Machine Learning for IoT Applications: Sensor Data Analytics and Data Reduction Techniques

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Electronics and Communication Engineering
by Research

by

Adarsh Pal Singh
20161225
adarshpal.singh@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
June 2020

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Machine Learning for IoT Applications: Sensor Data Analytics and Data Reduction Techniques" by Adarsh Pal Singh, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Sachin Chaudhari

To

My Family and Friends

Acknowledgments

It was the day of the result. My heart was pounding as I opened the admission portal of IIIT Hyderabad on my laptop's browser. Somehow, I managed to type in my credentials even though my hand was trembling with nervousness. "Congratulations!" said the website and my life was never the same. The four years I spent in IIIT Hyderabad were undoubtedly some of the most memorable years of my life, filled with learning, exciting opportunities and new experiences. I couldn't have asked for more. I'm ever so grateful to God for not only blessing me with so many opportunities but also giving me the strength to see them through.

The first person I'd like to thank and express my gratitude towards is my research advisor, Dr. Sachin Chaudhari. I joined his research group as a total amateur and am walking out as an experienced researcher. None of this would have been possible without his constant guidance and unbounded support. He never doubted my capabilities even during the low points of our research journey and always encouraged me back on track. The dedication and passion he puts into his own work have been a constant source of inspiration for me.

I would also like to express my gratitude towards Prof. Stefan Werner and Dr. Frank Alexander Kraemer for their invaluable guidance on the occupancy project. They were generous enough to host me as a research intern at their university, NTNU Trondheim. The two months I spent working under them in Norway had a significant impact not only on my skills as a researcher, but also on my personal development.

I would like to extend my deepest thanks to my good friend and mentor, Anish Shastri, who has been a constant pillar of support and knowledge throughout my research years. Special thanks to my good friend and collaborator, Vivek Jain, who has been my partner in crime throughout college life. Many thanks to my A3-203 lab mates: Zakir, Ayush, Akhil, Ruchi, Kali and Rajashekar, for making the lab environment productive as well as cheerful.

I would also like to give a shout out to all my friends without whom my college life would have been pretty dull. High five to all the idiots in my hostel wing: Malani, KT, Manan, Sharique, Vaibhav and Bakhtiyar. Fist bump to those who were not so fortunate enough to be in the same wing as me: Daga, Ishan, Alakh, Modi, Jayaganesh, Manas, Jagadale and all others (please forgive my limited memory). I would also like to thank my good buddy since high school, Kadri, on whom I can count on no matter how big the crisis. A heartfelt thanks to my cousin, Ravjot, for always being there to listen to my rants and advising appropriate course of action.

My endless gratitude goes towards my parents and my sister. Words cannot describe how much they've supported me throughout my life. None of this would have been possible without their unwavering support and blessings. Thanks for always believing in me and constantly encouraging me to achieve new heights. I would also like to thank my relatives for their constant love and blessings. A special shout out to my cousin brothers and sisters for regularly checking up on me.

Abstract

Internet of things (IoT) devices are steadily becoming mainstream in our lives and simplifying the manner in which we perform everyday tasks. With billions of smart devices, embedded with all kinds of transducers, communicating with each other and us humans via the internet, data is being generated at an alarming rate. Consequently, data analytics has become an integral part of the IoT ecosystem. Several IoT applications, including those that come under the umbrella of smart home, intelligent transportation system, smart healthcare and smart grid, rely on machine learning (ML) to gain inference from sensor data and aid in decision making. Apart from user applications, ML is also used in the lower layers, including the network layer and the medium access control (MAC) layer, to improve the efficiency of IoT networks.

The focus of this thesis is on the application of ML in making IoT systems smarter and more efficient. More specifically, a new ML-based paradigm for occupancy estimation is introduced along with an ML-based data transmission reduction scheme that is validated for occupancy applications. Having knowledge of the occupancy status of rooms can help in automating lighting systems, cooling/heating systems as well as a myriad of other appliances throughout the building. The proposed occupancy estimation paradigm employs ML algorithms on a deployment of multiple non-intrusive sensor nodes in a room. Two new datasets were created in the process which are thoroughly investigated by applying various preprocessing, feature engineering and ML techniques. Multiple performance metrics like accuracy, F1 score and confusion matrix are reported for a variety of homogeneous and heterogeneous feature combinations. A novel ML-based data transmission reduction scheme is also proposed for application-specific IoT networks that takes the application into account when deciding the initiation of transmission. Wireless data transmission is infamous for being the most energy consuming activity of a sensor node. Therefore, intelligently reducing data transmissions can prolong the lifetime of battery-powered sensor nodes without comprising much on the data quality. To enable the proposed scheme for constrained sensor nodes, the complexities of various ML algorithms are discussed, both theoretically and practically, from the perspective of constrained microcontrollers. One of the occupancy estimation datasets along with a standard dataset for occupancy detection are used for validating the proposed data transmission reduction scheme. Experimental results demonstrate a humongous reduction in the number of data transmissions while upholding similar performance metrics. The proposed scheme is also shown to significantly outperform the Shewhart change detection algorithm for occupancy applications.

Contents

List of Figures

List of Tables

List of Abbreviations

AC	Air Conditioner
ADC	Analog to Digital Converter
AI	Artificial Intelligence
ANN	Artificial Neural Network
ARIMA	Autoregressive Integrated Moving Average
BIRCH	Balanced Iterative Reducing and Clustering using Hierarchies
BLE	Bluetooth Low Energy
CLIQUE	Clustering In Quest
CNN	Convolutional Neural Network
CO_2	Carbon Dioxide
CSMA/CA	Carrier-Sense Multiple Access with Collision Avoidance
CURE	Clustering Using Representatives
CV	Computer Vision
D2D	Device to Device
DBN	Deep Belief Network
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DKF	Dual Kalman Filter
DL	Deep Learning
DT	Decision Tree
ECG	Electrocardiogram
ELM	Extreme Learning Machine
EWMA	Exponential Weighted Moving Average
GNB	Gaussian Naive Bayes
GPS	Global Positioning System
HDI	Human-Device Interaction
HMI	Human-Machine Interface
HMM	Hidden Markov Model
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IoV	Internet of Vehicles

ISP	Internet Service Provider
KNN	K-Nearest Neighbor
LAN	Local Area Network
LDA	Linear Discriminant Analysis
LMS	Least Mean Square
LoRa	Long Range
LPWAN	Low-Power Wide Area Network
LR	Linear Regression
LSTM	Long Short-Term Memory
LTE	Long Term Evolution
MA	Moving Average
MAC	Medium Access Control
ML	Machine Learning
MLDR	Machine Learning-based Data Reduction
MLP	Multilayer Perceptron
MP	Megapixel
MSE	Mean Squared Error
MST	Minimum Spanning Tree
NB-IoT	Narrowband IoT
NDR	No Data Reduction
NLP	Natural Language Processing
NMF	Non-negative Matrix Factorization
OPTICS	Ordering Points To Identify the Clustering Structure
PCA	Principal Component Analysis
PIR	Passive Infrared
PPM	Parts Per Million
QDA	Quadratic Discriminant Analysis
RBF	Radial Basis Function
RES	Renewable Energy Sources
RF	Random Forest
RFID	Radio Frequency Identification
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
SDR	Shewhart-based Data Reduction
SLR	Simple Linear Regression
SMS	Short Message Service
STING	Statistical Information Grid

SVD	Singular Value Decomposition
SVM	Support Vector Machine
SVR	Support Vector Regression
T-SNE	T-distributed Stochastic Neighbor Embedding
VUI	Voice User Interface
WAN	Wide Area Network
WiFi	Wireless Fidelity
WSN	Wireless Sensor Network
YOLO	You Only Look Once

List of Symbols

a	Action taken by an RL agent
A	Set of all possible actions defined in the RL environment
b	Bias term
c	Constant term
d	Dimensionality (number of features of the dataset)
D_{avg}	Average depth of the decision tree
$f_k(\ .\ )$	Probability density function (PDF) of (.) conditioned under class k
$g(\ .\ )$	Non-linear activation function
k	Class label
$\hat{k}$	Estimated class label
m	Slope of linear regression
n	Number of quantities present in the sensed array
N_{leaf}	Number of leaf nodes of a DT
p_i	Inverse of standard deviation of the ith feature
$\mathbf{p}$	Element-wise inverse array of the standard deviation array
$P(\ .\ )$	Probability of (.)
s	Current state of an RL agent
s_i	Standard deviation of the ith feature
$\mathbf{s}$	Standard deviation array
S	Set of all possible states defined in the RL environment
u_i	Mean of the ith feature
$\mathbf{u}$	Mean array of standard scaling
w_i	ith value of the weight vector
$\mathbf{w}$	Weight vector
x	Independent variable in linear regression
x_i	ith value of the input feature vector
$x_{s,i}$	ith value of the sensed feature vector
$x[n]$	Sensed array at the sensor node comprising of n features
$\mathbf{x}$	Input feature vector
$\mathbf{x}_s$	Sensed feature vector

$\bar{\mathbf{x}}_s$	Standard-scaled sensed feature vector
$\mathbf{X}$	Feature vectors of the training dataset
Y	Set of class labels of the training dataset
$\boldsymbol{\Sigma}$	Covariance matrix
$\boldsymbol{\Sigma}_k$	Covariance matrix under class k
$\boldsymbol{\Sigma}_{diff}$	Difference of covariance matrices of two different classes
$\mu_{k,i}$	Mean of feature i under class k
$\boldsymbol{\mu}$	Mean vector
$\boldsymbol{\mu}_k$	Mean vector under class k
$\rho^2_{k,i}$	Inverse of variance of feature i under class k
$\boldsymbol{\rho}^2_{\boldsymbol{k}}$	Element-wise inverse of variance vector under class k
$\sigma^2_{k,i}$	Variance of feature i under class k
$\boldsymbol{\sigma}^2_{\boldsymbol{k}}$	Variance vector under class k

Chapter 1

Introduction

1.1 Motivation

Internet of things (IoT) is a fairly recent and novel paradigm that envisions a network of connected objects and appliances that use the internet for communicating with each other as well as humans. Gone are the days when full-fledged computing devices were the only ones with internet access! Nowadays, millions of *smart* objects embedded with microcontrollers, transducers and wireless transceivers are pinging the internet servers every minute and this number is growing at an unprecedented rate [1]. Consequently, IoT devices are generating massive amounts of sensor data and are currently relying heavily on cloud for data storage and analytics.

A plethora of IoT applications rely on machine learning (ML) to infer useful patterns from sensor data [2, 3, 4]. One such application is of occupancy estimation in smart homes and buildings. Having a real-time account of the occupancy status of rooms can help in automating a lot of systems including lighting, cooling as well as other electrical appliances. Clearly, occupancy-driven automation in smart buildings can make them much more energy efficient [5]. Early approaches for occupancy estimation resulted in the use of intrusive systems which relied on cameras, WiFi, wearables and radio-frequency identification (RFID) [6, 7, 8, 9]. With the rise in concerns regarding privacy and smarter paradigms that do not require direct human intervention gaining traction, research in recent years has moved towards the use of non-intrusive environmental sensors like temperature, humidity, light, CO_2, sound, etc., for inferring the occupancy count. In continuation of this trend, a new ML-based paradigm for occupancy estimation using non-intrusive sensors is presented in this thesis.

Many IoT applications require wireless sensor networks (WSNs) to be deployed in regions that lack the infrastructure to provide continuous wired power supply to the sensor nodes. In such applications, it is common to have battery-powered sensor nodes with an optional energy harvesting module like photovoltaic cells. Needless to say, these nodes are constrained, both in terms of the computational power and the energy quota. In order to conserve energy, these sensor nodes typically follow a sleeping pattern wherein they wake up periodically to transmit the sensor data. It is well known that wireless data transmission consumes the most energy in comparison to the other activities performed by the sensor

node [10]. Therefore, a good amount of research has been carried out on techniques that reduce the amount of data transmissions without compromising much on the data quality. Most of these techniques employ a data prediction model that runs on both the sensor node and the edge device responsible for collecting the data. The sensor node initiates transmission only when the predicted sensor value and the actual sensor value differ by some predefined threshold [11]. Clearly, such thresholding-based techniques do not take the IoT application into account when deciding the initiation of transmission. In a first, an ML-based data transmission reduction scheme for application-specific IoT networks is proposed in this thesis. The proposed scheme is tailored and practically validated for occupancy applications.

1.2 Summary of Contributions

The main contributions of this thesis are split among two chapters:

- **Chapter 3**
  - A new occupancy estimation paradigm is proposed that uses ML on a deployment of multiple non-intrusive sensor nodes.
  - Two new labelled datasets pertaining to the proposed occupancy estimation paradigm have been created.
  - ML-based sensor fusion analysis for various homogeneous and heterogeneous combinations of sensors is presented.
  - A new feature is engineered out of CO_2 values and is proposed for occupancy applications.
- **Chapter 4**
  - Five ML algorithms are analysed (both theoretically and practically) for space and time complexity in the context of constrained microcontrollers.
  - A new ML-based data transmission reduction scheme is proposed for constrained sensor nodes belonging to some application-specific IoT network.
  - The proposed data transmission reduction scheme is tailored and validated for occupancy applications and is shown to outperform Shewhart-based data reduction algorithm.

1.3 Thesis Organization

The remainder of this thesis abides by the following organization structure:

- *Chapter 2* gives a brief introduction to IoT and ML. Moreover, a review of how ML is playing a pivotal role in many IoT applications is also presented.

- In *Chapter 3*, the proposed occupancy estimation paradigm is described. The description includes the hardware deployment of the sensor nodes for dataset collection as well as the ML-based analysis of the dataset. A real-time implementation of the occupancy system is also presented.
- *Chapter 4* presents an analysis of various ML algorithms from the perspective of constrained microcontrollers. This analysis is taken as a stepping stone in the proposition of a new data transmission reduction scheme for application-specific IoT networks. The proposed scheme is validated for occupancy applications and compared to the Shewhart-based data reduction algorithm.
- *Chapter 5* serves as the conclusion of the thesis.

Chapter 2

Machine Learning for Internet of Things

2.1 Internet of Things

Internet of things (IoT), as the name suggests, refers to the ecosystem of *things* or common physical objects that are accessible via the internet. The initial years of the 21st century saw tremendous advancements in portable computer devices in tandem with communication technologies which led to the rapid adoption and expansion of the internet. The entire world was literally compressed into the palm of a common man which, needless to say, changed everything. As more and more devices got connected to the internet and semiconductors became cheaper by the minute, a new wave of *smart* devices started to make way into people's homes. The idea was simple: why can't our stupid manual appliances be controlled and automated via the internet? Why can't I access my refrigerator from my phone when I'm at the convenience store to check what items I need to buy? Why can't I program my microwave remotely from my phone so that my food is hot and ready the moment I walk into my home? Why can't my home understand my behavior and automate appliances on its own to make my life easy? Ideas like these extended the internet connectivity to non-standard computing devices thereby making an internet of *things*. By the year 2025, it is expected that the number of connected IoT devices would surpass the 40 billion mark [1]!

2.1.1 Interaction Between Humans and Smart Devices

IoT aims to revolutionize the way humans interact with their surroundings. The IoT ecosystem relies on two types of interactions:

1. **Human-Device Interaction (HDI):** Humans interact with the IoT devices in two ways: directly command devices for a particular task using some form of human-machine interface (HMI) or monitor some automated process [12]. IoT applications for the first type include interacting with smart electrical appliances like smart refrigerators, smart televisions, smart microwave ovens, smart lighting and air conditioning (AC) systems, etc. Such devices typically come with a mobile or a web interface that the user can interact with from anywhere in the world. Nowadays, with

the advent of natural language processing (NLP), speech is being used more and more rapidly to interact with these smart objects thanks to the explosion of the smart speaker market [13]. In the second form of interaction, humans are a part of the IoT process but do not engage with the ecosystem directly. Most sensor-based IoT systems use this form of interaction to push analysed data to the end user. The most popular example for this are the fitness trackers or the smart watches that sense various biological parameters. The processed data can be visualized on a mobile or a web interface. Other popular examples include smart surveillance systems that can notify various security aspects directly to the user and smart energy monitoring devices. The systems that use this second type of interaction generally require data analytics to be performed at the edge or the cloud before the end user can be apprised of the results.

2. **Device to Device (D2D):** Device to device interaction forms the basis of automation in IoT. The holy grail of IoT is complete automation with as little human intervention as possible. Think of examples like home appliances such as lights being automatically controlled based on human activity in the house or a refrigerator learning from one's eating habits and directly contacting an online grocery delivery store for products that are running low. Many practical IoT systems like smart speakers require a mixture of HDI and D2D interactions wherein the user interacts with one device which in turn interacts with other devices to complete the required task. D2D communication is a big challenge considering the diverse portfolio of standards and communication technologies that exist in the market today [14].

2.1.2 Components of an IoT System

A typical IoT system comprises primarily of four components: IoT devices, IoT gateway, cloud and a user interface. The interaction between these components is shown in Figure 2.1.

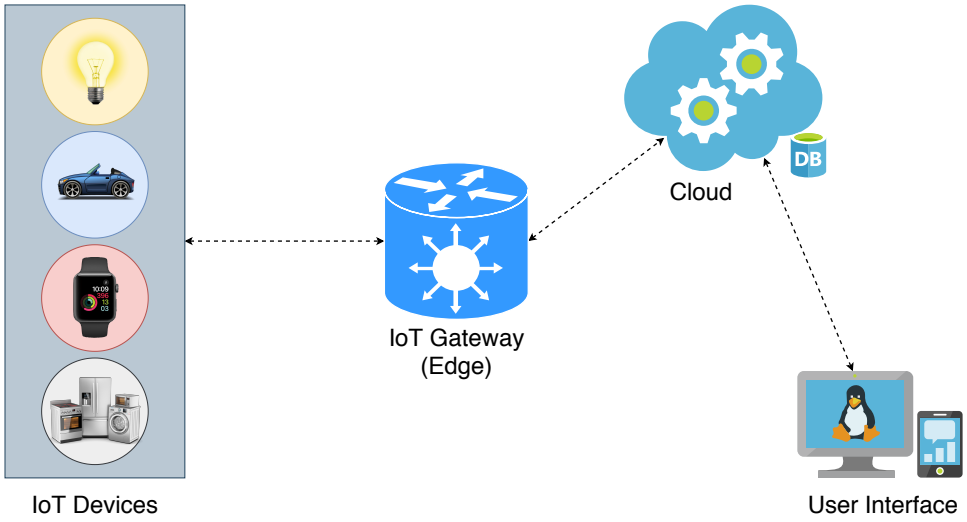

Figure 2.1 The four basic building blocks of a typical IoT system.

1. **IoT Device:** An IoT device is simply a connected object that can exchange data via the internet. This includes a myriad of devices that are termed *smart* such as smart bulbs, smart refrigerators, smart televisions, smart coffee machines, smart meters, smart vehicles, smart valves, smart watches, etc. Typically, general purpose computing devices like computers and smartphones are omitted from this list. Diving into a lower level, we can describe an IoT device as an embedded system having a microcontroller, one or more transducers (sensors and/or actuators) and a communication interface to connect to the internet via a gateway.

   A sensor is a transducer that senses and quantifies physical parameters like temperature, humidity, light, gas concentration, acceleration, pressure, etc. Sensing is an important task for many IoT systems such as smart watches, smart meters and smart homes. Such systems often employ various data analytics techniques to process raw data from one or more sensors into parameters that form the basis of decision making [15]. Apart from sensors, many IoT devices employ actuators to interact with the physical environment. Actuators are, in a way, the antithesis of sensors because they take an electrical signal as input and produce some physical action based on it. Even the most rudimentary task of turning on an electrical appliance remotely requires a relay actuator. Sensor and actuator-based IoT systems may either be standalone (as exemplified) or integrated by a pipeline following an automation cycle of *sense* $\rightarrow$ *analyze* $\rightarrow$ *actuate*.

2. **IoT Gateway:** A gateway, as the name suggests, acts as a gate that allows interconnection of two dissimilar networks. In other words, a gateway acts as a translation medium that allows seamless communication between two networks that communicate using different protocols. A broadband wireless router present in most homes throughout the world is a common example of a residential gateway that connects all sorts of devices on the local area network (LAN) to the wide area network (WAN) (operated by the internet service provider (ISP)) via a modem. Since these gateways are present at the periphery of a LAN and an ISP's WAN, they are also referred to as edge gateways.

   There are many communication technologies and protocols that are used by the IoT devices. These include WiFi (802.11b/n/g/ah), Bluetooth low energy (BLE), ZigBee (IEEE 802.15.4), Z-wave, LoRa, NB-IoT and LTE-M [15]. Naturally, an IoT device using any of these communication standards would require an appropriate gateway to reach the internet (cloud). Most smart home devices use traditional WiFi to ensure compatibility with existing residential WiFi gateways. These gateways not only provide connectivity from device to the cloud, but also among the devices themselves. For example, a smart speaker can ping a smart bulb over a WiFi router without needing to access the cloud. Low-power IoT devices like smart watches prefer BLE and communicate with the cloud using a smartphone as a gateway. LoRa, NB-IoT and LTE-M are shaping up to be the standards for low power wide area networks (LPWANs) [16].

   Low-latency applications for IoT have driven the need for performing computations closer to the source of data generation (edge). After all, one needs to propagate through the packet-switched

internet to reach the cloud which adds a considerable latency. Edge computing, therefore, is a welcome addition to the existing IoT paradigm which relied heavily on the cloud. By definition, any computing resource in between the data generating IoT devices and the cloud, that requires just one or two network hops, is referred to as an edge device [17]. This includes the IoT devices themselves, the IoT gateway and also the edge data centers that ISPs provide. With the advent of edge computing, manufacturers have started manufacturing edge gateways with high computational capabilities. Edge computing not only arms the IoT system with real-time computational capabilities, but also saves considerable amount of bandwidth by not sending each and every thing to the cloud. It should be clear that the purpose of edge computing is not to eradicate the need for cloud computing in IoT but to complement it.

3. **Cloud:** It is no secret that with the explosion in the number of IoT devices, data is being generated at an alarming rate. In fact, by 2025, these devices would be generating data close to 79.4 zettabytes (ZBs) [1]! Thankfully, even before IoT became a mainstream concept, cloud had already integrated itself into the application development cycle. Cloud empowers IoT with distributed storage, computational power as well as networking capabilities. With the big data and machine learning tools already integrated into the cloud ecosystem coupled with benefits like elasticity, reliability, scalability of resources and the fact that cloud machines are globally accessible, one might say that IoT could not have enjoyed its success without cloud [18]. For many applications like those involving direct HDI interactions, cloud may serve as a simple network router between the IoT devices and the end user. For more complex IoT applications, the cloud machines are integrated with the IoT devices (over the gateway, of course) to handle and process raw data. The processed data can either be pushed to the IoT devices (automation) via the gateway or pushed to the user via some kind of user interface.

4. **User Interface:** User interfaces are what fuel the human-device interactions in IoT. They enable the user of the IoT system to easily configure devices or consume processed information or both. The interface may be something as simple as an SMS, internet chat or an email-based push notification service or a full fledged web/native mobile application. These applications ping a cloud endpoint in the backend and display the data and virtual control buttons in an appropriate manner so as to make things easy for the user to understand. Figure 2.2 shows a web-based user interface from ThingSpeak, a popular cloud-based IoT analytics platform. Data from four sensors: temperature, light, sound and CO_2 were being monitored by a NodeMCU and pushed to ThingSpeak (cloud) in real time via a WiFi gateway. The latest advancement in the domain of IoT user interfaces is the voice user interface (VUI) that is primarily employed in smart speaker-based IoT solutions. In fact, the smart speakers with their voice activated technology has helped fuel the IoT adoption all over the globe [13].

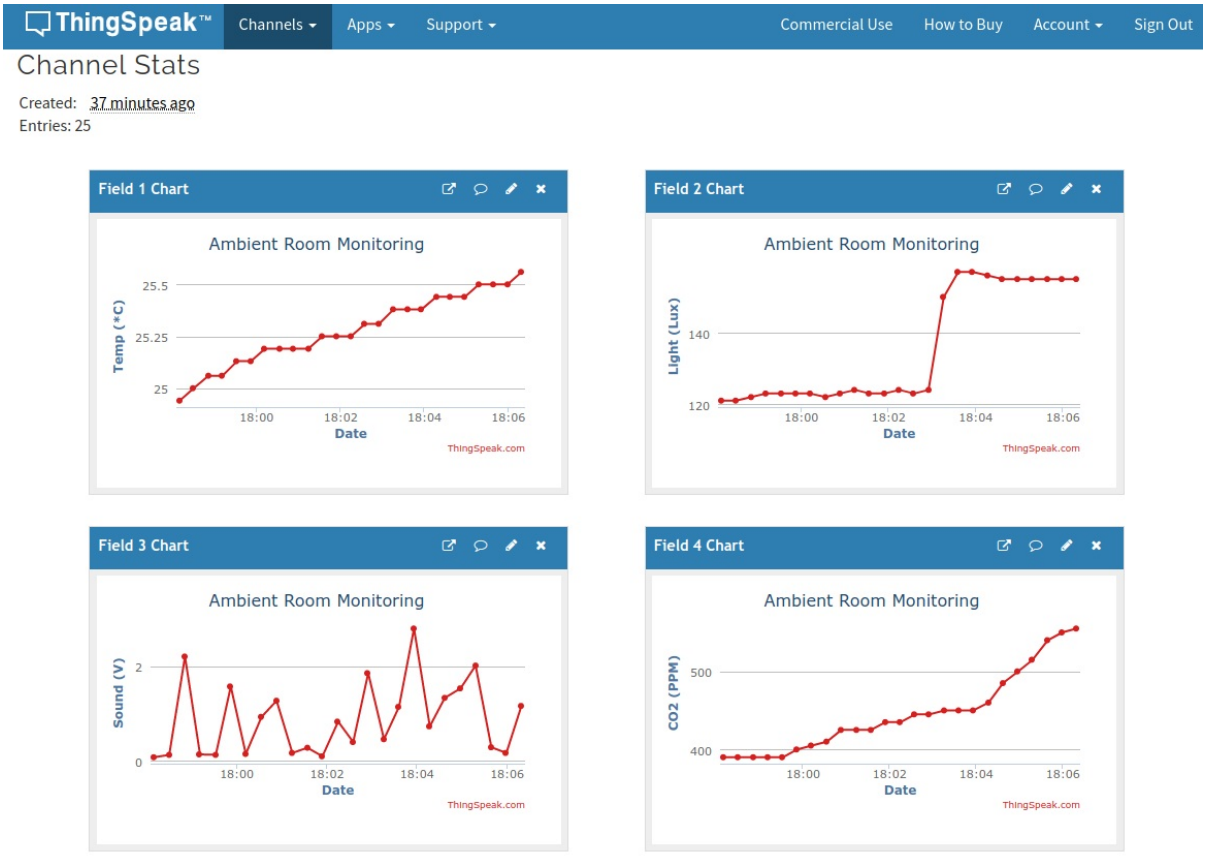

Figure 2.2 An exemplar IoT user interface for a data monitoring system.

We have talked about how the data from IoT devices can be analyzed at the edge and the cloud. The next section describes a very popular branch of artificial intelligence that has become synonymous with data analytics.

2.2 Machine Learning

Machine learning (ML) is a branch of artificial intelligence (AI) that enables computers to learn to perform specific tasks without being explicitly programmed to do so. Even though the foundation of many ML algorithms were laid down in the 20th century, it was not until the second decade of the 21st century that these algorithms actually saw practical deployments (thanks to the massive increase in data and the computing power per dollar). ML algorithms have proven to be quite effective in solving real-world problems including those posed by IoT [2, 3, 4]. Based on the type of input data and application constraints, ML algorithms can be roughly divided into three categories: supervised, unsupervised and reinforcement learning.

2.2.1 Supervised Learning

As the name suggests, supervised ML algorithms require *supervision* when it comes to learning (training) in the form of a labelled dataset (a dataset having the ground truth of the output variable along with the data). Based on the output variable type, supervised learning algorithms can be further divided into two categories:

1. **Classification-Based:** The output variable in classification problems (more popularly referred to as the class label) is categorical or discrete in nature. Examples for this include determining whether an image has a cat or a dog in it, classifying an email as spam or not spam, determining the type of tree based on its features, etc. These algorithms learn discriminating patterns among different outcome classes in the training phase and post training, predict the outcome class by recognizing these patterns in new and unseen data in what is known as the testing or the inference phase. Since the upcoming chapters rely heavily on the use of classification-based supervised ML algorithms, they are discussed in brief here.

   - **Linear Discriminant Analysis (LDA):** LDA is a simple yet popular linear classifier that assumes the class conditional distribution of the data, $P(\mathbf{X} = \mathbf{x}|Y = k)$, to be a multivariate Gaussian given by

   $$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)} \tag{2.1}$$

   Here, $\mathbf{X}$ represents the feature vectors of the training set, Y denotes some class k out of the set of classes $\{1, ..., K\}$, d signifies the dimension or the number of features the data has, $\boldsymbol{\mu}_k$ is the mean vector of the data under class k and $\mathbf{\Sigma}$ is the covariance matrix which is common for all the classes. In the training phase, $\boldsymbol{\mu}_k$, $\mathbf{\Sigma}$ and class priors $P(Y = k)$ are learnt from the training data. In the testing phase, class label for a new data vector $\mathbf{x}_s$ is predicted using Bayes' rule:

   $$P(Y = k|\mathbf{X} = \mathbf{x}_s) = \frac{f_k(\mathbf{x}_s)P(y = k)}{\sum_{i=1}^{K} f_i(\mathbf{x}_s)P(y = i)} \tag{2.2}$$

   - **Quadratic Discriminant Analysis (QDA):** As the name suggests, QDA is a quadratic decision boundary classifier that has the same formulation as LDA except for the fact that the covariance matrix $\mathbf{\Sigma}_k$ is calculated separately for each class in the training phase. Testing is done using Bayes' rule as formulated in (2.2).
   - **Gaussian Naive Bayes (GNB):** Gaussian naive Bayes, as the name suggests, is based on the Bayes' theorem coupled with the *naive* assumption that all the d features of the dataset given the class label k are Gaussian and mutually independent. Let $\mathbf{x}_s = (x_1, x_2, ..., x_d)$ be a d-dimensional data vector. Using Bayes' theorem,

   $$P(Y = k|\mathbf{X} = \mathbf{x}_s = (x_1, ..., x_d)) = \frac{P(Y = k)P(x_1, ..., x_d|Y = k)}{P(x_1, ..., x_d)} \tag{2.3}$$

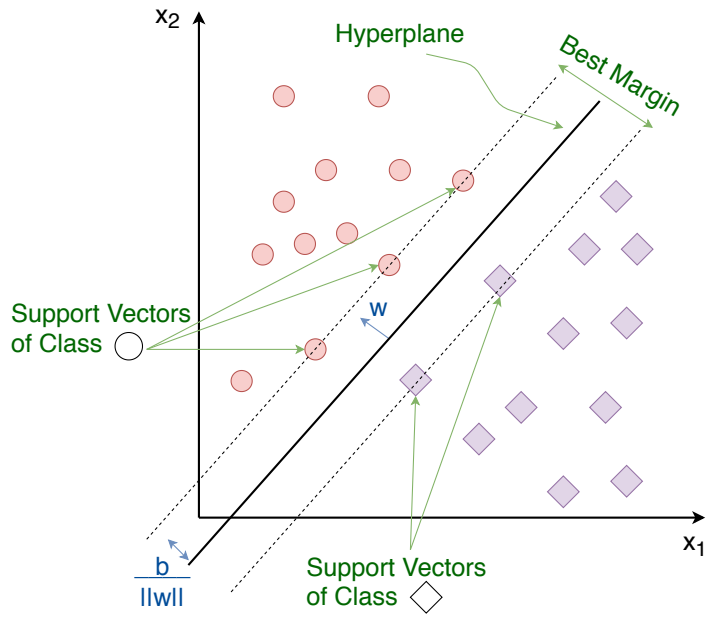

Figure 2.3 A trained linear SVM classifier for a 2-dimensional binary-class dataset.

The denominator in the above equation is same for all the classes. In the testing phase, the class k that maximizes the above probability equation is chosen as the inference. Using the independence assumption of the features, a generic classifier can be formulated:

$$\hat{k} = \arg\max_{k} P(Y = k) \prod_{i=1}^{d} P(x_i | Y = k) \tag{2.4}$$

- **Support Vector Machine (SVM):** SVM is one of the most popular supervised machine learning classifiers. Unlike LDA, QDA and GNB, SVM does not make any assumptions about the distribution of the data. Rather, it simply attempts to fit an optimal hyperplane in the training phase that separates one class from the other. Think of each d-dimensional feature vector as a point in a d-dimensional space. Then, for 2D, this hyperplane is simply a line that separates the points of one class from the other (Figure 2.3) and for 3D, it is a plane and so on. This separating hyperplane, in a nutshell, is what is learnt in the training phase. For multiclass classification, a one-versus-all scheme is used. The optimization formulation that is used during the training phase of SVM can be found in [19]. The class label of a new data vector $\mathbf{x}_s$ can be predicted by checking in which half of the separating hyperplane it lies in:

$$\mathbf{w}^T \bar{\mathbf{x}}_s \underset{0}{\overset{1}{\gtrless}} b \tag{2.5}$$

Here, $\mathbf{w}$ is the $d \times 1$ floating-point weight vector corresponding to the coefficients of the hyperplane, b is the negative of the intercept of the hyperplane and $\bar{\mathbf{x}}_s$ is the standard-scaled transformed vector of $\mathbf{x}_s$. Standard scaling is necessary since the algorithm is scale variant.

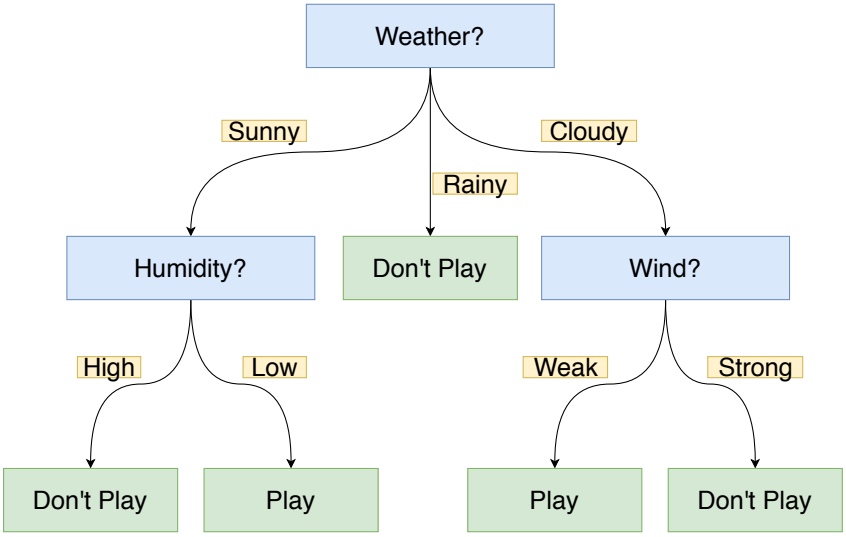

Figure 2.4 A simple decision tree for deciding whether to play badminton outdoors or not based on the weather forecast.

This seemingly linear boundary classifier can be transformed into a non-linear boundary classifier by using a kernel trick. This idea is simple: a non-linearly separable dataset in its current space may become linearly separable when projected in a higher dimensional space (the linear separation in a higher dimensional space will map to a non-linear boundary in the current space). Kernel functions make the computations for carrying out such a procedure simple by avoiding explicit mapping between the spaces. The most popular kernel function that is used in SVM is the radial basis function (RBF).

- **Decision Tree (DT):** Decision tree, in simple terms, learns optimum decision rules from the training set and discriminates between different class labels based on these rules. In a nutshell, a trained DT model is an *if-else ladder* in the shape of a tree with branches feeding to the test nodes (a logical test is performed on the value of a particular feature) and the leaves corresponding to one of the class labels. Figure 2.4 shows an exemplar DT for deciding whether to play badminton outdoors or not based on the weather forecast. Clearly, only the logical comparison operation is needed for inference. A more popular variant of DT that is mostly used in practical applications is random forest (RF). RF is an ensemble of DTs where each tree is grown on some part of the dataset with replacement and has a vote. The final outcome is the average of the votes of all the trees.

- **Multilayer Perceptron (MLP):** A perceptron, sometimes referred to as a neuron, is a linear classifier that forms the basis of the infamous artificial neural networks (ANN). Figure 2.5 shows the model of a perceptron. For a d-dimensional input vector $\mathbf{x}$, output y is computed

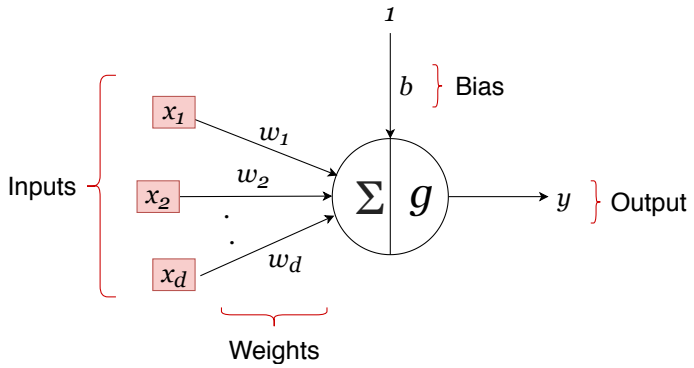

Figure 2.5 Model of a perceptron (neuron).

as

$$y = g\left(\sum_{i=1}^{d} w_i x_i + b\right) \tag{2.6}$$

Here, $\mathbf{w}$ is the d-dimensional weight vector, b is the bias and g is a non-linear activation function. For binary classification problems, a step or a threshold-based activation function is chosen that maps the formulation inside g in (2.6) to one of the two possible classes.

A multilayer perceptron shatters this *linear classification* limitation of the perceptron by stacking multiple perceptrons in a layer-wise fashion. Figure 2.6 shows the architecture of an MLP. Each MLP consists of an input layer, one or more hidden layers (optional) and an output layer. The input layer has the same number of neurons as the dimension of the dataset. The data from the input layer gets fed into the neurons of the first hidden layer. The number of hidden layers in an MLP decides how complex a representation the network can learn [20].

- *0 Hidden Layers:* Capable of representing linearly separable decision boundaries.
- *1 Hidden Layer:* Capable of approximating any function that can produce a continuous mapping from one finite space to another.
- *2 Hidden Layers:* Capable of representing any arbitrary decision boundary.
- *3+ Hidden Layers:* Capable of performing automatic feature engineering to learn complex representations.

For simple datasets, there is never a need to go beyond 2 hidden layers. However, when it comes to complex datasets like those belonging to NLP, computer vision (CV) and time-series problems, it is very common to have networks as deep as 100 layers (hence the term *deep* learning (DL)). There is no single rule to decide the optimum number of neurons in each hidden layer. Having too few neurons may underfit and having too many may overfit the training dataset. As a rule of thumb, the number of neurons can be kept between the size of the input and the output layers or less than twice the size of the input layer [20].

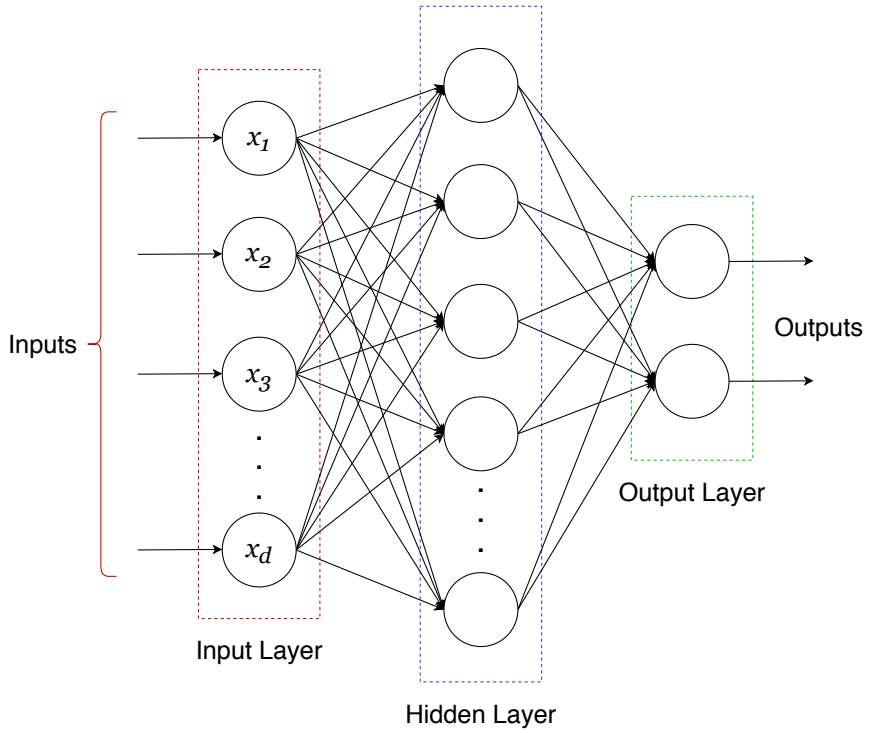

Figure 2.6 Architecture of a multilayer perceptron having a single hidden layer.

An MLP network is generally trained using a backward error propagation technique called backpropagation [21].

2. **Regression-Based:** The output variable in this case is continuous in nature. Think of examples like predicting the price of a house based on its features, predicting the electrical load in a building for the next hour, etc. The most common regression technique is the linear regression (LR) which is used to model the relationship between a dependent variable (which is to be predicted) and one or more independent variables (the features used in predicting the dependent variable). For a single independent variable, the algorithm is referred to as simple linear regression (SLR) which learns a straight line as formulated in (2.7).

$$y = mx + b \tag{2.7}$$

Here, y is the dependent variable and x is the independent variable. The slope m (weights) and the constant b (bias) are what the algorithm attempts to optimize in the training phase. Typically, the mean squared error (MSE) cost function is used in SLR which is minimized using a technique called gradient descent. Other popular regression techniques include polynomial regression, support vector regression (SVR), DT regression, RF regression, MLP regression, etc.

2.2.2 Unsupervised Learning

Unsupervised learning algorithms are used to discover patterns in unlabelled data (hence the name *unsupervised* learning). Based on the type of analysis that needs to be done on the data, unsupervised learning algorithms can be categorised into two categories:

1. **Dimensionality Reduction:** The algorithms under this umbrella are used for reducing the dimensionality or the number of features of the dataset. There are several reasons to do this: model simplification (less features correspond to a simpler model), faster train and test times, mitigating the curse of dimensionality, visualizing data by bringing it down to 3D or 2D, etc. Such a reduction can be achieved in two ways: feature selection and feature extraction. Feature selection-based algorithms attempt to reduce the data dimension by choosing a subset of the original features that are most important to the application at hand (the importance can be judged by employing some classification metric like accuracy). Naturally, such techniques require a labelled dataset. Algorithms like DT and RF have in-built feature selectors. Feature extraction techniques, on the other hand, do not need a labelled dataset. These algorithms project the features of the original dataset into a new reduced feature space. These new extracted features are, of course, completely different from the original features.

   Principal component analysis (PCA) is a popular feature extraction technique in ML and has also been employed in the upcoming chapters. In PCA, a linear mapping of data from the original space to a lower dimensional space is performed in a way that maximizes the variance of data in the lower dimensional space. The features of the lower dimensional space are called the principal components. The first principal component exhibits the largest variance, then comes the second principal component and so on. Note that before performing PCA, the dataset needs to be scaled (each feature must have zero mean and unit variance). An in-depth mathematical review of PCA can be found in [22]. Other popular unsupervised dimensionality reduction techniques include truncated-SVD (singular value decomposition), t-distributed stochastic neighbor embedding (t-SNE) and non-negative matrix factorization (NMF).

2. **Clustering:** Clustering is the process of segregating data points into groups or clusters such that the data points belonging to the same cluster are more similar to each other than to the data points belonging to other clusters. Clustering is used in a plethora of applications like grouping news articles, recommender systems, anomaly detection, medical sequence analysis, etc. Since the data points are not labelled, similarity between data points is judged using distance and similarity functions like Minkowski distance, Euclidean distance, cosine distance, Mahalanobis distance, Jaccard similarity, Hamming distance, etc. Popular clustering algorithms along with their type are listed below. Since the focus of this work is not on clustering, only a summary on the workings of these algorithms is provided. Detailed information regarding these and other clustering algorithms can be found in [23].

- **Partition-Based:** These algorithms form non-overlapping clusters by iteratively selecting cluster centers and computing distances between the data points and these cluster centers. Examples of this type of clustering include k-means and k-medoids.
- **Density-Based:** These algorithms form clusters out of regions having high density of data points. Examples of this type of clustering include DBSCAN and OPTICS.
- **Hierarchy-Based:** These algorithms construct a hierarchical relationship among the data points for clustering. Examples of this type of clustering include BIRCH, CURE and Chameleon.
- **Grid-Based:** These algorithms discretize the original data space into a grid structure for clustering. Examples of this type of clustering include CLIQUE and STING.
- **Graph-Based:** These algorithms take the data points as nodes in a graph. Examples of this type of clustering include MST and CLICK.

2.2.3 Reinforcement Learning

Both the supervised and unsupervised learning algorithms take a data-centric approach to learning. Reinforcement learning (RL), on the other hand, defines an agent in an interactive simulated environment and enables it to learn through trial and error, using feedback from its own actions to become better generation after generation. Before moving on, let us define some key terms related to RL.

- **Environment:** This is the simulated world that the agent explores and trains in.
- **State:** The current situation of the agent. At any instant of time, the agent must be in one of the many states defined in the environment ($s \in S$).
- **Action:** The agent can take actions out of all the possible actions defined in the environment ($a \in A$) to change its state.
- **Reward:** Reward is the feedback (can be positive or negative) that the environment delivers based on the action.
- **Policy:** Guidelines that the agent adheres to on what action to take in a particular state so as to maximize the total rewards.

Let's take an example of the famous dinosaur game in Google Chrome (the one that shows up if one is not connected to the internet). Now, our aim is to make the dinosaur learn to play the game without giving it explicit programming instructions. Naturally, there is no dataset for this. The game world with all its defined rules and objects is the environment. The dinosaur is the agent that we need to train. The agent receives a negative reward (death) for colliding with either the bird or the cactus. The dinosaur agent can perform 3 actions: jump, duck and keep running. Positive reward is imparted just by staying alive in this ever-moving game. The agent will initially start with a random set of parameters so as to

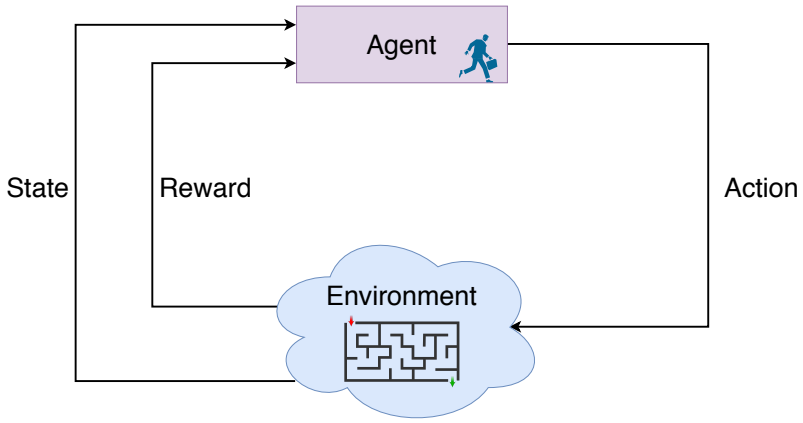

Figure 2.7 A basic block diagram showing the interrelation between the key components of RL.

explore the environment and gradually learn an optimal policy to play the game while maximizing the total reward. Figure 2.7 represents the basic idea behind RL. Since RL is beyond the scope of this work, interested readers can refer to [24] for more information on RL and its algorithms.

2.3 Use of ML in IoT Applications

With massive amounts of data being generated by the IoT devices, it is no surprise that a plethora of IoT systems require data analytics to function. With ML becoming mainstream in data analytics and IoT relying heavily on powerful edge and cloud machines, the union of ML and IoT was inevitable. A majority of the current IoT use cases come under the umbrella of smart cities.

- **Smart Home:** Smart homes aim to elevate the standards of living by simplifying and automating many aspects of human house life through a myriad of smart devices. There have been several approaches to non-intrusive and non-privacy-invasive *occupancy detection and estimation* in rooms which utilize environmental sensors and ML techniques like SVM and RF. Occupancy status of rooms can help in automating several appliances (turning them on or off, finding the perfect setting, etc.) like lights, fans and ACs (more details in Chapter 3). *Indoor localization* of occupants enables services like elderly and baby monitoring, intruder detection and appliance automation. Due to the lack of GPS signal in closed spaces, aspects of wireless signals from WiFi routers and BLE stations coupled with DL and RL have proven to be quite effective in inferring the positions of people [25, 26]. If a smart home is able to infer the activity that the occupants in the house are engaging in, a lot of doors to automation would open. Several works have shown positive results in *activity recognition* using techniques like recurrent neural networks (RNN) and long-short term memory (LSTM) on multi-sensor data [27]. DL-based CV models have been quite effective in fruit recognition systems that can be used in *smart refrigerators* [28]. There have been several strides made in indoor autonomous navigation using DL and CV for *assistive robots* that would

one day make us couch potatoes [29]. NLP is heavily used for HDI in *smart speakers* like Alexa and Google Home that have taken the world by storm.

- **Intelligent Transportation System (ITS):** With smart sensors installed on the roads coupled with the Internet of Vehicles (IoV) and GPS-enabled smartphones, new strides are being made in making commute easier than ever before. With more and more vehicles hitting the road every day, *traffic flow prediction* is a fundamental problem for implementing a city-wide smart transportation system. Various ML techniques like supervised regression, time-series and DL have been explored to tackle this challenge [30]. These flow predictions help in *route optimization* systems to *avoid congestion* [31]. Vehicles are not the only objects pertaining to traffic, people are too. *Crowd flow prediction* is therefore the other side of the coin that has been extensively researched upon by using several clustering and DL techniques [32, 33]. Smart parking systems have also come up that use techniques ranging from simple sensor-based WSNs to DL on camera-based systems [34, 35]. Traffic video analysis is another major challenge in smart transportation that can help *automate traffic lights*, *track objects* and *detect road accidents*. DL and RNN are heavily used in object detection and tracking in videos [36, 37]. DL has also been proven to be effective in detecting road accidents [38]. The crown jewel of smart transportation is *autonomous driving* vehicles on which a plethora of research is being done [39].

- **Smart Healthcare:** IoT has revolutionised the personal healthcare sector. Fitness trackers and smart watches that come fitted with sensors which can monitor various vital biological parameters, have become a norm in the society. These devices generate a lot of data that needs to be analysed before the end user can be apprised of the results. The authors in [40] applied DL and RNN to the movement data from wearable sensors to *predict the freezing gaits* in Parkinson patients. In [41], convolution neural network (CNN) was employed on accelerometer and heart rate data from a fitness band to accurately *predict energy expenditure* of the human body. The authors in [42] demonstrated an end-to-end DL-based approach in *classifying a broad range of arrhythmias* from electrocardiogram (ECG) data and interestingly, obtained an F1 score greater than that of human cardiologists. A small-footprint *pill recognition system* was built in [43] that used a multi-CNN model to classify images of prescription pills taken from smartphone or smartwatch cameras. A complete *medicine recognition system* was built in [44] which consisted of a smart pillbox having an NVIDIA Jetson TX2 to run DL models, an android app and a cloud-based management system.

- **Smart Gird:** Smart grid strives to improve the energy consumption of buildings. With smart meters making their way into homes, the energy provider can analyze data from millions of homes in real time. The consumers themselves can monitor the energy usage of their homes from anywhere in the world. Traditional ML algorithms like LR and SVR and DL algorithms like CNN and deep belief network (DBN) have shown positive results in *forecasting hourly load* at a state level [45, 46]. Various DL-based methods have also been employed for load forecasting at building level by using data from smart meters [47, 48]. Microgrids are often integrated with renewable

energy sources (RES) like wind and solar for local power generation. Because of the uncertainty in the production of energy by RES, *microgrid energy management* is a serious challenge that has seen a lot of contributions from the RL community [49]. With the energy producers and the consumers being connected thanks to the grid becoming smarter, a new paradigm of demand response has sprung up that aims to improve the reliability of the grid and make energy consumption cheaper by making the consumer shed load during peak hours. Several RL-based *dynamic pricing models for demand response* have been proposed for this [50]. Anything that becomes smart is bound to inherit the security flaws of computers and networks. Various DL and RL techniques have been explored to *detect cyber attacks on smart grids* [51, 52].

Chapter 3

Proposed ML-Based Occupancy Estimation Scheme Using Multivariate Sensor Nodes

In buildings, a large chunk of energy is spent on heating, ventilation and air conditioning (HVAC) and lighting systems. One way to optimize the usage of these systems is to automate them based on human occupancy. This chapter introduces a new occupancy estimation paradigm that uses a deployment of multiple heterogeneous non-intrusive sensor nodes in tandem with ML. A description of the sensor nodes and their physical deployments for collecting two new occupancy datasets is presented along with the insights gained from applying various preprocessing, feature engineering and ML techniques on these datasets. A section on the real-time implementation of the proposed system is also presented in this chapter.

3.1 Occupancy Estimation Using Environmental Sensors

Real-time occupancy information can give rise to intelligent HVAC and lighting systems in buildings which can not only conserve energy, but also provide better comfort to the occupants. Recent studies have demonstrated energy savings up to 30% in buildings in which the occupancy pattern was known [5]. With the rise in concerns regarding privacy, the research in recent years has moved towards the use of non-intrusive sensors like temperature, humidity, light, CO_2, air pressure, motion, sound, etc., for occupancy inference. A lot of research has been carried out in literature for occupancy detection, i.e., whether the room is occupied or not [53, 54, 55]. Although detection alone can help in improving energy savings, estimating also the precise number of occupants can make the system even more adaptive and energy efficient. As such, the focus of this chapter is on occupancy estimation.

3.1.1 Literature Review

There are quite a few papers on ML-based occupancy estimation [56, 57, 58, 59, 60]. In [56], an environmental sensor-based WSN, spanning multiple rooms of an office, was deployed with each room having only one sensor of each type. Three ML techniques namely hidden Markov model (HMM),

ANN and SVM were used on each individual datasets with HMM giving the best performance of 75% accuracy. In [57], a similar single-node-per-room sensor network was deployed and RBF neural network was used for classification. The authors reported a high accuracy of 88.74%. A similar setup was used in [59] wherein an RF classifier was employed. However, the paper binned the occupancy levels instead of giving a point estimate. In the three experiments listed above, the occupancy levels were described to be very dynamic indicating that all the class labels may not have equal number of data points. As such, F1 score and confusion matrix are more suitable performance metrics for such studies as compared to only the accuracy metric which is reported. A similar approach of binning was used in [60], which achieved a high accuracy using a convolutional deep bidirectional LSTM approach. The first work to incorporate multiple heterogeneous sensor nodes in a single space was [58]. The authors used a network of three sensor nodes and used extreme learning machines (ELM) to implement a wrapper model of feature selection. However, the primary focus of the paper was on various feature selection techniques for occupancy estimation. Only the accuracy metric was employed and that too on a binned dataset.

3.1.2 Proposed Occupancy Estimation Paradigm

The proposed occupancy estimation paradigm consists of a star network-based deployment of multiple multivariate sensor nodes in a single room. The intuition behind having multiple sensor nodes is to allow fusion of homogeneous and heterogeneous features among spatially-distant sensors. The performance comparison in terms of estimation accuracy and F1 score are carried out for various combinations of features, both homogeneous and heterogeneous, using ML techniques such as LDA, QDA, SVM, GNB and RF. Data preprocessing and feature engineering techniques to handle such a vast dataset and infer new features to boost the performance, such as CO_2 slope, are also discussed in the upcoming sections. Apart from supervised learning techniques, PCA is also employed to see how well a transformed but reduced feature set performs in comparison to the original but large feature set.

3.2 Experimental Setup for Occupancy Estimation

The experimental testbed for occupancy estimation was deployed in a $6m \times 5m$ room as shown in Figure 3.1. The room consists of 4 office desks, a big rear window with blinds and a self-closing glass door for entry and exit. The setup deployed in the room consisted of 7 sensor nodes and one edge node in a star configuration with the sensor nodes transmitting data to the edge periodically using wireless transceivers. No HVAC systems were in use while the dataset was being collected.

Six different types of non-intrusive sensors were used in this experiment: temperature, humidity, light, sound, CO_2 and digital passive infrared (PIR). The CO_2, sound and PIR sensors needed manual calibration. For the CO_2 sensor, zero-point calibration was manually done before its first use by keeping it in a clean environment for over 20 minutes and then pulling the calibration pin (HD pin) low for over 7s. The sound sensor is essentially a microphone with a variable-gain analog amplifier attached to it.

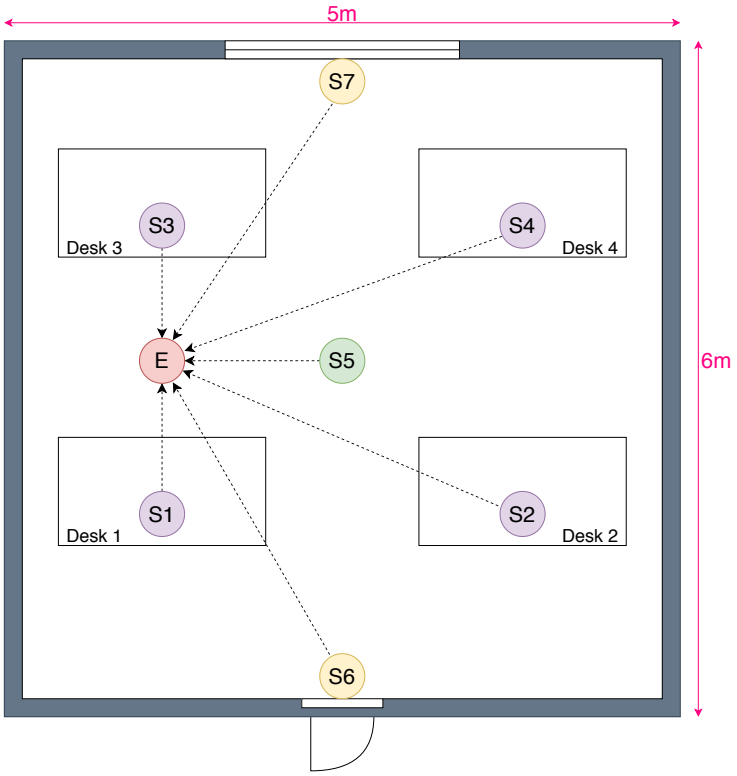

Figure 3.1 A star network-based data acquisition system deployed in a room.

Therefore, the output of this sensor is analog which is read by the microcontroller's ADC in volts. The potentiometer tied to the gain of the amplifier was adjusted to ensure the highest sensitivity. The PIR sensor has two trimpots: one to tweak the sensitivity and the other to tweak the time for which the output stays high after detecting motion. Both of these were adjusted to the highest values. Table 3.1 lists the accuracy, resolution and the communication protocol of each sensor used. As it is evident from Figure 3.1, sensor nodes S1-S4 were deployed at the desks where people sat (referred to as the desk nodes). Since there were multiple desks in the room, the desk nodes were planned to have low-cost sensors and therefore had temperature, humidity, light and sound sensors only. Node S5 had a CO_2

Sensor	Parameter	Communication Protocol	Resolution	Accuracy
DS18B20	Temperature	1-Wire	0.0625 °C	0.5 °C
DHT22	Humidity	Single-wire Serial	0.1 %	2 %
BH1750	Light	I2C	1 lux	1 lux
MAX4466	Sound	ADC	0.01 V	-
MH-Z14A	CO_2	UART	5 ppm	50 ppm
HC-SR501	Motion Detection	Digital	-	-

Table 3.1 Specifications of the sensors used in the occupancy estimation experiment.

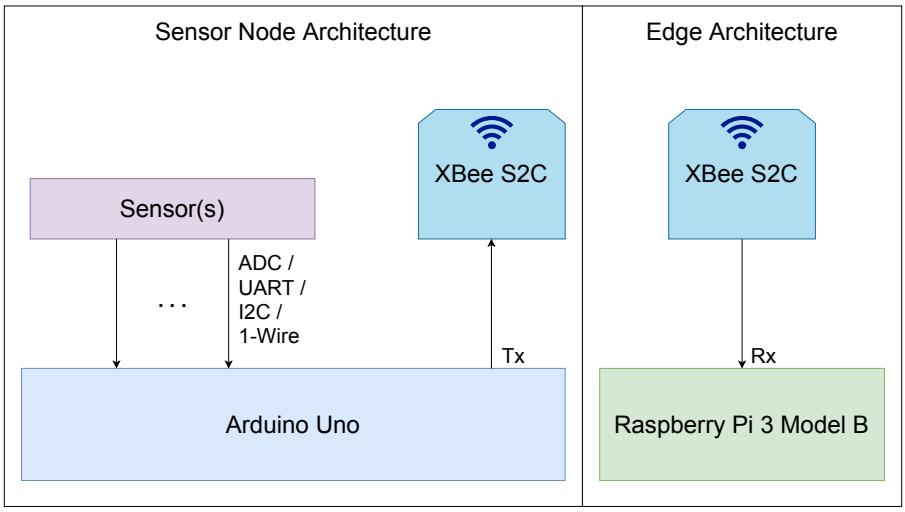

Figure 3.2 Block architecture of the sensor nodes and the edge.

sensor which was kept in the middle to get the best possible reading of the room. Nodes S6 and S7 only had one PIR sensor each and were deployed on the ceiling ledges at an angle that maximized the sensor's field of view for motion detection. It was later observed that among a pool of so many sensors, humidity did not contribute much to the final metrics and was therefore dropped from the dataset to keep the number of sensors as low as possible.

Figure 3.2 shows the block architecture of the sensor nodes as well as the edge. The Arduino Uno microcontroller board, present in each sensor node, was programmed to sample the data from the sensors and transmit it periodically (every 30s) to the edge via an XBee module. The temperature, light and CO_2 sensors were sampled only once in the 30s time frame since these quantities seldom change in such a short duration. The PIR and sound sensors, however, need constant polling or else the events of interest are lost. Since the output pin of the PIR sensor remains high for about 3s in repeat-trigger mode, the Arduino polled the PIR every 2.5s to check for the motion events. If even a single motion event was captured in the frame of 30s, a '1' was sent to the edge. For sound sensors, the algorithm churned out the maximum peak-to-peak voltage that was achieved in the time frame of 30s. The edge only had an XBee module for receiving data from the sensor nodes and appending it to the corresponding sensor node file along with the current time-stamp. The ground truth of the occupancy count in the room was noted manually. The occupancy of each table was also noted in order to observe different correlations among spatially-distant sensors. The data was collected for a period of 4 days in a controlled manner with the occupancy in the room varying between 0 and 3 people.

3.3 Data Preprocessing and Feature Engineering

A 10-hour slice from the collected dataset is shown in Figure 3.3. The CO_2 data is of S5, the PIR of S6 and the rest belong to S1. The juxtaposition of different sensory features and the occupancy count

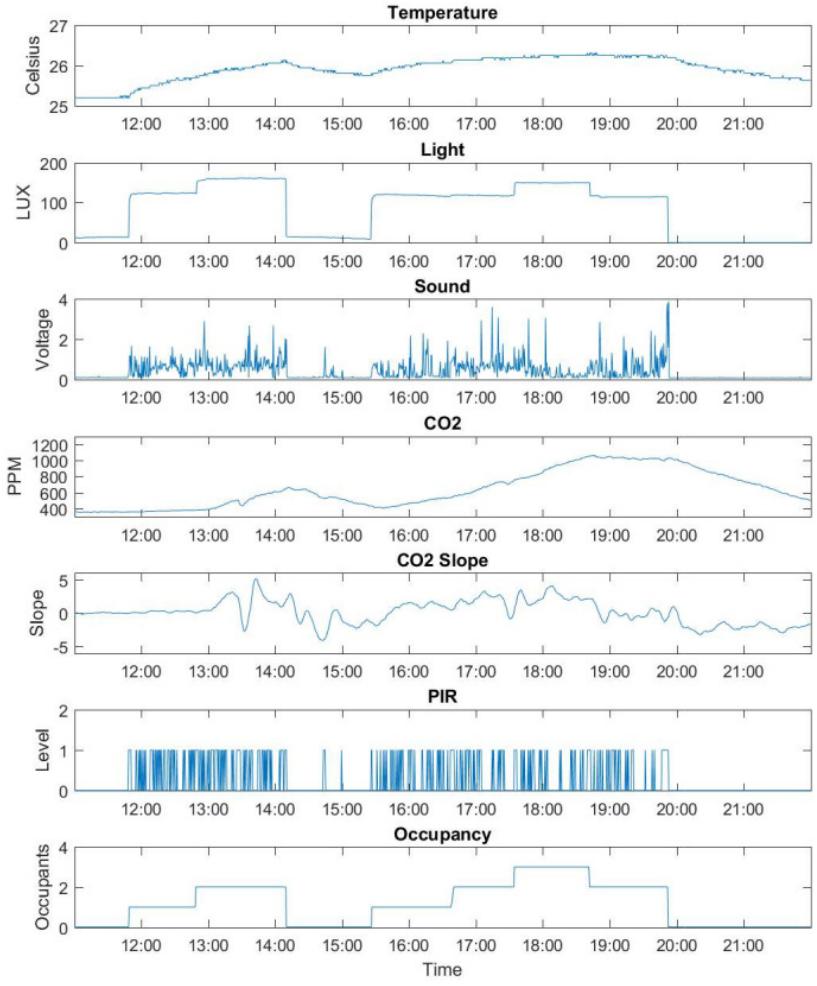

Figure 3.3 Data from a few representative sensors and the CO_2 slope feature for a period of about 10 hours.

reveals various correlations among them. After the data collection phase was over, the individual raw sensor node datasets had to be merged into a single dataset. The reasoning behind having a single dataset pertaining to all the multiple sensor nodes was so that the ML algorithms can exploit correlations among spatially distant homogeneous and heterogeneous sensors. The following tasks were performed in order to convert the raw data received from the sensor nodes into a usable dataset.

1. Even though each sensor node was programmed to transmit data every 30s, the data packets reached the edge at different times (a few seconds of variation was observed between the arrival times of different data packets). There were two main reasons for this discrepancy. Firstly, each sensor node acted independently and did not have equivalent start times and secondly, because of the CSMA/CA protocol of 802.15.4, the sensor node transmissions kept going out of sync. Since a single dataset was required, the time-stamps within a time window of 30s were merged into a common vector.

2. Feature vectors with missing data were deleted since there was no concrete model to approximate PIR and sound values from historical data. This resulted in fewer but more credible vectors in the dataset. In this experiment, only 8 such cases were observed. Missing data can be an issue for real-time systems because in a given time interval, if the data from even a single sensor is missing, the incomplete data vector cannot be propagated through the trained ML model.

3. The final dataset has over 10,000 points and 16 features, with each feature belonging to a particular sensor. After looking at the time-series plots of the sensors in Figure 3.3, it can be observed that the CO_2 data gives an excellent indication about the number of occupants in the room. However, it takes several minutes for the readings to rise or fall to a steady state. For example, just a few minutes before 19:00, there are 3 occupants in the room and the CO_2 readings reach a steady 1000 ppm. But when the room becomes empty an hour later, the CO_2 readings take around 2 hours to slowly come down to 400 ppm (standard empty-room value). Therefore, a new feature was derived in the form of slope of CO_2. This was calculated by fitting a linear regression in a window of 25 points at each instance and calculating the slope of the line. This parameter of 25 was obtained by trial and error with respect to the accuracy and F1 score metrics. The sign of this slope (negative/positive) gives an indication about whether the occupancy has decreased or increased in the room and the magnitude of this slope gives some idea of the count.

The collected dataset, like most real-world datasets, is skewed with more data points pertaining to the empty room (0 occupancy). Because of this, the F1 score metric and the confusion matrix are also reported in the results as the accuracy metric cannot be completely trusted in such situations. Macro F1 score, which calculates the metric for class labels separately, is more reliable than micro F1 score for unbalanced class labels [61] and has therefore been used.

3.4 Experiments and Results

The dataset was divided into various sets of homogeneous and heterogeneous features to understand the importance of different sensors when it comes to occupancy estimation. Six different ML algorithms namely LDA, QDA, linear SVM, SVM with RBF kernel, GNB and RF were applied on the feature sets. Metrics such as accuracy, F1 score and confusion matrix were evaluated by doing a 10-fold stratified cross-validation. Since the data is of time-series nature, data was not shuffled during the cross-validation process in order to avoid data points similar to the test data getting into the training data. In case of SVM, the training set features were normalized to have zero mean and unit variance since the algorithm is scale variant. The normalization constants from the training set were used to scale the testing set at each iteration of the cross-validation loop. The penalty hyperparameter was varied from 10^{-4} to 10^{4} for each feature set and the best metric value is reported. For RF, the number of trees in the forest was kept at 30 and the *min_samples_split* hyperparameter was tuned to ensure that the model didn't overfit on the training data. LDA, QDA and GNB do not require any hyperparameter tuning.

Feature Set	Metric	LDA	QDA	GNB	SVM (L)	SVM (R)	RF
Temp{1,2,3,4}	A	0.840	0.862	0.823	0.866	**0.895**	0.869
	F1	0.479	0.590	0.479	0.554	**0.730**	0.657
Light{1,2,3,4}	A	0.973	0.919	0.881	**0.973**	0.973	0.972
	F1	0.928	0.854	0.674	**0.929**	0.927	0.925
Sound{1,2,3,4}	A	0.851	0.879	0.874	0.875	0.885	**0.887**
	F1	0.449	0.544	0.526	0.542	0.591	**0.601**
PIR{6,7}	A	**0.869**	0.869	0.869	0.870	0.870	0.870
	F1	**0.474**	0.474	0.474	0.466	0.460	0.460
CO_2	A	0.809	0.808	**0.808**	0.812	0.812	0.763
	F1	0.383	0.409	**0.411**	0.286	0.314	0.329
Slope	A	0.852	0.831	0.831	0.870	0.870	**0.876**
	F1	0.387	0.394	0.394	0.462	0.510	**0.564**
CO_2, Slope	A	0.891	0.867	0.824	0.890	**0.888**	0.873
	F1	0.556	0.590	0.454	0.592	**0.635**	0.559

Table 3.2 Accuracy and F1 score obtained by using ML algorithms on homogeneous feature sets.

The experiments described above were performed using the *scikit-learn* library [62]. In the first phase of experimentation with supervised learning, only homogeneous fusion of sensor data was done. This implies that the sensors of same type but present on different spatially-distant sensor nodes were combined to assess their performance. This is tabulated in Table 3.2. The best result of each feature set is shown in bold. Clearly, the best performance is given by linear SVM on light data. Other feature sets do not even come close to this result. This heavy bias towards light can be explained from the observation that in most cases, people tend to switch on the lights above their desks when they arrive and turn them off when they leave. However, when this assumption fails or is not valid, relying only on light sensors for occupancy estimation would leave the system vulnerable to false positives. For example, a person may leave the room with lights on. Also, in systems where the lights in a room need to be controlled based on the occupancy status, light cannot be taken as a feature. Non-linear SVM on temperature set takes the second place with a high F1 score of 0.730. For the digital PIR couple set, only four states are possible: '00', '01', '10' and '11', and these do not correlate with the occupancy count in general. For example, a single occupant in the room can generate all of the above states. That is why the F1 score on the PIR feature set does not exceed 0.5. It can be observed that the proposed CO_2 slope feature shows promising results. It performs better than CO_2 for most algorithms and the performance improves significantly when both the features are combined. In general, since the environmental conditions and constraints can vary significantly from one room to the other, heterogeneous fusion of data on top of homogeneous fusion seems like a more reliable option intuitively. For example, rooms with adaptive ACs may not exhibit deviations in temperature with variations in occupancy, or rooms with air ventilation systems may not allow CO_2 to increase beyond a certain point, or rooms in which people sit still or sleep for long periods may fool the PIR sensors, etc.

Feature Set	Metric	LDA	QDA	GNB	SVM (L)	SVM (R)	RF
Temp{1,2,3,4}, CO_2, Slope	A	0.903	0.881	0.815	0.904	**0.912**	0.894
	F1	0.653	0.680	0.479	0.667	**0.750**	0.684
Temp{1,2,3,4}, CO_2, Slope, Sound{1,2,3,4}	A	0.920	0.908	0.863	**0.933**	0.924	0.918
	F1	0.735	0.749	0.628	**0.793**	0.782	0.731
Temp{1,2,3,4}, CO_2, Slope, Sound{1,2,3,4}, PIR{6,7}	A	0.922	0.910	0.873	**0.934**	0.924	0.919
	F1	0.737	0.748	0.643	**0.793**	0.780	0.734
Temp{1,2,3,4}, CO_2, Slope, Sound{1,2,3,4}, PIR{6,7}, Light{1,2,3,4}	A	0.980	0.957	0.932	0.982	**0.984**	0.978
	F1	0.946	0.911	0.817	0.948	**0.953**	0.933

Table 3.3 Accuracy and F1 score obtained by using ML algorithms on homogeneous and heterogeneous feature sets.

Heterogeneous fusion of data was done in the next phase of experimentation with supervised learning and is documented in Table 3.3. To generate these feature combinations, one sensor type at a time was added in a greedy fashion until the complete dataset was used. Light, however, was considered only in the end because of the reasons listed previously. The last row of Table 3.3 is the complete dataset and the row above that is the complete dataset devoid of light features. As expected, the complete dataset which includes all the sensors and the derived slope feature, performs the best at estimating the number of occupants accurately. SVM with RBF kernel gives the highest accuracy of 98.4% and an F1 score of 0.953. Other algorithms also exhibit similar performance except for GNB, which settles at an F1 score of 0.817. In the complete dataset devoid of all the light features, a good accuracy of 93.4% and a moderately high F1 score of 0.793 is achieved with linear SVM. This same F1 score is also exhibited by the row above, which is devoid of PIR sensors in addition to the light sensors. The confusion matrices for the best cases in the last and the second last rows are shown in Table 3.4.

	Predicted 0	Predicted 1	Predicted 2	Predicted 3
Actual 0	8117	43	41	27
Actual 1	104	336	19	0
Actual 2	65	48	502	133
Actual 3	21	7	154	512
Actual 0	8196	1	3	28
Actual 1	0	453	6	0
Actual 2	0	0	712	36
Actual 3	10	1	67	616

Table 3.4 Confusion matrix for the linear SVM case of the complete dataset devoid of light features (upper table) and for the SVM (RBF) case of the complete dataset (lower table).

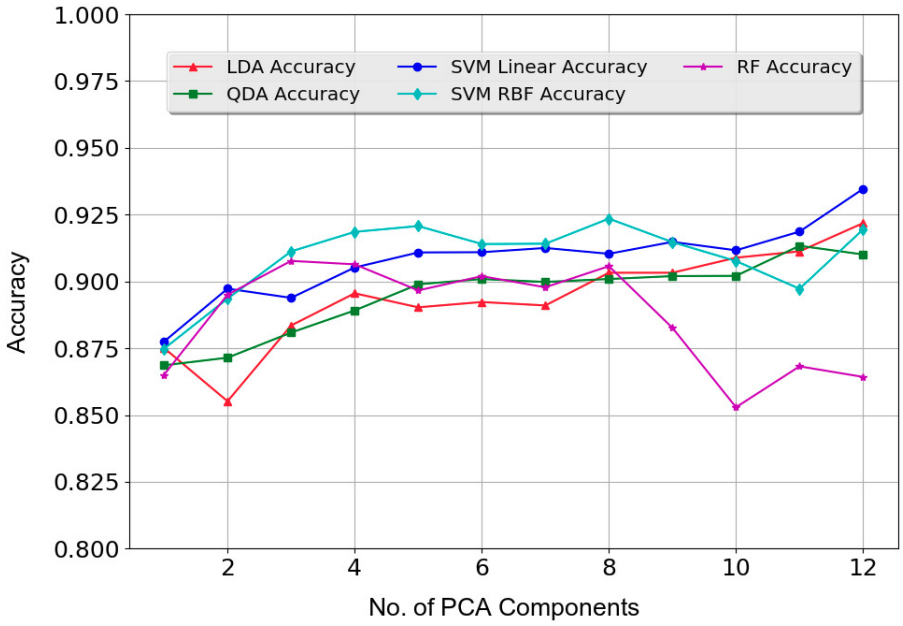

Figure 3.4 Accuracy of different ML models with respect to the number of PCA components.

SVM with RBF kernel performed better than linear SVM for most cases. The difference is significant when considering fewer features (rows 1, 3, 5, 6, 7 of Table 3.2 and row 1 of Table 3.3) while the F1 score for both are comparable when the number of features are high (rows 2, 3, 4 of Table 3.3) suggesting that the data points in the n-dimensional space to be linearly separable to some extent. In the latter case, linear SVM can be considered as a better choice than SVM with a non-linear kernel because of faster computations. It can be said that the Gaussian assumption for the features of the dataset holds well as LDA and QDA perform only a tad bit poorer than SVM. RF also performs equally good for most cases. GNB performs the worst out of all the ML algorithms employed on the feature sets having more than one feature. This is because GNB assumes all the features to be independent and does not exploit correlations among them which the other algorithms do.

The best accuracy, as described above, is achieved when all the 16 features of the dataset are considered. Having a dataset of high dimensionality puts strain on complex algorithms like non-linear SVM and RF when it comes to training and testing. It is well known that the lesser the number of features, the faster the ML algorithms perform. Therefore, in an attempt to reduce the number of features of the dataset, PCA is employed. The reduced dataset is then fitted with the aforementioned supervised models for performance evaluation. Since the four light features were already giving an excellent accuracy and F1 score (second row of Table 3.2), PCA was performed on the complete dataset without light in an attempt to reduce the dimensions from 12 to a significantly smaller value. Figure 3.4 and Figure 3.5 show the variation in accuracy and F1 score of all the ML models respectively (except for GNB which was exhibiting a very poor performance since the beginning) with the number of PCA components.

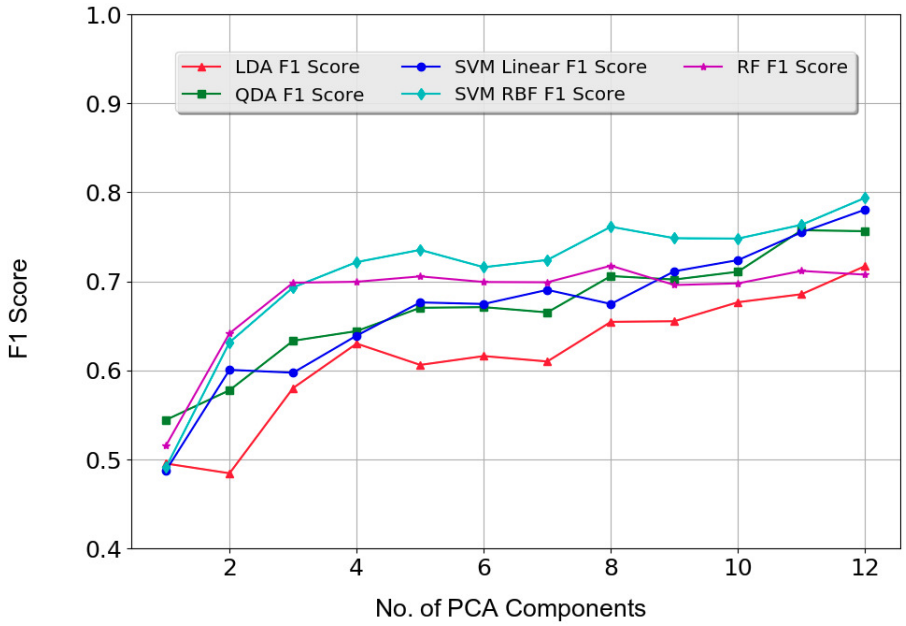

Figure 3.5 F1 score of different ML models with respect to the number of PCA components.

Even with as low as four components, SVM with RBF kernel gives an accuracy of around 92% and a moderate F1 score of 0.72. Linear SVM also shows a similar performance.

3.5 Real-Time Occupancy Estimation System

The results depicted in the previous section were obtained by experimenting offline on the collected dataset. A real-time implementation of the occupancy estimation system was also experimented with. To convert the deployment of Figure 3.1 into a real-time system, the only change that needed to be done was at the edge. Instead of just storing the data from the sensor nodes into separate files, the real-time algorithm at the edge waits for a time interval of 30s in order to collect data from all the sensor nodes and then precisely merges them into a single vector in the same format defined in the training dataset. Based on the results of the previous section (last row of Table 3.3), SVM (RBF) was trained on the complete dataset and stored at the edge to provide inference in real time. If, however, even a single data point was found missing in the time frame of 30s, the entire vector was dropped by the edge as missing data is not handled by the algorithm due to the reasons discussed in the data preprocessing section. Algorithm 1 depicts the algorithmic flow followed by the edge to make real-time occupancy predictions. The code for this was written in Python3.

Environmental quantities such as temperature and CO_2 take some time to react to the occupancy stimulus. On the other hand, light, sound and motion sensors swiftly capture this stimulus. Nevertheless, because of the slow reaction of the first two sensors, the overall system takes around 2-3 minutes after the change in the occupancy status before accurate and steady predictions are made. This small delay

Algorithm 1: Real-time occupancy estimation algorithm at the edge.

```
svm_rbf = SVM(C=0.1, kernel='rbf');
svm_rbf.train(complete_dataset, occupancy_labels);
while True do
    SensorList = [1, 2, 3, 4, 5, 6, 7];
    node_vectors = {};
    while SensorList != empty AND time < 30s do
        packet = ReadSerialXBEE();
        nodeID, data_vector = Split(packet);
        if nodeID == CO2_nodeID then
            slope = Calc_Slope(data_vector));
            data_vector.append(slope);
        node_vectors[nodeID] = data_vector;
        SensorList.delete(nodeID);
    if SensorList is empty then
        frame = Merge(node_vectors);
        label = svm_rbf.predict(frame);
        print(label);
    else
        print("Missed Something....Resetting!");
```

is not a deal breaker for most use cases considering that no intrusive or privacy-invasive sensor is in use. Three snapshots pertaining to zero, one and two occupancy of the real-time occupancy estimation system is shown in Figure 3.6. A Raspberry Pi 3 with an 8MP Camera v2 module was deployed on the ledge above the door to stream the live video feed of the room. It can be observed in the first image that when the system starts, the occupancy level erroneously offshoots to 3 but then settles down to a steady state of 0. A case of missing data point can also be observed in the same image. The 0 to 1 transition goes smoothly without any offshoots. This scenario can be observed in the second image. The 1 to 2 transition, as depicted in the third image, makes one erroneous offshoot to 3 but quickly settles down to the correct value of 2.

3.6 Estimating Occupants in Large Workspaces

This work is a direct extension of the previously discussed occupancy estimation scheme. The previous work cemented a foundation of using multiple multivariate sensor nodes for occupancy estimation in small rooms for a handful of people. The results were very positive, so much so that a real-time system was also implemented. This demanded a study on large workspaces with a head count of at least a dozen. For this, a new sensor network was deployed in a workspace with the data collection period of over 3 months. Naturally, manually noting the occupancy ground truth was not a feasible option for an experiment of such a duration and thus, a DL-based camera setup was also deployed along with

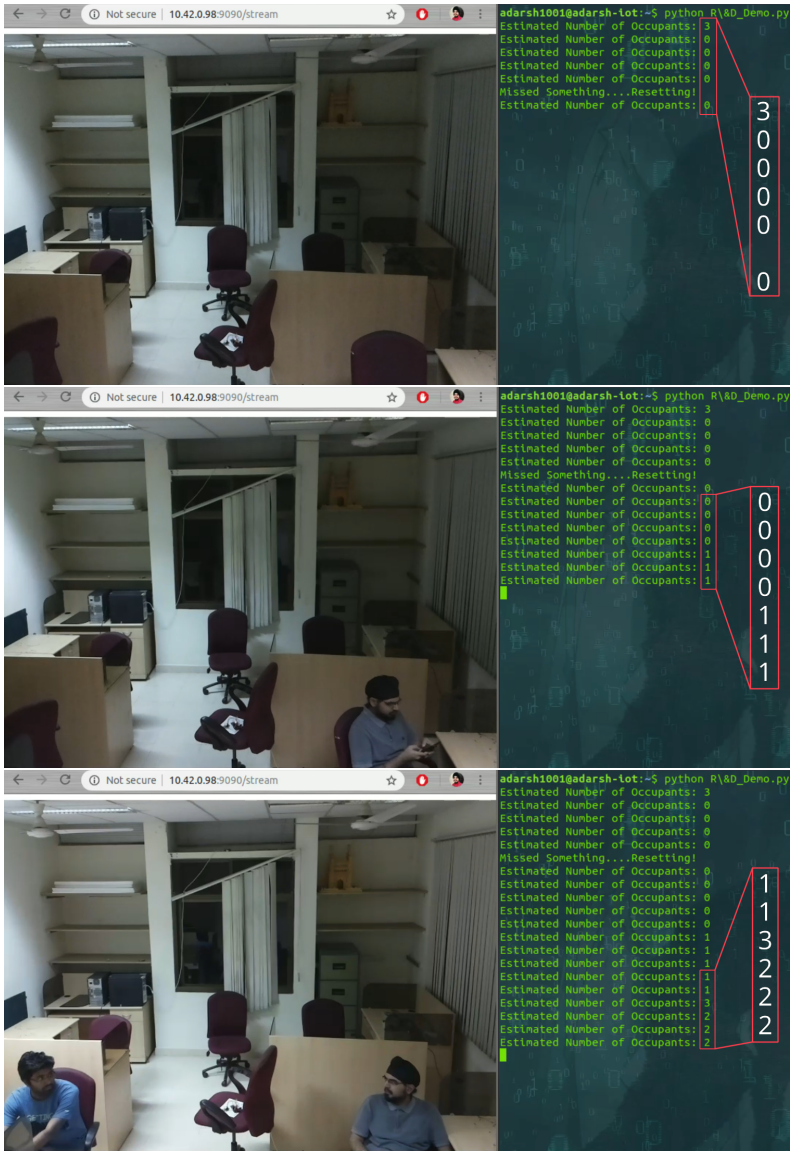

Figure 3.6 Occupancy snapshots pertaining to 0, 1 and 2 occupancy in the real-time occupancy estimation system.

the sensor network to take note of the number of occupants. Therefore, the data collection phase this time around was completely automated and defied the human-in-the-loop constraint. The results and the problems associated with such a large scale occupancy experiment is discussed in the upcoming sections.

3.6.1 Experimental Setup

Taking inspiration from the previous work, three different types of sensor nodes were deployed in the workspace as shown in Figure 3.7.

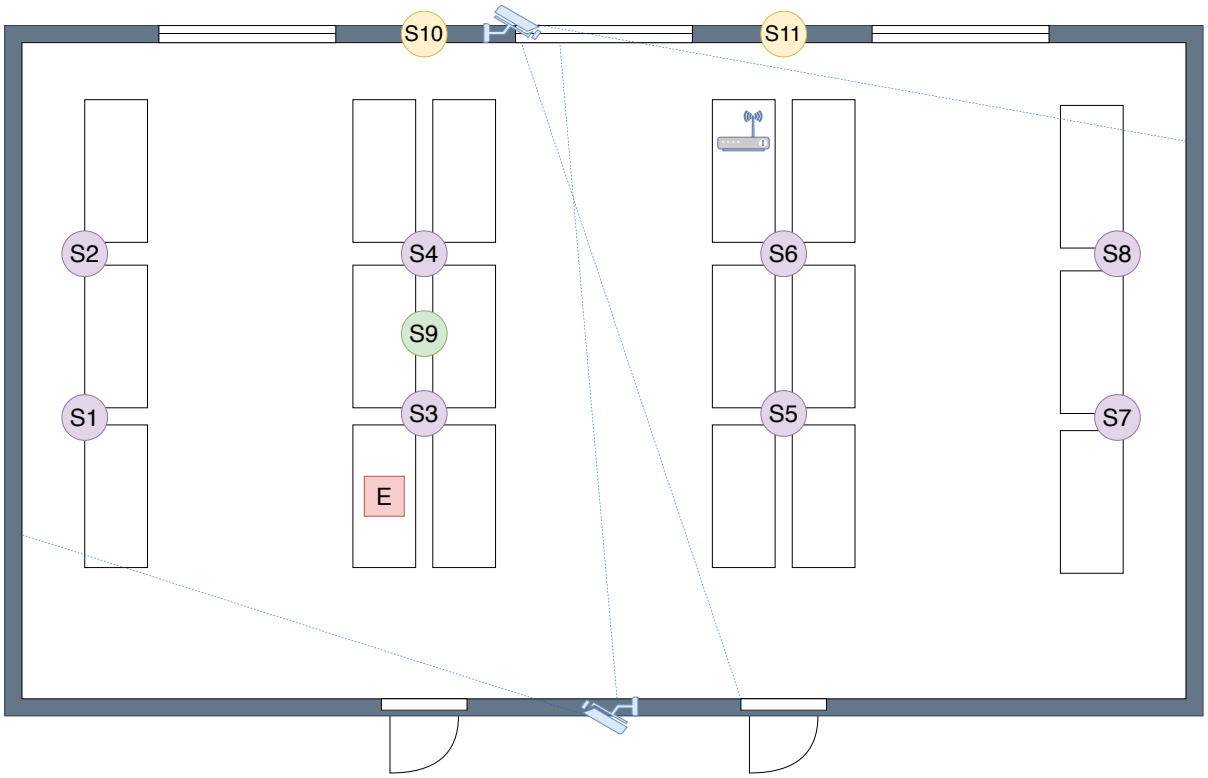

Figure 3.7 A WiFi-based automated occupancy estimation setup having multiple multivariate sensor nodes and two cameras.

1. S1-S8: Temperature, humidity, light and sound.
2. S9: CO_2 sensor
3. S10-S11: Digital PIR motion sensor

This time around, instead of having Arduino + XBee-based sensor nodes, a single NodeMCU was used. NodeMCU is an open source IoT platform which comes with an inbuilt WiFi chip. Not to mention that it costs as much as an Arduino Uno but since there is no need to pair an expensive XBee module for wireless connectivity, the cost of each sensor node goes down considerably. Two Raspberry Pi 3 + Camera (Pi-cam) setups were deployed on the room ledges at strategic angles to capture the entire seating arrangement of the workspace. This time, the edge was upgraded to a full computer with an NVIDIA graphics card in order to allow DL operations to run swiftly. All the sensor nodes and the Pi-cams were assigned different TCP/IP ports and were programmed to periodically dump their data (every 30s) to the WiFi router. The router, in turn, was programmed to forward the data packets to the edge computer which ran Python3-based socket programming scripts, one for each socket. Whenever a new packet arrived at the computer from any one of the sensor nodes, it was time-stamped and appended to its corresponding .csv file. This is identical to what was done in the previous occupancy estimation experiment. However, the images received from the Pi-cams were first propagated through a YOLOv3-

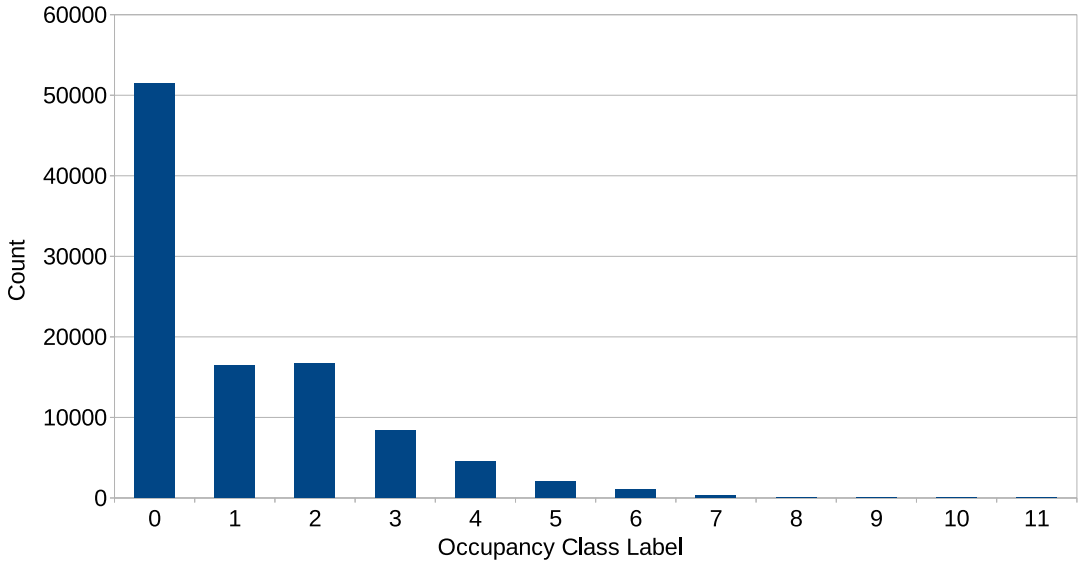

Figure 3.8 Class distribution of the large-workspace occupancy dataset.

based object detector to detect and return the count of humans in each image. This occupancy count was then time-stamped and stored in a separate ground truth file. YOLOv3 is a state-of-the-art DL-based object detection network. Prior to the start of the data collection phase, a manual examination of around 1000 images and YOLOv3 predictions on them were done which revealed that for human count estimation in our lab, the full YOLOv3 model exhibits an accuracy of around 90%. Of those 10% incorrect cases, in around 95% of them, the predictions were off by just a single occupant. This entire automated setup for data collection was made to run for a period of 3 months. The workspace conditions (turning on/off ACs, fans, windows) were not controlled or influenced in any way and nor were the count of the occupants. This posed some major hurdles later during the evaluation of the dataset which are described in the upcoming sections.

3.6.2 Dataset Description

The raw datasets (recall that each sensor node had a separate .csv file) that were collected over a period of 3 months had a lot of missing and erroneous data values. After cleaning those out, a dataset merging algorithm, similar to the previous one, was applied on the individual datasets to create a single master dataset. The master dataset consists of 101,176 data points spanning 45 features. Out of these 45 features, 35 belong to one of the sensors in the sensor network and the rest are derived features: CO_2 slope, eight temperature slopes and weekday (binary feature in which '1' depicts weekday and '0' depicts weekend). The ground truth of the occupants range from 0-11. Figure 3.8 shows the class label distribution of the dataset. Clearly, the dataset is heavily imbalanced. The maximum number of data points or feature vectors belong to the 0-occupancy label (51,517) whereas the 10 and 11 occupancy labels just have 8 and 2 feature vectors respectively. In fact, all occupancy count labels after 4 are very

small as compared to the first three occupancy labels. Therefore, experimentation with class label binning and data sampling techniques are also carried out. Since the dimensionality of the dataset is quite high (nominal convention says that if $d > 20$, one is likely suffering from the curse of dimensionality), a PCA reduced dataset is also explored in the experiments. The elbow-method on the PCA variance plot gives the appropriate number of components as 5 but for redundancy, 15 are chosen.

3.6.3 Experiments and Results

As mentioned before, the master dataset has 101,176 points with 45 features and 12 class labels (0-11 occupancy). Taking inspiration from our previous experiments, five ML algorithms namely LDA, QDA, SVM (linear), SVM (RBF) and RF were employed on the dataset. Since the dataset this time around was much more complex, MLP was also used with two hidden layers. In the no PCA cases, these two layers had 40 and 20 neurons respectively whereas in the PCA cases, these layers had 15 and 15 neurons respectively. The train and test data were split into 75:25 ratio respectively with stratified folds without shuffling. Stratification helps to preserve the quantities of all the class labels in the train and test datasets. Since the dataset is highly imbalanced, macro F1 score is reported along with accuracy.

The first row of Table 3.5 reports the performance of the ML algorithms with and without PCA when all the 12 class labels are used as it is. Clearly, the F1 score is abysmal. With no more options left in the feature engineering department, class labels were binned according to two schemes. In the first

Experiment	PCA?	Metric	SVM (L)	LDA	QDA	RF	SVM (R)	MLP
All 12 labels	No	A	0.301	0.467	0.336	**0.511**	0.176	0.231
		F1	0.090	0.123	0.096	**0.141**	0.069	0.094
	Yes	A	0.079	0.317	0.277	**0.487**	0.195	0.339
		F1	0.041	0.073	0.076	**0.131**	0.071	0.106
First Binning	No	A	0.100	0.487	0.364	**0.491**	0.276	0.227
		F1	0.073	0.250	0.185	**0.261**	0.133	0.164
	Yes	A	0.109	0.312	0.318	**0.462**	0.296	0.204
		F1	0.094	0.139	0.164	**0.219**	0.131	0.118
Second Binning	No	A	0.215	0.569	0.402	**0.607**	0.341	0.473
		F1	0.160	0.420	0.295	**0.485**	0.204	0.388
	Yes	A	0.295	0.420	0.319	**0.555**	0.226	0.371
		F1	0.219	0.294	0.218	**0.443**	0.168	0.301
Sampling	No	A	0.086	0.391	0.360	**0.425**	0.270	0.363
		F1	0.077	0.262	0.227	**0.277**	0.154	0.247
	Yes	A	0.286	0.283	0.308	**0.409**	0.173	0.133
		F1	0.217	0.163	0.186	**0.242**	0.122	0.130

Table 3.5 Accuracy and F1 score of various ML algorithms under different experiments pertaining to the large-workspace occupancy dataset.

binning scheme, six class labels were used: 0, 1, 2, 3-4, 5-7, 8-11. In the second scheme, four class labels were used: 0, 1-2, 3-5, 6-11. These numbers were chosen as per Figure 3.8 with the aim to increase the representation of each class. The results of the first binning experiment are jotted down in the second row of Table 3.5. The F1 score is nearly double than before for most of the algorithms but is still very less. The result of the second binning experiment is even better. RF gives the best accuracy of 60.7% and an F1 score of 0.485. However, considering that 45 features are being used to predict just 4 classes, this is not good at all. One problem is that even with binning, the dataset is still imbalanced. There are some undersampling and oversampling techniques that can be applied in situations like this one. To keep the results of this sampling experiment comparable to all the others, only the training set was sampled and not the testing set. Using random undersampling, the 0, 1 and 2-occupancy classes were first brought down to around 10,000 data points. Note that Tomek link-based undersampling was also experimented with but it didn't undersample enough data points. Classes 3 and 4 were then combined which made this combined class reach the 10,000 mark. Since the classes after 4 have very few representative points, all of them were combined into one label. Therefore, the binning scheme applied here is: 0, 1, 2, 3-4, 5-11. The least proportioned class (5-11) was then oversampled using the SMOTE algorithm to make the number of data vectors equal to the others. These steps made the dataset balanced. However, sampling the dataset (especially oversampling) sometimes skews the results of the experiment. Downsampling provides loss of information. The results of this experiment are represented in the fourth row of Table 3.5. Clearly, no improvement is observed. For all the experiments, RF outshines all the other algorithms. Both the SVMs did not converge for most of the non-PCA experiments making their performance inferior to the other algorithms.

Several techniques were applied on the newly collected occupancy dataset. However, the results are clearly not as good as expected. The dataset being highly imbalanced is one reason for this. A deeper study of the predicted class labels revealed that the higher occupancy classes were not even being predicted by the ML models. Although many techniques were applied to counter this, the best F1 score that was achieved was 0.485 and that too when only four class labels were used. Another major reason for the shortcomings of this experiment was that the environment of the workspace in which the data collection occurred, was left completely uncontrolled. There are two ACs, several fans and three windows in the workspace. During the data collection phase that lasted three months, a time-stamped note of the status of these factors was not jotted down. Because of this, the correlations among different features were not as apparent as Figure 3.3. For example, in later experiments, it was observed that when the ACs were powered on, the temperature sensors initially exhibited a decrease in temperature irrespective of the occupancy count and after settling to a steady state, did not show correlations with occupancy. Therefore, if one knows the status of ACs, one can train the model to put more emphasis on sensors other than temperature. The same can be said about CO_2 and windows. Had a note of what appliance is being used and when been kept, a deeper analysis would have been possible.

Chapter 4

Proposed ML-Based Transmission Reduction Scheme for Application-Specific IoT Networks

Reducing the amount of wireless data transmissions in constrained battery-powered sensor nodes is an effective way of prolonging their lifetime. This chapter introduces an ML-based scheme for reducing the number of data transmissions in wireless sensor nodes deployed as part of some application-specific IoT network. Although several data transmission reduction schemes have been proposed in the past, this is the first work to incorporate ML for the same. An in-depth analysis, both theoretical and experimental, of five supervised ML algorithms in the context of memory and computationally constrained microcontrollers is presented in the subsequent sections. The proposed data transmission reduction scheme is experimentally validated for occupancy applications and compared to the Shewhart-based data reduction scheme both in terms of the accuracy and F1 metrics and the number of transmissions.

4.1 Analysis of ML Algorithms for Constrained Devices

Supervised ML algorithms require a substantial amount of memory and computing resources in the training phase, far greater than what a typical microcontroller can provide. However, post training, many of these algorithms boil down to simple parameters that require standard arithmetic and logical operations for inference on unseen data. It is feasible to run the inference operation of such algorithms on microcontrollers in real time. Therefore, after training the ML model on a resource-abundant machine (which may be present at the edge or the cloud), the trained model parameters can be offloaded to the sensor nodes to run the inference in real time.

Embedded ML has been explored in the past for specific use cases and applications [63, 64, 65] but not in the context of data transmission reduction. Also, there is a dearth of a generic study that compares the run times and memory constraints of different supervised ML algorithms in the context of microcontrollers. Five traditional supervised ML algorithms with specific insights into offloading the trained model parameters and running the inference on a microcontroller are discussed in the following sections. The scope of this research is limited to binary classification.

4.1.1 Linear Discriminant Analysis (LDA)

As discussed in Chapter 2, LDA is a linear classifier that assumes the class conditional distribution of the data, $P(\mathbf{X} = \mathbf{x}|Y = k)$, where $\mathbf{X}$ represents the feature vectors of the training set and Y denotes some class k out of the set of classes $\{1, ..., K\}$, as a multivariate Gaussian:

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)} \tag{4.1}$$

To recap, the d in the above equation signifies the dimension or the number of features that the data has, $\boldsymbol{\mu}_k$ is the mean vector of the data under class k and $\mathbf{\Sigma}$ is the covariance matrix which is common for all the classes. For a new sensed data vector $\mathbf{x}_s$, predictions are made using Bayes' rule

$$P(Y = k|\mathbf{X} = \mathbf{x}_s) = \frac{f_k(\mathbf{x}_s)P(y = k)}{\sum_{i=1}^{K} f_i(\mathbf{x}_s)P(y = i)} \tag{4.2}$$

In the training phase of LDA, class means $\boldsymbol{\mu}_k$, covariance matrix $\mathbf{\Sigma}$ and class priors $P(Y = k)$ are learnt from the training data. As mentioned before, the training is done in a resource-abundant machine either at the edge or the cloud. In the *scikit-learn*'s implementation of LDA, post training, one can get these values from the *means_*, *covariance_* and *priors_* attributes respectively. For inference, the class k that maximizes the above equation (note that the denominator is same for all the classes) is chosen as the predicted class. So, for binary classification ($Y \in \{0, 1\}$), inference is made using the rule

$$P(Y = 1|\mathbf{X} = \mathbf{x}_s) \underset{0}{\overset{1}{\gtrless}} P(Y = 0|\mathbf{X} = \mathbf{x}_s) \tag{4.3}$$

Putting (4.1) and (4.2) in (4.3), we get

$$f_1(\mathbf{x}_s)\Pi_1 \underset{0}{\overset{1}{\gtrless}} f_0(\mathbf{x}_s)\Pi_0 \tag{4.4}$$

$$e^{-\frac{1}{2}(\mathbf{x}_s-\boldsymbol{\mu}_1)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_1)}\Pi_1 \underset{0}{\overset{1}{\gtrless}} e^{-\frac{1}{2}(\mathbf{x}_s-\boldsymbol{\mu}_0)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_0)}\Pi_0 \tag{4.5}$$

Here, Π_i denotes the class prior probability $P(Y = i)$. Taking log on both sides and rearranging,

$$-\frac{1}{2}(\mathbf{x}_s - \boldsymbol{\mu}_1)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_s - \boldsymbol{\mu}_1) \underset{0}{\overset{1}{\gtrless}} -\frac{1}{2}(\mathbf{x}_s - \boldsymbol{\mu}_0)^T\mathbf{\Sigma}^{-1}(\mathbf{x}_s - \boldsymbol{\mu}_0) + ln\frac{\Pi_0}{\Pi_1} \tag{4.6}$$

Expanding (4.6) and simplifying, we get

$$-\frac{1}{2}\boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \mathbf{x}_s^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0 - \mathbf{x}_s^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0 \underset{0}{\overset{1}{\gtrless}} ln\frac{\Pi_0}{\Pi_1} \tag{4.7}$$

$$\mathbf{x}_s^T\mathbf{\Sigma}^{-1}[\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0] \underset{0}{\overset{1}{\gtrless}} ln\frac{\Pi_0}{\Pi_1} - \frac{1}{2}(\boldsymbol{\mu}_0^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^T\mathbf{\Sigma}^{-1}\boldsymbol{\mu}_1) \tag{4.8}$$

Equation (4.8) can be simply written as

$$\mathbf{x}_s^T.\mathbf{w} \underset{0}{\overset{1}{\gtrless}} c \tag{4.9}$$

where $c = ln\frac{P(Y=0)}{P(Y=1)} - \frac{1}{2}(\boldsymbol{\mu}_0^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1)$ is a 4-byte floating-point constant and $\mathbf{w} = \boldsymbol{\Sigma}^{-1}[\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0]$ is a constant $d \times 1$ floating-point vector (or array in programming notation). Both of these constants can be calculated from the aforementioned formulations post the training phase and then offloaded to the microcontroller to enable it to run the inference (4.9) in real time. Time complexity-wise, d floating-point multiplications, $d-1$ floating-point additions and only one comparison is required for inference.

4.1.2 Quadratic Discriminant Analysis (QDA)

QDA is a quadratic decision boundary classifier that differs from LDA only in the aspect that the covariance matrix $\boldsymbol{\Sigma}_k$ is calculated separately for each class. Therefore, the class conditional distribution of the data in this case is formulated as

$$f_k(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_k|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)} \tag{4.10}$$

In the *scikit-learn*'s implementation of QDA, post training, one can get $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$ and $P(Y=k)$ from the *means_*, *covariance_* and *priors_* attributes respectively. As before, for any new sensed data vector $\mathbf{x}_s$, predictions are made using Bayes' rule as formulated in (4.2) and the inference for binary classification is made using (4.3). Putting (4.10) and (4.2) in (4.3), we get

$$\frac{e^{-\frac{1}{2}(\mathbf{x}_s-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_1)}}{|\boldsymbol{\Sigma}_1|^{1/2}}\Pi_1 \underset{0}{\overset{1}{\gtrless}} \frac{e^{-\frac{1}{2}(\mathbf{x}_s-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_0^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_0)}}{|\boldsymbol{\Sigma}_0|^{1/2}}\Pi_0 \tag{4.11}$$

Taking log on both sides and rearranging,

$$-\frac{1}{2}(\mathbf{x}_s-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_1) \underset{0}{\overset{1}{\gtrless}} -\frac{1}{2}(\mathbf{x}_s-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_0^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_0) + ln\frac{\Pi_0|\boldsymbol{\Sigma}_1|^{\frac{1}{2}}}{\Pi_1|\boldsymbol{\Sigma}_0|^{\frac{1}{2}}} \tag{4.12}$$

$$(\mathbf{x}_s-\boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_0^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_0) - (\mathbf{x}_s-\boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}_1^{-1}(\mathbf{x}_s-\boldsymbol{\mu}_1) \underset{0}{\overset{1}{\gtrless}} 2ln\frac{\Pi_0|\boldsymbol{\Sigma}_1|^{\frac{1}{2}}}{\Pi_1|\boldsymbol{\Sigma}_0|^{\frac{1}{2}}} \tag{4.13}$$

Expanding (4.13) and simplifying, we get

$$\mathbf{x}_s^T\left[\boldsymbol{\Sigma}_0^{-1} - \boldsymbol{\Sigma}_1^{-1}\right]\mathbf{x}_s + \mathbf{x}_s^T\left[2\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - 2\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0\right] \underset{0}{\overset{1}{\gtrless}} 2ln\frac{\Pi_0|\boldsymbol{\Sigma}_1|^{\frac{1}{2}}}{\Pi_1|\boldsymbol{\Sigma}_0|^{\frac{1}{2}}} + \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0 \tag{4.14}$$

Equation (4.14) can be simply written as

$$\mathbf{x}_s^T\boldsymbol{\Sigma}_{diff}^{-1}\mathbf{x}_s + \mathbf{x}_s^T\mathbf{w} \underset{0}{\overset{1}{\gtrless}} c \tag{4.15}$$

where, $c = 2ln\frac{P(Y=0)|\boldsymbol{\Sigma}_1|^{\frac{1}{2}}}{P(Y=1)|\boldsymbol{\Sigma}_0|^{\frac{1}{2}}} + \boldsymbol{\mu}_1^T\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0^T\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0$ is a 4-byte floating-point constant, $\boldsymbol{\Sigma}_{diff}^{-1} = \boldsymbol{\Sigma}_0^{-1} - \boldsymbol{\Sigma}_1^{-1}$ is a $d \times d$ constant floating-point matrix and $\mathbf{w} = 2\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\mu}_1 - 2\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0$ is a $d \times 1$ constant floating-point vector. These constants can be calculated post training and then offloaded to the microcontroller.

Time complexity-wise, $(d^2 + 2d)$ multiplications, $(d^2 + d - 1)$ additions and one comparison is required for inference. Clearly, this is much more expensive than LDA both in terms of the memory requirement and the arithmetic operations but is easily doable for low dimensional data. As per the experiments done on Atmel ATmega328P (described in detail in the later sections), it is not possible to go beyond 17 dimensions with QDA since the microcontroller runs out of memory.

4.1.3 Gaussian Naive Bayes (GNB)

Recall that in GNB, all the d features of the dataset given the class label k are mutually independent. Let $\mathbf{x}_s$ be a d-dimensional data vector as before. Then, $\mathbf{x}_s = (x_1, x_2, ..., x_d)$. Each x_i is nothing but a quantity sensed or derived by the sensor node. Using Bayes' theorem,

$$P(Y = k|\mathbf{X} = \mathbf{x}_s = (x_1, ..., x_d)) = \frac{P(Y = k)P(x_1, ..., x_d|Y = k)}{P(x_1, ..., x_d)} \tag{4.16}$$

The classifier will choose the class k, that maximizes (4.16), as the inference for the given input x_s. Using the independence assumption of the features and the fact that the denominator in the above equation is same for all the classes, we can formulate a generic classifier as

$$\hat{k} = \arg\max_k P(Y = k) \prod_{i=1}^{d} P(x_i|Y = k) \tag{4.17}$$

For binary labels, the inference rule given in (4.17) boils down to

$$P(Y = 1) \prod_{i=1}^{d} P(x_i|Y = 1) \underset{0}{\overset{1}{\gtrless}} P(Y = 0) \prod_{j=1}^{d} P(x_j|Y = 0) \tag{4.18}$$

Under GNB, the distribution of the features given a class label is assumed to be Gaussian. Therefore,

$$P(x_i|Y = k) = \frac{1}{\sqrt{2\pi\sigma_{k,i}^2}} e^{-\frac{(x_i - \mu_{k,i})^2}{2\sigma_{k,i}^2}} \tag{4.19}$$

where $\sigma_{k,i}^2$ and $\mu_{k,i}$ denote the variance and the mean of feature i under class k respectively. In *scikit-learn*'s implementation of GNB, the feature variances, means and class priors $P(Y = k)$ can be taken from *sigma_*, *theta_* and *class_prior_* attributes respectively. Putting (4.19) in (4.18), we get

$$P(Y = 1).v_1.e^{-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i - \mu_{1,i})^2}{\sigma_{1,i}^2}} \underset{0}{\overset{1}{\gtrless}} P(Y = 0).v_0.e^{-\frac{1}{2}\sum_{j=1}^{d} \frac{(x_j - \mu_{0,j})^2}{\sigma_{0,j}^2}} \tag{4.20}$$

where, $v_1 = \prod_{i=1}^{d} \frac{1}{\sqrt{\sigma_{1,i}^2}}$ and $v_0 = \prod_{j=1}^{d} \frac{1}{\sqrt{\sigma_{0,j}^2}}$. Taking log on both sides and rearranging,

$$-\frac{1}{2}\sum_{i=1}^{d} \frac{(x_i - \mu_{1,i})^2}{\sigma_{1,i}^2} + \frac{1}{2}\sum_{j=1}^{d} \frac{(x_j - \mu_{0,j})^2}{\sigma_{0,j}^2} \underset{0}{\overset{1}{\gtrless}} ln\frac{P(y = 0).v_0}{P(y = 1).v_1} \tag{4.21}$$

On simplifying, we end up with the inference rule

$$\sum_{j=1}^{d} \rho_{0,j}^2 (x_j - \mu_{0,j})^2 - \sum_{i=1}^{d} \rho_{1,i}^2 (x_i - \mu_{1,i})^2 \underset{0}{\overset{1}{\gtrless}} c \tag{4.22}$$

where $c = 2ln \frac{P(y=0) \prod_{j=1}^{d} \frac{1}{\sqrt{\sigma_{0,j}^2}}}{P(y=1) \prod_{i=1}^{d} \frac{1}{\sqrt{\sigma_{1,i}^2}}}$ is a 4-byte floating-point constant and $\boldsymbol{\rho}_0^2 = \frac{1}{\boldsymbol{\sigma}_0^2}$ and $\boldsymbol{\rho}_1^2 = \frac{1}{\boldsymbol{\sigma}_1^2}$ are d-length floating-point arrays. The inverse of variances are taken to convert the division operations to multiplications since the latter is much more computationally efficient in microcontrollers. So, post training, we need to offload four d-length floating-point arrays: $\boldsymbol{\mu}_0$, $\boldsymbol{\mu}_1$, $\boldsymbol{\rho}_0^2$ and $\boldsymbol{\rho}_1^2$ and a floating-point constant c. Time complexity-wise, $4d$ multiplications, $4d-1$ additions/subtractions and one comparison is required for inference.

4.1.4 Support Vector Machine (SVM)

Recall that in SVM, for any new sensed data vector $\mathbf{x}_s$, we just need to check in which half of the separating hyperplane this vector lies in:

$$\mathbf{w}^T \bar{\mathbf{x}}_s \underset{0}{\overset{1}{\gtrless}} c \tag{4.23}$$

Here, w is the $d \times 1$ floating-point weight vector (coefficients of the hyperplane equation), c is the negative of the intercept of the hyperplane and $\bar{\mathbf{x}}_s$ is the standard-scaled transformed vector of $\mathbf{x}_s$ which is calculated as

$$\bar{x}_{s,i} = \frac{x_{s,i} - u_i}{s_i}, \ \ \forall i \in \{1, 2, ..., d\} \tag{4.24}$$

where $\mathbf{u}$ and $\mathbf{s}$ are the d-length floating-point arrays having means and standard deviations of the training samples respectively. As it was done in the GNB case, the more complex division operation is converted to multiplication by taking the inverse of $\mathbf{s}$ so that the standard-scaling transformation can be done by

$$\bar{x}_{s,i} = p_i(x_{s,i} - u_i), \ \ \forall i \in \{1, 2, ..., d\} \tag{4.25}$$

This scaling of data is necessary since the SVM algorithm is scale variant. Hence, apart from $\mathbf{w}$ and c, scaling parameter arrays $\mathbf{u}$ and $\mathbf{p}$ also need to be offloaded. During the training phase, the first step is training data standardization which is usually done using *scikit-learn*'s *StandardScaler* function. Once the training data is fitted and transformed, attributes *mean_* and *scale_* will give $\mathbf{u}$ and $\mathbf{s}$ respectively ($\mathbf{p}$ can be calculated from $\mathbf{s}$ by taking the inverse of each element). After linear SVM is trained on this scaled data, we can get $\mathbf{w}$ from the *coef_* attribute and c from the negative of *intercept_* attribute. Note that the inference operation of kernelized non-linear SVM is too complex for simple microcontrollers and that is why only linear SVM is explored. Time complexity-wise, $2d$ multiplications, $2d-1$ additions/subtractions and a single comparison is required for inference.

4.1.5 Decision Tree (DT)

Recall that a trained DT is an *if-else ladder* in the shape of a tree with branches feeding to test nodes and the leaves corresponding to one of the class labels. Unlike the algorithms listed above, only the logical comparison operation is needed for inference. Once trained, each root-to-leaf path of the decision tree can be represented as

$$\textit{if } \text{condition}_1 \textit{ and } \text{condition}_2 \textit{ ... and } \text{condition}_l \textit{ then } \text{class } k$$

Here, each condition is a node in the path from root to one of the leaves (assuming l nodes including the root and excluding the leaf). The class label k is the corresponding leaf node. This conversion has to be done for all the unique root-to-leaf paths in the trained DT and then offloaded to the microcontroller. This can be done manually by printing the trained DT model. Note that the number of root-to-leaf paths (or the number of *if* statements) would be equal to the number of leaves (N_{leaf}). Therefore, the inference memory and time complexity is dependent on the number of *if* statements and the average depth (D_{avg}) of the tree which is nothing but the average number of comparisons in a single *if* statement. Both of these parameters can be controlled during the training phase. In *scikit-learn*'s DT implementation, *max_depth* and *max_leaf_nodes* parameters can be specified to control the growth of the tree.

4.1.6 Complexities of Different ML Algorithms

The arithmetic time and space complexities of the five ML algorithms discussed in the earlier sections is tabulated in Table 4.1. Apart from DT, all other algorithms are directly dependent on the dimension of the data. In the DT case, as discussed in the previous section, only logical comparisons are needed for inference. The number of such comparisons can vary between 1 and $D_{avg}N_{leaf}$. The memory requirement of the DT inference algorithm jotted in the table is for the average case as the depth of each root-to-leaf path in a tree varies considerably making it impossible to come up with an exact number.

We can draw some clear theoretical conclusions from Table 4.1. Clearly, the inference operation of LDA is the fastest and the lightest among all others. SVM comes in second. Even though the inference formulations of both LDA and SVM are similar (Equation (4.9) and Equation (4.23)), SVM

Algorithm	Multiplications	Additions	Bytes Offloaded
LDA	d	$d-1$	$4d+4$
QDA	d^2+2d	d^2+d-1	$4d^2+4d+4$
GNB	$4d$	$4d-1$	$16d+4$
SVM	$2d$	$2d-1$	$12d+4$
DT	-	-	$4D_{avg}N_{leaf}$

Table 4.1 Arithmetic time and space complexities of the inference operation of five traditional ML algorithms.

requires extra space and computations as the sensed data vector needs to be scaled first. The heaviest algorithm in terms of the arithmetic and space requirements is clearly QDA which depends on the square of the data dimension. This grows much faster than the others when the dimension is increased. It is difficult to draw any conclusions for DT as it depends on completely different parameters. Higher data dimensionality does generally produce more massive trees but by how much is something that depends on the dataset. Moreover, one can always control the growth of the tree making the number of comparisons and the memory requirement for trees trained on the datasets of different dimensions virtually similar (performance may or may not take a hit).

Apart from the five supervised ML algorithms analysed above, k-nearest neighbor (KNN), RF and MLP are widely used in ML applications. However, their inference operations require heavy memory and computing resources making them not ideal to be run on constrained microcontrollers. For instance, in KNN, the training algorithm simply stores all the training points in the dataset and when testing on an unseen data point, it computes the distance to each training point and churns out the k-nearest ones. The datasets in IoT applications are often huge ranging from hundreds of megabytes to several gigabytes making KNN not ideal for devices having only a few kilobytes of storage. RF models can be offloaded to a microcontroller if the number of trees are small and the trees are not too deep and spread out (the $D_{avg}N_{leaf}$ factor would need to be controlled appropriately since there are multiple trees). These constraints, however, are contrary to why the ensemble techniques are used in the first place and it may so happen that a single bigger tree-based model may outperform multiple small trees-based ensemble model. MLP and neural networks pose a severe challenge to the computational and memory limits of even computers during training and testing. While it is possible to train a small neural network with only tens of nodes in each layer and offload the feedforward structure to the microcontroller, real-world datasets and IoT applications generally demand larger networks for higher accuracy.

4.1.7 ML on Atmel ATmega328P

So far, only a theoretical understanding of different ML algorithms and how they perform under the constraints that embedded devices exhibit has been achieved. This section showcases the performance of the aforementioned algorithms on an actual microcontroller. Figure 4.1 shows the execution times of LDA, QDA, SVM and GNB with respect to the dimensionality of the data as measured on Atmel ATmega328P. To get this, synthetic datasets of different dimensions ranging from 1 to 20 were first created. Then, each of these synthetic datasets were used to train the aforementioned ML algorithms and post training, the inference parameters were calculated by the procedures discussed in the previous sections. In the final step, the inference parameters were offloaded to the microcontroller to be run in real time. The execution times were calculated by leveraging the internal timer of ATmega328P which has a resolution of 4μs. DT has been omitted from the plot since it is not directly dependent on the data dimension. However, just for reference, for $d = 3$, an exemplar DT model took 48μs when trained with *min_samples_split* $= 50$. LDA and SVM, being simple linear classifiers, take the least amount of time for inference. As discussed before, LDA is a tad bit lighter as well as faster than SVM since the data

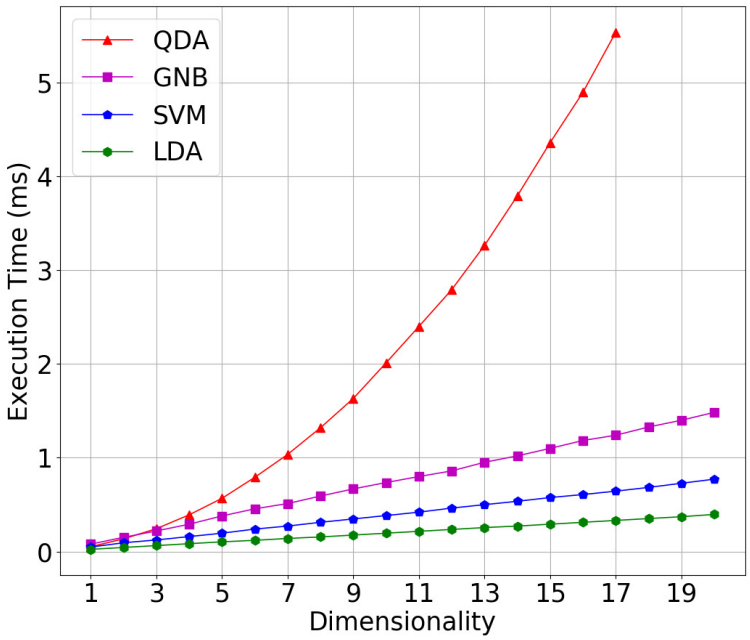

Figure 4.1 Execution times of QDA, GNB, SVM and LDA inference algorithms with respect to the data dimension as measured on Atmel ATmega328P.

points in case of LDA do not require scaling. QDA's quadratic dependence on the data dimension is clearly visible in the plot. Note that the execution times of the QDA inference doesn't exist for $d > 17$ since the ATmega328P runs out of memory beyond that.

4.2 Data Reduction Schemes for Constrained IoT Networks

Numerous IoT applications require WSNs to be deployed in constrained environments devoid of regular wired power supply. Naturally, such applications have boosted the usage of battery-powered and energy harvesting-based sensor nodes. These nodes comprise of a memory and computationally constrained low-power microcontroller, sensor(s) and a radio frequency transceiver. The radio transmission is known to be the biggest consumer of energy in WSNs [10]. Consequently, increasing the lifetime of these constrained wireless sensor nodes by decreasing the amount of data transmissions without compromising much on the data quality is an important area of research.

In a typical star WSN, the sensor nodes transmit data to the edge periodically while following a *sleep* $\rightarrow$ *wake up* $\rightarrow$ *sense* $\rightarrow$ *transmit* $\rightarrow$ *sleep* schedule. Without any data reduction scheme, the array comprising of sensor data $x[n]$, where n is the number of quantities sensed by the node, would be transmitted to the edge each time the sensor node wakes up. In this case, the data analytics part is taken care of by the edge or the cloud. In case where a data reduction scheme is employed, the decision of whether to initiate transmission or not is typically governed by a mathematical model involving the

current sensed array $x[n]$, the previous sensed array and/or the previous transmitted array. Several data reduction techniques have been explored in the literature.

4.2.1 Literature Review

The data reduction techniques for IoT explored in the literature can be divided into 2 main categories:

1. **Data Compression Techniques:** Data compression is the process of transforming the structure of the data so as to reduce the overall size. The process may be lossy or lossless. Several low-complexity data compression techniques have been explored for data reduction in WSNs [66]. However, even though these algorithms shorten the data packets, the transmission periodicity remains the same.

2. **Dual Prediction Schemes:** Since the sensors themselves are bounded by an accuracy threshold, some amount of error can generally be tolerated by the user. Hence, a majority of data reduction schemes for WSNs revolve around data prediction models that run on both the sensor node and the edge. The sensor node initiates transmission only when the predicted value exceeds the current sensed value by a predefined error threshold [11]. These schemes can be roughly divided into 2 categories:

   (a) **Adaptive Filter-Based Schemes:** In [67], the authors proposed a dual Kalman filter (DKF) for WSNs. The filter weights require prior statistical information about the phenomenon under measurement in the training phase and then the trained model is offloaded to both the sensor node and the edge to perform predictions in real time. A simpler and more popular adaptive filter approach that doesn't require prior information about the data was proposed by authors in [68]. Among different adaptive filter algorithms, least mean square (LMS) is widely used since it exhibits a lower computational cost and more flexibility [69, 70].

   (b) **Time-Series-Based Forecasting Schemes:** Time-series forecasting algorithms such as Shewhart change detection (naive algorithm), moving average (MA), exponential weighted moving average (EWMA) and autoregressive integrated moving average (ARIMA) have also been explored for data reduction in WSNs [71]. These algorithms model the historical data and predict future observations based on the learned model.

   The simplest time-series-based forecasting algorithm is the naive or the Shewhart change detection algorithm. In Shewhart, an error threshold is defined on each of the sensors and if the current value of a sensor exceeds or goes below the previously transmitted value by that error threshold, the data is transmitted. Therefore, if the edge doesn't receive any data from the sensor node in a particular interval, it assumes the previously received data to be true for this interval as well. Interestingly, the authors in [71] experimentally showed Shewhart to be more optimal than MA, EWMA and even ARIMA both in the number of transmissions and the root mean squared error (RMSE) metric for temperature data. Shewhart has also been

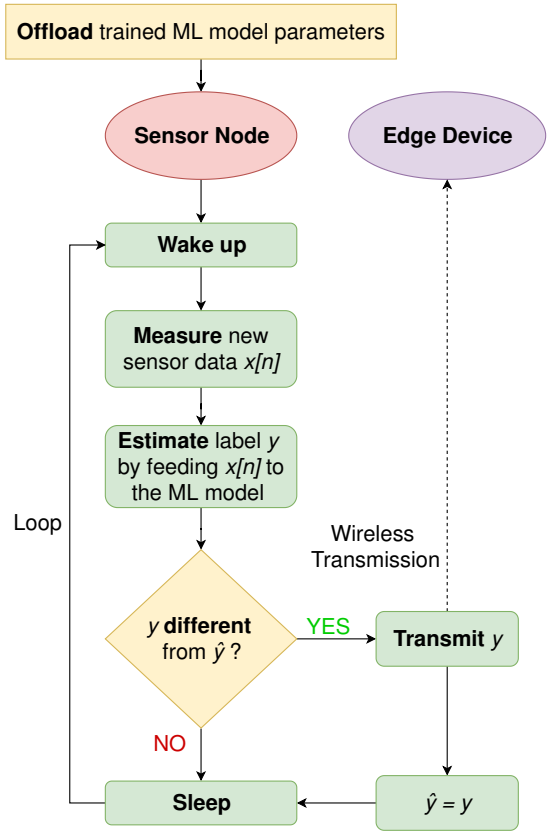

Figure 4.2 Algorithmic flowchart for data transmission reduction using ML on constrained sensor nodes.

shown to outperform LMS in [72, 73, 74]. Hence, Shewhart has been used in comparison with the proposed data transmission reduction scheme.

The data prediction algorithms explored in the WSN literature so far are all thresholding-based schemes that do not take the application into account when deciding the initiation of transmission. The authors in [75] recently pioneered the work on ML-based data importance calculation to reduce the number of transmissions for an IoT vehicle mobility use case. The data importance calculation in the work is based on feature selection using an RF model and not embedded ML per se. Also, their work is focused on decreasing the volume of data transmitted by resource-abundant IoT devices such as smart cars and smartphones and not on increasing the lifetime of constrained microcontroller-based sensor nodes.

4.2.2 Proposed ML-Based Data Transmission Reduction Scheme

Figure 4.2 shows the flow of the proposed algorithm. The initial step requires a trained ML classifier to be offloaded to the sensor node. This can be achieved via the techniques discussed in Section 4.1. Post that, the sensor node follows an infinite cycle of *sleep* $\rightarrow$ *wake up* $\rightarrow$ *sense* $\rightarrow$ *decide transmission* $\rightarrow$

sleep. Since a trained ML model is present at the sensor node itself, the sensor data $x[n]$ is propagated through the model to find the inference class label y here itself. Therefore, instead of transmitting the sensor data, the class label y can be transmitted. Let the previously transmitted label be $\hat{y}$. The inference y is only transmitted in case y differs from $\hat{y}$. In a nutshell, the transmission occurs only when the class label pertaining to the application changes. This way, the edge device always has the current scenario of the application at hand with the least amount of transmissions from the sensor node.

Note that since supervised ML is involved, annotated training data is obviously required in the training phase. Therefore, unlike the thresholding-based prediction algorithms that can function in any scenario, this approach requires a specific application out of the sensor network. Once trained, the trained model parameters can be offloaded to the sensor node(s) which can then run the inference in real time.

4.3 Reducing Transmissions for Occupancy Applications

The data transmission reduction scheme described in the earlier section is generic and can be adapted to many IoT use cases. For validation of the proposed scheme, one of the most explored applications in the IoT domain is undertaken: estimating the presence as well as the number of humans in a room using non-intrusive sensors.

4.3.1 Experimental Testbeds

Since the main objective now is to validate the proposed data transmission reduction scheme and not just perform simple occupancy experiments, occupancy detection is also incorporated along with estimation. The following two sections provide a description of the occupancy testbeds used for validating the proposed data transmission reduction scheme.

4.3.1.1 Occupancy Detection

A standard occupancy detection dataset was used for experimentation. The authors in [54] deployed a single sensor node consisting of Arduino Uno, an XBee wireless transceiver and sensors (temperature, humidity, light and CO_2) in a small 5.85m $\times$ 3.5m room in Mons, Belgium. The node was programmed to transmit the data samples to the recording station every 14s but this is averaged out for the corresponding minute in the final datasets. Three different datasets are provided by the authors: one for training and the other two for testing. The training dataset contains 8,143 points. The door was mostly kept closed when the room was occupied. The first testing dataset consists of 2,665 points, again, with the door mostly closed when occupied. The second testing dataset consists of 9,752 points with the door mostly kept open when occupied. The class label is, of course, binary: '0' for no occupants and '1' for at least one occupant. All the three datasets are imbalanced with more number of points corresponding to the 0 class and therefore, the F1 metric is also reported along with accuracy.

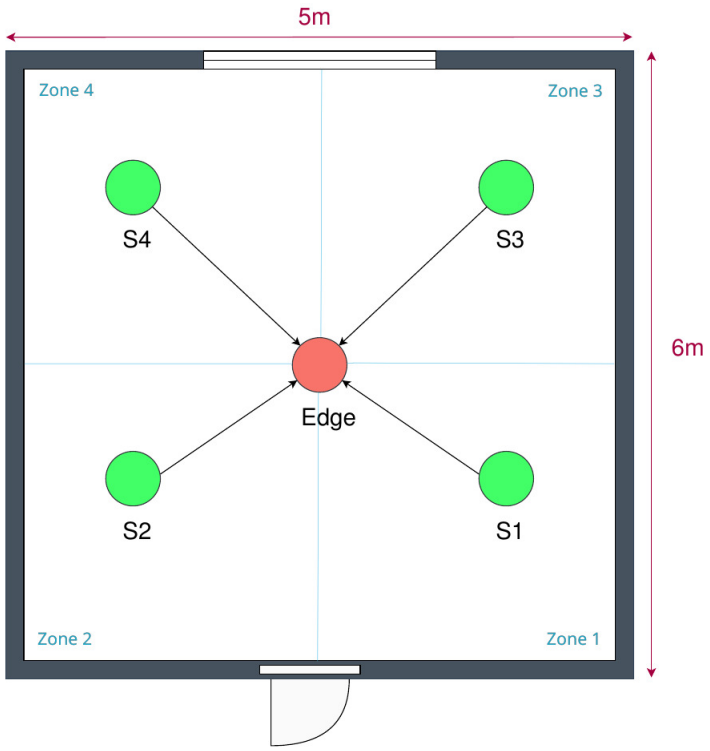

Figure 4.3 Testbed for data transmission reduction in room occupancy estimation.

Since the motivation for data transmission reduction is energy savings on constrained sensor nodes, it makes sense for the sensor nodes to follow a periodic sleep-wake up pattern. Consequently, the CO_2 sensor, which has a very high power consumption and requires continuous power supply (CO_2 sensor has a burn-in time of around 10-15 minutes before the readings can be used), is ignored. The other three features are used without any changes.

4.3.1.2 Occupancy Estimation

For occupancy estimation, the same small-room dataset which was used in the previous chapter is used for experimenting with data reduction. As before, the CO_2 feature is ignored along with the features pertaining to the PIR and the sound sensors that require continuous polling (sound and motion sensors would be useless if one were to use them only for a split second). In a nutshell, the estimation setup boils down to Figure 4.3 with each sensor node having three sensors: temperature, humidity and light. In other words, each individual sensor node dataset has 3 features and the merged complete dataset has 12 features. The room is virtually divided into four zones, each having only a single sensor node. Recall that during the formation of this dataset, the readings of each sensor node was recorded in a separate file and the manual recording of the room occupancy ground truth included recording the table or the zonal occupancy as well. The ground truth of the number of people range from 0-3 with at most one

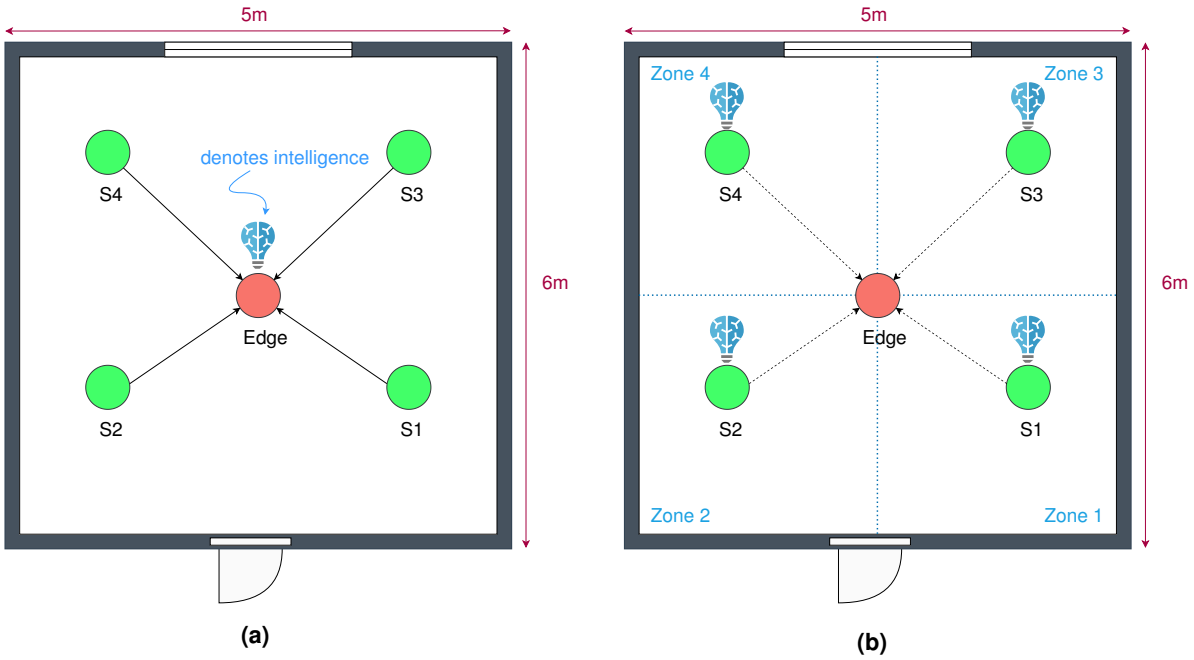

Figure 4.4 Occupancy estimation testbed depicting the sensor network. Figure 4.4(a) is a reference for experiments 1 and 2 wherein the intelligence resides at the edge. Figure 4.4(b) is a reference for experiment 3 wherein the sensor nodes use ML themselves.

person in each zone. The reason as to why the individual node datasets and the zonal occupancy count is necessary will be clear when the logistics of the experiments are explained in the upcoming section.

4.3.2 Logistics of the Experiments

To validate the proposed data reduction scheme, three different experiments were simulated on each of the occupancy datasets described in the previous sections:

1. **No Data Reduction (NDR):** This experiment simulates the original occupancy experiments in which the sensor node(s) follow no data reduction scheme and simply wake up periodically to transmit data and the edge does the real-time occupancy predictions using an ML model trained on the complete dataset. Figure 4.4(a) shows this scheme in action. Since ML is being run on the edge, there is no restriction on the complexity of the model. Therefore, apart from the five ML models described in Section 4.1, SVM with RBF kernel and RF are also employed. For occupancy detection, the training dataset was used for training and testing was done separately for each of the two testing datasets. For occupancy estimation, the accuracy and F1 score metrics were evaluated by doing a 10-fold stratified cross-validation on the complete occupancy estimation dataset (the one described in the previous section).

2. **Shewhart-Based Data Reduction (SDR):** This experiment simulates Shewhart-based data reduction scheme running on the sensor node(s). The data analytics part in this case is also handled by the edge and thus a complete dataset can be exploited for the occupancy estimation case as in the previous experiment. For the occupancy detection and estimation setups discussed previously, 3 error thresholds are needed, one for each sensor (e_t for temperature, e_h for humidity and e_l for light). The error thresholds for predictive schemes like Shewhart are usually derived from the sensor accuracy, sensor resolution (10 times the resolution in general) and mean of the data collected (2-5% in general). Based on this, we arrive at four different experiments for this case:

   (a) SDR1: $e_t = 0.5$, $e_h = 1.0$ and $e_l = 1$.

   (b) SDR2: $e_t = 0.5$, $e_h = 2.0$ and $e_l = 1$.

   (c) SDR3: $e_t = 0.5$, $e_h = 1.0$ and $e_l = 10$.

   (d) SDR4: $e_t = 0.5$, $e_h = 2.0$ and $e_l = 10$.

   Note that the minimum and the maximum error thresholds derived for each sensor is used. For the occupancy detection case, as before, the training dataset was used for training and the testing datasets for testing. Shewhart data reduction algorithm was simulated during the testing. The accuracy and F1 metrics for the occupancy estimation case were again calculated using a stratified 10-fold cross-validation approach on the complete dataset. In each iteration of the 10-fold loop, the testing set was simulated to run Shewhart separately for each sensor node and then combined into a single vector based on the outcome of the test for each of the testing data points.

3. **ML-Based Data Reduction (MLDR):** This experiment simulates the proposed ML-based data reduction scheme wherein the sensor node(s) run their own ML model and only transmit the inference when it differs from the previously transmitted inference. Hence, the data analytics part is taken care of by the sensor node(s) themselves. For the occupancy detection case which just has a single sensor node, this experiment is simple as the room occupancy is already in binary. For the occupancy estimation case, however, the model for each sensor node is trained on the data from that sensor node and its corresponding zonal occupancy count which is binary (Figure 4.4(b)). Therefore, each sensor node works independently without sharing data with other nodes making this a distributed detection problem. The edge, in this case, has the occupancy count of each zone which it simply sums up for all the zones to find the total room occupancy. In general, a single ML model pertaining to the complete dataset is more powerful than separate models for each sensor node as the latter is restricted to heterogeneous fusion of sensory data in a single location alone while the former also exploits patterns via homogeneous and heterogeneous fusion across spatially scattered sensor nodes. For example, in Figure 4.4(b), if there were an occupant in zone 1, then the sensor nodes in zones 2, 3 and 4 have been observed to be affected by it. The extent of this affect is, of course, dependent on the sensor type and the distance between the occupant and the sensor node.

To calculate the accuracy and F1 score for the occupancy detection case, we simply train the five ML models discussed in the previous sections on the training dataset and then test on the testing datasets. This essentially gives us the same result as experiment 1 (excluding SVM (RBF) and RF) albeit with less transmissions. For occupancy estimation, however, the result is quite different as compared to experiment 1 since separate models are used for each sensor node. To calculate the metrics for this case, stratified 10-fold cross-validation was applied on each of the sensor node models and the predictions were summed up to correspond to the room occupancy. The number of transmissions were calculated by counting the 0 to 1 and 1 to 0 transitions in the predicted class array of the testing sets.

4.3.3 Results and Comparisons

The ML experiments described in the previous sections were performed in Python3 in tandem with the *scikit-learn* library [62]. For the occupancy estimation case, macro F1 score is used instead of micro F1 score since the dataset is skewed with more points corresponding to the zero occupancy case than the others [61]. The penalty hyperparameter was fine-tuned in the case of SVM (RBF) and linear SVM for each of the experiments. For decision tree, the *min_samples_split* parameter was tuned to avoid overfitting on the training data. This parameter basically controls the amount of splitting in the decision tree. Random forest was used with 30 estimators and the tuned *min_samples_split* parameter. LDA, QDA and GNB do not require any hyperparameter tuning.

4.3.3.1 Occupancy Detection

Table 4.2 presents the accuracy and F1 score of each of the experiments mentioned in the previous section. Metrics for the two testing datasets were calculated separately. Since only a single sensor node is involved and the room occupancy is binary, NDR and MLDR experiments exhibit identical results since the edge and the sensor node run the inference with the exact same set of trained parameters. Of course, this is true only for the five algorithms that have low inference cost. SDR schemes also work very well on this dataset and exhibit almost identical results when compared to NDR with only minor variations present in SDR3 and SDR4. This can be attributed to the fact that the output class is a simple binary attribute and the occupancy detection for this dataset is heavily biased towards light which is mostly kept on during occupancy and off when the room is empty. The best result of each experiment is highlighted in the table. Metrics for testing set 2 are higher than testing set 1 which is in line with the results obtained by the authors. Linear SVM and LDA perform better than non-linear SVM suggesting a linear boundary to be a better fit for this dataset. RF either equals or outperforms DT which is expected because of its ensemble nature. GNB also gives results comparable to the best of each experiment suggesting that the feature independence assumptions holds well in this case.

Figure 4.5 shows the number of transmissions encountered by SDR and MLDR schemes for testing datasets 1 and 2 respectively. It can be clearly observed that all the ML-based data reduction schemes

Experiment	Test Set	Metric	SVM (R)	SVM (L)	LDA	QDA	GNB	DT	RF
NDR	1	A	0.975	0.979	**0.979**	0.977	0.977	0.979	0.979
		F1	0.967	0.971	**0.972**	0.970	0.970	0.971	0.971
	2	A	0.982	**0.992**	0.985	0.990	0.987	0.976	0.993
		F1	0.958	**0.981**	0.965	0.978	0.971	0.946	0.983
SDR1	1	A	0.975	0.979	**0.979**	0.977	0.977	0.979	0.979
		F1	0.967	0.971	**0.972**	0.970	0.970	0.971	0.971
	2	A	0.982	**0.992**	0.985	0.990	0.987	0.982	0.993
		F1	0.958	**0.981**	0.965	0.978	0.971	0.959	0.983
SDR2	1	A	0.975	0.979	**0.979**	0.977	0.977	0.979	0.979
		F1	0.967	0.971	**0.972**	0.970	0.970	0.971	0.971
	2	A	0.982	**0.992**	0.985	0.990	0.987	0.984	0.993
		F1	0.958	**0.981**	0.965	0.978	0.971	0.964	0.983
SDR3	1	A	0.976	0.979	**0.979**	0.977	0.977	0.979	0.979
		F1	0.967	0.971	**0.972**	0.970	0.970	0.971	0.971
	2	A	0.981	**0.992**	0.985	0.990	0.987	0.982	0.993
		F1	0.958	**0.981**	0.965	0.978	0.970	0.958	0.983
SDR4	1	A	0.976	0.979	**0.979**	0.977	0.977	0.979	0.979
		F1	0.967	0.971	**0.972**	0.970	0.970	0.971	0.971
	2	A	0.981	**0.992**	0.985	0.990	0.987	0.984	0.993
		F1	0.958	**0.981**	0.965	0.978	0.970	0.963	0.983
MLDR	1	A	N/A	0.978	**0.979**	0.977	0.977	0.979	N/A
		F1	N/A	0.971	**0.972**	0.970	0.970	0.971	N/A
	2	A	N/A	**0.992**	0.985	0.990	0.987	0.976	N/A
		F1	N/A	**0.981**	0.965	0.978	0.971	0.946	N/A

Table 4.2 Accuracy and F1 score of different ML algorithms under NDR, SDR and MLDR experiments for the occupancy detection dataset.

outperform Shewhart-based schemes by a huge margin. For testing set 1, the best case of just 7 transmissions is achieved by linear SVM, LDA and DT. This is a 25 times reduction in the number of transmissions as compared to the best case of Shewhart. For testing set 2, the best case of 39 transmissions is achieved by linear SVM. As compared to the best of Shewhart, 25 times reduction in the number of transmissions is achieved while exhibiting nearly identical accuracy and F1 metrics. Overall, in the best case, 99.71% reduction in the number of data transmissions is achieved for testing set 1 and 99.60% for testing set 2 while exhibiting identical performance metrics as the NDR case.

4.3.3.2 Occupancy Estimation

Table 4.3 presents the accuracy and the F1 score of each of the experiments described previously. The best score of each experiment is shown in bold. For the NDR and SDR cases where ML is applied at the edge on the aggregate of all the sensor values of all the sensor nodes, SVM with RBF kernel shines

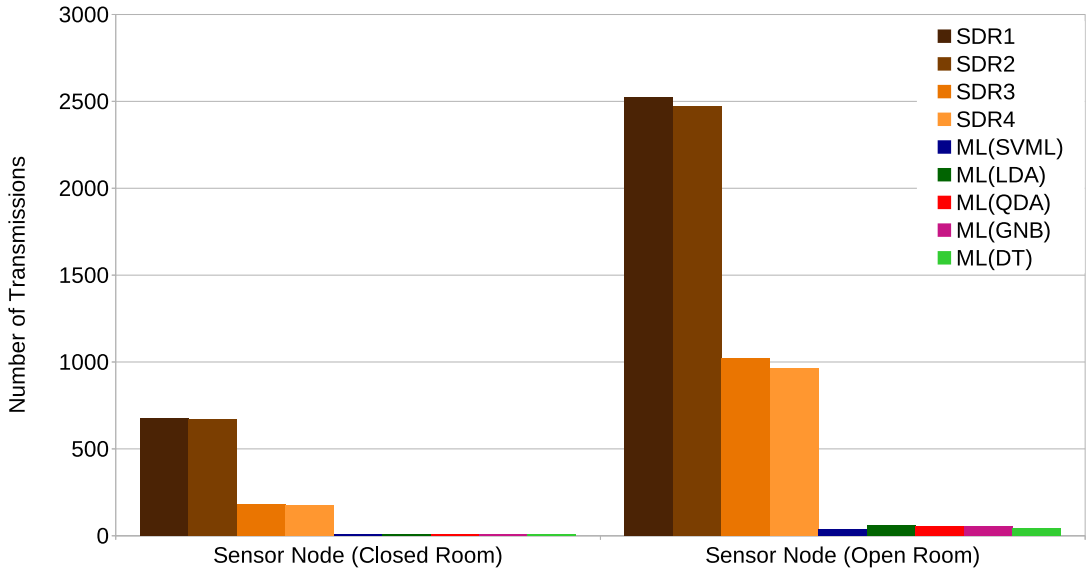

Figure 4.5 Number of transmissions encountered by the sensor node of the occupancy detection setup while running SDR and MLDR schemes on testing set 1 and 2.

out more than the other algorithms suggesting a non-linear decision boundary to be a better fit for this data. Even though both LDA and linear SVM are linear classifiers, LDA performs much better than the latter suggesting that the Gaussian assumption of the features holds well in this case. RF outperforms DT easily which is expected because of its ensemble nature. GNB performs the worst out of the seven algorithms which clearly implies that the feature independence assumption isn't valid for this particular application. As compared to the single sensor node case of occupancy detection in which it performed admirably, one can conclude that GNB fails to exploit correlations among sensory features of spatially distant sensor nodes. In the MLDR case, the best result is given by linear SVM and the numbers are comparable to the best of the other experiments. The hypothesis of the complete dataset performing better than the individual models, as discussed in the previous section, can be observed from the NDR and MLDR rows of the table. Except for the GNB case which does not exploit correlations among features, ML algorithms on the complete dataset either equal or outperform the same on distributed datasets.

Figure 4.6 shows the number of transmissions experienced by each sensor node while running different data reduction schemes. Note that S4 has been omitted from the plot as there was no occupancy in zone 4 during the data collection phase which made training ML models for it irrelevant. Clearly, all the five ML algorithms of the MLDR case outperform the four Shewhart-based schemes for each sensor node by a huge margin. S3, being closer to the window, faced the most variation in the light data (hence the spike in SDR1 and SDR2 cases for S3). Shewhart-based schemes are clearly affected by fast changing variations in data unlike the ML-based schemes. Linear SVM gives the least amount of transmissions (10, 11 and 9 out of 10,153 points for S1, S2 and S3). It may seem as if we can down-

Experiment	Metric	SVM (R)	SVM (L)	LDA	QDA	GNB	DT	RF
NDR	A	**0.977**	0.964	0.972	0.940	0.892	0.960	0.976
	F1	**0.939**	0.914	0.926	0.886	0.738	0.886	0.924
SDR1	A	**0.972**	0.962	0.972	0.852	0.872	0.921	0.969
	F1	**0.928**	0.909	0.926	0.815	0.717	0.825	0.907
SDR2	A	**0.972**	0.962	0.972	0.889	0.869	0.943	0.967
	F1	**0.931**	0.909	0.926	0.838	0.717	0.865	0.901
SDR3	A	0.970	0.960	**0.972**	0.786	0.866	0.905	0.964
	F1	0.922	0.903	**0.924**	0.781	0.721	0.807	0.891
SDR4	A	**0.969**	0.960	0.967	0.804	0.872	0.925	0.964
	F1	**0.922**	0.901	0.901	0.777	0.728	0.830	0.895
MLDR	A	N/A	**0.968**	0.965	0.933	0.915	0.890	N/A
	F1	N/A	**0.916**	0.907	0.812	0.759	0.743	N/A

Table 4.3 Accuracy and F1 score of different ML algorithms under NDR, SDR and MLDR experiments for the occupancy estimation dataset.

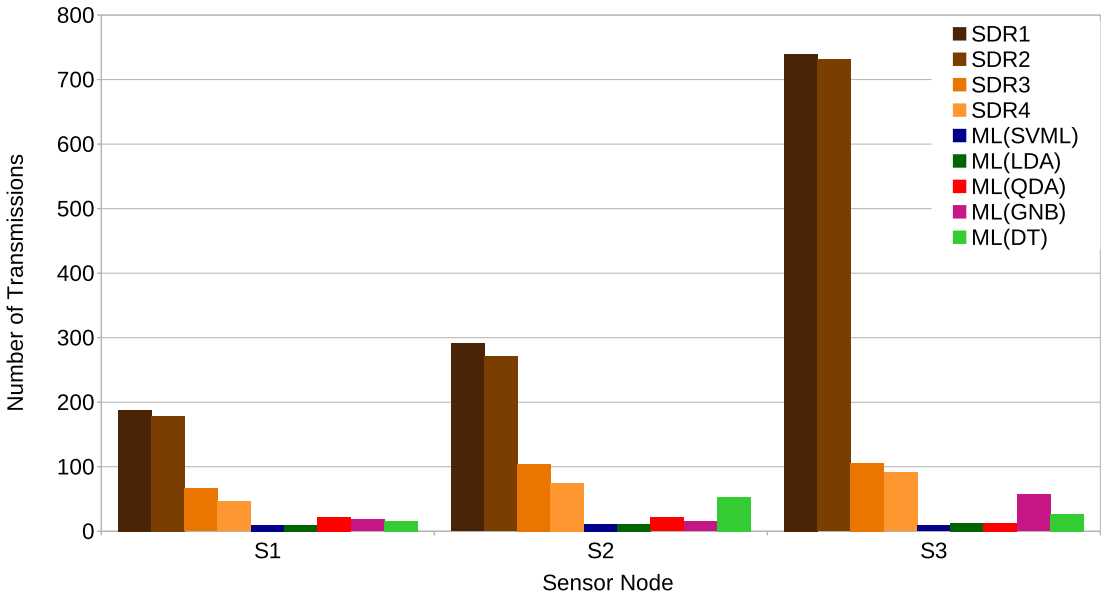

Figure 4.6 Number of transmissions encountered by sensor nodes S1, S2 and S3 of the occupancy estimation setup while running SDR and MLDR schemes.

sample the dataset more to make SDR match the linear SVM in data reduction and while this can be achieved through trial and error, performance takes a big hit. For example, if we use $e_t = 2.0$, $e_h = 5.0$ and $e_l = 120$, the number of transmissions get reduced to almost what the linear SVM gives but the F1 score drops down to 0.61 (SVM with RBF kernel). In general, for the occupancy estimation application, ML-based data reduction schemes show 18 to 82 times reduction in data transmissions as compared to Shewhart and 99.91% data transmission reduction overall in the best case while maintaining similar performance metrics.

Chapter 5

Concluding Remarks

5.1 Conclusions

The work presented in this thesis can be broken down into three parts: occupancy estimation using ML, analysis of ML algorithms from the perspective of constrained microcontrollers and ML-based data transmission reduction for occupancy applications. The following three paragraphs conclude the work presented on these three topics.

In Chapter 3, a new occupancy estimation paradigm was proposed that employed ML on a deployment of multiple multivariate sensor nodes. A description of the sensor node designs as well as the two WSN deployments for dataset collection were presented. The preprocessing and feature engineering techniques applied on the datasets were discussed. In particular, a new slope feature engineered out of CO_2 values via regression was shown to complement the performance of the system. Metrics like accuracy, F1 score and confusion matrix, obtained by applying various ML algorithms on the small-room dataset, were presented for various homogeneous and heterogeneous feature combinations. The results demonstrated a promising 98.4% accuracy for occupancy estimation and a high F1 score of 0.953 using SVM with RBF kernel on the complete dataset. A PCA-reduced dataset was also shown to exhibit a high accuracy of 92% and a moderate F1 score of 0.72 with just four principal components and no light data. A description of the real-time occupancy estimation system with real-world snapshots was also presented. Various shortcomings of the large-workspace occupancy experiment were also discussed in the chapter.

In the first half of Chapter 4, five ML algorithms namely LDA, QDA, GNB, SVM and DT, were analyzed theoretically from the perspective of constrained microcontrollers. The time and space complexities for performing the inference operation in real time was discussed along with various strategies to offload the trained parameters to constrained sensor nodes. All of these algorithms were practically tested on Atmel ATmega328P for up to 20 dimensions. The experimental results were shown to obey the theoretical inferences.

Based on the foundations laid in the first half of Chapter 4, an ML-based data transmission reduction scheme was proposed in the second half of the chapter for constrained sensor nodes that are a part

of some application-specific IoT network. As per the proposed scheme, ML was run on the sensor nodes themselves and only the inference was transmitted to the edge and that too when it differed from the previously transmitted inference. The proposed scheme was validated by performing three different experiments on the occupancy detection and estimation datasets: no data reduction, Shewhart-based data reduction and ML-based data reduction. For occupancy detection, a 99.71% reduction in the number of transmissions was achieved by the proposed scheme for testing set 1 and 99.6% for testing set 2 while upholding identical performance metrics as the no data reduction case. For both the testing sets, the proposed scheme outperformed Shewhart by a factor of 25. For occupancy estimation, the proposed scheme achieved 99.91% data transmission reduction in the best case while maintaining similar performance metrics. When compared to Shewhart, the proposed scheme showed 18 to 82 times reduction in the number of data transmissions.

5.2 Future Directions

- The next natural step after occupancy estimation is localization of occupants and that too by using a similar setup of non-intrusive sensor nodes in rooms. While localization by training separate ML models for four virtual zones of the small room is explored implicitly, a deeper study pertaining to this is required.
- A new occupancy dataset pertaining to large workspaces needs to be collected with a time-stamped note of the status of ventilation and cooling systems. This would allow for a much deeper analysis of occupancy estimation in large workspaces than what is possible with the current dataset.
- Chapter 4 provides an in-depth analysis of five supervised ML algorithms in the context of constrained microcontrollers. More supervised, unsupervised and RL algorithms can be studied for embedded devices. Moreover, a generic library can be built in Python to translate a trained ML model directly to Arduino code. This would make the offloading process a lot simpler.
- The proposed ML-based data transmission reduction scheme performs very well for occupancy applications. This scheme can be adapted and tested for more IoT applications.
- For resource-abundant IoT devices like smart vehicles and smart watches, more complex ML models can be used to intelligently prioritize data transmissions. IoT devices can generate a huge amount of data. Reducing data right at the source would lessen the burden on the edge and the cloud. Thus, the IoT device, the edge and the cloud can collaborate to form a more efficient system.

Related Publications

1. Adarsh Pal Singh and Sachin Chaudhari, "Embedded Machine Learning-Based Data Reduction in Application-Specific Constrained IoT Networks," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC '20)*, 2020.

2. Adarsh Pal Singh, Vivek Jain, Sachin Chaudhari, Frank Alexander Kraemer, Stefan Werner and Vishal Garg, "Machine Learning-Based Occupancy Estimation Using Multivariate Sensor Nodes," in *2018 IEEE Globecom Workshops (GC Wkshps)*, 2018.

Bibliography

[1] International Data Corporation, "The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025," June 2019. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=prUS45213219

[2] X. Ma, T. Yao, M. Hu, Y. Dong, W. Liu, F. Wang, and J. Liu, "A Survey on Deep Learning Empowered IoT Applications," *IEEE Access*, vol. 7, pp. 181 721–181 732, 2019.

[3] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine Learning for Internet of Things Data Analysis: A Survey," *Digital Communications and Networks*, vol. 4, no. 3, pp. 161–175, 2018.

[4] T. J. Saleem and M. A. Chishti, "Deep Learning for Internet of Things Data Analytics," *Procedia Computer Science*, vol. 163, pp. 381–390, 2019.

[5] V. Garg and N. Bansal, "Smart occupancy sensors to reduce energy consumption," *Energy and Buildings*, vol. 32, no. 1, pp. 81–87, 2000.

[6] V. L. Erickson, S. Achleitner, and A. E. Cerpa, "POEM: Power-Efficient Occupancy-Based Energy Management System," in *Proceedings of the 12th International Conference on Information Processing in Sensor Networks*, ser. IPSN '13. New York, NY, USA: ACM, 2013, pp. 203—-216.

[7] B. Balaji, J. Xu, A. Nwokafor, R. Gupta, and Y. Agarwal, "Sentinel: Occupancy Based HVAC Actuation Using Existing WiFi Infrastructure within Commercial Buildings," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. New York, NY, USA: ACM, 2013.

[8] J. Scott, A. Bernheim Brush, J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar, "PreHeat: Controlling Home Heating Using Occupancy Prediction," in *Proceedings of the 13th International Conference on Ubiquitous Computing*, ser. UbiComp '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 281—-290.

[9] N. Li, G. Calis, and B. Becerik-Gerber, "Measuring and monitoring occupancy with an RFID based system for demand-driven HVAC operations," *Automation in Construction*, vol. 24, pp. 89–99, 2012.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The Design of an Acquisitional Query Processor for Sensor Networks," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '03. New York, NY, USA: ACM, 2003, pp. 491—502.

[11] G. M. Dias, B. Bellalta, and S. Oechsner, "A Survey About Prediction-Based Data Reduction in Wireless Sensor Networks," *ACM Comput. Surv.*, vol. 49, no. 3, Nov. 2016.

[12] E. Rubio-Drosdov, D. Díaz-Sánchez, F. Almenárez, P. Arias-Cabarcos, and A. Marín, "Seamless Human-Device Interaction in the Internet of Things," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 4, pp. 490–498, 2017.

[13] S. Robinson, "Voice-Activated Technology Redefines the Internet of Things," March 2020. [Online]. Available: https://www.iotworldtoday.com/2020/03/24/voice-activated-technology-redefines-the-internet-of-things/

[14] O. Bello and S. Zeadally, "Intelligent Device-to-Device Communication in the Internet of Things," *IEEE Systems Journal*, vol. 10, no. 3, pp. 1172–1182, 2016.

[15] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[16] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "A comparative study of LPWAN technologies for large-scale IoT deployment," *ICT Express*, vol. 5, no. 1, pp. 1–7, 2019.

[17] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[18] A. R. Biswas and R. Giaffreda, "IoT and cloud convergence: Opportunities and challenges," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 2014, pp. 375–376.

[19] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: Association for Computing Machinery, 1992, pp. 144—-152.

[20] J. Heaton, *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks*, ser. Artificial Intelligence for Humans. Createspace Independent Publishing Platform, 2015.

[21] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-Propagating Errors*. Cambridge, MA, USA: MIT Press, 1988, pp. 696—699.

[22] I. Jolliffe and J. Cadima, "Principal component analysis: A review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, p. 20150202, 04 2016.

[23] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms," *Annals of Data Science*, vol. 2, 08 2015.

[24] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[25] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J. Oh, "Semisupervised Deep Reinforcement Learning in Support of IoT and Smart City Services," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 624–635, 2018.

[26] Y. Gu, Y. Chen, J. Liu, and X. Jiang, "Semi-supervised deep extreme learning machine for Wi-Fi based localization," *Neurocomputing*, vol. 166, pp. 282–293, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121500418X

[27] J. Park, K. Jang, and S. Yang, "Deep neural networks for activity recognition with multi-sensor data in a smart home," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, 2018, pp. 155–160.

[28] W. Zhang, Y. Zhang, J. Zhai, D. Zhao, L. Xu, J. Zhou, Z. Li, and S. Yang, "Multi-source data fusion using deep learning for smart refrigerators," *Computers in Industry*, vol. 95, pp. 15–21, 2018.

[29] B. A. Erol, A. Majumdar, J. Lwowski, P. Benavidez, P. Rad, and M. Jamshidi, *Improved Deep Neural Network Object Tracking System for Applications in Home Robotics", bookTitle="Computational Intelligence for Pattern Recognition*. Cham: Springer International Publishing, 2018, pp. 369–395.

[30] Y. Lv, Y. Duan, W. Kang, Z. Li, and F. Wang, "Traffic Flow Prediction With Big Data: A Deep Learning Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2015.

[31] K. S. Sang, B. Zhou, P. Yang, and Z. Yang, "Study of Group Route Optimization for IoT Enabled Urban Transportation Network," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2017, pp. 888–893.

[32] X. Ma, Y.-J. Wu, Y. Wang, F. Chen, and J. Liu, "Mining smart card data for transit riders' travel patterns," *Transportation Research Part C: Emerging Technologies*, vol. 36, pp. 1–12, 2013.

[33] J. Zhang, Y. Zheng, and D. Qi, "Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, pp. 1655—1661.

[34] F. Al-Turjman and A. Malekloo, "Smart parking in IoT-enabled cities: A survey," *Sustainable Cities and Society*, vol. 49, p. 101608, 2019.

[35] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo, "Deep learning for decentralized parking lot occupancy detection," *Expert Systems with Applications*, vol. 72, pp. 327–334, 2017.

[36] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[37] J. Dequaire, P. Ondrúška, D. Rao, D. Wang, and I. Posner, "Deep tracking in the wild: End-to-end tracking using recurrent neural networks," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 492–512, 2018.

[38] D. Singh and C. K. Mohan, "Deep Spatio-Temporal Representation for Detection of Road Accidents Using Stacked Autoencoder," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 3, pp. 879–887, 2019.

[39] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. Jesus, R. Berriel, T. Paixão, F. Mutz, L. Veronese, T. Oliveira-Santos, and A. F. D. Souza, "Self-Driving Cars: A Survey," 2019.

[40] N. Y. Hammerla, S. Halloran, and T. Plötz, "Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, pp. 1533—-1540.

[41] J. Zhu, A. Pande, P. Mohapatra, and J. J. Han, "Using Deep Learning for Energy Expenditure Estimation with wearable sensors," in *2015 17th International Conference on E-health Networking, Application Services (HealthCom)*, 2015, pp. 501–506.

[42] A. Hannun, P. Rajpurkar, M. Haghpanahi, G. Tison, C. Bourn, M. Turakhia, and A. Ng, "Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network," *Nature Medicine*, vol. 25, 01 2019.

[43] X. Zeng, K. Cao, and M. Zhang, "MobileDeepPill: A Small-Footprint Mobile Deep Learning System for Recognizing Unconstrained Pill Images," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 56—-67.

[44] W. Chang, L. Chen, C. Hsu, C. Lin, and T. Yang, "A Deep Learning-Based Intelligent Medicine Recognition System for Chronic Patients," *IEEE Access*, vol. 7, pp. 44 441–44 458, 2019.

[45] Y. He, J. Deng, and H. Li, "Short-Term Power Load Forecasting with Deep Belief Network and Copula Models," in *2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 1, 2017, pp. 191–194.

[46] Xishuang Dong, Lijun Qian, and Lei Huang, "Short-term load forecasting in smart grid: A combined CNN and K-means clustering approach," in *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2017, pp. 119–125.

[47] K. Amarasinghe, D. L. Marino, and M. Manic, "Deep neural networks for energy load forecasting," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, 2017, pp. 1483–1488.

[48] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using Deep Neural Networks," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 2016, pp. 7046–7051.

[49] B. V. Mbuwir, F. Ruelens, F. Spiessens, and G. Deconinck, "Battery Energy Management in a Microgrid Using Batch Reinforcement Learning," *Energies*, vol. 10, p. 1846, 11 2017.

[50] R. Lu, S. H. Hong, and X. Zhang, "A Dynamic pricing demand response algorithm for smart grid: Reinforcement learning approach," *Applied Energy*, vol. 220, pp. 220–230, 2018.

[51] J. Yan, H. He, X. Zhong, and Y. Tang, "Q-Learning-Based Vulnerability Analysis of Smart Grid Against Sequential Topology Attacks," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 1, pp. 200–210, 2017.

[52] Y. He, G. J. Mendis, and J. Wei, "Real-Time Detection of False Data Injection Attacks in Smart Grid: A Deep Learning-Based Intelligent Mechanism," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2505–2516, 2017.

[53] I. B. A. Ang, F. Dilys Salim, and M. Hamilton, "Human occupancy recognition with multivariate ambient sensors," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, 2016, pp. 1–6.

[54] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models," *Energy and Buildings*, vol. 112, pp. 28–39, 2016.

[55] E. Hailemariam, R. Goldstein, R. Attar, and A. Khan, "Real-time Occupancy Detection Using Decision Trees with Multiple Sensor Types," in *Proceedings of the 2011 Symposium on Simulation for Architecture and Urban Design*, ser. SimAUD '11. San Diego, CA, USA: Society for Computer Simulation International, 2011, pp. 141–148.

[56] B. Dong, B. Andrews, K. P. Lam, M. Höynck, R. Zhang, Y.-S. Chiou, and D. Benitez, "An information technology enabled sustainability test-bed (ITEST) for occupancy detection through an environmental sensing network," *Energy and Buildings*, vol. 42, no. 7, pp. 1038–1046, 2010.

[57] Z. Yang, N. Li, B. Becerik-Gerber, and M. Orosz, "A Multi-Sensor Based Occupancy Estimation Model for Supporting Demand Driven HVAC Operations," in *Proceedings of the 2012 Symposium on Simulation for Architecture and Urban Design*, ser. SimAUD '12. San Diego, CA, USA: Society for Computer Simulation International, 2012.

[58] M. K. Masood, Yeng Chai Soh, and V. W. . Chang, "Real-time occupancy estimation using environmental parameters," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1–8.

[59] A. Dey, X. Ling, A. Syed, Y. Zheng, B. Landowski, D. Anderson, K. Stuart, and M. E. Tolentino, "Namatad: Inferring occupancy from building sensors using machine learning," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, pp. 478–483.

[60] Z. Chen, R. Zhao, Q. Zhu, M. K. Masood, Y. C. Soh, and K. Mao, "Building Occupancy Estimation with Environmental Sensors via CDBLSTM," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 12, pp. 9549–9559, 2017.

[61] P. Perner, "Machine Learning and Data Mining in Pattern Recognition," in *10th International Conference on Machine Learning and Data Mining in Pattern Recognition (MLDM)*, ser. Lecture Notes in Computer Science. Springer, 2014.

[62] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-Sized Machine Learning Models to Tiny IoT Devices," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: ACM, 2019, pp. 79—95.

[64] S. Jacobs and F. Rios-Gutierrez, "Novel Method for Using Q-Learning in Small Microcontrollers," in *Proceedings of the 51st ACM Southeast Conference*, ser. ACMSE '13. New York, NY, USA: ACM, 2013.

[65] P. Karvelis, T. Michail, D. Mazzei, S. Petsios, A. Bau, G. Montelisciani, and C. Stylios, "Adopting and Embedding Machine Learning Algorithms in Microcontroller for Weather Prediction," in *2018 International Conference on Intelligent Systems (IS)*, 2018, pp. 474–478.

[66] T. Sheltami, M. Musaddiq, and E. Shakshuki, "Data compression techniques in Wireless Sensor Networks," *Future Generation Computer Systems*, vol. 64, pp. 151–162, 2016.

[67] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive Stream Resource Management Using Kalman Filters," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 11—22.

[68] B. Qutub Ali, N. Pissinou, and K. Makki, "Approximate replication of data using adaptive filters in Wireless Sensor Networks," in *2008 3rd International Symposium on Wireless Pervasive Computing*, 2008, pp. 365–369.

[69] L. Tan and M. Wu, "Data Reduction in Wireless Sensor Networks: A Hierarchical LMS Prediction Approach," *IEEE Sensors Journal*, vol. 16, no. 6, pp. 1708–1715, 2016.

[70] A. Shastri, V. Jain, R. P. Singh, S. Chaudhari, and S. S. Chouhan, "On the Implementation of LMS-based Algorithm for Increasing the Lifetime of IoT Networks," in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, 2018, pp. 1–6.

[71] F. A. Aderohunmu, G. Paci, D. Brunelli, J. D. Deng, and L. Benini, "Prolonging the lifetime of wireless sensor networks using light-weight forecasting algorithms," in *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 2013, pp. 461–466.

[72] F. A. Aderohunmu, G. Paci, D. Brunelli, J. D. Deng, L. Benini, and M. Purvis, "An Application-Specific Forecasting Algorithm for Extending WSN Lifetime," in *2013 IEEE International Conference on Distributed Computing in Sensor Systems*, 2013, pp. 374–381.

[73] A. Shastri, V. Jain, S. Chaudhari, S. S. Chouhan, and S. Werner, "Improving Accuracy of the Shewhart-based Data-Reduction in IoT Nodes using Piggybacking," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, 2019, pp. 943–948.

[74] A. Shastri, "Data Reduction for IoT Networks : Algorithms and Implementation," Master's thesis, International Institute of Information Technology, Hyderabad, India, 2019.

[75] Y. Inagaki, R. Shinkuma, T. Sato, and E. Oki, "Prioritization of Mobile IoT Data Transmission Based on Data Importance Extracted From Machine Learning Model," *IEEE Access*, vol. 7, pp. 93 611–93 620, 2019.