12-2018

Efficacy of Deep Learning in Support of Smart Services

Basheer Mohammed Basheer Qolomany

Western Michigan University, basheer.qolomany@wmich.edu

Follow this and additional works at: https://scholarworks.wmich.edu/dissertations

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.wmich.edu/dissertations/3374

This Dissertation-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Dissertations by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
© 2018 Basheer Mohammed Basheer Qolomany
DEDICATION

I dedicate this dissertation to the blessed soul of my father, may Allah have mercy on him.

To the inexhaustible spring of tenderness and the meanings of sacrifice my mother.

To my brother and sisters, may Allah bless them.
ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious and the Most Merciful. First, I would like to offer my utmost gratitude to Allah S.W.T the Almighty, the Creator of all things, for giving me the blessing, the strength and endurance to complete this dissertation.

The completion of this dissertation would not have been possible without the support and encouragement of several special people. Hence, I would like to express my sincere gratitude to my advisors, Dr. Ajay Gupta and Dr. Ala Al-Fuqaha, for their continuous guidance, support and encouragement to complete this dissertation. I would also like to acknowledge the committee members, Dr. Driss Benhaddou and Dr. Alvis Fong, for their constructive feedback and brilliant comments and suggestions throughout the process.

I would like to express my special appreciation to my late father. Your memory, decency, sincerity and kind soul will stay with me always. My tender mother. I always knew that you both believed in me and wanted the best for me. I do believe that your prayers and satisfaction are greatly influenced every success in my life.

I would also like to express my heartfelt thanks to my brother Badeel and my dearest sisters Gulzar, Anwaar, Nawshar, Khitam and Muntaha, for their continuous prayers and unparalleled love.

Finally, my thanks go to all the people who have supported me to complete this work directly or indirectly.

Basheer Mohammed Basheer Qolomany
EFFICACY OF DEEP LEARNING IN SUPPORT OF SMART SERVICES

Basheer Mohammed Basheer Qolomany, Ph.D.
Western Michigan University, 2018

The massive amount of streaming data generated and captured by smart service appliances, sensors and devices needs to be analyzed by algorithms, transformed into information, and minted to extract knowledge to facilitate timely actions and better decision making. This can lead to new products and services that can dramatically transform our lives. Machine learning and data analytics will undoubtedly play a critical role in enabling the delivery of smart services. Within the machine-learning domain, Deep Learning (DL) is emerging as a superior new approach that is much more effective than any rule or formula used by traditional machine learning. Furthermore, it can even alleviate engineers from the task of defining features. This research explores the efficacy of DL in support of smart services. Despite recent advancements in DL for Internet of Things (IoT) smart services, there are still significant challenges that need to be addressed for this technology to mature. Hyper parameter tuning for DL models, besides the expensive computational cost and heavy memory overhead make DL unaffordable. In order to address these challenges, we propose a two-pronged solution.

Successful application of DL requires careful and sometimes very expensive hyper parameter searches, tuning, and testing. Manual parameter setting, and grid search are common approaches that can ease the users tasks for setting these important parameters values of the DL models. Nonetheless, these two approaches can be very time-consuming. Optimization methods
therefore need to be used to help find optimal parameter settings. In this research, we show that the Particle Swarm Optimization (PSO) technique holds a great potential to optimize parameter settings; thus, saving valuable computational resources during the tuning process of DL models.

Another goal that we focus on during this research, is to investigate techniques to optimize execution of DL models on resource-constrained wearable IoT devices. Because of their overhead (e.g., memory, computation and energy), DL models are yet to become mainstream on mobile/IoT embedded platforms. Therefore, one of the main objectives of this research is to utilize cloud hosted Machine Learning as a Service (MLaaS) providers to collect big data from the resource-constrained devices and build prediction models on the cloud. Those prediction models are then sent to the underlying resource-constrained devices to be run locally without necessarily being always connected to the cloud. However, models’ trust represents a potentially serious threat in this paradigm. In this research, we propose a heuristic that maximizes the level of trust of DL models by selecting a subset of models from a superset of models. During each period, our proposed heuristic switches between the subset of selected models in order to maximize the trustworthiness while respecting given reconfiguration budget and rate. Due to the difficulty of the problem in real-world scenarios, we propose an intelligent real-time heuristic that can be used in large-scale deployments of IoT resource-constrained devices. The heuristic algorithm strives to make learning techniques more trustworthy since it avoids frequent reconfigurations. This also minimizes the communications overhead between the cloud and the resource-constrained devices. We prove that the competitive ratio of our proposed heuristic is $O(1)$ when the reconfiguration rate is proportional to the total time and the reconfiguration budget is constant. Therefore, our proposed heuristic achieves an optimal competitive ratio in a polynomial time approximation scheme for the problem.
TABLE OF CONTENTS

ACKNOWLEDGMENTS ................................................................. ii

LIST OF TABLES ................................................................. viii

LIST OF FIGURES ................................................................. ix

ABBREVIATIONS ................................................................. xi

CHAPTER

1 INTRODUCTION AND BACKGROUND ........................................... 1

1.1 Introduction ................................................................. 1

1.2 Challenges of Utilizing Deep learning in IoT Smart Services .......... 5

1.2.1 Lack of Massive IoT Dataset ........................................ 5

1.2.2 Preprocessing ........................................................ 5

1.2.3 Security, Privacy and Trust Preserving Deep Learning .............. 6

1.2.4 Deep Learning for IoT Devices .................................... 6

1.2.5 Parameter Tuning ..................................................... 7

1.3 Dissertation’s Contributions .............................................. 7

2 SMART BUILDINGS: A SURVEY ON APPLICATIONS, ANALYTICS AND MA-
CHINE LEARNING TECHNIQUES ......................................... 10

2.1 ML Background for SBs ............................................... 10

2.2 ML Models ............................................................. 12

2.2.1 Supervised Learning ............................................... 13

2.2.2 Unsupervised Learning ........................................... 24

2.2.3 Semi-Supervised Learning ....................................... 27
CHAPTER

2.2.4 Reinforcement Learning ............................................. 28

2.3 ML Tasks for SBs .......................................................... 29
   2.3.1 Data Collection and Acquisition ................................. 30

2.4 ML Tools and Platforms for SBs ....................................... 37
   2.4.1 H2O ................................................................. 38
   2.4.2 Apache SINGA ..................................................... 39
   2.4.3 MLlib (Spark) ..................................................... 40
   2.4.4 TensorFlow ....................................................... 40
   2.4.5 Torch ............................................................ 41
   2.4.6 Deeplearning4j ................................................. 42
   2.4.7 Massive Online Analysis (MOA) ............................... 42
   2.4.8 Caffe ........................................................... 43
   2.4.9 Azure ML ......................................................... 44
   2.4.10 Amazon ML ..................................................... 44

2.5 Real-time Big Data Analytics Tools for SBs .......................... 45
   2.5.1 Apache Storm ................................................... 47
   2.5.2 Apache Kafka .................................................... 47
   2.5.3 Oracle .......................................................... 48
   2.5.4 Apache Samza ................................................... 49
   2.5.5 Amazon Kinesis Streams ....................................... 50
   2.5.6 Apache Spark Streaming ...................................... 50
Table of Contents—Continued

CHAPTER

2.5.7 Apache Flume .......................................................... 51
2.5.8 Apache SAMOA .......................................................... 52
2.6 Applications of ML-Based Context-Aware Systems for SBs ........ 53
2.7 Open Issues and Future Research Directions ............................. 66
  2.7.1 Security and Privacy .................................................. 66
  2.7.2 SBs and Context-Aware Computing ................................ 67
  2.7.3 Personal Data Stream Management in SBs .......................... 69
  2.7.4 Big Data Challenges in SBs .......................................... 70
  2.7.5 Interoperability ........................................................ 71
  2.7.6 Reliability ............................................................ 71
  2.7.7 Integration ............................................................. 72

3 PARAMETERS OPTIMIZATION OF DEEP LEARNING MODELS USING PARTICLE SWARM OPTIMIZATION .......................... 75
  3.1 Introduction ............................................................... 76
  3.2 Motivations .............................................................. 78
  3.3 PSO-Based Parameter Optimization Model .............................. 81
  3.4 Experimental Results and Lessons Learned ........................... 85
  3.5 Conclusions and Future Work ......................................... 87

4 ROLE OF DEEP LSTM NEURAL NETWORKS AND WI-FI NETWORKS IN SUPPORT OF OCCUPANCY PREDICTION IN SMART BUILDINGS ...... 90
  4.1 Introduction ............................................................... 91
  4.2 Related Work ............................................................ 93
Table of Contents—Continued

CHAPTER

4.3 Models .................................................. 94
   4.3.1 ARIMA Time Series Model ....................... 95
   4.3.2 Deep LSTM Model ................................. 97
   4.3.3 Forecast Evaluation Methods ................... 100

4.4 Experimental Setup ..................................... 101
   4.4.1 Dataset ........................................... 101
   4.4.2 Data Pre-Processing and Preparation ............ 102
   4.4.3 Experimental Results ............................ 103
   4.4.4 Lessons Learned .................................. 105

4.5 Conclusions ............................................. 109

5 TOWARD TRUSTED AND FAIL-SAFE EXECUTION OF CLOUD-BASED MACHINE LEARNING MODELS ......................... 110

5.1 Introduction ........................................... 111
   5.1.1 Motivation ....................................... 111
   5.1.2 Contributions .................................. 115
   5.1.3 Case Studies .................................... 116

5.2 Related Work ........................................... 119

5.3 System Model .......................................... 123

5.4 Problem Formulation ................................... 123
   5.4.1 Lower Bound .................................... 125
   5.4.2 Upper Bound .................................... 126
### Table of Contents—Continued

**CHAPTER**

- 5.4.3 Competitive Solution ........................................... 126
- 5.4.4 Proof of NP-Completeness ..................................... 129
- 5.5 Worst-Case Analysis (Competitive Ratio Analysis) ............... 130
- 5.6 Experimental Setups .................................................. 132
  - 5.6.1 Datasets .......................................................... 133
  - 5.6.2 Setup ............................................................. 133
- 5.7 Experimental Results .................................................. 136
  - 5.7.1 Traffic Flow Volume Prediction ................................. 136
  - 5.7.2 Predictive Maintenance in IIoT ................................. 139
  - 5.7.3 Lessons Learned and Results Discussion ...................... 141
- 5.8 Conclusions and Future Works ...................................... 143
- 6 CONCLUSIONS AND FUTURE WORKS ................................. 146

REFERENCES ................................................................. 177
LIST OF TABLES

2.1 Comparison of ML techniques. 17
2.2 Comparison between deep learning and ML tools. 46
2.3 Comparison between real-time data analytics tools. 53
3.1 Training time for week ends and work days by SVM, and DNN. 84
3.2 Summaries of 7 datasets for each days of week. 87
4.1 The best configurations for LSTM combined and separate models for the three time scales 60, 30, and 15 minutes for the whole building as well as the AP level. 109
LIST OF FIGURES

1.1 Smart service dimensions.................................................. 2

2.1 ML styles............................................................................. 12

2.2 ML tasks in SB environment............................................... 30

2.3 Steps involved in applying ML models in an SB environment. 55

2.4 SB services taxonomy......................................................... 56

3.1 The effect of manual configuration setting on the accuracy. . . 80

3.2 Comparison between three different swarm sizes (10, 25 and 50). 88

3.3 Comparison between PSO and grid search to predict within a 60 minute interval. 88

3.4 Comparison between PSO and grid search to predict within a 30 minute interval. 88

3.5 Comparison between PSO and grid search to predict within a 15 minute interval. 89

4.1 LSTM module....................................................................... 99

4.2 Comparison between separate and combined LSTM models. 106

4.3 Comparison between LSTM and ARIMA models................ 107

4.4 Total number of neurons used to build the three time scales models for the building level vs. the AP level. 107

4.5 The effect of the lag size on reducing the RMSE in LSTM models. 108

4.6 The effect of the number of neurons on reducing the RMSE in LSTM models. 108

4.7 The effect of the number of hidden layers on reducing the RMSE in LSTM models. 108

5.1 General architecture for the proposed system....................... 114
List of Figures—Continued

5.2 Trust-based model selection problem. .................................................. 116
5.3 The exchange of data/models between resource constrained devices and the cloud using message ferries. .................................................. 119
5.4 The research gap addressed in the paper. .............................................. 120
5.5 An illustration of our proposed splice heuristic. ..................................... 126
5.6 The pipeline of system implementation. ............................................... 132
5.7 Example of swap $x$ and $100 - x$ percentiles attack model. ............... 134
5.8 RMSE of the selected models using the fixing heuristic vs. individual models. . . . 136
5.9 Trust level of upper bound, lower bound, and proposed heuristics. .......... 137
5.10 The effect of the number of selected models on the trust level. .............. 138
5.11 The effect of $\lambda$ on the trust level. .................................................. 139
5.12 The effect of malicious models on the trust level. ................................ 140
5.13 RMSE of the selected models using the fixing heuristic vs. individual models. . . . 141
5.14 Trust level of upper bound, lower bound, and proposed heuristics. ........ 142
5.15 The effect of the number of selected models on the trust level. .............. 143
5.16 The effect of $\lambda$ on the trust level. .................................................. 144
5.17 The effect of malicious models on the trust level. ................................ 145
ABBREVIATIONS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
<td>Access Point</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Networks</td>
</tr>
<tr>
<td>ARIMA</td>
<td>Auto Regression Integrating Moving Average</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DBN</td>
<td>Deep Belief Networks</td>
</tr>
<tr>
<td>DBM</td>
<td>Deep Boltzmann Machine</td>
</tr>
<tr>
<td>DL</td>
<td>Deep Learning</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop Distributed File System</td>
</tr>
<tr>
<td>HMMs</td>
<td>Hidden Markov Models</td>
</tr>
<tr>
<td>ILP</td>
<td>Integer Linear Programming</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>LDA</td>
<td>Linear Discriminant Analysis</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short-Term Memory</td>
</tr>
<tr>
<td>ML</td>
<td>Machine learning</td>
</tr>
<tr>
<td>MLaaS</td>
<td>Machine Learning as a Service</td>
</tr>
<tr>
<td>PSO</td>
<td>Particle Swarm Optimization</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root Mean Square Error</td>
</tr>
<tr>
<td>SB</td>
<td>Smart Building</td>
</tr>
<tr>
<td>SVMs</td>
<td>Support Vector Machines</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Introduction

"Internet of Things", and "Smart Services” are the today’s buzzwords when tracking economic news. Smart service environment is any real or virtual location equipped with different types of devices which communicate via some kind of network and expected to be heterogenous [1]. These devices can be any kind of sensors and actuators, mobile devices, resource-constrained IoT embedded platforms, and wearable systems. Examples of smart services include smart homes, smart buildings, smart factories, smart offices, smart libraries, smart hospitals, smart shops, smart vehicles and transportation systems, smart grids or ambient assisted living environments. Figure 1 shows a general structure for smart services with their all dimensions.

There is a growing interest in IoT smart services and predicting behaviors of inhabitants is a key element for the success of smart services. Sensors and smart objects within smart services simultaneously produce raw data in an automated way; such devices may store the data for a certain time interval or report it to governing components [2]. Analytics are at the core of smart services. There are two main methods to implement smart service analytics, either use conventional rule-based systems or apply Machine Learning Algorithms to smart environment datasets [3]. The conventional rule-based systems are supposedly easier to analyze. However, this advantage is negated as the system evolves, with patches of rules being stacked upon each other to account
Figure 1.1: Smart service dimensions.

for the proliferation of new rule exceptions, thus resulting in a hard-to-decipher tangle of coded rules. As human programmers manage the hard work of rule creation and modification, rule-based systems suffer from compromised performance. They have shown to be less responsive in adapting to new types of data, such as data sourced from an upgraded sensor, or a new sensor of previously unutilized data. Rule-based systems can also fail to adapt to a changing domain, e.g., a new furniture layout or new lighting sources.

Machine learning techniques have been widely used to develop smart systems which have the
ability to sense and to react according to context modifications in smart Services [4]. However, the methodology for machine learning implementations is different from traditional techniques. There are many different machine learning algorithms, according to the two well-known theorems, namely No Free Lunch theorem and Ugly Duckling theorem – there are no algorithms that can be said to be better than any other. According to the No Free Lunch theorem, any two algorithms may perform equally well in solving a problem if there is no prior information about the problem. While Ugly Duckling theorem states it is impossible to say that any two different patterns would be more similar to each other than any other pairs [5].

Deep learning is the machine learning approach that has drawn heavily on the knowledge of the human brain (artificial neural networks), statistics and applied mathematics [6]. It is the artificial neural networks (ANN) that are composed of many layers. Deep learning neural network architectures differ from “normal” neural networks because they have more hidden layers and differ from shallow machine learning algorithms (e.g. SVM, linear regression etc.) because they can be trained in an unsupervised or supervised manner for both unsupervised and supervised learning tasks. On the other hand, deep learners have the potential to extract better representations from the raw data to create much better models. In the rule-based system world, and even with traditional machine learning, the system engineer requires extensive information about the domain in order to build a good system. In the deep-learning world, this is no longer necessary.

Broadly speaking, the goal of deep learning is to model complex, hierarchical features in data. “Deep learning” is not a particular type of algorithm, such as feedforward feedforward neural networks or SVMs, but rather a set of machine learning algorithms. Recently, different deep learning architecture such as Recurrent Neural Network (RNN) [7], Deep Boltzmann Machine
Deep Belief Networks (DBN) [9], Convolutional Neural Network (CNN) [10], and Auto encoder [11] has been employed to the smart service problem.

In recent years, deep learning has been used successfully and has made substantial improvements in all types of big data analytics applications, especially natural language processing (NLP), machine language translation, medical diagnosis, stock market trading signals, network security and image identification. This is mainly due to more powerful computers, larger datasets and techniques to train deeper networks. In addition, deep learning models are flexible (enabling similar models to be used in wide range of problems).

At Facebooks AI lab, theyve built a deep learning system capable of answering simple questions to which it had never previously been exposed. The Echo, Amazons smart speaker, uses deep learning techniques. Three years ago, Microsofts chief research officer impressed attendees at a lecture in China with a demonstration of deep learning speech software that translated his spoken English into Chinese, then instantly delivered the translation using a simulation of his voice speaking Mandarin with an error rate of just 7%. It now uses the technology to improve voice search on Windows mobile and Bing. A few years ago, a Google deep learning network was shown 10 million unlabeled images from YouTube, and proved to be nearly twice as accurate at identifying the objects in the images (cats, human faces, flowers, various species of fish, and thousands of others) as any previous method. When Google deployed deep learning on its Android voice search, errors dropped by 25% overnight [12].

After building the machine learning models, they are applicable to many real-life problems in smart service environments. Generally, the potential uses of machine learning in a smart service environment can be divided into four categories: detection, recognition, prediction, and Optimiza-
tion. In this study we explore the role of deep learning methods as a proxy of IoT smart services.

1.2 Challenges of Utilizing Deep learning in IoT Smart Services

In this section we review several challenges to utilize deep learning models in IoT smart services.

1.2.1 Lack of Massive IoT Dataset

The main difference between deep learning and other Machine learning approaches is the amount of training data involved and the computational power needed by deep learning models. Deep Learning (DL) is fed with large data sets of diverse examples, from which the model learns for features to look for and produces an output with probability vectors in place. However, the lack of availability of large real-world datasets for IoT applications is a main hurdle for incorporating DL models in IoT smart services [13], as more data is needed for DL to achieve more accuracy. With small amount of data, the DL methods tend to overfit. In addition, the outliers become much more dangerous [14].

1.2.2 Preprocessing

The massive amounts of raw data that are being generated and captured by IoT smart sensors, and devices cannot be directly treated by DL models. This data need to be prepared in an appropriate representation to be fed in DL models. Most DL approaches need some sort of preprocessing to yield good results [15]. For example, most data generated from IoT smart devices need to be normalized and scaled into a specific range, or transformed into a standard representation before feed this massive amount of data to deep learning models. For IoT smart services, this preprocessing is
very complex and challenging, due to the heterogeneity of this data that have been collected from
different sources that may have various formats and distributions [16].

1.2.3 Security, Privacy and Trust Preserving Deep Learning

Recent studies have shown that deep learning models are vulnerable to well-designed input samples
at prediction phase, called adversarial examples [17]. Adversarial examples are the input artifacts
that are imperceptible to human observer but can easily fool deep learning model to misclassify
the input. The vulnerability to adversarial examples becomes one of the major risks for applying
deep learning in IoT smart and safety-critical spaces [18]. Thus, ensuring data security, privacy
and trust is a main concern in such smart and sensitive applications. Since IoT smart service big
data streaming will be transferred through the Internet for analytics, and can be thus observed
around the world. For instance, attackers could use adversarial examples to mislead a self-driving
vehicle to take unexpected action if the vision camera recognizes a stop sign as a speed limit sign.
Similarly, adversarial attacks could confuse a face recognition authentication camera or a voice
recognition system to breach a financial or government entity with misplaced authorization.

1.2.4 Deep Learning for IoT Devices

Despite the remarkable performance of DL in various smart service tasks, the computational in-
tensity that DL models need has hindered their widespread utilization in resource-constrained em-
bedded and IoT systems [19]. Hosting high computational complexity of DL models on low-cost
IoT devices may inevitably be constrained by limited computation resource, making the devices
hard to respond in real-time [20]. These requirements are expected to grow as the datasets sizes are
growing every day, and new techniques (e.g. model compression) arise to be part of the solutions for DL in IoT resource constrained devices.

1.2.5 Parameter Tuning

One of the challenges in a successful implementation of deep machine learning is setting the values for its hyper-parameters, particularly the topology of its network [21]. These hyper-parameters include the number of layers, the number of hidden units per layer, the activation function for a layer, the kernel size for a layer, the arrangement of these layers within the network, etc. Manually tuning these parameters (essentially through trial and error method) and finding high-quality settings for a previously unused dataset can be time-consuming and tedious task [22].

1.3 Dissertation’s Contributions

Subsections below show the problem(s), methods, major results and evaluations contained in each chapter of the dissertation as solutions to the discussed limitations and challenges from Section 1.2 that this dissertation focus on.

**Chapter 2: Smart Buildings: A Survey on Applications, Analytics and Machine Learning Techniques**

This chapter shows the survey paper for machine learning techniques in smart building environment which is one of the important part under the umbrella of smart services. In this chapter, we survey the area of smart building with a special focus on the role of machine learning and data analytic techniques in this context. This survey also reviews the current trends and challenges faced in the development of smart building services.
Chapter 3: Parameters Optimization of Deep Learning Models using Particle Swarm Optimization

This chapter shows the role of Particle Swarm Optimization (PSO) to optimize parameter settings and thus saves valuable computational resources during the tuning process of deep learning models. Specifically, we use a dataset collected from a Wi-Fi campus network to train deep learning models to predict the number of occupants and their locations. Our preliminary experiments indicate that PSO provides an efficient approach for tuning the optimal number of hidden layers and the number of neurons in each layer of the deep learning algorithm when compared to the grid search method. Our experiments illustrate that the exploration process of the landscape of configurations to find the optimal parameters is decreased by 77% - 85%. In fact, the PSO yields even better accuracy results.

Chapter 4: Role of Deep LSTM Neural Networks And Wi-Fi Networks in Support of Occupancy Prediction in Smart Buildings

This chapter shows the role of deep Long Short-Term Memory (LSTM) neural networks and Wi-Fi networks to replace sensor technology with time series models that can predict the number of occupants at a given location and time. We use Wi-Fi datasets readily available in abundance for smart services and train Auto Regression Integrating Moving Average (ARIMA) models and LSTM time series models. As a use case scenario of smart building services, these models allow forecasting of the number of people at a given time and location in 15, 30 and 60 minutes time intervals at building as well as Access Point (AP) level. For LSTM, we build our models in two ways: a separate model for every time scale, and a combined model for the three time scales. Our experiments show that LSTM combined model reduced the computational resources with respect
to the number of neurons by 74.5% for the AP level, and by 67.13% for the building level. Further, the root mean square error (RMSE) was reduced by 88.2% - 93.4% for LSTM in comparison to ARIMA for the building levels models and by 80.9% - 87% for the AP level models.

Chapter 5: Toward Trusted and Fail-Safe Execution of Cloud-Based Machine Learning Models

This chapter shows the effect of adversarial attacks on the trustworthiness of machine learning models that are built by MLaaS providers and then sent to the underlying IoT resource constrained devices to be run locally without necessarily being always connected to the cloud. We propose a heuristic that maximizes the level of trust of machine learning models by selecting a subset of models from a superset of models. During each period, our proposed heuristic switches between the subset of selected models in order to maximize the trustworthiness while respecting given reconfiguration budget and rate. We prove that the competitive ratio of our proposed heuristic is $O(1)$. Therefore, our proposed heuristic achieves an optimal competitive ratio in a polynomial time approximation scheme for the problem. To measure the performance of the proposed heuristic, we used an experimental transportation dataset to predict the number of cars as a proxy for smart cities services. We also used turbofan engine degradation simulation dataset to predict the remaining useful life of an engine as a proxy for Industrial IoT services. In the smart city service case scenario, the trust level of the selected models using the proposed algorithm is 0.7%-2.53% less compared the results obtained using ILP. Also in the IIoT service scenario, the trust level of the selected models using the proposed algorithm is 0.49% - 3.17% less compared the results obtained using ILP. These results clearly demonstrate the ability of the proposed heuristic to achieve real-time competitive results in real-life scenarios.
CHAPTER 2

SMART BUILDINGS: A SURVEY ON APPLICATIONS, ANALYTICS AND MACHINE LEARNING TECHNIQUES

Future buildings will offer new convenience, comfort, and efficiency possibilities to their residents. Changes will happen to the way people live and interact with their environments as technology recedes into the background of their lives and information processing is thoroughly integrated into everyday objects and activities. The future of smart buildings includes making the users experience as easy and seamless as possible. The massive streaming data generated and captured by smart building appliances and devices contains valuable information that needs to be mined to facilitate timely actions and better decision making. Machine learning and data analytics will undoubtedly play a critical role to enable the delivery of such smart services.

In this paper, we survey the area of smart building with a special focus on the role of machine learning and data analytic techniques in this context. This survey also reviews the current trends and challenges faced in the development of smart building services.

2.1 ML Background for SBs

Massive data generated from sensors, wearable devices, and other IoT technologies provide rich information about the context of users and building status and can be used to design SB management. This context information is needed to extract useful and interesting insights for various stakeholders. When the data volume is very high, developing predictive models using traditional
approaches does not provide accurate insight and we require newly developed tools from big data. Big data is primed to make a big impact in SBs and is already playing a big role in the architecture, engineering, and construction (AEC) industries [23], notably for waste analytics [24] and waste minimization [25].

ML is a powerful tool that enables us to crunch petabytes of data and make sense of a complicated world. ML algorithms apply a model on new data by learning the model from a set of observed data examples called a training set. For example, after being trained on a set of sample accelerometer data marked as walking or jogging, a ML algorithm can classify the future data points into walking and jogging classes. ML makes it relatively easy to develop sophisticated software systems without much effort on the human side. They are applicable to many real-life problems in SB environments. One can also design and develop self-learning and collaborative systems.

ML does not remove the human element from data science—it draws on computers’ strengths in handling big data to complement our understanding of semantics and context. It also only requires training data to learn better features or parameters needed to improve a given system. ML algorithms can be used to make predictions based on data patterns. It gives computers the ability to learn without being explicitly programmed so that they can create algorithms that can learn from and make predictions on data [26][27]. Nest thermostat is an example of a device that leverages data patterns to predict the preferred temperature in a specific room at a certain time of day. Other consumer devices include those that learn from voice patterns (such as Amazon’s Echo, a personal-assistant type device from Amazon) to those that learn from much more complex behavior and activity patterns.
2.2 ML Models

ML techniques have been widely used to develop smart systems which can sense and react according to context modifications in SBs [4]. There are many different ML algorithms, according to the two well-known theorems No Free Lunch theorem and Ugly Duckling theorem there are no algorithms that can be said to be better than any other. According to the No Free Lunch theorem any two algorithms may perform equally well in solving a problem if there is no prior information about the problem. While Ugly Duckling theorem states it is impossible to say that any two different patterns would be more similar to each other than any other pairs [5].

Mainly, ML has branched into four categories dealing with different types of learning tasks as follows: Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning (RL) algorithms Figure 2.1 shows ML styles. These categories are described next and a summarized comparison between these ML techniques is presented in Table 2.1.
2.2.1 Supervised Learning

Supervised Learning refers to developing algorithms based on a labeled training data set, from which the learner should generalize a representation by building the system model representing the learned relation between the input, output and system parameters. A model is prepared through a training process, the process continues until the model reaches the desired level of accuracy on the training data [28], [29]. Some of the best known supervised ML algorithms are: Naive Bayes model, decision tree, Linear discriminant functions such as support vector machines (SVMs), artificial neural networks (ANNs), hidden Markov models (HMMs), instance-based learning such as k-nearest-neighbor learning, ensembles (bagging, boosting, random forest), logistic regression, genetic algorithms and logistic regression [30] [31]. Supervised learning methods are widely used in smart environments to solve several problems.

Application in SBs: Boger et al. [26] proposed a supervised learning system based on Markov decision processes to guide the people with dementia the process of hand washing. Altun et al. [32] make a comparative study on the supervised human activity classification approaches using body-worn miniature inertial and magnetic sensors. Mozer [33] developed the occupant comfort control of home environment system using neural networks and reinforcement learning to control air heating, lighting, ventilation, and water heating in the smart home environment. Bourobou et al. [3] proposed a hybrid approach using the neural network algorithm and K-pattern clustering to recognize and predict user activities in the smart environments. Hsu et al. [34] proposed a TV recommendation system using neural network model based on user personalized properties such as activities, interests, moods, experiences, and demographic information data. Fleury et al. [35]
proposed health smart home system using SVM algorithm to classify daily living activities based on the data from the different sensors.

Supervised learning problems can be further grouped into classification, regression, time series, and ensemble method problems.

**Classification**

In classification problems the task is to classify an instance into one of a discrete set of possible categories. Mathematically, the classification algorithm can be defined as follows: Given a set of labeled data and a set of unlabeled data, the set of labeled data is used to train the classifier (i.e., the hyperline or prediction function) while the set of unlabeled data will then be classified by the classifier. To evaluate the quality of the classification results, an intuitive way is to count the number of test patterns that are assigned to the right groups, which is also referred to as the accuracy rate (AR) defined by [36].

\[
AR = \frac{N_c}{N_t} \tag{2.1}
\]

Where \( N_c \) denotes the number of test patterns correctly assigned to the groups to which they belong; \( N_t \) the number of test patterns. To measure the details of the classification results, the so-called precision (\( P \)) and recall (\( R \)) are commonly used. Given that the four possible outcomes are true positive (\( TP \)), false negative (\( FN \)), false positive (\( FP \)), and true negative (\( TN \)), the precision (\( P \)) and recall (\( R \)) are generally defined as:

\[
P = \frac{TP}{TP + FP} \tag{2.2}
\]
respectively. Given $P$ and $R$, a simple way to represent the precision and recall of the overall classification results, called F-score or F-measure, is defined as:

$$F = \frac{2PR}{P + R}$$

(2.3)

Typical algorithms for classification are: decision trees, SVM, rule-based induction, neural networks, deep learning, memory-based reasoning, and Bayesian networks [37].

**Decision Tree Algorithms** Decision tree methods is one of the predictive modeling approaches in ML. It constructs a model of decisions made based on actual values of attributes in the data. Decision trees are trained on data for classification and regression problems. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels [38].

The decision trees that the target variable takes continuous values called regression trees. Decision trees are often one of the favorites of ML algorithms because of its speed and accuracy. The most popular decision tree algorithms are [39]: Classification and Regression Tree (CART), Iterative Dichotomiser 3 (ID3), C4.5 and C5.0 (different versions of a powerful approach), Chi-squared Automatic Interaction Detection (CHAID), Decision Stump, M5, Conditional Decision Trees.

**Application in SBs:** Delgado et al. [40] propose a machine-learning technique based on decision trees to mine the most frequent actions of a behavior and their temporal relationship to provide a fast detection of the human behavior in a smart environment. Viswanathan et al. [41] introduce a prototype distributed data mining system for healthcare environment using C4.5 classification algorithm that can provide the patient monitoring and health services. Decision trees algorithm are
non-parametric algorithm and easy to interpret and explain. Their main disadvantage is that they easily overfit.

**Bayesian Algorithms** Bayesian methods apply Bayes’ Theorem for problems such as classification and regression. The most popular Bayesian algorithms are [42]: Naive Bayes, Gaussian Naive Bayes, Multinomial Naive Bayes, Averaged One-Dependence Estimators (AODE), Bayesian Belief Network (BBN), Bayesian Network (BN).

*Application in SBs:* Parnandi et al. [43] propose an indoor localization approach based on Naive Bayes classification and dynamic time warping, they exploit the embedded sensors of smartphones to determine the building that the user entered and the activities that the user is performing inside the building. Verbert et al. [44] proposed a model-based Bayesian network approach to fault diagnosis in HVAC systems. The model has been constructed based expert knowledge regarding component interdependencies and conservation laws and historical data using virtual sensors Naive Bayes classifiers have been used with promising results for activity recognition in [45], [46]. Naive Bayes classifiers identify the activity that corresponds with the greatest probability to the set of sensor values that were observed.

**Support Vector Machine (SVM)** SVM is a supervised ML algorithm which can be used both for classification and regression problems though mostly used in classification challenges [88]. SVM is among the most widely used for many statistical learning problems, such as spam filtering, text classification, handwriting analysis, face and object recognition, and countless others [89]. The basic concept of SVM is the optimal separating hyperplane between the two classes by maximizing the margin between the classes’ closest points. The points lying on the boundaries are called
<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Algorithms</th>
<th>Pros</th>
<th>Cons</th>
<th>Applicability in SBs</th>
<th>Cited</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>Classification</td>
<td>Neural networks</td>
<td>Requires little statistical training; Can detect complex nonlinear relationships</td>
<td>computational burden; prone to overfitting; Picking the correct topology is difficult; Training can take a long time;</td>
<td>Used for classification, control and automated home appliances, next step/action prediction.</td>
<td>[47][48][49][50][51][52]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SVM</td>
<td>Can avoid overfitting using the regularization; expert knowledge using appropriate kernels</td>
<td>Computationally expensive; Slow; Choice of kernel models and parameters sensitive to overfitting</td>
<td>Classification and regression problems in SBs such as activity recognitions, human tracking, energy efficiency services</td>
<td>[53][54][55][35][56]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bayesian networks</td>
<td>Very simple representation does not allow for rich hypotheses</td>
<td>You should train a large training set to use it well.</td>
<td>Energy management system and human activity recognition.</td>
<td>[57][44][45][46]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Decision trees</td>
<td>Non-parametric algorithm that is easy to interpret and explain.</td>
<td>Can easily overfit</td>
<td>Patient monitoring, health care services, awareness and notification services.</td>
<td>[40][41]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hidden Markov</td>
<td>Flexible generalization of sequence profiles; Can handle variations in record structure</td>
<td>Requires training using annotated data; Many unstructured parameters</td>
<td>Daily living activities recognition classification</td>
<td>[58][59][60][61]</td>
</tr>
<tr>
<td></td>
<td>Deep Learning</td>
<td>Orthogonal matching pursuit</td>
<td>Fast</td>
<td>Can go seriously wrong if there are severe outliers or influential cases</td>
<td>Test regression problem such as energy efficiency services in SBs.</td>
<td>[68]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>clustered-based</td>
<td>Straightforward to understand and explain, and can be regularized to avoid overfitting.</td>
<td>It is not naturally flexible enough to capture more complex patterns</td>
<td>Gesture recognition.</td>
<td>[69]</td>
</tr>
<tr>
<td></td>
<td>Ensemble methods</td>
<td>N/A</td>
<td>Increased model accuracy as the number of models increases.</td>
<td>Difficulties in interpreting decisions; Large computational requirements.</td>
<td>Human activity recognition and Energy efficiency services.</td>
<td>[70][71][72]</td>
</tr>
<tr>
<td></td>
<td>Time series</td>
<td>N/A</td>
<td>Can model temporal relationships; Applicable to settings where traditional between-subject designs are impossible or difficult to implement</td>
<td>Model identification is difficult; Traditional measures may be inappropriate for TS designs; Generalizability cannot be inferred from a single study.</td>
<td>Occupant comfort services and energy efficiency services in SBs.</td>
<td>[73][74][75]</td>
</tr>
<tr>
<td>Unsupervised</td>
<td>Clustering</td>
<td>KNN</td>
<td>Simplicity; Sufficient for basic problems; Robust to noisy training data.</td>
<td>High computation cost; Lazy learner</td>
<td>Human activity recognition.</td>
<td>[76]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K-pattern clustering</td>
<td>Simple; Easy to implement and interpret; Fast and computationally efficient</td>
<td>Only locally optimal and sensitive to initial points; Difficult to predict K-Value.</td>
<td>Predict user activities in smart environments.</td>
<td>[3]</td>
</tr>
<tr>
<td></td>
<td>Others</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Semi-Supervised</td>
<td></td>
<td>N/A</td>
<td>Overcome the problem of supervised learning having not enough labeled data.</td>
<td>False labeling problems and incapable of utilizing out-of-domain samples.</td>
<td>Provide context aware services such as health monitoring and elderly care services.</td>
<td>[79][80][81][82][83]</td>
</tr>
<tr>
<td>Reinforcement</td>
<td></td>
<td>N/A</td>
<td>Uses “deeper” knowledge about domain</td>
<td>Must have a model of environment and where actions lead in order to evaluate actions</td>
<td>Lighting control services and learning the occupants, preferences of music and lighting services.</td>
<td>[84][85][86][87]</td>
</tr>
</tbody>
</table>
support vectors, and the middle of the margin is optimal separating hyperplane [90].

**Application in SBs:** Fu et al. [53] proposed a SVM method to predict the system level electricity loads of public buildings that have electricity sub-metering systems. A real-time human tracker system proposed Nguyen et al. [54] using SVM for predicting and recognizing human motion based on the input images from a network of four cameras in the ubiquitous smart homes. Petersen et al. [55] developed a SVM model to identify periods where visitors are present in the home using only the data provided by wireless motion sensors in each room. Fleury et al. [35] presented a study for automatic recognition of activities of daily living in a smart home based on SVM. They collected the data from various sensors such as Infra-Red Presence Sensors, door contacts, temperature and hygrometry sensor, and microphones. Das et al. [56] proposed a one-class classification approach for a real-time activity error detection in smart homes using one-class SVM. Zhao et al. [91] proposed a ML approach based on SVM and RNN to detect the occupancy behavior of a building through the temperature and heating source information for the energy efficiency consumption purposes.

**Artificial Neural Network Algorithms (ANNs)**  ANN models are inspired by the function of biological neural networks. ANN models are classes of pattern matching that are commonly used for regression and classification problems. The most popular artificial neural network algorithms are [42]: Perceptron, Back-Propagation, Hopfield Network, and Radial Basis Function Network (RBFN).

ANNs offer a number of advantages including requiring less statistical training, ability to implicitly detect complex nonlinear relationships between the predictor and response variables, abil-
ity to detect all possible relationships between predictor variables and the availability of multiple training algorithms [52]. On the other hand, disadvantages include its “black box” nature, greater computational burden and proneness to overfitting. However, due to the inherent features of neural networks, their main limitations are difficulty in training the network with no local optima, its later adaptation to changes in the behavior, the validation of the results, and the interpretation of the network performance.

*Application in SBs:* Badlani and Bhanot [47] developed an energy-efficient smart home system by using pattern recognition based on ANNs, the system comprises of a Recurrent Neural Network to capture Human behavior patterns and a Feed Forward Architecture in ANN for security applications in the smart homes. Other researchers have kept using ANNs in order to provide personalized services. Campo et al. [48] developed a system that calculated the probability of occupation for each area of the house and systematically compared the probability with the current situation. See [49] for a survey focused on ANNs for Smart Homes. Ermes et al. [50] used a hybrid classifier combining a tree structure containing a priori knowledge and ANN to recognize the activities such as rowing, biking, playing football, walking, running, sitting, or hiking. Ciabattoni et al. [51] proposed a home energy management system design using the neural network algorithm to forecast the power production of the photovoltaic plant and the household consumptions over a determined time horizon.

**Deep Learning Algorithms** Deep learning methods represent an evolved form of ANNs in which a deep architecture (many layers composed of multiple linear and non-linear transformations [92]) is used. One of the promises of deep learning is replacing handcrafted features with efficient
algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction. The most popular deep learning algorithms are [93]: Deep Boltzmann Machine (DBM), Deep Belief Networks (DBN), Convolutional Neural Network (CNN), Stacked Auto-Encoders. Deep learning has been used successfully in all types of big data analytics applications, especially natural language processing (NLP), machine language translation, medical diagnosis, stock market trading signals, network security and image identification.

Deep learning is now ubiquitously used in major businesses and companies. Microsoft research on a deep learning system demonstrated real-time speech translation between Mandarin Chinese and English [94]. Apple’s Siri uses a deep learning trained model, and the voice recognition in the Google Android phone also uses a deep learning trained model [95]. Deep learning utilizes a number of mechanisms such as convolutions and drop-out that allows them to efficiently learn from high-dimensional data. However, deep learning still requires much more data to train compared to other algorithms because the models have orders of magnitudes more parameters to estimate.

Application in SBs: Choi et al. [62] propose two prediction algorithms deep belief network and restricted Boltzmann machines based on the deep learning framework for predicting various activities in a home. They also presented a hybrid model which combines for predicting human behavior. The paper [63] proposes a generic deep learning framework based on convolutional and Long-short-term memory recurrent neural networks for human activity recognition that is suitable for multimodal wearable sensors, such as accelerometers, gyroscopes or magnetic field sensors. Alsheikh et al. [64] proposed a hybrid approach of deep learning and hidden Markov model for human activity recognition using triaxial accelerometers. Baccouche et al. [65] propose a two-steps neural-based deep model to classify human activities, the first step of the model is automat-
ically learned spatiotemporal features based on Convolutional Neural Networks. Then the second step of the model uses Recurrent Neural Network to classify the entire sequence of the learned features for each time-step. In [66], they propose an acceleration-based human activity recognition method using Convolution Neural Network. In [67] a deep convolutional neural network as the automatic feature extractor and classifier for recognizing human activities is proposed using the accelerometer and gyroscope on a smartphone. Hammerla et al. [96] explore the performance of deep, convolutional, and recurrent approaches of deep learning for human activity recognition using wearable sensors. For the sake of measuring the performance they used three representative datasets that contain movement data captured with wearable sensors.

**Hidden Markov models (HMM)** An HMM is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols.

*Application in SBs:* Wu et al. [61] proposed an improved hidden Markov model to predict personalized user behaviors to provide services for people with disabilities. They developed a temporal state transition matrix to replace the fixed state transition matrix. Lv and Nevatia [60] used hidden Markov models for both automatic recognition and segmentation of 3-D human action to enable real-time assessment and feedback for physical rehabilitation. Cheng et al. [58] proposed an inference engine based on the hidden Markov model that provides a comprehensive activity of daily living recognition capability. They integrated both Viterbi and BaumWelch algorithms to enhance the accuracy and learning capability. Chahuara et al. [59] proposed a sequence-based models for on-line activities of daily living recognition in SBs environment. They presented three
of sequence-based models: HMM, conditional random fields and a sequential Markov logic network.

**Time Series Analysis**  A time series is a collection of temporal data objects; the characteristics of time series data include large data size, high dimensionality, and updating continuously [97]. One of the major reasons for time series representation is to reduce the dimension, and it divides into three categories: model based representation, non-data-adaptive representation, and data adaptive representation [98] [99].

Application in SBs: Survadevara et al. [73] proposed a wellness model using seasonal auto regression integration moving average time series with sleeping activity scenario in a smart home environment to forecast the elderly sleeping tendency. Zhou et al. [74] proposed a statistical time series analysis framework to examine causal relationships among time series that are highly non-stationary in the case of data sensors in SBs. Jakkula and Cook [75] propose a framework to derive temporal rules from a time series representation of observed inhabitant’s physical and instrumental activities in a smart home, for improving the prediction of events based on observed temporal relations in a smart home environment.

**Regression**  In regression problems the aim is to approximate a real-valued target function. It is concerned with modeling the relationship between variables that is iteratively refined using a measure of error in the predictions made by the model [100]. The most popular regression algorithms are [101]: Ordinary Least Squares Regression (OLSR), Linear Regression, Logistic Regression, Stepwise Regression, Multivariate Adaptive Regression Splines (MARS), Locally Estimated Scatterplot Smoothing (LOESS).
**Application in SBs:** Chen et al. [68] used the orthogonal matching pursuit algorithm which is one of regression techniques, to identify the physical and environmental parameters that providing the energy efficiency in a SB. Bouchard et al. [69] presented a gesture recognition system using linear regression combined with the correlation coefficient to recognize the gesture direction and estimate the segmentation of continuing gestures of daily usage activities in the smart environment.

**Ensemble methods** A combination of multiple classifiers often referred to as a classifier ensemble, group of classification models that are independently trained and whose predictions are combined in some way to make the overall prediction [102]. The most popular ensemble learning based classification techniques are [103]: Boosting, Bootstrapped Aggregation (Bagging), AdaBoost, Stacked Generalization (blending), Gradient Boosting Machines (GBM), Gradient Boosted Regression Trees (GBRT), Random Forest.

**Application in SBs:** Jurek et al. [70] proposed a cluster-based ensemble approach solution for activity recognition within the application domain of smart homes. With this approach, activities are modeled as collections of clusters built on different subsets of features. Fatima et al. [71] proposed an ensemble classifier method for activity recognition in smart homes using genetic algorithm optimization to combine the measurement level output of different classifiers that makes up the ensemble. They used the ANN, hidden Markov model, Conditional Random Field (CRF), SVM) [104] as base classifiers for activity recognition. Guan and Ploetz [72] proposed a deep LSTM ensemble method for activity recognition using wearables, they developed modified training procedures for LSTM networks and combine sets of diverse LSTM learners into classifier collectives.
2.2.2 Unsupervised Learning

Unsupervised Learning refers to developing algorithms that uses data with no labels to analyze the behavior or the system being investigated [105]. Thus, the algorithm does not know anything about the correctness of the outcome. In other words unsupervised learning algorithm classifies the sample sets to different clusters by investigating the similarity between the input samples. Clustering is done using different parameters taken from the data which enable us to identify correlations which are not so obvious. A model is prepared by deducing structures present in the input data to extract general rules. It may through a mathematical process systematically reduce redundancy, or organize data by similarity [77].

The use of an unsupervised approach was applied for different activities recognition in smart spaces when it is difficult to have labels for the data [78]. Some of common problems of unsupervised learning are clustering, dimensionality reduction and association rule learning. There are quite a large amount of algorithms commonly used in unsupervised learning, some of which are based on supervised-learning algorithms: the Apriori algorithm and k-Means. In unsupervised learning, there is no outcome measure; we observe only the features and the goal is to describe the associations and patterns among a set of input measures [28].

The major disadvantage of unsupervised learning is the absence of direction for the learning algorithm and that the absence of any interesting knowledge discovered in the set of features selected for the training. Clustering is a form of unsupervised learning that consists of finding patterns in the data by putting each data element into one of K-clusters, where each cluster contains data elements most similar to each other [106]. Unsupervised learning problems can be further grouped
into clustering and association problems, which are described next.

**Clustering**

A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior. Clustering methods are typically organized by the modeling approaches such as centroid-based and hierarchical. All methods are concerned with using the inherent structures in the data to best organize the data into groups of maximum commonality [107]. The quality of the clustering result is evaluated depends on the application to which a clustering algorithm is applied. For instance, the sum of squared errors (SSE) is widely used for data clustering while the peak-signal-to-noise ratio (PSNR) is widely used for image clustering [36]. The most popular clustering algorithms are [101]: k-Means, k-Medians, Expectation Maximization (EM), Hierarchical Clustering.

*Application in SBs:* Fahad et al. [76] propose an activity recognition approach that combines the classification with the clustering, in their approach the activity instances are clustered using Lloyd’s clustering algorithm. Then, they apply the learning method Evidence Theoretic K-Nearest Neighbors that combines KNN with the Dampster Shafer Theory of evidence. The paper [3] proposes a hybrid approach to recognize and predict user activities in the smart environment. They use K-pattern clustering algorithm to classify so varied and complex user activities, and ANN to recognize and predict user activities inside his/her personal space. Lapalu et al. [30] used an unsupervised learning approach to address the problem of learning the activities of daily living in smart home for cognitive assistance to an occupant suffering from some type of dementia, such as Alzheimer’s disease, using Flocking algorithm for clustering analysis. Aicha et al. [31] present
an unsupervised learning approach for detecting abnormal visits of an elderly in the smart home environment based on a Markov modulated Poisson process model. The model combines multiple data streams, such as in the front-door sensor transitions and the general sensor transitions. With respect to handling social interaction, Cook et al. [77] applied an unsupervised learning algorithm to detect social interaction and monitor activity daily living in the smart environment that can adapt to the changes in discovered patterns based on the occupant implicit and explicit feedback and can automatically update its model to reflect the changes. Rashidi et al. [78] introduce an unsupervised method that identifies and tracks the frequent activities that naturally occur in an individual’s routine in a smart environment. The activity discovery method of the system is designed to cluster the sequences based on the simple k-means algorithm. Fiorini et al. [108] proposed an unsupervised ML approach to identify the behavioral patterns of the occupants using unannotated data collected from low-level sensors in an SB. Their methodology included processing and analyzing sensor data from 17 older adults living in community-based housing to extract activity information at different times of the day.

**Association**

An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy $X$ also tend to buy $Y$. Association analysis are rules created by analyzing data for frequent if/then patterns and using the criteria support and confidence to identify uncover relationships between seemingly unrelated data in a relational database or other information repository. Support indicates how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true. Many
algorithms for generating association rules have been proposed. Apriori algorithm is most well-known algorithm [109].

*Application in SBs:* Aztiria et al. [110] proposed learning frequent patterns of user behavior system using association, workflow mining, clustering, and classification techniques. The core part of the system is the learning layer which is made up of two modules: the language module, which provides a standard conceptualization of the patterns; and the algorithm module, which discovers the patterns. Kang et al. [111] proposed a service scenario generation scheme for translating association rules mined from the statuses of all appliances that are collected periodically at a specific time interval in a smart home environment. Nazerfard et al. [112] propose a framework for discovering and representing temporal aspects of activity patterns, including temporal ordering of activities and their usual start time and duration using temporal association rule mining techniques in smart home.

2.2.3 Semi-Supervised Learning

Semi-Supervised learning lies between supervised and unsupervised methods. Input data is a mixture of labeled and unlabeled examples. These hybrid algorithms aim to inherit the strengths of these main categories, while minimizing their weaknesses. The model has to learn structures present in the data and also make predictions. Example problems are classification and regression [113].

There are some popular semi-supervised learning models, including self-training, help-training, mixture models, co-training and multi-view learning, graph-based methods and semi-supervised support vector machines, Generative models, Heuristic approaches [42].
Application in SBs: Cook [79] combined between fully-supervised and semi-supervised learning to recognizing and track ADL activities and to provide context-aware services, such as health monitoring and assistance to individuals experiencing difficulties living independently at different smart spaces. Liu et al. [80] present a vision based semi-supervised learning mechanism for detecting falls and other ADL in smart environments to overcome the exhaustive labeling of human activities by automatic annotating the activities with the highest confidence. Fahmi et al. [81] present a semi-supervised fall detection method. They first train a supervised algorithm using decision trees, and then they use profiles to develop a semi-supervised algorithm based on multiple thresholds. Radu et al. [82] present semi-supervised ML method using only the low power sensors on a smartphone to consider the problem of determining whether a user is indoors or outdoors. Guan et al. [83] propose a semi-supervised learning algorithm for activity recognition named EnCo-training to make use of the available unlabeled samples to enhance the performance of activity learning with a limited number of labeled samples. The proposed algorithm extends the co-training paradigm by using ensemble method.

2.2.4 Reinforcement Learning

Reinforcement learning is a learning paradigm concerned with learning to control a system so as to maximize a numerical performance measure that expresses a long-term objective [114]. Reinforcement learning is an area of ML inspired by behaviorist psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. RL algorithms learn more control policies, especially in the absence of a priori knowledge and a sufficiently large amount of training data. However, they suffer from a major drawback:
high calculation cost because an optimal solution requires that all states be visited to choose the optimal one. The well-known approaches of RL are: Brute force, Monte Carlo methods, Temporal difference methods and Value function approaches [115]. Q-learning [116] is a model-free reinforcement learning method based on learning the expected utility given a state decision.

Application in SBs: Mozer [84] applied Q-learning for lighting regulation to predict when lights in a home will be turned on or off in order to schedule activations of lights in a home to conserve energy. Li and Jayaweera [85] proposed a Q-learning based approximate dynamic programming algorithm to provide a more efficient, flexible and adaptive way for on-line decision making in “smart-homes” that enable customers to make optimal sequential decisions to maximize their own profits based on both local fully observable information and the estimated hidden information of the environment. Khalili and Aghajan [86] proposed temporal differential class of reinforcement learning method for autonomous learning of user’s preference of music and lighting service settings in presence of different states of the user in SB environment. The preferences are learned through user’s explicit or implicit feedback to the system when the user opts to react to the provided service. Xu et al. [87] give a survey of developments in RL algorithms with function approximation. They evaluated and compared different RL algorithms using several benchmark learning prediction and learning control tasks.

2.3 ML Tasks for SBs

In this section, we will describe the major ML tasks that are relevant for SB. The reader is referred to Figure 2.2 for a general depiction of ML tasks in SBs and the steps taken to implement ML in an SB environment.
2.3.1 Data Collection and Acquisition

A variety of technologies are used for data collection, each of which has different tradeoffs in terms of capabilities, energy efficiency, and connectivity. Sensors and similar objects in SBs produce raw data simultaneously in an automated way and such devices may store the data for a certain time interval or report it to governing components [2]. Data may be collected at gateways within the network; the collected data is further filtered and processed, fused into compact forms for efficient transmission. A wide variety of communication technologies such as Zigbee, Wi-Fi and cellular are used to transfer data to collection points.

Data gathered from a global-scale deployment of smart-things, are the base for making intelligent decisions and providing services. If data are of poor quality, decisions are likely to be unsound
Zhao et al. [118] propose a data acquisition and transmission system which could be used for monitoring systems to collect energy consumption data (e.g., electricity, water, gas, heating, etc.) from terminal meters which are installed in buildings. The system stores the data periodically after analyzing and processing, and finally transmits the data to servers through Ethernet. Rowley et al. [119] propose the data acquisition and modeling approaches that can support the delivery of building energy infrastructure in both new building and renovated real-world contexts. Such methods provide a means to achieve short, medium and long term forecasting of possible scenario pathways to multi objective sustainable outcomes.

*CLEEN MMEA* [120] platform handles the collection and processing of data and initiates situational knowledge extraction. The aim is to establish an online market place where services and data from different companies can meet. The interfaces are made public so that any company can easily join the network to buy or sell services. The analysis results can be given to an energy services company which can then make a service offer to the owners.

A typical example of open access data collection system is *e3Portal* [121] developed by VTT in collaboration with Finnish municipalities. e3Portal offers information and tools when planning savings measures and energy retrofitting in municipal buildings. In addition, it includes continuously updated data about energy and water consumption in thousands of municipal buildings like schools, kindergartens, offices, hospitals, other health care facilities, etc. Decision makers, designers, operation and maintenance personnel as well as users of buildings can utilize it.

There are projects that provide publicly available SB datasets for researchers to conduct further studies; A list of “Home Datasets” [122] includes the datasets collected by projects from UC
Berkeley, MIT, Washington State University, University of Amsterdam, University of Edinburgh and University of Tokyo. The WARD [123] project supported by NSF TRUST Center at UC Berkeley provides a benchmark dataset for human action recognition using a wearable motion sensor network. The dataset was collected from 13 repetitive actions by 13 male and 7 female participants between the ages of 19 and 75. An MIT project [124] collected dataset from two single-person apartments about human activity for two weeks. Eighty-four sensors were installed in everyday objects such as drawers, refrigerators, containers, etc. to record opening-closing events. Baos et al. [125] introduced an open benchmark dataset collected from a set of inertial sensors attached to different parts of the body. They considered 33 fitness activities, recorded using 9 inertial sensor units from 17 participants. The CASAS project [126] at Washington State University provides a publicly-available dataset for a three-bedroom apartment with one bathroom, a kitchen, and a living/dining room. The apartment is equipped with motion sensors, digital sensors to provide ambient temperature readings, and analog sensors to provide readings for hot water, cold water, and stove burner use [127]. The PlaceLab project [128] of MIT provides a dataset collected from a one-bedroom apartment with more than 900 sensors, including those coming from motion, switch and RFID sensors. That is being used to monitor activity in the environment in the context of the smart home [129]. A collection of smart meters data from five houses in the United Kingdom (UK) [130]. This dataset includes 400 million raw records the active power drawn by individual appliances and the whole-house apparent power demand every 6 seconds.

The major challenges that arise for data collection are scalability, privacy, security and heterogeneity of resources [131]. Automated sensor data collection process collects a large amount of data that overwhelms the collection and analysis centers in comparison to the data available from
more traditional human-centered sources such as IoT devices and social media. This leads to a huge number of small synchronous write operations to the database storage system, consequently, resulting in serious performance bottlenecks to the storage system design [132]. Because of the widespread use of RFID technology, privacy issues arise in data collection, for example the tags carried by a person may become a unique identifier for that person. Also, other security concerns appear, for example, the radio signals of RFID technology are easily jammed. Hence, that can disrupt the data collection process [133].

The heterogeneity of data that is being collected from different resources is another major challenge, such that the data are usually very noisy, large scale and distributed, so that it is difficult to use the collected data effectively without a clear description of the available data processing [132].

subsectionData Preprocessing Vast and ever-increasing quantities of data are produced by sensors in SBs; this data comes from different sources with varying formats and structures. Usually, this data is not ready for analysis as it might be incomplete or redundant due to low battery power, poor calibration, exposure to various malicious elements and interference. Therefore, raw data typically needs to be preprocessed to handle missing data, discard noisy and redundant data and integrate data from different sources into a unified schema before being committed to storage. This preprocessing is called data cleaning. Data quality can be greatly enhanced by cleaning the data before it reaches its end user [2][134]. Data cleaning is an important task in data processing. It is not a new process specific to the IoT context. It has already been defined as a process for database systems. Providing a data cleaning system would help applications to concentrate on their core logic without worrying about data reliability post-processing overheads [132].
There are many techniques that have been employed to deal with the problem of cleaning noisy data streams such as Kalman filters [135], statistical models [134] and outlier detection models [132]. One of the major challenges with data cleaning techniques in the SBs is the heterogeneity of data sources especially WSN- and RFID-enabled data streams. Such that the techniques used should be able to handle different variables of interest to fulfill IoT applications’ requirements (e.g., adjust home temperature based on observed outer temperature, user habits, energy management, etc.) [117] Any type of failures such as a failed sensor, network issues, camera failure, or database crashes in the process of collecting data would invalidate the data. Consequently, this type of impediment will dramatically increase the time required to collect data [127].

subsection Dimensionality Reduction

There are huge volumes of raw data that may be generated in SBs, due to the heterogeneity and ubiquity of sensors used. Much of the data is highly redundant and can be efficiently brought down to a much smaller number of variables without a significant loss of information. The mathematical procedures making possible this reduction are called dimensionality reduction techniques [136]. The key idea is to find a new coordinate system in which the input data can be expressed with many less variables without a significant error. The dimensionality reduction can be made in two different ways: by extracting of the features that represent the significant data characteristics (this technique is called feature extraction), or by only maintaining the most relevant variables from the original dataset (this technique is called feature selection) [137] [138].

Like clustering methods, dimensionality reduction explores and exploits the internal structure of the data, but in this case in an unsupervised manner using less information. Many of these methods can be adapted for use in classification and regression. Examples about some used algo-
rithms are [101]: Principal Component Analysis (PCA), Principal Component Regression (PCR), and Linear Discriminant Analysis (LDA). Chen et al. [139] propose a framework using the classification information of local geometry of data to reduce the dimensionality of data set on human activity recognition from wearable, object, and ambient sensors.

**Feature Extraction**

The main components of the original data are the features. After extracting the features from the raw dataset, such features contain important information that are used by the learning algorithms for the activities discrimination. The most commonly used approaches of feature extraction operate in three domains: time domain, frequency domain and discrete domain [140]. Among time domain method, mean and standard deviation are the key approaches for almost all sensor types. While the frequency domain method focuses on the periodic structure of sensor data. Wavelet Transformation and Fourier Transform are the most common approaches. And discrete domain methods include Euclidean-based Distances, Dynamic Time Warping and Levenshtein Edit Distance are key approaches used to evaluate string similarity for classifying human activities and modeling behavioral patterns [141], [142].

**Feature Selection**

The main role of feature selection is to discriminate the most relevant subset of features from within a high dimensional feature vector, so that reduces the load of noise and computational expense on the learning models. Principal Component Analysis (PCA), Independent Component Analysis (ICA) and Linear Discriminant Analysis (LDA) are used to map the high dimensional...
feature vector into a lower dimensional one [143].

Hausmann and Ziekow [144] proposed an approach for automatically adapting the feature selection for SBs application ML models from the time-series data based on wrapper methods and genetic optimization. Fahad et al. [145] propose an activity recognition approach for overlapping activities using a learning method Evidence Theoretic K-Nearest Neighbors (ET-KNN) that identifies key features from the information obtained using the sensors deployed in multiple locations and objects. Fang et al. [146] Show that the different feature sets generate different human activity recognition accuracy, and the selection of unsuitable feature data sets increases the computational complexity and degrades the human activity recognition accuracy in smart home environment. The wrapper and filtering are the two main statistical methods of feature selection problem. It is argued that, although the wrapper approach may obtain better performances, filters are faster and are not resource intensive [147]. In [148] different feature selection methods are used to reduce the dimensionality of the learning problem to recognize the human activities from observed sensors. They show that the performance of the learning models to recognize the human activity have a strong relationship with the utilized features.

*Feature projection*

Generally, feature projection can be formulated as a mapping from an original feature space to an appropriate subspace such that a learning criterion is optimized. Feature projection enables high-dimensional feature vectors to be visualized in a low dimension, and allows the distribution of the reduced feature vectors to be analyzed [149]. As a result, the classifier with the best performance can be selected for the reduced feature vectors. Plus, dimensionality reduction of feature projection
reduces the processing time for pattern recognition, and makes real-time implementation possible [147]. Chu et al. [150] proposed a linear supervised feature projection that utilizes a linear discriminant analysis (LDA) for EMG pattern recognition that attempted to recognize nine kinds of hand motion.

2.4 ML Tools and Platforms for SBs

A variety of ML toolkits have been created to facilitate the learning process, with an ever-increasing amount of options, the task of selecting ML tools for big data can be difficult. There is no single toolkit that truly embodies a one-size fits-all solution. The available tools have advantages and drawbacks, and many have overlapping uses. One issue is that ML is a broad field of study and many of the available toolkits require expertise in the areas of programming and system architecture, in addition, many people lack a full understanding of what the various platforms are capable of [151].

The important factors that must be considered when selecting a specific ML tool are scalability, speed, coverage, usability, extensibility, and programming languages support. With respect to the scalability factor, the size and complexity of the data should be considered to determine if a particular toolkit will be appropriate. The processing platform the library is running on and the complexity of the algorithm affects the speed factor. Not all the projects require high speed factor, however, if models require frequent updates, the speed may be a crucial concern. If models do not require frequent updates, speed is not a concern. Coverage represent the number of ML algorithms implemented in the tool. With a growth rate in data production, the challenge faced by the ML community is how to efficiently process and learn from big data.
In general, the available big data tools do not implement all varieties of different classes of ML algorithms, and typically their coverage ranges from a few algorithms to around two dozen. The usability factor include elements such as initial setup, ongoing maintenance, programming languages available, user interface available, amount of documentation, or availability of a knowledgeable user community. The extensibility factor means that the implementations included in the tools can be used as building blocks towards new platforms or systems. It is important to evaluate tools in terms of how well they are able to meet this factor. There are variety of ML libraries that are available in different programming languages. Depending on the task you are trying to accomplish, certain languages, libraries and tools can be more effective than others. The following provides an in-depth look at the strengths and weaknesses of the top used deep learning and ML tools. The reader is also referred to Table 2.2 for a concise tabulated summary of the described deep learning and ML tools.

2.4.1 H2O

\textit{H2O} [152] is an open-source in-memory, distributed, and scalable ML framework for big-data analysis that provides a parallel processing engine, analytics, math, and ML libraries, along with data preprocessing and evaluation tools. It is produced by the start-up H2O.ai (formerly 0xdata), which launched in 2011 in Silicon Valley. The most notable feature of this product is that it provides numerous tools for deep neural networks. The H2O software APIs can be called from Java, Python, R and Scala. Users without programming expertise can still utilize this tool via the web-based User Interface. While H2O maintains their own processing engine, they also offer integrations that allow use of their models on Spark and Storm. H2O’s engine processes data completely in-memory using
multiple execution methods, depending on what is best for the algorithm. The general approach used is Distributed Fork/Join, a divide-and-conquer technique, which is reliable and suitable for massively parallel tasks.

The H2O software can be run on conventional operating-systems: Microsoft Windows (7 or later), Mac OS X (10.9 or later), and Linux (Ubuntu 12.04; RHEL/CentOS 6 or later). It also runs on big-data systems, particularly Apache Hadoop Distributed File System (HDFS) and Spark. It also operates on cloud computing environments, for example using Amazon EC2, Google Compute Engine, and Microsoft Azure. As of July 2016, the algorithms included in H2O cover a range of tasks, including classification, clustering, generalized linear models, statistical analysis, ensembles, optimization tools, data preprocessing options and deep neural networks. On their roadmap for future implementation are additional algorithms and tools from these categories as well as recommendation and time-series.

2.4.2 Apache SINGA

Apache SINGA [152] is a general distributed deep learning platform for training big deep learning models over large datasets. A variety of popular deep learning models are supported, namely feedforward models including convolutional neural networks (CNN), energy models like restricted Boltzmann machine (RBM), and recurrent neural networks (RNN). Many built-in layers are provided for users.

SINGA provides a general architecture to exploit the scalability of different training frameworks. It supports different neural net partitioning schemes to parallelize the training of large models, namely partitioning on batch dimension, feature dimension or hybrid partitioning. SINGA
includes APIs for development in Python and C++. SINGA can be run on Linux operating system [153].

2.4.3 MLlib (Spark)

MLlib [154] is Apache Spark’s ML library. Its goal is to make practical ML scalable and easy. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs. For classification, they have Support Vector Machines, Logistic Regression, Naïve Bayes, Decision Trees, Random Forest, and Gradient-Boosted Trees. Clustering algorithms include k-Means, Gaussian Mixture, and Power Iteration Clustering. They offer implementations for Linear Regression and Isotonic Regression, and one collaborative filtering algorithm using Alternating Least Squares. For dimensionality reduction, they support principal component analysis (PCA). In general, MLlib’s relies on Spark’s iterative batch and streaming approaches, as well as its use of in-memory computation. MLlib includes APIs for development in Scala, Java, Python and SparkR, but not every tool is available in all languages.

2.4.4 TensorFlow

Tensorflow [155] is an open source software library for numerical computation and deep ML in various kinds of perceptual and language understanding tasks using data flow graphs. TensorFlow was originally developed by the Google Brain team for Google’s research and production purposes and later released under the Apache 2.0 open source license on November 9, 2015. TensorFlow is about more than deep learning. TensorFlow actually has tools to support reinforcement learning
and other algorithms. TensorFlow implements what are called data flow graphs, where batches of data ("tensors") can be processed by a series of algorithms described by a graph.

The movements of the data through the system are called "flows"—hence, the name. TensorFlow can run on multiple CPUs and GPUs. It runs on 64-bit Linux or Mac OS X desktop or server systems, and Windows support on roadmap, as well as on mobile computing platforms, including Android and Apple’s iOS. TensorFlow is written with a Python API over a C/C++ engine that makes it run fast. TensorFlow uses symbolic graph of vector operations approach, specifying a new network is fairly easy. However, TensorFlow has a major weakness in terms of modeling flexibility. Every computational flow has been constructed as a static graph. That makes some computations difficult, such as beam search.

2.4.5 Torch

*Torch* [156] is an open source ML library, a scientific computing framework with wide support for ML algorithms that puts GPUs first. Torch was originally developed at NYU. It is easy to use and efficient, thanks to a script language based on the Lua programming language and an underlying C/CUDA implementation, which was designed to be portable, fast, extensible, and easy to use in development. Some version of it is used by large tech companies such as Google DeepMind, the Facebook AI Research Group, IBM, Yandex and the Idiap Research Institute. Torch has been extended for use on Android and iOS. Torch features a large number of community-contributed packages, giving Torch a versatile range of support and functionality. It provides a wide range of algorithms for deep ML, computer vision, signal processing, parallel processing, image, video, audio and networking among others, and builds on top of the Lua community [157].

41
2.4.6 Deeplearning4j

*Deeplearning4j* [158] is an open source distributed deep learning library, primarily developed by a ML group in San Francisco led by Adam Gibson. It is written for Java and the Java Virtual Machine and a computing framework with wide support for deep learning algorithms. Deeplearning4j composable, meaning shallow neural nets such as restricted Boltzmann machine, deep belief net, convolutional nets, recurrent nets, deep autoencoder, stacked denoising autoencoder and recursive neural tensor network. These algorithms all include distributed parallel versions that integrate with Hadoop and Spark. Deeplearning4j relies on the widely used programming language, Java—though it is compatible with Clojure and includes a Scala API. Deeplearning4j is used in real-world applications such as fraud detection for the financial sector, anomaly detection in industries such as manufacturing, recommender systems in e-commerce and advertising, and image recognition. Deeplearning4j is designed to be used in business environments, rather than as a research tool.

2.4.7 Massive Online Analysis (MOA)

*MOA* [159] is one of the popular open source framework for data stream mining, with an active growing community. MOA is written in Java related to the WEKA project that developed at the University of Waikato, New Zealand. It includes a set of learners and stream generators that can be used from the GUI, the command-line, and the Java API. It includes a collection of ML algorithms (classification, regression, clustering, outlier detection, concept drift detection and recommender systems) and tools for evaluation [160].
Caffe [161] is a deep learning framework made with expression, speed, and modularity in mind. It is a widely used machine-vision library that ported Matlab’s implementation of fast convolutional nets to C and C++. It is developed by the Berkeley Vision and Learning Center (BVLC) and by community contributors. In Caffe, multimedia scientists and practitioners have an orderly and extensible toolkit for state-of-the-art deep learning algorithms, with reference models provided out of the box. While Caffe was first designed for vision, it has been adopted and improved by users in speech recognition, robotics, neuroscience, and astronomy. For rapid prototyping and interfacing with existing research code, Caffe provides Python and MATLAB bindings. Caffe performs image classification with convolutional nets, which represent the state of the art. Caffe is chiefly used as a source of pre-trained models hosted on its Model Zoo site.

Caffe is useful for performing image analysis (Convolutional Neural Networks, or CNNs) and regional analysis within images using convolutional neural networks (Regions with Convolutional Neural Networks, or RCNNs). Speed makes Caffe perfect for research experiments and industry deployment. Caffe can process over 60M images per day with a single NVIDIA K40 GPU. Caffe has already been used in a large number of research projects at UC Berkeley and other universities, achieving state-of-the-art performance on a number of tasks object classification, object detection, and Learning Semantic Features. It provides a complete toolkit for training, testing, finetuning, and deploying models, with well-documented examples for all of these tasks. The Caffe framework benefits from having a large repository of pre-trained neural network models suited for a variety of image classification tasks, called the Model Zoo [162]
2.4.9 Azure ML

Microsoft first launched *Azure ML* [163] as a preview in June 2014. Azure ML allows Microsoft Azure users to create and train models, then turn them into APIs that can be consumed by other services. Users get up to 10GB of storage per account for model data, although you can also connect your own Azure storage to the service for larger models. With the Azure service, programmers can use either the R programming language or Python. ML algorithms may be purchased in the Microsoft Azure Marketplace, or obtained for no cost in a new community gallery Microsoft has created for users to share their formulas. It shares many of the real-time predictive analytics of the new personal assistant in Windows Phone called Cortana. Azure ML also uses proven solutions from Xbox and Bing.

Azure now offers the ability to run Hadoop over Ubuntu Linux. This feature provides a way for Hadoop deployments that use Linux scripts to run on Azure. Azure also offers a hosted version of Storm, open source software for analyzing data streams. Azure allows developers to easily connect .Net and Java libraries to Storm. Azure ML studio has a number of modules ranging from data ingress functions to training, scoring, and validation processes. Azure ML comes with a large library of algorithms for predictive analytics. The popular families of algorithms are: regression, anomaly detection, clustering, and classification.

2.4.10 Amazon ML

*Amazon ML* [164] is a service provided by Amazon that uses ML technology, aiming to help developers makes it easy for developers of all skill levels to use ML technology and to build predictive applications, such as fraud detection systems. Amazon ML provides visualization tools
and wizards that guide you through the process of creating ML models without having to learn complex ML algorithms and technology.

Amazon ML can process free-form feedback from your customers, including email messages, comments or phone conversation transcripts, and recommend actions that can best address their concerns. For example, you can use Amazon ML to analyze social media traffic to discover customers who have a product support issue, and connect them with the right customer care specialists. Amazon ML connects to data stored in Amazon S3, Redshift, or RDS, and can run binary classification, multiclass categorization, or regression on said data to create a model. To train binary classification models, Amazon ML uses logistic regression algorithm. For training multiclass models, multinomial logistic regression algorithm. For training regression models, Amazon ML uses linear regression algorithm.

2.5 Real-time Big Data Analytics Tools for SBs

Several applications need to have real-time data analysis for stream data. Waiting the information to be archived and then analyzed is not practical for these type of applications. Stream processing is designed to analyze and act on real-time streaming data, using “continuous queries” (i.e., SQL-type queries that operate over time and buffer windows) to handle high volume in real-time with a scalable, highly available and fault tolerant architecture. Essential to stream processing is Streaming Analytics. More and more tools offer the possibility of real-time streaming data. The following gives an overview about well-known and widely adopted options.
Table 2.2: Comparison between deep learning and ML tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Creator</th>
<th>OS</th>
<th>Open source?</th>
<th>Written In</th>
<th>Interface</th>
<th>CUDA support?</th>
<th>Algorithms</th>
<th>Release date</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>Google Brain team</td>
<td>Linux, Mac OS X (Windows support on roadmap)</td>
<td>Yes</td>
<td>C++, Python</td>
<td>Python, C/C++</td>
<td>Yes</td>
<td>deep learning algorithm: RNN, CNN, RBM and DBN</td>
<td>Nov. 2015</td>
</tr>
<tr>
<td>Theano</td>
<td>Université de Montréal</td>
<td>Cross-platform</td>
<td>Yes</td>
<td>Python</td>
<td>Python</td>
<td>Yes</td>
<td>deep learning algorithm: RNN, CNN, RBM and DBN</td>
<td>2007</td>
</tr>
<tr>
<td>SINGA</td>
<td>Apache Incubator</td>
<td>Linux</td>
<td>Yes</td>
<td>Python, C++</td>
<td>Python, C++</td>
<td>Yes</td>
<td>deep learning algorithm: CNN, RBM, RNN</td>
<td>Oct., 2015</td>
</tr>
<tr>
<td>H2O Deep Learning</td>
<td>H2O.ai</td>
<td>Linux, Mac OS, Microsoft Windows And Cross-platform inc</td>
<td>Yes</td>
<td>Java, Scala, Python, R</td>
<td>Python, R</td>
<td>No</td>
<td>classification, clustering, generalized linear models, statistical analysis, ensembles, optimization tools, data preprocessing options and deep neural networks.</td>
<td>2011</td>
</tr>
<tr>
<td>Deeplearning4j</td>
<td>Adam Gibson</td>
<td>Linux, OSX, Windows, Android, Cyanogen-Mod</td>
<td>Yes</td>
<td>Java, Scala, C, CUDA</td>
<td>Java, Scala, Clojure</td>
<td>Yes</td>
<td>Deep learning algorithms including: RBM, DBN, RNN, deep autoencoder</td>
<td>2013</td>
</tr>
<tr>
<td>MLlib Spark</td>
<td>Apache Software Foundation, UC Berkeley AMPLab, Databricks</td>
<td>Microsoft Windows, OS X, Linux</td>
<td>Yes</td>
<td>Scala, Java, Python, R</td>
<td>Scala, Java, Python, R</td>
<td>No</td>
<td>classification, regression, clustering, collaborative filtering, dimensionality reduction, lower-level optimization primitives</td>
<td>May 2014</td>
</tr>
<tr>
<td>Azure</td>
<td>Dave Cutler from Microsoft</td>
<td>Microsoft Windows, Linux</td>
<td>No</td>
<td>C++</td>
<td>C++, Java, ASP.NET, PHP, Node.js, Python</td>
<td>Yes</td>
<td>classification, regression, clustering</td>
<td>Oct. 2010</td>
</tr>
<tr>
<td>Torch</td>
<td>Ronan Collobert, Koray Kavukcuoglu, Clement Farabet</td>
<td>Linux, Android, Mac OS X, iOS</td>
<td>Yes</td>
<td>C, Lua</td>
<td>Lua, LuaJIT, C, utility library for C++/OpenCL</td>
<td>Yes</td>
<td>deep algorithms</td>
<td>October, 2002</td>
</tr>
<tr>
<td>AWS ML</td>
<td>Amazon Web services</td>
<td>Linux, MacOS, Windows</td>
<td>No</td>
<td>Javascript, CSS, and HTML</td>
<td>.NET, Java, Python, Ruby, PHP and Node.js</td>
<td>No</td>
<td>classification, regression, clustering</td>
<td>April 2015</td>
</tr>
<tr>
<td>MOA</td>
<td>University of Waikato</td>
<td>Cross-platform</td>
<td>Yes</td>
<td>Java</td>
<td>GUI, the command-line, and Java</td>
<td>No</td>
<td>classification, regression, clustering, outlier detection, concept drift detection and recommender systems</td>
<td>Nov. 2014</td>
</tr>
</tbody>
</table>
2.5.1 Apache Storm

*Storm* [165] is an open source distributed real-time data processing framework that provides massively scalable event collection. The initial release was on 17 September 2011, it was created by Nathan Marz and team at BackType, and now owned by Twitter. It is designed for easily processing unbounded streams, and can be used with any programming language. It has been benchmarked at processing over one million tuples per second per node, is highly scalable, fault-tolerant- if a node dies the worker will be restarted on another node-and reliable providing processing job guarantees. Storm is written in Java and Clojure. Keep in mind that Storm is a stream processing engine without batch support. Storm does not support state management natively; however, Trident, a high level abstraction layer for Storm, can be used to accomplish state persistence. Storm is a system of Complex Event Processing (CEP). This type of solution allows companies to respond to the arrival of sudden and continuous data (information collected in real-time by sensors, millions of comments generated on social networks such as Twitter, WhatsApp and Facebook, bank transfers etc.). Some of specific applications of Strom include: real-time customer service management, data monetization, operational dashboards, or cyber security analytics and threat detection.

2.5.2 Apache Kafka

*Kafka* [166] is a fast, scalable, fault-tolerant and durable open-source message broker project developed by the Apache Software Foundation written in Scala. Apache Kafka was originally developed by LinkedIn, and was subsequently open sourced in early 2011. Graduation from the Apache Incubator occurred on 23 October 2012. In November 2014, several engineers who built Kafka at LinkedIn created a new company named Confluent that develops streaming platform based on
Kafka. Apache Kafka supports a wide range of use cases as a general-purpose messaging system for scenarios where high throughput, reliable delivery, and horizontal scalability are important. It can message geospatial data from a fleet of long-haul trucks or sensor data from heating and cooling equipment in office buildings.

### 2.5.3 Oracle

In 2013, Oracle moved toward managing all of its data and big-data technologies with Oracle Enterprise Manager. That includes the Oracle Big Data Appliance, which packages Cloudera’s Hadoop distribution and the Oracle NoSQL database. The company has also brought together multiple low-latency technologies as part of its “Oracle Fast Data” family. The components include Oracle Event Processing, Oracle Coherence, Oracle NoSQL, GoldenGate and Data Integrator (data integration), Oracle Business Analytics, and Oracle Real-Time Decisions. This collection spans low-latency demands from filtering and correlation to data movement and transformation to analysis and real-time decision support. Oracle Event Processing is a complete solution for building applications to filter, correlate and process events in real-time. It enables Fast Data and IoT delivering actionable insight and maximizing value on large volumes of high velocity data from varied data sources in real-time. It enables distributed intelligence and low latency responsiveness by pushing business logic to the network edge [167].

The target applications of Oracle Stream Explorer (OSX) and Oracle R Enterprise (ORE) are: equipment monitoring through sensors, anomaly detection and failure prediction for large systems made of a high number of components. ORE [168] is used for model building, in batch mode, at low frequency, and OSX handles the high frequency streams and pushes data toward a scoring ap-
plication, performs predictions in real-time and returns results to consumer applications connected to the output channels.

OSX [169] is a middleware platform for developing streaming data applications. These applications monitor and process large amounts of streaming data in real-time, from a multitude of sources like sensors, social media, financial feeds, etc. Oracle GoldenGate [170] for Big Data 12c product streams transactional data into big data systems in real-time, without impacting the performance of source systems. It streamlines real-time data delivery into most popular big data solutions, including Apache Hadoop, Apache HBase, Apache Hive, Apache Flume and Apache Kafka to facilitate improved insight and timely action. Oracle Real-Time Decisions [171] is a complete decision management solution with self-adaptive learning that prescribes optimized recommendations and actions with messaging, imagery, products, and services within frontline business processes.

2.5.4 Apache Samza

Samza [172] is an open-source distributed stream processing framework, developed by the Apache Software Foundation, written in Java and Scala. The framework aims to provide a near-realtime, asynchronous computational framework for stream processing. Apache Samza has been developed on top of Apache Kafka for messaging, which were both originally developed by LinkedIn, and YARN for cluster resource management. Samza is simple API- such that it provides very simple callback-based “process message” API, and can handle large amounts of state (many gigabytes per partition), fault tolerant- it works with YARN to transparently migrate your tasks to another machine, durable- it uses Kafka to guarantee that no messages are ever lost. Scalable- it can be partitioned and distributed at every level.
2.5.5 Amazon Kinesis Streams

Amazon Kinesis [173] is a platform for collecting and processing large streams of data on AWS in real-time, AWS launched Kinesis in November of 2013, offering powerful services to make it easy to load and analyze streaming data, and also providing the ability for you to build custom streaming data applications for specialized needs. Web applications, mobile devices, wearables, industrial sensors, and many software applications and services can generate staggering amounts of streaming data—sometimes TBs per hour—that need to be collected, stored, and processed continuously. A typical Amazon Kinesis Streams application reads data from an Amazon Kinesis stream as data records. These applications can use the Amazon Kinesis Client Library, and they can run on Amazon EC2 instances. The new Kinesis Storm Spout routes data from Kinesis to a Storm cluster for processing.

2.5.6 Apache Spark Streaming

Apache Spark [174] is an open-source platform for processing data in real-time, and may be executed and operated using four types of different languages: Scala, the syntax in which the platform is written; Python; R; and Java. Spark Streaming is an extension of core Spark API. Spark Streaming makes it easy to build fault-tolerant processing of real-time data streams. Spark Streaming allows the processing of millions of data among the clusters, and Spark SQL which makes it easier to exploit the data through the SQL language. The way Spark Streaming works is it divides the live stream of data into batches (called microbatches) of a pre-defined interval (N seconds) and then treats each batch of data as Resilient Distributed Datasets (RDDs). Then we can process these RDDs using the operations like map, reduce, reduceByKey, join and window. The results of these
RDD operations are returned in batches.

Spark Streaming can be used for real-time monitoring of application server logs and performing data analytics on those logs. These log messages are considered time series data. Time series data examples include sensor data, weather information, and click stream data. Time series analysis is about processing the time series data to extract insights that can be used for business decision making. This data can also be used for predictive analytics to predict future values based on historical data. Apache promises a calculation speed 100 times quicker than that currently offered by Hadoop MapReduce in memory and 10 times better in disc. Spark can be executed on Hadoop, Apache Mesos, and EC2, in independent cluster mode or in the cloud. In addition, Spark can access numerous databases such as HDFS, Cassandra, HBase or S3, Amazon’s data warehouse.

2.5.7 Apache Flume

*Flume* [175] is a distributed, reliable and open-source log data aggregation framework. The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible into the Hadoop Distributed File System (HDFS).

Flume has a simple and flexible architecture based on streaming data flows; and is robust and fault tolerant with tunable reliability mechanisms for failover and recovery. In one specific example, Flume is used to log manufacturing operations. When one run of product comes off the line, it generates a log file about that run. Even if this occurs hundreds or thousands of times per day, the large volume log file data can stream through Flume into a tool for same-day analysis with
Apache Storm or months or years of production runs can be stored in HDFS and analyzed by a quality assurance engineer using Apache Hive.

2.5.8 Apache SAMOA

SAMOA [176] is a distributed streaming ML framework that contains a programing abstraction for distributed streaming ML algorithms. Its name stands for Scalable Advanced Massive Online Analysis; it was originally developed at Yahoo! Labs in Barcelona in 2013 and has been part of the Apache incubator since late 2014. SAMOA is both a platform and a library. As a platform, it allows the algorithm developer to abstract from the underlying execution engine, and therefore reuse their code to run on different engines. It also allows to easily write plug-in modules to port SAMOA to different execution engines. SAMOA enables development of new ML algorithms without directly dealing with the complexity of underlying distributed stream processing engines. It can be run locally or on one of a few stream processing engines, including Storm, S4, and Samza.

SAMOA includes algorithms for the most common ML tasks such classification, clustering, regression, and frequent pattern mining, along with boosting, and bagging for ensemble creation. Additionally, there is a common platform provided for their implementations, as well as a framework for the user to write their own distributed streaming algorithms. For clustering, they now offer CluStream, and for classification there is the Vertical Hoeffding Tree, which utilizes vertical parallelism on top of the Very Fast Decision Tree, or Hoeffding Tree. This is the standard decision tree algorithm for streaming classification. Regression can be accomplished through the Adaptive Model Rules Regressor, which includes implementations for both vertical and horizontal parallelism [177].
Table 2.3: Comparison between real-time data analytics tools.

<table>
<thead>
<tr>
<th>Tool</th>
<th>First released in</th>
<th>Main Owner</th>
<th>Platform</th>
<th>Written in</th>
<th>API languages</th>
<th>Auto-Scaling?</th>
<th>Event Size</th>
<th>Fault Tolerance</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storm</td>
<td>Sep 2011</td>
<td>Backtype, Twitter</td>
<td>Cross-platform</td>
<td>Clojure and Java</td>
<td>Any programming language</td>
<td>No</td>
<td>Single</td>
<td>Yes</td>
<td>Distributed stream processing</td>
</tr>
<tr>
<td>Kafka</td>
<td>2011</td>
<td>LinkedIn, Confluent</td>
<td>Cross-platform</td>
<td>Scala</td>
<td>Java, C++, Node.js</td>
<td>Yes</td>
<td>Single</td>
<td>Yes</td>
<td>Message broker</td>
</tr>
<tr>
<td>Oracle</td>
<td>2013</td>
<td>Oracle</td>
<td>Cross-platform</td>
<td>Java</td>
<td>Java, Node.js, Python, PHP, and Ruby</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>Distributed stream processing</td>
</tr>
<tr>
<td>Spark</td>
<td>May, 2014</td>
<td>AMPLab, Databricks</td>
<td>Microsoft Windows, OS X, Linux</td>
<td>Scala, Java, Python, R</td>
<td>Scala, Java, Python, R</td>
<td>Yes</td>
<td>Mini-batch</td>
<td>Yes</td>
<td>Streaming analytics.</td>
</tr>
<tr>
<td>Amazon Kinesis</td>
<td>Dec 2013</td>
<td>AWS</td>
<td>Microsoft Windows, OS X, Linux</td>
<td>C++</td>
<td>C++, Java, Python, Ruby, Node.js, .NET</td>
<td>Yes</td>
<td>Data blob of 1 MB size</td>
<td>Yes</td>
<td>Real-time streaming data</td>
</tr>
<tr>
<td>Samza</td>
<td>Jul 2013</td>
<td>LinkedIn</td>
<td>Cross-platform</td>
<td>Scala, Java</td>
<td>Java</td>
<td>No</td>
<td>Single</td>
<td>Yes</td>
<td>Distributed stream processing</td>
</tr>
<tr>
<td>Flume</td>
<td>Jan 2012</td>
<td>Apple, Cloudera</td>
<td>Cross-platform</td>
<td>Java</td>
<td>Java</td>
<td>No</td>
<td>Single</td>
<td>Just with file channel only</td>
<td>Distributed stream processing</td>
</tr>
<tr>
<td>SAMOA</td>
<td>2013</td>
<td>Created at Yahoo Labs</td>
<td>Cross-platform</td>
<td>Java</td>
<td>Java</td>
<td>Yes</td>
<td>NA</td>
<td>Yes</td>
<td>Distributed stream processing</td>
</tr>
</tbody>
</table>

A summarized comparison between various real-time data analytics tools is provided in Table 2.3.

2.6 Applications of ML-Based Context-Aware Systems for SBs

The potential uses of ML in a SB environment can be divided into four categories: detection, recognition, prediction, and optimization [5]. We discuss these categories separately next.

In general, detection is the extraction of particular information from a larger stream of information. Many detection applications in SBs such as fire detection, leak detection, and anomaly detection [178]. Many different applications have been studied by researchers in activity recognition in SBs; examples include fitness tracking, health monitoring, fall detection. [179].

53
The goal from *recognition* is to classify an object or an event to a predefined category. Recognition was the first concept to be introduced in image processing, and eventually evolved to machine learning. It focuses on how to make computer programs perform intelligent and human-like tasks, such as an object from an image.

The goal of *prediction* is to find a model of temporal relations between certain events to predict what is going to happen in the near future. Prediction problems can be classification or regression problems [180]. An example of classification problems is event prediction, where the goal is to predict the most probable event or subsequent activity, while latency prediction is a regression problem in which the output (the latency value) takes on continuous values. The general steps of applying ML processes to predict an event in SBs environment is shown in Figure 2.3.

The goal of *optimization*, on the other hand, is to maximize the long-term rewards by making suitable decisions in different situations. It is possible to use reinforcement learning for these problems. Some optimization problems can be handled as prediction problems so that the rewards for different actions are predicted and the action with the highest reward would be chosen. Decision making is a good example of an optimization problem. It requires taking into account many different variables and solving trade-offs between the benefits of different areas of the environment [181].

Building environments are becoming increasingly equipped with numerous, diverse sensors that measure a variety of different parameters, and data from these sensors is analyzed by ML algorithms and used for a range of services and applications for the activities of the building occupants. SBs go far beyond saving energy and contributing to sustainability goals. The application and services provided by the SBs can be both residential and commercial ranging from e-health,
e-marketing, intelligent car parking system, intelligent transportation system, automation, and logistic services.

Figure 2.4 shows the taxonomy of basic domains of SB services. In the rest of the section, we will briefly describe the major domains in providing the following SB services: 1) environmental services (lighting, HVAC); (2) care of the elderly population; (3) enhancing energy efficiency; (4) enhancing comfort or providing entertainment; (5) enhancing safety and security; and (6) miscellaneous projects.
**Environmental Services: Lighting, HVAC**  Lighting service is related to the comfort of people depending upon their activities and has sensors to save energy when lights are not needed. The power and electricity system may have onsite renewable energy sources to supply a percentage of the power used in the building services.

HVAC is related to the humidity, ventilation and air conditioning system, designed for the comfort of users and with an effective interaction with the environment. The water management service looks to promote savings, to treat and reuse wastewater as possible for flushing, landscaping and air-cooling systems. The waste management is related to the strategy to collect and separate efficiently. Parking service should promote car sharing, electric vehicles and a place for bicycles.
The security is important in all the building, managing automated locks, biometric devices as well as video surveillance systems.

The operations control center is the place where all the analytics of systems are executed and the decisions to support the operations are taken. Visual interfaces integrate a dashboard of the building services showing the status and must support human operators to better manage the building resources. The user interfaces connects the people with the building to set up their optimal parameters for comfort and productivity improvement on daily activities. Finally, the communications center is the core of the network connecting sensors and actuators in the building as well as the operations control center. Based on our literature survey, we have identified that the application areas of SBs can be elderly care, comfort/entertainment, security/safety, energy management and other projects follows:

**Elderly Population’s Home Care**  SB technology such as sensors, voice activation, GPS, Bluetooth, cellular connectivity via mobile phones, smartphone monitoring apps and sophisticated computers can be especially useful for elderly or disabled persons who wish to live independently. Elderly persons can take the advantages of such technologies such as monitoring system, emergency system, dangerous kitchen appliance detection, fall detection and etc. to maintain healthy and safety living while living independently [182] [183].

Such smart technology in the SBs aims to collect real-time information on monitored people’s daily activity levels and this way learn recognition of their personal patterns. ML techniques have the potential for a very wide array of new innovations in healthcare that will be transformative for both providers and their patients. Whenever those monitored patterns deviate from the norm
patterns, SB systems alert both caregivers and family members letting them take immediate action. By using big data analytics and ML algorithms it is possible to analyze large scale data contained in electronic medical records, to learn automatically, for example, how physicians treat patients including the drugs they prescribe [184].

Some prominent projects in this space are described next. Chernbumroong et al. [182] proposed an activity recognition and classification approach for detection of activities of daily livings of an elderly person using SVM. They used wrist worn multi-sensors namely accelerometer, temperature sensor and altimeter for detection basic five activities namely feeding, grooming, dressing, mobility, and stairs. And other instrumental activities such as washing dishes, ironing, sweeping and watching TV. Taleb et al. [185] proposed a middleware-level solution that integrates both the sensing and the monitoring services for assisting elders at smart homes environment. The appliances used in the proposed framework include RFID readers with coverage of the whole house, video cameras, sound sensors, smart door lock, microphone and speakers for interaction with the system. CAALYX [186] is a European Commission-funded project that aims at increasing older people’s autonomy and self-confidence. The service is composed of three distinct subsystems including elderly monitoring subsystem, home monitoring subsystem and the caretaker’s monitoring subsystem. The system relays a high priority message to an emergency service including the geographic position and clinical condition of the elder user. EasyLine+ [187] project funded by the European Commission to support elderly people with or without disabilities in carrying out a longer independent life at home. The system uses neural network, assistive software, and a variety of sensors such as illumination sensor, temperature sensor, door sensors, and RFID giving the capacity of controlling the white goods. Hossain et al. [188] proposed cyberphysical
cloud-based multi-sensory smart home framework for elderly people that supports gesture-based appliance control. Suryadevara et al. [189] proposed a model for generating sensor activity pattern and predicting the behavior of an elderly person using house-hold appliances.

**Energy Efficiency** When temperatures rise or fall in various zones of your home, heaters, air conditioners, fans, and other devices will turn on or off (or increase or decrease in speed or temperature). In order to be able to reduce the amount of energy consumed by improving the efficiency of the supply systems, a crucial step is to analyze how energy is currently consumed in buildings [190]. Analysis of the energy efficiency of the built environment has received growing attention in the last decade. Various approaches have addressed energy efficiency of buildings using predictive modeling of energy consumption based on usage profile, climate data and building characteristics [191] [192]. For instance, lights throughout your home might turn on and off depending on the time of day.

In the past, several attempts have been made to improve energy efficiency in the SBs through the use of smart metering and sensor networks at residential level facilitates, it is a fact that these types of infrastructure are becoming more widespread but due to their variety and size, they cannot be directly used to make conclusions that help to improve the energy efficiency. ML approaches will be the key to handling of energy efficiency problem in SBs. Learning about the users’ consumption habits is able to generate collaborative recommendations and consumption predictions that help the user to consume better, which will in turn improve the demand curve. Moreover, from consumption values, the system learns to identify devices, enabling the demand to be anticipated [193].
Some prominent projects in this space are described next. Reinisch et al. [194] proposed a comprehensive system that supports the optimized application of artificial intelligence methods to the building environment, focusing on relevant features like ubiquity, context awareness, conflict resolution, and self-learning capabilities. The system operates on a knowledge base that stores all information needed to fulfill the goals of energy efficiency and user comfort. Jahn et. al [195] proposed an energy efficiency features system built on top of a Hydra middleware framework [196]. The system provides both, stationary and mobile user interfaces for monitoring and controlling the smart environment. Pan et al. [197] proposed an IoT framework that uses smartphone platform and cloud-computing technologies to improve the energy efficiency in SBs. They built an experimental testbed for energy consumption data analysis. Fensel et al. [198] proposed the SESAME-S project (SEmantic SmArt Metering - Services for energy efficient houses). The project focuses on designing and evaluating end consumer energy efficient services to assist the end-consumers in making well-informed decisions and controlling their energy consumption. The system integrates a variety of smart home components, such as real smart meters, different types of sensors and actuators, as well as a simulator that can flexibly integrate virtual appliances such as the washing machine. Vastardis et al. [199] proposed a user-centric smart-home gateway system architecture to support home-automation, energy usage management, and smart-grid operations. The gateway is supported by ML classification algorithms component such as C4.5 and RIPPER that is able to extract behavioral patterns of the users and feed them back to the gateway.

Irrigation systems monitoring and smart watering system that keep track of rain and soil conditions and irrigate appropriately are a very cost-effective way to reduce outdoor water consumption. Investment in the water management segments-software and services, water efficient plumbing,
and irrigation management-delivers economic and sustainability benefits. One such benefit is water conservation and management, which is a growing issue in commercial buildings. Advocates have long struggled to drive widespread investment in efficiency technologies and the adoption of intelligent building solutions to address water use [200].

**Comfort/Entertainment** One of the main objectives of SB research is to ease daily life by increasing user comfort. SBs offer a better quality of life by introducing automated appliance control and assistive services. They optimize user comfort by using context awareness and predefined constraints based on the conditions of the building environment. A user can control building appliances and devices remotely, which enables him or her to execute tasks before arriving building. Typical examples are ambience control (for example lighting and background music), advanced user interfaces (for example based on voice or gestures), increasing the level of automation of routine activities, etc. [201]. Other services related to comfort services in SB environments are: Indoor Climate Control and Intelligent Thermostat [200]. Indoor Climate Control: Measurement and control of temperature, lighting, CO2 fresh air. In the SB environment HVAC systems play the central role in shaping workplace indoor environmental quality. In large buildings, HVAC systems are designed and operated not only to heat and cool the air, but also to draw in and circulate outdoor air [202].

**Safety/Security** As the SB technology progresses, the role of ML and deep learning in security and connected devices will increase. Deep learning will continue to help gain insights using big data that were previously inaccessible, particularly in image and video. Advanced technologies such as behavioral analysis and ML to detect, categorize, and block new threats will be beneficial.
In a traditional home Fire/smoke detector are activated as soon as a fire is detected, but SB can do much better than the regular fire alarm system. It not only activates the alarm but also turns on light only in the safest route and guides the residents of house out, it will unlock the doors and windows for smoke ventilation, turn off all the appliances and dial to the nearest fire service station. Other than this, it can take video of the areas surrounding the house, provide status of window breakage alarms, and automatically lock all the doors and the windows when the last person of the house leaves [201].

The main services for security and safety in SBs are: Perimeter Access Control, Liquid Presence, Intelligent Fire Alarm, Intrusion Detection and Motion Detection Systems [200]. Perimeter Access Control service provides control to restricted areas and detection of people in non-authorized areas. Here a range of access card solutions that enable staff members, vendors or contractors to enter specific areas at the times you designate. The same access card can also be used to monitor time and attendance. In addition, there is a forefront of biometric technology that comprises fingerprint and facial recognition and iris scans [203]. Liquid Presence detection in data centers, warehouses and sensitive building grounds to prevent break downs and corrosion [204].

Intelligent Fire Alarm and its corresponding safety systems are crucial parts of an intelligent building. It is a system with multi-function sensors (i.e., chemical gas sensors, integrated sensor systems and computer vision systems) measuring smoke and carbon monoxide, giving both early warnings, howling alarms and speaks with a human voice telling where the smoke is or when carbon monoxide levels are rising, in addition to giving a message on the smartphone or tablet if the smoke or CO alarm goes off [205]. Intrusion Detection Systems such as detection of window and door openings and violations to prevent intruders [200]. Motion can be detected using Infrared mo-
tion sensors, which reliably sends alerts to alarm panel (or dialer) and with a system implementing reduced false alarms algorithms and adaption to environmental disturbances [200].

Image recognition solution can be used in security software to identify people, places, objects, and more. It can also be used to detect unusual patterns and activities. Clarifai [206] specializes in a field of ML known as “computer vision” that teaches computers to “see” images and video. Clarifai’s technology can play a key role in security surveillance and at present, the company works only with home security. Each image is processed on a pixel by pixel basis through convoluted neural networks. Bangali and Shaligram [207] proposed a home security system that monitors the home when the user is away from the place. The system is composed of two methods: one uses web camera to detect the intruder—whenever there is a motion in front of the camera, a security alert in terms of sound and an email is delivered to the owner. And the other one is based on GSM technology that sends SMS. A multilevel home security system that sends alert messages to the house owner and police station in case of illegal invasion at home is proposed in [208]. The system is basically a multilevel security system which consists of different sensor nodes as the input elements while the output elements react to the signal received from the input elements. The sensor nodes consist of a thief alarm, presence detecting circuit and the break-in camera. Zhao and Ye [209] proposed a wireless home security system that utilizes low cost, low power consumption, and GSM/GPRS. The system has a user interface and it can respond to alarm incidents.

Other projects CASAS [126] is a project by Washington State University that provides a non-invasive assistive environment for dementia patients at SBs. The project focuses on three main areas for SBs: medical monitoring, green living, and general comfort. CASAS project comprises
of three layers: physical layer, middleware layer, and software applications layer. **Aware Home Research Initiative (AHRI)** [210] is a project that has constructed by a group at the Georgia Institute of Technology for SB services in the fields of health and well-being, digital media and entertainment, and sustainability. AHRI employs a variety of sensors such as smart floor sensors, as well as assistive robots for monitoring and helping elderly.

**House_n** [211] is a multi-disciplinary project lead by group of researchers at the MIT. The project is focused on how the design of a home and its related technologies, products, and services should evolve to better meet the opportunities and challenges of the future. Hundreds of sensing components are installed in nearly every part of the home, and are being used to develop user interface applications that help people easily control their environment, save resources, remain mentally and physically active, and stay healthy.

The **EasyLiving** project [212] at Microsoft Research is concerned with the development of a prototype architecture and technologies to aggregate diverse devices into a coherent user experience for intelligent environments. The EasyLiving Project was designed to provide context-aware computing services through video tracking and recognition, and sensor readings in the sensing room by using the geometric model of a room and taking readings from sensors embedded within.

The **Gator Tech Smart House** project [213] is a programmable space specifically designed for the elderly and disabled developed by The University of Florida’s Mobile and Pervasive Computing Laboratory. The project’s goal is to create assistive environments such as homes that can sense themselves and their residents and provide special services to the residents to compensate for cognitive, mobility, health, and other age-related impairments. In these spaces, service discovery and gateway protocols automatically integrate system components by using generic middleware that
maintains a service definition for every sensor and actuator in the space. The middleware contains separate physical, sensor platform, service, knowledge, context management, and application layers [214].

Other notable smart home projects include DOMUS [215] at University of Sherbrooke in Canada, which is a research project focused on providing cognitive assistance in smart homes and mobile computing to people suffering from Alzheimer’s type dementia, schizophrenia, cranial trauma, or intellectual deficiencies.

Adaptive House project [84] at The University of Colorado has constructed a prototype system that is equipped with an array of sensors that provide information on environmental conditions such as temperature, ambient light levels, sound, motion, door and window openings, and actuators to control the furnace, space heaters, water heater, lighting units, and ceiling fans.

In Asia, also some smart home projects have been developed, such as the early “Welfare Techno House” project, which measured indicators such as ECG, body weight, and urinary volume using sensors placed in the bathroom and bathtub [216]. Ubiquitous Home project [217] is another smart home project in Japan, which uses passive infrared (PIR) sensors, cameras, microphones, pressure sensors, and radiofrequency identification (RFID) technology for monitoring the older adults.

Summary Several methods are used in context-aware system to provide services in SBs, ML based approaches are capable to make better prediction and adaptation than others. The philosophy behind ML is to automate the creation of analytical models in order to enable algorithms to learn continuously with the help of available data. ML can be applied in cases where the desired outcome is known (Supervised learning), or the data is not known beforehand (Unsupervised
learning), or when there is unlabeled data, in conjunction with a small amount of labeled data (semi-supervised learning). or the learning is the result of interaction between a model and the environment (reinforcement learning). The general uses of ML for SB services are detection, recognition, prediction, and Optimization. In the section we also talk about how to acquire the context from multiple distributed and heterogeneous sources and the techniques for modeling and processing such context to be used in the application services of SBs. We also talk about the most used tools and platforms ML and others for Real-Time Data Analytics by ML community to efficiently process and learn from big data. Without such ML tools you would have to implement all of the techniques from scratch requiring expertise in the techniques and in efficient engineering practices.

2.7 Open Issues and Future Research Directions

Research on SBs has made great strides in recent years, but a number of challenges remain. We present some major challenges related to SBs in this part of the work. These challenges will channelize the research directions for future SBs.

2.7.1 Security and Privacy

Where there is interconnection of two systems or networks (wired or wireless), there are issues of security and privacy and the same is true in the case of SB. Security is a key role in building pervasive environments. The applications running on the SB system should guarantee the confidentiality and integrity of data. Access to SB systems must include access control to ensure, for example, that a thief will not be able to disconnect the alarm system by connecting the pervasive
system [218]. With all of this SBs data being transmitted, the risk of losing privacy increases. For instance, how well encrypted will the data be kept and transmitted with. Do you want your neighbors or employers to know what medications that you are taking or your financial situation? [219].

There are specific challenges related to the user privacy include the data privacy of personal information and the privacy of the individual’s physical location and tracking. That needs for privacy enhancement technologies and relevant protection laws and tools for identity management of users and objects [220]. The emerging trend of ML research in security-related use cases, such as determining safe device behavior and general usage patterns, which can subsequently help to detect and block abnormal activity and potentially harmful behavior, can be part of the solution to the issue that contributes to more security and privacy in the SB environments [221].

SBs have the potential to reduce security gap by identifying and detecting risks. Because ML learns the habits of its user, it can detect patterns or behaviors that are out of the ordinary, predicting risks and intrusions before they happen. The adaptive learning algorithm benefits the consumer by anticipating the needs of the connected device owner and making suggestions for rules based on the user’s habits. For example, ML allows a device to learn the routine of its user, such as the time they get home or go to sleep, and then suggests rules based on those behaviors for all connected devices to better work together [222].

2.7.2 SBs and Context-Aware Computing

In the SBs environment, there exists a vast amount of raw data being continuously collected about the human activities and behaviors. It is important to develop techniques that convert this raw data
into usable knowledge [223]. Context awareness and ML techniques are expected to provide a
great support to process and store the big data, and create important knowledge from all this data [224].

Main challenges for data interpretation and the formation of knowledge include addressing
noisy, physical world data, and developing new inference techniques that do not suffer the limita-
tions of traditional algorithms. Usually the contextual information related to the human behaviors
and activities is very complex and it is not formalized in a standard way, modeling human be-
haviors is extremely challenging due to the complex physiological, psychological and behavioral
aspects of human beings [225].

The humans communicate through rich languages as well as gestures and expressions. Modern
ubiquitous computer systems lack an automatic mechanism of inferring information like humans
do. New research is necessary to raise human activities and behaviors recognition to a central
principle in system and app design and to understand the complex dependencies between the apps
and humans [226], [227]. The foundation of context-aware prompting systems have a significant
advantage of SB applications such as ambient assisted living to support heart rate monitoring, med-
ication prompting, generation of agenda reminders, weather alerts, and emergency notifications.
But issuing prompts for all detected errors can possibly be false positives, and consequently, lead
to annoyance and sometimes prove to be unsafe for specific activities. ML methods can be used for
an accurate and precise prediction when an individual faces difficulty while performing everyday
activities [228].
2.7.3 Personal Data Stream Management in SBs

The data streaming management system is able to process raw sensor data to information or to receive information and fuse this to a feature or can directly process features. These features are according to different processing level [229]. While the data processing for a single SB is simple, the data processing for multiple SBs is more complex because there are different people that tend to have more oppositional interests and share less common interests concerning the processed data [230]. The simple sensors in an SB environment detect phenomena like temperature, motion, light, or weather. Furthermore, other appliances like a television or a telephone can also be seen as sensors, when they send their status or other data as events. All this data from different sensors can be used by SB services to detect certain states and calls some actuators by some predefined rules, for instance, turn on the light if the television is used [231].

Although this approach works well for one certain person, it is not generalizable for everyone. Such that the individual preferences can be automatically learned for each person by observing and detecting frequently emerging patterns that are derived from sensor events. Thus, such an approach provides a very adaptively processing for each resident. That is why each of the residents has to define their own set of rules [232]. Because of that, the increasing number of sensors that produce potential infinite sequences of data- so called data streams, custom solutions for analyzing and processing of these data streams are mostly not practical anymore [233].

Despite the availability of new tools and systems for handling massive amounts of data at incredible speeds, however, the real promise of advanced data analytics lies beyond the realm of pure technology [230]. In [234] discusses research challenges for data streams, originating from
real-world applications. They analyze issues concerning privacy, timing and availability of information, relational and event streams mining, preprocessing, model complexity, evaluation, and legacy systems.

2.7.4 Big Data Challenges in SBs

Nowadays, a wide range of sensing technology in the SBs can be used to gather different types of data and generally at a reasonable cost. Indeed, a single SB can generate hundreds of thousands transactions every day and it is not always simple to store them adequately over the long term [193]. While it is not currently the case, we can imagine that in the future companies or government will have to manage incoming data from dozen of SBs creating tremendous challenges and opportunities. This new data could lead us to obtain more information on the context and then consequently provide much better services to the resident [235].

In the world of big data, despite the availability of data, but obtaining useful information from is not necessarily easy for the traditional approaches like trial and error to extract meaningful information from this dataset. Such that analyzing these massive datasets required new technologies to store, organize and process big data effectively, it needs more powerful, high-performance processors that provided the tools to uncover the insights in big data. It also requires flexible cloud computing and virtualization, software such as Apache Hadoop and Spark [236]. It requires providing appropriate ML techniques which differs from the traditional approaches for effective and efficient solution of the above issues. For this reason, researchers need to start thinking about the problems and opportunities arising from the use of big data of SB environment [237] [238]. Big Data could greatly contribute to help identifying and confirming what are the good prompt methods
to use. This information has significant value for the future of SBs as assistive tools and for better services delivery. That is why researchers need to analyze and think about the future in which SBs will be numerous and systems will require to interact with them all [239].

2.7.5 Interoperability

Interoperability means that two (or more) systems work together unchanged even though they were not necessarily designed to work together. When equipment, devices or appliances having different communication and networking technologies can communicate effectively, interoperability is satisfied. The challenge of ensuring that an environment will be intelligible when it comprises a number of components, each of which may have been acquired at different times, from different vendors, and which were created under different design constraints and considerations [240]. Therefore it becomes essential to satisfy interoperability so that a number of heterogeneous communication and networking technologies could coexist in various parts of SBs. For example an energy management system may use Wi-Fi and ZigBee for communication purposes. A lot of work can be done in this context [241].

2.7.6 Reliability

We can expect that a paramount concern of occupants (if not developers) of SB technologies is reliability. The range of domestic technologies present in the building today televisions, telephones, washing machines, microwave ovens are, by and large, exceedingly reliable, even though these are devices of great complexity. A modern digital television set-top box, for example, contains a number of specialized microprocessors devoted to high-bandwidth decompression, cryptography,
rendering, and network communications back to the service provider. And yet, these devices virtually never crash, unlike our desktop computer systems. Achieving expected levels of reliability, especially when coupled with the ad hoc accretion of devices that may be expected in SBs, is a great challenge. Dealing with that challenge depends on understanding the reasons that these devices are so much more reliable than “traditional” desktop software systems. Some of these reasons include: differences in development culture; differences in technological approaches; differences in expectations of the market; differences in regulations [240].

2.7.7 Integration

The key to a successful SB implementation is integration: linking building systems such as lighting, power meters, water meters, pumps, heating and chiller plants together using sensors and control systems, and then connecting the building automation system to enterprise systems. Integration is enabling facility executives to reap smart-building benefits, both in new construction and also by gradually transforming existing buildings into SBs. What these SBs have in common is integration. Seamless integration based on building automation systems brings a range of benefits to both the facility executive and the larger organization. These benefits range from energy savings to productivity gains to sustainability. And once building systems have been integrated, the building automation system can be tied to enterprise business systems to add another level of intelligence to enhance decision-making and improve building performance [242].

However, integrating multiple systems is very challenging as each individual system has its own assumptions, strategies to control the physical world, and semantics. For example, assume IoT apps responsible for energy management (controlling thermostats, windows, and doors) and
home health care (controlling lights, body nodes measuring heart rate and temperature, and sleep apnea machines) are running concurrently. The integrated system should not turn off medical appliances to save energy while they are being used as suggested by the home health care system [226].

As a future perspective for SBs, that you will awake in the morning to the sound of your alarm, and the hidden sensors in the room will know you are getting up. The lights will automatically, but gradually turn on and the thermostat will warm the rooms you are about to use—the bathroom, the kitchen and, a few minutes later, your car. The coffee will start to brew. You will get push notifications about the weather. Your kitchen will remind you which ingredients you will need to pick up on your way home from work and the items you will want to include in dinner that night before they spoil. When you leave the house, you will press a button via an app that will self-drive your car out of the garage. Right then, security measures will then snap into place. House doors will automatically lock. Appliances will switch to an energy-saving mode. When the house senses you are on your way back (using geofencing technology as your smartphone crosses a certain mile radius), it will get ready for your arrival—the thermostat will warm things up, the garage door will open as you pull up, and your favorite music will start to play when you walk in [89].

**Summary:** Although a lot of research has been done in the SBs field, there is a need for a lot more efforts for it to mature there are many researches to solve the challenges available; however, we believe that SBs are possible for the mass market in the near future. The main challenges of this field for the future can be summarized as follows:

- Although security has attracted a lot of attention, but still it is remained as an issue, and with all of this SBs data being transmitted, the risk of losing privacy increases.
• User context in term of behavior and intention should be studied and respected whenever possible.

• Further research is needed into context-aware prompting systems, personal data streaming and big data analysis of occupants in SB environment.

• Some of the other challenges like the interoperability, reliability, and integration still require more attention.
CHAPTER 3
PARAMETERS OPTIMIZATION OF DEEP LEARNING MODELS USING PARTICLE SWARM OPTIMIZATION

Deep learning has been successfully applied in several fields such as machine translation, manufacturing, and pattern recognition. However, successful application of deep learning depends upon appropriately setting its parameters to achieve high-quality results. The number of hidden layers and the number of neurons in each layer of a deep machine learning network are two key parameters, which have main influence on the performance of the algorithm. Manual parameter setting and grid search approaches somewhat ease the users tasks in setting these important parameters. Nonetheless, these two techniques can be very time-consuming. In this paper, we show that the Particle swarm optimization (PSO) technique holds great potential to optimize parameter settings and thus saves valuable computational resources during the tuning process of deep learning models. Specifically, we use a dataset collected from a Wi-Fi campus network to train deep learning models to predict the number of occupants and their locations. Our preliminary experiments indicate that PSO provides an efficient approach for tuning the optimal number of hidden layers and the number of neurons in each layer of the deep learning algorithm when compared to the grid search method. Our experiments illustrate that the exploration process of the landscape of configurations to find the optimal parameters is decreased by 77% - 85%. In fact, the PSO yields even better accuracy results.
3.1 Introduction

Deep learning is an aspect of artificial neural networks that aims to imitate complex learning methods that human beings use to gain certain types of knowledge. We can think of deep learning as a technique that employs neural networks that utilize multiple hidden layers of abstraction between the input and output layers. This is in contrast to traditional shallow neural networks that employ one hidden layer [11].

Deep learning models are utilized in a wide variety of applications including the popular iOS Siri and Google voice systems. Recently, deep neural networks have been utilized to win numerous contests in pattern recognition and machine learning. Some leading examples include Microsoft research on a deep learning system that demonstrated the ability to classify 22,000 categories of pictures at 29.8 percent of accuracy. They also demonstrated real-time speech translation between Mandarin Chinese and English [95]. Deep learning is made available by open source projects as well, currently various commonly used deep learning platforms include: H2O platform, Deeplearning4j (DL4j), Theano, Torch, TensorFlow, and Caffe.

One of the challenges in a successful implementation of deep machine learning is setting the values for its many parameters, particularly the topology of its network. Let \( L \) be the number of hidden layers, \( N_i \) be the number of neurons in layer \( i \) and \( N = \{ N_1, N_2, \ldots, N_L \} \). Parameters \( L \) and \( N \) are very important and have a major influence on the performance of deep machine learning. Manually tuning these parameters (essentially through trial and error method) and finding high-quality settings is a time-consuming process [243]. Besides, the solutions obtained by the manual process are usually not equally distributed in the objective space.
To address this challenge, grid search is a common approach for setting parameter values of the deep learning models. Grid search is more efficient and saves time in setting $L$ and $N$; with this approach, a list of discrete values of $L$ and $N$ are prepared in advance, where each entry shows the number of hidden layers and its corresponding number of neurons. The deep learning algorithm trains multiple different models using all the lists entries. Finally, the selection of the parameters is measured using the models accuracy. However, grid search is still a computationally demanding process as the number of possible combinations is exponential, especially when the number of parameters increases and the interval between discrete values is reduced. In addition if the list of parameters are poorly chosen, the network may learn slowly, or perhaps not at all [244].

This paper proposes another parameter selection method for deep learning models using PSO. PSO is a popular population-based heuristic algorithm that simulates the social behavior of individuals such as birds flocking, a school of fish swimming or a colony of ants moving to a potential position to achieve particular objectives in a multidimensional space [245]. PSO is found to have the extensive capability of global optimization for its simple concept, easy implementation, scalability, robustness, and fast convergence. It employs only simple mathematical operators and is computationally inexpensive in terms of both memory requirements and speed [246].

Several researchers have explored parameter optimization of various machine learning algorithms. PSO has been applied to train shallow neural networks [247]. There are a number of studies about specifying and optimizing the initial weights of Artificial Neural Networks (ANN) learning [248][249][250][251]. Finding the best number of hidden neurons, learning rate, momentum coefficient and initial weights have been studied in the literature. Bovis et al. [252] worked on mammographic mass to find an optimum number of hidden neurons for classification. Mirjalili et
al. [253] proposed a hybrid of PSO and gravitational search algorithm to train feed forward neural networks. PSO has been used to optimize the parameters of SVM. Bamakan et al. [254] proposed a hybrid approach for parameter determination of the non-parallel SVM using PSO. They considered the number of support vectors along with the classification accuracy as a weighted objective function.

In this study, we use PSO to optimize the number of hidden layers \( L \) and the number of neurons \( N_i \)'s in each layer for deep learning models [243]. To the best of our knowledge, no one has used PSO for setting these parameters. Currently, the \( H_2O \) platform utilizes grid search for parameter selection. In our experiments, we observed that PSO results in a significant decrease in the number of configurations that need to be evaluated to find optimal parameters for deep learning models. Specifically the decrease was by 77% - 85% while achieving higher model accuracy compared to grid search. While the results presented in this paper are based on a dataset collected from a campus Wi-Fi network, we believe that PSO would result in similar results in other application domains.

The remainder of the paper is organized as follows: Section II presents the motivations behind this work. In Section III, we present our proposed deep learning parameter selection method using PSO. Section IV presents our experimental results and the lessons learned, and finally, Section V concludes this study and discusses future research directions.

3.2 Motivations

To the best of our knowledge, there is no theory yet to determine the best number of hidden layers and the number of neurons in each layer that should be used by a deep learning model to
approximate a given function. There are several alternatives, rules of thumb that could mitigate the modelers effort and time. For instance, number of hidden layers could be selected to be between the number of inputs and outputs [255]. Another rule suggests that the number of hidden layers can be based on the following formula [256]:

\[ H \approx (I + O) \cdot \frac{2}{3} \tag{3.1} \]

Where \( H \) is the number of neurons in the hidden layers, \( I \) is the number of features in the input layer, and \( O \) is the number of neurons in the output layer.

In [257], Swingler argues that the number of hidden layers should never exceed the number of input variables. In terms of neurons, the number of hidden layer neurons should be less than twice of the number of neurons in the input layer [258].

Configuring deep learning models using the above rules is almost free of any computations, since what all needed is a basic and simple calculation. However, these rules of thumb are not applicable all the time because they ignore the number of trainings, the amount of noise in the targets, and the complexity of the function. Further experiments using a large number of different datasets are needed in order to find good rules of thumb for the different application domains.

In our experiments, we use deep learning models for predictive modeling. H2O uses a purely supervised training protocol [259]. The configurations of deep learning algorithms in the H2O platform and other popular platforms have no default settings for the hidden layer size and the number of neurons in each layer. Experimenting with building deep learning models using different
network topologies and different datasets will lead to intuition for these parameters. For manual parameter selection, we selected different configurations in our experiments in terms of the number of hidden layers and the number of neurons in each layer of the deep learning model. Figure 3.1 shows the effect of different configurations on the accuracy. The figure illustrates that the parameter selection process has a significant impact on the accuracy of the deep learning model. However, the number of potential configurations is large. In fact searching for the best configuration is like searching for a needle on a haystack.

As illustrated in Figure 3.1, the number of hidden layers and the number of neurons in each layer play a major role to efficiently enhance the accuracy. For example, in comparison to the deep learning model that employs 10 hidden layers and 170 neurons in each layer, the accuracy is improved by 40% for the deep learning model that employs 5 layers and 200 neurons per layer. This accuracy is further improved by 76% when the deep learning model employs 1 hidden layer

![Figure 3.1: The effect of manual configuration setting on the accuracy.](image-url)
and 61 neurons in the layer. By running numerous configurations, one can find the best parameter values. However, that is a computationally intensive endeavor. Thus, it can be easily seen that finding high-quality parameter settings of a deep learning model is a time consuming process that requires an in-depth knowledge of the underlying algorithms, properties of the learning domain and the nature of the dataset that are being used in the training process. In section 4 of this paper, we compare our proposed PSO based parameter selection method with the grid search technique.

3.3 PSO-Based Parameter Optimization Model

The PSO algorithm is an iterative optimization method that was originally proposed in 1995 by Kennedy and Eberhart [245]. PSO was developed to mimic bird and fish swarms. Animals who move as a swarm can reach their aims easily. The basic form of the PSO algorithm is composed of a group of particles which repeatedly communicate with each other; the population is called a swarm. Each particle represents a possible solution to the problem (i.e., the position of one particle represents the values of the attributes of a solution) [260]. Each particle has its position, velocity and a fitness value that is determined by an optimization function. The velocity determines the next direction and distance to move. The fitness value is an assessment of the quality of the particle. The position of each particle in the swarm is tweaked to move closer to the particle which has the best position. Each particle updates its velocity and position by tracking two extremes in each iteration. One is called the personal best, which is the best solution that the particle was able to obtain individually so far. The other is called the global best which is the best solution that all
particles were able to find collectively so far. PSO is mathematically modeled as follows [245]:

\[ v_{t+1}^i = w.v_t^i + c_1.\text{rand.}(pbest_i - x_t^i) + c_2.\text{rand.}(gbest - x_t^i) \]  

(3.2)

Each step \( t \), the position of particle \( i \), \( x_t^i \) is updated based on the particle’s velocity \( v_t^i \):

\[ x_{t+1}^i = x_t^i + v_{t+1}^i \]  

(3.3)

In Equations 3.2 and 3.3 above, \( v_t^i \) and \( x_t^i \) are the \( t \)th speed and position components of the \( i \)th particle. \( c_1 \) and \( c_2 \) are the acceleration coefficients and represent the weights of approaching the \( pbest_i \) and \( gbest \) of a particle. \( w \) is the inertia coefficient as it helps the particles to move by inertia towards better positions. \( \text{rand} \) is a uniform random value between 0 and 1. The parameters utilized in our experiments are listed in Table 3.1.

Algorithm 1 above provides the details of our proposed PSO based parameter selection techniques for deep learning models. The algorithm is presented for campus occupant prediction scenario using Wi-Fi collected data. This scenario will be fully explored in the next section (i.e., Section 3.4).

In our implementation of PSO, the \( i \)th particles velocity is calculated according to the following:

- Velocity of number of layers

\[ V_{L,i}^{t+1} = w.V_{L,i}^{t} + c_1.\text{rand.}(L_{i}^{best} - V_{L,i}^{t}) + c_2.\text{rand.}(G_{L,\text{best}}^{L} - V_{L,i}^{t}) \]  

(3.4)
Algorithm 1 PSO for Parameter Optimization of Deep Learning Models.

Input: Wi-Fi dataset, location, time and MAC addresses.
Output: Optimal configuration in terms of the number of hidden layers and number of neurons in each layer for the deep learning model.

Begin
1: Initialization
   a. Set the values of acceleration constants ($c_1$ and $c_2$), W, PopSize, MaxIt, and specify the range bounds: MinLayer, MaxLayer, MinNeurons, MaxNeurons, MaxLayerVelocity and MaxNeuronVelocity.
   b. Define the fitness function (i.e., deep learning model accuracy).
   c. Establish initial random population for the number of hidden layers and number of neurons in each layer.
   d. Calculate the fitness value for each particle and set the personal best (pbest) for each particle and the global best (gbest) for the population.
2: Repeat the following steps until the gbest solution does not change anymore or the maximum number of iterations is reached.
   a. Update the number of hidden layers, the number of neurons in each layer, the velocity of the number of hidden layers and the number of neurons in each particle according to the Equations 3.2 through 3.7.
   b. Calculate the fitness value for each particle. If the fitness value of the new location is better than the fitness value of personal best, the new location is updated to be the personal best location.
   c. If the currently best particle in the population is better than the global best, the best particle replaces the recorded global best.
3: Return the optimal number of hidden layers, the number of neurons in each layer for the deep learning model.
End

Where $V_L$ is the velocity of the number of hidden layers, $L_i^{best}$ is the particles best local value of the number of hidden layers, and $G^{Lbest}$ is the best global value of the number of hidden layers.

- Velocity of number of neurons

$$V_{N,i}^{t+1} = w.V_{N,i}^t + c_1.rand.(N_i^{best} - V_{N,i}^t) + c_2.rand.(G^{Nbest} - V_{N,i}^t)$$ (3.5)
Table 3.1: Training time for week ends and work days by SVM, and DNN.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>10, 25, or 50</td>
</tr>
<tr>
<td>Learning coefficients: c1, c2</td>
<td>uniformly distributed between [0, 4]</td>
</tr>
<tr>
<td>Maximum number of iterations</td>
<td>10</td>
</tr>
<tr>
<td>Number of hidden layers</td>
<td>within the range [1, 200]</td>
</tr>
<tr>
<td>Number of neurons in each layer</td>
<td>within the range [1, 10]</td>
</tr>
<tr>
<td>Particle dimensions</td>
<td>represents the number of hidden layers and the number of neurons in each layer</td>
</tr>
<tr>
<td>Hidden layers velocity</td>
<td>MinLayerVelocity(= -0.1(\text{MaxLayers} - \text{MinLayers}))</td>
</tr>
<tr>
<td></td>
<td>MaxNeuronVelocity(= 0.1(\text{MaxNeurons} - \text{MinNeurons}))</td>
</tr>
<tr>
<td>Neuron velocity</td>
<td>MaxNeuronVelocity(= 0.1\ (\text{MaxNeurons} - \text{MinNeurons}))</td>
</tr>
<tr>
<td></td>
<td>MinNeuronVelocity(= -(0.1 \ (\text{MaxNeurons} - \text{MinNeurons}))</td>
</tr>
</tbody>
</table>

Where \(V_N\) is the velocity of the number of neurons in each hidden layer, \(N_i^{best}\) is the particles best local value of the number of neurons in each hidden layer, and \(G^{Nbest}\) is the best global value of the number of neurons in each hidden layer.

- **Position for number of layers**

\[
L_i^{t+1} = L_i^t + V_{L,i}^{t+1} \tag{3.6}
\]

- **Position for number of neurons**

\[
N_i^{t+1} = N_i^t + V_{N,i}^{t+1} \tag{3.7}
\]
3.4 Experimental Results and Lessons Learned

In our experiments, we select a smart building application to assess the performance of our proposed PSO based parameter selection technique. We built a deep learning model based on 6 weeks (January 15, 2016 – Feb 29, 2016) of Wi-Fi access data collected from 14 buildings of the campus of the University of Houston campus. Our goal is to build a deep learning model that predicts the number of occupants at a given location in 15, 30 and 60 minutes from the current time. Awareness of the number of occupants in a building at a given time is crucial for many smart building applications including energy efficiency and emergency response services [261].

Our experiments were conducted using the R language. We executed our experiments on a 24-core machine with 2.40GHz Intel Xeon CPU and 32 GB RAM. In our scenarios, we split a 6 weeks dataset into 7 parts; each part corresponds to a day of the week. Each dataset has the following features: Access Point ID (APID), Date, Time, User MAC address and Building number. The three features that our deep learning model needs to predict are the count of MAC addresses within 15, 30 and 60 minutes from the current time at a given date, time and location (i.e., APID and Building number). In the process, we built a deep learning model for each day of the week. Table 3.2 summarizes the different parts of the dataset. Further, each dataset for a specific day of the week has been split into training and testing sets. Specifically, the first five weeks of the dataset were used as a training set while the data that pertains to the sixth week is used as a testing set. We then set out to address the main goal of this paper which is to compare our proposed PSO based parameter selection technique vis-à-vis the grid search technique in terms of finding the best parameters for the seven models that correspond to the days of the week.
In order to evaluate and compare the grid search and PSO approaches, both the accuracy and the number of configurations that need to be explored to get the best accuracy are evaluated. In case of PSO, the algorithm terminates when the maximum number of iterations is reached or when there is no difference between the accuracies of two consecutive iterations. Since the count of occupants at a given time and location is a continually changing number (i.e., regression problem), it does not make much sense to predict the exact number of occupants $N$ at a given date, time and locations. Rather, it is more practical to allow a small tolerance in the count, for example $N \pm n$. Therefore, we consider clusters with window size $\pm$ (e.g., 20) when we evaluate the accuracy of the predicted occupancy for each dataset.

Three different swarm sizes of 10, 25 and 50 particles are used in our experiments. Figure 3.2 shows the accuracy (c.f. Figure 3.2a) and the number of configurations that need to be evaluated (c.f. Figure 3.2b) to achieve that accuracy for predicting the occupancy within a 60 minute time window. These two figures jointly illustrate the by using a small population size (e.g., 10 particles), the PSO based parameter value selection technique was able to achieve an accuracy that is almost the same as that achieved by using a larger population size (e.g., 25 and 50 particles) for almost all the datasets that we experimented with. Therefore, the PSO based parameter value selection approach does not require a large number of particles to produce competitive results. Another observation drawn from Figure 3.2(b) is that the number of iterations needed to reach the globally best solution is almost one-third and one-fifth the number of configurations that need to be evaluated by the grid search method when the PSO based techniques employs 25 and 50 particles, respectively. This demonstrates that the PSO based technique can be computationally efficient to determine the deep learning parameters. Therefore, in the following experiments can simply consider the PSO
Table 3.2: Summaries of 7 datasets for each days of week.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size of training set</th>
<th>Size of testing set</th>
<th>Actual occupancy in the next 60 minute</th>
<th>Actual occupancy in the next 30 minute</th>
<th>Actual occupancy in the next 15 minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sats</td>
<td>335137</td>
<td>71551</td>
<td>167</td>
<td>110</td>
<td>93</td>
</tr>
<tr>
<td>Suns</td>
<td>213434</td>
<td>108597</td>
<td>184</td>
<td>100</td>
<td>80</td>
</tr>
<tr>
<td>Mons</td>
<td>1686200</td>
<td>795439</td>
<td>715</td>
<td>648</td>
<td>488</td>
</tr>
<tr>
<td>Tues</td>
<td>2129033</td>
<td>411025</td>
<td>732</td>
<td>628</td>
<td>474</td>
</tr>
<tr>
<td>Weds</td>
<td>2141754</td>
<td>404023</td>
<td>792</td>
<td>618</td>
<td>481</td>
</tr>
<tr>
<td>Thurs</td>
<td>1986703</td>
<td>269976</td>
<td>794</td>
<td>689</td>
<td>493</td>
</tr>
<tr>
<td>Fri</td>
<td>1200046</td>
<td>253995</td>
<td>323</td>
<td>262</td>
<td>234</td>
</tr>
</tbody>
</table>

based technique with 10 particles and compare our results with the grid searching technique.

Figures 3.3 - 3.5 show the number of different configurations that need to be evaluated to reach the globally best solution in terms of predicting the occupancy in the next 60, 30 and 15 minutes, respectively. These figures illustrate that better accuracy can be achieved when using our proposed PSO based parameter value selection technique while having to evaluate a significantly lower number of configurations compared to the grid search approach. This clearly exhibits the supremacy of the PSO based technique over grid search. Thus, it can serve as a great candidate for parameter tuning of deep machine learning models. Of course, one needs to carefully analyze dataset biases or domain specific properties that give rise to these results, but that is beyond the scope of this paper and is left for future extensions.

3.5 Conclusions and Future Work

Multiple parameters have to be set and tuned for deep learning models. These parameters can have a significant influence on the results and the computational needs of deep learning models. Optimization methods therefore need to be used to help find optimal parameter settings. Consequently,
Figure 3.2: Comparison between three different swarm sizes (10, 25 and 50).

Figure 3.3: Comparison between PSO and grid search to predict within a 60 minute interval.

Figure 3.4: Comparison between PSO and grid search to predict within a 30 minute interval.
the user can focus on the results of deep learning rather than on spending time and efforts on deciding the optimal parameter values. This paper presents a PSO based parameter value selection technique to optimize the performance of deep learning models, by selecting the number of hidden layers and the number of neurons in each layer. Our results show that the proposed PSO algorithm is useful in the process of training deep learning models. We demonstrated the performance of the proposed technique in a smart building scenario where the number of occupants needs to be predicted in the next 60, 30 and 15 minutes based on collected Wi-Fi data. The results obtained show that training times decreased by 77% - 85% when using the PSO based approach compared to the grid search method. Our proposed PSO based technique also gives a better classification accuracy compared to the grid search approach. As a future extension, we intend to explore the use of PSO to tune other deep learning parameters such as: the activation functions and the number of epochs. Note that it is easy to implement parallel versions of PSO on GPUs. Therefore, resulting in further reduced training times, while letting researchers focus on extracting subject matter knowledge using deep learning models, rather than letting them focus on the parameter value selection process itself.
Knowing how many people occupy a building, and where they are located, is a key component of smart building services. Commercial, industrial and residential buildings often incorporate systems used to determine occupancy. However, relatively simple sensor technology and control algorithms limit the effectiveness of smart building services. In this paper we propose to replace sensor technology with time series models that can predict the number of occupants at a given location and time. We use Wi-Fi datasets readily available in abundance for smart building services and train Auto Regression Integrating Moving Average (ARIMA) models and Long Short-Term Memory (LSTM) time series models. As a use case scenario of smart building services, these models allow forecasting of the number of people at a given time and location in 15, 30 and 60 minutes time intervals at building as well as Access Point (AP) level. For LSTM, we build our models in two ways: a separate model for every time scale, and a combined model for the three time scales. Our experiments show that LSTM combined model reduced the computational resources with respect to the number of neurons by 74.5% for the AP level, and by 67.13% for the building level. Further, the root mean square error (RMSE) was reduced by 88.2% - 93.4% for LSTM in comparison to ARIMA for the building levels models and by 80.9% - 87% for the AP level models.
4.1 Introduction

Being able to accurately count the number of occupants in a smart building has high utility for a number of applications. Information on building occupancy can be used to save energy by controlling temperature and ventilation more accurately. Number of users in the environment is important to accurately recognize the activities of (groups of) agents [262]. In the event of an emergency, first responders often need to know if people are trapped and where they might be located in large buildings. In the context of large facilities like conference centers or in the retail space, knowing how many people are in certain locations and how long they dwell can be used to value shelf-space or storefront locations [263].

Commercial, industrial and residential buildings often incorporate many approaches to determine occupancy including: passive infra-red (PIR) sensors, ultrasonic ranging sensors, microwave sensors, smart cameras, break beam sensors and laser range-finders [264]. However, these sensors extend across a wide range of cost and performance. The ability to determine the actual number of occupants in a place is often beyond the range of current common sensing techniques. Low-cost sensors, like PIR and ultrasonic ranging sensors are typically error-prone and usually only detect binary occupancy values rather than estimating load. Expensive sensors tend to require the complicated site-specific installation and standardization methods [263]. Motion detectors have inherent limitations when occupants remain relatively still [265]. Furthermore, distant passersby and wafts of warm or cold air are interpreted as motion leading to false positives [266]. Video cameras raise privacy concerns and require large amounts of data storage and complex video processing [267]. Other work has focused on the use of carbon dioxide ($CO_2$) sensors in conjunction with build-
ing models for estimating the number of people generating the measured CO₂ level [268]. Smart buildings have a high degree, if not full, Wi-Fi coverage and thus this paper explores the use of Wi-Fi as sensory data to predict occupancy using Wi-Fi beacon time series. In particular, we use the data to predict the number of occupants in building using LSTM deep learning networks.

Recently, time series methods have been successfully applied in a wide range of IoT application that have a time dimension such as energy usage prediction, non-intrusive activity detection, demand side management and control [74]. One of the main purposes of time series data is that past observations of the data can be used to forecast future values. The use of observations from a time series available at time \( t \) to predict its value at time \( t + 1 \) is called forecasting. In this paper we develop and compare two time series models, ARIMA model and LSTM deep recurrent neural network to forecast the number of occupants in a smart building at a specific time under three time scales, namely, 15, 30 and 60 minutes. We focus on the following three questions:

- How to predict the number of occupants using Wi-Fi beacons?
- Which model yields better accuracy, ARIMA or LSTM?
- For LSTM models, which is better – to build a separate model for a specific time scale or to build a combined model for the three time scales with respect to computational performance and accuracy?
- For ARIMA and LSTM models, which way is better – to build one model for the entire building for a specific time scale or to build one model for every AP (Access Point) in the building for a specific time scale?
This is the first paper exploring the use of Wi-Fi beacons to predict the number of occupants in a commercial building, to the best of our knowledge. The advantage of using Wi-Fi is that there is no need to deploy an infrastructure to count the number of occupants in a building which reduces the cost and the complexity of the system. Another contribution of this paper is, answering the above questions which are crucial for improving the state-of-the-art in time series forecasting.

The remainder of this paper is organized as follows: Section II presents the most recent related work. In Section III, we discuss ARIMA and LSTM models. Section IV presents our experimental results and the lessons learned, and finally, Section V concludes the paper.

4.2 Related Work

A significant amount of work has been done in the past two decades to enable accurate and robust occupants counting. In brief, Krumm and Brush [269] presented an occupancy prediction algorithm that gives probabilities of occupancy at different times of day. However, this algorithm computes a representative Sunday, Monday, etc. for each day of the week, without being able to respond to changing occupancy patterns as PreHeat [270] does. Lu et al. [271] formulate a hidden Markov model (HMM) to predict occupancy and control HVAC systems. They collected occupancy data in eight US households for one to two weeks. Using leave-one-out cross-validation to train and test the HMM, they evaluate their approaches MissTime (i.e., total occupied time not at set point) and energy savings for each day in a week using the US Dept. of Energys Energy-Plus simulator. Mozer et al. [270] describe a Neurothermostat which utilizes a hybrid occupancy predictor, making use of an available daily schedule and a neural network which was trained on five consecutive months of occupancy data. Mozer et al. show that the Neurothermostat results in
a lower unified cost, where energy and occupant comfort are expressed. Recently, some studies focused on counting pedestrians with binary sensors and Monte-Carlo methods [272] but those are once again hardly usable in homes as they make use of an important number of landmarks such as doors, stairs and elevators, that may not be present in regular homes. Yang et al. [273] propose image-based counting technique that uses a network of simple image sensors. They introduce a geometric algorithm that computes bounds on the number and possible locations of people using silhouettes computed by each sensor through background subtraction.

In this paper we use ARIMA and LSTM time series algorithms to predict the number of occupants in a smart building using Wi-Fi network data. To the best of our knowledge, this work is the first attempt that addresses the role of time series methods to forecast the number of occupants in the smart building. One of the main advantages of this technique is that it does not require an additional infrastructure to be able to count the number of people in a building.

4.3 Models

Time series data is any data that has a timestamp, such as IoT device data, stocks, and commodity prices. Different time series prediction models exist that work on different patterns. In this paper our focus is on an ARIMA model as this is one of the most widely used statistical models for time series forecasting and thus has a strong potential for occupancy prediction using Wi-Fi time-series data. Deep recurrent neural networks have recently gained a lot of attention in exhibiting good prediction capabilities. Therefore, we also focus on the LSTM model of deep recurrent neural networks.
4.3.1 ARIMA Time Series Model

ARIMA is one of the most common univariate time series models, it is also well-known as Box-Jenkins methodology in the model selection procedure, and the popularity of this model is due to its statistical properties [274]. The AutoRegressive (AR) part of ARIMA indicates that the variable of interest is regressed on its lagged values. The Moving Averages (MA) part indicates that the regression error is actually a linear compound of error terms whose values occurred simultaneously and at various times in the past. The Integrated (I) indicates that the data values have been replaced with the difference between their values and the previous values [275][276].

An AutoRegressive of order p, AR(p), component of an ARIMA model is a discrete time linear equations with noise. It can be written in the form:

\[
X_t = \sum_{i=1}^{p} \alpha_i X_{t-i} + Z_t
\]

(4.1)

Where the terms \( \alpha_i \) are autocorrelation coefficients at lags 1, 2, \ldots, p and \( z_t \) is a residual error term. Note that this error term specifically relates to the current time period \( t \).

A Moving Average with order q, MA(q), model can be used to provide a good fit to some datasets. A simple form of such models, based on prior data, can be written as:

\[
X_t = \sum_{i=0}^{q} \beta_i Z_{t-i}
\]

(4.2)
Where the $\beta_i$ terms are the weights applied to prior values in the time series, and it is usual to define $\beta_i = 1$, without loss in generality.

An AutoRegressive Moving Average with orders p and q, ARMA(p, q), is the one where these two models are combined by simply adding them together as a model of order $(p, q)$, where we have $p$ AR terms and $q$ MA terms. An ARMA discrete time linear equation with noise has the following form:

\[ X_t = \alpha_0 + \sum_{i=1}^{p} \alpha_i X_{t-1} + \sum_{i=0}^{q} \beta_i Z_{t-i} \quad (4.3) \]

where $X_t$ is a stationary stochastic process with non-zero mean, $\alpha_0$ is constant term, and $Z_t$ is a white noise disturbance term. A time series is said to be stationary, if the mean of the series and the covariance among its observations do not change over time and do not follow any trend [277]. The ARIMA model which generally overcomes the limitation of non-stationary time series by introducing a differencing process of subtracting the observation in the current period from the previous one. This process effectively transforms the non-stationary data into a stationary one [278]. Hence, the ARIMA model is called Integrated ARMA because of the stationary model that is fitted to the differenced data that has to be summed or integrated in order to provide a model for the original non stationary data. Eq. 4.3 denotes by notation ARIMA $(p, d, q)$ where $p$ is the order of the autoregressive part, which is the number of dependent variable lagged in the right hand side, $d$ is the order of differencing performed on $X$, before estimating the above model, and $q$ is the order of the moving-average process, which is the lagged error term in the right hand side of
Eq. 4.3. The AR part of the model indicates that the future values of $X_t$ are weighted averages of current and past realizations. Similarly, the MA part of the model shows how current and past random shocks will affect the future values of $X_t$. The more general ARIMA process model can be written as an AR if the MA process is invertible. One of the best ways to make a series stationary on variance is through transforming the original series through log transform.

4.3.2 Deep LSTM Model

The Long Short-Term Memory network, or LSTM network, is a special kind of recurrent neural network (RNN) developed in 1997 by Hochreiter & Schmidhuber [279]. LSTM is trained using Back propagation Through Time (BPTT) and overcomes the vanishing and exploding gradient problem in standard RNN by learning tasks involving long term dependencies. Instead of neurons, LSTM networks have memory blocks that are comprised of memory cell units that are able to remember the value of a state for an arbitrary long time, as well as three different gate units that can learn to keep, utilize, or destroy a state when appropriate. The memory blocks are connected through layers [280]. We can make a deep LSTM by stacking multiple LSTM layers. Although LSTM networks are already deep architectures in the sense that they can be considered as a feed-forward neural network unrolled in time where each layer shares the same model parameters. But the deep LSTM has an additional meaning; it has been argued that deep layers in LSTM allow the network to learn at different time scales over the input. With the deep LSTM, the input to the network at a given time step goes through multiple LSTM layers in addition to propagation through time and LSTM layers [281]. Figure 4.1 shows the architecture for one module in an LSTM network. The network takes three inputs. $X_t$ is the input of the current time step, $h_{t-1}$
is the output from the previous LSTM unit and $C_{t-1}$ is the memory of the previous unit. As for outputs, $h_t$ is the output of the current network. $C_t$ is the memory of the current unit. While the internals of the module are as follows:

\[
f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{4.4}
\]

\[
i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{4.5}
\]

\[
c_t = f_t c_{t-1} + i_t \text{tanh}(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{4.6}
\]

\[
o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{4.7}
\]

\[
h_t = o_t \text{tanh}(c_t) \tag{4.8}
\]

$\sigma$ is the logistic sigmoid function, and $f$, $i$, $c$ and $o$ are respectively the forget gate, input gate, cell state and output gate. $W_{cf}$, $W_{ci}$, and $W_{co}$ are denoted weight matrices for peephole connections. $b_f$, $b_i$, $b_c$, and $b_o$ are respectively the bias vectors for forget, input, cell state and output gates.

In LSTM, three gates ($f$, $i$, $o$) control the information flow. The forget gate determines whether to pass the previous memory $h_{t-1}$. The ratio of the previous memory is calculated in equation 4.4 and used for equation 4.6. The input gate decides the ratio of input. When calculating the cell state, this ratio has an effect on equation 4.6. The output gate determines whether to pass the output of memory cell or not [282]. Equation 4.8 captures this process.

While building an LSTM model, one of the difficulties is tuning the numerous parameters when it comes to train the model because there is no good theory on how to do it. One way to tune the parameters is by applying grid search. Or use a systematic method to explore different
configurations for the network. Qolomany et al. [22] proposed a systematic way to tune the deep neural network parameters using particle swarm optimization. Since our goal in this paper is to assess efficacy of LSTM and compare it to ARIMA, we simply use grid search to tune the number of hidden layers, number of neurons in a layer, batch size, number of epochs, and lag size parameters that play the major role in building a time series model using LSTM algorithm. In the experimental results section we discuss more on the role of each parameter to decrease error while building the model.

The computational complexity in terms of the total number of neurons needed to build an LSTM model for a time scale is

$$T_s = N.H + I + 1$$  \hspace{1cm} (4.9)
Where $T_s$ is the total number of neurons needed to build a separate model with single output, $N$ is the number of neurons in each layer, $H$ is the number of hidden layers, and $I$ is the number of inputs features represented here as the lag size. While the number of neurons needed to build a single combined model for the three time scales can be calculated as

$$T_c = N.H + m.I + m$$

(4.10)

Where $T_c$ is the total number of neurons needed to build a single combined model with multiple output, $N$ is the number of neurons in each layer, $H$ is the number of hidden layers, $m$ is the number of different models, which is three in our case, each model represents a specific time scale (15, 30 and 60 minutes time scales), and $I$ is the number of inputs features represented here as the lag size.

4.3.3 Forecast Evaluation Methods

The criterion that we use to make comparisons between the forecasting ability of the ARIMA time series models and LSTM models is the root mean square error (RMSE), a standard statistical metric to measure model performance. RMSE represents the sample standard deviation of the differences between predicted values and observed values. These individual differences are called residuals when the calculations are performed over the data sample that was used for estimation [283]. The formula for RMSE is
\[ RMSE = \sqrt{\frac{1}{N} \sum (P_t - A_t)^2} \]  

(4.11)

Where \( P_t \) is the predicted value for time \( t \), \( A_t \) is the actual value at time \( t \), and \( N \) is the number of predictions.

4.4 Experimental Setup

4.4.1 Dataset

The data set is collected from the Wi-Fi network at the University of Houston main campus. The campus has full Wi-Fi coverage with Access Points (AP) managed through controllers that manage hand off allowing users to move around without losing the connection. Whenever a user carrying a Wi-Fi enabled device is passing by the network, the device is automatically exchanging beacons with the Wi-Fi network regardless whether the user is actively using the device. The Wi-Fi network captures these beacons and archive them in a storage device for further analysis. The data collected includes, connection time, connection duration, MAC address, access point ID.

In our experiments we used 6 weeks (January 15, 2016 - Feb 29, 2016) of Wi-Fi dataset for a building that has 18 AP from the University of Houston campus. We preprocessed the raw dataset as time series format for a use case scenario of services for smart buildings environment. We predict the number of occupants at a given time and location. All our experiments are conducted using ARIMA time series packages in R and Keras package under Theano platform in Python. We run our experiment on a server that has 24 cores of 2.40GHz Intel(R) Xeon(R) CPU and 32 GB
4.4.2 Data Pre-Processing and Preparation

The LSTM and ARIMA algorithms like most of other machine learning algorithms require preparing and preprocessing the raw data into a specific form in order to get the best results. First, we prepared the raw Wi-Fi dataset as a time series format for every time scale (e.g. 60, 30 and 15 minutes), such that the dataset has time with the corresponding number of occupants at that period of time. Then before feeding the data to the model, we transform the time series format into a supervised learning format by dividing it into input and output components. For time series we can achieve this by using the observation from the last time step \( (t - 1) \) as the input and the observation at the current time step \( (t) \) as the output. Because the Wi-Fi dataset that we have is not stationary, the next step of preparing the dataset is to make it stationary by removing trends in the non-stationary data. We transform the time series data into stationary time series data by differencing the data. That is the observation from the previous time step \( (t - 1) \) is subtracted from the current observation \( (t) \). The last step for preparing the time series data is to scale the data. We make the scaling for LSTM different than ARIMA models. The LSTM models like other neural networks expect data to be within the scale of the activation function used by the network. The default activation function for LSTM is the hyperbolic tangent \( \text{tanh} \), which outputs values between -1 and 1. So we scaled the time series data for LSTM into the range -1 and 1. While in the case of ARIMA models we use common logs transform for scaling, due to the popularity of log-returns it is easy to aggregate the log-returns over time.
4.4.3 Experimental Results

In our experiments we build and compare two time series models ARIMA and LSTM to forecast the number of occupants in the smart building at a specific time using three time scales, namely 15, 30 and 60 minutes duration. We build models for the whole building as well as for an individual AP level. In the case of LSTM, we further build our models in two ways, building a separate model for every time scale and building a single combined model for all the three time scales. Combined model is built as many to many architecture of LSTM, such that we feed the network with the inputs of every time scale but all the three models share the same hidden layers, and the outputs of this combined model are three outputs, each representing the output of a specific time scale model. As a clarification for the shortcomings of the labels in Figures 4.2-4.7, Com: combined model, Sep: separate model, AP: the models that are on the Access Point level, and Bld: the models that are on the Building-level. For example, LSTMCom15Bld refers to the LSTM combined model for the 15 time scale duration on the Building-level; LSTMSep30AP refers to the LSTM separate model for the 30 time scale duration on the AP-level; and ARIMA60Bld refers to the ARIMA model for the 30 time scale duration on the AP-le Building-level.

Figures 4.2 compares separate models and combined models with respect to the reduction of the root mean square error (RMSE) and computationally in terms of the number of neurons used to build such models for the three time scales on the building level and on the AP level.

We use grid search to explore the best LSTM model configurations for the number of hidden layers, number of neurons in a layer, batch size, number of epochs, and lag size parameters that play a major role in building a time series model for the LSTM algorithm. Table 4.1 shows the best
configurations to build the building-level as well as the AP-level models for the three time scales in case of LSTM combined and separate individual time scales. So according to Eq. 4.9 the total number of neurons needed for the three separate individual time scale models for building level prediction model is \((3 * 48 + 24 + 1) + (3 * 32 + 48 + 1) + (2 * 48 + 12 + 1) = 423\) neurons. And the number of neurons needed for the three separate individual time scale models in case of AP level prediction models is \((3*16 + 48 + 1) + (2*48 + 24 + 1) + (4*48 + 24 + 1) = 435\) neurons. While according to Eq. 4.10 the total number of neurons needed to build a single combined model for the three time scales in case of building level prediction is \(2*32 + 3*24 + 3 = 139\). And the total number of neurons needed to build a single combined model for the three time scales in case of AP level prediction is \(3*32 + 3*4 + 3 = 111\).

As it is shown in Figure 4.2(b), by building a single combined model for the three time scales we reduced the number of neurons by 74.48% in case of AP level, and by 67.13% in case of building level occupancy prediction. And at the same time as Figure 4.2(a) shows, by building a single combined model we reduced the RMSE by 17.14% - 41.33% in case of building level models and by 20.64% - 40.15% in case of AP level models, except an anomaly for the AP-level 60 minute time scale model where the RMSE is increased by 16%. Therefore, without loss in generality, we can observe that the LSTM combined model performs better at least from a computation resource point of view. Next, we compare combined LSTM model with the ARIMA (there is no need to compare LSTM individual time scale models because LSTM combined seems to perform better).

Figure 4.3 shows the comparison of LSTM models with the corresponding ARIMA models for the three time scales for both the building-level as well as the AP-level prediction models. LSTM models exhibit RMSE reduction by 88.2% - 93.4% in case of building level models, and by 80.9%
- 87% in case of AP level models when compared to ARIMA models. LSTM seems to outperform ARIMA.

Figures 4.5-4.7 show the effect of the parameters (number of lags, number of neurons in a layer, and number of hidden layers) on reducing the RMSE in case of LSTM models in (a) building level models and (b) AP level models. In every case we fix two parameters and set their values (to the ones that we found are the best in terms of reducing the RMSE using grid search) and start changing the third parameter. For instance, in Figure 4.5, we are fixing the number of neurons in a layer, and the number of hidden layers to the best corresponding model values that is listed in the Table I, and start changing the number of lags. It can be easily seen from Figures 4.5-4.7 that the RMSE values in case of building level models are always greater than the RMSE values for the corresponding time scale models for the AP-level prediction (as expected because of the fine grain modeling). However, as Figure 4.4 shows, there is a significant computational saving (of almost 94.28%) in terms of the number of neurons needed to build the models for building-level prediction instead of the AP level.

4.4.4 Lessons Learned

We can conclude the following based on the results presented in this paper:

- Wi-Fi is a practical way to count the number of occupants in building. This information can be utilized for emergency management as well as energy efficient applications. The main advantage of this application is that it does not require an additional infrastructure to count the number of people. One of the challenges is that those who are not carrying a device will not be counted. This could be another research project that estimate the number of occupants
that are missed and make it part of the overall accuracy calculation.

- As Figures 4.2 and 4.3 show LSTM models beat ARIMA models in all three time scale models. LSTM models reduced the RMSE by 88.2% to 93.4% on the building level models. And by 80.9% to 87% on the AP level when compared to ARIMA models.

- Comparing LSTM separate individual time scale models with the combined model, we find that training a single combined model for different time scales is better in terms of achieving less RMSE and better use of computation resources.

- In case of LSTM and ARIMA models, the decision to build a model on the building-level vs. AP-level is application dependent. The applications that care more about localization of the occupants in the building, it is better to build the models on the AP-level, while the applications that care more about the computational resources and energy savings and less about the accuracy, it is better to build the models on the building level.
Figure 4.3: Comparison between LSTM and ARIMA models.

Figure 4.4: Total number of neurons used to build the three time scales models for the building level vs. the AP level.
Figure 4.5: The effect of the lag size on reducing the RMSE in LSTM models.

Figure 4.6: The effect of the number of neurons on reducing the RMSE in LSTM models.

Figure 4.7: The effect of the number of hidden layers on reducing the RMSE in LSTM models.
Table 4.1: The best configurations for LSTM combined and separate models for the three time scales 60, 30, and 15 minutes for the whole building as well as the AP level.

<table>
<thead>
<tr>
<th></th>
<th>CombBuilding</th>
<th>Sep60Building</th>
<th>Sep30Building</th>
<th>Sep15Building</th>
<th>CombAP</th>
<th>Sep60AP</th>
<th>Sep30AP</th>
<th>Sep15AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neurons</td>
<td>32</td>
<td>48</td>
<td>32</td>
<td>48</td>
<td>32</td>
<td>16</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Layers</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Lags</td>
<td>24</td>
<td>24</td>
<td>48</td>
<td>12</td>
<td>4</td>
<td>48</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Batch size</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Epochs</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
</tr>
</tbody>
</table>

4.5 Conclusions

This study built and compared two of state-of-the-art time series prediction methods in statistics and machine learning, namely LSTM and ARIMA models based on Wi-Fi networks as an infrastructure to forecast the number of occupants at a given time and location in smart building environments. Results showed that with LSTM combined strategy we are able to reduce the number of neurons needed to build the models for every time scale by 67.13% - 74.48% when compared to individual time scale models. LSTM forecasts were considerably more accurate than those of the traditional ARIMA models, our observations revealed a RMSE reduction of 80.9% - 93.4% by LSTM. Although LSTMs are able to achieve a lower RMSE, they are extremely slow to train, can take a long time to run, often require more data to train than ARIMA models, and have lots of input parameters to tune.
The global machine learning as a service market has gained a significant impetus over the last few years; mainly due to their increasing rate of adoption in IoT smart services and automation systems. These services and systems utilize billions of IoT resource constrained devices. Data generated by those devices have been increasing exponentially. In this paper, we envision a paradigm where the cloud service providers collect big data from the resource-constrained devices and build prediction models. Those prediction models are then sent to the underlying resource-constrained devices to be run locally without necessarily being always connected to the cloud. However, models’ trust represents a serious problem in this paradigm.

We propose a heuristic that maximizes the level of trust of machine learning models by selecting a subset of models from a superset of models. During each period, our proposed heuristic switches between the subset of selected models in order to maximize the trustworthiness while respecting given reconfiguration budget and rate. Due to the difficulty of the problem in real-world scenarios, we propose an intelligent real-time heuristic that can be used in large-scale deployments of IoT resource constrained devices. The heuristic algorithm strives to make learning techniques more trustworthy since it avoids frequent reconfigurations. This also minimizes that communications overhead between the cloud and the resource constrained devices. We prove that the competitive ratio of our proposed heuristic is $O(1)$ when the reconfiguration rate is proportional to the
total time and the reconfiguration budget is constant. Therefore, our proposed heuristic achieves an optimal competitive ratio in a polynomial time approximation scheme for the problem.

To measure the performance of the proposed heuristic, we used an experimental transportation dataset to predict the number of cars as a proxy for smart cities services. We also used turbofan engine degradation simulation dataset to predict the remaining useful life of an engine as a proxy for Industrial IoT services. In the smart city service case scenario, the trust level of the selected models using the proposed algorithm is 0.7%-2.53% less compared the results obtained using ILP. Also in the IIoT service scenario, the trust level of the selected models using the proposed algorithm is 0.49% - 3.17% less compared the results obtained using ILP. These results clearly demonstrate the ability of the proposed heuristic to achieve real-time competitive results in real-life scenarios.

5.1 Introduction

5.1.1 Motivation

Most of today's Internet of Things (IoT) predictive analytics rely on a cloud-based services, in which IoT resource-constrained devices continuously send their collected data to the cloud [284]. On the cloud, Machine Learning as a Service (MLaaS) providers carry out the prediction process and provide data pre-processing, model training, model evaluation, and model update capabilities [285]. The MLaaS Market is expected to exceed $3,754 Million by 2024 at a CAGR of 42 % in the given forecast period [286]. Typical systems include: electrical power grids [287], intelligent transportation and vehicular management [288], health-care devices [289], household appliances [290], predictive maintenance systems [291] in Industrial IoT (IIoT) and many more. However, Machine Learning (ML) models can be targeted by malicious adversaries [292] due to the partic-
The performance of Machine learning models can be quantified based on their decision time, accuracy, and precision of resulting decisions [302]. However, as such models are used for more
critical and sensitive decisions, like whether a drug should be administered to a patient, or an autonomous vehicle should stop for pedestrians, it is important to ensure that they provide high accuracy and precision guarantees. Assessing learning models in terms of trustworthiness, along the traditional criteria of decision time, accuracy, and precision, establishes a tradeoff between simplicity and power [303]. Machine learning classifiers are vulnerable to adversarial examples, which are samples of input data that are maliciously modified in a way that is intended to cause a machine learning classifier to misclassify similar examples. Moreover, it is known that adversarial examples generated by one classifier are likely to cause another classifier to make the same mistake [304]. In many cases, the modifications should be small, in order to remain unnoticeable by human observer, yet the classifier still makes a mistake. In general, Attacks based on adversarial examples have one of the following goals:[305] [306] 1) Misclassification Attack: The adversary causes the classifier to output a label different from the original label, and 2) Targeted Attack: The adversary causes the classifier to output a specific misleading label.

In this paper, we envision the paradigm where resource-constrained IoT devices execute machine learning models locally, without necessarily being always connected to the cloud. This enables a number of applications scenarios, beyond the pale of the traditional paradigm, where it is not desirable to execute the model on the cloud due to latency, connectivity, energy, privacy and security concerns. In such applications, embedded sensors collect short-term measurements, and send such data through communications infrastructure to MLaaS providers, which build predictive models. Next, predictive models are sent to be hosted on IoT resource-constrained devices. However, users should be able to determine the trustworthiness of service providers in order to select them with confidence and with some degree of assurance that their service offerings will not
behave unpredictably or maliciously.

To this end, we propose a heuristic that maximizes the level of trust of ML models by selecting a subset of models from a superset of ML models. During each period, our proposed heuristic switches between the subset of selected models in order to maximize the trustworthiness while respecting given reconfiguration budget and rate. In addition, during any period, if our proposed heuristic detects that there are no trusted ML models in the superset of models, the heuristic sends a fail-safe execution alert informing the resource-constrained devices that there are no trusted ML models to be hosted on the devices. Consequently, the resource-constrained devices can fail safely as required by the application that they service. Figure 5.1 shows a general architecture for the proposed system. On the cloud side we have $M$ ML models, $model_1, model_2, \ldots, model_M$. 

Figure 5.1: General architecture for the proposed system.
5.1.2 Contributions

In this research, we formulate the problem of finding a subset of ML models that maximize the trustworthiness while respecting given reconfiguration budget and rate constraints. We also prove the problem is NP-Complete and propose a real-time heuristic that finds a near optimal solution in polynomial time.

Consequently, our proposed heuristic strives to minimize the communications overhead between the cloud and the resource constrained devices. Selected ML models are sent to resource-constrained devices to be used. Furthermore, if the proposed heuristic does not find a trusted ML model, it sends a fail-safe execution alert. This alert informs the resource-constrained devices that no trusted ML model exists in the system. As a result, the resource-constrained devices can fail safely as required by the application that they service.

We apply the proposed heuristic to two different datasets. The first dataset, based on the CityPulse project [307], is used to predict the number of vehicles as a surrogate use case for smart city services. The second data set, provided by the Prognostics CoE at NASA Ames [308], is used to predict the remaining useful life of a turbofan engine as a surrogate use case for IIoT smart services. For each use case, we built 27 LSTM models. To evaluate the performance of the proposed heuristic for detecting malicious models, we poisoned the datasets with 10 such models. We applied the swap $x$ and $100 - x$ percentiles approach as a causative adversarial attack by altering the training dataset label as described in Section 5.6.
5.1.3 Case Studies

Several case studies could be considered, in which the proposed heuristic helps to gain the best trust level. Here, we discuss two representative case studies. The first case study considers IIoT predictive maintenance while the second one considers real-time traffic flow prediction in smart cities. Figure 5.2 illustrates the trust-based model selection problem addressed in this research.

**Figure 5.2: Trust-based model selection problem.**

**Case Study: IIoT Predictive maintenance**

A predictive maintenance (PM) strategy uses machine learning methods to identify, monitor, and analyze system variables during operation. Also, PM alerts operators to preemptively perform maintenance before a system failure occurs [309]. Being able to stay ahead of equipment shut-downs in a mine, steel mill, or factory, PM can save money and time for a busy enterprise [310].
In PM, data is collected over time to monitor the state of the machine, and then analyzed to find patterns to predict failures. In many cases, it is desirable to have prediction models hosted on resource-constrained embedded devices. Predictive maintenance systems need to provide real-time control of the machines based on the deviation of the real-time flow readings from the predicted ones. In such systems, embedded sensors collect short-term state of the machine readings, which are relayed to the cloud directly through communications infrastructure, or indirectly through the use of ferry nodes. Because of its compute and store capabilities, the cloud is capable of collecting the short-term readings to build long-term big data of sensor readings. These readings are then utilized to build a PM model for each of the underlying flow sensors. The constructed models are then sent back to the flow sensors, so that they actuate their associated machines when a deviation is observed between the actual and projected flow readings. There are scenarios, in which cyber-attackers attempt to compromise PM models directly. Consequently, data that leaves its internal operating environment is subject to third party attacks. For instance, an adversary can create a causative attack to poison the learners classifications. This is possible by altering the training process through influence over the training data. Therefore, when the system is re-trained, the learner learns an incorrect decision-making function. Thus, it is important to ensure the trustworthiness of ML models before they are hosted and used on resource-constrained devices.

Case Study: Traffic Flow Prediction

Traffic flow prediction plays an important role in intelligent transportation management and route guidance. Such prediction relieves traffic congestions, reduces air pollution, and provides secure traffic conditions [311]. Traffic flow prediction heavily depends on historical and real-time traffic
data collected from various sensor sources. These sources include inductive loops, radars, cameras, mobile Global Positioning Systems, crowd sourcing, social media, etc. Transportation management and control is now becoming more data driven [312]. However, inferring traffic flow under real-world conditions in real-time is still a challenging research problem due to the computational complexity of building, training, learning and storing traffic flow models on resource constrained devices. In our proposed approach, various sensor technologies are used to automatically collect the short-term data of the traffic flow and send them to the cloud through communications infrastructure or through the use of ferry nodes. The cloud is capable of collecting the short-term readings to build long-term big data of sensor readings. These readings are then utilized to build a model. The constructed models are then sent back to be hosted on the resource constrained devices, in order to predict the traffic flow in real-time. Intelligent transportation systems are highly visible, and attacks against them result in high impact on critical infrastructure. For instance, the attacks can cause vehicular accidents, create traffic jams that affect essential services, freight movements, and daily commutes, etc. Thus, to make the traffic movement more efficient and improve road safety, road operators need to constantly monitor traffic and current roadway conditions by using an array of cameras and sensors that are strategically placed on the road network. These cameras and sensors send back real-time data to the control center [313]. The data is subject to causative adversarial attacks, which are conducted by altering the training process through influence over the training data. Consequently, causing the learner to learn an incorrect decision-making function.

Figure 5.3 illustrates the use of message ferries to collect data from resource constrained devices. Collected data is delivered to the cloud in order to build the ML models by the MLaaS service providers. Next, ferrying nodes deliver the ML models to be hosted on resource constrained
5.2 Related Work

Automatic machine learning algorithm selection, machine learning models for resource constrained devices, and trusted-based machine learning models are three areas of researches, that researchers have recently considered in their works. Here, we review some of the recent works in those areas. Figure 5.4 shows the research gap that we address in this research.

Recent research shows that machine learning models trained entirely on private data, are still vulnerable to adversarial examples [304][305][18][17] that have been maliciously altered so as to be misclassified by a target model while appearing unaltered to the human eye. Ghosh et al. [314] proposed Trusted Machine Learning (TML) framework for self-driving cars that uses principles from formal methods for learning ML models. These ML models satisfy properties in temporal logic by using model repair or the data from which the model is learned. Zhang et al. [315] propose Debugging Using Trusted Items (DUTI) algorithm that uses trusted items to detect outlier
and systematic training set bugs. The approach looks for the smallest set of changes in the training set labels, such that, the model learned from this corrected training set predicts labels of the trusted items correctly. Ribeiro et al. [316] proposed LIME algorithm, that explains the predictions of any classifier or regressor in an interpretable manner by approximating an interpretable model locally around the prediction. They also proposed a method called SP-LIME, to select representative and non-redundant predictions, that provides a global view of the model to users. They applied the proposed algorithm on both simulated and human subjects in order to decide between models. Also, they assessed trust and identified reasons for not trusting a classifier. In [80], the authors make three arguments about the trustworthiness of deep learning systems to prevent the deception of the algorithm. First, the trustworthiness should be an essential and mandatory component of a
deep learning system for algorithmic decision making. Second, the trust of a DL model should be evaluated along multiple dimensions in terms of its correctness, accountability, transparency and resilience. Third, there should be a proactive safe guard mechanisms to enforce the trustworthiness of a deep learning framework.

Researchers have proposed various automatic selection methods for machine learning algorithms. ML model selection is the problem of determining which algorithm, among a set of machine learning algorithms, is the best suited to the data [317]. Choosing the right technique is a crucial task that directly impacts the quality of predictions. However, deciding which machine learning technique is well suited for processing specific data is not an easy task, even for an expert, as the number of choices is usually very large [318]. Auto-WEKA [319] considers all 39 machine learning classification algorithms implemented in Weka to automatically and simultaneously choose a learning algorithm. Auto-WEKA uses sequential model-based optimization and a random forest regression model to approximate the dependence of a model’s accuracy on the algorithm and hyper-parameter values. Using an approach similar to that in Auto-WEKA, Komer et al. [320] developed the software hyperopt-sklearn, which automatically selects machine learning algorithms and hyper-parameter values for scikit-learn. Sparks et al.[321] proposed MLbase, an architecture for automatically selecting machine learning algorithms, that supports distributed computing on a cluster of computers by combining better model search methods, bandit methods, batching techniques, and a cost-based cluster sizing estimator. Lokuciejewski et al.[322] presented a generic framework for automatically selecting an appropriate machine learning algorithm for the compiler generation of optimization heuristics. Van Rijn et al. [323] proposed a method for automatically selecting algorithms. They addressed the problem of algorithm selection under a
budget, where multiple algorithms can be run on the full data set until the budget expires. Their method produces a ranking of classifiers and takes into account the run times of classifiers. Leite et al. [324] proposed a method called active testing for automatically selecting machine learning algorithms, that exploits metadata information concerning past evaluation results to recommend the best algorithm using a limited number of tests on the new dataset.

Other researchers work in the area of inferencing on tiny and resource-constrained IoT devices without necessarily being always connected to the cloud. Kumar et al. [325][326] developed tree and k-nearest neighbor based algorithms, called Bonsai and ProtoNN respectively, for classification, regression, ranking, and other common IoT tasks. Their algorithm can be trained on the cloud and then be hosted onto resource constrained IoT devices based on the Arduino Uno board. Bonsai and ProtoNN maintain prediction accuracy while minimizing model size and prediction costs. Motamedi et al. [19] presented a framework for synthesis of efficient Convolutional Neural Networks (CNN) inference software targeting mobile System on Chip (SoC) based platforms. They use parallelization approaches for deploying a CNN on SoC based platforms. Meng et al. [327] presented Two-Bit Networks (TBNs) approach for CNN model compression to reduce the memory usage and improve computational efficiency in terms of classification accuracy on resource-constrained devices. They utilized parameter quantization for computation workload reduction. Shoeb et al. [328] present a machine learning approach on a wearable device to identify epileptic seizures through analysis of the scalp electroencephalogram, a non-invasive measure of the brains electrical activity.

To the best of our knowledge, this is the first attempt at designing an intelligent real-time heuristic on the cloud that selects the machine learning models that should be hosted on IoT re-
source constrained devices in order to maximize the trustworthiness of the overall system.

5.3 System Model

In this paper, we assume a MLaaS provider that has $M$ ML models from which a subset needs to be selected and deployed on IoT devices for $T$ time slots. $P$ is a constant matrix of size $M \times T$, where element $p_{i,j}$ indicates the trust value obtained by model $i$ at time $j$. $B$ is the maximum number of allowed ML model reconfigurations during $T$ time slots. $A$ is a variable matrix of size $M \times T$, where element $a_{i,j} \in \{0, 1\}$. $a_{i,j} = 1$ indicates that selecting model $i$ at time $j$ is trustworthy; zero otherwise.

To compute the trust level of model $i$ at time $j$, we use Equation (5.1), which assigns a higher trust level to models that agree more with the average of all models.

$$p_{i,j} = \min \left( p_{\text{max}}, \left| \frac{O_{i,j} - \sum_{k=1}^{M} O_{k,j}}{M} \right| \right)$$  \hspace{1cm} (5.1)

Where $p_{\text{max}}$ is the maximum attainable trust level and $p_{i,j}$ is the trust level of model $i$ at time $j$, $O_{i,j}$ is the output of model $i$ at time $j$, and $O_{i,j} \in \mathbb{R}$.

5.4 Problem Formulation

The goal of this work is to maximize the trust level gained by selecting a subset of ML models from a superset of models to be hosted on resource-constrained devices for a period of time $R$, where $0 \leq R \leq T$. The number of reconfigurations is limited to $B$ and the maximum rate of
reconfiguration is limited to $R$. We formulate the problem using Integer Linear Programming (ILP) as follows:

$$\max \sum_{j=0}^{T} \sum_{i=1}^{M} a_{i,j} \cdot p_{i,j} \quad (5.2)$$

subject to

$$\sum_{i=1}^{M} a_{i,j} = 1 \quad \forall j \in 0 \ldots T \quad (5.3)$$

$$a_{i,j} \in \{0, 1\} \quad \forall i \in 1 \ldots M$$

$$\forall j \in 0 \ldots T \quad (5.4)$$

$$\frac{1}{2} \cdot \sum_{i=1}^{M} \sum_{j=0}^{T-1} |a_{i,j} - a_{i,j-1}| \leq B \quad (5.5)$$

$$\frac{1}{2} \cdot \sum_{i=1}^{M} \sum_{j=k}^{k+[\frac{T}{B}]} |a_{i,j} - a_{i,j-1}| \leq R \quad (5.6)$$

The first constraint in (5.3) is for making sure that only one ML model is selected at each time slot. The second constraint in (5.4) indicates that this formulation is combinatorial, where result values can either be 0 or 1. A value of 1 indicates the ML model is trusted in this time slot while 0 indicates that it is not. In order to comply with the maximum number of allowed reconfigurations ($B$), the third constraint in (5.5) is used. The fourth constraint in (5.6) restricts the solution to adhere to the models’ maximum reconfiguration rate $R$ (i.e., maximum number of reconfiguration per time unit).
5.4.1 Lower Bound

To find a lower bound solution, we propose the **Splice heuristic** shown in Algorithm (2). The heuristic accepts $A$, a matrix of size $M \times T$ where element $a_{i,j}$ represents the trust level of model $i$ at time slot $j$. Initially, the heuristic considers $A$ as one unselected segment. Next, the heuristic uses three steps iteratively. In the first step and for each unselected segment that is at least $R$ in length, the heuristic finds model (row) $k$ with the longest consecutive sequence of 1s (i.e. highest trust level). In the second step, the segment that has the highest trust level is marked as selected. Additionally, row $k$ is selected by setting all values in all rows except row $k$ to 0 while values in row $k$ are set to 1. The third step merges adjacent selected segments (from the previous rounds) into a single selected segment if they share the same selected model. These three steps are repeated until at most $B$ segments are selected or on unselected segments are left. Finally, the heuristic identifies unselected segments, if such segments exist. For each unselected segment, the heuristic finds the trust level using the highest trust level from a selected adjacent segment, if one exists. Finally, the heuristic compares the trust level resulting from the adjacent ML models (if they exist) and chooses the one with the highest trust level.

The example in Figure 5.5 illustrates the details of our proposed Splice heuristic. In this example, we assume that $R = 4$, $B = 2$. Consequently, the heuristic selects $B + 1 = 3$ segments that maximize the trust level. The first section covers time slots $T_1 - T_4$ with selected ML model $M_3$. $M_2$ is selected in the second segment, which covers time slots $T_7 - T_{10}$. Finally, the last segment has $M_4$ selected in time slots $T_{13} - T_{16}$. After that the heuristic determines which ML model to use for the remaining unselected segments. For time slots $T_5 - T_6$, $M_2$ is selected based on the
selected adjacent segment to the right. In addition, for time slots $T_{11} - T_{12}$, $M_4$ is selected based on the selected adjacent segment to the right.

### 5.4.2 Upper Bound

We relax the ILP formulation presented in Section 5.4 to an LP problem by replacing constraint (5.4) with

$$a_{i,j} \in [0, 1] \quad \forall i \in 1 \ldots M$$ (5.7)

This relaxed formulation produces an upper bound solution for our problem.

### 5.4.3 Competitive Solution

To produce a competitive solution, we propose the **Fixing Heuristic** shown in Algorithm (3). The algorithm accepts matrix $A$, an $M \times T$ matrix where element $a_{i,j}$ represents the trust level of model $i$ at time slot $j$. The heuristic selects a maximum of $B + 1$ ML models (i.e., resulting in a maximum
**Algorithm 2** Proposed Splice Heuristic to find a lower-bound solution

Input: Matrix $A$ of size $M \times T$ where element $a_{i,j}$ represents the trust level of model $i$ at time slot $j$, maximum number of allowed reconfigurations ($B$), maximum reconfiguration rate $R$.

Output: matrix $A$ with each column having only one value as 1 to indicate the selected ML model at the given time slot.

1: Mark $A$ as one unselected segment
2: Set $i = 0$
3: while $i \leq B$ AND number of unselected segments $> 0$ do
4:     Set flag = False
5:     Identify unselected segment $j$ with at least $R$ columns that has the longest consecutive sequence of 1s in row $k$.
6:     if Segment $j$ exists then
7:         Set all entries of row $k$ to 1, and set all entries of other rows of segment $j$ to 0
8:         Mark segment $j$ as selected
9:     if $w$ is a selected segment that is adjacent to $j$ and both have 1s in the same row then
10:        Merge segments $w$ and $j$
11:        Set flag=True
12:     end if
13: end if
14: if flag=False then
15:     Set $i = i + 1$
16: end if
17: end while
18: Merge adjacent unselected segments into one
19: for every unselected segment $j$ do
20:     Set leftSum = 0, rightSum = 0, selectedRow = 0
21:     if there is a selected segment $w$ with selected row $k$ left adjacent to segment $j$ then
22:         Set leftSum = sum of values of row $k$ in segment $j$
23:         Set selectedRow = $k$
24:     end if
25:     if there is a selected segment $w$ with selected row $z$ right adjacent to segment $j$ then
26:         Set rightSum = sum of values of row $z$ in segment $j$
27:         if rightSum > leftSum then
28:             Set selectedRow = $z$
29:         end if
30:     end if
31:     Set all entries of selectedRow of segment $j$ to 1 and all entries of the other rows to 0
32: end for
33: Return $A$ as the best solution.
Algorithm 3 Proposed Fixing Heuristic to produce a competitive solution

Input: Matrix $A$ of size $M \times T$ where element $a_{i,j}$ represents the trust level of model $i$ at time slot $j$, maximum number of allowed reconfigurations ($B$), maximum reconfiguration rate $R$.
Output: matrix $A$ with each column having only one value as 1 to indicate the selected ML model at the given time slot.

PART I - Fixing

1: Set $X = A$
2: Run the Splice heuristic on matrix $X$
3: Set PreviousTrustLevel = element-wise sum of $A$ & $X$ where & is the bitwise AND operator
4: Set $X = A$ and apply linear programming to generate a fractional solution
5: Compute CurrentTrustLevel using PART II
6: while $H > 0$ AND equation 1 through 3 are satisfied AND CurrentTrustLevel > PreviousTrustLevel do
7: Set PreviousTrustLevel = CurrentTrustLevel
8: Set $H = H - \epsilon$
9: Set $X = A$ and apply linear programming to generate a fractional solution
10: Compute CurrentTrustLevel using PART II
11: end while
12: Return $X$ as the best solution.

PART II - Computing CurrentTrustLevel

13: Set rowNum = -1
14: for $t$ from $t_0$ to $T$ do
15: if maximum value from column $t$ of matrix $X \geq H$ then
16: Set this maximum value to 1 and set rest of values in column $t$ to 0
17: Set rowNum = row number of the maximum value
18: else if rowNum = -1 then
19: Set this maximum value to 1 and set rest of values in column $t$ to 0
20: else
21: Set the value at rowNum to 1 and set rest of values in column $t$ to 0
22: end if
23: end for
24: Set currentTrustLevel = element-wise sum of $A$ & $X$ where & is the bitwise AND operator
of $B$ model reconfigurations) to be used during $T$ in order to maximize the overall trust level. The proposed heuristic employs two constants; namely, a threshold $H$ that represents the maximum trust level selected from the fractional solution ($0 < H < 1$), and epsilon $\epsilon$ which is a small value that is subtracted from the value of $H$ during each iteration of the fixing process ($0 < \epsilon < 0.1$).

The proposed heuristic finds the lower bound solution first using the Splice heuristic 2 on matrix $A$. Next, the proposed fixing heuristic applies Linear Programming (LP) on matrix $A$ to find a fractional upper bound solution using $H$. Actually, the ML model with the highest trust level in each time slot of $A$ is compared with $H$. The highest trust level is rounded to 1 if it is greater than or equal to $H$ while setting all other ML models to 0 during that time slot. The same process is applied for trust levels less than $H$. If the highest trust level is less than $H$ in any time slot, the selected ML model in the previous time slot is selected for this time slot and is rounded to 1 while other ML models are set to 0. After converting the matrix into a binary one (i.e., 0 or 1 entries), the upper bound solution is computed by counting the number of entries in $A$ that are set to 1. If the upper bound solution is found to be greater than the lower bound solution, the lower bound solution is set to the value of the upper bound solution. Also, $H$ is reduced by $\epsilon$ and the upper bound solution is recomputed in the hope of finding a better solution. This process is repeated as long as the upper bound solution is improved.

5.4.4 Proof of NP-Completeness

In this section, we show that the problem discussed in this paper can be reduced from the decision version of the set cover problem, which is known to be NP-Complete.

We define the universe $\mathcal{U}$ as a set of tuples $(i, j), i, j \in T$ and $i \leq j$. Each tuple $(i, j)$ represents
a time interval that starts at time $i$ and ends at time $j$ during which the system uses the same model without any reconfigurations. We also define $S$ as a family of subsets of $\mathcal{U}$. The union of $S$ results in a period that covers $\mathcal{U}$. In other words, the union of $S$ results in a period that starts at time $0$ and ends at time $T$. Now the cardinality of $S$ is represented as follows:

$$0 \leq ||S|| \leq \left[ \sum_{i=1}^{T} \binom{T}{i} \right] \cdot M$$

(5.8)

If $k$ represents the maximum number of model reconfigurations, the objective of our problem is to find $k$ subsets from $S$ while maximizing the total trust level. This problem is similar to the decision version of the set cover problem. The universe $\mathcal{U}$ and the set $S$ of our problem are the same as the universe $\mathcal{U}$ and set $S$ in the set cover problem. However, in our problem, every element is a tuple. The maximum number of model reconfigurations $k$ is the same as the integer number $k$ in the set cover problem. Consequently, the problem introduced in this paper is NP-Complete.

5.5 Worst-Case Analysis (Competitive Ratio Analysis)

The performance of our proposed fixing heuristic is at least as good as that of the splice heuristic. Consequently, the worst case scenario is encountered when the proposed heuristic performs as the splice heuristic. When the maximum number of allowed reconfigurations is set to $B$, the splice heuristic finds $(B + 1)$ segments, each of length $R$, that provide the maximal trust level.

**Proposition 1.** For any configuration $X$, the splice heuristic’s worst-case performance has a competitive ratio of $O(1)$ when $R$ is proportional to $T$ and $B$ is constant.
Proof. Let ALG be the splice heuristic and OPT be the optimal algorithm. The first part of the splice heuristic (Algorithm 2), specifically steps 1 through 17, finds the segments with the longest consecutive sequence of 1s. Actually, both ALG and OPT select those segments since they have the largest sum of values (i.e., maximum trust levels). Specifically, those segments have a total length of \( R(B + 1) \). However, the two approaches differ in the rest of the solution, which is the unselected segments in ALG. Now, at the end of the first part and in the worst-case scenario, Algorithm (2) may already performed \( B \) reconfigurations and cannot use more reconfigurations. In other words, for every unselected segment, Algorithm (2) can only use either of the selected model in the adjacent selected segments but never a different model. Consequently, in the worst-case scenario, the two models in the selected segments adjacent to the unselected segment are different. Thus, the second part of Algorithm (2), specifically steps 17 through 32, will pick the model that has the largest sum in the unselected segment. In the worst-case scenario, both the left and right adjacent selected segments may have the same value when both used in the unselected segment and therefore Algorithm (2)’s maximum loss is half the segment length. However, the loss can never be less than the half the segment length. Mathematically, in the second part of the solution, OPT achieves a maximum of \( T - R(B + 1) \) while ALG achieves a minimum of \( \frac{1}{2}[T - R(B + 1)] \). Consequently, the following proof is concluded:
Figure 5.6: The pipeline of system implementation.

Competitive Ratio = \( \frac{ALG(X)}{OPT(X)} \)

\[
= \frac{R(B + 1) + \frac{1}{2}[T - R(B + 1)]}{R(B + 1) + [T - R(B + 1)]}
\]

\[
= \frac{\frac{1}{2}T + \frac{1}{2}R(B + 1)}{T}
\]

\[
= \frac{1}{2}T + \frac{1}{2}R(B + 1)
\]

\[
= \frac{1}{2} \left( \frac{T}{T} + \frac{RB}{T} + \frac{R}{T} \right)
\]

\[
= O\left( \frac{R(B + 1)}{T} \right)
\]

This competitive ratio is \( O(1) \) when \( R \) is proportional to \( T \) and \( B \) is constant.

5.6 Experimental Setups

In order to evaluate the performance of the proposed fixing heuristic, we design and implement the system shown in Figure 5.6. In our experiments, we focus on two case studies that serve as proxies for smart city and industrial IoT services.
5.6.1 Datasets

The first case study is a proxy for smart city services in which the City Pulse EU FP7 project [307] dataset is used for traffic prediction. This dataset conveys the vehicular traffic volume collected from the city of Aarhus, Denmark, observed between two points for a set duration of time over a period of 6 months.

In the second case study is a proxy for industrial IoT services in which the Turbofan engine degradation simulation dataset, provided by the Prognostics CoE at NASA Ames [308], is used for predicting the remaining useful life of engines. Engine degradation simulation was carried out using a C-MAPSS tool. The goal is to predict the remaining useful life, or the remaining number of cycles before the turbofan engine reaches a level that no longer performs up to requirements. The requirement is based on data collected from sensors located on the turbofan and also on the number of cycles completed. The prediction helps to plan maintenance in advance. The training data consists of multiple multivariate time series with "cycle" as the time unit, together with 21 sensor readings for each cycle. Each time series can be assumed as being generated from a different engine of the same type. The testing data has the same data schema as the training data. The only difference is that the data does not indicate when the failure occurs. Finally, the ground truth data provides the number of remaining work cycles for the engines in the test data.

5.6.2 Setup

Each dataset is divided into training and test subsets. We divide each dataset to build 27 deep LSTM models (17 of those models are non-malicious while 20% of the remaining 10 models are injected with causative attacks before building the LSTM models). Specifically, we use the swap
$x$ and $100 - x$ percentiles attack model as a causative attack to intentionally poison the learners’ classifications by altering the labels of the training dataset. In the swap $x$ and $100 - x$ percentiles attack, the $x$ percentile value is exchanged with the $100 - x$ percentile value. As an example of the swap $x$ and $100 - x$ percentiles attack, consider the numeric dataset in Figure 5.7. To find the $i^{th}$ percentile, we need to sort the values in the unsorted list in ascending order. Next, we multiply $i\%$ by the total number of items in the list (i.e., 10 items). Now, for example, let us find $20^{th}$ and $80^{th}$ percentiles in the list. $20^{th}$ percentile = $0.2 \times 10 = 2$ (item index), which is value 174 in the list. $80^{th}$ percentile = $0.8 \times 10 = 8$ (item index), which is value 188 in the list. Now, to swap the $x$ and $100 - x$ percentiles in this dataset, every 174 will be replaced with 188, and every 188 will be replaced with 174 in the region in which we want to introduce the swap $x$ and $100 - x$ percentiles attack.

![Diagram showing the swap of 20th and 80th percentiles](image)

Figure 5.7: Example of swap $x$ and $100 - x$ percentiles attack model.
During the training phase, we train every model using different configurations. Each configuration includes different values for the number of hidden layers, the number of neurons in each layer, and activation functions. Finally, we select the configuration that gives the best accuracy. After building the models, we test them using the two datasets. Every row in the traffic dataset represents the number of vehicles during a specific hour. However, a row in the Turbofan engine dataset represents the remaining useful life during a specific cycle. Next, we utilize $\lambda$ standard deviations strategy, which is inspired by the Six Sigma strategy [329] to exclude the malicious models by identifying and removing the causes of defects and minimizing variability using statistical methods; namely, mean and standard deviation as shown in Equations (5.9) and (5.10). Thus, leading to better trust prediction models.

$$Out_{Upper} = \mu + \lambda \times \sigma$$  \hfill (5.9)  

$$Out_{Lower} = \mu - \lambda \times \sigma$$  \hfill (5.10)

Every time step, we compute $\mu$, which is the mean of the outputs of all models. Also, we compute $\sigma$, the standard deviation of the outputs of all models. $\lambda$ defines the model exclusion strategy (i.e., any model that has an output that is $> \mu + \lambda \times \sigma$ or $< \mu - \lambda \times \sigma$ is excluded). The $\lambda$ standard deviations strategy produces a trust matrix of size $M \times T$, with 1 indicating a trusted model and 0 indicating a malicious model. The resulting matrix is then used as the input (i.e., matrix $A$) for the proposed fixing heuristic.
5.7 Experimental Results

In this section, we discuss the results of using the proposed algorithm along with the lower bound and upper bound algorithms on the two datasets mentioned in the previous section. We conducted a number of experiments as explained next.

5.7.1 Traffic Flow Volume Prediction

In our first experiment, we studied the Root Mean Square Error (RMSE) of the models selected using our proposed fixing heuristic vis-à-vis the individual models. We set the reconfiguration budget $B$ to 7 as shown in Figure 5.8. As the figure shows, the proposed fixing heuristic results in 11% - 66.95% less RMSE when compared to the individual models.

![Figure 5.8: RMSE of the selected models using the fixing heuristic vs. individual models.](image)

In our second experiment, the trust levels resulting from the three heuristics are compared under
different reconfiguration budgets as illustrated in Figure 5.9. The figure shows the confidence interval for 5 replications. In each replication, the malicious model is applied on a different model (e.g., $MM_1$, $MM_2$, ... , or $MM_n$). In this experiment, we set $\lambda$ to 0.85, number of malicious models $C$ to 1, and $M$ to 7. The number of non-malicious models is $M - C$.

In our third experiment, the trust level of the selected models is studied as the number of models $M$ is varied (5, 9, and 17) as illustrated in Figure 5.10. In this experiment, we set $\lambda$ to 0.85 and $C$ to 1. In addition, the figure indicates that as $M$ is increased, the trust level of the selected models is increased too.
Figure 5.10: The effect of the number of selected models on the trust level.

Figure 5.11 shows the results of our fourth experiment. In this experiment, the effect of using different values of $\lambda$ (0.8, 0.85, 0.9, 0.95) on the trust level of the selected models is analyzed. In this experiment, we set $C$ to 1 and $M$ to 7. The figure shows this effect given different reconfiguration budgets $B$. The figure indicates that as $\lambda$ is increased, the trust level of the selected models is increased too.

Figure 5.12 shows the effect of the number of the malicious LSTM models $C$ (3, 5, and 7) on the trust level of the selected models for different values of $\lambda$ (0.8, 0.85, 0.9). The figure also shows the actual number of malicious LSTM models versus the identified number of malicious LSTM models. In this experiment, we set $M$ to 17 and $B$ to 7.
5.7.2 Predictive Maintenance in IIoT

In our first experiment, we studied the Root Mean Square Error (RMSE) of the models selected using our proposed fixing heuristic vis-à-vis the individual models. We set the reconfiguration budget $B$ to 7 as shown in Figure 5.13. As the figure shows, the proposed fixing heuristic results in 0.5% - 15% less RMSE when compared to the individual models.

In our second experiment, the trust levels resulting from the three heuristics are compared under different reconfiguration budgets as illustrated in Figure 5.14. The figure shows the confidence interval for 5 different replications. In each replication, the malicious model is applied to a different model (e.g. $MM_1$, $MM_2$, …, or $MM_n$). In this experiment, we set $\lambda$ to 0.75, number of malicious
models $C$ to 1, and $M$ to 7. The number of non-malicious models is $M - C$.

In the third experiment, the trust level of the selected models is studied as the number of models $M$ is varied (5, 7, and 9) as illustrated in Figure 5.15. In this experiment, we set $\lambda$ to 0.75 and $C$ to 1. In addition, the figure indicates that as $M$ is increased, the trust level of the selected models is increased too.

Figure 5.16 shows the results of our fourth experiment. In this experiment, the effect of using different values of $\lambda$ (0.7, 0.75, 0.8) on the trust level of the selected models is analyzed. In this experiment, we set $C$ to 1 and $M$ to 7. The figure shows this effect given different reconfiguration budgets $B$. The figure indicates that as $\lambda$ is increased, the trust level of the selected models is increased too.

Figure 5.17 shows the effect of the number of the malicious LSTM models $C$ (1, 2, and 3) on
the trust level of the selected models for different values of $\lambda$ (0.7, 0.75, 0.8). The figure also shows the actual number of malicious LSTM models versus the identified number of malicious LSTM models. In this experiment, we set $M$ to 7 and $B$ to 7.

5.7.3 Lessons Learned and Results Discussion

We can conclude the following based on the results presented in the previous section:

1. It is important to evaluate machine learning models used in critical and sensitive decisions in terms of trustworthiness and reliability. Additionally, other traditional criteria of machine learning model evaluation must be considered (e.g., accuracy, run time, etc.).

2. Our proposed fixing heuristic strives to maximize the trust level while not affecting the accuracy of the selected models, as Figures 5.8 and 5.13 indicate.
3. Figures 5.9 and 5.14 show that our proposed fixing heuristic is able to obtain a trust level that is 0.7% - 2.53% lower than that obtained by the upper bound solution in smart city case study, and 0.49% - 3.17% lower than that obtained by the upper bound solution in IIoT case study. Figures 5.9 and 5.14 also indicate that by increasing the reconfiguration budget, the trust level is increased. However, there is a limit beyond which increasing the number of reconfigurations does not increase the trust level.

4. Figures 5.10 and 5.15 indicate that increasing the number of selected models lead to an increase in the trust level of the overall system.
Figure 5.15: The effect of the number of selected models on the trust level.

5. Figures 5.11 and 5.16 indicate that increasing $\lambda$, the number of the excluded models is decreased. However, increasing $\lambda$ beyond a specific threshold may lead to the use of malicious models. On the other hand, using a small value for $\lambda$ leads to excluding more models, which might not be malicious.

6. Figures 5.12 and 5.17 indicate that increasing the number of malicious models leads to a decrease in the trust level of the overall system.

5.8 Conclusions and Future Works

In this paper, we envisage a paradigm in which resource-constrained IoT devices execute machine learning algorithms locally, without necessarily being always connected to the cloud. This
paradigm is desirable in systems with strict latency, connectivity, energy, privacy and security requirements. Also, we present the need to evaluate the level of trustworthiness of machine learning models built by different service providers. We formulate the problem of finding a subset of ML models that maximize the trustworthiness while respecting given reconfiguration budget and rate constraints. We also prove the problem is NP-Complete and propose a real-time fixing heuristic that finds a near optimal solution in polynomial time.

To measure the performance of the proposed fixing heuristic compared to ILP, we applied our proposed fixing heuristic to two different case studies. The traffic flow volume dataset, to predict the number of vehicles as a proxy case study for smart cities services. The turbofan engine degradation simulation dataset, to predict the remaining useful life for the engine as a proxy for
industrial IoT services. Our proposed fixing heuristic achieves 0.7% - 2.53% less trust level compared to the ILP solution in the smart city service case study. In addition, Our proposed fixing heuristic achieves 0.49% - 3.17% less trust level compared to the ILP solution in the industrial IoT service case study.

There are a number of avenues of future work that we would like to explore. Although in this paper we only use the LSTM algorithm to build the models, other types of models can be studied (e.g., CNN, deep neural networks, and SVM). It would be interesting to perform a comparative study on these models and their immunity to adversarial attacks when used in conjunction with our proposed fixing heuristic. Additionally, we would like to investigate other potential applications of our proposed heuristic in speech, video, and medical domains, as well as recommendation systems.

Figure 5.17: The effect of malicious models on the trust level.
The promise of the smart services is a world of appliances that anticipate your needs and do exactly what you want them to at the touch of a button. To some facility executives, the term smart services may conjure up images of a smart environment of the future from a science-fiction movie. But the reality is smart services exist today, and their number is growing. There are smart homes, smart buildings, smart offices, smart health care facilities, smart hospitality complexes, smart educational facilities and many other types of smart services. Increasing connectivity may one day connect everything around us in the real life, from your placemats to your plant vases. The more connected objects, the more functionality the smart services will possess. Smart service field covers a broad spectrum of technologies, including prediction, decision-making, robotics, smart materials, wireless and sensor networking, multimedia, mobile computing, and cloud computing. With the massive amount of streaming data generated and captured by smart service appliances, sensors and devices, there is no doubt at all that ML, big data analytics and streaming processing will play a critical role in enabling the delivery of smart services. Within the machine-learning domain, deep learning is emerging as a superior new approach that is much more effective than any rule or formula used by traditional machine learning. Furthermore, it can even alleviate engineers from the task of defining features. In this However, there are still significant challenges that need to be addressed to utilize DL technology.

In this dissertation, the main goal is to explore the efficacy of deep learning in support of IoT
smart services. Successful application of deep learning requires careful and sometimes very expensive hyper parameter searches, tuning, and testing. Manual parameter setting, and grid search approaches can ease setting these important parameters. But, these two approaches can be very time-consuming, and resource constrained devices obviously cannot handle these computational loads. In order to address this computational challenge, we propose a two prong solution. First, designing optimization methods to help find optimal parameter settings. Towards this we show that the PSO technique holds a great potential; thus, saving valuable computational resources during the tuning process of deep learning models. Second, we investigate techniques to optimize execution of deep learning models that have built by cloud hosted MLaaS providers to be run locally on resource-constrained devices without necessarily being always connected to the cloud. Note that, prediction models’ trust represents a potentially serious threat in this paradigm. We thus propose an intelligent real-time heuristic algorithm that maximizes the level of trust of deep learning models. The heuristic algorithm strives to make learning techniques more trustworthy since it avoids frequent reconfigurations.

In the future, we will primarily focus on the following:

**Context-awareness Computing in Smart services:**

In the smart service environment, there exists a vast amount of raw data being continuously collected about the human activities and behaviors. It is important to develop techniques that convert this raw data into usable knowledge. Usually the contextual information related to human behaviors and activities is very complex and it is not formalized in a standard way, modeling human behaviors is thus extremely challenging due to the complex physiological, psychological and behavioral aspects of human beings. In this research the focus will be on collection, modelling,
reasoning, and distribution of context techniques to understand the complex dependencies between the apps and human behavior for an accurate and precise prediction when an individual faces difficulty while performing everyday activities. The foundation of context-aware systems have a significant advantage in smart service applications such as ambient assisted living to support heart rate monitoring, medication prompting, generation of agenda reminders, weather alerts, and emergency notifications.

**Personal Big Data Stream Management in Smart services:**

Nowadays, a wide range of sensing technology in the smart services can be used to gather different types of data and generally at a reasonable cost. In the world of big data, large and heterogeneous datasets are two major challenges for the traditional approaches like trial and error to extract meaningful information from this dataset. In this research my focus will be on the problems and opportunities arising from the Big Data of smart service environments. Analyzing these massive datasets requires new technologies to store, organize and process big data effectively, it needs more powerful, high-performance processors that provide the tools to uncover the insights in big data. It also requires flexible cloud computing and virtualization, software such as Apache Hadoop and Spark. It requires providing appropriate machine learning techniques which differs from the traditional approaches for effective and efficient solutions.
REFERENCES


155


160


S. Murphy, This is your future smart home and no, it doesn’t look like ’The Jetsons’. Jan. 2015. [Online]. Available: https://mashable.com/2015/01/10/future-smart-home/.


