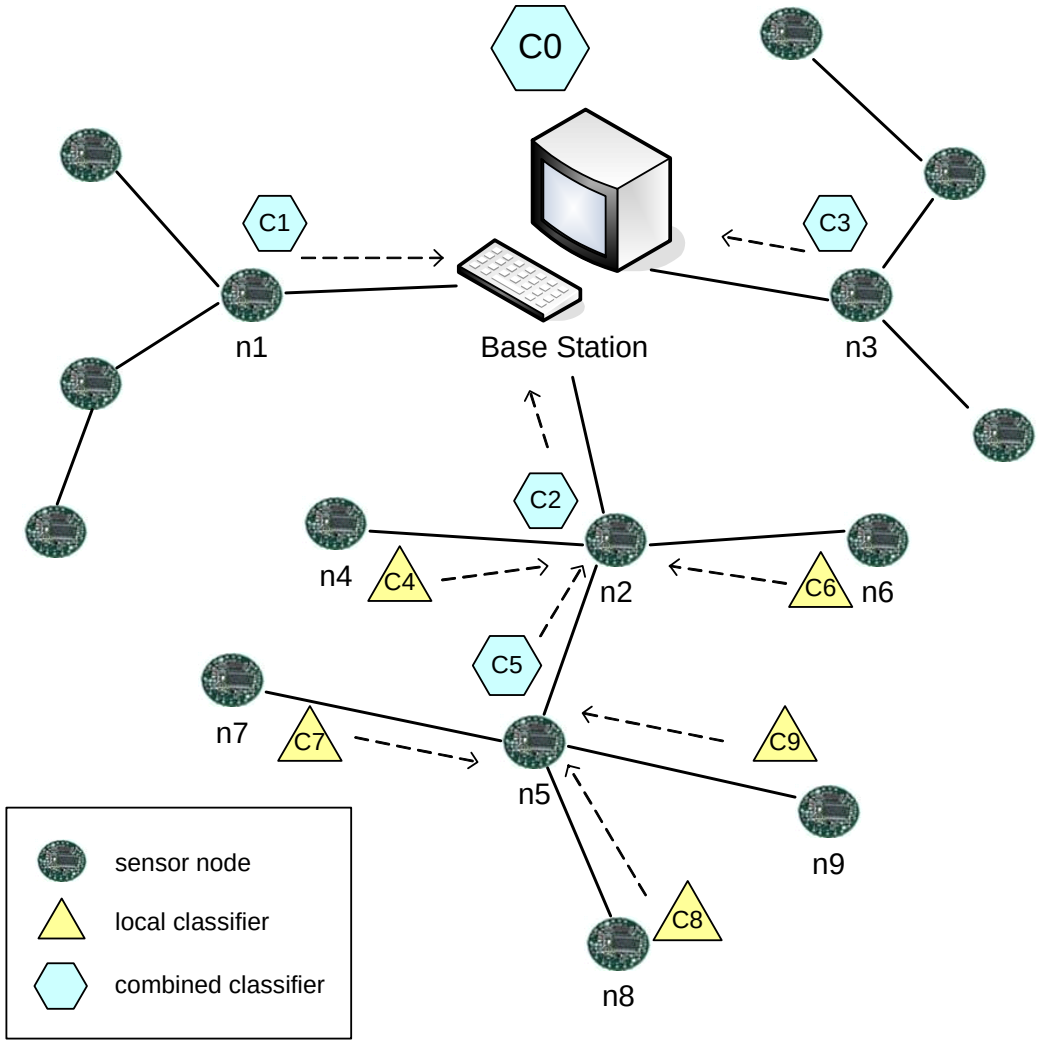

Figure 3.2: A hierarchical structure of the sensor network

updates the classifier, it can discard the old one, and re-performs the operations as above.

In Algorithm 4, we summarize the detailed procedure of the hierarchical classification.

Algorithm 4 HierarchicalClassification ()

```
if sensor node is leaf then
  periodically collect data D
  C ← LearningAlgorithm (D)
  send C to its parent
else if sensor node is not base station then
  periodically collect data D
  for all new classifiers C_k from child k do
    D'_k ← GeneratePseudoData (C_k, a)
  end for
  C ← LearningAlgorithm (D ∪ (⋃ D'_k))
  send C to its parent
else {//base station}
  if no classifier has been built then
    for all classifiers C_j from child j do
      D'_k ← GeneratePseudoData (C_k, 1)
    end for
    C ← LearningAlgorithm (⋃ D'_k)
  else if receive new classifier C_k from child k then
    replace old classifier of child k with C_k
    for all classifiers C_k from child k do
      D'_k ← GeneratePseudoData (C_k, 1)
    end for
    T ← LearningAlgorithm (⋃ D'_k)
  end if
end if
```

3.6 Further Discussion

Our approach utilizes the ID3 as the basis for classification, so it inherits the effectiveness and efficiency of ID3 when building local classifiers [41]. To make it fit our application scenario better, we suggest that all leaf nodes in the decision tree have the same depth as the number of attributes, so that we can keep all the attribute information when abstracting the rule from each branch. Thanks to the modification, we can generate the pseudo data that is very similar to the original data. Our decision tree can also be presented as a kind of compression method. But compared to the normal compression method, our approach keep

all the information of multi-dimension and the detail for classification. Apparently, this modification will increase the size of the decision tree, however, such increase is acceptable (roughly 1% in our system) and it noticeably increases the classification accuracy, as will be validated in our performance evaluation.

In order to keep the high accuracy and achieve our goals of saving energy and storage space, we have introduced two parameters in our solution. One is the *class count*. It not only records the count but also indicates the distribution of all the class labels in the original dataset when we build the decision tree. In other words, we keep the "noise" information because the "noise" may be important in some cases, in particular, the heterogeneous data distribution. For example, suppose one decision tree branch has the negative label because the numbers of positive and negative training data satisfying the constraint are 1 and 9, while another decision tree branch has the positive label on the same attribute constraint because the counts are 99 and 1. Respectively without recording the class label counts, we have no idea about which one is more accurate, and we will likely treat them equally. In fact, combining the two training datasets, we will obtain the positive label with the class counts 100 versus 10. The problem is particularly severe when the training dataset has the heterogeneous distributions, which critically demand the recording of the class count.

The other parameter is the preservation factor, which determines the amount of pseudo data to generate. We introduce this parameter due to the limited memory of the sensor node. The smaller the preservation factor is, the more dominant the local training data is. For example, suppose a node has three children, each having 200 training data tuples, and the node itself also has 200 training data tuples. If the preservation factor is set to 1.0, the node will learn from 800 data tuples, in which 25% is its local data. If the preservation factor is set to 0.1, the node will learn from 260 data, in which 77% is its local data. Intuitively, if the pseudo data can represent the original data very well, the greater the preservation factor is, the more accurate the classifier is, because every area should be treated equally. Otherwise, the greater the preservation factor is, the more noise it will make, hence decreasing the accuracy. Therefore, the representativeness of the pseudo data of the original data is crucial in our solution. In the next section, we will closely examine its impact to the classification accuracy.

Chapter 4

Performance Evaluation

4.1 Configuration and Datasets

We consider a square field consisting of $m \times m$ randomly deployed sensor nodes, with the base station being located in the center. We set m from 5 to 20, and for each m value, we build different topologies so that the spanning tree has height of 2, 3 and 4, respectively. Figure 4.1 shows an example of 25 nodes ($m = 5$) with height 2. Our results however show that the performance of our algorithm is mainly affected by the height of the spanning tree, while not the node population. This is intuitive because the algorithm is distributed and localized. Hence, in this section, we will focus on the average results (and the associated standard deviations) over the tested m values.

The dataset has three attributes and a class label. The three attributes are temperature, humidity and sunlight. The first two are numerical attributes, ranging from 0 to 49 and from 0 to 99 respectively, and the last one is a categorical attribute, having values *weak*, *normal* and *strong*. For simplicity, we consider the two numerical attributes as categorical attributes, as the temperature has values $[0, 10)$, $[10, 20)$, $[20, 30)$, $[30, 40)$ and $[40, 50)$, and the humidity values $[0, 20)$, $[20, 40)$, $[40, 60)$, $[60, 80)$ and $[80, 100)$. The class label is either *positive* or *negative*, say, indicating whether the environment is suitable for the animal. We randomly generate data tuples having the temperature between 0 and 49, humidity between 0 and 99, sunlight being 0, 1 or 2. We manually define some rules to assign each data tuples a class label. We also consider noise and thus add a factor ε, which means a data tuples has the probability of ε to be the other class label determined by the rules.

We generate 10 training datasets, each having $200 \cdot (m^2 - 1)$ data tuples. Among the

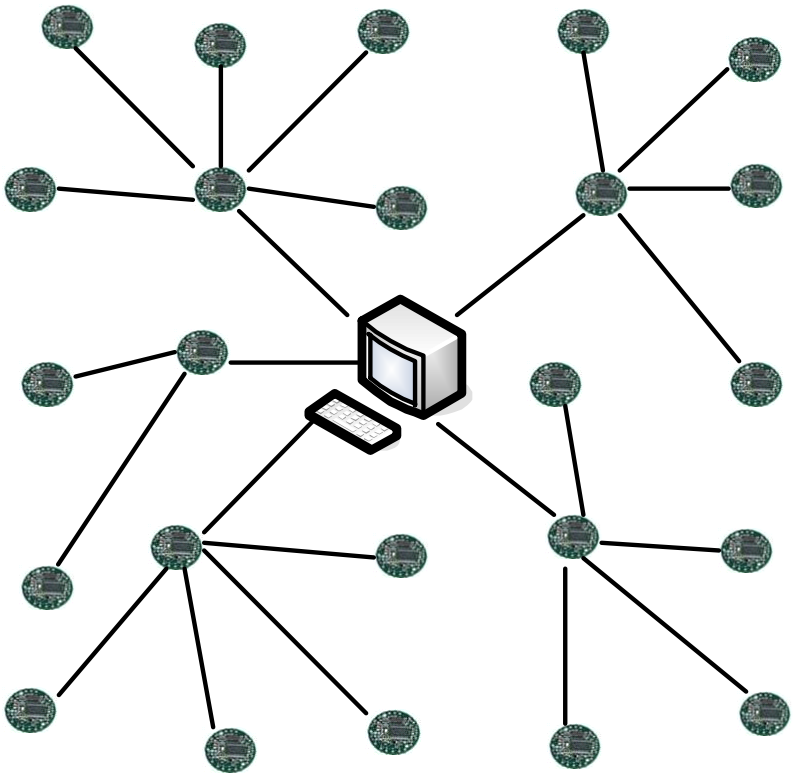

Figure 4.1: An example of topology of 25 nodes and height being 2

10 datasets, half of them have $\varepsilon = 1\%$ noise, and half of them have $\varepsilon = 10\%$ noise. For each dataset, we make it into two versions, one is called *heterogeneous data* and the other is called *homogeneous data*. In the heterogeneous data set, the data distribution depends on the location of the sensor nodes, while the homogeneous data is independent on the location, and is randomly and uniformly distributed across sensor nodes.

For homogeneous data, we just randomly divide and assign all the training data to the $(m^2 - 1)$ sensor nodes as the local training data, thus each sensor node has 200 training tuples. For heterogeneous data tuples, we assume that the temperature increases from left to right, and the humidity increases from bottom to top in the sensor field; the attribute of sunlight is uniformly distributed. For example, a node in the bottom right corner is supposed to have the data with temperature between 40 and 49, humidity between 0 and 19, and a node in the top left has the data with temperature between 0 and 9, humidity between 80 and 99. For each tuples in one dataset, we first calculate its coordinates according to the above, and then assign the tuples to the sensor node of that location if the training set of the node is not full (200 tuples). If that node is full, we then assign this tuple to another random node. For each training dataset, we perform different simulations that have different data distributions. For example, one is described as above, and another one is that temperature increases from bottom to top, humidity increases from right to left, and so forth.

In our experiments, we generate 10 test datasets, each having 1000 tuples. Among the 10 datasets, half of them have $\varepsilon = 1\%$ noise, and half of them have $\varepsilon = 10\%$ noise. If we use training tuples with $\varepsilon = 1\%$ noise to learn, we use test tuples with $\varepsilon = 1\%$ noise to test (same as tuples with $\varepsilon = 10\%$ noise).

4.2 Baseline for Comparison

We also implement an *ensemble method* [17] for the baseline comparison. The ensemble method is to construct a set of base classifiers and take a majority voting on predictions in classification. The method can significantly improve the accuracy of prediction, because if the base classifiers are independent, then the ensemble makes a wrong prediction only if more than half of the base classifiers are wrong. For example, suppose there are two classes and each base classifier has an error rate of 35%. With 25 base classifiers, the error rate will be $\sum_{i=3}^{25} \binom{25}{i} \times 0.35^i \times 0.65^{25-i} = 0.06$ only.

The ensemble method has been widely adopted in distributed classification [36]. In our

evaluation, we customize the ensemble method to our application scenario. Specifically, all the nodes learn from their local training data to build the classifiers, and send the classifiers to the base station through multi-hop routing. The base station then conducts the second step of the classification, in particular, the base station tests the unseen data with all the classifiers and takes a majority voting to get the final decision. Obviously, sending all the local classifiers to the base station consumes more energy than only sending the local classifier to the parents. Moreover, the ensemble method does not accommodate to heterogeneous data, thus the accuracy of classification can be low, as will be shown later.

4.3 Impact of Preservation Factor, Noise and Height for Heterogeneous Data

We first examine our algorithm with different preservation factors, noises and heights for the heterogeneous data. We plot the results in Figure 4.2. The x axis is the preservation factor ranging from 0.1 to 1.0, and we also evaluate the best-possible accuracy of learning from the entire dataset (assuming one sensor node collects all the data and builds the classifier), referred to as "A", and the accuracy of the ensemble method, referred to as "E".

From the figure, we find that the preservation factor does affect the accuracy, especially when it is small, the height is big, and the noise is large. When the noise is very small (1%), the preservation factor does not affect the accuracy when the factor is larger than 0.4, regardless of the height. When the noise becomes greater (10%), the preservation factor should be at least 0.8, so as not to decrease the accuracy. When the preservation factor is very small, the larger the height is, the less accuracy it achieves. We also find that when the preservation factor is large enough, the accuracy is almost the same as that of learning from the entire training data, i.e., the practically optimal accuracy.

The figure indicates that our mechanism to generate pseudo data works quite well, in particular, when the factor is large enough, the pseudo data can well represent the original data. When the preservation factor is small, the accuracy becomes relatively lower. The reason is that, from the whole system's view, the areas that are close to the base station generally become dominative, but as mentioned before, we should consider different areas equally. Moreover, when the height is high, the noise will be accumulated, and hence the accuracy will be reduced.

Comparing with the ensemble method, when the preservation factor is large enough, our

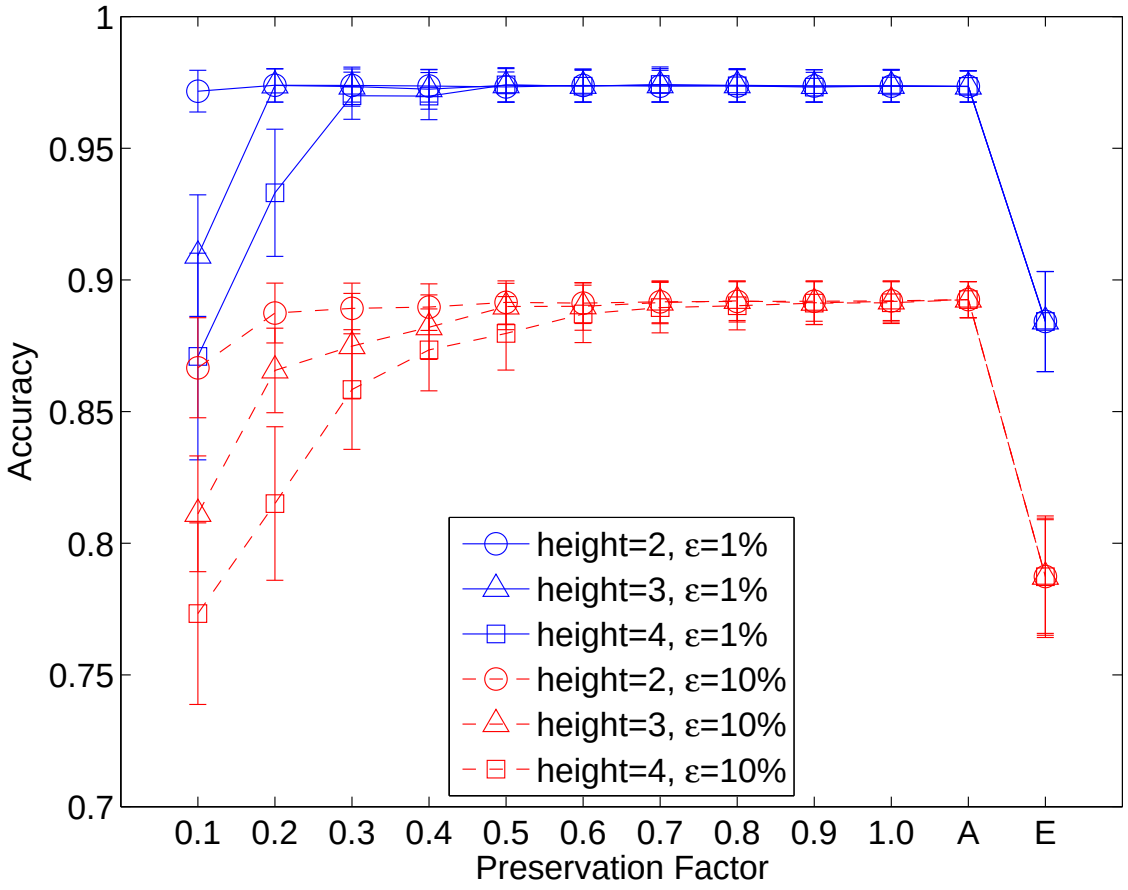

Figure 4.2: Comparison of accuracy for different preservation factor, noise and height with heterogeneous data

approach achieves a much higher accuracy, and when the noise is greater, the difference is even larger. This is because in the ensemble method, for heterogeneous data, each sensor node only learns a part of the data (e.g., low temperature and low humidity). In other words, in the ensemble method, only a few classifiers are responsible for a certain test data. If a given unlabeled data has the attribute value that is much different from its training data, it probably needs to randomly guess a class label, which will greatly decrease the accuracy.

4.4 Comparison of Enhanced and Basic ID3 Algorithm

We next compare our enhanced ID3 algorithm with the basic ID3 algorithm. We clarify that we modify the basic ID3 because it does not suitable for generating pseudo data in our approach for the heterogeneous data distribution. We again use the heterogeneous data with different noises, and plot the results in Figure 4.3 and Figure 4.4, respectively.

We find that the enhanced ID3 algorithm is not noticeably affected by the preservation factor (when the factor is large enough), nor the height. On the other hand, the accuracy of basic ID3 algorithm is much lower than ours, and is particularly lower when the preservation factor is smaller and the height is larger.

The difference is because when all the samples belong to the same class label, the basic algorithm stops the recursion, while our enhanced version continues. As mentioned before, in heterogeneous data distribution, it is probably that all the data from one node is below 10 in temperature and below 20 in humidity, and all the labels are negative. If we utilize the basic ID3, the decision tree is likely to contain only one attribute, say temperature, with the information of humidity being completely missing. As such, when generating the pseudo data, the humidity has to be uniformly distributed from 0 to 99, adding remarkable noises.

To validate this, we also examine their performance with the homogeneous data, and the results show that the two algorithms perform almost the same as shown in Figure 4.5 and Figure 4.6.

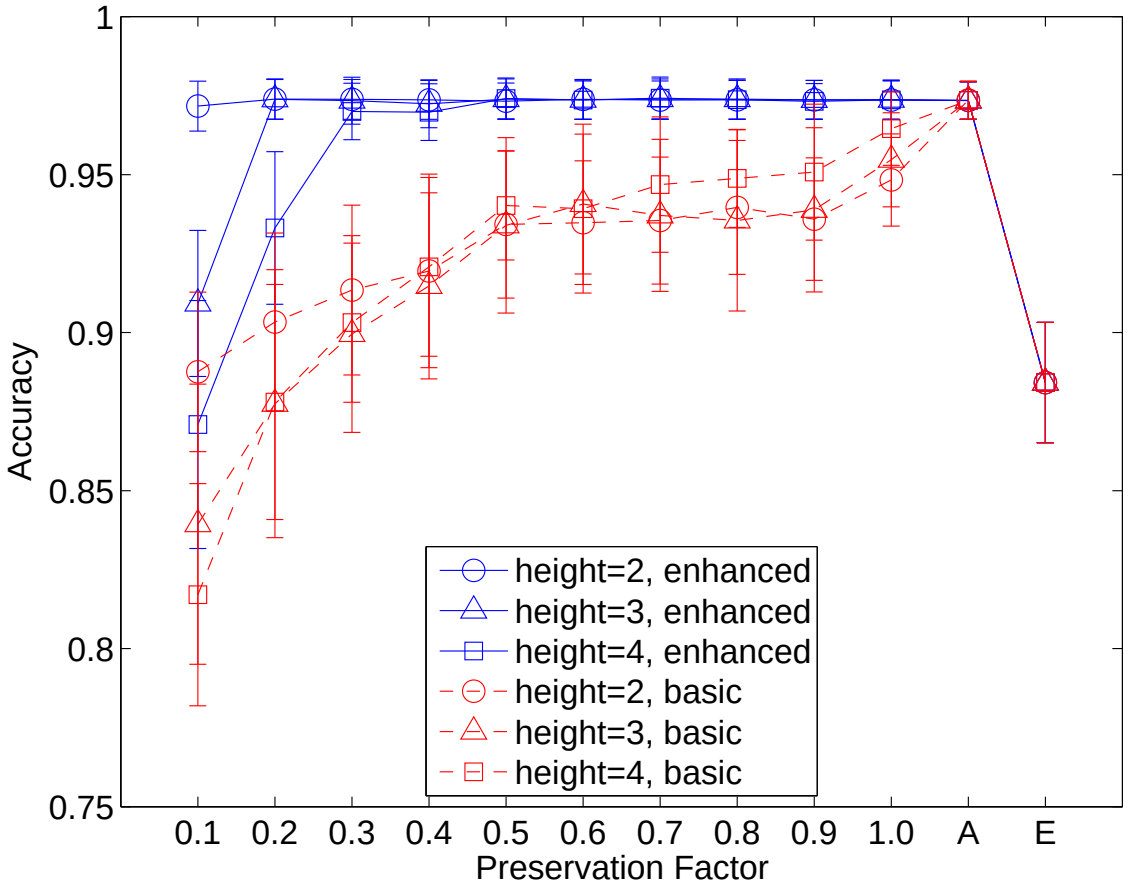

Figure 4.3: Comparison of enhanced and basic ID3 algorithms for heterogeneous data with $\varepsilon = 1\%$

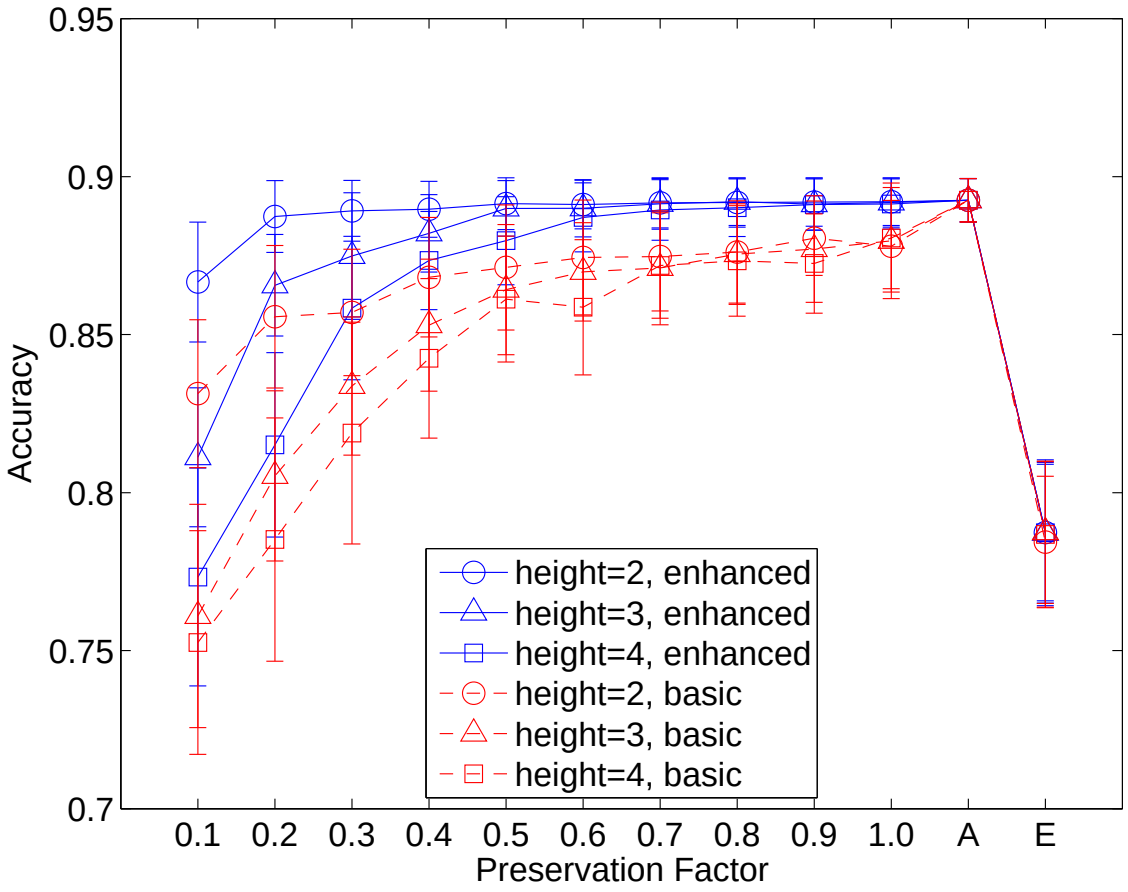

Figure 4.4: Comparison of enhanced and basic ID3 algorithms for heterogeneous data with $\varepsilon = 10\%$

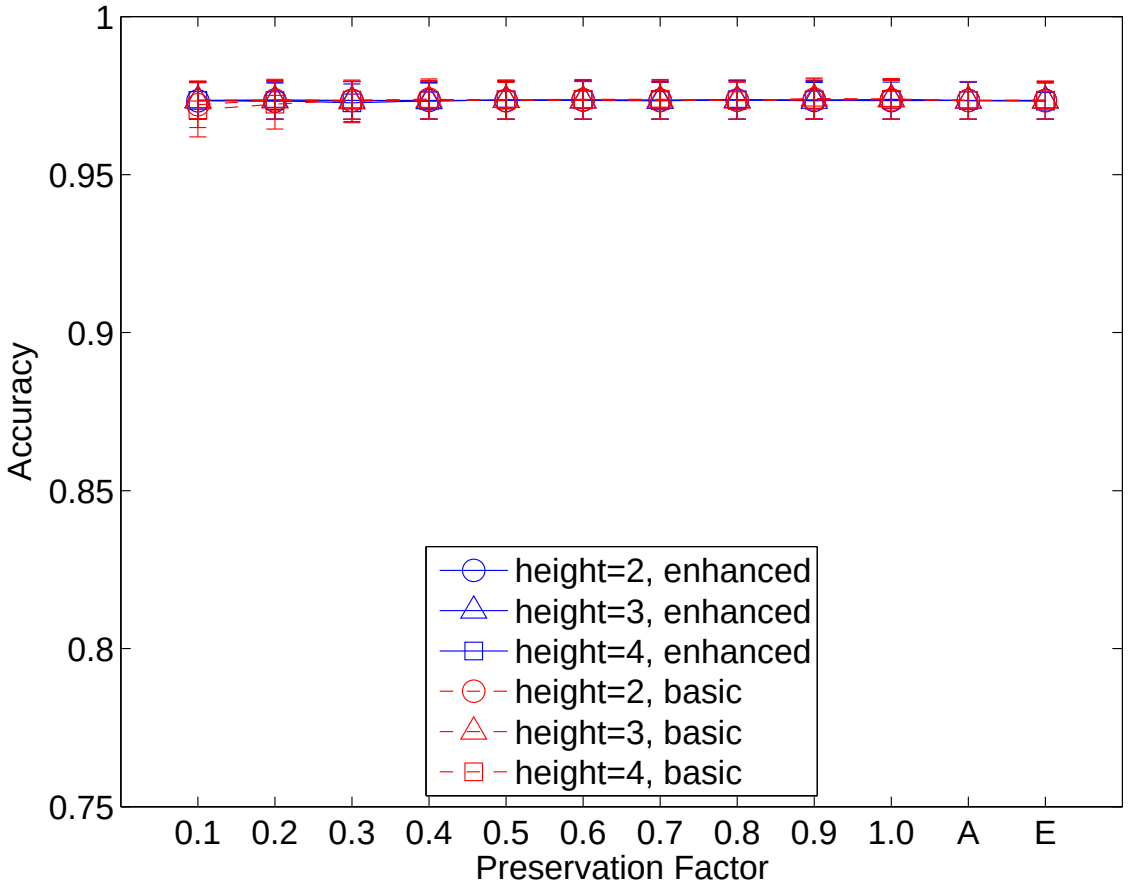

Figure 4.5: Comparison of enhanced and basic ID3 algorithms for homogeneous data with $\varepsilon = 1\%$

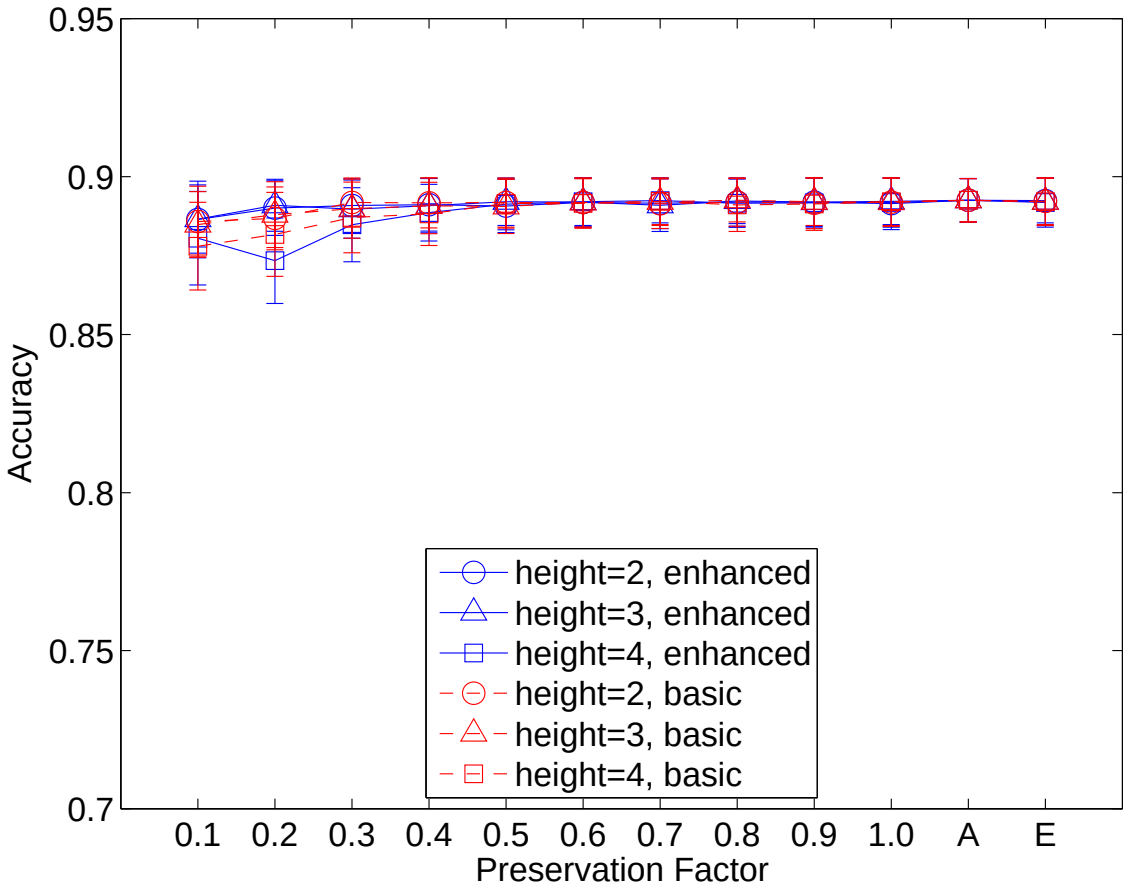

Figure 4.6: Comparison of enhanced and basic ID3 algorithms for homogeneous data with $\varepsilon = 10\%$

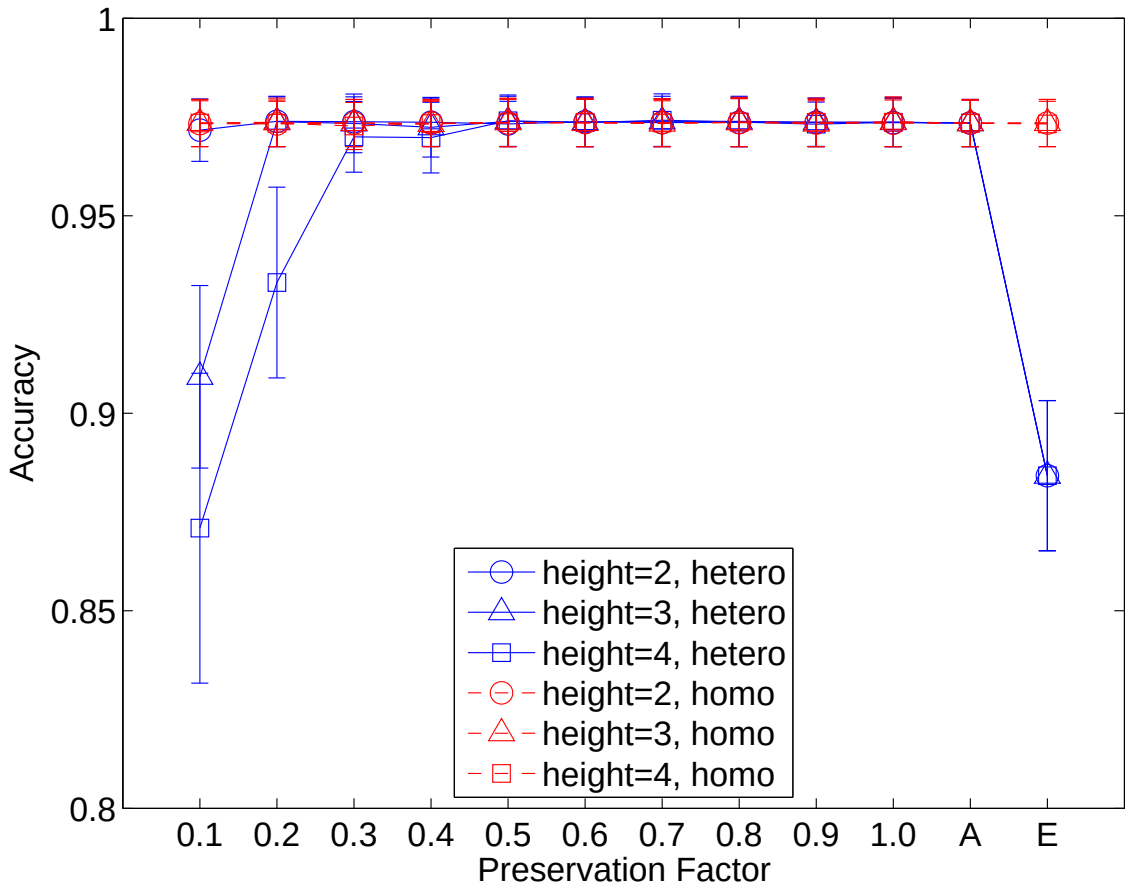

Figure 4.7: Comparison of heterogeneous and homogeneous data distribution with $\varepsilon = 1\%$

4.5 Impact of Training Data Distribution

To further understand the impact of the distribution of the data sets, we perform comparative experiments with both the heterogeneous data and the homogeneous data. Figure 4.7 and Figure 4.8 plot their respective accuracy results with different noises.

We find that for the heterogeneous data, when the preservation factor is small, the accuracy is low. On the other hand, for the homogeneous data, the factor does not affect the accuracy. For data with small noise, when the preservation factor is greater than 0.4, the two achieve the same accuracy. If the noise is larger, the preservation factor has to go beyond 0.8 to achieve the same accuracy. This is because in homogeneous data distribution, all the nodes have similar training data, thus the classifiers built by the nodes are almost

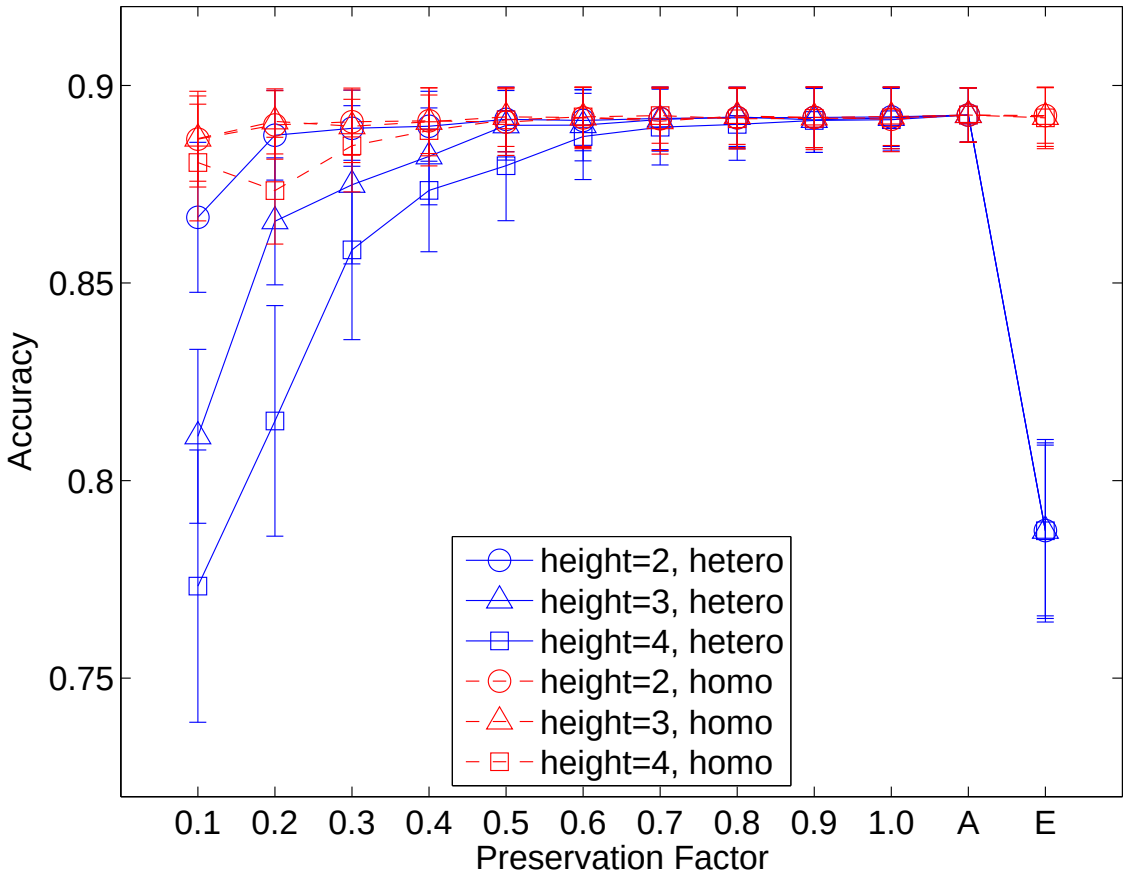

Figure 4.8: Comparison of heterogeneous and homogeneous data distribution with $\varepsilon = 10\%$

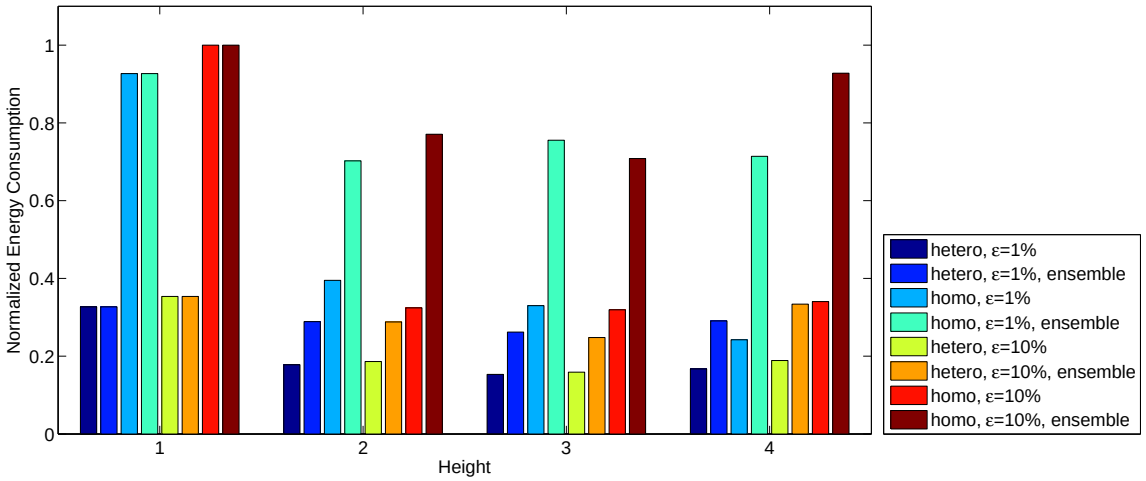

Figure 4.9: Comparison of energy consumption

the same, which leads to the high similarity between the generated pseudo data and the local training data. Therefore, the accuracy is independent on the preservation factor for homogeneous data.

In the ensemble method, the accuracy for heterogeneous data is much lower than that of homogeneous data and our hierarchical approach. This has been explained in the first experiment. For the homogeneous data, the ensemble method has the same accuracy as our approach.

4.6 Comparison of Energy Consumption

Finally, we investigate the energy consumption, which is one of the most important concerns in wireless sensor networks. We compare our hierarchical classification with the ensemble method that transmits all the classifiers to the base station. The energy consumption depends on the size of transmitted data and the distance between the two nodes. Generally, the energy consumption is proportional to the data size and the square of the distance [27]. We normalize the result and plot in Figure 4.9.

From the figure, we find that the energy consumption of our approach is much lower than the ensemble method in all the situations when the height is greater than 1. On average, our method saves nearly half of the energy spent in the ensemble method in all situations. The greater the height is, the more energy we save, simply because a classifier is only forwarded

to the parent in our solution, while it is forwarded all the way to the base station in the ensemble method.

We also find that the energy consumption for the heterogeneous data is lower than that for homogeneous data. This is because in the enhanced ID3 algorithm, the recursion stops when there is no sample at the node, and thus for the heterogeneous data, it is likely that the data from a node all exists in one branch. Therefore the size of the decision tree is smaller than that in the homogeneous data scenario.

According to the figures, the noise does not affect the energy consumption. We believe that it is because the noise does not change the transmission distance, and it also does not noticeably change the size of the decision trees.

As we modify the basic ID3 algorithm, the constructed decision tree is thus larger in our enhanced ID3 algorithm. We also examine the size of the decision tree built in the two algorithms, and find that the size in enhanced ID3 is 45% larger than that in the basic ID3 algorithm. Considering that the basic ID3 algorithm cannot even work due to its low accuracy in our application, this increased size is reasonably acceptable.

Chapter 5

Conclusion

In this thesis, we proposed a novel hierarchical distributed classification approach in wireless sensor networks. In particular, we consider a practical scenario that the data distribution is heterogeneous, which is seldom studied before. In our solution, local classifiers are built by individual sensor nodes and merged along the routing path. The classifiers are iterative enhanced by combining strategically generated pseudo data and new local data, eventually converging to a global classifier for the whole network. We demonstrate that our approach maintains a high classification accuracy with very low storage and communication overhead.

5.1 Summary of work

In this thesis, we presented the hierarchical distributed classification in wireless sensor networks. Since sending a large amount of data consumes the energy of the sensor node rapidly, our approach is more feasible in sensor networks. In our approach, the decision tree is used as the classifier. The sensor network is organized as a spanning tree, and the base station is located at the tree root. Each leaf node applies a learning algorithm to build a classifier to its parent. Upon receiving a classifier from one child, the intermediate sensor node generates a *pesudo raining dataset* for the received classifier. It combines the generated pseudo data with its local training data or the pseudo training dataset generated from its local classifier that has been built already, applies learning algorithm to build the new classifier, and sends to its parent. At the end, the base station builds a combined classifier from all the collected training data.

To generate pseudo data that is similar to the original data as much as possible, a

modified ID3 algorithm is adopted, so that the decision contains more information about the original data. There was a control factor that controls the amount of the generated pseudo data due to the limited memory of the sensor node. Another factor is the class count which keeps the enough "noise" information.

The evaluation of simulation showed that our approach is well performed. We generated two kinds of data sets: location data, in which the data distribution depends on the location of sensor node and random data, which is independent on the location. We compared our approach with the ensemble method and the basic ID3 algorithm under both location data and random data. For the location data, our method obtained a much more better accuracy and much less energy consumption than ensemble method.

5.2 Future Work

So far we have presented the design and evaluation of the hierarchical distributed classification for wireless sensor networks. Within this framework, there are many possible avenues to further explore; we now list three of them in which we are particularly interested:

First, the module of pseudo data generation is of critical importance in the accuracy and efficiency of our proposal. We plan to further enhance its effectiveness, making the generated data as close to the original as possible with limited side information. The novel idea of generating pseudo data for multi-level classification may also be applied in applications other than sensor networks, particularly for those with large distributed and dynamic data sets.

Second, we have assumed the popular decision tree as the classifier building algorithm in our current work. Our framework however is flexible in accommodating diverse other classification approaches. We are interested in investigating the possibility of customizing other advanced classifier building algorithms, seamlessly incorporating them into our framework, and evaluating and comparing their performance.

Finally, we have addressed certain implementation issues in our design. Yet there remain a series of practical concerns and an efficient implementation may involve cross-layer optimizations. For example, we have assumed that the spanning tree for routing the collected data is fixed and the packet losses are negligible. In real sensor networks, however, the network-layer routing tree may change over time, and our algorithm thus needs to update the relations among the nodes periodically to reflect the change; Forward Error Correction

(FEC) may have to be added to mask the packet losses, and we may need to re-transmission mechanisms to recover from severe losses. We plan to address these issues and possibly implement a working prototype, which will be evaluated over a real sensor network testbed, e.g., a collection of Mica-2 motes.

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[2] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Network*, 46(5):605–634, 2004.

[3] F. Attneave. *Applications of Information Theory to Psychology:A Summary of Basic Concepts, Methods, and Results*. Holt, New York, 1959.

[4] F. Bouhafs, M. Merabti, and H. Mokhtar. A Node Recovery Scheme for Data Dissemination in Wireless Sensor Networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, 2007.

[5] I. Bratko and M. Bohanec. Trading accuracy for simplicity in decision trees. *Mach. Learn.*, 15(3):223–250, 1994.

[6] L. Breiman. Pasting bites together for prediction in large data sets. *Machine Learning*, 36(2):85–103, 1999.

[7] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[8] L. Breiman. Bagging Predictors. *Machine Learning*, 24(2):123–140, 2002.

[9] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Statistics/Probability Series. Wadsworth Publishing Company, Belmont, California, U.S.A., 1984.

[10] R. R. Brooks, P. Ramanathan, and A. M. Sayeed. Distributed Target Classification and Tracking in Sensor Networks. *Proceedings of the IEEE*, 91(8):1163–1171, 2003.

[11] M. Castillo-Effen, D.H. Quintela, R. Jordan, W. Westhoff, and W. Moreno. Wireless sensor networks for flash-flood alerting. In *Proceedings of the Fifth IEEE International Caracas Conference on Devices, Circuits,and Systems, Dominican Republic*, 2004.

[12] J. Catlett. *Mega Induction: Machine Learning on Vary Large Databases*. PhD thesis, Univ. Sydney.

[13] P. Chan and S. Stolfo. On the accuracy of meta-learning for scalable data mining. *Journal of Intelligent Information Systems*, 8:5–28, 1996.

[14] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning Ensembles from Bites: A Scalable and Accurate Approach. *Machine Learning Research*, 5:421–451, 2004.

[15] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

[16] D. Cruller, D. Estrin, and M. Srivastava. Overview of sensor networks. *Computer ISSN 0018-9162*, 37(8):41–49, 2004.

[17] T. G. Dietterich. Ensemble Methods in Machine Learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems (MCS)*, 2000.

[18] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the World with Wireless Sensor Networks. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2001.

[19] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks. In *Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, 1999.

[20] Y. Freund. Boosting a Weak Learning Algorithm by Majority. In *Proceedings of Workshop on Computational Learning Theory*, 1990.

[21] Y. Freund and R. E. Schapire. A Decision-theoretic Generalization of On-line Learning and An Application to Boosting. In *Proceedings of European Conference on Computational Learning Theory (EuroCOLT)*, 1995.

[22] T. Gao, D. Greenspan, M. Welsh, R.R. Juang, and A. Alm. Vital signs monitoring and patient tracking over a wireless network. In *Proceedings of the 27th IEEE EMBS Annual International Conference*, 2005.

[23] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. In *In VLDB*, pages 416–427. Morgan Kaufmann, 1998.

[24] G.Simon, M.Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J.Sallai, and K. Frampton. Sensor network-based countersniper system. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems (Sensys)*, Baltimore, MD, 2004.

[25] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. A. Stankovic, T. Abdelzaher, and B. H. Krogh. Lightweight Detection and Classification for Wireless Sensor Networks in Realistic Environments. In *Proceedings of the ACM International Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

[26] J. Han and M. Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, September 2000.

[27] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An Application-Specific Protocol Architecture for Wireless Microsensor Networks. *IEEE Transactions on Wireless Communications*, 1(4):660–670, 2002.

[28] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Appl. Statist.*, 29(2):119–127, 1980.

[29] S. B. Kotsiantis and P. E. Pintelas. Combining bagging and boosting. *International Journal of Computational Intelligence*, 1:324–333, 2004.

[30] S. Kumar, F. Zhao, and D. Shepherd. Collaborative Signal and Information Processing in Microsensor Networks. *IEEE Signal Processing Magazine*, March 2002.

[31] B. Lantow. Impact of wireless sensor network data on business data processing. In *Proceediing of the Forum Poster Session in Conjunction with Business Informatics Research (BIR) 2005*, 2005.

[32] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *KDD*, pages 311–316.

[33] A. Lazarevic and Z. Obradovic. The Distributed Boosting Algorithm. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2001.

[34] S. Lindsey, C. Raghavendra, and K. M. Sivalingam. Data Gathering Algorithms in Sensor Networks Using Energy Metrics. *Transactions on Parallel and Distributed Systems*, 13(9):924–935, 2002.

[35] K. Lorincz, D. Malan, T.R.F. Fulford-Jones, A. Nawoj, A. Clavel, V.Shnayder, G. Mainland, M. Welsh, and S. Moulton. Sensor networks for emergency response: challenges and opportunities. *Pervasive Computing, IEEE*, 3(4):16–23, October–December 2004.

[36] P. Luo, H. Xiong, K. Lu, and Z. Shi. Distributed Classification in Peer-to-Peer Networks. In *Proceedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2007.

[37] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2002.

[38] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. pages 18–32, 1996.

[39] C. J. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36:33–58, 1999.

[40] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, March 1986.

[41] J. R. Quinlan. Induction of Decision Trees. In *Machine Learning*, chapter 1, pages 81–106. Kluwer Academic Publishers, 1986.

[42] J. R. Quinlan. Decision trees and multi-valued attributes. 11:305–318, 1988.

[43] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, California, U.S.A., 1993.

[44] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers – a survey, 2005.

[45] J. C. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *Proc. of the 22nd International Conference on Very Large Databases*, pages 544–555. Morgan Kaufmann, 1996.

[46] P. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, May 2005.

[47] G. Tsoumakas and I. Vlahavas. Effective stacking of distributed classifiers. In *the 15th European Conference on Artificial Intelligence*, pages 340–344.

[48] G. Wener-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Walsh. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, (2):18–25, March–April 2006.

[49] J. Wieselthier, G. D. Nguyen, and A. Ephremides. On the Construction of Energy-Efficient Broadcast and Multicast-trees in Wireless Networks. In *Proceedings of IEEE INFOCOM*, 2000.

[50] J. Yick, B. Mukherjee, and D. Ghosal. Analysis of a Prediction-based Mobility Adaptive Tracking Algorithm. In *Proceedings of the IEEE Second International Conference on Broadband Networks (BROADNETS)*, Baltimore, MD, 2005.

[51] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, August 2008.