# Machine Learning Methods for Efficient Data Reduction and Reconstruction in the concept of Internet of Things

**Čulić Gambiroža, Jelena**

UNIVERSITY OF SPLIT
FACULTY OF ELECTRICAL ENGINEERING, MECHANICAL ENGINEERING
AND NAVAL ARCHITECTURE

**Jelena Čulić Gambiroža**

# MACHINE LEARNING METHODS FOR EFFICIENT DATA REDUCTION AND RECONSTRUCTION IN THE CONCEPT OF INTERNET OF THINGS

DOCTORAL THESIS

Split, 2023.

UNIVERSITY OF SPLIT
FACULTY OF ELECTRICAL ENGINEERING, MECHANICAL ENGINEERING
AND NAVAL ARCHITECTURE

**Jelena Čulić Gambiroža**

# *Machine Learning Methods for Efficient Data Reduction and Reconstruction in the concept of Internet of Things*

DOCTORAL THESIS

Split, 2023.

The research reported in this thesis was carried out at University of Split, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture.

Supervisor: Full Professor Mario Čagalj, PhD, FESB, University of Split, Croatia
Disertation number: 195

BIBLIOGRAPHIC INFORMATION

Keywords: Internet of Things, IoT, Sensors, Machine learning, Time series, Signal pattern recognition, Energy efficiency
Scientific area: Technical sciences
Scientific field: Computer science
Scientific branch: Information processing
Institution of PhD completion: University of Split, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture
Supervisor of the thesis: Full Professor Mario Čagalj, PhD
Number of pages: 119
Number of figures: 54
Number of tables: 16
Number of references: 139

Committee for assessment of doctoral dissertation:

1. Full Professor Dinko Begušić, PhD, FESB, University of Split, Croatia
2. Adjunct Professor Darko Huljenić, PhD, Ericsson Nikola Tesla, Zagreb and FER, University of Zagreb, Croatia
3. Associate Professor Ivo Stančić, PhD, FESB, University of Split, Croatia
4. Assistant Professor Maja Braović, PhD, FESB, University of Split, Croatia
5. Senior Research Associate Ivana Nižetić Kosović, PhD, Split, Croatia

Committee for defence of doctoral dissertation:

1. Full Professor Maja Štula, PhD, FESB, University of Split, Croatia
2. Adjunct Professor Darko Huljenić, PhD, Ericsson Nikola Tesla, Zagreb and FER, University of Zagreb, Croatia
3. Full Professor Damir Krstinić, PhD, FESB, University of Split, Croatia
4. Associate Professor Ivo Stančić, PhD, FESB, University of Split, Croatia
5. Assistant Professor Maja Braović, PhD, FESB, University of Split, Croatia

Dissertation defended on: 17. 11. 2023.

# Machine Learning Methods for Efficient Data Reduction and Reconstruction in the concept of Internet of Things

## Abstract:

The concept of Internet of Things (IoT) has grown in recent years, and the number of IoT devices is rapidly increasing. Consequently, the amount of collected and stored data is increasing, thus leading to Big data and its related challenges. While individual IoT sensors consume a relatively small amount of energy, they are mostly battery powered and numerous, which limits their lifetime and creates a great load on the backend systems. The main goal of herein presented research is to detect places in the IoT network where data velocity and volume reductions can be made while preserving the value and variety of the data. Through the research, three challenges were tackled. The first two challenges address the issue of sensor online time reduction and the effect on battery life prolongation. The third challenge mitigates risks introduced by reducing the amount of data circulating through the IoT network, as well as simplifying the configuration process when adding new sensors to an IoT network. The thesis is structured accordingly.

In the first part of research, we propose a dynamic monitoring frequency (DMF) algorithm that aims at collecting data only when sensor readings change by more than a tolerable value between consecutive readings. Thus, a sensor is turned on only when a change in monitored phenomenon value exceeds a predefined threshold. Two algorithms are analysed, namely statistical and machine learning. DMF shows notable performance, resulting either with up to $\sim 70\%$ less missed readings, or collects up to $\sim 40\%$ less data compared to the baseline algorithm that collects data with static monitoring frequency.

The focus of the second part of the research are low-cost sensors as they need to preheat for several minutes to reliably collect monitored value from the environment. However, instead of waiting for a sensor to heat up, a transient, i.e., a data trend that the sensor collects while heating up is analysed. It is shown that long short-term memory (LSTM) neural network can be used to learn and later predict actual value from a part of the transient. This way, instead of being constantly online or fully preheating, the sensor needs to be turned on for only 20 seconds and then sleep for 120 seconds. Our approach decreases energy consumption with high accuracy up to 85% compared to a system where sensors are constantly online, and more than 50% compared to a system where a sensor collects actual values instead of a part of the transient.

In the third part of the research, different approaches for signal type classification that can be used to recognise a signal type being read from an IoT sensor are evaluated and compared. This is performed by using the machine learning methods for modeling a signal represented as raw time series data. Three machine learning classification approaches are taken into a consideration, namely one class, two class and multi class. According to the results of the evaluation, the most accurate multi class random forest algorithm can correctly classify unknown signals in $\sim 75\%$ of the cases based on only 20 consecutive sensor readings. Moreover, multi class random forest can detect two most probable classes of monitored signal with the accuracy of $95\%$.

**Keywords:**
Internet of Things, IoT, Sensors, Machine learning, Random forest, LSTM, Time series, Signal pattern recognition, Energy efficiency

## Metode strojnog učenja za učinkovito smanjenje količine podataka i njihovu rekonstrukciju u konceptu Interneta stvari

**Sažetak:**

Koncept Interneta stvari (engl. *Internet of Things - IoT*) u posljednje vrijeme sve se više širi, a broj IoT uređaja sve se više povećava. Posljedično, povećava se i količina prikupljenih i pohranjenih podataka, što dovodi do izazova prepoznatih unutar koncepta Velikih podataka (engl. *Big data concept*). Pojedinačni IoT senzori troše relativno malu količinu energije, no napajaju se uglavnom baterijama koje imaju ograničen vijek trajanja. Osnovni cilj ovog istraživanja je pronaći područja u IoT mreži na kojima se može smanjiti brzina prikupljanja i volumen podataka, pritom uzimajući u obzir raznolikost podataka i njihovu vrijednost. U istraživanju su adresirana tri izazova. Prvi i drugi izazov bave se smanjenjem vremena rada senzora što rezultira produživanjem životnog vijeka baterije. Treći izazov smanjuje rizike koji proizlaze iz smanjenja količine podataka koji se šalju kroz IoT mrežu, te također pojednostavljuje konfiguracijski proces prilikom dodavanja novog IoT senzora u mrežu. Disertacija je strukturirana u skladu s 3 navedena izazova.

Kroz prvi dio istraživanja, predložen je algoritam s dinamičkom frekvencijom/periodom prikupljanja podataka (DMF) koji prikuplja podatke samo kada se očekuje značajna promjena između uzastopnih očitanja. Stoga se senzor uključuje samo kada se očekuje da će se promatrana vrijednost promijeniti više od toleriranog raspona. Pritom se analiziraju i uspoređuju statistički pristup i pristup strojnog učenja. U usporedbi s algoritmom koji prikuplja podatke sa statičkom frekvencijom/periodom prikupljanja, DMF algoritam pokazuje značajno poboljšanje te rezultira s do $\sim 70\%$ manje nedetektiranih promjena ili s do $\sim 40\%$ manje prikupljenih podataka.

Drugi dio istraživanja bavi se niskobudžetnim senzorima kojima je potrebno prethodno zagrijavanje prije nego postignu stabilne radne uvjete. Na primjeru senzora plina, umjesto čekanja da senzor postigne stabilne radne uvjete, analizira se početni dio tranzijenta, odnosno trend podataka koje senzor prikuplja dok se zagrijava. Pokazano je da je primjenom Long Short-Term Memory (LSTM) neuralne mreže moguće predvidjeti stvarnu razinu plina iz početnog dijela tranzijenta. Na taj način, umjesto da se senzor u potpunosti zagrije, dovoljno ga je uključiti na 20 sekundi i zatim ga staviti u stanje mirovanja 120 sekundi. S visokom preciznošću ovakav pristup smanjuje potrošnju energije na strani senzora do 85%, u usporedbi sa sustavima u kojima je senzor konstantno upaljen, odnosno više od 50% u usporedbi sa sustavima kada se senzor u potpunosti zagrijava.

Kroz treći dio istraživanja uspoređuju se različiti pristupi klasifikacije tipa signala koji se mogu koristiti za prepoznavanje tipa signala kojeg prikuplja pojedini senzor. Pritom se koriste metode strojnog učenja. Analizirana su tri pristupa klasifikacije signala, odnosno prepoznavanje tipa signala na temelju jedne klase, na temelju dvije klase i na temelju više klasa. Rezultati pokazuju da najtočniji pristup, klasifikacija na temelju više klasa korištenjem *random forest* algoritma, može ispravno klasificirati nepoznati signal u $\sim 75\%$ slučajeva na temelju samo 20 uzastopnih očitanja. Nadalje, isti model može pronaći kojim dvjema klasama signal najvjerojatnije pripada s točnošću od 95%.

**Ključne riječi:**
Internet stvari, IoT, senzori, strojno učenje, *random forest*, LSTM, vremenski niz podataka, prepoznavanje uzorka signala, energetska učinkovitost

# Acknowledgments

***Prediction is very difficult, especially if it's about the future!***
**Niels Bohr**

Ako doktoriram, željela bih zahvaliti mentoru Mariu za sav uložen trud i podršku tijekom zadnjih 10 godina. Bez vaših savjeta te *brojnih* komentara i **iskrižanih verzija**, nebi ni bilo ove teze. Nadam se da ste u našoj suradnji uživali koliko i ja.

Hvala i mom drugom, iako neslužbenom mentoru - **šefu** Toniju. Hvala na savjetima i usmjeravanju tijekom ovih 5+ godina. Bez tebe ne bih ni imala prilike upisati doktorat. Hvala ti što si me učinio ~~menadžerom~~ istraživačem!

Hvala svim **bivšim** kolegama iz Ericsson Nikola Tesla istraživačkog laba. Zahvaljujem Ivani na svim brojevima u mom doktoratu, kao i na pravoj količini kofeina i papirnatih maramica. Bez nje bi ovaj doktorat imao sasvim drugačije rezultate. Hvala Dinku koji je pročitao ovu tezu više puta, iako je umirao od dosade. Bez tebe bih pisala *disetaciju koja se bavi stojnim ućenjem.* Zahvaljujem i Darku što je prepoznao moj potencijal i omogućio mi rad u istraživačkom odjelu, te upis doktorata.

Na kraju se zahvaljujem mojoj obitelji - hvala Tomislave, Magdalena, Ivane, Željka i Jure. Ana, iako si srednje dijete, nisam te zaboravila, naprotiv: Hvala mojoj sestri i supatnici koja me bodrila na cijelom putu. Žao mi je što ti je moja disertacija oštetila laptop. Bez tebe bi *ovdje elaborirani* doktorat imao još kompliciranije rečenice.

**Hvala što jeste!**

*P.s. odlično uloženih 5 godina i 8392,06 eura.*

# Contents

Abstract . . . . iv

Sažetak . . . . vi

Acknowledgments . . . . ix

List of Tables . . . . xiii

List of Figures . . . . xvi

**1 Introduction** **1**

1.1 IoT System Design . . . . 2

1.1.1 Sensors . . . . 3

1.1.2 Gateways . . . . 3

1.1.3 Clouds . . . . 4

1.2 Open Challenges/Research Goals . . . . 5

1.3 Hypothesis . . . . 7

1.4 Dissertation Outline . . . . 8

**2 Related Work** **11**

2.1 Data Reduction in IoT . . . . 11

2.1.1 Sensors . . . . 11

2.1.2 Gateways . . . . 13

2.1.3 Clouds . . . . 15

2.1.4 Energy Efficient Approaches for Data Velocity and Volume Reduction 16

2.2 Data Velocity and Volume Reduction Approaches the Research Will Focus On 17

2.2.1 Dynamic Monitoring Frequency on a Sensor . . . . 17

2.2.2 Predicting Final Value from Part of Transient for Sensors with Preheating 20

2.2.3 Signal Type Classification - Time Series Data Classification . . . . 22

2.3 Summary . . . . 24

**3 Overview of Machine Learning Techniques** **25**

3.1 Linear and Logistic Regression (LR) . . . . 26

3.1.1 Linear Regression . . . . 27

3.1.2 Logistic Regression . . . . 27

3.1.3 Comparison between Linear and Logistic Regression . . . . 28

3.2 Random Forest (RF) . . . . 28

3.3 Support Vector Machine (SVM) . . . . . . . . . . . . . . . . . . . . . . . . . 30
3.4 K-Nearest Neighbours (kNN) . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
3.5 Artificial Neural Networks (ANN) . . . . . . . . . . . . . . . . . . . . . . . . 35
3.5.1 Multilayer Perceptron (MLP) . . . . . . . . . . . . . . . . . . . . . . 35
3.5.2 Recurrent Neural Networks (RNN) . . . . . . . . . . . . . . . . . . 36
3.6 One Class Classification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39
3.6.1 One Class Support Vector Machine (OCSVM) . . . . . . . . . . . 40
3.6.2 Isolation Forest (IF) . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
3.7 Comparison and Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

**4 Dynamic Monitoring Frequency for Energy-Efficient Data Collection in IoT 45**
4.1 Problem Statement and Datasets . . . . . . . . . . . . . . . . . . . . . . . . 46
4.1.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.1.2 Baseline Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.1.3 Solution Space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
4.1.4 Datasets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
4.2 Statistical and Machine Learning Algorithms . . . . . . . . . . . . . . . . . 50
4.2.1 Statistical Algorithm (S-DMF) . . . . . . . . . . . . . . . . . . . . . 51
4.2.2 Machine Learning Algorithm (ML-DMF) . . . . . . . . . . . . . . . 52
4.3 Performance Comparison between Static and Dynamic Monitoring Frequencies 53
4.3.1 Statistical Algorithm (S-DMF) Performance . . . . . . . . . . . . . 54
4.3.2 Machine Learning Algorithm (ML-DMF) Performance . . . . . . . 56
4.3.3 Comparison of DMF vs SMF Algorithms . . . . . . . . . . . . . . . 59
4.4 Implications on Energy Efficiency and Processing Complexity . . . . . . . 61
4.4.1 Energy Efficiency . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.4.2 Optimizing the Size of the Training Dataset . . . . . . . . . . . . . 64
4.4.3 Processing Complexity . . . . . . . . . . . . . . . . . . . . . . . . . . 67
4.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67

**5 Predicting Final Value from Part of Transient for Sensors with Preheating 69**
5.1 Gas Sensor Characterization . . . . . . . . . . . . . . . . . . . . . . . . . . 70
5.1.1 Environment Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
5.1.2 Online Period Characteristics . . . . . . . . . . . . . . . . . . . . . 72
5.1.3 Sleep Period Characteristics . . . . . . . . . . . . . . . . . . . . . . 73
5.1.4 Collection of Transients . . . . . . . . . . . . . . . . . . . . . . . . . 75
5.2 LSTM Model Training and Validation . . . . . . . . . . . . . . . . . . . . . 76
5.2.1 LSTM Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
5.2.2 Dataset Stratification and Diversification . . . . . . . . . . . . . . . 76
5.2.3 LSTM Parameters Selection . . . . . . . . . . . . . . . . . . . . . . . 78
5.2.4 Frequency Selection for Input Data . . . . . . . . . . . . . . . . . . . 78

5.2.5 Performance Testing with Scarce Data . . . . . . . . . . . . . . . . 80
5.3 Evaluation and Discussion . . . . . . . . . . . . . . . . . . . . . . . . . 82
5.3.1 MQ-5 Sensor Results . . . . . . . . . . . . . . . . . . . . . . . 82
5.3.2 MQ-6 Sensor Results . . . . . . . . . . . . . . . . . . . . . . . 82
5.3.3 Discussion on Energy Efficiency . . . . . . . . . . . . . . . . . . 84
5.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 87

**6 Signal Type Classification - Time Series Data Classification 89**
6.1 Dataset and Methodology . . . . . . . . . . . . . . . . . . . . . . . . . 90
6.1.1 Dataset . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90
6.1.2 Methodology . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91
6.2 Comparing Classification Algorithms . . . . . . . . . . . . . . . . . . . 93
6.3 Algorithms Robustness . . . . . . . . . . . . . . . . . . . . . . . . . . . 95
6.3.1 Window Size Analysis . . . . . . . . . . . . . . . . . . . . . . . 95
6.3.2 Streaming Data Analysis . . . . . . . . . . . . . . . . . . . . . . 96
6.3.3 Stored Data Analysis . . . . . . . . . . . . . . . . . . . . . . . . 99
6.4 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100
6.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102

**7 Dissertation contribution 105**

**8 Conclusion 107**

**BIBLIOGRAPHY 109**

# List of Tables

2.1 Data reduction in IoT - Related work overview . . . . . 18

4.1 Abbreviations and Notations . . . . . 48

4.2 ML hyper-parameters for 2 features and 4 features (*max_features is used only for 4 features RF) . . . . . 53

4.3 Results of S-DMF algorithm . . . . . 56

4.4 Results of ML-DMF algorithms . . . . . 60

5.1 Gas concentration classes . . . . . 76

5.2 Parameter combinations for building LSTM . . . . . 77

5.3 Sets of datasets with included/excluded classes . . . . . 78

5.4 Selecting optimal parameters for LSTM NN . . . . . 79

5.5 MQ-2 RMSE for different transient sizes . . . . . 81

5.6 RMSE for different classes (transient size 186) . . . . . 81

5.7 RMSE for different classes (transient size 20) . . . . . 82

5.8 MQ-5 RMSE . . . . . 83

5.9 MQ-6 RMSE . . . . . 83

5.10 Energy consumption . . . . . 84

6.1 Dataset overview . . . . . 90

# List of Figures

1.1 *IoT system design* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2.1 *Locations in IoT our research is focused on* . . . . . . . . . . . . . . . . . 18

3.1 *Illustration of Linear and Logistic regression [1]* . . . . . . . . . . . . . . . 26

3.2 *Illustration of random forest [2]* . . . . . . . . . . . . . . . . . . . . . . . 29

3.3 *Illustration of Support vector machine [1]* . . . . . . . . . . . . . . . . . . 31

3.4 *Illustration of an example of the k-nearest neighbor algorithm [3]* . . . . . 32

3.5 *Multilayer Perceptron [4]* . . . . . . . . . . . . . . . . . . . . . . . . . . 35

3.6 *A recurrent neuron [4]* . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

3.7 *Long short-term memory (LSTM) cell [4]* . . . . . . . . . . . . . . . . . . 38

3.8 *One class classification vs multi-class classification/detection [5]* . . . . . . 39

3.9 *One class support vector machine [6]* . . . . . . . . . . . . . . . . . . . . 40

3.10 *Isolation forest [7]* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

4.1 *Problem Description* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47

4.2 *Minimum period for maximum delta* . . . . . . . . . . . . . . . . . . . . . 49

4.3 *One day data collection with the ideal reading algorithm* . . . . . . . . . . 50

4.4 *Dataset* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51

4.5 *Distribution of differences between neighbouring values for target_accuracy =* 99%, $\Delta_{max} = 1°C$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52

4.6 *Results of S-DMF algorithm compared to SMF for temperature dataset with* $\Delta_{max} = 0.5°C$*; Inner image depicts results on logarithmic scale* . . . . . . . 55

4.7 *Results of S-DMF algorithm for temperature dataset* . . . . . . . . . . . . 55

4.8 *Selecting the best parameters of ML algorithms (*$\Delta_{max} = 0.5°C$*)* . . . . . . . 57

4.9 *Comparison of results of ML algorithms with 2 and 4 features (*$\Delta_{max} = 0.5°C$*)* 59

4.10 *Overall results* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

4.11 *Comparison of S-DMF and ML-DMF results with SMF* . . . . . . . . . . . 63

4.12 *3 hours S-DMF and ML DMFs with* $\Delta_{max} = 0.5°C$*; target_accuracy* $= 99\%$*;* $n_{cc} = 5$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

4.13 *1 day S-DMF and ML DMFs with* $\Delta_{max} = 0.5°C$*; target_accuracy* $= 99\%$*;* $n_{cc} = 5$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

4.14 *1 week S-DMF and ML DMFs with* $\Delta_{max} = 0.5°C$*; target_accuracy* $= 99\%$*;* $n_{cc} = 5$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65

4.15 *1 month S-DMF and ML DMFs with* $\Delta_{max} = 0.5°C$*; target_accuracy* $= 99\%$*;* $n_{cc} = 5$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

4.16 *6 months S-DMF and ML DMFs with* $\Delta_{max} = 0.5°C$*; target_accuracy* $=$ $99\%$*;* $n_{cc} = 5$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

4.17 *1 year S-DMF and ML DMFs with* $\Delta_{max} = 0.5°C$*; target_accuracy* $= 99\%$*;* $n_{cc} = 5$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66

5.1 *Three scenarios for sensors setup, namely a) always online, b) with full preheat and sleep period and c) with prediction based on a partial preheat and sleep period* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70

5.2 *Setup scheme with the test and control sensors, and real-time clock for turning on and off the test sensor* . . . . . . . . . . . . . . . . . . . . . . . . . . 71

5.3 *Transient for different online (O) – sleep (S) ratios (without gas)* . . . . . . 72

5.4 *Correlation of transients in case of different Online/Sleep ratios* . . . . . . 73

5.5 *Transients in different gas levels* . . . . . . . . . . . . . . . . . . . . . . . 74

5.6 *Low vs High gas level transients* . . . . . . . . . . . . . . . . . . . . . . . 74

5.7 *Actual values for 381 transients (top figure) grouped in 8 classes (bottom figure)* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75

5.8 *Comparison of predicted and expected values for MQ-2 gas sensor* . . . . . 80

5.9 *Comparison of predicted and expected values for MQ-5 gas sensor* . . . . . 83

5.10 *Comparison of predicted and expected values for MQ-6 gas sensor* . . . . . 83

5.11 *Energy consumption compared to the sensor that is always online (*Transmission energy is insignificant compared to others, thus not visible in this figure)* . . 85

6.1 *Splitting datasets for training, validation and evaluation* . . . . . . . . . . 91

6.2 *Illustration of three approaches* . . . . . . . . . . . . . . . . . . . . . . . . 92

*6.3 Models accuracy* . . . . . . . . . . . . . . . . . . . . . . . . . . . 93

*6.4 One Class results* . . . . . . . . . . . . . . . . . . . . . . . . . . . 94

*6.5 Two class results* . . . . . . . . . . . . . . . . . . . . . . . . . . . 94

*6.6 Multi Class results* . . . . . . . . . . . . . . . . . . . . . . . . . . 94

*6.7 Chunk selection* . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96

*6.8 Models accuracy for various window sizes* . . . . . . . . . . . . . . . . . 97

*6.9 IF varying window size* . . . . . . . . . . . . . . . . . . . . . . . . . 97

*6.10 mc RF varying window size* . . . . . . . . . . . . . . . . . . . . . . . 97

*6.11 Streaming data mode - combined mean accuracy per number of chunks (window size = 20)* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98

*6.12 Accuracy of mc RF streaming data* . . . . . . . . . . . . . . . . . . . . 98

*6.13 Stored data mode - combined mean accuracy per number of chunks (window size = 20)* . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 99

*6.14 Accuracy of mc RF stored data* . . . . . . . . . . . . . . . . . . . . . 100

# 1 Introduction

Ericsson in its whitepapers [8] and [9] introduces cellular Internet of Things (IoT) network. Cellular IoT enables integration of 5G into the existing and evolving industrial networks used for real-time automation, thus enabling the Industry 4.0 revolution to meet its promise for full digitalization. Consequently, IoT concept is growing and its topics are highly researched. Ericsson forecasts that there will be up to 4.1 billion devices connected by Massive IoT and other emerging cellular technologies by 2025 [10].

A basic IoT system consists of "the things", i.e., sensors collecting data, gateways as midpoint devices connecting sensors to the cloud, and clouds where data is stored, streamed, analysed and presented. Most of the IoT systems have numerous sensors distributed within some area. Every sensor has its own battery with limited lifetime and that represents one of the main bottlenecks of described technology [11,12]. Currently, IoT sensor tasks have to be carefully planned with large energy savings requirements because they have small batteries that cannot handle large energy consumption demands. Furthermore, a great number of sensors collect data in short time intervals, which results in a substantial amount of data.

Since sensors collect data in short time intervals over long time periods, i.e., months or even years [13], it is expected that IoT devices will generate up to 79.4 ZB of data by 2025 [14]. Finally, all the collected data is sent towards cloud where it must be processed and stored in large data centers. Furthermore, it is expected that 175 ZB of data will be stored by 2025 [15]. The increasing amount of stored data requires an increase in the number and size of data centers, which are connected on electrical grid and consume electrical energy.

However, more data is not always better than less data. Often, collecting redundant data simply takes storage capacity without providing new information [16] since large amount of redundant data could be irrelevant and excessive. Collection of such data consequently costs money and creates unnecessary overheads, thus it should be omitted [17]. In order to identify redundant data early in the process, pre-processing on the edge of an IoT network, namely on sensors and gateways is required. Eliminating "garbage" data before it is moved to the cloud helps to keep data centers less loaded and future data analysis more efficient [18].

As the volume of data generated by various IoT devices increases, storing and processing it becomes significantly challenging. This is already recognised as the Big data concept, commonly described with **five Vs** [19], namely **Volume**, **Velocity**, **Variety**, **Veracity** and **Value**. **Volume** represents the amount of data, while **Velocity** is the speed of gathering new data. **Variety** stands for different types of data. **Veracity** is the measure of data uncertainty, while **Value** is the information obtained from the collected data.

Today, IoT requirements increase volume and velocity of the data, as well as variety. Finding the optimal balance between the volume reduction and the information loss (value) requires the utilization of data variety [20] and thus reducing data velocity at its source [21]. The expected grow in the number of IoT data sources gives rise to network-edge computing; data pre-processing, filtering [22], compression [23], aggregation and/or correlation [24] close to the data sources. This includes novel concepts such as edge-mining, which stands for data processing on battery-powered devices placed at the edges of an IoT network [25]. Such a solution would achieve reduction of data volume at the network edge and thus reduce energy consumption, bandwidth, as well as storage capacity and processing power at the cloud backend systems.

However, before performing any of the above mentioned data reduction methods, data has to be collected. Therefore, battery-powered IoT sensors are still constantly turned on, i.e., still collecting data and consuming energy. Consequently, moving data reduction closer to the edges of an IoT network reduces the amount of data circulating through the network. However, such approach still does not reduce energy consumption of sensors. Therefore, although the amount of data is decreased, sensors are still constantly turned on and consume energy.

**The main goal of herein elaborated research is to detect places in IoT network where data velocity and volume reductions can be made, while taking into a consideration data variety and preserving its value. While doing so, it is important to understand compromises, either to accept them or to mitigate them.**

## 1.1 IoT System Design

IoT systems consist of three main components; *sensors* that collect data from the environment and send it through *gateway* towards *cloud*, where data is analysed and used for decision making. Each component receives the *input data*, performs internal data processing, and forwards *output data* (Figure 1.1).

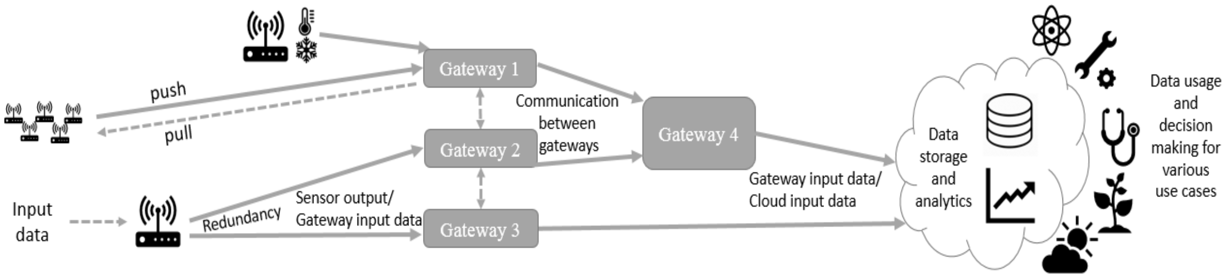

push
pull
Input data
Redundancy
Sensor output/ Gateway input data
Gateway 1
Gateway 2
Gateway 3
Communication between gateways
Gateway 4
Gateway input data/ Cloud input data
Data storage and analytics
Data usage and decision making for various use cases

*Figure 1.1. IoT system design*

*Input* and *output data* can be received/forwarded using either *push* or *pull* mechanisms, or as their combination. The initiator for *push* mechanism is the southbound component, e.g., a sensor pushes the data towards a gateway at any moment, which requires the gateway to asynchronously wait for data transfer. For *pull* mechanism the initiator is the northbound component, e.g., a gateway requests data from sensor, which requires the sensor to be always on and waiting for data transfer.

### 1.1.1 Sensors

Sensors are endpoint devices that read physical measurements from the environment, e.g., temperature or humidity sensors, as well as more complex devices such as LiDAR sensors[1] or even cameras. Being endpoint devices, sensors are commonly battery powered and have low data processing power, while being numerous due to growth of IoT and requirement for more data. Today, sensors are usually consolidated to collect more than one measurement type due to drastically reduced production costs, so data collected by single sensor can contain heterogenous values. Data collected by sensors is referred to as *input data*, while *output data* is data processed by sensors and forwarded further throughout the IoT network.

- *input data* – a measurement collected by the sensor. For basic analog sensors, such as temperature sensors, it is commonly a voltage value acquired through a measurement circuit, which must be translated into a temperature value. For cameras this can be a raw image acquired through an optical sensor. For physical measurements input is commonly read with pull mechanism.

- *output data* – once the data is processed it is transmitted towards northbound component. Since sensors commonly run on batteries, they usually push data towards a gateway and then enter a sleep mode, rather than waiting for the pull request all the time. Data format depends on the input data. However, communication channel used for data transfer in IoT sensor networks is commonly some type of radio channel due to remote locations of the sensors, e.g., LoRa[2], ZigBee[3] or BLE[4].

### 1.1.2 Gateways

Midpoint devices placed between sensors and a data analytics backend in the cloud are referred to as gateways [26]. A gateway allows sensors to communicate over shorter distances and abstracts the underlying communication layer such as LoRa [27] on its southbound, while on its northbound it forwards the data towards the cloud. Furthermore,

[1] Light Detection and Ranging – a remote sensing method

[2] Long range, low power wireless platform

[3] IEEE 802.15.4-based specification for a suite of high-level communication protocols

[4] Bluetooth Low Energy

before forwarding the data it can perform additional data processing for purpose of optimization or on-site analytics and decision making. This is commonly feasible as gateways are implemented with more powerful hardware devices such as Raspberry Pi[5], Arduino[6] and STM32[7] or even a smart phone [28].

Similar to sensors, there can be more than one gateway within an IoT sensor network as shown in Figure 1.1. Reasons can be multifold, namely redundancy (where multiple gateways receive data from the same sensors), scalability (when a single gateway cannot endure the load from all sensors in the area), hierarchy (where gateways are sequentially connected due to a layered architecture or even distance prolongation), etc. Data sent by sensors and received by gateways are gateway *input data*, while gateway *output data* is data processed at the gateway which is forwarded towards the cloud.

- *input data* - represents data coming from sensors. The data is received over a communication channel that sensors support, such as LoRa or even some proprietary solution. Consequently, the data format can also be arbitrary, which requires a gateway to speak the same language as the sensors. While the data is commonly pushed towards the gateway as previously explained, gateways can still support some type of push back communications towards the sensors as they are required to manage the sensor network, e.g., sending a configuration to sensors, load balancing different communication channels, etc.

- *output data* – data received from sensors is processed by the gateway and then forwarded to the cloud for further analysis and decision making. With the rise of Fog and Edge, more functionalities are pushed towards gateways [11, 19]. Due to their low computational power and storage capacity compared to the cloud, the data is still pushed to cloud. This is commonly done over the Internet, while communication channels can vary from 3G/4G/5G, to WiFi or even cable connection if the gateway is located in (sub)urban area.

### 1.1.3 Clouds

Data collected from the sensors and processed by the gateways ends up in the cloud. Since cloud is backed by huge data centers hosting thousands of servers its computational and storage capacity makes it feasible for data analytics and archiving. However, due to its distance from the endpoint sensors, real-time systems may suffer due to high latency. Nevertheless, with introduction of 5G and high speed Internet backbone, this latency decreases [29]. Furthermore, utilization of machine learning and prediction algorithms is able to fill the gap between real-time systems and distant clouds [30]. Finally, distribution

[5] Credit-card-sized computer that plugs into a TV and a keyboard

[6] Open-source electronics platform based on easy-to-use hardware and software

[7] A series of 32-bit microcontroller integrated circuits

of cloud towards Fog and Edge reduces this lag even more [31]. Cloud *input data* is data sent from the gateways towards the cloud, while cloud *output data* is stored, visualized and analysed data that can be used for decision making.

- *input data* - while data transmitted from the sensors to the gateways is still arbitrarily formatted, IoT cloud platforms are beginning to gain a standard interface with the advent of protocols such as MQTT [32]. It is being used in gateways as well. That said, input data represents data that is received by the cloud services, commonly in a push manner, i.e., gateways push the data towards the cloud.
- *output data* – once received by the cloud services the data is then processed, visualized and stored. Consequently, data format on the output depends on its purposes, as well as the use case. Due to high velocity of data, more processing is being performed using streaming analytics, while storing only relevant data. Finally, even visualization is done only after preprocessing the data due to its high volume.

## 1.2 Open Challenges/Research Goals

Sensors usually collect data over an evenly spaced time period defined by the experts in accordance with the requirements of a specific domain. Selecting such a fixed period is a challenging task because the expert has to find a compromise between the amount of collected data and its accuracy. A domain expert has to select a period that is optimal with respect to both; length to eliminate irrelevant and redundant data and brevity to detect all significant changes in observed signal. As a result, a fixed period can be too short on one part of the signal while being too long on the other part of the same signal due to changes in the signal over time [33].

When an expert is tasked with selecting the optimal period after each data collection based on the most recent readings, previous readings, and the current state of the system, data collection with a dynamic period takes place. The selection of such a time period is even more challenging. After every reading, the next period is selected. This new period should be selected so that it can simultaneously identify all significant changes and remove any redundant data. The latter challenge will also be tackled within the research.

Therefore, in order to reduce sensors online time, they can be put to sleep for a longer period of time and turned on for a very short period just to collect data. The challenge with described approach are sensors which require preheating before being able to collect data from the environment and cannot collect data at the exact moment they are turned on.

Inspired by Jia et al. research on gas sensors [34], the focus of second part of herein elaborated thesis will be on sensors that require preheating. During preheat time, voltages and currents take time to stabilize so that readings of the actual value can be obtained. A momentary variation in the current or the voltage during preheating transition is called a transient. Therefore, if actual value can be predicted from the first part of the transient rather than having to wait for the sensors to fully preheat, only the first part of the transient needs to be collected in order to save energy.

Once collected, all the data has to be stored. Data is usually stored in data centers in a predefined and consistent format, as inconsistent storage may potentially lead to the Digital Dark Age [35], i.e., data from the digital age can be lost irrevocably [36]. Digital preservation attempts to mitigate such risks. However, there are a number of challenges in the preservation and archiving of digital materials, i.e., type of storage, number of copies, data and metadata models, etc. [37]. There are three main challenges in this area that are recognised by the experts:

(1) Growth of IoT network requires adding more sensors, either to replace broken ones or as an addition to an existing solution. Discovering and configuring sensors and associated data streams usually requires a lot of manual work that needs to be performed by technicians, thus becoming labour-intensive [38]. Therefore, numerous researchers [38–40] propose their methods and frameworks that can be used to automate the configuration process by automatically mapping sensor metadata with data streams.

(2) IoT network with large number of battery powered sensors requires IoT tasks to be carefully planned due to a limited lifetime of sensor batteries. Therefore, data collection is commonly optimized [24]. On one hand, in cases with predefined structures, data optimization can be done by excluding timestamps, information on sensor types and variable names which are later recovered in the cloud where the configuration metadata is stored. On the other hand, in cases where the connection towards a configuration metadata is broken or metadata is corrupted due to unexpected problems, sensors will continue to send data, however the knowledge of their data type is lost. Such failures may be very expensive as data being sent by the sensors is meaningless, while the entire system is still up and running and consuming resources and energy [41].

(3) Even with the above mentioned optimizations, IoT with time still generates zettabytes of data resulting in Big data. However, traditional data storage systems are not suitable for Big data storage [42, 43]. Commonly, Big data is stored in Data Lakes that aim to store metadata separately from raw data [44], thus assisting users and applications to find data more efficiently [45]. Hence, in a case of unexpected problems between distributed storage systems, previous knowledge about collected data, including configuration information and metadata as well as raw data can be lost. Therefore, all data generated by sensors looses their context and may have to be reconfigured from scratch.

All three mentioned challenges require metadata discovery/recovery and hence, can be tackled with a single solution i.e., signal type recognition. It defines that a signal from a specific sensor type in an IoT network could be recognised and its type could be determined from a small signal chunk to enable quick signal classification.

## 1.3 Hypothesis

The main motivation behind this research is optimization; of collected data volume, transmitted data volume, energy consumption and time consumption when collecting data in an IoT system. Therefore, the main hypothesis (**H0**) follows from the assumption that **using machine learning techniques it is possible to significantly optimize volume of data circulating through IoT network, and consequently energy consumption, as well as to reconstruct time series data lost due to such optimizations, in case of network or storage malfunctions.**

Three additional hypotheses are derived from the main hypothesis:

- (**H1**) **It is possible to predict the change in value of a monitored phenomenon based on knowledge of historical data, without constant environmental data collection.**
  In more detail, the dynamic monitoring frequency algorithm can estimate the future time moment when an expected change (delta) will happen, without introducing a significant error. Thus, sensors can only collect data when the change is expected and sleep in the meanwhile.

- (**H2**) **It is possible to estimate an actual value based on a short part of a transient, for sensor with a long preheat time.**
  Long short-term memory (LSTM) neural network can be used to learn and later predict the actual value from a part of the transient. This way, instead of being constantly online or fully preheating, the sensor needs to be turned on for only a short period of time and sleep afterwards. This strategy can result in significant energy savings.

- (**H3**) **It is possible to differentiate between readings from different sensors in real time, on small chunks of time series data.**
  Therefore, we hypothesize that random forest algorithm can be used to model time series data for signal type classification in order to solve 3 problems recognised by the experts: (1) recognition of a streaming signal type in the case of configuration data loss due to unexpected communication issues; (2) classification of signal types in the case of unexpected data storage issues and thus loss of configuration data and/or metadata and (3) automatic recognition of a signal type when a new sensor is added to an IoT network.

## 1.4 Dissertation Outline

The main goal of this research is the investigation of new optimization methods that can reduce the volume and velocity of data circulating through the IoT network with a large number of sensors and, consequently, energy consumption. Such improvements frequently result in trade-offs. These compromises will be taken into account and the ways in which they can be mitigated will also be examined.

In order to do so, predictive analytics will be used, i.e., the prediction of future events based on data collected in the past. In more detail, a data-driven predictive approach will be applied. On the one hand, in a model-driven approach, a theoretical mathematical model is constructed based on domain expertise or prior knowledge. On the other hand, a data-driven approach relies on patterns and relationships observed in data [46]. Thus, instead of starting with a theoretical model, data-driven approaches use statistical and machine learning techniques to find patterns and relationships in data, and generate predictions or make decisions. Therefore, a model-driven approach may be more appropriate if the underlying processes are well understood and there is enough domain expertise to construct an accurate model. In other cases, a data-driven approach may be more suitable if the underlying processes are complex or poorly understood, and the available data is rich enough to support an accurate analysis. In the research, focus will be on machine learning (ML) algorithms [4, 47–49].

The research is split into 3 parts, where the first one (Chapter 4) covers investigation of an adaptive, dynamic sampling period, the second one (Chapter 5) covers energy optimization and deep sleep application of the sensors with pre-heat period, while the third part (Chapter 6) covers signal type classification and, thus, enabling:

1. distinguishing signals from different sources if the configuration is lost
2. correct classification of signals after malfunctions between distributed storage
3. automatic recognition of a newly added sensor.

In the Chapter 4, a novel dynamic monitoring frequency (DMF) algorithm is developed and tested. DMF algorithm aims at collecting data only when sensor readings change by more than a predefined value (delta) between consecutive readings. Datasets used in this research consist of the meteorological data collected every minute during almost 3 years. DMF algorithms are prepared for multiple deltas on a temperature dataset and are later verified on humidity and air pressure datasets.

Two data-driven approaches are used for DMF algorithm implementation, namely statistical and machine learning. On one hand, the statistical algorithm is implemented to calculate probability of the acceptable change taking place based on the distribution of historical data. On the other hand, the ML algorithm compares different ML classification algorithms that predict future period when the acceptable change is expected, based on previous readings. Finally, the accuracy of DMF algorithm is compared to the results of two baseline algorithms with static monitoring frequency (SMF) i.e., with a fixed reading period. The first SMF baseline algorithm should collect the same amount of data as the DMF algorithm, while the second SMF baseline algorithm should have the same number of errors as DMF.

Inspired by Jia et al. research [34], in the Chapter 5, we will focus on sensors that require preheating such as gas sensors, which cannot collect data at the exact moment when they are turned on. During this preheat time a voltage needs to stabilize, where a set of unstable values during this period are referred to as a transient. Our research aims to explore a new approach for obtaining sensor data. This approach involves reading multiple values from the initial phase of the transient, rather than periodically reading data while continuously being online or fully preheating after every sleep cycle. To estimate the actual value, we plan to use a long short-term memory (LSTM) recurrent neural network, which will be trained on the collected data.

For defining methodology, low-cost MQ-2 [50] gas sensors are utilized. Since the goal is to predict the actual value from the transient and turn the sensor off, two sensors are used, namely, a test sensor with a predefined sleep period for predicting the actual values and a control sensor for the ground truth, without a sleep period. Finally, to reduce external influences, the entire setup is placed in an isolated environment, i.e., a sealed isolated container. Prior to building the prediction model, a preliminary analysis is performed to characterize the behaviour of the transient. Transients for the different online–sleep ratios are compared, as well as different gas concentrations. Finally, the approach is tested on the MQ-5 [51] and MQ-6 [52] gas sensors to verify its applicability on different low-cost sensors.

Chapter 6 is focused on signal type classification that enables reconstruction of time series data type in case of unexpected network or storage issues, as well as automatic sensor configuration. Therefore, the goal is to compare and evaluate applicability of different approaches and corresponding ML models for time series signal type recognition from IoT sensors using signal pattern classification. The dataset used in this research contains 11 different signal types from 2 separate data sources - meteorological data and computer resource utilization data.

Furthermore, in order to select the best model, three approaches are considered and analysed – one class [53], two class and multi class classification [54]. One class and two class approaches require $n$ models, one for each of the $n$ signal classes, while multi class classification should have the knowledge of all $n$ classes and thus results with a single ML model which outputs $n$ predicted probabilities. Moreover, streaming data and stored data modes are considered. First, a streaming mode which defines continuously arriving data in real time will be evaluated on consecutive windows. Second, stored data mode will be used when data is already collected and stored, however metadata is lost, and stored data types are unknown. Therefore, windows can be selected randomly from the entire set of stored data.

# 2 Related Work

This chapter systematically covers existing solutions for all three researched areas defined in a previous chapter. The first part overviews existing data reduction techniques that can be applied across all parts of the IoT network. In the second part, the focus is on the specific techniques that this research will investigate. This includes approaches that eliminate data before it is collected (such as the dynamic monitoring frequency approach), the use of LSTM neural networks with a focus on gas sensor applications, and an overview of the existing techniques for time series data classification.

## 2.1 Data Reduction in IoT

As shown in Figure 1.1, there are three main components of IoT systems; sensors that collect data from the environment and send it through gateway towards cloud, where data is processed, analysed and stored. Data reductions can be made on each component, either when collecting the input data or when performing internal data processing and outputting the reduced data. Therefore, researchers are trying to reduce the amount of collected data and energy consumption in all parts of an IoT network, either in an infrastructure or in monitored data [24].

### 2.1.1 Sensors

While collecting data from the environment, sensors are dealing with a large amount of data in short time intervals. Consequently, sensors consume a lot of battery power to collect data, even though many of the collected values are similar. Sensors should implement algorithm that dynamically adapts reading interval based on previously collected values as well as filter collected data, classify them and self-control data transmission according to their trends.

**Input data**

In [55] Mastelic et al. introduce a dynamic monitoring frequency algorithm for collecting monitoring data from ultrascale systems. The algorithm deterministically reduces data velocity by dynamically self-adapting the monitoring frequency to the volatility of collected data. It also reduces data volume because it reads data less often but still keeps the same data value as the equivalent static monitoring frequency. The algorithm requires human (administrator) input of maximum and minimum periods as well as factor for calculating window size.

Trihinas et al. [56] introduce AdaM, a lightweight adaptive monitoring framework for battery-powered IoT devices with limitations in processing capabilities. Framework consists of two algorithms, one for adaptive sampling and other for adaptive filtering. In this part of the system, adaptive sampling, where algorithm dynamically adapts monitoring intensity based on the current evolution and variability of the metric stream with minor processor usage, is used. Results show that it reduces volume and energy consumption while preserving relatively high accuracy.

Jia et al. in [34] propose a new, low-power, automatic, accurate, and wireless ammonia monitoring approach that uses metal oxide sensors. This approach does not wait for equilibrium as this consumes significant amount of energy, rather it tries to predict the resistance at equilibrium using the sensor's transient measurements in the short heating window using LSTM neural networks. Proposed model accurately predicts the equilibrium state resistance value with an average error rate of 0.12%.

Arendt et al. [57] present a model-predictive communication framework based on historical data analysis. Data collected in three previous days is used to predict fourth day values. Two algorithms, autoregression based ARIMA and a neural network with LSTM cells, are compared, in order to reduce communication effort. Results show a significant potential to reduce communication effort per day of at least 60% up to 95%, depending on the required accuracy, significantly contributing to the achievement of mMTC performance targets. The main bottleneck of this solution is its usage of historical data. The framework is used to predict one day after data is constantly collected for 3 days, thus it can either predict every 4th day, or use predicted values of 4th day to predict values of the 5th day. Latter option will increase error with time since only predicted data is used.

In [58], Cecchinel et al. have investigated whether there is significant battery lifetime gain when using adaptive sampling and sending periods. Self-adaptive approach using machine learning and deep-sleep is used to provide an optimal configuration extending the battery lifetime of a sensor platform. Prediction model based on historical data is implemented on a middle layer, and it is used for the generation of the optimal configuration. Gain in the battery lifetime is compared to solution that uses fixed periods (with and without deep-sleep). It is shown that adaptive periods successfully lower the battery discharge. In experimental validation, authors have successfully scaled up the battery lifetime of a temperature sensor from a monthly to a yearly basis, while ensuring a proper level of data quality and freshness.

### Output data

The second part AdaM algorithm [56], adaptive filtering, is useful for preparing sensor output. After sampling algorithm selects data to read, filtering algorithm analyses the dataset and decides, for every input, is it relevant and should it be passed to the next stage.

In [59] Zordan et al. present a lossy compression scenario which uses machine learning techniques for signal classification to improve energy efficiency while tolerating some distortion. The algorithm first collects time series data and extracts features. Next step is feature normalization and selection of relevant ones. Last step is signal classification. Experiments show that small number of features (< 20) achieve classification performance over 97%.

In [60] Baharudin et al. propose low-energy algorithm for sensor data transmission from sensor nodes where the sensors can self-control data transmission according to the trends of data. It uses Adaptive Duty Cycle for data transmission adjustment frequency and Compressive Sensing (CS) for data compression. Simulation results show that collective transmission with CS-based data compression significantly reduces transmission energy.

In [58], Cecchinel et al. have investigated whether there is significant battery lifetime gain when using adaptive sampling and sending periods. While adaptive sampling is optimization in input sensor data, adaptive sending periods optimize output of a sensor. As already written in previous section, experimental validation shows that this combination of adaptive parameters extends battery lifetime of a temperature sensor from a monthly to a yearly basis, while ensuring a proper level of data quality and freshness.

## 2.1.2 Gateways

After data is collected and processed on sensors, it is forwarded to the gateways. There are two approaches how data from sensors arrives at the gateway. In the first approach sensor decide when to push data to gateway, while in the other gateway send pull request to one of sensors in the group, or to all of them, depending on their position, battery level or Internet connection. When data is received, gateway should filter, compress and correlate data in order to forward only relevant values towards the cloud.

### Input data

Mushunuri et al. in [11] incorporate optimization libraries within the Robot Operating System deployed on a robotic sensor – actuators. To overcome the limitations of IoT devices in terms of computation power, Fog computing is used. One robot acts as a master node with others as client nodes. Whenever there is data available for computation, the client robot shares its computation load in order to extend its battery lifetime. The Edge client robot having data for computation sends its entire data to Fog server robot that divides computation among peer robots depending on their communication path loss and power availability.

Samie et al. in [12] propose a technique for managing computation offloading in a local IoT network under bandwidth constraints. The gateway gives each device its minimum bandwidth demand. Then it calculates the battery life of each device under this configuration and remaining bandwidth. The remaining bandwidth must be allocated to the devices, while prioritizing the devices with lower battery life. Results show more than 40% improvement in utilization of gateway's bandwidth as well as up to 1.5 hour improvement in battery life.

Liyanage et al. [61] proposes a proactive IoT gateway service scheduling scheme in the opportunistic Internet sharing environment. The scheme is to be used as a background service of a device to continuously retrieve the information of available gateways and to schedule the connection among the available gateways. The scheduling scheme reduces the re-connection between the collaborative devices and minimises the unnecessary energy consumption derived from re-connection processes.

In [62] Natarajan et al. propose an end-to-end system prototype for power-efficiently compression of continuous bio-signals in sensor network. The gateway dynamically tunes compression parameters to adapt to signal changes during continuous monitoring and transmits them to the sensor. Gateway dynamically adapts and tunes compression parameters based on current sparsity levels. Real-time parameter tuning is achieved by periodical transmission of raw measurements from the node to the gateway. This results with 100X faster solution with energy consumption less than 3% of standard encoder.

In [63] Wang et al. propose sleep scheduling and wake up protocol. In this solution gateway calculates sleep interval for sensor in order to save their battery and achieve energy efficiency.

Khandel et al. in [64] aim to reduce the communication overhead and propose a method that is able to determine which sensors should send their data to the central node and which one should drop data. Some sensors have high data correlation or they may have data that are not essential at the central node for current operation. The authors have designed a controller that selects a subset of sensors to send data to the cloud, while keeping the accuracy at an acceptable level. Considering the dynamics of the data collected at the sensors, Advantage Actor-Critic based reinforcement learning (RL) scheme is implemented to train the controller. The authors have also demonstrated how the parameters of the RL reward function can be tuned to make an appropriate tradeoff between the accuracy and communication overhead.

### Output data

In [21] NECtar agent, a solution for IoT network-edge data reduction is presented by Papageorgiou et al. This solution presents three main elements; streamification, one-click data handler instantiation and IoT-specific analysis. Streamification includes new data reduction algorithms that work upon data streams and take decisions per incoming data item. One-click data handler instantiation enables network-edge devices to select one of many possible algorithms and instantiate a handler that enforces the respective data reduction logic. Finally, IoT-specific analysis stands for one of the first network-edge data reduction algorithms evaluated specifically for IoT devices and datasets. NECtar achieves accuracies 76.1% to 93.8% while forwarding 1/3 of data items, without adding significant forwarding delays.

Razafimandimby et al. [65] present Bayesian Inference Approach. Sensors are collecting multiple values, including temperature and humidity, every 30 seconds. Gateways compute the probability of making an inference error in the cloud given the temperature and the humidity, before sending their data. If there is a strong chance that the error magnitude exceeds a predefined threshold, the gateway sends both humidity and temperature, else the gateway sends only the temperature data, and the humidity value will be inferred in the cloud. Evaluation results show that this approach drastically reduces the number of transmitted data and the energy consumption, while maintaining an acceptable level of data prediction accuracy.

### 2.1.3 Clouds

After the data collected by sensors reaches the cloud, it should be stored somewhere. Storing all of the collected data results in the large amount of similar data with great need for storage capacity. Accordingly, data should be filtered, compressed and normalized in a smart way, so only really relevant data is stored.

#### Input data

In [66] Soultanopoulos et al. describe the data collection from Bluetooth Low Energy devices to identify and track data in real time and allow processing in the cloud to improve analysis. One of the main characteristics of this solution is smart mode of data transmission including the time interval, on demand-violation and rules. A gateway enables the processing of sensor data before they are transferred to the cloud. The experimental results demonstrate that the system supports real-time communication and fast data collection with data transmission in less than 130 ms.

In [67] Ge et al. propose energy efficient solutions such as putting idle servers in data center (DC) to sleep as well as shutting down idle DCs during off-peak hours.

#### Output data

Meng and Liu [68] introduce the concept of monitoring-as-a-service (MaaS), its main components and requirements. There are three enhanced MaaS capabilities: window-based violation detection which can save 50% to 90% communication cost because of its parameter tuning ability; violation-likelihood based state monitoring which can adjust monitoring intensity based on the probability to detect important events which results with significant gain in monitoring service consolidation; and multitenant state monitoring techniques. Monitoring topology planning technique minimises monitoring data delivery overhead that leads to improvements in the scalability of the service.

Moon et al. in [20] compare lossy compression algorithms Discrete Cosine Transform (DCT), Fast Walsh-Hadamard Transform (FWHT), Discrete Wavelet Transform (DWT), and Lossy Delta Encoding (LDE) on weather sensor data. Results indicate that DCT and FWHT generate higher compression ratios than others. Regarding information loss, LDE outperforms others. Compression error is much severe in DCT and FWHT while LDE is able to maintain lower error rate than others.

In [69] Eliseev et al. give an overview of modern methods, models and technologies to process Big data in large-scale system such as Map-Reduce. It also shows that Big data processing takes significantly less time on a cluster, compared to a single computer.

In [70] Kosović et al. estimate value of daily solar radiation using machine learning (ML) algorithms. Input to ML algorithm are other collected parameters that are correlated to solar radiation. Using such approach, the amount of data that is being sent through whole IoT network is reduced. Cost is additionally lowered since one expensive sensor is removed from IoT network.

### 2.1.4 Energy Efficient Approaches for Data Velocity and Volume Reduction

In order to build an energy efficient IoT system that is able to tackle with Big data challenges it is important to have a systematic approach and understand where reduction in data volume and velocity can be made and what are the trade-offs. This section provides categorization of the approaches previously described with their applications on specific components of an IoT system as well as their trade-offs. This includes dynamic monitoring frequency, data filtering, compression, consolidation, aggregation, correlation, and finally on-site data analytics.

While dynamic monitoring frequency enable sensors to collect data from the environment on event, by dynamically changing the monitoring frequency to detect and store only changes, sensors may filter data and transmit only relevant data, while ignoring the rest. On one hand, they save on transmission as less data is transmitted. On the other hand, sensors still have to collect data and consume energy while doing so. Furthermore, data consolidation enables sensors to agree on transferring bulk of data instead of transmitting each value separately, e.g., collect data for 5 minutes and then bulk transfer all data at once. Similarly, a single sensor that collects different types of values, e.g. temperature and humidity, can bulk transfer these two values. Problem is that both sensor types must be read at the same interval, which excludes dynamic frequency approach. Data aggregation is a type of data and information mining process where data is searched, gathered and presented in a report-based, summarized form, e.g., storing average values or a trend instead of raw data. Using this approach, sensor can aggregate data in some predefined interval and send only aggregated values. It may also compare collected value with aggregated one and send both values if they vary significantly. Moreover, data correlation refers to a process of combining the data in some manner, either by correlating it with some other data and thus applying the soft(ware) sensor concept, namely correlating different sensor values to obtain additional one instead of measuring it with a hardware sensor, or by correlating the same type of sensor data between different sources. Lossless data compression on sensors can be achieved in several ways, one of which is to send only changes of values (deltas), instead of full values. If sensors are collecting data in predefined period, they could exclude timestamp and it can be calculated later from a known period. Furthermore, if sensors send a predefined structure, they can exclude variable names. Instead, gateways could be implemented to read the predefined structure. Finally, on-site data analytics performed on a gateway represents Edge – a concept for collecting and analysing data on the spot. In such scenarios, data is only forwarded to the cloud for storing or post-analysis, while all the decision making is moved closer to the Edge, namely gateways and sensors. Moreover, on-site data analytics applied in the cloud would refer to the streaming analytics, i.e., the ability to constantly, in real time, measure and analyse data while moving within the large data streams. Afterwards, data can be stored (in compressed, aggregated, consolidated form, or as raw data) for archive.

## 2.2 Data Velocity and Volume Reduction Approaches the Research Will Focus On

In the previous section, it is shown that researchers commonly focus on reductions of already collected data. However, in our research, the goal is to reduce the amount of collected data and sensor online time as well as to mitigate possible risks introduced by those reductions (Table 2.1). Therefore, our research will focus on improvements in three areas (Figure 2.1): (1) data collection with dynamic monitoring frequency; (2) estimation of the final value from the first part of the transient for sensors with preheating; and (3) signal type classification, which identifies the type of sensor signal being sent.

### 2.2.1 Dynamic Monitoring Frequency on a Sensor

This chapter gives an overview of methods which enable data volume reduction before even collecting the data. Such methods are not widely researched, however, some researchers did tackle this challenge.

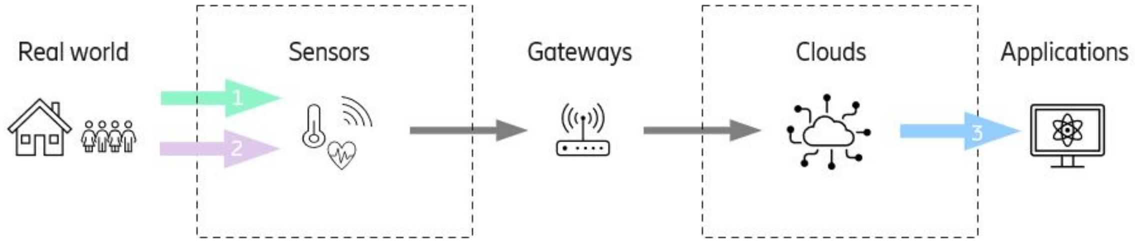


*Figure 2.1. Locations in IoT our research is focused on*

| | |
|---|---|
| Sensor input | [55–58]*, [34, 71]* |
| Sensor output | [56, 58–60], [72]* |
| Gateway input | [11, 12, 61–64] |
| Gateway output | [21, 65] |
| Cloud input | [66, 67] |
| Cloud output | [20, 68–70], [73]*, [74]*, [75–83]* |

*Data collection with dynamic monitoring frequency

*Estimating the final value from the first part of the transient

*Signal type classification

*Table 2.1. Data reduction in IoT - Related work overview*

In [55] Mastelić et al. introduce a dynamic monitoring frequency algorithm for collecting monitoring data from ultrascale systems. The algorithm deterministically reduces data velocity by dynamically self-adapting the monitoring frequency to the volatility of the collected data. It also reduces data volume because it collects data less often, however it still keeps the same data value as the equivalent static monitoring frequency. This algorithm solves the same problem, but it requires human (administrator) input of maximum and minimum periods, as well as the factor for calculating window size.

Trihinas et al. [56] introduce AdaM, a lightweight adaptive monitoring framework for battery-powered IoT devices with limitations in processing capabilities. The framework consists of two algorithms, namely one for adaptive sampling (i.e. dynamic monitoring frequency) and the other for adaptive filtering. Adaptive sampling algorithm dynamically adapts monitoring frequency based on the current evolution and variability of the metric stream with minor processor usage. Results show that it reduces volume and energy consumption while preserving relatively high accuracy. The second part of the AdaM algorithm, namely adaptive filtering, is useful for preparing sensor output. After sampling algorithm selects data to read, filtering algorithm analyses the dataset and decides for every input is it relevant and whether it should be passed to the next stage. AdaM framework can reduce data volume and energy consumption by approximately 70%, while preserving the accuracy greater than 89%. AdaM calculates standard deviation of distances for a time window, while in comparison, our solution learns offline from the past data, and does not consume additional energy on computation when applied online.

In [73] Lujić et al. present a three layer (gathering, edge and cloud) architecture model for data storage management on the Edge. This includes an adaptive algorithm that is dynamically finding a trade-off between data accuracy and data volume, while focusing on time series data. By using the proposed approach it is possible to reduce the amount of stored data by an average 73.02% and 80.27% in each cycle for the two datasets respectively, while satisfying demands for forecast accuracy and thereby showing potential for saving limited storage space.

Arendt et al. [57] present a model-predictive communication framework based on historical data analysis. Data collected in three previous days is used to predict fourth day values. Two algorithms, autoregression based ARIMA and a neural network with LSTM cells, are compared, in order to reduce communication effort. Results show a significant potential to reduce communication effort per day of at least 60% up to 95%, depending on the required accuracy, significantly contributing to the achievement of mMTC performance targets. The main bottleneck of this solution is its usage of historical data. The framework is used to predict one day after data is constantly collected for 3 days, thus it can either predict every 4th day, or use predicted values of 4th day to predict values of the 5th day. Latter option will increase error with time since only predicted data is used.

In [58], Cecchinel et al. investigates whether there is a significant battery lifetime gain when using adaptive sampling and sending periods (i.e. dynamic monitoring frequency). A self-adaptive approach using machine learning and deep-sleep is used to provide the optimal configuration extending the battery lifetime of a sensor platform. Gain in the battery lifetime is compared to a solution that uses fixed periods. It is shown that dynamic monitoring frequency successfully lowers the battery discharge. While adaptive sampling is optimization in input sensor data, adaptive sending periods optimize output of a sensor. Experimental validation shows that this combination of adaptive parameters extends battery lifetime of a temperature sensor from a monthly to a yearly basis, while ensuring a proper level of data quality and freshness.

Therefore, a sensor should only collect data using the lightweight algorithm when a defined delta is anticipated, sleeping otherwise. As a result, the monitoring interval increases and decreases depending on whether the current delta is greater or lower than a significant delta. In this manner, if data is changing slowly, the sensor may be in sleep mode for hours instead of minutes, rather than consuming energy for data collection and analysis only after data has been collected.

### 2.2.2 Predicting Final Value from Part of Transient for Sensors with Preheating

This section gives an overview of LSTM neural networks and their applications to time series data prediction.

Jia et al. in [34] propose a new, low-power, automatic, accurate, and wireless ammonia monitoring approach that uses metal oxide sensors. This approach does not wait for equilibrium as this consumes significant amount of energy, rather it tries to predict the resistance at equilibrium using the sensor's transient measurements in the short heating window (as short as 200ms) to predict the actual value. Prediction model is built on LSTM neural networks. Proposed model accurately predicts the equilibrium state resistance value with an average error rate of 0.12%. The final average estimation error for the ammonia concentration level is 9.38ppm.

Salhi et al. in [74] propose implementation of a preventive system for gas leakage and fire incidences in smart home environment to enhance safety using low-cost and low-energy consumption devices through M2M standard communication protocols. They applied supervised machine learning on several algorithms, predict the level of risks for fire and gas leakage and alert responsible person.

In [71] Osorio-Arrieta et al. present a new method to reduce the measurement time by the prediction of the steady-state using the transient response to ethanol for quartz crystal microbalance gas sensors coated with ethyl cellulose. The successive curve-fitting algorithm was implemented using the Gauss–Newton method to observe the evolution and behaviour of data obtained from several experimental measurements. The parameters of the model were determined, and the time constants and the magnitude of the steady-state response were analysed. Therefore, the steady-state response could be predicted with a 55% reduction in the measurement (detection) time, which can be of vital importance in many cases, especially for poisonous or inert gases.

Zhang et al. in [72] propose a systematic scheme including data collection and prediction in order to minimise both the required time and amount of measurement data for training, while achieving high prediction accuracy to achieve fast measurement with chemical sensors. For data collection, the proposed sliding window sampling approach with data augmentation is able to generate more sequence sets for training from limited amount of measurement data. Using such method, any small segment on the early transient response curve can be used for data sampling and test, enabling ease of implementation for practical applications. Further, a model combining long short-term memory (LSTM) neural network and polynomial fitting is designed for mixed feature extraction with less required data, which are then input to multilayer perceptron for prediction. The developed scheme is applied to real measurement data showing significant improvement of prediction accuracy compared to previous methods.

In order to estimate the final value from part of the transient, the LSTM neural network will be used, as it is a recurrent neural network that provides the best results for time series data. Researchers use LSTM for different time series problems. Chen et al. in [84] use LSTM network as a method to predict the mechanical state. The simulation results of LSTM network are compared with the results obtained with support vector regression machine (SVRM). It is found that using the same window width the MSE network test results of LSTM are smaller than the results of SVRM, making the LSTM model better than the SVRM model in the field of mechanical state monitoring and prediction. In [85] Wang et al. predict water quality using LSTM neural network. LSTM is compared with back propagation neural network (BP NN) and online sequential extreme learning machine (OS-ELM). Results on several simulations show that, compared with BP NN and OS-ELM, the predictive accuracy of LSTM NN is higher and more generalized. Yu et al. in [86] use LSTM neural network for spectrum prediction. LSTM network is compared to Back Propagation (BP) network results and results show that LSTM has better performance than BP in case of same number of hidden layers and neurons. Liu et al. in [87] use LSTM neural networks to analyse and predict stock transaction data. Results show accuracy of about 72% for the short period of data. Furthermore, Yao et al. in [88] propose LSTM network combined with fuzzy-rough set (FRS) theory for short-term wind speed prediction. The experimental results show that the FRS-LSTM model has about 40% higher prediction accuracy than the traditional BP neural network. In [89] Kim et al. propose short-term electricity consumption prediction method. LSTM network is used to predict month-ahead electricity consumption. Results on the real data show that the proposed method performs well with accuracy above 80%. They also state that the test accuracy can be improved with a longer period of training time and deliberate hyperparameter setting. Qian and Chen in [90] conduct a stationary analysis of the stock's time series data and then use the LSTM network to predict stock data under different stationary conditions. Results are compared with ARIMA algorithm results. As shown on large number of experimental results, the error rate of the LSTM algorithm is 66.78% lower than that of ARIMA. They also point that the main disadvantage of LSTM algorithm is that it takes a lot of time to train the model and requires large sample of data.

Finally, in this work, energy consumption of the gas sensor is considered, as well as the consumption for data transmission. While the focus is given on the gas sensor due to its higher consumption, there is also a large body of work focusing on energy consumption optimization in Wireless Sensor Networks [91,92] as forerunner of IoT. There is also a number of researches focusing on energy consumption optimization in IoT systems addressing other power-hungry components such as radio [93,94]. These approaches can be further used for decreasing the overall power consumption of such IoT systems.

### 2.2.3 Signal Type Classification - Time Series Data Classification

Time series classification is an important task in time series data mining. Researches have worked in this field during last decades and with the increase of time series data availability, hundreds of time series classification algorithms have been proposed. However, it remains a challenging problem due to high dimensionality, large data size and continuous update of time series data [75].

In [76] Geurts proposes tools to allow machine learning classifiers to cope with time series data. Patterns are extracted from models and combined in decision trees to give interpretable classification rules. Experiments show that 1-NN has the best results on all problems. One conclusion is that very good results can be obtained with simple feature sets; however by sacrificing interpretability. This observation justifies the fact that the application of machine learning on temporal data has focused on interpretability rather than on accuracy. In terms of accuracy, better results can be obtained by using either boosting, 1-NN with naive features or, in case where start time of characteristic events are highly variable, by pattern extraction.

In [77] Papadimitriou et al. introduce SPIRIT (Streaming Pattern dIscoveRy in multIple Time series) comprehensive approach to discover correlations that summarise large collections of streams. Given $n$ numerical data streams, all of which values are observed at each time tick $t$, SPIRIT can incrementally find correlations and hidden variables, which summarise the key trends in the entire stream collection. The discovered trends can also be used to spot potential anomalies, to do efficient forecasting and to simplify further data processing. Experimental evaluation shows that SPIRIT can incrementally capture correlations and discover trends. Moreover, SPIRIT-based forecasting is several times faster than other methods.

In [78], Wang et al. propose a baseline for time series classification from scratch with deep neural networks. Proposed baseline models are pure end-to-end models without any heavy preprocessing on raw data or feature crafting. The proposed fully convolutional network achieves premium performance compared to other state-of-the-art approaches. A simple multilayer perceptrons are found to be identical to the 1NN-DTW as the previous golden baseline.

In [79], Ismail et al. present a large empirical study of deep neural networks (DNNs) for time series classification. They give an overview of the most successful deep learning applications in various time series domains and with practical examples explain how deep learning can be adapted to one dimensional time series data. In their study, authors train 8,730 deep learning models across 97 time series datasets. The results show that end-to-end deep learning can achieve the current state-of-the-art performance for time series classification with architectures such as fully convolutional neural networks and deep residual networks.

In [80], Jastrzebska proposes a new approach for time series classification that transforms the scalar time series into a two-dimensional space of amplitude and a change of amplitude. Subsequently, this representation is used to plot the data producing one figure for each time series thus converting time series classification problem to the visual pattern recognition problem. This transformation allows applying a wide range of algorithms for pattern recognition and opens the possibility to apply clustering algorithms. Results on real-world datasets achieve a satisfying classification accuracy. Furthermore, the lengths of time series do not have to be the same, thus approach is well-suited for large-scale datasets with long time series.

In [81], Lamrini et al. investigate an unsupervised machine learning method based on one class support vector machine for anomaly detection in network traffic. In the absence of any prior expert knowledge on anomalous data, they propose the use of a similarity measure for multivariate time series to evaluate the output results and select the best model. The data samples that are provided to the OC-SVM classification algorithm are multivariate, each composed of four KPI segments over the same time-window. Each segment is characterized by seven statistical attributes, namely minimum, maximum, mean, median, standard deviation, number of average crossings and squared mean error, computed between the raw data and the linear fitting. OC-SVM method is applied to detect anomalies in real network traffic, contributing with an automatic method based on the similarity index for setting the hyper-parameters, which defines the learning configuration and provides satisfactory results.

In [82], Zhao et al. propose novel convolutional neural network framework for time series classification. Instead of using human designed features, CNN can automatically discover deep features. Experiments are conducted on both simulated datasets and real-world datasets from different application domains. Experimental results show that the CNN method outperforms the state-of-the-art methods in terms of classification accuracy and noise tolerance. Nevertheless, limitations include time consuming training of CNN as most of its parameters are determined by lots of experiments and fixed length of time series which is determined by the architecture of CNN.

In [83], Shahid et al. propose a new network discovery technique in order to find which IoT devices are connected to the network. Four IoT devices (sensor, camera, bulb, plug) are recognised by analysing streams of sent and received packets, using machine learning. Six different ML classifiers are used and the results are promising, with an overall accuracy as high as 99.9% achieved by random forest on a test set.

Most of the above mentioned researches focus on multi class classification approach. However, in [81] Lamrini et al. use one class classification, but they classify only one signal and try to detect anomalies. However, the goal of our approach is to compare three approaches for signal type recognition, namely one class classification ($1c$), two class classification ($2c$) and multi class classification ($mc$). Furthermore, our research is focused on both, recognition of streaming signal (as in [77]) as well as recognition of signal which is already collected and stored (as in [76], [78], [79], [80], [83] and [82]).

The main contribution of our analysis is that it covers multiple approaches, namely, one class, two class and multi class, which can be used for different scopes of problems. On one hand, multi class approach is the most simple, but most accurate one since it has the knowledge of all classes and their amount. On the other hand, one class approach recognises signal type without knowledge of any other class. All proposed approaches can be used not only to recognise one specific signal, but also to eliminate the least probable signal classes.

## 2.3 Summary

A number of researchers are dealing with data reductions in IoT. In the first part of the chapter, a general overview of existing solutions for data reductions on sensors, gateways and cloud was presented. This includes dynamic monitoring frequency, data filtering, compression, consolidation, aggregation, correlation, and finally on-site data analytics. The focus of the second part is on the specific techniques that this research is investigating. The goal of the presented research is to reduce the amount of collected data by eliminating data before it is even collected and therefore reduce sensor online time, through methods like the dynamic monitoring frequency approach or estimating the final value from the transient of sensors with preheat time. Furthermore, eliminating data before even collecting it, comes with trade-offs. In order to mitigate possible risks introduced by such reductions, signal type classification approach which identifies the type of sensor signal being collected and stored is investigated, and an overview of the existing techniques for time series data classification is presented.

# 3 Overview of Machine Learning Techniques

"Machine learning is the science (and art) of programming computers so they can learn from data" [4]. In 1997 Tom Mitchel described machine learning (ML) as: "A computer program is said to learn from experience $E$ with respect to some task $T$ and some performance measure $P$, if its performance on $T$, as measured by $P$, improves with experience $E$." Geron in [4] explains spam filter as an example of Mitchel's definition. Spam filter can learn to flag spam given examples of spam emails (e.g. flagged by users) and examples of regular (non-spam) emails. The examples that the system uses to learn are called the training set. Each training example is called a training sample. In this scenario, the task $T$ is to flag spam for new emails, the experience $E$ is the training data, and the performance measure $P$ needs to be defined. For example, this can be the ratio of correctly classified emails; this measure is called accuracy and it is often used in classification tasks.

Machine learning is commonly divided by the way algorithm is trained – with or without human help:

- *Supervised learning* – requires human to label data.
- *Unsupervised learning* – works with unlabeled data; system itself tries to learn data characteristics.
- *Semisupervised learning* – as it name says, semisupervised learning is the combination of previous two categories. Algorithms are usually trained in unsupervised manner, while fine tuning is done in supervised manner.
- *Reinforcement learning* – consists of the learning system, called an agent that observes the environment. Agent selects and performs actions, and gets positive or negative „rewards" in return. This way, it tries to learn the best strategy in order to get the largest reward.

Furthermore, there are two different types of problems in machine learning, both of which involve predicting a target variable based on input data, namely regression and classification. The main difference between these two is in the type of target variable being predicted. On the one hand, regression involves predicting a continuous target variable, such as a numerical value like temperature, stock price, or length of an object. Regression algorithms aim to find a relationship between the input variables and the output variable, which is represented by a function that maps the inputs to a continuous output. On the other hand, classification involves predicting a categorical target variable, such as a binary value (yes/no) or a multi-class label (e.g. type of vehicle). Classification algorithms aim to classify input data into the one of several predefined classes based on the patterns from the input variables. In summary, the key difference between regression and classification is that regression involves predicting a continuous target variable, while classification involves predicting a categorical target variable [4, 47].

Through our research we utilize data-driven predictive approach, using supervised learning [95]. In the following sections, supervised machine learning algorithms used in this dissertation will be explained and compared.

## 3.1 Linear and Logistic Regression (LR)

Linear and logistic regression (LR) are developed by many researchers over time. Linear regression was first introduced by English statistician Sir Francis Galton in the late 19th century [96]. Furthermore, logistic regression was independently developed by the several statisticians and mathematicians in the early 20th century, while the logistic regression as a general statistical model was originally developed and popularized primarily by Joseph Berkson in the 1940s [97].

Both linear and logistic regression are statistical methods used for modeling the relationship between a dependent variable and one or more independent variables (Figure 3.1). However, they differ in their approach and the type of data they are suited to model [4, 47]; linear regression is used for regression problems, while logistic regression is used for classification problems.

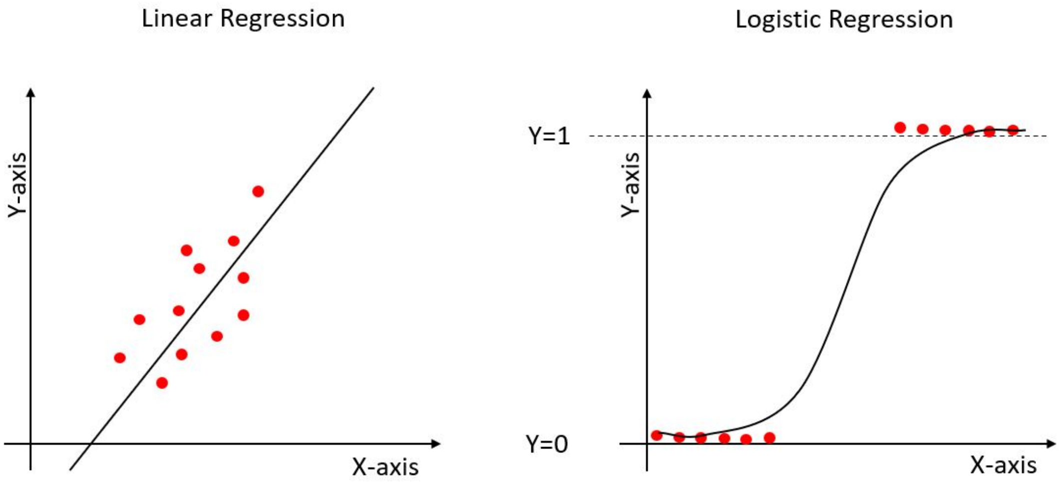


*Figure 3.1. Illustration of Linear and Logistic regression [1]*

### 3.1.1 Linear Regression

Linear regression is used when the dependent variable is continuous and the relationship between the dependent variable and the independent variables is linear. The goal of linear regression is to find the best-fitting line (or hyperplane in higher dimensions) through the data points that minimises the sum of the squared differences between the predicted and actual values. Furthermore, linear regression can also be used to solve simple and multiple regression problems, with simple regression involving only one independent variable and multiple regression involving multiple independent variables [96].

Linear regression tries to model the relationship between a dependent variable $y$ and one or more independent variables (features) $x$, assuming a linear relationship between them [4]:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n + \varepsilon \tag{3.1}$$

where $\hat{y}$ is predicted value; $n$ is the number of features; $x_i$ is the $i^{th}$ feature; $\theta_j$ is $j^{th}$ model parameter (including the bias term $\theta_0$ and the feature weights $\theta_1$, $\theta_2$,..., $\theta_n$); and $\varepsilon$ is the error term (the part of $y$ that cannot be explained by the independent variables).

Or in vectorized form:

$$\hat{y} = h_\theta(x) = \theta^T \cdot x + \varepsilon \tag{3.2}$$

where $\theta$ is the model's parameter vector, containing the bias term $\theta_0$ and the feature weights $\theta_1$ to $\theta_n$; $\theta^T$ is the transpose of $\theta$ (a row vector instead of a column vector); $x$ is the instance's feature vector, containing $x_0$ to $x_n$, with $x_0$ always equal to 1; $\theta^T \cdot x$ is the dot product of $\theta^T$ and $x$; and $h_\theta(x)$ is the hypothesis function, using the model parameters $\theta$; $\varepsilon$ is error term (the part of $y$ that cannot be explained by the independent variables).

The goal of linear regression is to estimate the values of the regression coefficients that minimise the error term $\varepsilon = \hat{y} - \theta^T \cdot x$, i.e. to minimise the difference between the predicted and actual values of $y$. Common method for $\varepsilon$ minimisation is sum of squared errors, mean square error or root mean square error.

### 3.1.2 Logistic Regression

When the dependent variable is categorical and binary, meaning it can only have two possible values (e.g., 0 or 1), logistic regression is used. The goal of logistic regression is to model the *probability* of a binary dependent variable $y$ (0 or 1) as a function of one or more independent variables $x$. The estimated probability of the model in vectorized form is [4]:

$$\hat{p} = h_\theta(x) = \sigma(\theta^T \cdot x) \tag{3.3}$$

where logistic $\sigma$ is a sigmoid function that calculates a probability that an instance $x$

belongs to the positive class and outputs the value between 0 and 1:

$$\sigma(t) = \frac{1}{1+e^{-t}} \tag{3.4}$$

where $t$ is a linear combination of the independent variables $x$ and their regression coefficients $\theta$:

$$t = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n \tag{3.5}$$

The goal of logistic regression is to estimate values of the regression coefficients $\theta$ that maximise the likelihood of observing the dependent variable $y$ given the independent variables $x$. This can be done using maximum likelihood estimation, which involves finding $\theta$ values that maximise the likelihood function. In practice, this is often done using iterative optimization algorithms such as gradient descent. Logistic regression is commonly used in binary classification problems, where the goal is to predict whether a data point belongs to one of two classes (e.g., yes/no, true/false, etc.). However, it can also be used in multi-class classification problems by extending the logistic function to model the probabilities of multiple classes and it is called Multinomial logistic regression.

### 3.1.3 Comparison between Linear and Logistic Regression

Both linear and logistic regression, in terms of similarities, involve fitting a model to data by estimating parameters that minimise some objective function, such as the sum of squared errors or the maximum likelihood. They are both based on the assumptions of independence, linearity, and homoscedasticity (equal variance). Additionally, both methods can be used for prediction and inference, where prediction involves using the model to make predictions on new data points, and inference uses the model to test hypotheses about the relationship between the dependent and independent variables.

However, the key differences between linear and logistic regression are in the type of dependent variable they are suited to model and the approach they take to modeling the relationship between the dependent and independent variables; linear regression is used for continuous outputs (regression) and predicts the values, while logistic regression is used for categorical outputs (classification) and predicts the probabilities (0,1) of particular outcomes rather than the outcomes themselves.

## 3.2 Random Forest (RF)

Random Forest (RF) is a popular ensemble learning method that combines multiple decision trees [98] at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees, to produce a more accurate and stable prediction [99, 100].

The algorithm is based on the bagging (bootstrap aggregating) technique, which involves creating multiple models from subsets of the training data, and then combining their predictions to reduce overfitting and improve the accuracy of the model. Random forest algorithm consists of the following steps (Figure 3.2):

- *Bootstrapping* - a random sample of size $n$ is drawn with replacement from the original training dataset. This creates a bootstrap sample used to train a decision tree.
- *Feature selection* - at each node of the decision tree, a random subset of $m$ features is selected from the total set of $p$ features. This helps to reduce the correlation between the trees and improve the diversity of the forest.
- *Building decision trees* - a decision tree is built for each bootstrap sample and selected features using a specific splitting criterion (e.g. information gain, Gini impurity). The process is repeated until a predetermined stopping criterion is met (e.g. maximum tree depth, minimum number of samples per leaf).
- *Aggregating predictions* - the final prediction of the RF model is obtained by aggregating the predictions of all decision trees.
    - For classification problems, this can be done by taking the majority vote:

$$\hat{y} = mode(y_1, y_2, ..., y_n) \tag{3.6}$$

where $y_1, y_2, ..., y_n$ are the predicted classes from each decision tree.

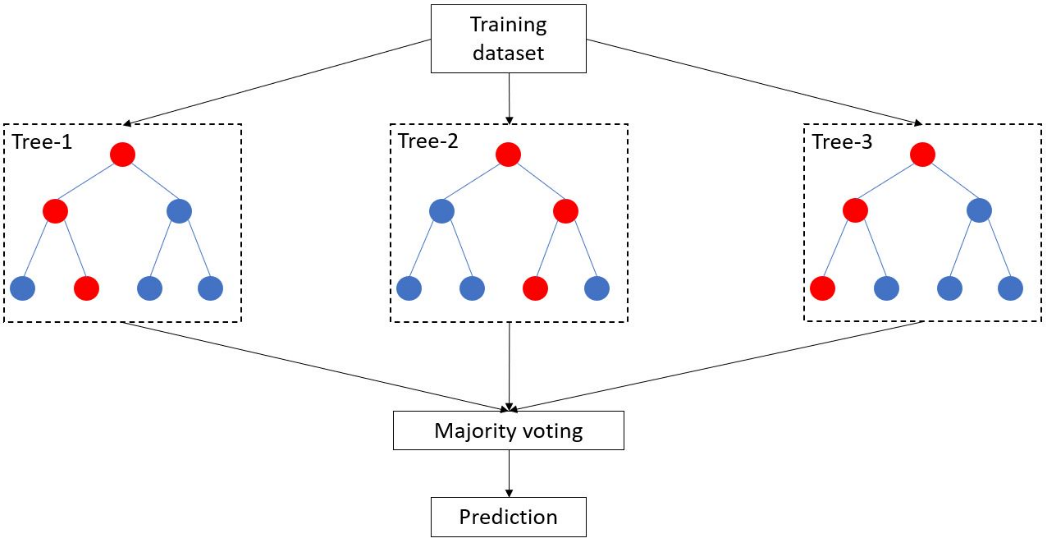


*Figure 3.2. Illustration of random forest [2]*

- For regression problems, this can be done by taking the average:

$$\hat{y} = \frac{y_1 + y_2 + ... + y_n}{n} \tag{3.7}$$

where $y_1, y_2, ..., y_n$ are the predicted values from each decision tree.

The combination of bootstrapping and feature selection ensures that each decision tree is constructed using a different subset of the data and features, leading to a diverse set of trees that are less prone to overfitting. Therefore, the random forest algorithm has several advantages over a single decision tree, including better accuracy and reduced overfitting. Furthermore, RF can handle noisy and high-dimensional data, missing values and outliers, and provide feature importance measures.

## 3.3 Support Vector Machine (SVM)

Support vector machine (SVM) [101] is a type of supervised machine learning algorithm used for classification and regression analysis. It is particularly useful for solving complex classification problems where the data is not linearly separable.

The basic idea behind SVM is to find the hyperplane that maximally separates the classes in the data. Hence, SVM is an example of large margin classifier. A hyperplane is a decision boundary that separates data points into different classes (Figure 3.3). In two-dimensional space, a hyperplane is just a line, while in higher-dimensional space, it is a plane or a hyperplane. Therefore, SVM tries to find a hyperplane that separates the training data into two classes, typically labeled as +1 and -1. The hyperplane is defined by the equation [47]:

$$Ax - b = 0 \tag{3.8}$$

where $A$ is the vector of weights of the linear function; $x$ is vector of input data which is typically represented as a feature vector of $n$ dimensions; and $b$ is the bias term.

To find the optimal hyperplane, SVM looks for the hyperplane that maximises the margin between the two classes. The margin is the distance between the hyperplane and the closest data points from both classes. By maximising the margin, SVM ensures that the hyperplane is as far away from the data points as possible. Therefore, it is less sensitive to noise or outliers in the data. Data points closest to the margin are called support vectors, because they are the only ones that matter in determining the hyperplane.

For linearly separable binary class data, in order to maximise the margin, the Euclidean (L2) norm $||A||$ of the weight vector $A$ needs to be minimised. This constraint can be written as:

$$y_i(Ax_i - b) \geq 1; \forall i = 1, ..., n \tag{3.9}$$

Since not all datasets are linearly separable, loss function for $n$ points that cross the margin lines is defined [47]:

$$\frac{1}{n}\sum_{i=1}^{n}\max(0,1-y_i(Ax_i-b))+\alpha||A||^2 \tag{3.10}$$

where $A$ is the vector of weights of the linear function; $b$ is the bias term; $x_i$ is support vector; $y_i$ is the class label, where $y_i=+1\ or\ -1$; and $\alpha$ is the soft separability regularization parameter.

The product $y_i(Ax_i-b)$ is always greater than 1 if the point is on the correct side of the margin. Thus, the left term of the loss function equals to zero, and the only influence on the loss function is the size of the margin, depending on value $\alpha$. The larger values of $\alpha$ result in more emphasis on widening the margin, and smaller $\alpha$ values result in the model acting more like a hard margin, while allowing data points to cross the margin, if needed.

Finally, if the data is not linearly separable, SVM uses a kernel function to map the data into a higher-dimensional space where it becomes linearly separable (kernel trick). Hence, different kernels are used for different ML problems. The most commonly used kernel functions are linear, polynomial, and radial basis function (RBF). Once the optimal hyperplane is found, SVM can be used to classify new data points by checking which side of the hyperplane they lie on. In summary, SVM is a powerful algorithm for solving complex classification problems by finding the optimal hyperplane that maximises the margin between the classes.

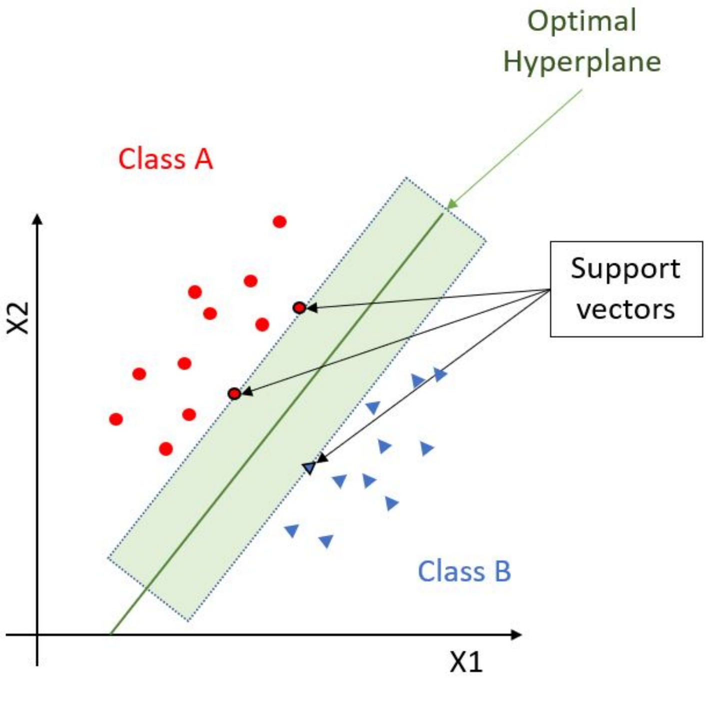


*Figure 3.3. Illustration of Support vector machine [1]*

## 3.4 K-Nearest Neighbours (kNN)

The k-Nearest Neighbors (kNN) algorithm [102, 103] is a non-parametric machine learning algorithm used for classification and regression tasks. It is a type of instance-based learning or lazy learning, where the model does not learn a function from the training data, but instead stores the training data points in memory to make predictions based on the similarity to the new data point (Figure 3.4). Hence, all of the calculation is executed at the time the prediction is requested.

The kNN algorithm consists of following steps:

1. Choosing $k$: Choosing the number of neighbors to consider when making predictions ($k$) [4].

2. Calculating distance metrics: Given a new data point, the algorithm calculates the distance between the new point and all training points in the dataset to find the k-nearest neighbors of a new data point. The distance can be calculated using various distance metrics, such as Euclidean distance, Manhattan distance, Chebyshev distance, Mahalanobis distance, Canberra distance, Bray-Curtis distance, Cosine similarity, etc. These are just a few examples of distance metrics that can be used in kNN algorithm. The choice of distance metric depends on the nature of the problem and the type of used data [47].

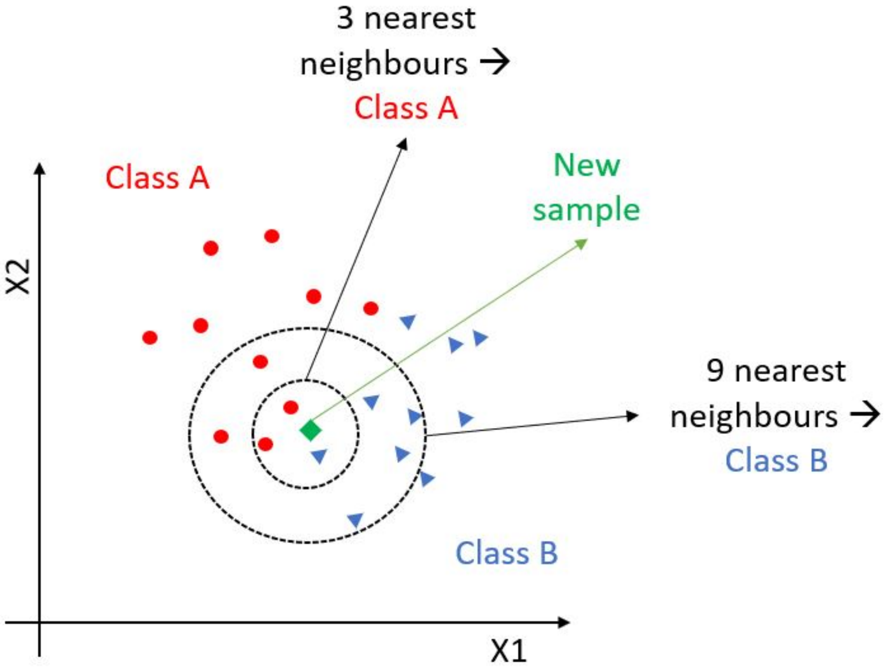


*Figure 3.4. Illustration of an example of the k-nearest neighbor algorithm [3]*

Common specifications of the distance metric for points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ are:

- Manhattan distance, also known as $L1$ norm, is distance metric that measures the distance between two points by calculating the sum of the absolute differences of their coordinates.

$$d_{L1}(x,y) = \sum_{i=1}^{n} |x_i - y_i| \tag{3.11}$$

- Euclidean distance, also known as $L2$ norm, measures the shortest distance between two points in a straight line:

$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2} \tag{3.12}$$

- Minkowski distance, commonly known as $Lp$ norm, is a generalized distance metric and can be defined as:

$$d_{Lp}(x,y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{3.13}$$

  where $p$ is a parameter that determines the order of the distance metric. When $p = 1$, the Minkowski distance reduces to the Manhattan distance, and when $p = 2$, it reduces to the Euclidean distance.

3. Selecting k-nearest neighbors: The algorithm then selects the k-nearest neighbors based on the calculated distance. The neighbors are $k$ training points that are closest to the new data point.

   kNN algorithm uses a decision boundary to separate the different classes. In the case of a binary classification task, the decision boundary is simply a line that separates the two classes. In the case of a multi-class classification task, the decision boundary is a hyperplane that separates the different classes.

4. Assignig the new data point: The algorithm then assigns the new data point. The way of assigning depends on type of the task:

   (a) Classification: After finding k-nearest neighbors, algorithm assigns the new data point to the class that is most common among its k-nearest neighbors. In other words, the algorithm uses the majority class of the k-nearest neighbors as the predicted class for the new data point.

$$f(z) = \max_{j} \sum_{i=1}^{k} \varphi(d_{ij}) I_{ij} \tag{3.14}$$

where $j$ is the number of classes, $i$ is training point, $\varphi(d_{ij})$ is the the weighted distance from the prediction point to the $i$, and $I_{ij}$ is an indicator function if point $i$ is in class $j$.

(b) Regression: For regression tasks, algorithm assigns the new data point the average value of the k-nearest neighbors. In other words, the algorithm calculates average of the target values of the k-nearest neighbors and assigns it as the predicted target value for the new data point.

$$f(z) = \frac{1}{k}\sum_{i=1}^{k}\varphi(d_i) \tag{3.15}$$

where $i$ is training data point, and $\varphi(d_{ij})$ is distance from the prediction point to the training point.

5. Repeat: The algorithm repeats steps $2-4$ for every new data point in the test set.

That being said, when using kNN algorithm, there are some important things to consider:

- The value of $k$ can also impact the performance of the algorithm. A smaller value of $k$ may lead to overfitting, while a larger value of $k$ may lead to underfitting.
- The choice of the distance metric can have a significant impact on the performance of the algorithm. For example, the Manhattan distance may not work as well as Euclidean distance in higher dimensions, as it can give equal importance to each dimension and may not take into account the correlations between them. On the one hand, Manhattan distance can be useful in cases where features have different units or scales and cannot be compared on the same scale, as well as, when the data is structured in a grid-like pattern. On the other hand, Euclidean distance is preferred when the data is spread out in all directions, such as in a continuous space. Furthermore, the computational complexity of calculating the Minkowski distance increases with the value of $p$, as it involves taking the $p-th$ power of the differences between the coordinates. Therefore, choosing a large value of $p$ can lead to slower performance of the kNN algorithm.
- The kNN algorithm can be sensitive to the scale of the features. For example, if one feature has a much larger range of values than the other features, then it will dominate the distance metric. To avoid this issue, it is often necessary to normalize or standardize the data before using kNN.
- The kNN algorithm can be computationally expensive, especially when dealing with large datasets. One way to reduce the computational complexity is to use data structures like KD trees [104] or Ball trees [105] to store the training data points, which can speed up the search for the nearest neighbors.

## 3.5 Artificial Neural Networks (ANN)

Artificial neural networks, commonly called neural networks got their name from biology, since they are inspired by animal brain i.e. biological neural networks [47–49]. They were introduced in 1943 by neurophysiologist Warren McCulloch and the mathematician Walter Pitts [106]. Neural networks are commonly used in supervised manner, but some neural network architectures can be unsupervised, such as autoencoders and restricted Boltzmann machines. Deep belief networks and unsupervised pretraining are example of semisupervised neural networks [4]. The most common deep neural networks are multilayer perceptrons (MLP), convolutional neural networks (CNN) and recurrent neural networks (RNN). Although CNN, especially one dimensional CCNs can be used for time series data, throughout the next subchapters only ANNs used in this dissertation will be explained, namely MLP and RNN.

### 3.5.1 Multilayer Perceptron (MLP)

A multilayer perceptron (MLP), also known as feedforward artificial neural network is the simplest ANN [107]. MLP is a mathematical function mapping set of input values to output values. The function is formed by composing many simpler functions. While mapping input values $x$ to output values $y = f(x;\theta)$, it learns the value of parameter $\theta$ that gives the best function approximation [48].

As shown in Figure 3.5, MLP consists of three main layers; one input layer, one or more hidden layers and one output layer. ANN that has two or more hidden layers is called a

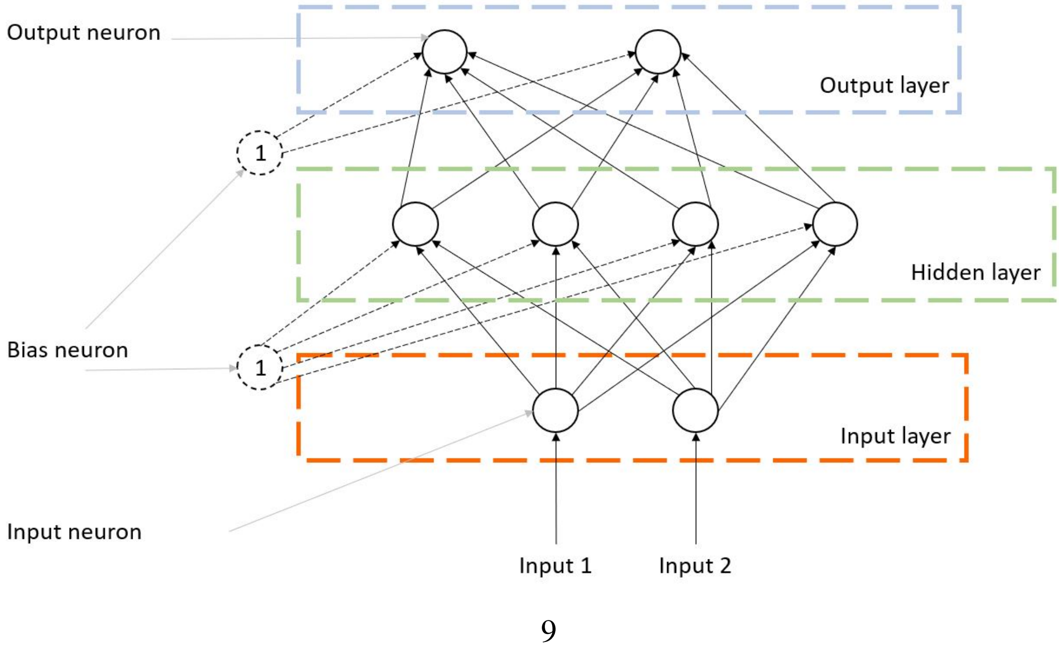


*Figure 3.5. Multilayer Perceptron [4]*

deep neural network (DNN). For example, there might be three functions $f1, f2$ and $f3$ connected in a chain, to form $f(x) = f3(f2(f1(x)))$. In this case, $f1$ is the first layer of the network, $f2$ is the second layer, and $f3$ is the third. The number of these functions is the depth of the model. Each hidden layer contains multiple neurons – elementary units in an ANN that determines the width of the model. The neuron in ANN receives one or more inputs and sums them to produce an output (or activation). MLP commonly uses a non-linear activation functions, such as sigmoid, softmax, hyperbolic tangent (tanh) and rectified linear unit (ReLU) [4, 48].

MLP activation functions:

- Sigmoid function - a non-linear activation function that maps any input value to a value between 0 and 1. It is often used in the output layer of MLPs for binary classification problems.
- Softmax function - a non-linear activation function that maps a vector of input values to a probability distribution that sums to 1. It is often used in the output layer of MLPs for multi-class classification problems.
- Hyperbolic tangent (tanh) - a non-linear activation function that maps any input value to a value between -1 and 1. It is often used in the hidden layers of MLPs.
- Rectified linear unit (ReLU) - a non-linear activation function that maps any input value less than 0 to 0, and any input value greater than 0 to the input value itself. ReLU is often used in the hidden layers of MLPs to overcome the numerical problems related to the sigmoids. It is commonly used for large datasets since it usually converges the fastest.

### 3.5.2 Recurrent Neural Networks (RNN)

Recurrent neural networks or RNNs [108] are a family of neural networks for processing sequential data. It looks very much like a MLP, except it has connections pointing backward. Figure 3.6 depicts one recurrent neuron receiving input $x$, producing an output $y$, and sending that output back to itself. At each time step $t$, neuron input $x(t)$ and the output the previous time step $y(t-1)$. RNN is specialized for processing a sequence of values $x(1),...,x(t)$.

RNN will be described in more detail on example of long short-term memory neural networks which are commonly used.

#### Long Short-Term Memory (LSTM) Neural Networks

Long short-term memory (LSTM) neural network [109] is a type of RNN that has a feedback connection. Along with single data points it can process data sequences as well. A common LSTM cell contains an input gate, an output gate and a forget gate, as depicted in Figure 3.7.

The cell remembers the values over a time period and three gates (input, output and forget gate) regulate the data flow in and out of the cell. The forget gate decides which information will be removed from the cell state. The input gate decides which states will be updated and the output gate decides which part of the cell states will be outputted. Therefore, LSTM has the ability to remove or add information to the cell state, instead of a mechanism that completely overrides cell states taken by classical RNN [47,49,84].

LSTM cell state is split in two vectors: $h_{(t)}$ and $c_{(t)}$; $h_{(t)}$ stands for short-term state while $c_{(t)}$ stands for the long-term state. On one hand, an input vector $x_{(t)}$ and a previous short-term state $h_{(t-1)}$ are used as inputs to the four different fully connected layers, namely $f_{(t)}$, $g_{(t)}$, $i_{(t)}$ and $o_{(t)}$ depicted in Figure 3.7. The main layer that analyses $x_{(t)}$ and $h_{(t-1)}$ outputs $g_{(t)}$, which is used for calculating $h_{(t)}$ and output vector $y_{(t)}$ expressed with:

$$g_{(t)} = tanh(W_{xg}{}^{T} * x_{(t)} + W_{hg}{}^{T} * h_{(t-1)} + b_g) \tag{3.16}$$

On the other hand, a previous long-term state $c_{(t-1)}$ enters the cell and goes through the forget gate, which defines parts of the long-term state that should be forgotten, and is controlled by the output of the $f_{(t)}$ layer expressed with:

$$f_{(t)} = \sigma(W_{xf}{}^{T} * x_{(t)} + W_{hf}{}^{T} * h_{(t-1)} + b_f) \tag{3.17}$$

where $b_g$ and $b_f$ are bias terms, $W_{xg}$ and $W_{xf}$ are weight matrices of the input vector $x_{(t)}$, and $W_{hg}$ and $W_{hf}$ are weight matrices of the previous short-term state $h_{(t-1)}$.

Other layers also give logistic output, namely 0 or 1, which control the closing and opening of the gates, respectively. Layer $i_{(t)}$ controls the input gates, while $o_{(t)}$ controls

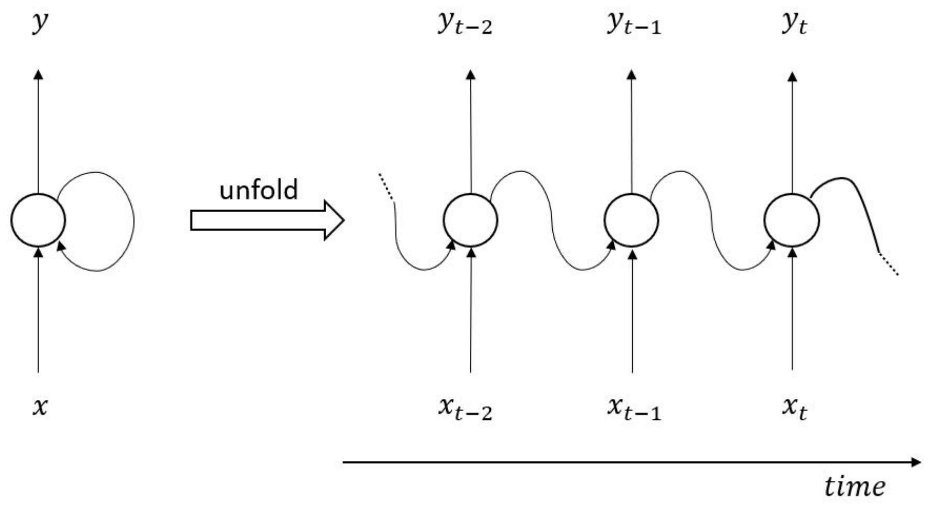


*Figure 3.6. A recurrent neuron [4]*

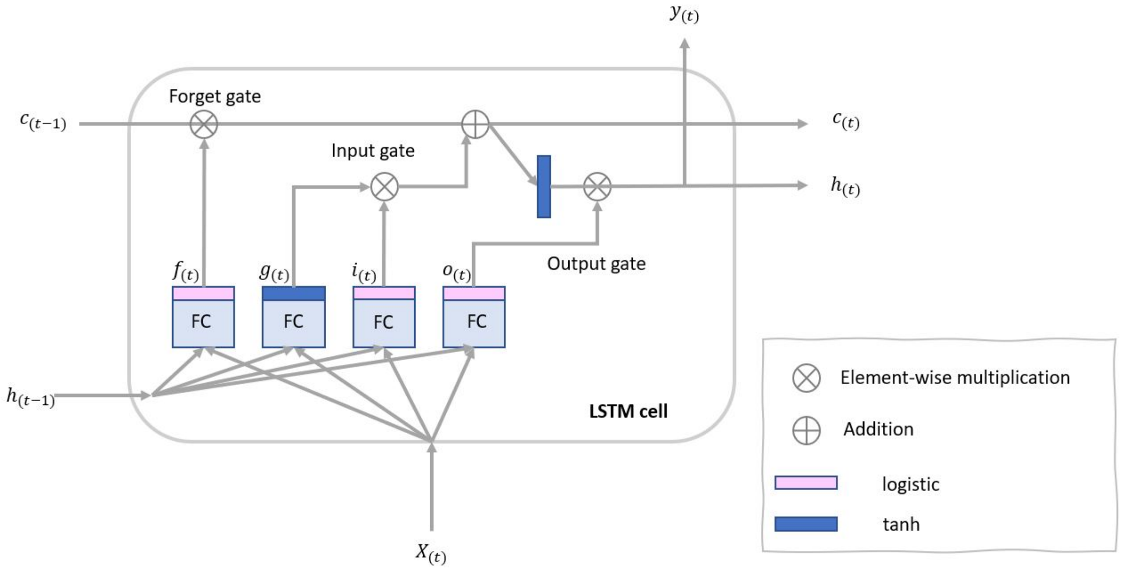


*Figure 3.7. Long short-term memory (LSTM) cell [4]*

the output gates, and both can be expressed with:

$$i_{(t)} = \sigma({W_{xi}}^T * x_{(t)} + {W_{hi}}^T * h_{(t-1)} + b_i) \tag{3.18}$$

$$o_{(t)} = \sigma({W_{xo}}^T * x_{(t)} + {W_{ho}}^T * h_{(t-1)} + b_o) \tag{3.19}$$

where $b_i$ and $b_o$ are again bias terms, while pairs of $W_{xg}$ and $W_{xf}$, as well as $W_{hg}$ and $W_{hf}$ are weight matrices of the input vector $x_{(t)}$ and the previous short-term state $h_{(t-1)}$, respectively.

When some long-term memories are removed, output from the forget gate is combined with $g_{(t)}$, which gives a long-term state:

$$c_{(t)} = f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \tag{3.20}$$

Additionally, the output from $g_{(t)}$ goes through the input gate, which defines what part of it should be combined with a duplicate of the long-term state $c_{(t)}$. Such, they enter the *tanh* function and go through the output gate, which defines what part of it should be outputted as both $h_{(t)}$ and $y_{(t)}$, expressed with:

$$y_{(t)} = h_{(t)} = o_{(t)} \otimes tanh(c_{(t)}) \tag{3.21}$$

Output $y_{(t)}$, as well as the short-term state $h_{(t)}$ and the long-term state $c_{(t)}$ are given for each input $x_{(t)}$, which is represented in a form of time series data.

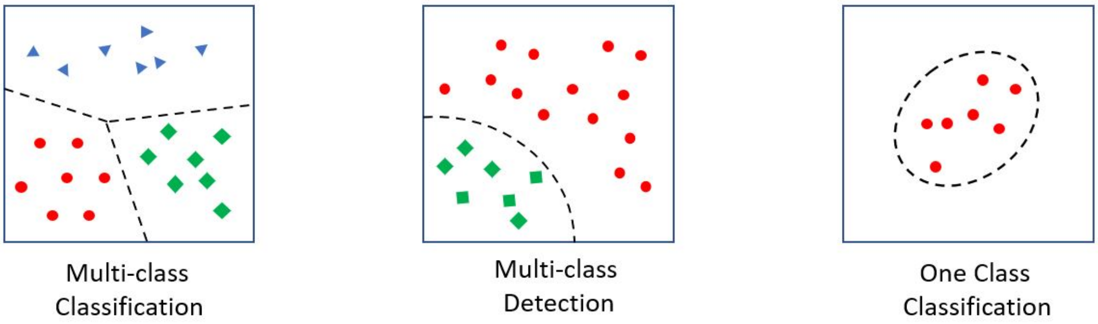


*Figure 3.8. One class classification vs multi-class classification/detection [5]*

## 3.6 One Class Classification

All algorithms previously described in this chapter can be used for either, regression or classification of binary or multiple classes. In traditional classification problems, the goal is to predict the class label of a new data point based on a set of features and a labeled dataset. However, in some cases, it may be difficult or even impossible to obtain negative examples or data from other classes, making it challenging to train a traditional classification model.

One class classification, which is commonly used for anomaly detection, addresses this problem by learning a model of the positive class using only positive examples. Later, the classifier encounters data from both the positive and negative class. The goal of one class classification is to determine whether an observed data point belongs to the class observed during training (positive class). The term one class classification (OCC) was introduced by Moya and Hush in 1996 [110].

Figure 3.8 illustrates differences between the OCC and multi-class detection and classification. The key difference is that OCC has only data from the positive class during training. The learned classifier defines a boundary that encompasses positive data with the hope that it will provide good separation from the other objects in the world. In the absence of multiple class data, both learning features and defining classification boundaries become more challenging in OCC compared to multi-class classification [5].

OCC has many applications, such as in fraud detection, intrusion detection, and anomaly detection, where the goal is to identify rare events or instances that do not conform to the norm. Furthermore, since OCC is commonly used to detect abnormal data points, it can be used to address issues related to severely imbalanced datasets, which are common in Big data [111].

In the remainder of this chapter, two OCC algorithms used in the dissertation will be described, namely one class support vector machine and an isolation forest.

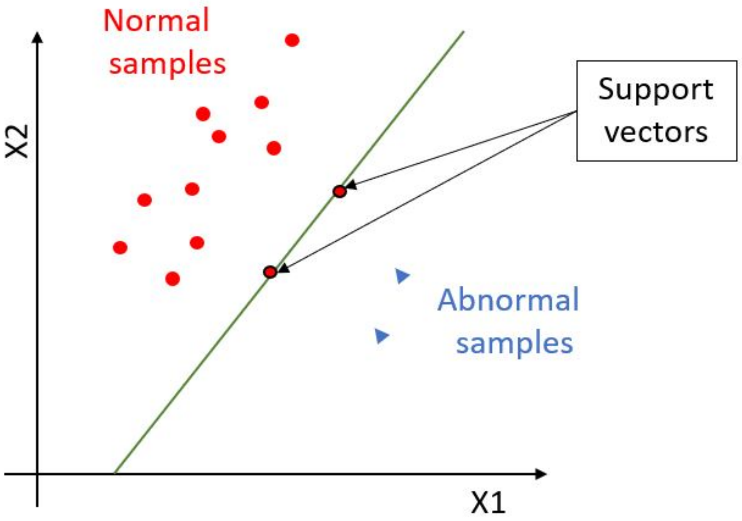


*Figure 3.9. One class support vector machine [6]*

### 3.6.1 One Class Support Vector Machine (OCSVM)

One class support vector machine (OCSVM) is unsupervised model for anomaly or outlier detection. Unlike the regular supervised SVM, the OCSVM does not have target labels for the model training process. Instead, it learns the boundary for the normal data points and identifies the data outside the border to be anomalies (Figure 3.9). It works on the basic idea of minimising the hypersphere of the single class of examples in training data and considering all other samples outside the hypersphere to be outliers or out of training data distribution [112].

To train a OCSVM, the algorithm first maps the input data to a higher-dimensional space, where it can more easily find a boundary between the data points. Then, it selects a hyperplane that maximises the margin between the data points and the hyperplane.

For a set of $n$ data points in $p$-dimensional space, denoted by $x_1, x_2, ..., x_n$, OCSVM is looking for a hyperplane that separates these points from the rest of the space. The hyperplane is represented with the equation:

$$Ax - b = 0 \tag{3.22}$$

where $A$ is the vector of weights of the linear function; $x$ is vector of input data which is typically represented as a feature vector of $n$ dimensions; and $b$ is the bias term.

To find the optimal hyperplane, following optimization problem needs to be solved [112]:

$$\min_{A \in F, \xi \in R^N, b \in R} \frac{1}{2}||A||^2 + \frac{1}{Nv}\sum_{i=1}^{N} \xi_i - b \tag{3.23}$$

subject to:

$$Ax_i \geq b - \xi_i, \xi_i \geq 0 \tag{3.24}$$

where $N$ is the dataset length used for training; $v$ is the regularization parameter; $\xi_i$ is the slack variable corresponding to each dataset; ω and ρ are the decision planes that can be decided with participation; and $||A||^2$ is the Euclidean (L2) norm of the weight vector $A$.

Weight vector $A$ and bias term $b$ that define the hyperplane can be found by solving the above optimization problem. The support vectors are the data points that lie on the margin or inside the margin, and they are used to determine the boundary of the class. To classify a new data point $x$, $Ax - b$ should be computed. If the result is $Ax - b \geq 0$, the point belongs to the same class as the training data. Otherwise, it is considered an outlier.

### 3.6.2 Isolation Forest (IF)

Isolation forest (IF) is an unsupervised algorithm that works by randomly partitioning data and creating a tree-based structure to isolate the anomalies, which are the instances that do not fit model of the normal data (Figure 3.10). The basic idea behind IF is that anomalies are more likely to be isolated in fewer steps than normal data points [113].

Let's say there is a dataset with $n$ data points, denoted by $x_1, x_2, ..., x_n$. In order to isolate anomalies, isolation forest algorithm will execute the following steps:

1. Randomly select a feature from the dataset.

2. Randomly select a split value: The split value is chosen uniformly at random between the minimum and maximum values of the selected feature.

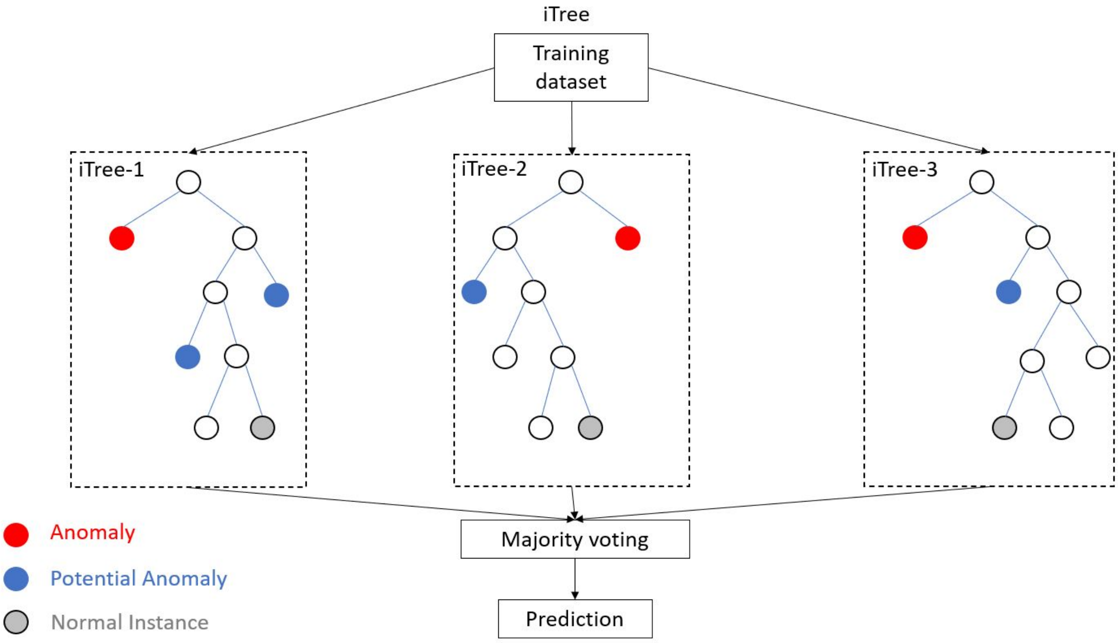


*Figure 3.10. Isolation forest [7]*

3. Partition the data based on the selected feature and split value: The data is then divided into two subsets based on the selected feature and split value. If a data point is less than or equal to the split value, it is assigned to the left subset, otherwise it is assigned to the right subset. The process is then repeated recursively for each subset until each data point is isolated in its own leaf node. The number of steps required to isolate a data point is defined as the data point's path length. The path length is the number of edges from the root node to the leaf node containing the data point. The average path length $h(x)$ for a given dataset is then computed by constructing multiple trees using random features and split values. The average path length is used to compute an anomaly score for each data point, which is defined as [113]:

   $$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{3.25}$$

   where $H(i)$ is the harmonic number and it can be estimated by $ln(i)$ + Euler's constant $(e)$; $c(n)$ is the average of $h(x)$ for given $n$, it is used to normalise $h(x)$.

   The anomaly score $s(x)$ is a measure of how easily a data point can be isolated in the tree. The anomaly score $s$ of an instance $x$ is defined as:

   $$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3.26}$$

   where $E(h(x))$ is the average of $h(x)$ from a collection of isolation trees. In Equation 3.26:

   - when $E(h(x)) \to c(n), s \to 0.5$;
   - when $E(h(x)) \to 0, s \to 1$;
   - when $E(h(x)) \to n-1, s \to 0$

4. Repeat steps 1-3 until each data point is isolated in its own leaf node.

   - if instances return $s$ very close to 1, then they are definitely anomalies,
   - if instances have $s$ much smaller than 0.5, then they are quite safe to be regarded as normal instances
   - if all the instances return $s \approx 0.5$, then the entire sample does not really have any distinct anomaly

5. Once the anomaly scores are computed, a threshold can be set to classify data points as anomalies or normal data points. Data points with scores below the threshold are classified as anomalies since they require fewer steps to be isolated.

Furthermore, IF can be easily parallelized, making it scalable to large datasets. It is also an efficient algorithm for high-dimensional data since it is not affected by the curse of dimensionality like other algorithms such as k-means clustering.

The main hyper-parameters of IF are the number of trees and the maximum depth of each tree. Increasing the number of trees improves accuracy of the algorithm but also increases the computational cost. However, increasing the maximum depth of each tree improves the accuracy but may lead to overfitting.

## 3.7 Comparison and Summary

This chapter gave an overview of the ML techniques used in the herein presented dissertation, namely linear and logistic regression, random forest, support vector machine, k-nearest neighbors, artificial neural networks (multilayer perceptrons and long short-term memory) and one class classification algorithms (one class support vector machine and isolation forest).

Linear regression is a simple and widely used regression algorithm which works by fitting a line to the data and minimising the sum of the squared distances between predicted and actual values. It is useful when the relationship between the input variables and the output variable is linear.

Logistic regression is a type of regression algorithm used for binary classification problems which works by fitting a sigmoid curve to the data and predicting the probability of the output variable being 1. It is useful for the binary outputs.

Random forest is an ensemble learning algorithm that uses multiple decision trees to make predictions which works by generating a large number of decision trees and then combining their predictions. It is useful when dealing with complex data with many input variables.

Support vector machine (SVM) is a popular algorithm used for classification problems which operates by finding a hyperplane in high-dimensional space that maximises the margin between the two classes. It is useful when working with high-dimensional data.

K-nearest neighbors (kNN) is a simple algorithm used for classification and regression problems which works by finding the k closest training examples to a new data point and makes a prediction based on their output values. It is useful when working with small datasets and simple classification problems.

Artificial neural networks (ANN) are algorithms modeled based on the structure and function of the human brain. They can learn complex non-linear relationships between inputs and outputs. This chapter describes two types of ANNs, multilayer perceptrons (MLP) and long short-term memory (LSTM). MLP is useful for problems with multiple inputs and outputs, whereas LSTM is useful for handling sequential data such as time series.

One class classification algorithms are used for outlier detection, where the goal is to identify anomalies in a dataset that do not belong to any known class. One class support vector machine (OCSVM) is a common type of one class classification algorithm that works by fitting a hyperplane that encloses the training data and then classifying new data points as either inside or outside of this hyperplane. Isolation forest (IF) is another OCC algorithm that works by isolating data points in trees based on how easily they can be separated from the rest of the data.

Finally, the choice of algorithm depends on the specific problem and the characteristics of the dataset. Therefore, it is important to understand the strengths and limitations of each algorithm and to experiment with different models to find the one that performs best for a specific problem.

# 4 Dynamic Monitoring Frequency for Energy-Efficient Data Collection in IoT

The goal of the first part of herein presented research [33] aims to build an IoT network that not only reduces the amount of data circulating through the network, but also reduces the amount of energy consumed by sensors. In order to do so, a domain expert must select an optimal equally spaced time period that is short enough to reduce data volume but still long enough to retain requested accuracy. Selecting such a fixed period i.e. *static monitoring frequency (SMF)* is a challenging task. On one hand, if a period is too short, lots of irrelevant and redundant data is collected. On the other hand, if a period is too long, significant changes in an observed signal may be missed. Consequently, due to signal variations over time, a fixed period can be too short on one part of the signal, while being too long on the other part of the same signal. Therefore, the expert has to find a compromise between the volume of collected data and its accuracy.

In this research, we propose to adapt a period to a signal as time passes i.e., *dynamic monitoring frequency (DMF)*. Ideal DMF algorithm should result without irrelevant or redundant data as well as without missed significant changes. In other words, DMF goes one step further by searching for an optimal period in every time step, thus reducing the energy consumption and saving a sensor battery. Consequently, DMF algorithm collects data by an *event*, namely an acceptable change of monitored sensor readings, i.e., sensors are turned on only when the acceptable change defined by domain expert is expected, and sleep in the meanwhile. For instance, an event can be the change in air temperature of ±0.5°C. Therefore, in an ideal case, the next reading should take place only when the temperature is expected to change by 0.5°C.

We use two methods to prepare an algorithm that can predict when an acceptable change in value is expected, namely statistical and machine learning (ML) [4, 47]. On one hand, the statistical algorithm calculates probability of the acceptable change taking place based on data distribution in the past. On the other hand, the ML algorithm compares different ML classification algorithms, namely logistic regression, random forest and artificial neural network that predict future period when the acceptable change is expected.

Our DMFs algorithms are trained using a temperature dataset and later verified on humidity and air pressure datasets. Predictions are made on a temperature change of 0.5°, 0.75°C, 1°C and 1.5°C, humidity change of 5% and pressure change of 0.5hPa. Results of

DMF algorithm are compared to the results of two baseline algorithms with *static monitoring frequency (SMF)* i.e. with a fixed reading period. The first SMF baseline algorithm collects the same amount of data as DMF algorithm, while the second SMF baseline algorithm has the same number of errors as DMF. An error is defined as a percentage of readings that should have been collected due to an occurrence of the acceptable change, but were skipped.

In summary, the contributions of this research include a systematical analysis and process definition for creating DMF algorithms. More specifically:

- We create DMF algorithms using statistical and machine learning approaches, where former reduces the number of readings more than $\sim 40\%$, or number of errors up to $\sim 70\%$, while latter when using random forest algorithm either reduces number of readings by $\sim 30\%$, or number of errors up to $\sim 70\%$.
- We address energy efficiency aspects of our proposed solution showing that $\sim 40\%$ less readings means $\sim 40\%$ longer sensor sleep period. Consequently, sensors using DMF algorithm may save more than 9 hours of battery life every day, compared to sensors that use standard SMF algorithm.
- We demonstrate that significant accuracy can be achieved when algorithms are trained on variable size datasets, i.e., 1-day training dataset has acceptable accuracy for some use cases, while training on 6-month dataset has accuracy comparable to the accuracy of DMF trained on 2-years dataset.
- We compare processing complexity of all DMF algorithms and show that the complexity of statistical DMF has quadratic dependence to a period length, while least complex are linear regression and random forest, which performs better than neural network that exhibits higher complexity.

The rest of this chapter is organized as follows. Introduction is followed by section 4.1, which describes our dataset and the baseline algorithms. Section 4.2 explains our proposed algorithms, while section 4.3 presents results. Section 4.4 discusses energy efficiency, training dataset size and processing complexity of our algorithms. Last section concludes the chapter.

## 4.1 Problem Statement and Datasets

In this section, we formally define the problem statement, describe the baseline algorithm compared with our approach, and present datasets used for the preparation and evaluation of our algorithms.

### 4.1.1 Problem Statement

When IoT sensors monitor an environment, they can read data at any time $t_i$. A moment when data is actually read is defined as $\tau_j$, while time passed between two consecutive readings is defined as a reading period $\Delta\tau_j = \tau_j - \tau_{j-1}$. If $\Delta\tau_j$ is always the same, it is called a static (fixed) period ($period_{SMF} = \Delta\tau_i = \Delta\tau_j, \forall\, i, j \in \mathbb{N}$). A sensor value read at $\tau_j$ is defined as $v(\tau_j)$, while a difference in values between two consecutive readings is denoted as $\Delta v(\tau_j) = abs(v(\tau_j) - v(\tau_{j-1}))$. On one hand, $\Delta v(\tau_j)$ can vary due to changes in an observed phenomenon, e.g., changes in a temperature. On the other hand, it changes as the reading period changes, i.e., by the rule of thumb longer $\Delta\tau_j$ gives higher $\Delta v(\tau_j)$, and *vice versa*.

For most use cases, not all $\Delta v(\tau_j)$ are equally important. Thus, a user wants to ignore every $\Delta v(t_i)$ that is lower than the maximum acceptable difference between two consecutive readings denoted as $\Delta_{max}$. If $\Delta\tau_j$ is too short, $\Delta v(t_i)$ is usually redundant or irrelevant ($\Delta v(\tau_j) << \Delta_{max}$), causing unnecessarily large number of readings (NoR). However, if $\Delta\tau_j$ is too long, $\Delta v(\tau_j)$ may be higher than the maximum acceptable difference ($\Delta v(t_i) > \Delta_{max}$). Consequently, long $\Delta\tau_j$ sometimes results with missed changes higher than $\Delta_{max}$, i.e, it creates large number of errors (NoE). Full terminology is depicted in Figure 4.1 and explained in Abbreviations and Notations table (Table 4.1).

### 4.1.2 Baseline Algorithm

Static monitoring frequency, i.e., having a fixed $\Delta\tau_j$ denoted as $period_{SMF}$, is used as a baseline algorithm. Figure 4.2 shows the relation between the fixed collection $period_{SMF}$ on $x - axis$ and expected differences between consecutive readings (deltas $\Delta v(\tau_j)$) on $y -$

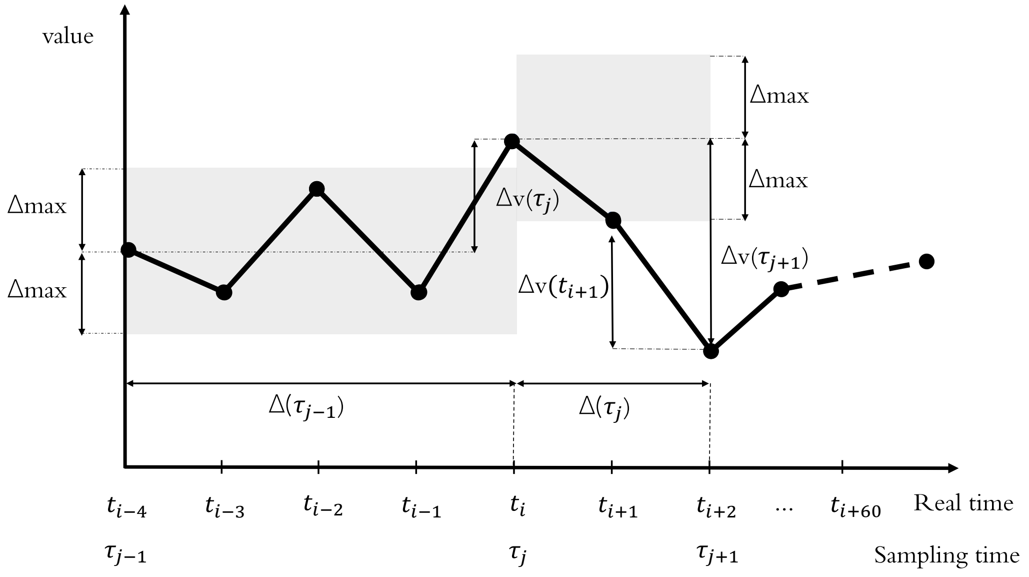


*Figure 4.1. Problem Description*

*Table 4.1. Abbreviations and Notations*

| Label | Description |
|---:|---|
| SMF | Static (uniform) monitoring frequency |
| DMF | Dynamic monitoring frequency |
| S-DMF | Statistical dynamic monitoring frequency |
| ML-DMF | Machine learning dynamic monitoring frequency |
| $\Delta_{max}$ | Maximum acceptable absolute difference between two consecutive readings |
| NoR | Number of Readings |
| NoE | Number of Errors |
| PoR | Percentage of Errors |
| $t_i$ | Discrete time moment when data collection is possible |
| $\tau_j$ | Discrete time moment when data is collected |
| $v(x_i)$ | Value in discrete moment $x_i$ |
| $\Delta v(x_i)$ | Absolute difference in value between two consecutive discrete time moments $x_i$ and $x_{i-1}$; $\Delta v(x_i) = abs(v(x_i) - v(x_{i-1}))$ |
| $target\_accuracy$ | minimal level of accuracy for statistical algorithm [%] |
| $percentile$ | Percentile is calculated on set of values representing the difference between current and last delta value, for defined $target\_accuracy$;<br>$percentile(target\_accuracy, \Delta v(t_i) - \Delta v(t_{i+P}))$, $P \in [1,60]$ |
| $period_{SMF}$ | Static period used in SMF algorithm |

$axis$ that can occur while using that fixed period. Deltas on $y-axis$ represent an upper boundary for 3 different scenarios, namely, 99%, 95% and 90% percentiles. Percentiles define a percentage of occurred deltas ($\Delta v(\tau_j)$) that are under the delta value on $y-axis$ for specific fixed period ($period_{SMF}$). For example, 99% of $\Delta v(\tau_j)$ is under 0.5°C when $period_{SMF} < 8\ minutes$. These calculations can be used for selecting $period_{SMF}$, i.e. NoR for a targeted delta and a desired level of $accuracy$, i.e. NoE.

In order to collect data only when $\Delta_{max}$ is expected, dynamic monitoring frequency (DMF) must be used, i.e., finding an ideal period $\Delta\tau_j$ for every $\tau_j$.This can be done offline by analysing a complete dataset and reading only those $\tau_j$ where $\Delta\tau_j < \Delta_{max}$. This method we denote as $DMF_{oracle}$, which is not feasible in real-time systems as it requires batch processing of the entire dataset, while in real-time systems data arrives by streaming. In order to use DMF in real-time systems where it is important to detect an event, we must predict $\tau_{i+1}$ when $\Delta_{max}$ is going to happen.

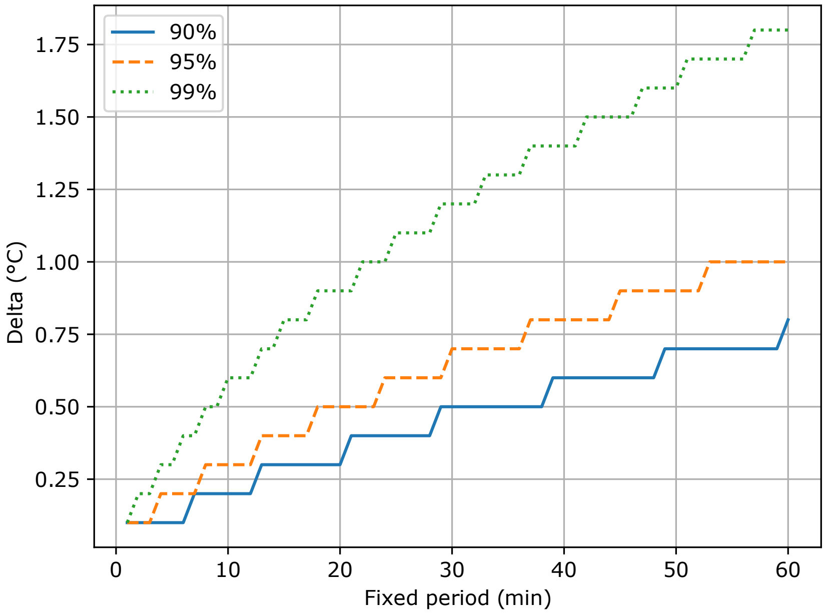


*Figure 4.2. Minimum period for maximum delta*

### 4.1.3 Solution Space

Figure 4.3 shows a single day temperature data and how data is collected with the two algorithms, namely the baseline algorithm that collects data with SMF and $DMF_{oracle}$. Solid blue line in Figure 4.3 represents raw data collected with the baseline algorithm that collects data every minute ($period_{SMF} = 1$). As it is visible from the blue line, there are many similar data points where $\Delta v(t_i) < 0.5$°C. Those data points can be omitted by selecting $\tau_j > t_i$. The dotted green line depicts an ideal data collection with $DMF_{oracle}$ for a single day data, where data is collected only when $\Delta_{max} = 0.5$°C. There are only 37 values collected that day, instead of 1440 actually collected values with the baseline algorithm where $period_{SMF} = 1$.

As DMF algorithm collects data dynamically, the amount of collected data is known only after the data is collected. Consequently, after $DMF_{oracle}$ collects the data, the results are compared with the baseline SMF algorithm that collects the same NoR, as shown with the dashed orange line. The $period_{SMF} = 39\ minutes$ results with $\Delta v(t_i) \in [0.1°C, 1.3°C]$.

### 4.1.4 Datasets

Datasets used in this research are collected every minute during almost 3 years (1/Jan/2017 - 6/Nov/2019), thus they contain 1 490 164 readings per monitored phenomenon. Datasets include weather data, namely temperature, humidity and air pressure. Temperature range is -4.8°C to 36.9°C, humidity ranges between 17.0% and 95.0%, while air pressure dataset has range between 989.89hPa and 1034.39hPa. Figure 4.4a depicts 3-year temperature dataset,

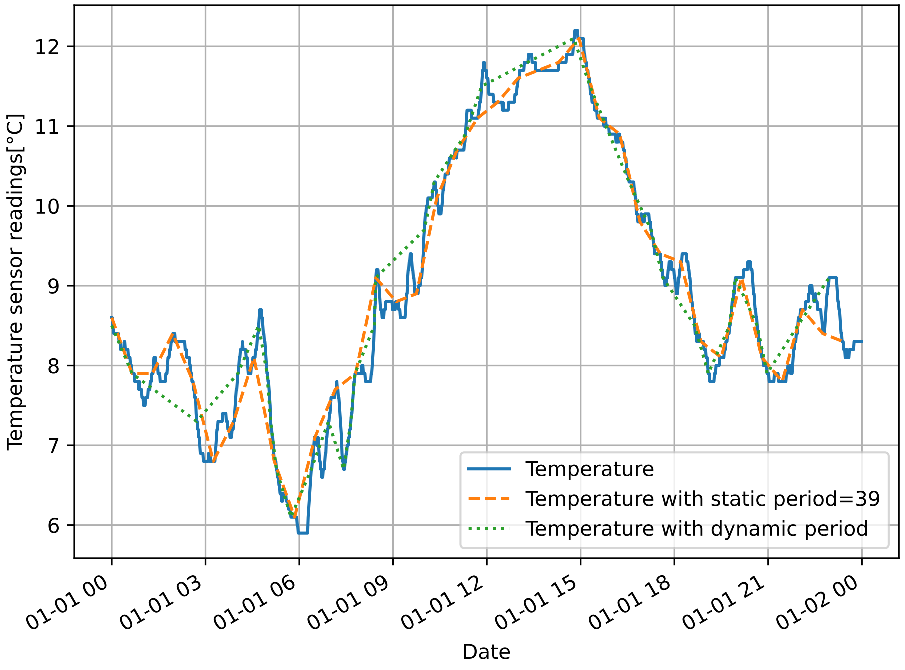


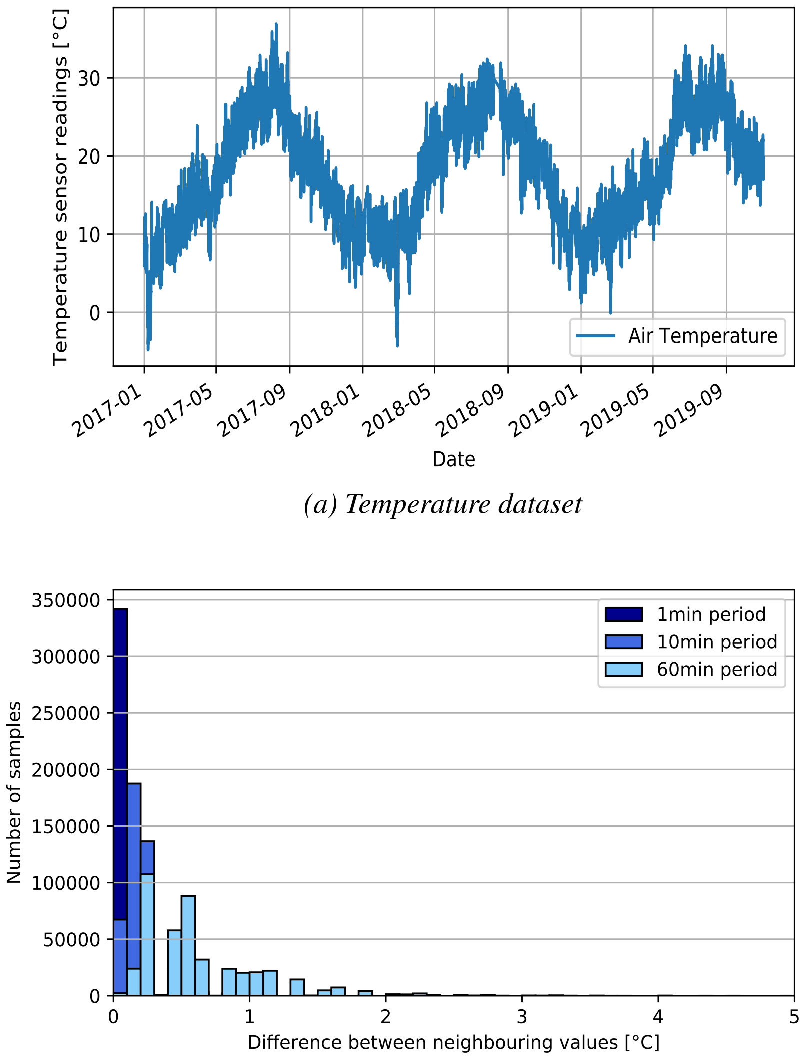
*Figure 4.3. One day data collection with the ideal reading algorithm*

while Figure 4.4b shows the change between neighbouring values ($\Delta v(\tau_i)$) for basic periods $\Delta\tau_i = period_{SMF} = 1, 10, 60$ minutes. As it is visible from the histogram, for $period_{SMF} = 1$ *minute* in most cases $\Delta v(\tau_i) \in [0°C, 0.2°C]$. As $period_{SMF}$ increases to 10 minutes and 60 minutes, $\Delta v(\tau_i)$ increases as well.

As a proof of concept, $DMF_{oracle}$ algorithm is executed on the entire 3-year temperature dataset. For $\Delta_{max} = 0.5°C$, only 21 305 data points are read, instead of total values, with the average reading period of 68 minutes. Larger difference between consecutive readings is tolerable in case of $\Delta_{max} = 1°C$ where only 8 752 data points are read, with an average reading period of around 166 minutes.

## 4.2 Statistical and Machine Learning Algorithms

In order to reduce the amount of collected data, dynamic monitoring frequency (DMF) algorithms are trained. The goal of DMF is to collect data by an event, i.e., only when $\Delta_{max}$ is expected. To achieve this goal, two algorithms are proposed, namely statistical (S-DMF) and machine learning (ML-DMF). Both algorithms are prepared on the temperature dataset and then verified on humidity and air pressure datasets. Data collected during first two years (2017 and 2018) is used as a training set. The algorithms are then evaluated against the last year (2019) dataset.



*(a) Temperature dataset*

*(b) Difference between neighbouring values*

*Figure 4.4. Dataset*

### 4.2.1 Statistical Algorithm (S-DMF)

The S-DMF algorithm works by calculating the distribution of historical data. The algorithm predicts period $\Delta\tau_j$ based on a distribution of change in neighbouring values ($\Delta v(\tau_j)$) for expected $\Delta_{max}$ and *target_accuracy*. The distribution $\Delta v(\tau_j)$ is based on a previous reading period $\Delta\tau_{j-1} = \tau_{j-1} - \tau_{j-2}$ and a previous value $v(\tau_{j-1})$. The example of $\Delta v(\tau_j)$ distribution for $target\_accuracy = 99\%$ and $\Delta_{max} = 1$°C is shown in Figure 4.5. As depicted in the figure, for 1 minute period $\Delta v(\tau_j)$ is rarely above 1°C, thus reading period $\Delta\tau_j$ should be longer.

The Algorithm 1 explains how the S-DMF algorithm works. As shown in the algorithm, the first step is to define $\Delta_{max}$ and expected *target_accuracy*. For expected *target_accuracy*, *percentiles* of $\Delta v(t_{i+P})$ are calculated for every period $P \in [1, 60]$. This step is followed by the calculation of *tolerance* defined as a difference between $\Delta_{max}$ and $\Delta v(\tau_{j-1})$. The

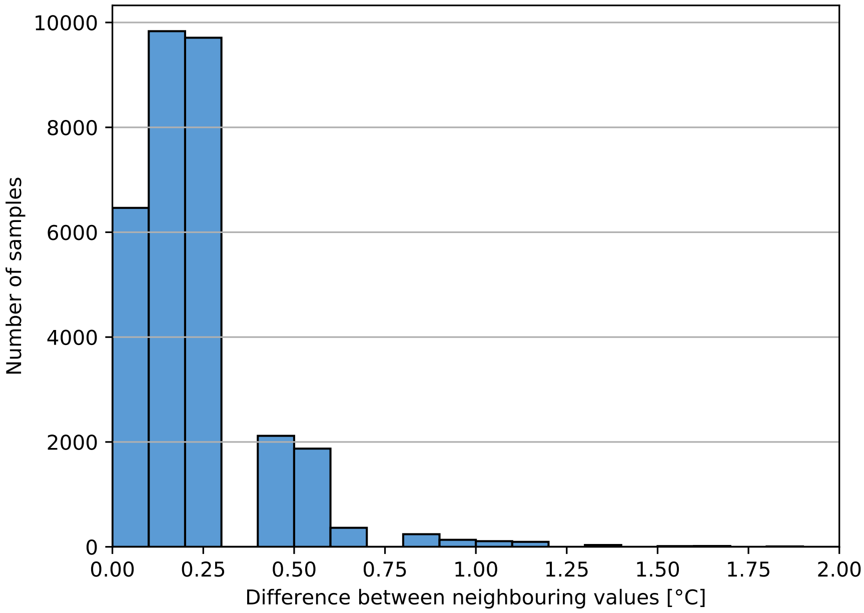


*Figure 4.5. Distribution of differences between neighbouring values for $target\_accuracy = 99\%$, $\Delta_{max} = 1°C$*

**Algorithm 1** S-DMF algorithm

```
**Input:** $\Delta_{max}, target\_accuracy$;
$\Delta v(\tau_j) = abs(v(\tau_j) - v(\tau_{j-1}))$;
$\tau_j = t_i$;
$percentile = table(target\_accuracy, \Delta v(t_{i+P}) - \Delta v(t_i))$,
  $P \in [1, 60]$
$tolerance = \Delta_{max} - \Delta v(\tau_{j-1})$;
**while** $(P \in [1, 60]$ **and** $percentile[t_{i+P}] < tolerance)$ :
  $period = P$;
  $P = P + 1$;
**Output:** $period$;
```

algorithm loops through a dataset and compares $tolerance$ with $percentile$ for the incoming 60 readings in discrete times $t_{i+1}, ..., t_{i+60}$. Loop ends when $percentile[t_{i+P}], P \in [1, 60]$ reaches $tolerance$, hence selecting $P$ as the future read period.

### 4.2.2 Machine Learning Algorithm (ML-DMF)

Since the S-DMF algorithm predicts the future period based on two features, namely the previous period $\Delta\tau_{j-1}$ and the value change $\Delta v(\tau_{j-1})$, the same two features are used to prepare a ML-DMF algorithm. The ML-DMF algorithm is trained by taking $\Delta\tau_{j-1}$ and $\Delta v(\tau_{j-1})$ as ML features to predict future period $\tau_j$ when $\Delta_{max}$ is expected. Three different

*Table 4.2. ML hyper-parameters for 2 features and 4 features (*$max\_features$ is used only for 4 features RF)*

| Algorithm | Parameters |
|---|---|
| Logistic Regression | Default parameters |
| Random Forest | number of trees = 100, 500, 1000<br>*$max\_features = 2, 3, 4$ |
| Artificial Neural Network | batch_size = 128, 512<br>nodes = 100, 300<br>learning rate = 0.01, 0.001, 0.0001 |

ML algorithms are used, namely logistic regression (LR) as linear model, random forest (RF) as tree-based model and artificial neural network (ANN) as non-linear model.

Since modeling ML-DMF is quite different than modeling S-DMF algorithm, i.e., adding additional features to ML model is quite straightforward, ML models with additional features can be easily trained. Additional features usually increase accuracy of the proposed solution, thus two new features are selected, namely a day of the year (1-366) and an hour of the day (0-23). The same three above mentioned ML algorithms, LR, RF and ANN, are used for training additional ML models. A set of hyper-parameters is used to train RF and ANN models, as shown in Table 4.2.

S-DMF algorithm has the additional parameter *target_accuracy*, which represents the trade-off between the accuracy and the amount of data, since more data has to be collected to acquire higher accuracy. Such parameter can not be added to the ML model in a straight-forward manner. Therefore, we use ML algorithms for classification, where we define a set $C$ that holds $n_C = 60$ classes, namely each class $c$ representing a period in range of 1 to 60 minutes $(C = \{1,2,3...59,60\})$. However, instead of taking a single most probable class given by the algorithm, we take a number of considered classes $CC \subset C$ with size $n_{CC}$, and select a class $c$ with the lowest value $(c = min(CC))$, namely the shortest period. Consequently, as $n_{CC}$ increases, the accuracy increases as well due to shorter reading period, while the highest possible period decreases ($period_{max} = 60 - n_{CC}$).

## 4.3 Performance Comparison between Static and Dynamic Monitoring Frequencies

In this section, we compare S-DMF and ML-DMF algorithms with SMF, using temperature, humidity and air pressure datasets. Subsection 4.3.1 compares S-DMF with SMF, while in subsection 4.3.2 we select representative algorithms of each of the three categories of

ML approaches and compare them with SMF. Last subsection 4.3.3 compares S-DMF, three ML-DMFs and SMF algorithms.

When comparing results, it is important to understand the compromises between the amount of collected data, namely the number of readings (NoR) and the number of errors (NoE). Decrease in NoE, results with higher NoR and *vice versa*. On one hand, for systems where the longest possible battery life is needed, i.e. non-critical systems such as smart agriculture systems, where there are a lots of sensors in large fields or inaccessible locations, higher NoE can be considered. On the other hand, sensitive systems that need the highest accuracy, i.e. critical systems such as elderly care systems, where missed information can create significant loss (of life), should utilize higher NoR. When comparing DMF algorithms and SMF algorithm, the above mentioned metrics are used. Each DMF algorithm is compared with two instances of the SMF algorithm. First SMF instance collects the same amount of data as DMF, i.e., NoR is the same, while other SMF instance is the one that has the same NoE as DMF algorithm.

### 4.3.1 Statistical Algorithm (S-DMF) Performance

Results for the temperature dataset with $\Delta_{max} = 0.5$°C are depicted in Figure 4.6. SMF curve contains data with collection period of 1 to 60 minutes. S-DMF contains NoR and NoE for multiple $target\_accuracy = \{10\%, ..., 99.9\%\}$. Logarithmic scale in Figure 4.6 is used to emphasize that SMF works slightly better when higher NoE is tolerated, while S-DMF algorithm introduces less errors in case of lower tolerance to errors. Table 4.3a and Figure 4.7 show the results for S-DMF algorithm for three *target_accuracy*, namely 95%, 99% and 99.9%. S-DMF algorithm guarantees that the percentage of errors will be below selected *target_accuracy*, e.g. 99%. As expected, higher *target_accuracy* results with lower NoE, while more data is collected. Results for maximum acceptable delta $\Delta_{max}$ of 0.5°C, 0.75°C, 1°C and 1.5°C are shown in Table 4.3a. The lower $\Delta_{max}$ is, the algorithm collects more data (i.e., higher NoR) with the shorter average collection period.

In case of the S-DMF algorithm with $target\_accuracy = 99\%$ for $\Delta_{max} = 1$°C, the algorithm collects 31 033 data points ($\sim$ 7% of total 438 963 data points). Moreover, it introduces 609 errors ($\sim 0.14\%$). As S-DMF collects data with an average period of 14.14 minutes, the results are compared to the results of SMF with $period_{SMF}$ of 14 minutes. In case of $period_{SMF} = 14$, 31 355 data points are collected, while introducing 1 093 errors. That said, our S-DMF has $\sim$ 45% less errors compared to SMF algorithm that collects the same amount of data. Period of SMF algorithm with the same NoE is 12 minutes. In case of $period_{SMF} = 12$, 36 581 data points are collected, while introducing 726 errors. However, S-DMF collects $\sim$ 15% less data compared to SMF. That being said, S-DMF results either with lower $NoE$ when $NoR$ is the same, or with lower $NoR$ when $NoE$ is the same, compared to SMF.

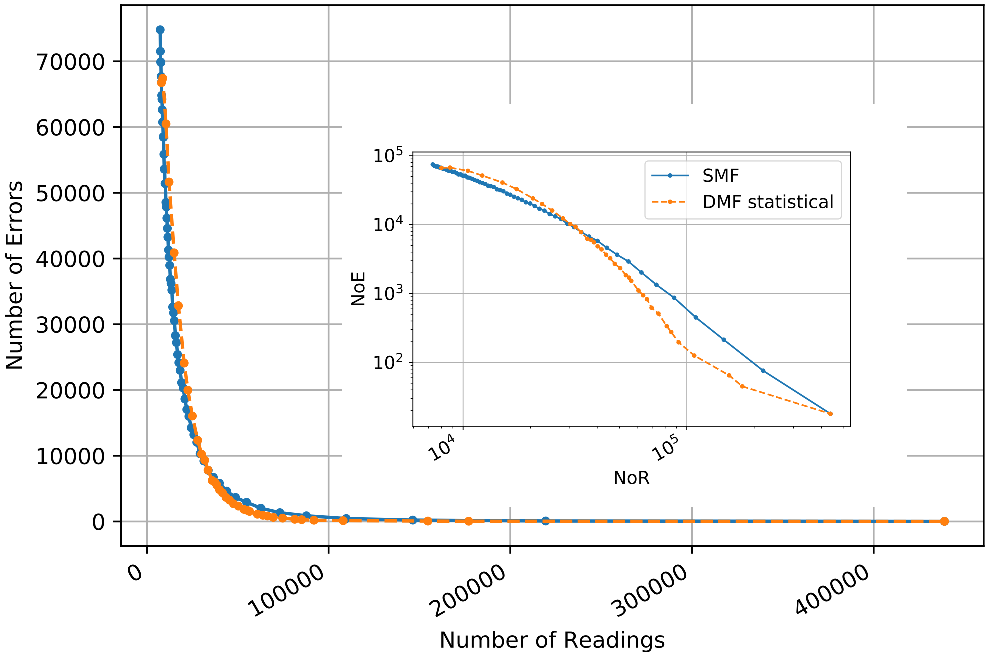


*Figure 4.6. Results of S-DMF algorithm compared to SMF for temperature dataset with $\Delta_{max} = 0.5°C$;*
*Inner image depicts results on logarithmic scale*

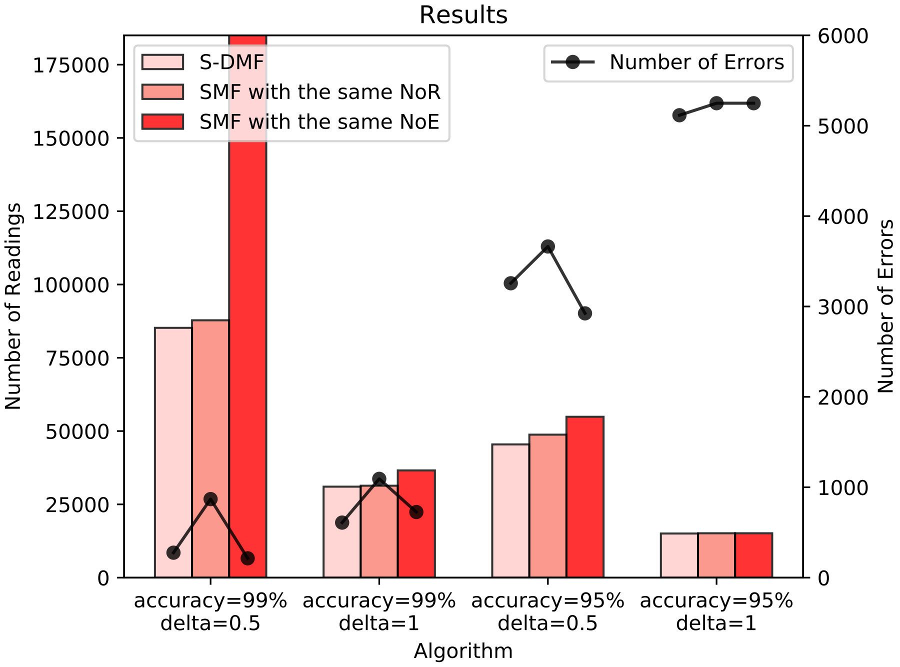


*Figure 4.7. Results of S-DMF algorithm for temperature dataset*

*Table 4.3. Results of S-DMF algorithm*

a) Results of S-DMF algorithm - temperature

| Algorithm | $\Delta_{max}$ | *target_accuracy* | Avg Period DMF | SMF Period | NoR[3] DMF[1] | NoR SMF[2] | NoE[4] DMF | NoE SMF | PoE[5] DMF | PoE SMF |
|---|---|---|---|---|---|---|---|---|---|---|
| S-DMF | 0.5 | 99.9% | 2.478168823 | 3 | 177132 | 146322 | 45 | 214 | 0.01% | 0.05% |
| | | | | 2 | 177132 | 219482 | 45 | 76 | 0.01% | 0.02% |
| | | 99% | 5.151664163 | 5 | 85208 | 87793 | 276 | 869 | 0.06% | 0.20% |
| | | | | 3 | 85208 | 146322 | 276 | 214 | 0.06% | 0.05% |
| | | 95% | 9.657939319 | 9 | 45451 | 48774 | 3257 | 3665 | 0.74% | 0.83% |
| | | | | 8 | 45451 | 54871 | 3257 | 2924 | 0.74% | 0.67% |
| | 0.75 | 99.9% | 4.258510463 | 4 | 103079 | 109741 | 27 | 157 | 0.01% | 0.04% |
| | | | | 2 | 103079 | 219482 | 27 | 26 | 0.01% | 0.01% |
| | | 99% | 10.02954281 | 10 | 43767 | 43897 | 916 | 1701 | 0.21% | 0.39% |
| | | | | 8 | 43767 | 54871 | 916 | 978 | 0.21% | 0.22% |
| | | 95% | 20.49218057 | 20 | 21421 | 21949 | 7702 | 7403 | 1.75% | 1.69% |
| | 1 | 99.9% | 5.897582996 | 6 | 74431 | 73161 | 29 | 128 | 0.01% | 0.03% |
| | | | | 3 | 74431 | 146322 | 29 | 21 | 0.01% | 0.00% |
| | | 99% | 14.14503915 | 14 | 31033 | 31355 | 609 | 1093 | 0.14% | 0.25% |
| | | | | 12 | 31033 | 36581 | 609 | 726 | 0.14% | 0.17% |
| | | 95% | 29.15341702 | 29 | 15057 | 15137 | 5116 | 5249 | 1.17% | 1.20% |
| | 1.5 | 99.9% | 9.676248209 | 10 | 45365 | 43897 | 40 | 114 | 0.01% | 0.03% |
| | | | | 7 | 45365 | 62710 | 40 | 38 | 0.01% | 0.01% |
| | | 99% | 26.82164243 | 26 | 16366 | 16884 | 845 | 1171 | 0.19% | 0.27% |
| | | | | 24 | 16366 | 18291 | 845 | 863 | 0.19% | 0.20% |
| | | 95% | 49.58353101 | 50 | 8853 | 8780 | 3740 | 3909 | 0.85% | 0.89% |

b) Results of S-DMF algorithm - humidity

| Algorithm | $\Delta_{max}$ | *target_accuracy* | Avg Period DMF | SMF Period | NoR[3] DMF[1] | NoR SMF[2] | NoE[4] DMF | NoE SMF | PoE[5] DMF | PoE SMF |
|---|---|---|---|---|---|---|---|---|---|---|
| S-DMF | 5% | 99.9% | 1 | 1 | 438963 | 438963 | 184 | 184 | 0.04% | 0.04% |
| | 5% | 99% | 4.015872725 | 4 | 109307 | 109741 | 417 | 1238 | 0.09% | 0.28% |
| | | | | 2 | 109307 | 219482 | 417 | 482 | 0.09% | 0.11% |
| | 5% | 95% | 11.59590543 | 12 | 37855 | 36581 | 4165 | 5713 | 0.95% | 1.30% |
| | | | | 10 | 37855 | 43897 | 4165 | 4246 | 0.95% | 0.97% |

c) Results of S-DMF algorithm - air pressure

| Algorithm | $\Delta_{max}$ | *target_accuracy* | Avg Period DMF | SMF Period | NoR[3] DMF[1] | NoR SMF[2] | NoE[4] DMF | NoE SMF | PoE[5] DMF | PoE SMF |
|---|---|---|---|---|---|---|---|---|---|---|
| S-DMF | 0.5 | 99.9% | 1.764637 | 2 | 248756 | 219482 | 432 | 457 | 0.10% | 0.10% |
| | 0.5 | 99% | 7.907694 | 8 | 55511 | 54871 | 818 | 1042 | 0.19% | 0.24% |
| | | | | 6 | 55511 | 73161 | 818 | 848 | 0.19% | 0.19% |
| | 0.5 | 95% | 18.91679 | 19 | 23205 | 23104 | 2201 | 2527 | 0.50% | 0.58% |
| | | | | 18 | 23205 | 24387 | 2201 | 2183 | 0.50% | 0.50% |

[1] Dynamic monitoring frequency [2] Static monitoring frequency [3] Number of Readings [4] Number of Errors [5] Percentage of Errors

The results for the S-DMF algorithm for humidity are shown in Table 4.3b. Results for air pressure are shown in Table 4.3c. As it is visible from tables 4.3b and 4.3c, all conclusions discussed for temperature are confirmed on both humidity and air pressure datasets.

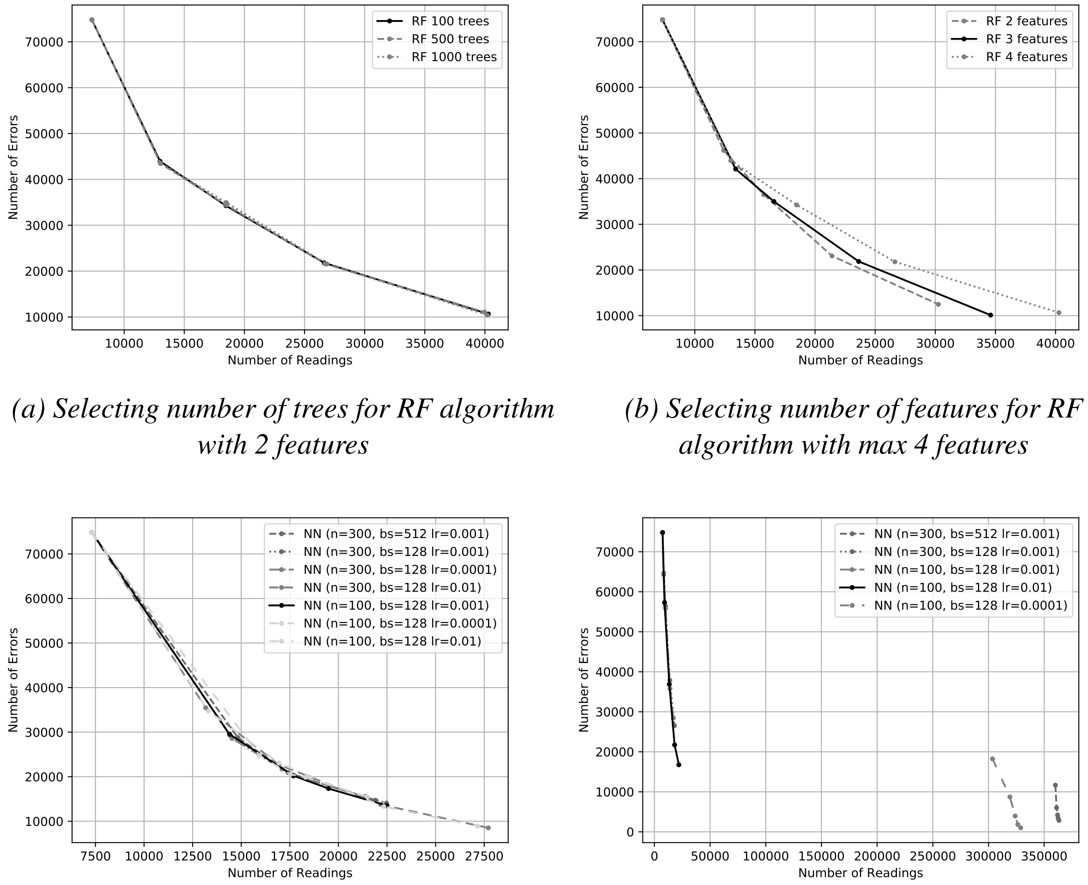
### 4.3.2 Machine Learning Algorithm (ML-DMF) Performance

Three ML algorithms, namely logistic regression, random forest and artificial neural network, are used and their results are discussed in this section. ML is trained on two sets of features. The first set of features is the same set used for the S-DMF algorithm, i.e., last period $\Delta\tau_j$ and last reading $v(\tau_{j-1})$ are used as inputs, while algorithm predicts future period when $\Delta_{max}$ is expected. Additional set of features contains information about a day of the year and an hour of the day. Thus, results are divided into results for algorithms trained on 2 features and algorithms trained on 4 features.

In order to find the best ML algorithm with 2 features, a representative for each of the three ML algorithm categories is selected:

- Logistic regression - default hyper-parameters are used.
- Random forest - algorithms with different number of trees/estimators=100,500,1000 are considered, but there is no significant difference as seen in Figure 4.8a, thus the simplest algorithm with 100 trees is selected.
- Artificial neural network - different combinations of batch size, number of nodes and learning rates are considered. Results for all executed ANNs are shown in Figure 4.8c. ANN with $batch\_size = 128$, $nodes = 100$, $learning\_rate = 0.001$ is selected as a representative since the results do not differ significantly and it is the simplest one.

Table 4.4 shows results of the best ML algorithms in their categories. The results of ML-DMF algorithms are calculated for three $n_{CC} = 1, 5, 10$, as it is shown in Table 4.4.



*(a) Selecting number of trees for RF algorithm with 2 features*

*(b) Selecting number of features for RF algorithm with max 4 features*

*(c) Selecting the best ANN algorithm on dataset with 2 fetaures*
$n = nodes, bs = batch\_size, lr = learning\_rate$

*(d) Selecting the best ANN algorithm on dataset with 4 features*
$n = nodes, bs = batch\_size, lr = learning\_rate$

*Figure 4.8. Selecting the best parameters of ML algorithms ($\Delta_{max} = 0.5°C$)*

For temperature dataset (Table 4.4a), maximum deltas $\Delta_{max}$ of 0.5°C, 0.75°C, 1°C and 1.5°C are considered.

For example, in case of an ML algorithm with $n_{CC} = 10$ for $\Delta_{max} = 1$°C, RF ML-DMF algorithm collects 9 774 data points ($\sim$ 22%) with $NoE = 9\ 672$ ($\sim$ 2.2% ). As ML-DMF collects data with an average period of 44.9 minutes, the results are compared to the results of SMF with static period of 45 minutes. In case of $period_{SMF} = 45$, 9 755 data points are collected, with $NoE = 11\ 261$. That said, SMF that collects the same amount of data introduces $\sim$ 2.6% errors. The period of the SMF algorithm with the same NoE is 41 minutes. In case of $period_{SMF} = 41$, 10 707 data points are collected, with $NoE = 9\ 350$. Therefore, NoE is similar, but more data is collected with SMF. Similar to S-DMF, ML-DMF either introduces less errors (when the same amount of data is collected), or collects less data (when the same number of errors is introduced) compared to SMF.

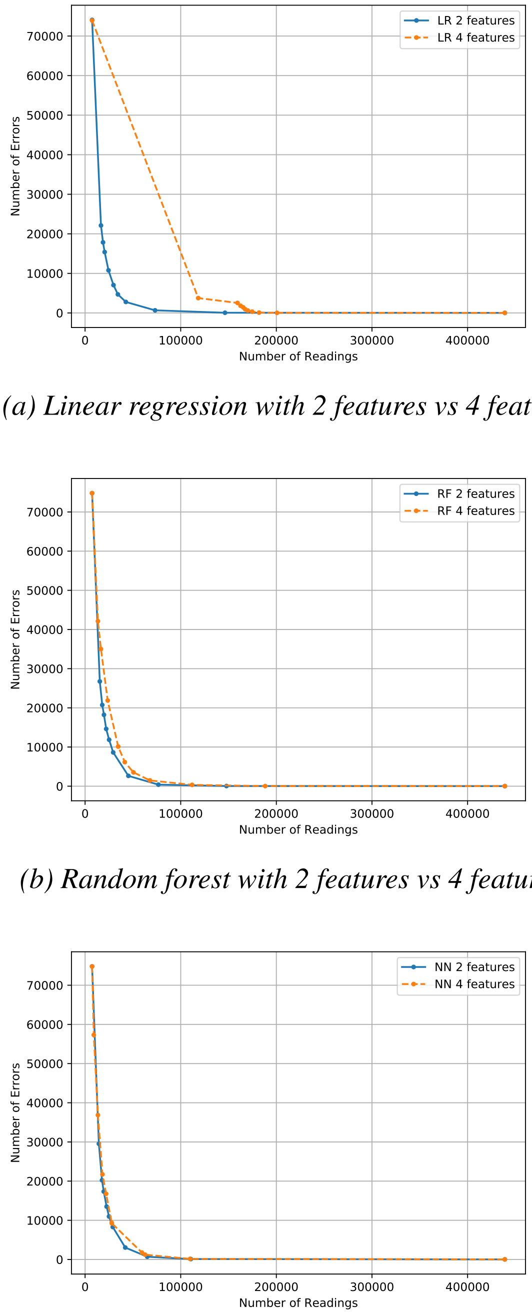
In order to find representative ML algorithms with 4 features, following algorithms are selected:

- Logistic regression - only default hyper-parameters are used.
- Random forest - algorithms with different number of trees/estimators=100,500,1000 are considered as well as algorithms with different number of $max\_features$=2,3,4. Increase of number of trees does not improve results (as shown in RF with 2 features), but algorithm provides the best results when $max\_features$ parameter is set to 3. Therefore, RF with 100 tress and $max\_features = 3$ is selected (Figure 4.8b).
- Artificial neural network - different combinations of hyper-parameters are tested. Results of all executed ANNs are shown in Figure 4.8d and ANN with $batch\_size = 128$, $nodes = 100$ and $learning\_rate = 0.001$ is selected as a representative since it gives slightly better results than the others.

Overall results of the algorithms with 4 features are not as good as the results of the algorithms with only two features, as depicted in Figure 4.9. The similar behaviour, when selecting one or two features gives near optimum results, is documented in [114].

Thus, further analysis is focused on ML-DMFs with 2 features. Comparing LR, representative RF and representative ANN, results are similar as shown on the logarithmic scale in Figure 4.10. Complex algorithm such as ANN does not give much better results than random forest. Furthermore, DMF (S-DMF and ML-DMF) algorithms give better results than basic SMF.

The best hyper-parameters of temperature ML-DMF algorithms are applied on humidity and air pressure datasets and results are consistent, as shown in Table 4.4b (humidity) and 4.4c (air pressure).



*(a) Linear regression with 2 features vs 4 features*

*(b) Random forest with 2 features vs 4 features*

*(c) Artificial neural network with 2 features vs 4 features*

*Figure 4.9. Comparison of results of ML algorithms with 2 and 4 features ($\Delta_{max} = 0.5°C$)*

### 4.3.3 Comparison of DMF vs SMF Algorithms

As demonstrated in the previous section, DMF can be implemented using either S-DMF or ML-DMF. ML-DMF with default parameters collects less data resulting with more errors, compared to S-DMF which by default decreases number of errors while collecting more data. However, amount of collected data can be controlled either by adjusting *target_accuracy* parameter for S-DMF or $n_{CC} - number\ of\ considered\ classes$ for ML-DMF. As it is shown in Figure 4.11, in order to achieve *NoE* as low as 276, S-DMF algorithm uses

*Table 4.4. Results of ML-DMF algorithms*

| | | | a) Results of ML-DMF algorithm - temperature | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Algorithm | $\Delta_{max}$ | Number of classes (n) | Avg Period DMF | SMF Period | NoR$^3$ DMF$^1$ | NoR SMF$^2$ | NoE$^4$ DMF | NoE SMF | PoE5 DMF | PoE SMF |
| Logistic Regression | 0.5 | 1 | 59.83683206 | 60 | 7336 | 7317 | 74085 | 74799 | 16.88% | 17.04% |
| | | 5 | 23.39763339 | 23 | 18761 | 19086 | 17848 | 21129 | 4.07% | 4.81% |
| | | | | 20 | 18761 | 21949 | 17848 | 17033 | 4.07% | 3.88% |
| | | 10 | 21.44002149 | 21 | 20474 | 20904 | 15397 | 18637 | 3.51% | 4.25% |
| | | | | 19 | 20474 | 23104 | 15397 | 15984 | 3.51% | 3.64% |
| | 0.75 | 1 | 59.95124283 | 60 | 7322 | 7317 | 42279 | 42441 | 9.63% | 9.67% |
| | | 5 | 42.17959066 | 42 | 10407 | 10452 | 22353 | 25585 | 5.09% | 5.83% |
| | | | | 38 | 10407 | 11552 | 22353 | 21935 | 5.09% | 5.00% |
| | | 10 | 39.34065245 | 39 | 11158 | 11256 | 20480 | 22846 | 4.67% | 5.20% |
| | | | | 37 | 11158 | 11864 | 20480 | 20711 | 4.67% | 4.72% |
| | 1 | 1 | 60 | 60 | 7317 | 7317 | 18297 | 18297 | 4.17% | 4.17% |
| | | 5 | 54.50248324 | 54 | 8054 | 8129 | 15367 | 15341 | 3.50% | 3.49% |
| | | 10 | 50.28789094 | 50 | 8729 | 8780 | 12805 | 13998 | 2.92% | 3.19% |
| | | | | 48 | 8729 | 9146 | 12805 | 12985 | 2.92% | 2.96% |
| | 1.5 | 1 | 60 | 60 | 7317 | 7317 | 5524 | 5524 | 1.26% | 1.26% |
| | | 5 | 55.05618964 | 55 | 7973 | 7982 | 4617 | 4454 | 1.05% | 1.01% |
| | | | | 54 | 7973 | 8129 | 4617 | 4634 | 1.05% | 1.06% |
| | | 10 | 49.39383369 | 49 | 8887 | 8959 | 3473 | 3773 | 0.79% | 0.86% |
| | | | | 46 | 8887 | 9543 | 3473 | 3390 | 0.79% | 0.77% |
| Random Forest | 0.5 | 1 | 60 | 60 | 7317 | 7317 | 74799 | 74799 | 17.04% | 17.04% |
| | | 5 | 24.45476323 | 24 | 17950 | 18291 | 20732 | 22982 | 4.72% | 5.24% |
| | | | | 22 | 17950 | 19953 | 20732 | 20308 | 4.72% | 4.63% |
| | | 10 | 22.23836061 | 22 | 19739 | 19953 | 18222 | 20308 | 4.15% | 4.63% |
| | | | | 21 | 19739 | 20904 | 18222 | 18637 | 4.15% | 4.25% |
| | 0.75 | 1 | 60 | 60 | 7317 | 7317 | 42441 | 42441 | 9.67% | 9.67% |
| | | 5 | 39.64264427 | 40 | 11073 | 10975 | 22057 | 23478 | 5.02% | 5.35% |
| | | | | 38 | 11073 | 11552 | 22057 | 21935 | 5.02% | 5.00% |
| | | 10 | 36.38317447 | 36 | 12065 | 12194 | 19446 | 19704 | 4.43% | 4.49% |
| | | | | 35 | 12065 | 12542 | 19446 | 19472 | 4.43% | 4.44% |
| | 1 | 1 | 60 | 60 | 7317 | 7317 | 18297 | 18297 | 4.17% | 4.17% |
| | | 5 | 48.63855956 | 49 | 9025 | 8959 | 11399 | 13298 | 2.60% | 3.03% |
| | | | | 45 | 9025 | 9755 | 11399 | 11261 | 2.60% | 2.57% |
| | | 10 | 44.91129527 | 45 | 9774 | 9755 | 9672 | 11261 | 2.20% | 2.57% |
| | | | | 41 | 9774 | 10707 | 9672 | 9350 | 2.20% | 2.13% |
| | 1.5 | 1 | 60 | 60 | 7317 | 7317 | 5524 | 5524 | 1.26% | 1.26% |
| | | 5 | 49.69579984 | 50 | 8833 | 8780 | 3730 | 3909 | 0.85% | 0.89% |
| | | | | 48 | 8833 | 9146 | 3730 | 3764 | 0.85% | 0.86% |
| | | 10 | 45.017229 | 45 | 9751 | 9755 | 2943 | 3224 | 0.67% | 0.73% |
| | | | | 42 | 9751 | 10452 | 2943 | 2944 | 0.67% | 0.67% |
| Neural Networks | 0.5 | 1 | 60 | 60 | 7317 | 7317 | 74799 | 74799 | 17.04% | 17.04% |
| | | 5 | 24.81418881 | 25 | 17690 | 17559 | 20222 | 24161 | 4.61% | 5.50% |
| | | | | 22 | 17690 | 19953 | 20222 | 20308 | 4.61% | 4.63% |
| | | 10 | 22.52247306 | 23 | 19490 | 19086 | 17342 | 21129 | 3.95% | 4.81% |
| | | | | 20 | 19490 | 21949 | 17342 | 17033 | 3.95% | 3.88% |
| | 0.75 | 1 | 60 | 60 | 7317 | 7317 | 42441 | 42441 | 9.67% | 9.67% |
| | | 5 | 39.58901515 | 40 | 11088 | 10975 | 21284 | 23478 | 4.85% | 5.35% |
| | | | | 37 | 11088 | 11864 | 21284 | 20711 | 4.85% | 4.72% |
| | | 10 | 36.38920667 | 36 | 12063 | 12194 | 18050 | 19704 | 4.11% | 4.49% |
| | | | | 33 | 12063 | 13302 | 18050 | 17920 | 4.11% | 4.08% |
| | 1 | 1 | 60 | 60 | 7317 | 7317 | 18297 | 18297 | 4.17% | 4.17% |
| | | 5 | 52.82982308 | 53 | 8309 | 8283 | 13303 | 15070 | 3.03% | 3.43% |
| | | | | 49 | 8309 | 8959 | 13303 | 13298 | 3.03% | 3.03% |
| | | 10 | 46.47570143 | 46 | 9445 | 9543 | 11376 | 11800 | 2.59% | 2.69% |
| | | | | 45 | 9445 | 9755 | 11376 | 11261 | 2.59% | 2.57% |
| | 1.5 | 1 | 60 | 60 | 7317 | 7317 | 5524 | 5524 | 1.26% | 1.26% |
| | | 5 | 52.00983412 | 52 | 8440 | 8442 | 3685 | 4260 | 0.84% | 0.97% |
| | | | | 47 | 8440 | 9340 | 3685 | 3639 | 0.84% | 0.83% |
| | | 10 | 47.0132805 | 47 | 9337 | 9340 | 2856 | 3639 | 0.65% | 0.83% |
| | | | | 43 | 9337 | 10209 | 2856 | 2918 | 0.65% | 0.66% |

b) Results of ML-DMF algorithms - humidity

| Algorithm | $\Delta_{max}$ | Number of classes (n) | Avg Period DMF | SMF Period | NoR[3] DMF[1] | NoR SMF[2] | NoE[4] DMF | NoE SMF | PoE5 DMF | PoE SMF |
|---|---|---|---|---|---|---|---|---|---|---|
| Logistic Regression | 5 | 1 | 59.8205233 | 60 | 7338 | 7317 | 37004 | 37826 | 8.43% | 8.62% |
| | | 5 | 21.34826379 | 21 | 20562 | 20903 | 9630 | 11735 | 2.19% | 2.67% |
| | | | | 18 | 20562 | 24387 | 9630 | 9661 | 2.19% | 2.20% |
| | | 10 | 17.52067534 | 18 | 25054 | 24387 | 6477 | 9661 | 1.48% | 2.20% |
| | | | | 13 | 25054 | 33767 | 6477 | 6366 | 1.48% | 1.45% |
| Random Forest | 5 | 1 | 60 | 60 | 7317 | 7317 | 37826 | 37826 | 8.62% | 8.62% |
| | | 5 | 25.92505315 | 26 | 16932 | 16884 | 14450 | 15391 | 3.29% | 3.51% |
| | | | | 24 | 16932 | 18291 | 14450 | 14204 | 3.29% | 3.24% |
| | | 10 | 22.74184022 | 23 | 19302 | 19086 | 12205 | 13439 | 2.78% | 3.06% |
| | | | | 22 | 19302 | 19953 | 12205 | 12149 | 2.78% | 2.77% |
| Neural Networks | 5 | 1 | 60 | 60 | 7317 | 7317 | 37826 | 37826 | 8.62% | 8.62% |
| | | 5 | 25.57463295 | 26 | 17164 | 16884 | 14093 | 15391 | 3.21% | 3.51% |
| | | | | 24 | 17164 | 18291 | 14093 | 14204 | 3.21% | 3.24% |
| | | 10 | 21.52834723 | 22 | 20390 | 19953 | 10831 | 12149 | 2.47% | 2.77% |
| | | | | 20 | 20390 | 21949 | 10831 | 10960 | 2.47% | 2.50% |

c) Results of ML-DMF algorithms - air pressure

| Algorithm | $\Delta_{max}$ | Number of classes (n) | Avg Period DMF | SMF Period | NoR[3] DMF[1] | NoR SMF[2] | NoE[4] DMF | NoE SMF | PoE5 DMF | PoE SMF |
|---|---|---|---|---|---|---|---|---|---|---|
| Logistic Regression | 0.5 | 1 | 60 | 60 | 7317 | 7317 | 20847 | 20847 | 4.75% | 4.75% |
| | | 5 | 53.99298893 | 54 | 8130 | 8129 | 16064 | 16653 | 3.66% | 3.79% |
| | | | | 53 | 8130 | 8283 | 16064 | 15574 | 3.66% | 3.55% |
| | | 10 | 49.97301913 | 50 | 8784 | 8780 | 13589 | 14287 | 3.10% | 3.25% |
| | | | | 49 | 8784 | 8959 | 13589 | 13445 | 3.10% | 3.06% |
| Random Forest | 0.5 | 1 | 60 | 60 | 7317 | 7317 | 20847 | 20847 | 4.75% | 4.75% |
| | | 5 | 49.96733068 | 50 | 8785 | 8780 | 12367 | 14287 | 2.82% | 3.25% |
| | | | | 47 | 8785 | 9340 | 12367 | 12554 | 2.82% | 2.86% |
| | | 10 | 46.60894033 | 47 | 9418 | 9340 | 10264 | 12554 | 2.34% | 2.86% |
| | | | | 43 | 9418 | 10209 | 10264 | 10425 | 2.34% | 2.37% |
| Neural Networks | 0.5 | 1 | 60 | 60 | 7317 | 7317 | 20847 | 20847 | 4.75% | 4.75% |
| | | 5 | 51.44298605 | 51 | 8533 | 8608 | 13299 | 14994 | 3.03% | 3.42% |
| | | | | 49 | 8533 | 8959 | 13299 | 13445 | 3.03% | 3.06% |
| | | 10 | 43.8130552 | 44 | 10019 | 9977 | 9456 | 10852 | 2.15% | 2.47% |
| | | | | 41 | 10019 | 10707 | 9456 | 9397 | 2.15% | 2.14% |

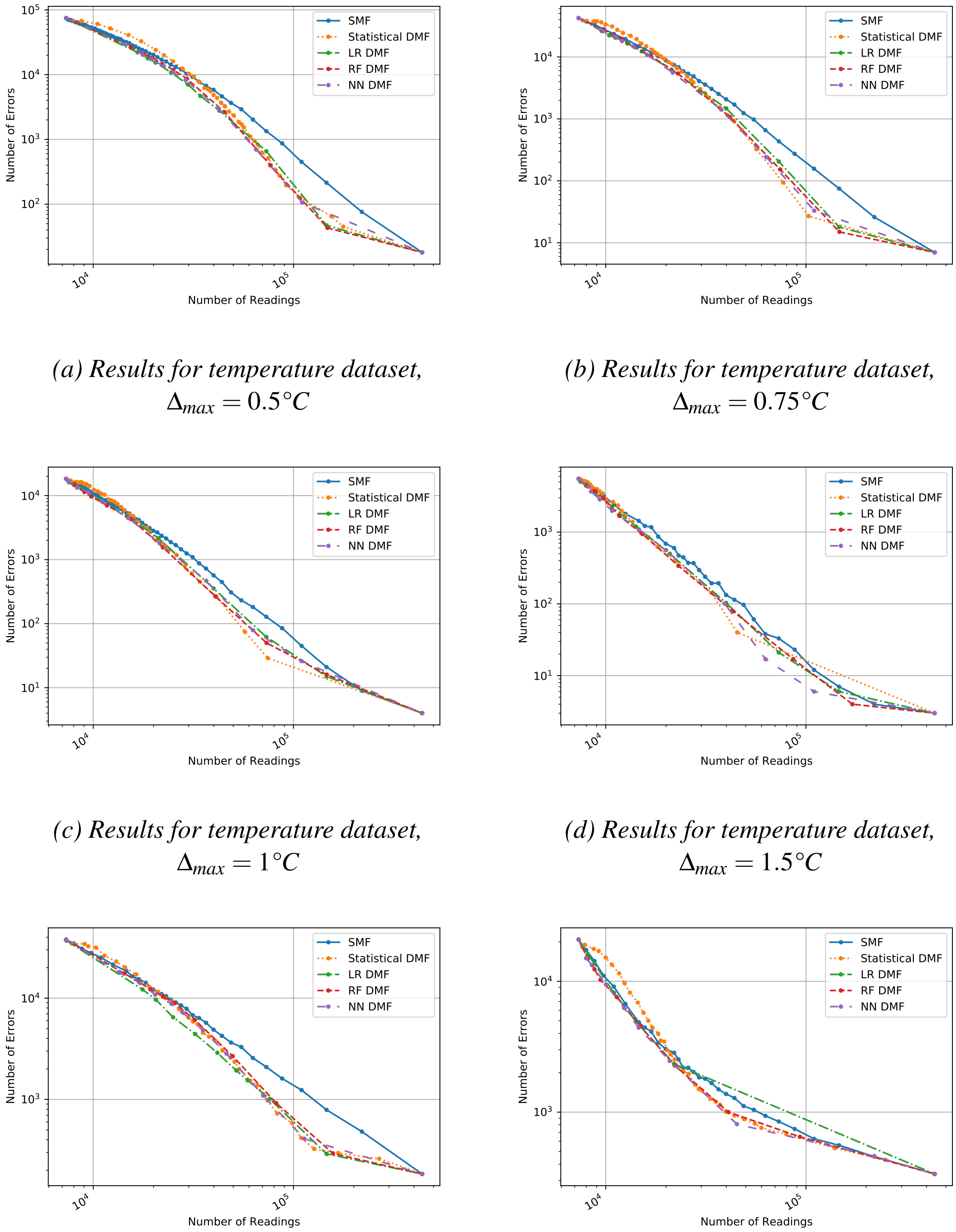
[1] Dynamic monitoring frequency [2] Static monitoring frequency [3] Number of Readings [4] Number of Errors [5] Percentage of Errors

$target\_accuracy = 99\%$. ML algorithm with similar $NoE$ has to use $n_{cc} = 55$. The problem with $n_{cc} = 55$ is that the highest period such algorithm can select is 5 minutes ($period_{max} = 60 - n_{cc} = 5$). In comparison, S-DMF algorithm can select any period in range of 1-60 depending on parameters. Discussed S-DMF with $target\_accuracy = 99\%$ can reach periods up to 22 minutes, but $period_{max} = 22$ is known only after algorithm is executed.

To summarize, both S-DMF and ML-DMF give better results than traditional SMF (Figure 4.10). ML-DMF works better when the emphasis is on lowering NoR, while S-DMF provides better results when tolerance to errors is low.

## 4.4 Implications on Energy Efficiency and Processing Complexity

It is shown that statistical dynamic monitoring frequency (S-DMF) as well as machine learning DMF (ML-DMF) can successfully find dynamic periods. In this section, we first show how DMFs may reduce energy consumption while preserving accuracy. Secondly, the impact of the training dataset size to the algorithms accuracy is analysed. Lastly, we discuss processing complexity of S-DMF and ML-DMF algorithms.



*(a) Results for temperature dataset, $\Delta_{max} = 0.5°C$*

*(b) Results for temperature dataset, $\Delta_{max} = 0.75°C$*

*(c) Results for temperature dataset, $\Delta_{max} = 1°C$*

*(d) Results for temperature dataset, $\Delta_{max} = 1.5°C$*

*(e) Results for humidity dataset, $\Delta_{max} = 5\%$*

*(f) Results for air pressure dataset, $\Delta_{max} = 0.5hPa$*

*Figure 4.10. Overall results*

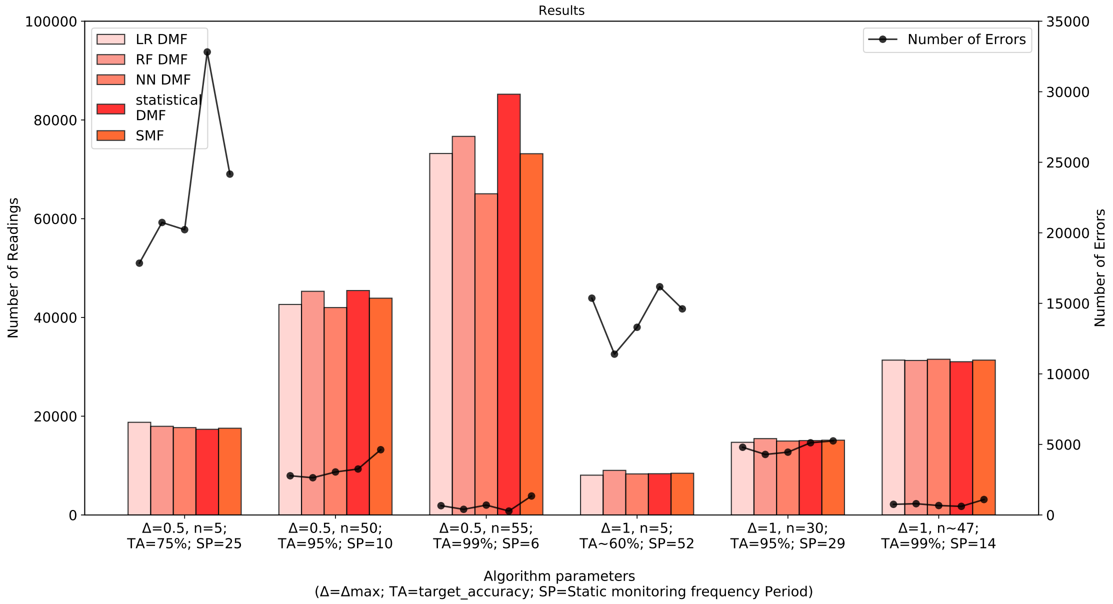


*Figure 4.11. Comparison of S-DMF and ML-DMF results with SMF*

### 4.4.1 Energy Efficiency

In given use cases, when monitoring temperature, humidity and pressure, existing solution collects data every minute. Since monitored phenomena may not change that fast, dataset can be full of redundant data. Temperature dataset used in this research consists of 1 451 120 data points. Since it contains no difference between consecutive values, i.e, $\Delta v(t_i) = 0$ for 1 128 297 readings, $\sim 22\%$ of collected values are redundant. In terms of an air temperature considered for this use case, the change of 0.5°C or 1°C may be declared irrelevant as well. Thus, more than 75% collected data points can be omitted (Figure 4.5). Putting a sensor to sleep mode between relevant readings may result with 4 times longer battery lifetime.

With proposed solution, S-DMF for $\Delta_{max} = 0.5$°C collects only 85 208 data points with 276 errors. Compared to SMF which has the same NoE and collects 146 322 data points, S-DMF collects $\sim 40\%$ less data points. Thus, battery lifetime is increased 2.5 times. Random forest ML-DMF, for $\Delta_{max} = 0.5$°C collects 17 950 data points with 20 732 errors. In this case, NoE is higher compared to the previously discussed S-DMF, but battery lifetime is increased 5 times. When compared to the baseline SMF algorithm, random forest ML-DMF collects only $\sim 4\%$ of data points with equivalent $\sim 4.7\%$ errors as the baseline SMF. In non-critical systems, where error rate of $\sim 4.7\%$ is acceptable, ML-DMF can result in significant battery savings. Given that ML-DMF is active only $\sim 4\%$ of SMF time, the same battery could support the operation time that is approximately $1/0.04 = 25$ longer than the baseline SMF time. For example, 6 months of battery lifetime with baseline SMF would result in over 12 years of battery lifetime with ML-DMF, if one can tolerate somewhat increase in error rate ($\sim 4.7\%$ in this case).

That said, when using DMF algorithms it is important to be aware how do $\Delta_{max}$, $target\_accuracy$ and $n_{CC} - number\ of\ considered\ classes$ affect both energy consumption and accuracy. Therefore, above mentioned parameters should be selected in dependence with specific use case. To sum up, DMF algorithm reduces energy consumption, since only relevant data is collected and sensor sleeps in the meanwhile, while preserving defined level of accuracy.

Comparing DMF algorithm with filtering, where data reduction is performed after data is collected, filtering algorithm will be 100% accurate, as optimal baseline case described in section 4.1. However, the problem with filtering is that a sensor has to be continuously online collecting data with a predefined period. After collection, data is compared and irrelevant data is ignored. However, when combined with DMF, filtering can retain accuracy while additionally reducing the amount of data circulating through the network.

### 4.4.2 Optimizing the Size of the Training Dataset

Since data is collected every minute, total 2-year training dataset contains 1 051 200 readings. Two years is a very long period to collect data before training a model. Thus, shorter data collection periods are investigated, namely periods of 3 hours, 1 day, 1 week, 1 month, 6 months and 1 year. For each training period, an algorithm is prepared for 30 randomly selected signals, as shown in Figures 4.12 - 4.17.

Red dashdotted line in Figures 4.12 - 4.17 depicts results (NoR and NoE) on the logarithmic scale for different periods. Training with 3 hour period is unstable and results with high dispersion of the results; in other words the results of 30 executions are inconsistent. However, using 2-year or 1-year data is unnecessary since similar results can be achieved using 6 months period. For most use cases, periods in range of 1 day to 6 months can be selected. A period should be selected depending on the use case and the amount of errors that can be tolerated. In most situations, an algorithm can be created on a 1-day or a 1-week dataset, and updated with chunks of constantly incoming data as time passes.

Green dotted line in Figures 4.12 - 4.17 depicts the dispersion of the results for 30 random signals for logistic regression algorithm. Blue dashdotted line in Figures 4.12 - 4.17 depicts the results dispersion for random forest algorithm, while orange dashed line shows the results for artificial neural network algorithm. For all three ML-DMF algorithms, just like for S-DMF, 6-month period is long enough to reach stability and accuracy as the algorithm that is trained on 2-year dataset for the most use cases. However, for some use cases, 1-week or even 1-day period is good enough to start data collection using DMF algorithm. The algorithm can be later updated with larger dataset.

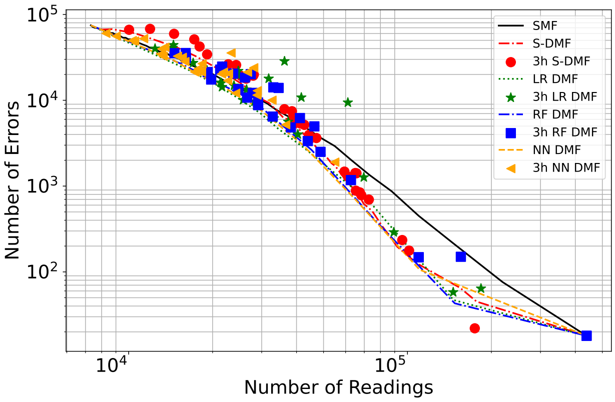


Figure 4.12. 3 hours S-DMF and ML DMFs with $\Delta_{max} = 0.5°C$; $target\_accuracy = 99\%$; $n_{cc} = 5$

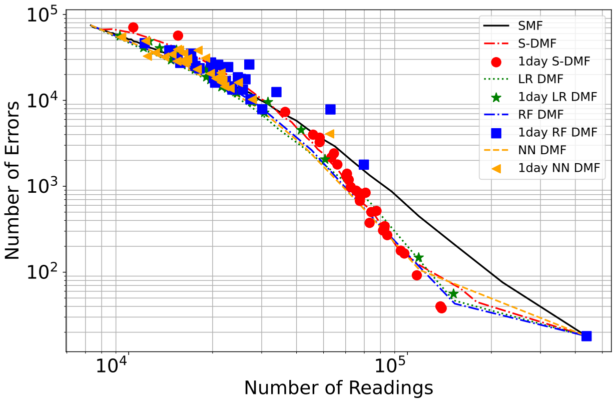


Figure 4.13. 1 day S-DMF and ML DMFs with $\Delta_{max} = 0.5°C$; $target\_accuracy = 99\%$; $n_{cc} = 5$

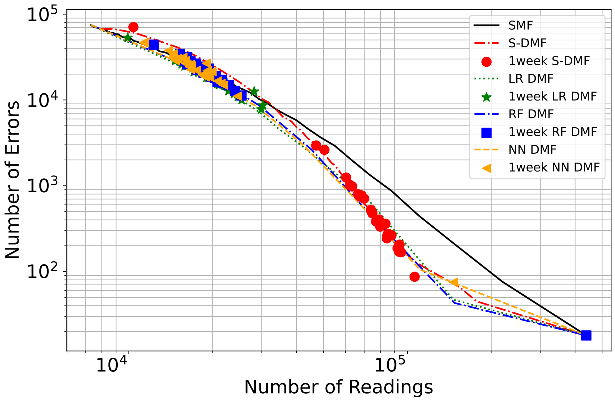


Figure 4.14. 1 week S-DMF and ML DMFs with $\Delta_{max} = 0.5°C$; $target\_accuracy = 99\%$; $n_{cc} = 5$

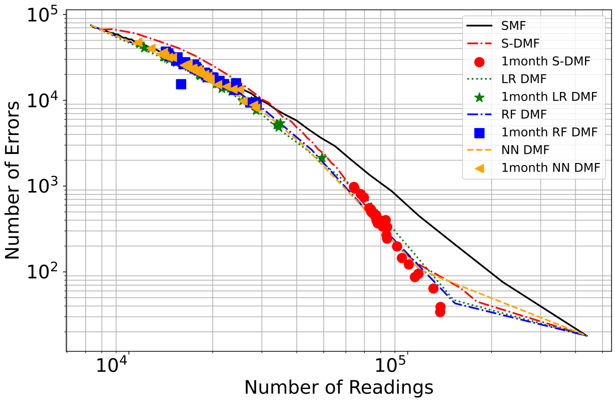


*Figure 4.15. 1 month S-DMF and ML DMFs with $\Delta_{max} = 0.5°C$; $target\_accuracy = 99\%$; $n_{cc} = 5$*

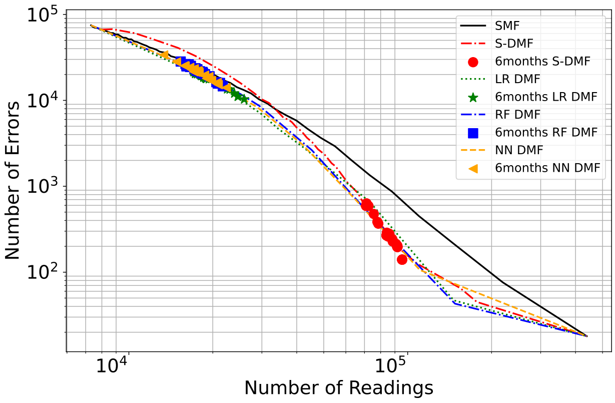


*Figure 4.16. 6 months S-DMF and ML DMFs with $\Delta_{max} = 0.5°C$; $target\_accuracy = 99\%$; $n_{cc} = 5$*

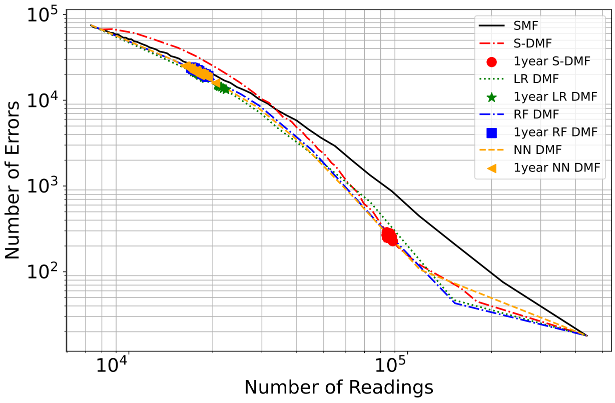


*Figure 4.17. 1 year S-DMF and ML DMFs with $\Delta_{max} = 0.5°C$; $target\_accuracy = 99\%$; $n_{cc} = 5$*

### 4.4.3 Processing Complexity

In terms of complexity, S-DMF algorithm is more complex than ML algorithms, especially during the training phase due to a large table of percentile values that has to be created. Table size depends on the period ($period^2$). For considered 60-minute period, the size of the table is $60^2$. Generated table is applicable for only one percentile, while every new percentile requires additional table. Once a table is generated it can be easily used, but in a case of large periods its size may be a problem for battery-powered IoT sensors.

In comparison, a complexity of ML algorithms does not depend on a period size, nor on $n_{CC}$. Logistic regression and random forest algorithms are the least complex and least time consuming solutions, especially during the training phase since they learn fastest. However, artificial neural network is the most complex and time consuming ML algorithm due to its complex mathematical background [48].

We have trained ML-DMFs with 2 and 4 features. As more features are added to ML-DMF, the more complex the algorithm becomes. However, our initial hypothesis was that more data results with better accuracy, thus increase in complexity is acceptable compromise. Nevertheless, results show that ML-DMFs with 4 features result with lower accuracy when compared to ML-DMFs with 2 features. Reason for such a behaviour is due to the selected features, which probably forced the ML algorithms to overfit, i.e., making them more biased towards seasonal changes rather then current trend.

## 4.5 Summary

In this research, we propose a dynamic monitoring frequency (DMF) algorithm that aims at collecting data only when sensor readings change by more than a predefined value between consecutive readings. Thus, a sensor is turned on only when a change in value of monitored phenomenon exceeds a predefined threshold. Two algorithms for reading sensor data using dynamic monitoring frequency are proposed and systematically analysed. The first algorithm is based on a statistical distribution of changes between consecutive readings on historically collected sensor data. The second algorithm is based on a machine learning (ML) that predicts future periods when the acceptable change in sensor data is expected.

Results show that the machine learning DMF gives better results for lower frequencies, while the statistical DMF algorithm gives better results for higher frequencies. The biggest advantage of the statistical DMF is that, compared to machine learning DMF, it does not limit the range of future periods. Comparing DMF algorithms with static monitoring frequency (SMF) algorithm that collects the same amount of data, DMF exhibits up to $\sim 80\%$ less errors. Comparing DMF and SMF algorithms with approximately the same amount of errors, DMF collects up to $\sim 40\%$ less data.

Collection of $\sim 40\%$ less data means that a sensor may sleep $\sim 40\%$ longer and thus save energy. Also, it is shown that shorter datasets can result with significant accuracy, i.e., using training dataset of 1 or 6 months instead of using full dataset that contains 2 years data. Lastly, processing complexity of both, statistical DMF and machine learning DMF algorithms are compared. Important highlight is that the complexity of machine learning DMF algorithms does not depend on a period length, while statistical DMF complexity has quadratic dependence to the length of period.

Next step in this field is the implementation of an online DMF algorithm. Such an algorithm would work in two phases. In first phase it collects data in very short periods and uses collected data for self-training. The second phase starts when significant accuracy is reached. During the second phase, the algorithm would apply gained knowledge and predict future periods. These two phases can interchange and algorithm can update its knowledge periodically.

# 5 Predicting Final Value from Part of Transient for Sensors with Preheating

The second part of the research [115] focuses on sensors that require preheating before being able to gather environmental data. Therefore, such sensors cannot collect data at the exact moment they are turned on but have to preheat first. Inspired by research by Jia et al. [34], we will analyse this topic on the use case of gas sensors. In the context of gas sensors, it is important to detect hazardous gases on time [116] using sensors with low power [117] and low cost [118], due to the rising scale of IoT.

When a circuit is turned on and off, voltages and currents take time to stabilize in order to obtain readings of the actual gas concentration. A momentary variation in the current or the voltage during this preheating transition is called a transient, only after which an actual value can be read. However, transients (especially in low-cost sensors) can take minutes, which consumes significant amount of energy from a battery powered sensors. Therefore, instead of waiting for the sensors to fully preheat, only parts of the transient can be collected. As Jia et al. show in their research [34] energy can be saved if gas values are predicted from a part of the transient.

In this research, instead of a high-precision sensor as used in [34], low-cost MQ-2 [50], MQ-5 [51] and MQ-6 [52] gas sensors are used. The goal is to investigate if an actual gas concentration can be predicted from the transient of a low-cost sensor and thus achieve low-cost but reliable IoT gas sensor network. On one hand, online period of the sensors must be long enough to provide a sufficient amount of data to distinguish one gas level from another. On the other hand, it should be short enough for the prediction process to be energy efficient. The resulting approach gives a minimised online period without significantly reduced accuracy.

Long short-term memory (LSTM) neural network (NN) is used to predict the actual value from the transient. LSTM network is suitable for predicting time series data, as it can distinguish and eliminate irrelevant and redundant data from the relevant one. In order to prepare LSTM algorithm that predicts actual values from the transients, data is collected with MQ-2 gas sensor. Collected data is normalized in the range between 0 and 1. Three-fold validation method is used to split the data and train LSTM neural network. Finally, algorithm parameters and the entire approach prepared for MQ-2 sensor are later evaluated on two other MQ sensors, namely MQ-5 and MQ-6.

Results show that MQ-2 gas sensor can predict values with RMSE 0.05938. When using parameters that provide the best results for MQ-2 gas sensor, RMSE of MQ-5 sensor is as low as 0.07002, but it is slightly higher for MQ-6 with a value of 0.1293. With high accuracy this solution can save up to 85% energy compared to a system where sensors are constantly online, and more than 50% compared to a system where sensors heat up to collect actual gas concentrations instead of a part of the transient.

This chapter is divided into six sections. The introduction is followed by section 5.1, which describes the preliminary analysis and methodology used in this research. Section 5.2 explains our implementation and results, while section 5.3 gives the evaluation of our approach on MQ-5 and MQ-6 sensors with discussion. Last section concludes this chapter.

## 5.1 Gas Sensor Characterization

In order to collect the actual value representing the gas concentration, a gas sensor needs to heat up. During this preheat time a voltage needs to stabilize, where a set of unstable values during this period are referred to as a transient. Once sensor readings are stabilized, the

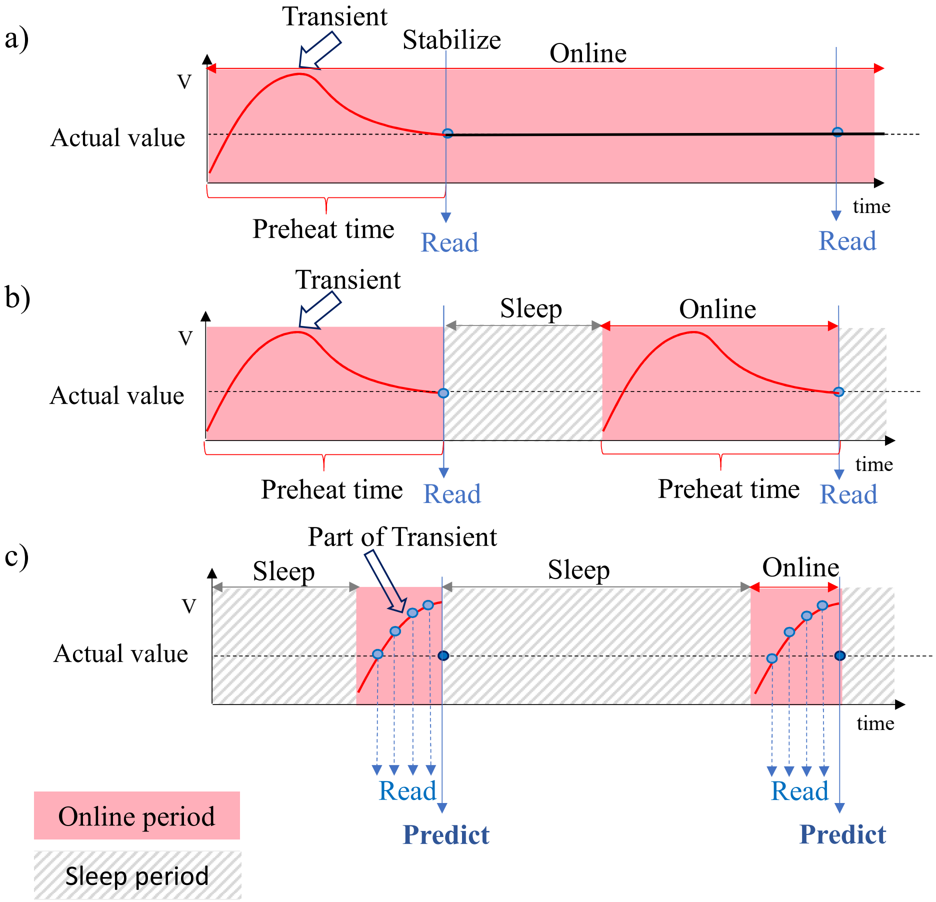


*Figure 5.1. Three scenarios for sensors setup, namely a) always online, b) with full preheat and sleep period and c) with prediction based on a partial preheat and sleep period*

sensor can periodically read data while continuously being online as shown in Figure 5.1a. In case the sensor is put to sleep to save energy, it has to preheat after every sleep and then read the last value of the transient, i.e., the actual value of gas concentration. This scenario is shown in Figure 5.1b. Our approach is depicted in Figure 5.1c, where a sensor reads multiple values from the first part of the transient, while the actual value is estimated from that part using machine learning. This way, the sensor will have a longer sleep period, while more values are read in its short online period.

### 5.1.1 Environment Setup

We utilize low-cost MQ gas sensors, more specifically MQ-2 for defining our methodology. MQ-2 sensor detects multiple hazardous gases, such as Liquefied Petroleum Gas (LPG), methane (CH4), Alcohol, Smoke and Propane. Since the goal is to predict the actual value from the transient and turn the sensor off, two sensors are used, namely a test sensor for predicting the actual values and a control sensor for the ground truth. Furthermore, the MQ sensors were exposed to a variety of gas concentrations.

The two sensors are connected to an Arduino Uno board that collects actual gas concentration values and then sends them via serial communications to the computer as shown in Figure 5.2. The control sensor MQ-2(1) is always online without a sleep period.

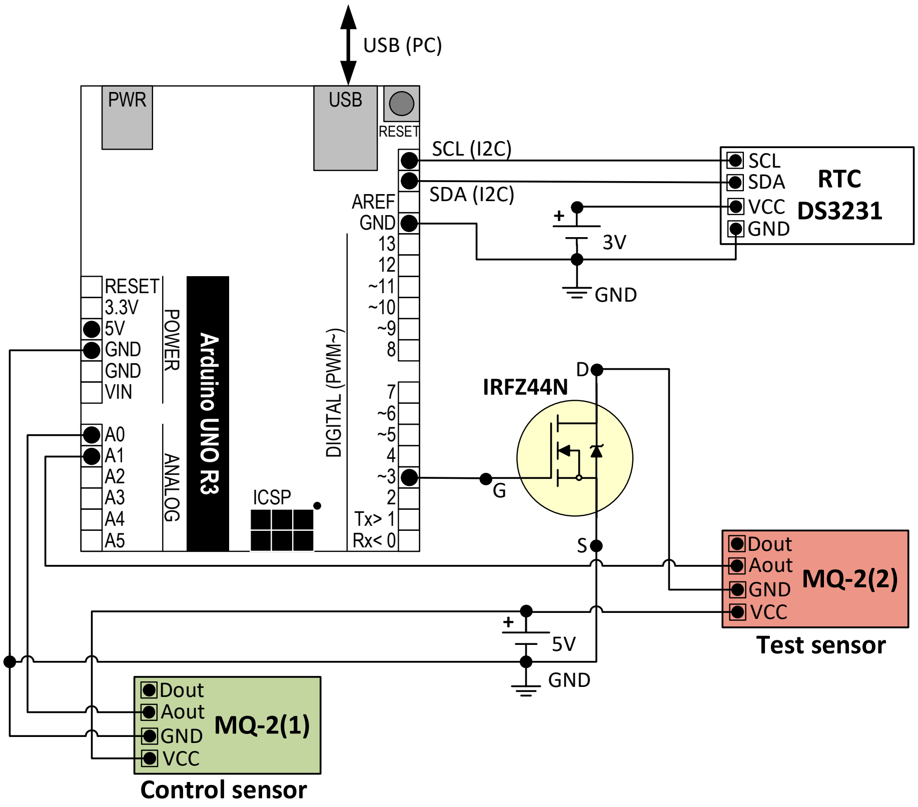


*Figure 5.2. Setup scheme with the test and control sensors, and real-time clock for turning on and off the test sensor*

The test sensor MQ-2(2) has a predefined sleep period after which it wakes up upon receiving a signal from the real-time clock (RTC) DS3231 [119]. In order to turn off the test sensor completely during the sleep period, a power MOSFET IRFZ44N [120] is used. Finally, to reduce external influences the entire setup, depicted in Figure 5.2, is placed in a sealed plastic container.

The data is collected with both sensors to find their correlation as they do not have the same nominal readings in the same environment. Therefore, linear regression is used to correlate their readings. The control sensor is left to collect the data continuously, while the test sensor collects only transients. The expected values of the test sensor are calculated from the known correlation with the control sensor. Finally, since MQ-2 sensors collect the data in a range of 0V to 5V, the data is normalized on a scale between 0 and 1. The highest value obtained for the gas during the experiments is 0.8 (4V) due to the specifics of MQ-2 sensors [50].

### 5.1.2 Online Period Characteristics

Prior to collecting a dataset used for building the LSTM prediction model, a preliminary analysis is performed in order to characterize behaviour of the transient. During online periods the sensor is warming up, while during sleep periods it is cooling down. That said, transients for different online-sleep ratios in atmosphere without gas are compared in Figure 5.3, where online periods of 10, 15, 20, 30, 40 and 60 seconds are considered, in combination with 60 and 120 seconds of sleep period, respectively.

As depicted in Figure 5.3, MQ-2 sensor transients in the environment without gas reach their maximum in first 15 seconds and the actual gas concentration (ground truth)

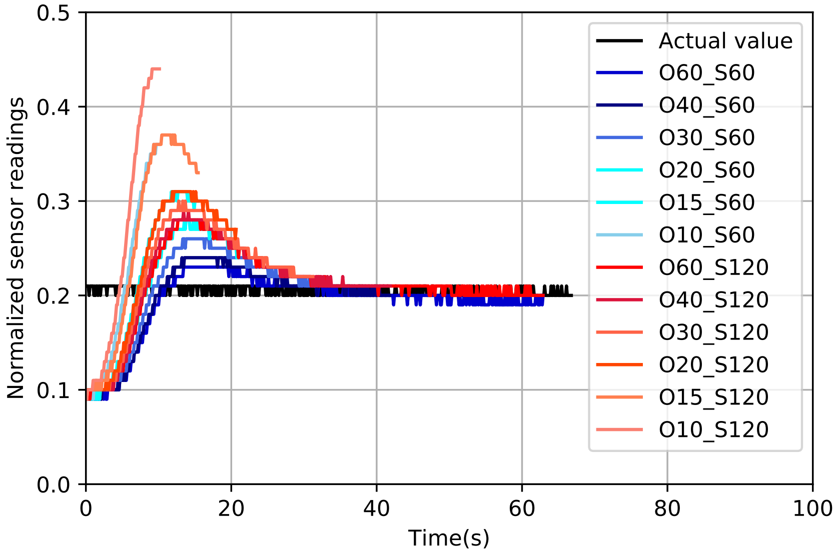


*Figure 5.3. Transient for different online (O) – sleep (S) ratios (without gas)*

in approximately 40 seconds. Regardless of the sleep periods, the readings stabilize at the actual gas level within those 40 seconds, where both sleep and online period affect only the maximum value, which is then regulated with its climbing and falling slopes to again reach their actual values after 40 seconds. Consequently, we select an online period less than 40 seconds in order to achieve savings with LSTM predictions. However, further experiments show that periods shorter than 20 seconds have high variance. As seen from Figure 5.4, in case of 10 second period consecutive readings result with inconsistent transients during longer runs, while the actual gas concentration remains the same. Thus, 20 seconds for the online period is selected, which shows stable readings during longer periods as depicted in Figure 5.4 where multiple continues readings show correlation above 95%.

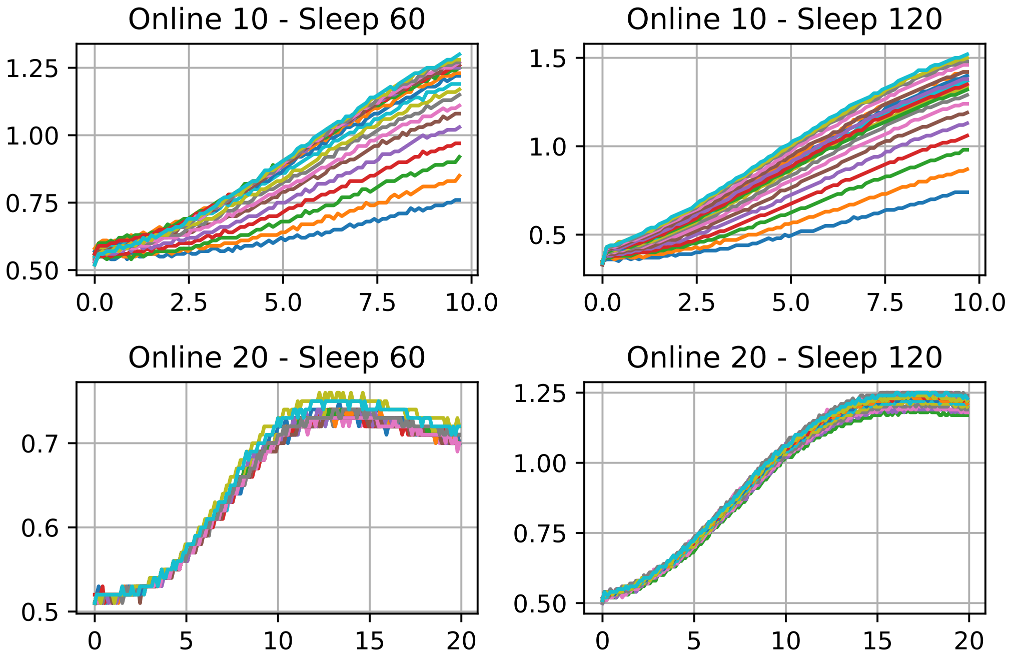


*Figure 5.4. Correlation of transients in case of different Online/Sleep ratios*

### 5.1.3 Sleep Period Characteristics

Further data collection is performed with 20 seconds online period and again 60 and 120 seconds sleep period, as well as with different gas levels inside the sealed container. Figure 5.5 shows transients for 5 different gas levels. The higher the gas concentration is, the higher the transient maximum is. However, transients at highest gas concentrations exhibit different patterns than the ones in lower gas concentrations, or without the gas at all. In the first 20 seconds the value jumps to 0.2 (1V), after which it starts to rapidly increase for the next 1,5 minutes, as seen in Figure 5.6. However, the initial jump is still significant enough to be distinguishable from lower gas levels, thus the online period of 20 seconds remains valid.

As depicted in Figure 5.6, MQ-2 sensor requires over 120 seconds to reach the actual

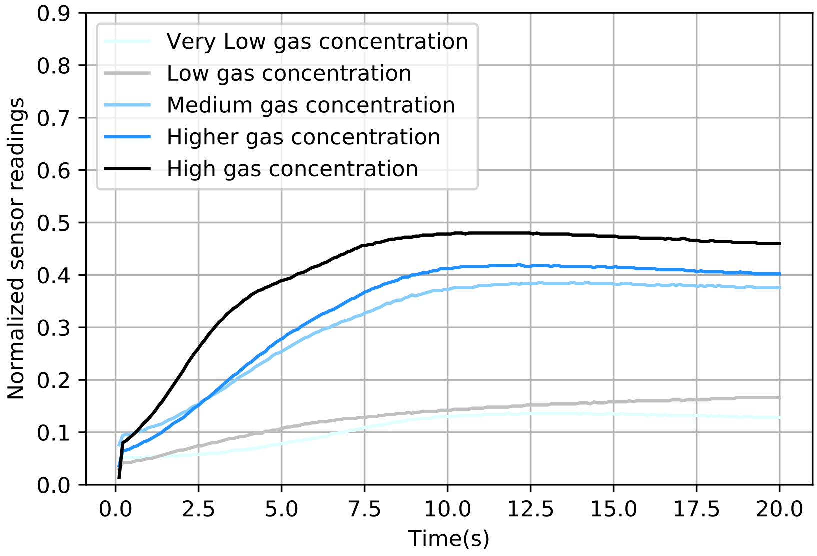


*Figure 5.5. Transients in different gas levels*

value in the environment with high gas concentrations. Consequently, any use of sleep functionality in such environment would require a preheat time of more than 120 seconds in order to read correct values. Therefore, we select 120 seconds for our sleep period along with 20 seconds for the online period, and thus get gas sensor readings approximately every 140 seconds.

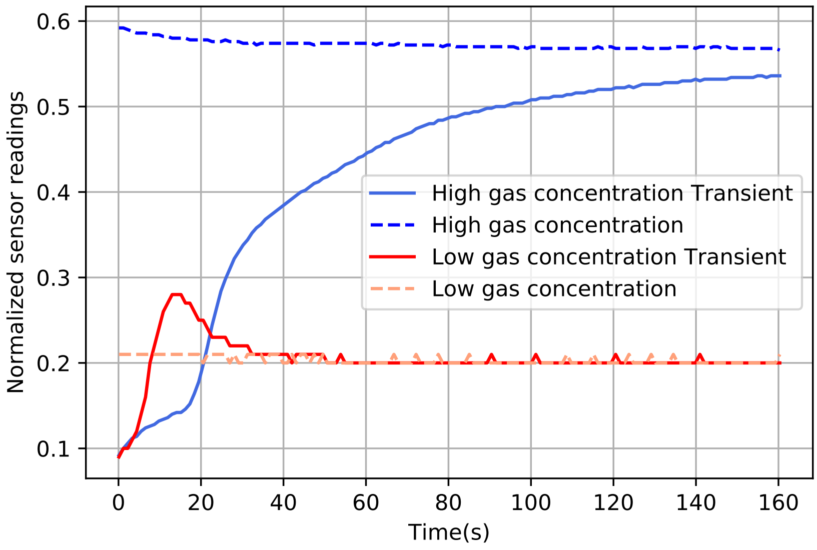


*Figure 5.6. Low vs High gas level transients*

### 5.1.4 Collection of Transients

In order to collect a dataset comprising a set of transients, previously defined online-sleep period is used, along with different gas concentrations. For 20 seconds online period, data is collected 10 times in a second (i.e., every 100ms) resulting with 186 values in a single transient (186 instead 200 since every reading takes $\sim$ 0.15 milliseconds). Total dataset contains 381 transients with their matching actual values, hence resulting in total of 70 866 values. Entire dataset is visualized in Figure 5.7.

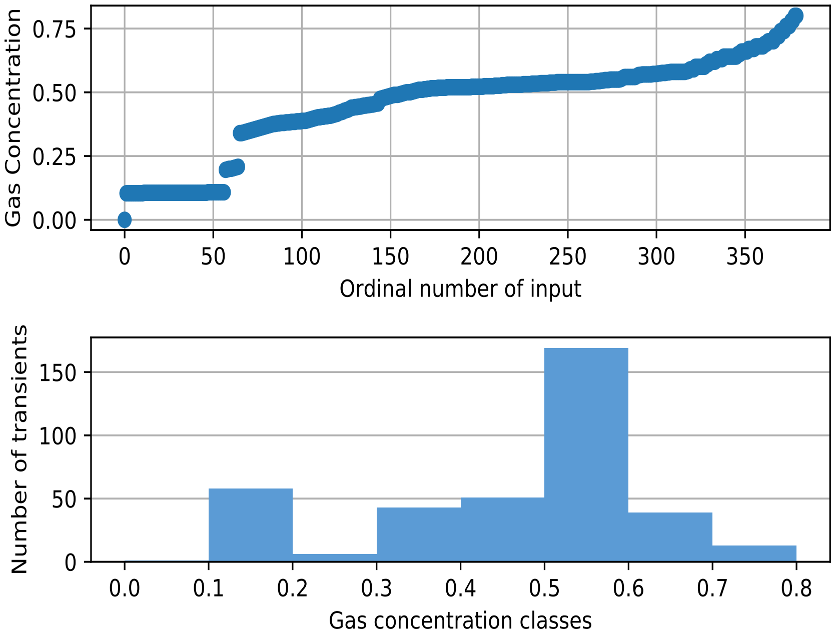


*Figure 5.7. Actual values for 381 transients (top figure) grouped in 8 classes (bottom figure)*

Furthermore, gas concentrations are classified in eight classes listed in Table 5.1, where the entire dataset $C$ comprises all classes, namely $C = c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7$. It is important to highlight that class $c_5$ contains the most elements since 0.5 value is recognised as a border between environments with and without significant gas concentration. Therefore, the classification is used for even stratification of transients when applying three-fold validation approach during LSTM model training and testing, i.e., to even out presence of all classes in training as well as in the test dataset.

*Table 5.1. Gas concentration classes*

| Class name | Gas concentration [0-1] |
|---|---|
| $c_0$ | $0-0.1$ |
| $c_1$ | $0.1-0.2$ |
| $c_2$ | $0.2-0.3$ |
| $c_3$ | $0.3-0.4$ |
| $c_4$ | $0.4-0.5$ |
| $c_5$ | $0.5-0.6$ |
| $c_6$ | $0.6-0.7$ |
| $c_7$ | $0.7-0.8$ |

## 5.2 LSTM Model Training and Validation

In this section, we explain the configuration and usage of LSTM neural network for predicting readings of MQ-2 gas sensor based on its initial transient. LSTM algorithm and the dataset stratification and diversification are explained. Afterwards, LSTM parameters are selected based on the best model accuracy and used for selecting the read frequency for input data as well as for testing models performance with scarce data.

### 5.2.1 LSTM Algorithm

LSTM is suitable for our application as the transient represents time series data. While LSTM neural network handles time series data very well, it is still important to carefully select parameters, as well as a dataset. In order to find the best parameters, different combinations of number of neurons, learning rates and epochs are tested. More neurons can help in case of underfitting, reducing number of epochs helps in case of overfitting, while smaller learning rate helps to eliminate exploding gradients. LSTM network used in this research contains 1 LSTM layer, Relu activation function, Adam optimizer and MSE loss calculation with fixed seed (123). Fixed seed is used in order to be able to reproduce the results. Different combinations of neurons, learning rates and epochs listed in Table 5.2 are used in order to obtain the best results.

### 5.2.2 Dataset Stratification and Diversification

To train LSTM algorithm for MQ-2 gas sensor 304 transients out of 381 from the dataset shown in the previous section are used. This leaves 77 transients for the test set, which is around 20% of the whole dataset. The training set is further split using stratified three-fold algorithm, which selects one third of transients from each class to validate LSTM algorithm.

*Table 5.2. Parameter combinations for building LSTM*

| **Parameter** | **Values** |
|---|---|
| Number of neurons | 20, 50, 80 |
| Number of Epochs | 250, 500, 750, 1000 |
| Learning Rate | 0.001, 0.0001, 0.00001 |

In order to diversify the training set and reduce data transmission, other combinations are considered as well besides 186 value long transient:

1. Analysing all 186 values collected during 20 seconds ($\sim$ 9 readings per second)
2. Analysing 20 values collected during 20 seconds
   (a) Data collected every second
   (b) Data from the first 20 seconds, 20 logarithmically spaced readings
   (c) Data from the first 20 seconds, 20 exponentially spaced readings
3. Analysing only 10 values collected during 20 seconds (the sensor still has to be online for 20 seconds, otherwise it won't heat up enough). Here we include five combinations:
   (a) Data from the first 20 seconds, collected every two seconds
   (b) Data from the first 10 seconds (collected every second)
   (c) Data from the first second (collected every 100ms)
   (d) Data from the first 20 seconds, 10 logarithmically spaced readings
   (e) Data from the first 20 seconds, 10 exponentially spaced readings

Furthermore, to test how the best of these solutions deal with scarce data, i.e., missing entire classes of readings, the algorithm is executed on the training set without data from every class ($c_0 - c_7$), as defined in Table 5.3. Finally, the results are discussed and compared using Root Mean Square Error (RMSE) [121]. Since RMSE reflects the distance between real and predicted values it is used to evaluate the performance of predictions, i.e., smaller RMSE implies higher prediction accuracy. Equation 5.1 defines a formula for RMSE, where $e$ is the error between the predicted and the actual value.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e^2} \tag{5.1}$$

*Table 5.3. Sets of datasets with included/excluded classes*

| Dataset | Included/excluded classes |
|---|---|
| $C \setminus \{c_0\}$ | $[\ ][c_1][c_2][c_3][c_4][c_5][c_6][c_7]$ |
| $C \setminus \{c_1\}$ | $[c_0][\ ][c_2][c_3][c_4][c_5][c_6][c_7]$ |
| $C \setminus \{c_2\}$ | $[c_0][c_1][\ ][c_3][c_4][c_5][c_6][c_7]$ |
| $C \setminus \{c_3\}$ | $[c_0][c_1][c_2][\ ][c_4][c_5][c_6][c_7]$ |
| $C \setminus \{c_4\}$ | $[c_0][c_1][c_2][c_3][\ ][c_5][c_6][c_7]$ |
| $C \setminus \{c_5\}$ | $[c_0][c_1][c_2][c_3][c_4][\ ][c_6][c_7]$ |
| $C \setminus \{c_6\}$ | $[c_0][c_1][c_2][c_3][c_4][c_5][\ ][c_7]$ |
| $C \setminus \{c_7\}$ | $[c_0][c_1][c_2][c_3][c_4][c_5][c_6][\ ]$ |
| $C \setminus \{c_0, c_2, c_3, c_4, c_6\}$ | $[\ ][c_1][\ ][\ ][\ ][c_5][\ ][c_7]$ |
| $C \setminus \{c_0, c_2, c_3, c_4, c_5\}$ | $[\ ][c_1][\ ][\ ][\ ][\ ][c_6][c_7]$ |

### 5.2.3 LSTM Parameters Selection

As described in the previous section, LSTM is created based on gas concentration collected by MQ-2 sensor. Table 5.4 shows the results of different combinations of cell numbers, epochs and learning rates with their respective RMSE. Some large updates to weights during training cause a numerical overflow or underflow often referred to as exploding gradients, which results with NaN values for some combination of parameters, especially in case of larger learning rate. Nevertheless, 50 cells, 500 epochs and the learning rate of 0,0001 gives the best performance during training. This set of parameters also outperforms others when applied on the test set by giving even better results, hence confirming that it is not either overfit or underfit.

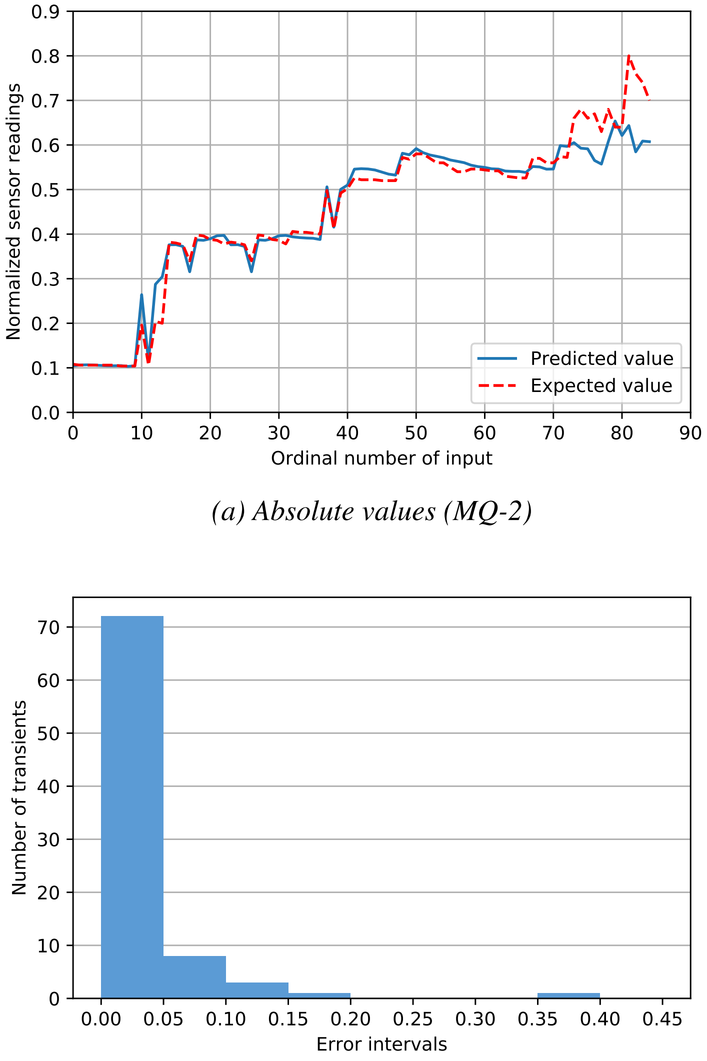
Figure 5.8b shows a histogram of errors for 186 values long transients for MQ-2 sensor on the test set. Most errors are below 5% although RMSE is 0.06829. There are few errors above 30%, however as visible from Figure 5.8a those errors occur for high gas concentrations. Since predicted values are above 0.5 (2.5V) they are still classified as dangerous gas level.

### 5.2.4 Frequency Selection for Input Data

In further analysis, we use the algorithm configuration that gives the best results for 186 values long transient and test it on a smaller number of transient values. Results in Table 5.5 show that more data is not necessarily better than less data. Despite expectations that a larger transient will result in better predictions, the best predictions are in the case when values are evenly sampled at 1Hz. If the algorithm learns from the data that contains large number of features, it often overfits the noise and does not work well on a real world data. Furthermore, logarithmically spaced data yields notable results when trained either on 10 or 20 values

*Table 5.4. Selecting optimal parameters for LSTM NN*

| Cells | Epochs | Learning rate | Train set RMSE | Test set RMSE |
|---|---|---|---|---|
| 20 | 250 | 0.001 | 0.1748 | 0.1557 |
| 20 | 500 | 0.001 | NaN | NaN |
| 20 | 750 | 0.001 | 0.2103 | 0.2505 |
| 20 | 1000 | 0.001 | NaN | NaN |
| 20 | 250 | 0.0001 | 0.1687 | 0.1599 |
| 20 | 500 | 0.0001 | 0.1116 | 0.0872 |
| 20 | 750 | 0.0001 | 0.1083 | 0.0919 |
| 20 | 1000 | 0.0001 | 0.1060 | 0.0881 |
| 20 | 250 | 0.00001 | 0.4042 | 0.4007 |
| 20 | 500 | 0.00001 | 0.2889 | 0.2890 |
| 20 | 750 | 0.00001 | 0.2162 | 0.2239 |
| 20 | 1000 | 0.00001 | 0.1945 | 0.1885 |
| 50 | 250 | 0.001 | NaN | NaN |
| 50 | 500 | 0.001 | NaN | NaN |
| 50 | 750 | 0.001 | NaN | NaN |
| 50 | 1000 | 0.001 | NaN | NaN |
| 50 | 250 | 0.0001 | 0.1217 | 0.0992 |
| **50** | **500** | **0.0001** | **0.0926** | **0.0683** |
| 50 | 750 | 0.0001 | NaN | NaN |
| 50 | 1000 | 0.0001 | 0.1496 | 0.1610 |
| 50 | 250 | 0.00001 | 0.2906 | 0.2923 |
| 50 | 500 | 0.00001 | 0.2011 | 0.2028 |
| 50 | 750 | 0.00001 | 0.1682 | 0.1678 |
| 50 | 1000 | 0.00001 | 0.1421 | 0.1280 |
| 80 | 250 | 0.001 | NaN | NaN |
| 80 | 500 | 0.001 | NaN | NaN |
| 80 | 750 | 0.001 | NaN | NaN |
| 80 | 1000 | 0.001 | NaN | NaN |
| 80 | 250 | 0.0001 | 0.1126 | 0.0807 |
| 80 | 500 | 0.0001 | 0.1175 | 0.0991 |
| 80 | 750 | 0.0001 | NaN | NaN |
| 80 | 1000 | 0.0001 | NaN | NaN |
| 80 | 250 | 0.00001 | 0.2347 | 0.2407 |
| 80 | 500 | 0.00001 | 0.1615 | 0.1577 |
| 80 | 750 | 0.00001 | 0.1517 | 0.1401 |
| 80 | 1000 | 0.00001 | 0.1226 | 0.1025 |



*(a) Absolute values (MQ-2)*

*(b) Errors histogram (MQ-2)*

*Figure 5.8. Comparison of predicted and expected values for MQ-2 gas sensor*

collected over a 20-seconds period. In contrast, exponentially spaced data results in a lower RMSE than evenly spaced readings. These findings suggest that the LSTM algorithm gains the most information from the first part of the transient. However, it is important to note that information about the last part of the transient is also crucial since logarithmically spaced data has a higher RMSE than a model trained solely on data collected during the first second or first 10 seconds.

### 5.2.5 Performance Testing with Scarce Data

To test how the models deal with scarce data, the algorithm is executed without transients from every class in the training set, and then tested on the test set that contains excluded classes. Results given in Table 5.6 show that some classes are smaller in both train and test

*Table 5.5. MQ-2 RMSE for different transient sizes*

| Test | Size | Train set RMSE | Test set RMSE |
|---|---|---|---|
| 1) | 186 values – every 100ms | 0.0926 | 0.0683 |
| **2)** | **20 values – every second** | **0.0615** | **0.0594** |
| 2b) | 20 values – logarithmically spaced | 0.0706 | 0.0654 |
| 2c) | 20 values – exponentially spaced | 0.1330 | 0.0874 |
| 3a) | 10 values – every 2s | 0.0940 | 0.0871 |
| 3b) | 10 values – first 10 seconds | 0.1226 | 0.1286 |
| 3c) | 10 values – first second | 0.1606 | 0.1673 |
| 3d) | 10 values – logarithmically spaced | 0.0808 | 0.0797 |
| 3e) | 10 values – exponentially spaced | 0.1136 | 0.0992 |

sets, hence they do not affect the results significantly, while largest $c_5$ class affects results significantly. The algorithm also tries to learn behaviour from the first and two last classes, however results are not promising. When the biggest $c_5$ class is included in training, test set RMSE is 0.16624, which is a satisfying result in terms of machine learning.

Since 20 values long transients show better results than 186 values long one, similar tests are performed with such setup as well. Results in Table 5.7 show that it is possible to train data only with first, last and biggest class $c_5$ with RMSE of 0.116627. In general, using 20 values long transient outperforms 186 long one.

*Table 5.6. RMSE for different classes (transient size 186)*

| Dataset | Train set RMSE | Test set RMSE |
|---|---|---|
| $C \setminus \{c_0\}$ | 0.0794 | 0.0859 |
| $C \setminus \{c_1\}$ | 0.0546 | 0.2074 |
| $C \setminus \{c_2\}$ | 0.1171 | 0.0985 |
| $C \setminus \{c_3\}$ | 0.1311 | 0.1102 |
| $C \setminus \{c_4\}$ | 0.1104 | 0.1140 |
| $C \setminus \{c_5\}$ | 0.1992 | 0.1580 |
| $C \setminus \{c_6\}$ | 0.1044 | 0.1204 |
| $C \setminus \{c_7\}$ | 0.0862 | 0.1013 |
| $C \setminus \{c_0, c_2, c_3, c_4, c_6\}$ | 0.2052 | 0.1662 |
| $C \setminus \{c_0, c_2, c_3, c_4, c_5\}$ | 0.1662 | 0.2279 |
| $C$ | *0.0926* | *0.0683* |

*Table 5.7. RMSE for different classes (transient size 20)*

| Dataset | Train set RMSE | Test set RMSE |
|---|---|---|
| $C \setminus \{c_0\}$ | 0.0715 | 0.0634 |
| $C \setminus \{c_1\}$ | 0.0409 | 0.1731 |
| $C \setminus \{c_2\}$ | 0.0751 | 0.0668 |
| $C \setminus \{c_3\}$ | 0.0564 | 0.0715 |
| $C \setminus \{c_4\}$ | 0.0648 | 0.0685 |
| $C \setminus \{c_5\}$ | 0.1916 | 0.1435 |
| $C \setminus \{c_6\}$ | 0.0611 | 0.0608 |
| $C \setminus \{c_7\}$ | 0.0664 | 0.0634 |
| $C \setminus \{c_0, c_2, c_3, c_4, c_6\}$ | 0.0819 | 0.1166 |
| $C \setminus \{c_0, c_2, c_3, c_4, c_5\}$ | 0.2210 | 0.1789 |
| $C$ | *0.0615* | *0.0594* |

## 5.3 Evaluation and Discussion

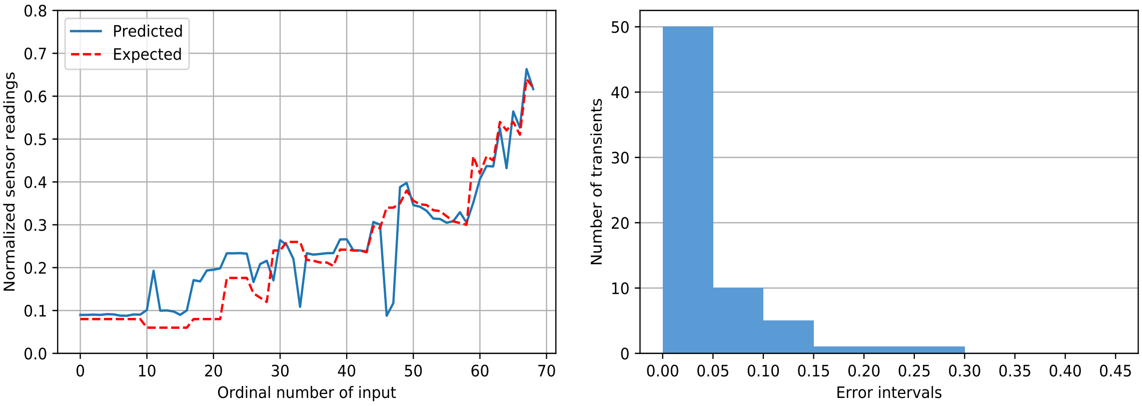
In order to evaluate our methodology, approach is tested on MQ-5 and MQ-6 gas sensors to verify its applicability on different low-cost sensors. The same online-sleep ratio of 20 seconds for an online and 120 seconds for a sleep period is used for testing MQ-5 and MQ-6 sensors.

### 5.3.1 MQ-5 Sensor Results

After data is collected with MQ-5 gas sensor following the same approach described in section 5.1, neural network with parameters that provide the best results for MQ-2 gas sensor are used. Collected dataset contains 223 transients in the train set and 70 transients in the test set (∼24%). In the case of 186 values long transient, RMSE on the test set is 0.10801, while in the case of 20 values long transient that has better results for MQ-2, test set RMSE is better as well with the value of 0.07002, as shown in Table 5.8. As it is visible from Figure 5.9a and Figure 5.9b, the most errors are less than 10%, with 2 predictions from class 0.3-0.4 with the error above 20%.

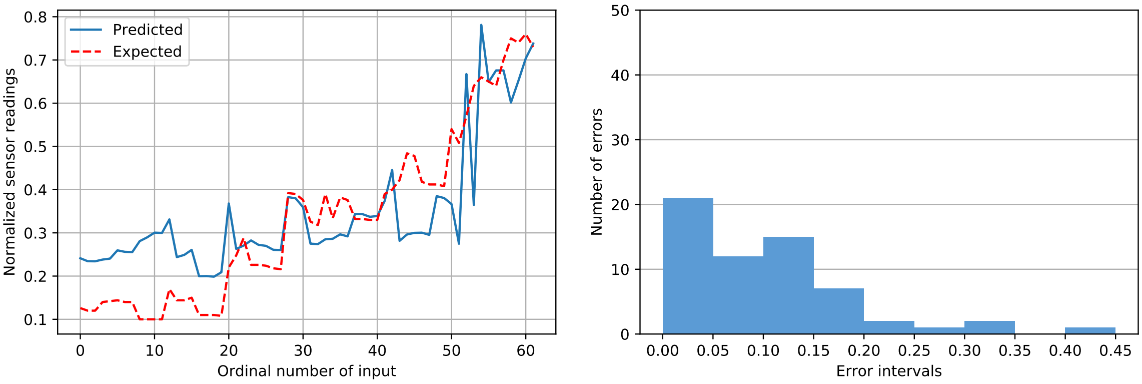
### 5.3.2 MQ-6 Sensor Results

The same procedure is used for MQ-6 gas sensor. However, in this case the results are only slightly worse than the ones for MQ-2 and MQ-5. Used dataset contains 216 transients in the train set and 59 transients in the test set (∼27%). Again, the test set results are acquired for both 186 and 20 values long transients. As with MQ-2 and MQ-5 gas sensors, MQ-6 also



*(a) Absolute values (MQ-5)* *(b) Errors histogram (MQ-5)*

*Figure 5.9. Comparison of predicted and expected values for MQ-5 gas sensor*



*(a) Absolute values (MQ-6)* *(b) Errors histogram (MQ-6)*

*Figure 5.10. Comparison of predicted and expected values for MQ-6 gas sensor*

*Table 5.8. MQ-5 RMSE*

| Size | Train set RMSE | Test set RMSE |
|---|---|---|
| 186 | 0.1104 | 0.1080 |
| **20** | **0.0702** | **0.0700** |

*Table 5.9. MQ-6 RMSE*

| Size | Train set RMSE | Test set RMSE |
|---|---|---|
| 186 | 0.1597 | 0.1405 |
| **20** | **0.1238** | **0.1238** |

gives better results with 20 values long transients having RMSE of 0.1239, while 186 values long transients give RMSE of 0.1405, as shown in Table 5.9. Figures 5.10a and 5.10b depict that more than 85% of predicted values have an error lower than 15% with the maximum error of 25%.

### 5.3.3 Discussion on Energy Efficiency

The results presented in this research show that LSTM prediction model can be used with low-cost gas sensors such as MQ-2, MQ-5 and MQ-6 with sufficient accuracy. Predictions for MQ-2 give the best accuracy measured with RMSE of 0.05938067 as the model is calibrated for the specific sensor. In case of MQ-5 sensor, RMSE is 0.07002, which is almost as good as in the case of MQ-2 sensor, while the results for MQ-6 are slightly worse with RMSE value of 0.1239 that is still acceptable in terms of machine learning. Note that parameters that give the best results for MQ-2 sensor are used *as is* without the calibration for MQ-5 and MQ-6, which implies that the model could be further tweaked to obtain better results.

Putting sensors to sleep only makes sense when trying to achieve energy savings. Energy consumption is described with the following expression:

$$E = P_{Tx} \cdot T_{OA} + P_{MQ} \cdot T_A + P_{MA} \cdot (T_{OA} + T_A) + P_{MS} \cdot T_S \tag{5.2}$$

where:

- $P_{Tx}$ is power consumed by LoRaWAN module used for data transmission [122]
- $P_{MQ}$ is power consumed by MQ-2 sensor [50]
- $P_{MA}$ is power consumed by microcontroller during online period [123]
- $P_{MS}$ is power consumed by microcontroller during sleep period [123]
- $T_{OA}$ is time on air calculated based on calculations shown in [124] using LoRaWAN with Code Rate = 1, Spreading Factor = 10 and Bandwidth = 125kHz
- $T_A$ is online period
- $T_S$ is sleep period

Energy consumption is calculated for three sensor setups described in section 5.1 (Figure 5.1) and results are shown in Table 5.10. Since transmission interval is less than a second,

*Table 5.10. Energy consumption*

| **Scenario** | $P_{Tx}$ | $P_{MQ}$ | $P_{MA}$ | $P_{MS}$ | $T_{OA}$ | $T_A$ | $T_S$ | $E$ |
|---|---|---|---|---|---|---|---|---|
| a) Always online | 92.4mW | 900mW | 615mW | 131.5mW | 0.206s | 140s | 0s | 212.25J |
| b-1) With full preheat and sleep period (the best scenario) | | | | | 0.206s | 40s | 100s | 73.89J |
| b-2) With full preheat and sleep period (the worst scenario) | | | | | 0.206s | 120s | 20s | 184.57J |
| **c) With prediction based on a partial preheat and sleep period** | | | | | **0.370s** | **20s** | **120s** | **46.22J** |

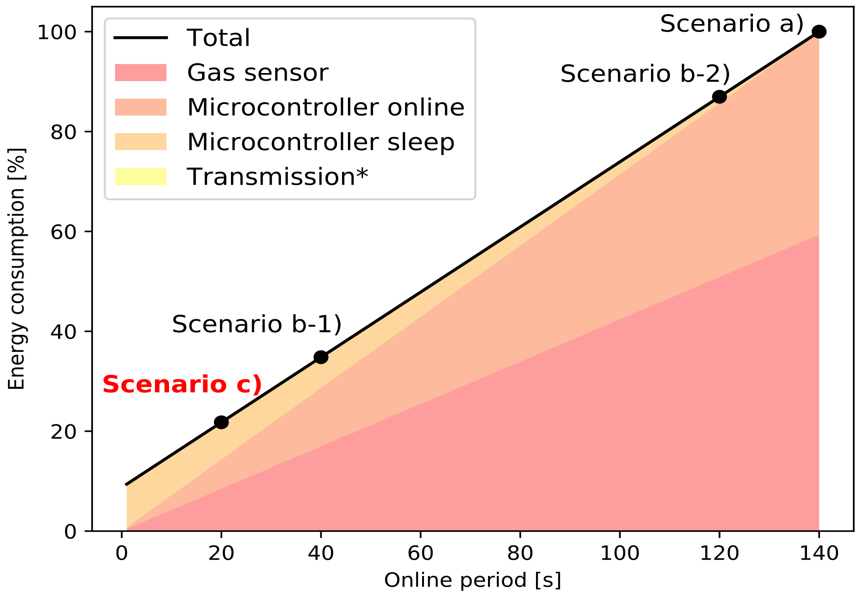


*Figure 5.11. Energy consumption compared to the sensor that is always online (*Transmission energy is insignificant compared to others, thus not visible in this figure)*

when combined with transmission power, it results with insignificant energy consumption compared to large power and intervals of online and sleep periods. The most significant energy consumer is a gas sensor which requires up to 900mW (Figure 5.11). Note that a microcontroller consumes energy during both periods; online and sleep. However, energy consumed during sleep period is $\sim$ 4.5 times lower compared to the power consumed during online period. Energy saving can be even bigger when using other microcontrollers instead of Arduino Uno microcontroller board. That said, energy consumption highly depends on the online period; the longer the online period is, the higher is the power consumption. Due to specifics of MQ gas sensors explained in section 5.1.2, the shortest online period that is suitable for such predictions is 20s, followed by 120s long sleep period.

As it is shown in Figure 5.11 and Table 5.10, our solution uses $\sim$80% less energy compared to Scenario a) when sensor is always online, while online period is decreased by $\sim$85%. Since MQ-2 sensor needs at least 40 seconds to read actual value in the atmosphere without gas, and around 120 seconds in the conditions with high gas concentration, our approach can be compared to the best and the worst case. As it is visible from Figure 5.11 and Table 5.10, comparing our solution to Scenario b-1) With full preheat and sleep period - the best scenario (40s online - 100s sleep); our solution consumes $\sim$40% less energy while collecting $\sim$50% less data. Note that Scenario b-1) will miss large gas concentrations. In the worst case, the sensor is online for 120s and sleeps 20s, which is the exact opposite of our approach, meaning that the sensor will reduce data amount for $\sim$15% with $\sim$13% energy savings.

To sum up, the longer sleep period results with the higher energy savings. In our case, the sensor is online for 20 seconds and then asleep for 120 seconds, resulting with approximately 500 - 520 seconds (∼8,5 minutes) of online time per hour, while sleeping for 3 000 - 3 120 seconds (∼51 minutes). Therefore, it is being online ∼3.5 hours a day; 1 day a week; 4.5 days a month or 52.15 days in a year. That said, our solution significantly reduces amount of data with duty cycle of (∼85%) while increasing energy efficiency for ∼80%.

Comparing our solution to a scenario with the full preheat and sleep period (Figure 5.1b), one trade-off has to be noted. Our solution sends 20 values towards a gateway and the prediction is calculated on the gateway side, while the sensor in scenario in Figure 5.1b sends only a single value. As it is visible from Figure 5.11 and Table 5.10, transmission energy in case of 1 and 20 bytes is almost the same and it can be ignored compared to the energy consumed by the gas sensor and Arduino Uno.

Moreover, even though a stabilised 5V external power supply is used in herein described research, it is important to be aware of the limitations of Arduino analog-to-digital (ADC) converter. The Arduino Uno supports operating frequencies up to 16 MHz, and the standard ADC clock frequency is 125 kHz. Therefore, the ADC requires approximately 13 clock cycles to complete a conversion, granting a standard maximum sampling rate of 9600 samples. The Arduino Uno has 6 analog inputs and an analog-to-digital converter with 10-bit resolution, meaning there are 1024 distinct values that can be returned as a result from the ADC [125]. However, the measurement and conversion of voltage to the voltage step of an analog input are not as stable as anticipated. If the Arduino Uno is plugged in via USB, the input voltage is around 5V, but it is not stable. Therefore, it is important to know the supply voltage at the exact moment the ADC reading is collected. Moreover, for a referent voltage of 5V, an ADC value of 1 would result in a voltage step of 4.88$mV$ since there are 1024 ADC values, from 0 to 1023. Since the maximum ADC value is 1023, the corresponding maximum voltage is 4.9951V.

To improve the accuracy of the Arduino ADC, additional external modules can be connected to improve accuracy via hardware, or external libraries freely available from the Arduino community on the Internet can be used to improve accuracy via software. For example, the 16-bit analog-to-digital converter ADS1115 can be used to enable higher resolution data acquisition than what was possible with an Arduino Uno alone. This precision ADC had 4 channels and was connected to the Arduino through an I2C-compatible serial interface. The ADS1115 can convert data at rates of up to 860 samples per second, enabling high-resolution measurements of both large and small signals [126]. External libraries [127] can also be used to improve ADC resolution by oversampling and/or averaging ADC data [128–130].

Furthermore, TensorFlow Light for Microcontrollers (including Arduino platform) [131] is now available and machine learning models can be deployed on the edges of an IoT system, i.e., a gateway [132] . Using this approach, another potential trade-off is the amount of energy consumed when machine learning algorithm is deployed on a battery powered sensor as the energy consumption can be higher than the energy savings gained with 120 seconds long sleep period. Although this is out of our scope in this research, it is important to be aware of the potential challenges and trade-offs.

## 5.4 Summary

The focus of this research is on sensors that need to preheat in order to reliably collect data from the environment. Instead of waiting for sensor to heat up, the data trend that sensor collects while heating up is analysed and referred to as a transient. It is shown that long short-term memory (LSTM) neural network can be used to learn and later predict actual value, from a part of the transient. In this case, LSTM neural network is trained to predict the actual values of gas concentration from the transient of MQ-2 gas sensor acquired during its preheat period. Furthermore, evaluation is performed on two different gas sensors, namely MQ-5 and MQ-6. In the case of MQ-2 gas sensors, obtained RMSE is 0.05938067. The optimal LSTM parameters used for MQ-2 were applied on MQ-5 and MQ-6 as well. Predictions for MQ-5 sensor give RMSE value of 0.07002, while only slightly higher for MQ-6 with the value of 0.1239.

The main contribution of this research is the investigation of MQ-2 sensor behaviour, as well as the methodology for building a prediction model based on LSTM that we applied on two additional low-cost gas sensors. The results indicate that in order to extend a battery life, a sensor should wake up periodically to collect data and sleep in the meantime. In the case of sensors that require heating up before collecting the actual value, only the beginning of the transient can be collected and the actual value predicted on the gateway side using LSTM neural network. This way, instead of being continuously online, the sensor collects data for 20 seconds and sleeps for 120 seconds. Compared to the solution when the sensor is online until the actual value is reached (up to two minutes) and then being sent to sleep, our approach collects half as much data in the same 140 seconds period.

Our future work includes investigating the energy consumption in terms of data transmission. We also plan to implement a machine learning algorithm on Arduino board and investigate its energy consumption compared to the energy savings achieved with 120 seconds sleep period.

# 6 Signal Type Classification - Time Series Data Classification

On the other side of the IoT infrastructure, data classification is considered as a last part of the research work. After data is collected, it has to be stored in data centers in a predefined and consistent format, as inconsistent storage may potentially lead to the Digital Dark Age [35], i.e., data from the digital age can be irrevocably lost [36]. Therefore, researchers have identified the three challenges: (1) recognition of the signal type of a newly added sensor; (2) distinguishing signals from different sources if the configuration is lost; and (3) correct classification of signals after malfunctions of a distributed storage system, which require metadata discovery/recovery. However, we propose a single solution, i.e., signal type recognition, that can address all of the three above mentioned challenges. Therefore, it is possible to recognise a signal coming from a specific type of sensor in an IoT network and determine its type from a small signal chunk, thus enabling quick signal classification based on the signal type recognition approach.

In this chapter [133], we research the ability of machine learning (ML) to learn characteristics describing a specific signal type, and use that knowledge to successfully classify signal chunks into specific signal types in an IoT network. In this research, three classification approaches are compared, namely ($1c$) one class classification [5], ($2c$) two class (binary) classification and ($mc$) multi class classification [134].

- ($1c$) One class classification approach contains $n$ models for $n$ different classes, where each model recognises corresponding class without knowledge of other $n-1$ classes.
- ($2c$) Two class classification approach is a multi-model classification which also consists of $n$ models for $n$ different classes, where each model recognises corresponding class, while using the information about the remaining $n-1$ classes.
- ($mc$) Multi class classification approach consists of only one model which differentiates between $n$ classes, thus information about all $n$ classes is required.

In this work, we consider both, streaming data and stored data modes. We show that multi class ($mc$) random forest model has the highest accuracy. It can correctly classify signals based on only 20 consecutive values in $> 75\%$ cases, regardless if it is used on consecutive signal data in case of streaming or on randomly selected data from a large pre-stored dataset.

The rest of the chapter is organized as follows. Introduction is followed by section 6.1, which describes our dataset and methodology. Section 6.2 presents results, while sections 6.3 and 6.4 discuss robustness evaluation for different use cases. Last section concludes the chapter.

## 6.1 Dataset and Methodology

Datasets and algorithms used to address the issues associated with signal type recognition from time series sensor data are covered in this section.

### 6.1.1 Dataset

The dataset used in this research contains 11 different signal types from 2 separate data sources - meteorological data and computer resource utilization data (Table 6.1).

A meteorological dataset consists of 7 data types, namely humidity, ultraviolet (UV) radiation, wind direction, wind speed, air pressure, temperature, and solar radiation. In this dataset, data is collected every minute during almost 3 years (1/Jan/2017 - 6/Nov/2019), thus containing 1 490 164 readings per monitored phenomenon.

The computer resource utilization dataset consists of 4 data types, namely available memory, CPU utilization, space utilization, and memory utilization. In this dataset, data is collected every minute during one week (2/Jul/2021 - 9/Jul/2021) from 14 computers, thus containing 140 508 readings.

Among the 11 signal types, there are signals with the same data range but different behaviour (i.e., CPU utilization and humidity), signals with a different range but similar behaviour (UV and solar radiation), signals with the same range and same behaviour (available memory and memory utilization), and signals with range and behaviour different from all the rest (air pressure).

The first step in building a machine learning (ML) model, i.e., an instance of an ML algorithm with specified hyper-parameters, is to prepare the ML dataset. Therefore, the data from all 11 data sources is split into 3 parts (Figure 6.1): 1) a train set, which is used for building an ML model; 2) a validation set for validation of prepared models; and 3) an evaluation dataset for additional analysis, i.e., streaming and stored data analysis. Since datasets from different sources do not have the same length, the size of validation and

*Table 6.1. Dataset overview*

| Dataset | Collection period | Data types | Data sources | Input data (per data type) | Input data Total |
|---|---|---|---|---|---|
| Meteorological Dataset | 1 minute | 7 | 1 | 1 490 164 | 10 431 148 |
| Computer Dataset | 1 minute | 4 | 14 | 140 508 | 562 032 |
| Entire Dataset | 1 minute | 11 | N/A | N/A | 10 993 180 |

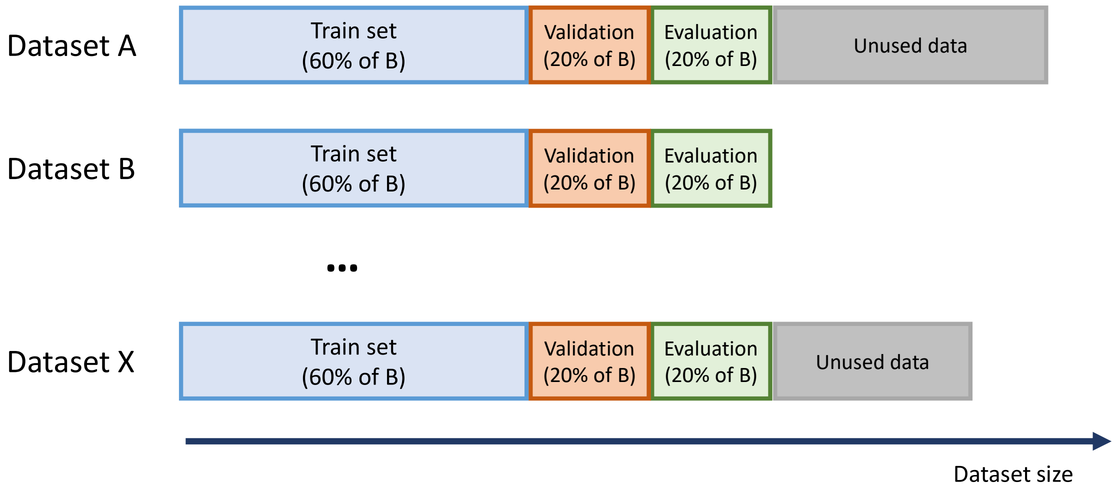


*Figure 6.1. Splitting datasets for training, validation and evaluation*

evaluation datasets is calculated as 20% of the smallest dataset (i.e., 20% of dataset B in Figure 6.1). The remaining part (at least 60% for the smallest dataset) is used for building an ML model.

### 6.1.2 Methodology

The goal of this research is to compare and evaluate the applicability of different approaches and corresponding ML models in order to successfully tackle three defined challenges: (1) recognising signal type when a new sensor is added to an IoT network; (2) recognising signal type when configuration metadata is lost due to connection issues and; (3) recognising signal type when signal configuration and metadata are lost due to unexpected problems with distributed storage.

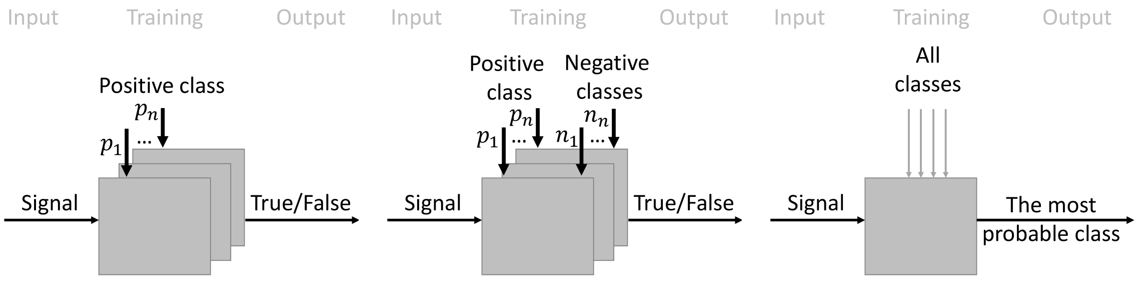
To meet these requirements, the model should successfully distinguish 11 signal types, i.e., 11 classes, based on IoT sensor readings. Distinguishing signal types in a short period requires the model to be able to classify data from a small part of the signal (*window*) collected in that short period. Furthermore, due to their limited configurability, models are trained on a fixed size of signal chunks (*window*). In addition, pre-calculated statistical values of that *window* (min, max, std, avg, median) are also used as inputs to ML models. Initial results are obtained for $window = 20$. Afterwards, different $window\ sizes = \{10, 50, 100\}$ are analysed and compared for the most accurate models from the best and the worst approaches.

In order to select the best model, three approaches are considered and analysed - one class, two class and multi class classification. Furthermore, all ML models are implemented using Python's Sklearn library with default hyper-parameters. Additionally, the deep learning algorithm long short-term memory (LSTM) neural network is analysed as well. To build such a network, one LSTM layer and default Keras hyper-parameters are used. However, since number of cells does not have a default value, it is calculated using Zaccone and Karim's equation defined in [135], and results in 16 LSTM cells.

One class ($1c$) approach is usually used to detect anomalies, thus, in this context it is used to recognise what is *not* an anomaly [53]. Thus, $1c$ classification requires 11 models, one for each of the 11 signal classes. Each model is trained only to recognise its own positive class, without knowledge of the other 10 classes (Figure 6.2a). One class models tested for solving this problem include one class support vector machine (OCSVM) and isolation forest (IF).

Two class ($2c$) classification also requires 11 models, each of which is trained on its positive class and a small amount of inputs from each of 10 negative classes (Figure 6.2b). Each algorithm contains 50% data from positive class and 50% data from negative class (5% from each of the remaining 10 classes), therefore the dataset is balanced. Thus, information about the remaining 10 classes is required. Two class models tested for solving this problem include logistic regression (LR) as a linear model, random forest (RF) as a tree-based model, k-nearest neighbors (kNN) as a non-linear model and long short-term memory (LSTM) as a deep learning model. Additionally, a support vector machine (SVM) model is selected to be compared with OCSVM. One class and two class ML models both output the predicted probability that a signal belongs to the monitored class, thus recognising only one class, i.e., the positive class. Furthermore, the same ML model is used for all 11 classes in both $1c$ and $2c$ approaches.

Finally, multi class ($mc$) [54] classification has the knowledge of all 11 classes and thus results with a single ML model, which outputs 11 predicted probabilities, one for each of 11 classes (Figure 6.2c). Multi class models tested for solving this problem are the same as the ones used for two class classification, i.e. RF, SVM, kNN, LR and LSTM.



*(a) One class (1c)* *(b) Two class (2c)* *(c) Multi class (mc)*

*Figure 6.2. Illustration of three approaches*

## 6.2 Comparing Classification Algorithms

In this section, the results of models from three approaches, i.e. 1*c*, 2*c* and *mc*, are presented in order to select a model that results with the highest number of correctly classified signals. Figure 6.3 depicts the accuracy for all observed models from all three approaches with a *window* of size 20. Figures 6.4 - 6.6 depict results as heatmaps. The horizontal axis of each heatmap represents the *actual* signal class, while the vertical axis represents the signal class recognised by an ML model. Thus, each row represents the actual signal, while a column shows which models classify the signal as their own class. Therefore, the diagonal shows an accuracy of an ML model, i.e., how accurately an ML model recognises the corresponding signal class. Hence, the darker the diagonal, the more accurate the ML model is, and *vice versa*, the more scattered the dark color is, the less accurate the ML model is.

The results of 1*c* approach models, trained only on a positive class, are depicted in Figures 6.4a and 6.4b. As it is shown in Figures 6.3, 6.4a and 6.4b, 1*c* models give poor results with an average accuracy lower than 20%. Nevertheless, in the 1*c* approach, IF has slightly higher accuracy than OCSVM since IF has at least a visible diagonal. However, the IF model classifies all signals except one as solar radiation. Therefore, the IF model incorrectly classifies signals as solar radiation in the majority of cases. However, the 1*c* approach can be used to eliminate the least probable classes and, thus, to reduce the number of possible classes.

Comparing 2*c* (Figures 6.5a - 6.5e) and *mc* (Figures 6.6a - 6.6e) approaches, it is clear that RF and kNN distinguish signals much better than SVM, LR or LSTM, since their average accuracies (Figure 6.3) are above 40% for 2*c* and above 70% for *mc* approach. Nevertheless, RF, with an average accuracy above 80%, is more accurate than kNN (Figure 6.3). Moreover, RF correctly classifies between all classes, while kNN can not differentiate between some classes, i.e., solar and UV radiation. As an opposite example, majority of the

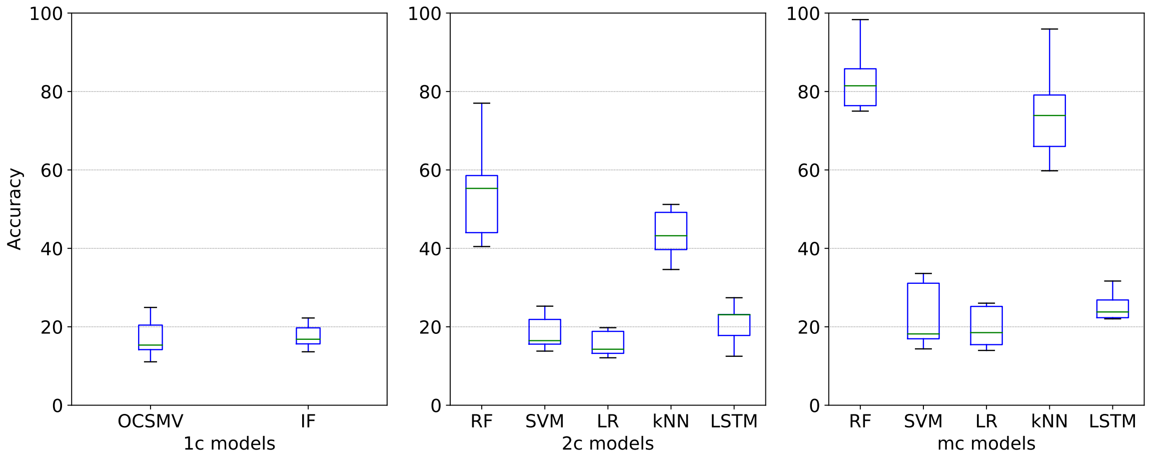
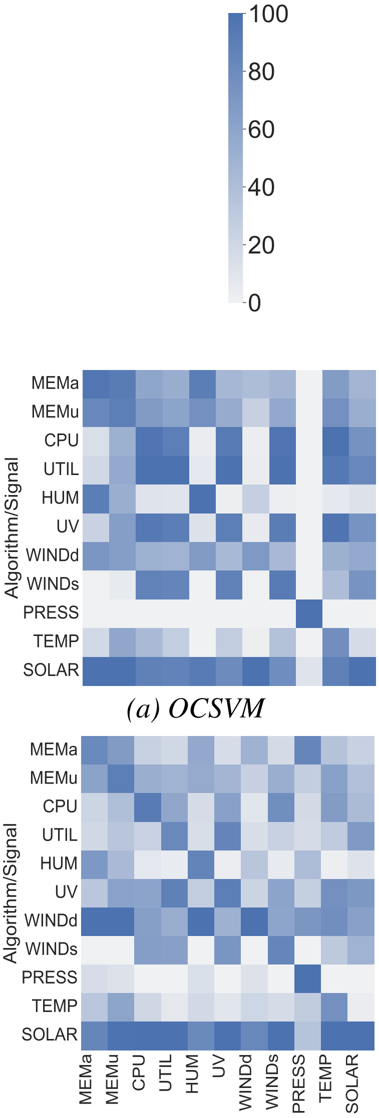


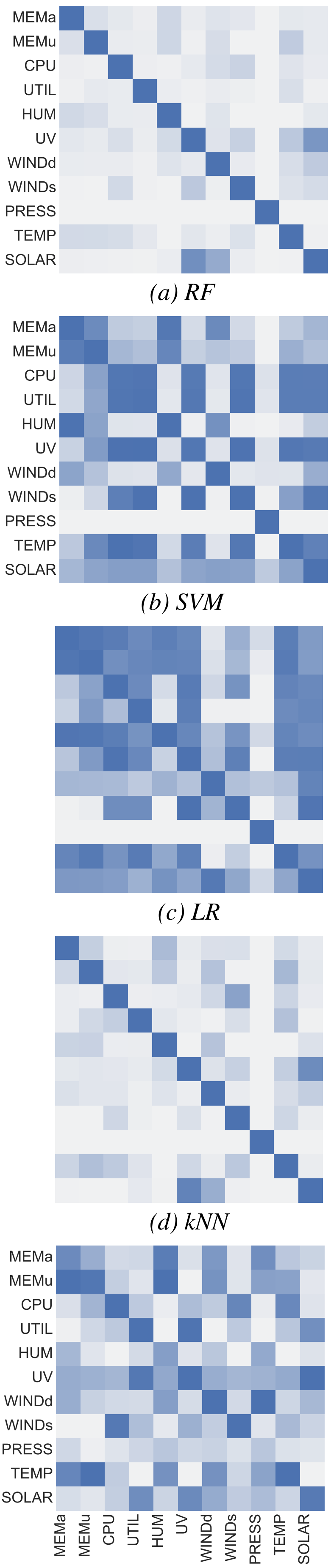
*Figure 6.3. Models accuracy*



*(a) OCSVM*

*(b) IF*

*Figure 6.4. One Class results*



*(a) RF*

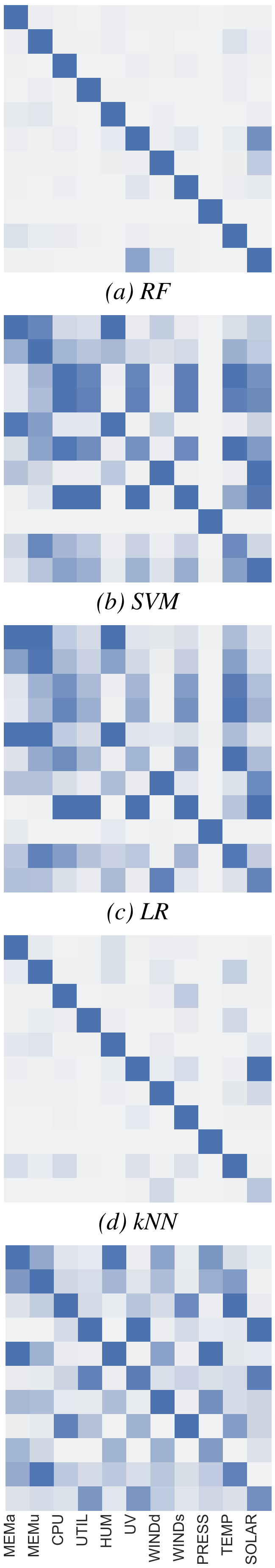
*(b) SVM*

*(c) LR*

*(d) kNN*

*(e) LSTM*

*Figure 6.5. Two class results*



*(a) RF*

*(b) SVM*

*(c) LR*

*(d) kNN*

*(e) LSTM*

*Figure 6.6. Multi Class results*

studied models, OCSMV (Figure 6.4a), SVM (Figure 6.5b and Figure 6.6b) and LR (Figure 6.5c and Figure 6.6c) correctly recognise one class, i.e., air pressure, while this instance of LSTM (Figure 6.5e and Figure 6.6e) among all classes only recognises wind speed.

Finally, as shown in Figures 6.3 - 6.6, the *mc* models performs better than 2*c* models. Such results are due to *mc* models having knowledge on all classes. In comparison, the 2*c* models know a positive class and a negative class as a combination of all other classes. Based on this analysis, in the following section we focus only on the IF model as the best model without knowledge of other classes and the *mc* RF as the best model with the knowledge of other classes.

## 6.3 Algorithms Robustness

Based on the findings from the previous section, further evaluation is performed only on 1*c* IF and *mc* RF algorithms as they give the best results for corresponding approaches. Our goal here is to evaluate the robustness of the algorithms (1*c* IF and *mc* RF) to variations of *window* size and two operation modes, streaming and stored data mode.

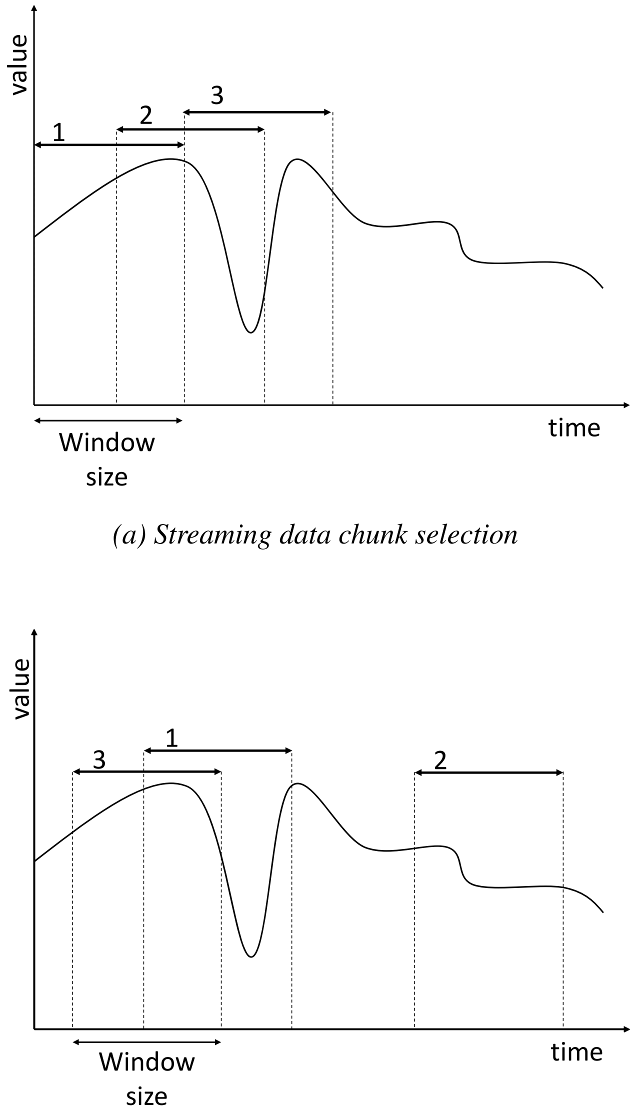
First, an analysis of *window* size, a *chunk* of a signal collected in a fixed time period, is conducted. Four *windows* with 100, 50, 20 and 10 readings are analysed to determine how a number of readings in a *window* affects the accuracy of the algorithms. Second, a streaming mode is evaluated, which defines data that is continuously arriving in real time, thus it is being evaluated on consecutive *chunks* of *window* size (Figure 6.7a). Third, stored data mode is used when data is already collected and stored, however metadata is lost and stored data types are unknown. Therefore, *chunks* can be selected randomly from the entire set of stored data (Figure 6.7b).

### 6.3.1 Window Size Analysis

*Window* size analysis is performed on a dataset of 4000 readings for every class. Those readings are split into *chunks* of different *window* sizes.

Results for the algorithm with the worst performance from previous section (1*c* IF) for *window* sizes of 100, 50, 20 and 10 are shown in bar chart (Figure 6.8) and heatmaps (Figures 6.9a - 6.9d), respectively. Results for the algorithm with the best performance from the previous section (*mc* RF) for *window* sizes 100, 50, 20 and 10 are shown in bar chart (Figure 6.8) and heatmaps (Figures 6.10a - 6.10d).

As depicted in Figures 6.8 - 6.10, the results show that the *window* size has no noticeable effect on the algorithms accuracy since average accuracy is $\sim 20\%$ for all 1*c* models and $\sim 80\%$ for all *mc* models. Thus, $window = 20$ is selected for further evaluation. In the following subsections, streaming and stored modes are evaluated on *chunks* of selected $window = 20$.



*(a) Streaming data chunk selection*

*(b) Stored data chunk selection*

*Figure 6.7. Chunk selection*

### 6.3.2 Streaming Data Analysis

Streaming data analysis implies that data is arriving in the real time. Analysis is evaluated on 100 random signals for all 11 classes resulting with validation set of 1 100 input signals. Each of 1 100 signals consists of 1 000 consecutive readings which are split on consecutive *chunks* of *window* size 20, thus resulting with total number of 980 *chunks* (Figure 6.7a). Figure 6.11a for *mc* RF and Figure 6.11b for $1c$ IF depict the average accuracy of the algorithms for all 11 classes and all 100 random executions, i.e., an average of 1 100 executions. The blue area at the bottom (*actual*) represents a case where the algorithm correctly classifies the positive class. Other areas denoted $2nd - 11th$ depict cases when the positive class is predicted as $2nd - 11th$ probable class. Detailed average results for all classes are depicted in Figure

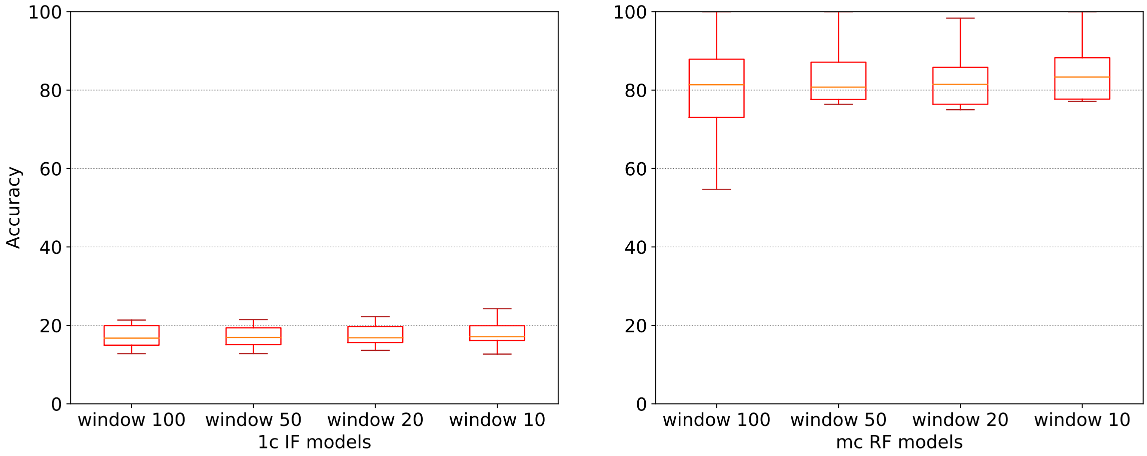
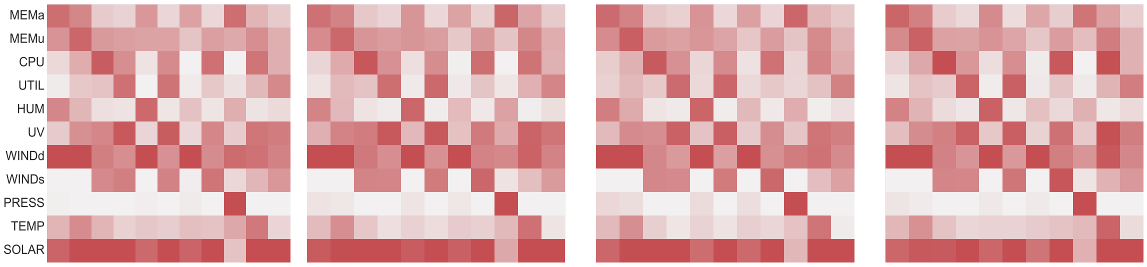
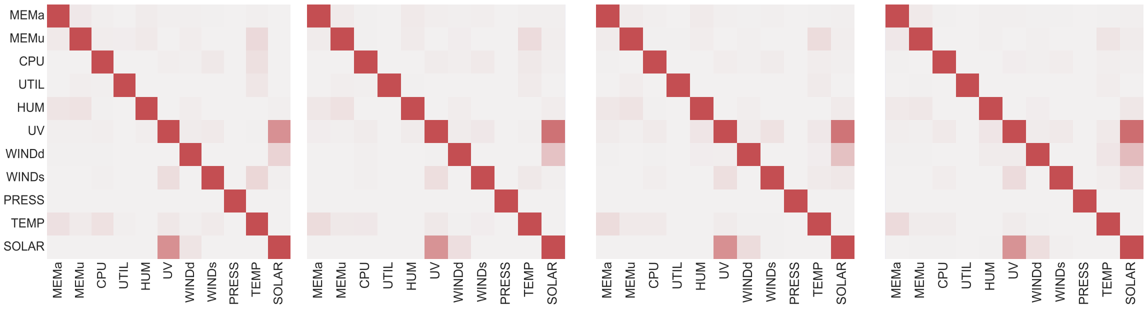


*Figure 6.8. Models accuracy for various window sizes*



*(a) window 100* *(b) window 50* *(c) window 20* *(d) window 10*

*Figure 6.9. IF varying window size*

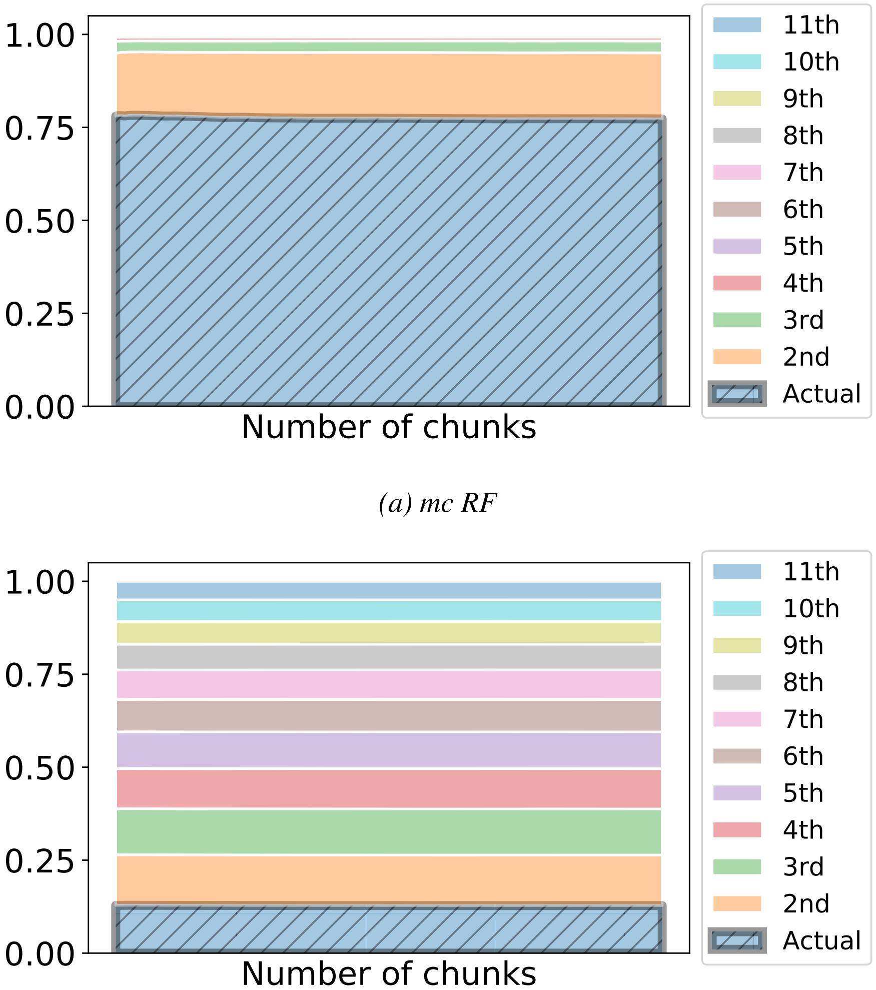


*(a) window 100* *(b) window 50* *(c) window 20* *(d) window 10*

*Figure 6.10. mc RF varying window size*

6.12.

Results show that RF multi class correctly classifies data in $> 75\%$ of cases (Figure 6.11a). Furthermore, if 2 most probable classes are considered, *mc* RF can eliminate the remaining 9 classes with 95% accuracy, while correctly classifying $> 98\%$ of signals into 3 most probable classes. In contrast, 1*c* IF results are not as accurate since 1*c* IF does not have information about classes other than the positive one (Figure 6.11b). A signal is correctly classified in only $\sim 13\%$ of cases. However, having in mind that one class approach is used only when the total number of classes is unknown, 1*c* IF can still be used. Thus, it can help to



*(a) mc RF*

*(b) 1c IF*

*Figure 6.11. Streaming data mode - combined mean accuracy per number of chunks (window size = 20)*

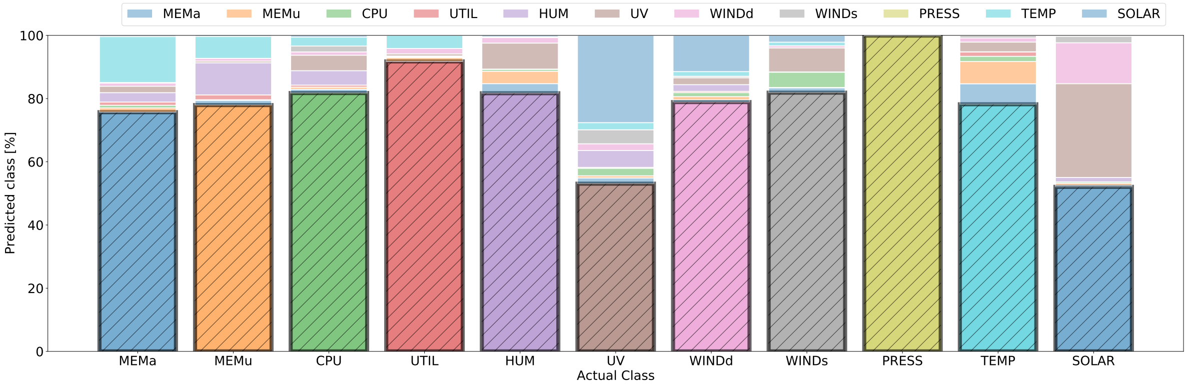


*Figure 6.12. Accuracy of mc RF streaming data*

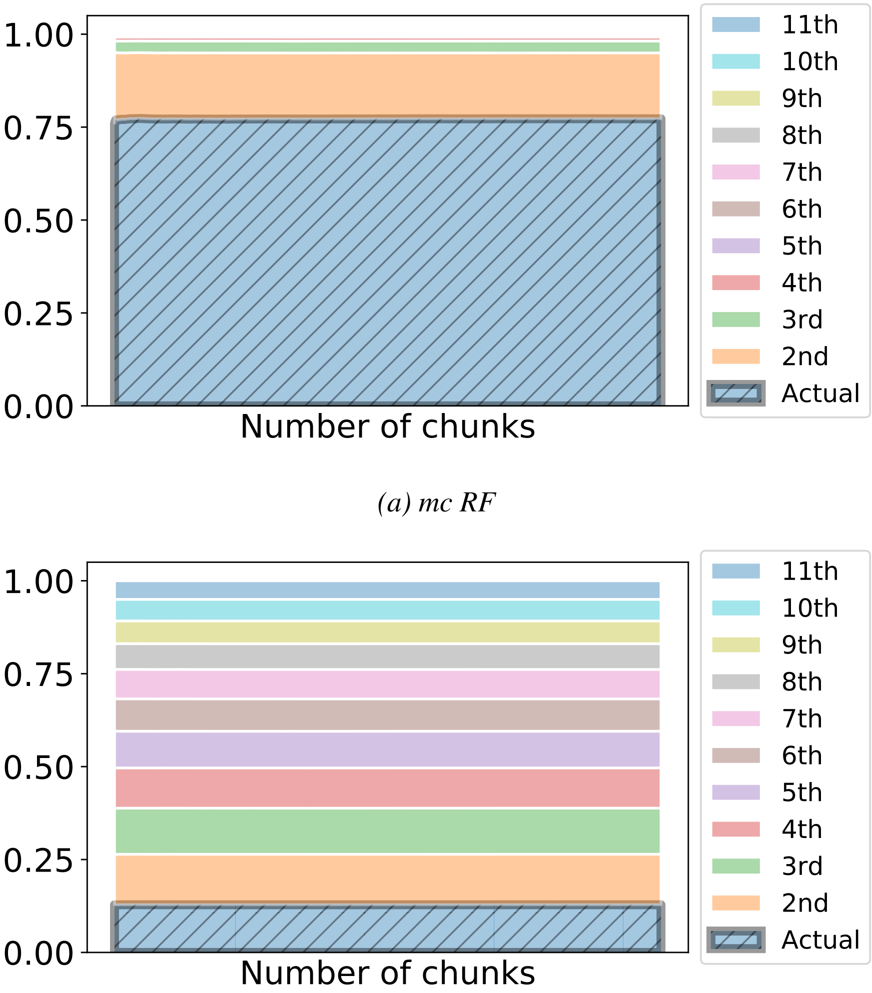
reduce the number of classes to consider since it classifies positive class in the first 6 classes in $\sim 70\%$ of cases.

### 6.3.3 Stored Data Analysis

In contrast to the streaming data, stored data is used when classifying previously collected data. The same analysis as for the streaming data is performed for the stored data mode. For all 11 classes, stored data analysis is evaluated on 100 random signals. Within each of the signals, 980 *chunks* of *window* size 20 are randomly selected (Figure 6.7b).

Figure 6.13a for *mc* RF and Figure 6.13b for 1*c* IF depict the average accuracy of the algorithms when all *chunks* of all 1 100 executions are selected randomly from the whole evaluation dataset. Figure 6.14 depicts the average results of *mc* RF.

Results show that RF multi class correctly classifies data in $\sim 77\%$ of cases for the most



*(a) mc RF*

*(b)* 1*c IF*

*Figure 6.13. Stored data mode - combined mean accuracy per number of chunks (window size = 20)*

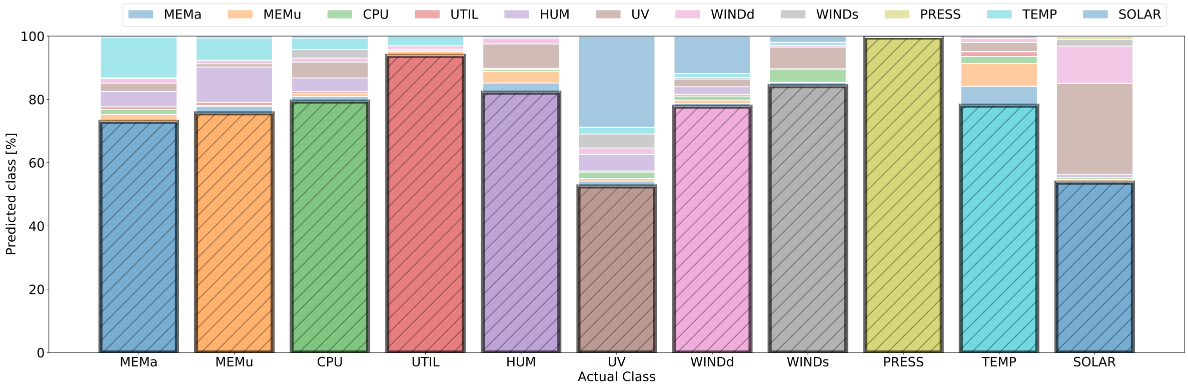


*Figure 6.14. Accuracy of mc RF stored data*

probable class, in 95% for two most probable classes and $> 98\%$ if 3 most probable classes are considered (Figure 6.13a). As for the streaming, the IF algorithm correctly classifies signals in $\sim 13\%$ cases and correctly classifies positive class in the first 6 classes in $\sim 70\%$ of cases (Figure 6.13b).

Therefore, there is no significant difference between streaming and stored data modes since both approaches result in $\sim 76\%$ for *mc* RF and $\sim 13\%$ for $1c$ IF model. Furthermore, the number of *chunks* is not important as well. Aggregated result depicted in Figures 6.11 and 6.13 have almost the same accuracy for the first and for the last *chunk*.

## 6.4 Discussion

In this research, we have evaluated and compared different approaches to model time series data for signal type classification in order to solve 3 problems; (1) automatic recognition of a signal type when a new sensor is added to an IoT network, (2) recognition of a streaming signal type in a case of configuration data loss due to unexpected communication issues and (3) classification of signal types in the case of unexpected data storage issues and thus loss of configuration data and/or metadata. A machine learning model is usually trained when setting up an IoT network. However, if the ML model is not pre-trained, it can be built from chunks of previously collected and stored data that are properly linked with metadata, or it can be built from scratch with completely new data arriving from known sensors.

In case there is an existing IoT network and the total number of signal types is fixed and known, *mc* RF is the most accurate option for signal classification, while $1c$ algorithm is *the only* option for new networks where the total number of signal types is unknown. Furthermore, the $2c$ classification is a compromise between $1c$ and *mc*. For example, if there are $k$ known classes but new classes are expected to be added, initial models can be trained for existing $k$ classes, and models for remaining $n-k$ classes can be trained afterwards. Thus, the remaining $n-k$ models will have information about previously added classes, while the initial $k$ models will only have information about each other's classes. However, all models can be retrained for each newly added class, but that is usually redundant. Optimally, only models that lose their accuracy when a new class is added should be retrained. Furthermore, cost-sensitive learning algorithms [136] that improve the classification performance of imbalanced classification by assigning different misclassification costs to the minority and the majority class samples should also be considered. Although the dataset used in this research is balanced, a cost sensitive learning algorithm can be considered in the future for $2c$ approach in order to assign different misclassification costs to the positive class and the negative class (remaining 10 classes). Additionally, different ML models can be used for $1c$ and $2c$ approaches, i.e., a $1c$ OCSVM can be used to recognise available memory (MEMa), while the remaining 10 classes can use a $1c$ IF.

The most complex part of the ML models training on our dataset was distinguishing between UV and solar radiation. On one hand, all computer resource utilization classes and humidity are calculated in the range of $0-100\%$. Nevertheless, all of those classes are correctly classified as their patterns are different. On the other hand, solar radiation and UV are similar in nature, since solar radiation contains UV as its component, along with the visible light, infrared, radio, X-rays, and gamma rays [137]. Furthermore, solar radiation and UV both depend on sunlight, i.e., although UV and solar radiation do not have the same range during the day, they both have value zero during the night. Night on average lasts for half of a day, making it much longer than selected $window = 20$. Thus, night readings (twenty zero values) are commonly classified as UV, for both UV and solar radiation. One of the reasons for such behaviour is the behaviour of Python's *max* function which selects a class with the highest predicted probability. In the case of two classes with the same probability, Python's *max* function chooses the first one in the list, i.e., UV in this case.

As shown in the Algotithm Robustness section, a window size does not have a significant effect on the algorithm accuracy. However, selecting a smaller window size reduces the energy consumption since the smaller volume of data is fed into the ML model. Furthermore, streaming and stored modes have similar results in terms of accuracy. Thus, accuracy is the same if the algorithm is executed with consecutive or random values.

Whereas *mc* RF has an accuracy of more than $75\%$, $1c$ IF has an accuracy of only $13\%$. However, both algorithms give additional information about $n$ most probable classes for the monitored signal type. If $n = 2$ most probable classes are considered, *mc* RF can eliminate the remaining 9 classes with $95\%$ accuracy, while correctly classifying $> 98\%$ of signals into $n = 3$ most probable classes. The same behaviour is applicable to $1c$ IF since it can eliminate 5 least probable classes with the accuracy of $> 70\%$. The latter behaviour is useful for new IoT systems where the total number of signal types is unknown and, thus, a *mc* model cannot be created.

However, there are still some limitations within this solution. Since sensors are expected to always send data at the same reading period, 1 minute in our case, the optimization of dynamically adapting the reading frequency is not applicable. Furthermore, there is also a limitation on the analysed number of signal classes, 11 in our case. Moreover, this research focuses on ML algorithms with default hyper-parameters to obtain preliminary results on the feasibility of such solutions and does not consider more complex ML algorithms nor hyper-parameter tuning. That being said, future research should include preparation of an algorithm that will not be limited by a fixed period between readings, investigating how a number of signal classes affects the accuracy of the algorithm and experimenting with more complex ML algorithms and their hyper-parameters.

Furthermore, in an IoT network, it is not uncommon for new sensor types to be introduced over time. However, collecting sufficient data to train a model that recognises the class of a new sensor can be time-consuming, expensive, or even unrealistic. To address this issue, transfer learning should be considered. Transfer learning is the process of transferring knowledge from a related source domain to the target learners in the target domain to improve their performance. In considered case, we propose use of the transfer learning for efficiently training models to recognise newly added sensors in IoT networks. Specifically, we leverage existing datasets that are similar, but not identical to the target domain to achieve improved performance in a shorter time and with less training data. Homogeneous transfer learning approaches are particularly suitable when domains are in the same feature space, allowing the use of a predictive model trained on a dataset drawn from a domain that is highly similar to the target domain. Our proposed methodology has the potential to accelerate the training of models in IoT networks and reduce the costs associated with collecting and labeling large amounts of training data [138, 139].

## 6.5 Summary

With the increase in number and size of Internet of Things systems, there is an ever-growing risk of (meta)data loss, as well as the maintenance overhead to mitigate such risks. The experts recognise three main challenges in this area that need to be tackled, namely: (1) downsizing the manual work required for configuring sensor networks, (2) recovering metadata in case of connection issues, malfunctions or malicious actions in sensor networks, (3) rebuilding metadata lost due to unexpected problems within a data storage. Fortunately, all three challenges can be tackled with an uniform solution, namely the signal type classification approach, which is able to match raw signal to an appropriate data type.

In order to find such uniform solution, in this research, three machine learning classification approaches, namely, one class, two class and multi class classification, are compared and evaluated. The results show that the most accurate multi class random forest algorithm can correctly classify signals in $\sim 75\%$ cases despite being trained on a *window* as short as 20 consecutive readings. Furthermore, the multi class random forest algorithm can eliminate 9 out of 11 classes with an accuracy of 95% by classifying signal into two most probable classes.

Finally, future steps in this field include preparing a similar algorithm that will not be limited by a fixed period, one minute in this case, between readings; investigating how a number of signal classes, 11 in this case, affects the accuracy of the algorithm; experimenting with more complex algorithms and hyper-parameter tuning; and applying transfer learning when adding new type of sensor.

# 7 Dissertation contribution

The main scientific contributions of this research include the development of new data volume optimization methods and the reconstruction of data lost due to those optimizations for IoT systems with a large number of sensors. More specifically:

1. A data collection method based on dynamic monitoring frequency that is used to reduce the amount of data in the sensor network within the concept of the Internet of Things
2. A method for predicting the real value of the observed phenomenon based on a limited part of the transient signal from the sensor that requires reaching operational conditions
3. A signal type classification method for metadata reconstruction and optimization of sensor configuration

All methods are validated on real data in a real environment.

# 8 Conclusion

The Internet of Things (IoT) is growing every day, along with the number of sensors collecting data and the amount of data itself. IoT sensors are typically battery-powered endpoints distributed across large areas. On the one hand, constant data collection over the short time intervals results in large energy consumption and short-lasting batteries. On the other hand, increase in the amount of stored data requires an increase in the number and size of data centers, and consequently, an increase in energy consumption.

Collected data can be filtered, aggregated, compressed, and/or correlated in order to reduce the amount of data that is transferred and stored, as well as the energy used by an entire IoT system. However, data must be collected before using any of the aforementioned data reduction techniques. Therefore, battery-powered IoT sensors are still constantly collecting data and consuming energy. Consequently, moving data reduction closer to the edges of an IoT network reduces the amount of data circulating through the network; however, it still does not reduce the energy consumption of sensors.

The main goal of this research is to locate places in IoT network where data velocity and volume reductions can be made, while maintaining the value and variety of the data. Through the research, we addressed three challenges and proposed three contributions: the first two are concerned about reducing sensor online time and thus extending battery life, while the third proposes a method for mitigating the risks introduced by reducing the amount of data circulating through the IoT network, as well as the simplification of the configuration process when adding a new sensor to an IoT network.

The first part of the research proposes and systematically analyses two approaches for reading sensor data using dynamic monitoring frequency (DMF). The first approach is founded on a statistical distribution of changes between consecutive readings, while the second approach employs machine learning techniques to predict upcoming periods during which the sensor data will experience acceptable change. Both algorithms are trained on data that has been collected over the first two years and later evaluated on the third year. Results indicate that the statistical DMF algorithm gives better results for higher frequencies, while the machine learning DMF gives better results for lower frequencies. The biggest advantage of the statistical DMF is that it does not limit the range of future periods. We also show that significant accuracy can be achieved with shorter datasets, i.e., when using training dataset of 1 or 6 months instead of full dataset that contains 2 years data.

Through the second part of the research, we have used an LSTM neural network to predict the actual values of gas concentration from the transient of MQ-2 gas sensor acquired during its preheat period. The key aspects of the research are the examination of MQ-2 sensor behaviour and the development of a methodology for building a prediction model using LSTM algorithms. Furthermore, this methodology is applied to two other low-cost gas sensors, namely the MQ-5 and MQ-6 gas sensors. The results show that only the beginning of the transient can be collected, and the actual value can be predicted on the gateway side using an LSTM neural network in the case of sensors that require heating up before collecting the actual value. This way, the sensor has to be online and collect data only for 20 seconds before sleeping for 120 seconds.

In the third part of the research, three different machine learning classification approaches for signal type recognition are compared and evaluated. Signal type classification is single solution that can tackle challenges recognised by the researchers; automatic configuration of sensors newly added to an IoT network and time series data classification when configuration metadata is lost due to malfunctions of the network or data storage. One class, two class, and multi class classification approaches were considered in order to achieve this goal. The results show that the most accurate multi class random forest algorithm can correctly classify signals in $\sim 75\%$ cases and eliminate 9 out of 11 classes with an accuracy of 95%.

Next steps in the field can be divided in three parts, one for each part of the research.

- Implementation of an online DMF algorithm. Such an algorithm would work in two phases. In first phase it collects data in very short periods and uses collected data for self-training. The second phase starts when significant accuracy is reached. During the second phase, the algorithm would apply gained knowledge and predict future periods. These two phases can interchange and algorithm can update its knowledge periodically.

- Investigating the energy consumption in a terms of data transmission when sending transient data to the cloud for data prediction. Furthermore, the future step is to implement a machine learning algorithm on Arduino Uno board and investigate its energy consumption compared to the energy savings achieved with 120 seconds sleep period.

- Preparing a signal type recognition algorithm that will not be limited by a fixed period, one minute in this case, between readings; investigating how a number of signal classes, 11 in this case, affects the accuracy of the algorithm; experimenting with more complex algorithms and hyper-parameter tuning; and applying transfer learning when adding new type of sensor.

# BIBLIOGRAPHY

[1] H. Arghandabi, A comparative study of machine learning algorithms for the prediction of heart disease, *International Journal for Research in Applied Science and Engineering Technology*, 8, 12, 677–683, Dec. 2020.

[2] C. Wang, Y. Sun, W. Wang, H. Liu and B. Wang, Hybrid intrusion detection system based on combination of random forest and autoencoder, *Symmetry*, 15, 3, 2023.

[3] A. M. Musolf, E. R. Holzinger, J. D. Malley and J. E. Bailey-Wilson, What makes a good prediction? feature importance and beginning to open the black box of machine learning in genetics, *Human Genetics*, 141, 9, 1515–1528, Dec. 2021.

[4] A. Géron, *"Hands-On Machine Learning with Scikit-Learn and TensorFlow"*, O'Reilly, Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 1st edn., 2017.

[5] P. Perera, P. Oza and V. Patel, One-class classification: A survey, 01 2021.

[6] G. Huang, J. Chen and L. Liu, One-class SVM model-based tunnel personnel safety detection technology, *Applied Sciences*, 13, 3, 1734, Jan. 2023.

[7] Y. Regaya, F. Fadli and A. Amira, Point-denoise: Unsupervised outlier detection for 3d point clouds enhancement, *Multimedia Tools and Applications*, 80, 18, 28161–28177, May 2021.

[8] Ali Zaidi, Andres Barnneby, Ala Nazari, Marie Hogan and Christian Kuhlins, Whitepaper on Cellular IoT in the 5G era, Tech. rep., Ericsson, 2020, available.

[9] M. H. Ali Zaidi, Yasir Hussain and C. Kuhlins, Whitepaper on Cellular IoT Evolution for Industry Digitalization, Tech. rep., Ericsson, 2019, available.

[10] Ericsson, Cellular IoT Evolution for Industry Digitalization, Tech. rep., Ericsson, 2019, Available: https://www.ericsson.com/en/white-papers/cellular-iot-evolution-for-industry-digitalization.

[11] V. Mushunuri, A. Kattepur, H. K. Rath and A. Simha, Resource optimization in fog enabled iot deployments, *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 6–13, 2017.

[12] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris and J. Henkel, Computation offloading and resource allocation for low-power iot edge devices, *2016 IEEE 3rd World Forum on Internet of Things*, 2016.

[13] P. Buonadonna, D. Gay, J. Hellerstein, W. Hong and S. Madden, Task: sensor network in a box, *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, 133–144, 2005.

[14] IDC, Worldwide Global DataSphere IoT Device and Data Forecast, 2020–2024, Tech. rep., IDC, 2019, Available: https://www.idc.com/getdoc.jsp?containerId=prUS45213219.

[15] J. G. D. Reinsel and J. Rydning , The Digitization of the World From Edge to Core, *IDC Whitepaper*, 2018.

[16] P. Derbeko, S. Dolev, E. Gudes and J. D. Ullman, Concise essence-preserving big data representation, *2016 IEEE International Conference on Big Data (Big Data)*, 3662–3665, 2016.

[17] S. Kumar and V. K. Chaurasiya, A Strategy for Elimination of Data Redundancy in Internet of Things (IoT) Based Wireless Sensor Network (WSN), *IEEE Systems Journal*, 13, 2, 1650–1657, June 2019.

[18] R. Radha and M. Muralidhara, Removal of Redundant and Irrelevant Data From Training Datasets Using Speedy Feature Selection Method, *International Journal of Computer Science and Mobile Computing (IJCSMC)*, 5, 359–364, July 2016.

[19] B. Kerr, Big Data: The 5 Vs Everyone Must Know, `https://www.linkedin.com/pulse/20140306073407-64875646-big-data-the-5-vs-everyone-must-know`, [Accessed December 15th 2022.].

[20] A. Moon, J. Kim, J. Zhang and S. W. Son, Lossy compression on iot big data by exploiting spatiotemporal correlation, *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 1–7, 2017.

[21] A. Papageorgiou, B. Cheng and E. Kovacs, Real-time data reduction at the network edge of internet-of-things systems, *2015 11th International Conference on Network and Service Management (CNSM)*, 284–291, 2015.

[22] T. Shu, J. Chen, V. K. Bhargava and C. W. de Silva, An Energy-Efficient Dual Prediction Scheme Using LMS Filter and LSTM in Wireless Sensor Networks for Environment Monitoring, *IEEE Internet of Things Journal*, 6, 4, 6736–6747, Aug 2019.

[23] I. Ez-Zazi, M. Arioua and A. El Oualkadi, Adaptive joint lossy source-channel coding for multihop iot networks, *Wireless Communications and Mobile Computing*, 2020, 1–15, 05 2020.

[24] J. Čulić Gambiroža and T. Mastelic, Big Data Challenges and Trade-offs in Energy Efficient Internet of Things systems, *26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 1–6, Sep. 2018.

[25] E. I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith and R. Rednic, Edge mining the internet of things, *IEEE Sensors Journal*, 13, 10, 3816–3825, 2013.

[26] S. A. Al-Qaseemi, H. A. Almulhim, M. F. Almulhim and S. R. Chaudhry, Iot architecture challenges and issues: Lack of standardization, *2016 Future Technologies Conference (FTC)*, 731–738, 2016.

[27] J. C. Gambiroza, T. Mastelic, P. Solic and M. Cagalj, Capacity in lorawan networks: Challenges and opportunities, *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, 1–6, 2019.

[28] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo and C. Savaglio, Enabling iot interoperability through opportunistic smartphone-based mobile gateways, *Journal of Network and Computer Applications*, 81, 74–84, 2017.

[29] A. Mukherjee, Energy efficiency and delay in 5g ultra-reliable low-latency communications system architectures, *IEEE Network*, 32, 2, 55–61, 2018.

[30] R. Han, F. Zhang, L. Y. Chen and J. Zhan, Work-in-progress: Maximizing model accuracy in real-time and iterative machine learning, *2017 IEEE Real-Time Systems Symposium (RTSS)*, 351–353, 2017.

[31] M. Aazam, S. Zeadally and K. A. Harras, Fog computing architecture, evaluation, and future research directions, *IEEE Communications Magazine*, 56, 5, 46–52, 2018.

[32] K. B. Andrew Banks, Ed Briggs and R. Gupta, MQTT Version 5.0, `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf`, March 2019, [Accessed December 15th 2022.].

[33] J. Čulić Gambiroža, T. Mastelić, I. Nižetić Kosović and M. Čagalj, Dynamic monitoring frequency for energy-efficient data collection in internet of things, *Journal of Computational Science*, 64, 101842, 2022.

[34] Z. Jia, X. Lyu, W. Zhang, R. P. Martin, R. E. Howard and Y. Zhang, Continuous Low-Power Ammonia Monitoring Using Long Short-Term Memory Neural Networks, *The 16th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 1–6, 2018.

[35] T. Kuny, The digital dark ages? Challenges in the preservation of electronic information, 1998.

[36] E. Tonkin, G. Tourte and A. Gill, *Crowd mining applied to preservation of digital cultural heritage*, 1 of *Cultural Computing*, 115–136, Springer International Publishing AG, Switzerland, 2018.

[37] L. Johnston, Challenges in preservation and archiving digital materials, *Information Services & Use*, 40, 193–199, 2020, 3.

[38] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos and P. Christen, Sensor discovery and configuration framework for the internet of things paradigm, *2014 IEEE World Forum on Internet of Things (WF-IoT)*, 94–99, 2014.

[39] S. S. Madsen, A. Q. Santos and B. N. Jørgensen, A QR code based framework for auto-configuration of IoT sensor networks in buildings, *Energy Informatics*, 4, 2, 46, Sep 2021.

[40] C. Perera, P. Jayaraman, A. Zaslavsky, P. Christen and D. Georgakopoulos, Dynamic configuration of sensors using mobile sensor hub in internet of things paradigm, *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 473–478, 2013.

[41] K. Ni, R. Nithya, M. Chehade, L. Balzano, S. Nair, S. Zahedi, E. Kohler, G. Pottie, M. Hansen and M. Srivastava, Sensor network data fault types, *TOSN*, 5, 05 2009.

[42] H. Cai, B. Xu, L. Jiang and A. V. Vasilakos, Iot-based big data storage systems in cloud computing: Perspectives and challenges, *IEEE Internet of Things Journal*, 4, 1, 75–87, 2017.

[43] O. B. Sezer, E. Dogdu and A. M. Ozbayoglu, Context-aware computing, learning, and big data in internet of things: A survey, *IEEE Internet of Things Journal*, 5, 1, 1–27, 2018.

[44] P. N. Sawadogo and J. Darmont, On data lake architectures and metadata management, *Journal of Intelligent Information Systems*, 56, 1–24, 02 2021.

[45] F. Ravat and Y. Zhao, Metadata management for data lakes, T. Welzer, J. Eder, V. Podgorelec, R. Wrembel, M. Ivanović, J. Gamper, M. Morzy, T. Tzouramanis, J. Darmont and A. Kamišalić Latifić, editors, *New Trends in Databases and Information Systems*, 37–44, Springer International Publishing, Cham, 2019.

[46] K. Tidriri, N. Chatti, S. Verron and T. Tiplica, Bridging data-driven and model-based approaches for process fault diagnosis and health monitoring: A review of researches and future challenges, *Annual Reviews in Control*, 42, 63–81, 2016.

[47] N. McClure, *"TensorFlow Machine Learning Cookbook"*, Packt Publishing Ltd., Livery Place, 35 Livery Street, Birmingham B3 2PB, UK, 1st edn., 2017.

[48] I. J. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, USA, 2016, http://www.deeplearningbook.org.

[49] R. Atienza, *Advanced Deep Learning with Keras*, Packt Publishing, 2018.

[50] M.-. G. Sensor, MQ-2 Semiconductor Sensor for Combustible Gas, https://www.pololu.com/file/0J309/MQ2.pdf, [Accessed March 29th 2023.].

[51] M.-. G. Sensor, Technical Data MQ-5 Gas Sensor, https://files.seeedstudio.com/wiki/Grove-Gas_Sensor-MQ5/res/MQ-5.pdf, [Accessed March 29th 2023.].

[52] M.-. G. Sensor, MQ-6 Semiconductor Sensor for LPG, https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-6.pdf, [Accessed March 29th 2023.].

[53] C. Bellinger, S. Sharma and N. Japkowicz, One-class versus binary classification: Which and when?, *2012 11th International Conference on Machine Learning and Applications*, 2, 102–106, 2012.

[54] K. Hempstalk and E. Frank, Discriminating against new classes: One-class versus multi-class classification, W. Wobcke and M. Zhang, editors, *AI 2008: Advances in Artificial Intelligence*, 325–336, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[55] T. Mastelic and I. Brandic, Data velocity scaling via dynamic monitoring frequency on ultrascale infrastructures, *2015 IEEE 7th International Conference on Cloud Computing Technology and Science*, 2015.

[56] D. Trihinas, G. Pallis and M. D. Dikaiakos, Adam: An adaptive monitoring framework for sampling and filtering on iot devices, *2015 IEEE International Conference on Big Data (Big Data)*, 717–726, 2015.

[57] C. Arendt, S. Böcker and C. Wietfeld, Data-driven model-predictive communication for resource-efficient iot networks, *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, 1–6, 2020.

[58] C. Cecchinel, F. Fouquet, S. Mosser and P. Collet, Leveraging live machine learning and deep sleep to support a self-adaptive efficient configuration of battery powered sensors, *Future Generation Computer Systems*, 92, 225–240, 2019.

[59] D. Zordan, M. Rossi and M. Zorzi, Rate-distortion classification for self-tuning iot networks, *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, 857–863, 2017.

[60] A. M. bin Baharudin, M. Saari, P. Sillberg, P. Rantanen, J. Soini and T. Kuroda, Low-energy algorithm for self-controlled wireless sensor nodes, *2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)*, 42–46, 2016.

[61] M. Liyanage, C. Chang and S. N. Srirama, Energy-efficient mobile data acquisition using opportunistic internet of things gateway services, *2016 IEEE International Conference on Internet of Things and IEEE Green Computing and Communications and IEEE Cyber, Physical and Social Computing and IEEE Smart Data*, 217–222, 2016.

[62] V. Natarajan and A. Vyas, Power efficient compressive sensing for continuous monitoring of ecg and ppg in a wearable system, *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 336–341, 2016.

[63] K. Wang, Y. Wang, Y. Sun, S. Guo and J. Wu, Green industrial internet of things architecture: An energy-efficient perspective, *IEEE Communications Magazine*, 54, 12, 48–54, 2016.

[64] P. Khandel, A. H. Rassafi, V. Pourahmadi, S. Sharifian and R. Zheng, Sensordrop: A reinforcement learning framework for communication overhead reduction on the edge, *CoRR*, abs/1910.01601, 2019.

[65] C. Razafimandimby, V. Loscrí, A. M. Vegni and A. Neri, A bayesian and smart gateway based communication for noisy iot scenario, *2017 International Conference on Computing, Networking and Communications (ICNC)*, 481–485, 2017.

[66] T. Soultanopoulos, S. Sotiriadis, E. G. M. Petrakis and C. Amza, Data management of sensor signals for high bandwidth data streaming to the cloud, *2016 IEEE 37th Sarnoff Symposium*, 53–58, 2016.

[67] C. Ge, Z. Sun, N. Wang, K. Xu and J. Wu, Energy management in cross-domain content delivery networks: A theoretical perspective, *IEEE Transactions on Network and Service Management*, 11, 3, 264–277, 2014.

[68] S. Meng and L. Liu, Enhanced monitoring-as-a-service for effective cloud management, *IEEE Transactions on Computers*, 62, 9, 1705–1720, 2013.

[69] A. Eliseev, D. Kachalov and M. Farhadov, Modern methods to collect, store, and process big data in large-scale systems, 179–182, 11 2017.

[70] I. N. Kosovic, T. Mastelic and D. Ivankovic, Using artificial intelligence on environmental data from internet of things for estimating solar radiation: Comprehensive analysis, *Journal of Cleaner Production*, 266, 2020.

[71] D. L. Osorio-Arrieta, J. L. Muñoz-Mata, G. Beltrán-Pérez, J. Castillo-Mixcóatl, C. O. Mendoza-Barrera, V. Altuzar-Aguilar and S. Muñoz-Aguirre, Reduction of the measurement time by the prediction of the steady-state response for quartz crystal microbalance gas sensors, *Sensors*, 18, 8, 2018.

[72] Q. Zhang, S. Li, W. Tang and X. Guo, Fast measurement with chemical sensors based on sliding window sampling and mixed-feature extraction, *IEEE Sensors Journal*, 20, 8740–8745, 2020.

[73] I. Lujic, V. De Maio and I. Brandic, Efficient edge storage management based on near real-time forecasts, *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 21–30, 2017.

[74] L. Salhi, T. Silverston, T. Yamazaki and T. Miyoshi, Early Detection System for Gas Leakage and Fire in Smart Home Using Machine Learning, *2019 IEEE International Conference on Consumer Electronics (ICCE)*, 1–6, Jan 2019.

[75] G. A. Susto, A. Cenedese and M. Terzi, Chapter 9 - time-series classification methods: Review and applications to power systems data, R. Arghandeh and Y. Zhou, editors, *Big Data Application in Power Systems*, 179–220, 2018.

[76] P. Geurts, Pattern extraction for time series classification, *PKDD*, 2001.

[77] S. Papadimitriou, J. Sun and C. Faloutsos, Streaming pattern discovery in multiple time-series, *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, 697–708, 2005.

[78] Z. Wang, W. Yan and T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, *2017 International Joint Conference on Neural Networks (IJCNN)*, 1578–1585, 2017.

[79] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar and P.-A. Muller, Deep learning for time series classification: a review, *Data Mining and Knowledge Discovery*, 33, 4, 917–963, Jul 2019.

[80] A. Jastrzebska, Time series classification through visual pattern recognition, *Journal of King Saud University - Computer and Information Sciences*, 2019.

[81] B. Lamrini, A. Gjini, S. Daudin, F. Armando, P. Pratmarty and L. Travé-Massuyès, Anomaly detection using similarity-based one-class svm for network traffic characterization, 08 2018.

[82] B. Zhao, H. Lu, S. Chen, J. Liu and D. Wu, Convolutional neural networks for time series classification, *Journal of Systems Engineering and Electronics*, 28, 1, 162–169, 2017.

[83] M. R. Shahid, G. Blanc, Z. Zhang and H. Debar, Iot devices recognition through network traffic analysis, *2018 IEEE International Conference on Big Data (Big Data)*, 5187–5192, 2018.

[84] Z. Chen, Y. Liu and S. Liu, Mechanical state prediction based on LSTM neural netwok, *2017 36th Chinese Control Conference (CCC)*, 3876–3881, July 2017.

[85] Y. Wang, J. Zhou, K. Chen, Y. Wang and L. Liu, Water quality prediction method based on LSTM neural network, *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 1–5, Nov 2017.

[86] L. Yu, J. Chen and G. Ding, Spectrum prediction via long short term memory, *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, 643–647, Dec 2017.

[87] S. Liu, G. Liao and Y. Ding, Stock transaction prediction modeling and analysis based on LSTM, *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2787–2790, May 2018.

[88] W. Yao, P. Huang and Z. Jia, Multidimensional LSTM Networks to Predict Wind Speed, *2018 37th Chinese Control Conference (CCC)*, 7493–7497, July 2018.

[89] N. Kim, M. Kim and J. K. Choi, LSTM Based Short-term Electricity Consumption Forecast with Daily Load Profile Sequences, *2018 IEEE 7th Global Conference on Consumer Electronics (GCCE)*, 136–137, Oct 2018.

[90] F. Qian and X. Chen, Stock Prediction Based on LSTM under Different Stability, *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 483–486, April 2019.

[91] I. Boulanouar, S. Lohier, A. Rachedi and G. Roussel, Dta: Deployment and tracking algorithm in wireless multimedia sensor networks, *Ad-Hoc and Sensor Wireless Networks*, 28, 01 2015.

[92] I. Boulanouar, A. Rachedi, S. Lohier and G. Roussel, Energy-aware object tracking algorithm using heterogeneous wireless sensor networks, *2011 IFIP Wireless Days (WD)*, 1–6, 2011.

[93] A. A. Khan, M. H. Rehmani and A. Rachedi, Cognitive-radio-based internet of things: Applications, architectures, spectrum related functionalities, and future research directions, *IEEE Wireless Communications*, 24, 3, 17–25, 2017.

[94] B. Liu, J. Wang, S. Ma, F. Zhou, Y. Ma and G. Lu, Energy-efficient cooperation in mobile edge computing-enabled cognitive radio networks, *IEEE Access*, 7, 45382–45394, 2019.

[95] J. Brownlee, *Machine learning mastery with python: Understand your data, create accurate models and work projects end-to-end*, Jason Brownlee, 2021.

[96] J. M. Stanton, Galton, pearson, and the peas: A brief history of linear regression for statistics instructors, *Journal of Statistics Education*, 9, 3, Jan. 2001.

[97] J. Cramer, The origins of logistic regression, *SSRN Electronic Journal*, 2003.

[98] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, *Classification And Regression Trees*, Routledge, Oct. 2017.

[99] L. Breiman, *Machine Learning*, 45, 1, 5–32, 2001.

[100] T. K. Ho, The random subspace method for constructing decision forests, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20, 8, 832–844, 1998.

[101] V. N. Vapnik, Pattern recognition using generalized portrait method, *Automation and Remote Control*, 24, 774–780, 1963.

[102] E. Fix and J. L. Hodges, Discriminatory analysis. nonparametric discrimination: Consistency properties, *International Statistical Review / Revue Internationale de Statistique*, 57, 3, 238, Dec. 1989.

[103] T. Cover and P. Hart, Nearest neighbor pattern classification, *IEEE Transactions on Information Theory*, 13, 1, 21–27, 1967.

[104] sklearn, sklearn KDTree, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html, [Accessed April 19th 2023.].

[105] sklearn, sklearn BallTree, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html, [Accessed April 19th 2023.].

[106] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics*, 5, 4, 115–133, Dec. 1943.

[107] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain., *Psychological Review*, 65, 6, 386–408, 1958.

[108] D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Nature*, 323, 6088, 533–536, Oct. 1986.

[109] S. Hochreiter and J. Schmidhuber, Long Short-Term Memory, *Neural Computation*, 9, 8, 1735–1780, 11 1997.

[110] M. M. Moya and D. R. Hush, Network constraints and multi-objective optimization for one-class classification, *Neural Networks*, 9, 3, 463–474, 1996.

[111] N. Seliya, A. A. Zadeh and T. M. Khoshgoftaar, A literature review on one-class classification and its potential applications in big data, *Journal of Big Data*, 8, 1, Sep. 2021.

[112] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola and R. C. Williamson, Estimating the Support of a High-Dimensional Distribution, *Neural Computation*, 13, 7, 1443–1471, 07 2001.

[113] F. T. Liu, K. M. Ting and Z.-H. Zhou, Isolation forest, *2008 Eighth IEEE International Conference on Data Mining*, 413–422, 2008.

[114] L. Breiman, Random forests, *Machine Learning*, 45, 1, 5–32, 2001.

[115] J. Čulić Gambiroža, T. Mastelić, T. Kovačević and M. Čagalj, Predicting low-cost gas sensor readings from transients using long short-term memory neural networks, *IEEE Internet of Things Journal*, 7, 9, 2020.

[116] C. Baratto, F. Rigoni, G. Faglia, E. Comini, D. Zappa and G. Sberveglieri, ZnO and SnO2one-dimensional sensors for detection of hazardous gases, *2017 IEEE SENSORS*, 1–3, Oct 2017.

[117] L. Scholz, A. O. Perez, B. Bierer, P. Eaksen, J. Wöllenstein and S. Palzer, Carbon dioxide sensor for mobile devices: A novel approach for low-power consuming, highly sensitive NDIR sensors, *2016 IEEE SENSORS*, 1–3, Oct 2016.

[118] S. Rademacher, K. Schmitt, M. Mengers and J. Wöllenstein, Sensor network with energy efficient and low-cost gas sensor nodes for the detection of hazardous substances in the event of a disaster, *2015 IEEE Topical Conference on Wireless Sensors and Sensor Networks (WiSNet)*, 59–61, Jan 2015.

[119] R. DS3231, RTC DS3231 Datasheet, https://datasheets.maximintegrated.com/en/ds/DS3231.pdf, [Accessed March 25th 2023.].

[120] M. IRFZ44N, MOSFET IRFZ44N Datasheet, https://datasheet.octopart.com/IRFZ44N-Inchange-Semiconductor-datasheet-15981338.pdf, [Accessed March 25th 2023.].

[121] M. Kuhn and K. Johnson, *Applied predictive modeling*, Springer, New York, NY, 2013.

[122] T. Bouguera, Energy consumption model for sensor nodes based on lora and lorawan, *Sensors*, 18, 06 2018.

[123] M. Al-Shorman, M. Al-Kofahi and O. Al-Kofahi, A practical microwatt-meter for electrical energy measurement in programmable devices, *Measurement and Control*, 51, 08 2018.

[124] D. Singh, O. G. Aliu and M. Kretschmer, Lora wan evaluation for iot communications, *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 163–171, Sep. 2018.

[125] L. Müller, M. Mohammed and J. W. Kimball, Using the arduino uno to teach digital control of power electronics, *2015 IEEE 16th Workshop on Control and Modeling for Power Electronics (COMPEL)*, 1–8, 2015.

[126] M. Zouheir, M. Zniber, S. Qudsia and T.-P. Huynh, Real-time humidity sensing by integration of copper sulfide nanocomposite with low-cost and wireless arduino platform, *Sensors and Actuators A: Physical*, 319, 112541, 2021.

[127] Arduino, ArduinoLibraries, https://www.arduino.cc/reference/en/libraries/, [Accessed March 14th 2023.].

[128] Arduino, SmoothADC Arduino Library, https://www.arduino.cc/reference/en/libraries/smoothadc/, [Accessed March 14th 2023.].

[129] Arduino, Oversampling Arduino Library, https://www.arduino.cc/reference/en/libraries/oversampling/, [Accessed March 14th 2023.].

[130] Arduino, OVS Arduino Library, https://www.arduino.cc/reference/en/libraries/ovs/, [Accessed March 14th 2023.].

[131] TensorFlow, TensorFlow Lite for Microcontrollers, https://www.tensorflow.org/lite/microcontrollers/overview, [Accessed March 27th 2023.].

[132] L. Dujić Rodić, I. Stančić, K. Zovko, T. Perković and P. Šolić, Tag estimation method for aloha rfid system based on machine learning classifiers, *Electronics*, 11, 16, 2022.

[133] J. Č. Gambiroža, T. Mastelić, I. N. Kosović and M. Čagalj, Lost in data: recognizing type of time series sensor data using signal pattern classification, *International Journal of Data Science and Analytics*, Jul. 2023.

[134] M. Fernández-Delgado, E. Cernadas, S. Barro and D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, *J. Mach. Learn. Res.*, 15, 1, 3133–3181, jan 2014.

[135] G. Zaccone and M. Karim, *Deep Learning with TensorFlow - Second Edition*, 03 2018.

[136] F. Feng, K.-C. Li, J. Shen, Q. Zhou and Y. Xuhui, Using cost-sensitive learning and feature selection algorithms to improve the performance of imbalanced classification, *IEEE Access*, PP, 1–1, 04 2020.

[137] S. Bhatia, 2 - solar radiations, S. Bhatia, editor, *Advanced Renewable Energy Systems*, 32–67, 2014.

[138] K. Weiss, T. M. Khoshgoftaar and D. Wang, A survey of transfer learning, *Journal of Big Data*, 3, 1, 9, May 2016.

[139] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong and Q. He, A comprehensive survey on transfer learning, *Proceedings of the IEEE*, 109, 1, 43–76, 2021.

# Curriculum Vitae

### Jelena Čulić Gambiroža

Jelena Čulić Gambiroža was born on July 15th 1991, in Split, Croatia. After finishing high school in Split in 2010, she enrolled to the Computer Engineering/Computer Science studies at the Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture at the University of Split. She received MSc. degree with excellent marks in 2015, with thesis led by mentor prof. dr. sc. Mario Čagalj. She is the recipient of the Rector's Award for Excellence in the academic year 2014-2015.

In February 2015 she has been employed as a student helper at the company Ericsson Nikola Tesla. In July of the same year she started working as a Software Engineer for the company. Since November 2018, she has been employed as a Researcher, and she is enrolled in PhD studies at the Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture at the University of Split as Ericsson Nikola Tesla company scholarship holder. In September 2023 she started a new position as Teaching and Research Assistant at Faculty of Maritime Studies, University of Split.

Since 2015 she is also employed as university associate at FESB, where she participated as teaching assistant in multiple courses, including 1st year programming in C, basic audio/image processing in Matlab, IP communications and software engineering. Furthermore, she has been mentor and organizer for Ericsson Nikola Tesla Summer Camp from 2019.

Her scientific interests include big data challenges in Internet of Things as well as data analytics, including statistical methods and machine learning. During her PhD studies, she published three (3) scientific papers in A category journals and two (2) full papers in proceedings with international review. She has also published posters and abstracts. Six (6) of above mentioned publications are related to her PhD dissertation.

# Životopis

## Jelena Čulić Gambiroža

Jelena Čulić Gambiroža rođena je 15. srpnja 1991. godine u Splitu. Nakon završene 3. gimnazije u Splitu, 2010. godine upisuje studij Računarstva na Fakultetu elektrotehnike, strojarstva i brodogradnje u Splitu. Diplomirala je s izvrsnim uspjehom u lipnju 2015. godine pod mentorstvom profesora Marija Čaglja. Dobitnica je Rektorove nagrade za izvrsnost u akademskoj godini 2014./2015.

Od veljače 2015. je zaposlena kao student u kompaniji Ericsson Nikola Tesla, a od srpnja iste godine radi kao inženjer za razvoj softvera u istoj kompaniji. Od studenog 2018. godine radi kao istraživač u kompaniji te kao stipendist kompanije upisuje poslijediplomski studij Elektrotehnike i informacijske tehnologije na Fakultetu elektrotehnike, strojarstva i brodogradnje u Splitu. U rujnu 2023. godine se zaposlila kao asistent na Pomorskom fakultetu u Splitu.

Od završetka studija 2015. godine radi i kao vanjski suradnik-asistent na FESB-u gdje je u okviru nastavnih djelatnosti sudjelovala u nastavi iz predmeta Računala i programiranje, Multimedija/Multimedijski sustavi, Transmisijski sustavi, Inženjerska grafika i prezentacija, IP komunikacije i Programsko inženjerstvo. Također je aktivno sudjelovala u mentoriranju studenata i organizaciji Ericsson Nikola Tesla ljetnog kampa od 2019. godine.

Znanstveni i stručni interes uključuje razvoj i obradu podataka korištenjem pristupa temeljenog na podatcima, s naglaskom na primjenu strojnog učenja, te optimizaciju količine podataka u paradigmi Interneta stvari. Tijekom studija objavila je tri (3) znanstvena rada u časopisima A kategorije, dva (2) rada u zbornicima s međunarodnom recenzijom, te stručne radove i sažetke u zbornicima skupova. Od navedenih, šest (6) je radova vezano za područje disertacije.