Comprehensive Machine and Deep Learning Fault Detection and Classification Approaches of Industry 4.0 Mechanical Machineries: With Application to A Hydraulic Test Rig

DISSERTATION

Zur Erlangung des Grades eines Doktors
der Ingenieurwissenschaften.

vorgelegt von

M.Sc. Ahlam Mallak

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät
der Universität Siegen
Siegen 2021

Betreuer und erster Gutachter
Prof. Dr.-Ing. habil. Madjid Fathi
Universität Siegen

Zweiter Gutachter
Prof. Dr.-Ing. habil. Roman Obermaisser
Universität Siegen

Mitglieder der Promotionskommission
Prof. Dr. rer. nat. Volker Blanz
Universität Siegen

Vorsitzender: Prof. Dr. Markus Lohrey

Tag der mündlichen Prüfung
15. März 2021

gedruckt auf alterungsbeständigem holz- und säurefreiem Papier

Dedication: *I dedicate this work to my family and my partner for always supporting me and standing by my side. I love you all more than anything, and I cannot thank you enough!*

Acknowledgments

First, I would like to thank my first supervisor Prof. Madjid Fathi for always offering great support and understanding. You have played such an important role in my personal and academic development and growth. You have chosen me, supported me, believed in me, and pushed me to be the best academic I can be. Thank you so much for everything.

I would like to also thank my second supervisor, Prof. Roman Obermaisser for being a great support for the past 3 years, while working on our joint DFG project resulting a lot of work contributing to this dissertation.

A special appreciation and thanks for Dr. Daniel Müller who is the head of the House of Young Talents office in the University of Siegen, for the valuable courses your office offers. They were an eye-opener and a valuable source of information. Thank you as well for offering me a scholarship at the end of my doctorate that helped me a lot during my academic journey.

Another great academic I would like to thank is Dr. Nina Finn for being a great support and a role model, and a female academic to look up to.

Thanks to my colleagues for the continuous intrinsic and moral support. I hope we can work together again in the near future.

Thanks to my family: Dad, Mom, Rana, Bahjat, Omar and Maram. And the family that fate and I chose: Kellum O'Connor, for always standing by my side and loving me no matter what. Thank you for believing in me and pushing me to keep fighting even when I was at my lowest. I am rich for having you all in my life.

Above all, I am so thankful to God for giving me the strength to keep going. For sending the right people at the right time. For keeping me and my loved ones healthy and safe. For being so kind to me, even when I was not kind to myself. I love you so much, and I will always be thankful to you until the day I die.

Abstract

Anomaly occurrences in mechanical equipment within industry 4.0 may lead to massive systems shut down, jeopardizing the safety of the machinery and its surrounding human operator(s) and environment, as well as the severe economic implications succeeding the faults and their associated damage. Various mechanical tools are mostly placed in harsh and ruthless environments, where the machines are consistently vulnerable to many fault types connected to their functionality nature. Hence, not only the machines and their components are prone to anomalies, but also the sensors attached to them necessary to collect viable signals to monitor and report the overall machine health and behavioural changes. Those sensors may likewise fail and carry out various anomalies.

This thesis elucidates a full research and analytical implementation of component and sensor faults detection and diagnosis, utilizing numerous machine and deep learning approaches in application of a hydraulic system extracted from a hydraulic test rig. It is unfortunate that hydraulic systems are rarely approached for anomaly detection subject comparing to other mechanical machines in the past decade. Specifically, comprehensive systems that cover all aspects of anomaly detection in hydraulic systems, which includes both sensor and component faults, essential feature engineering methods, and innovative detection algorithms based on the latest technologies such as, the application of deep learning.

In this work, three main contributions to anomaly detection in hydraulic systems extracted from a hydraulic test rig are thoroughly achieved. Firstly, we provided a combination of LSTM autoencoders and supervised machine and deep learning methodologies to perform two separate stages of fault detection and diagnosis. The two phases are condensed by: (1) the detection phase using the LSTM autoencoder. Followed by (2) the fault diagnosis phase represented by the classification schema. The previously mentioned framework is applied to component and sensor faults in hydraulic systems, deployed in the form of two in-depth applicational experiments. In the detection phase declared by the classification process, diversified machine and deep learning supervised methods are compared and analysed for their component and sensor fault detection performance in hydraulic systems. In addition, we provided comparisons of plentiful feature engineering techniques in the time-domain, to showcase the influence of each feature engineering method on its corresponding supervised classifiers in the detection phase. Secondly, we provided an unprecedented feature selection method called Recursive *k*-means Silhouette Elimination (R*k*SE), and it is deployed to perform feature selection for component fault classification in multi-variate hydraulic test rig dataset. Moreover, R*k*SE is utilized as a window compression method when deployed to achieve sensor fault identification in univariate sliding window-structured datasets. Finally, an innovative application of Random Forests (RF) in a hybrid architecture between data-driven and model-based diagnosis approaches is introduced and applied to hydraulic systems for dynamic diagnostic rules generation.

Zusammenfassung

Das Auftreten von Anomalien in mechanischen Geräten innerhalb der Industrie 4.0 kann zu massiven Systemabschaltungen führen, die die Sicherheit der Maschinen und der sie umgebenden menschlichen Bediener und der Umwelt gefährden, sowie zu schwerwiegenden wirtschaftlichen Auswirkungen, die auf die Fehler und die damit verbundenen Schäden folgen. Verschiedene mechanische Werkzeuge werden meist in rauen und unbarmherzigen Umgebungen eingesetzt, in denen die Maschinen aufgrund ihrer Funktionsweise durchweg anfällig für viele Fehlertypen sind. Daher sind nicht nur die Maschinen und ihre Komponenten anfällig für Anomalien, sondern auch die an ihnen angebrachten Sensoren, die notwendig sind, um brauchbare Signale zur Überwachung und Meldung des allgemeinen Maschinenzustands und der Verhaltensänderungen zu sammeln. Diese Sensoren können ebenfalls ausfallen und verschiedene Anomalien hervorrufen.

Diese Arbeit erläutert eine vollständige Forschung und analytische Implementierung der Erkennung und Diagnose von Komponenten- und Sensorfehlern unter Verwendung zahlreicher maschineller und Deep-Learning-Ansätze in der Anwendung eines hydraulischen Systems, das aus einem hydraulischen Prüfstand stammt. Es ist bedauerlich, dass hydraulische Systeme im Vergleich zu anderen mechanischen Maschinen im letzten Jahrzehnt nur selten zum Thema Anomalieerkennung herangezogen werden. Insbesondere umfassende Systeme, die alle Aspekte der Anomalieerkennung in hydraulischen Systemen abdecken, was sowohl Sensor- als auch Komponentenfehler, wesentliche Feature-Engineering-Methoden und innovative Erkennungsalgorithmen auf Basis neuester Technologien wie die Anwendung von Deep Learning umfasst.

In dieser Arbeit werden drei Hauptbeiträge zur Anomalieerkennung in hydraulischen Systemen, die von einem hydraulischen Prüfstand extrahiert wurden, gründlich erreicht. Erstens haben wir eine Kombination aus LSTM-Auto-Encodern und überwachten Maschinen- und Deep-Learning-Methoden bereitgestellt, um zwei separate Phasen der Fehlererkennung und -diagnose durchzuführen. Die beiden Phasen werden zusammengefasst durch: (1) die

Erkennungsphase unter Verwendung des LSTM-Autoencoders. Gefolgt von (2) der Fehlerdiagnosephase, die durch das Klassifikationsschema repräsentiert wird. Das zuvor beschriebene Framework wird auf Komponenten- und Sensorfehler in hydraulischen Systemen angewandt, die in Form von zwei vertiefenden Anwendungsexperimenten eingesetzt werden. In der durch das Klassifikationsverfahren deklarierten Erkennungsphase werden diversifizierte maschinelle und Deep-Learning-überwachte Methoden verglichen und auf ihre Leistung bei der Erkennung von Komponenten- und Sensorfehlern in hydraulischen Systemen analysiert. Darüber hinaus haben wir Vergleiche zahlreicher Feature-Engineering-Techniken im Zeitbereich durchgeführt, um den Einfluss jeder Feature-Engineering-Methode auf die entsprechenden überwachten Klassifikatoren in der Erkennungsphase zu zeigen. Zweitens haben wir eine noch nie dagewesene Merkmalsauswahlmethode namens Recursive *k*-means Silhouette Elimination (R*k*SE) entwickelt, die zur Merkmalsauswahl für die Klassifizierung von Komponentenfehlern in einem multivariaten Hydraulikprüfstandsdatensatz eingesetzt wird. Darüber hinaus wird R*k*SE als Fensterkompressionsmethode eingesetzt, um eine Sensorfehleridentifikation in univariaten, gleitenden, fensterstrukturierten Datensätzen zu erreichen. Schließlich wird eine innovative Anwendung von Random Forests (RF) in einer hybriden Architektur zwischen datengetriebenen und modellbasierten Diagnoseansätzen vorgestellt und auf hydraulische Systeme zur dynamischen Generierung von Diagnoseregeln angewendet.

Table of Contents

List of Figures

List of Tables

List of Acronyms

FDD	Fault detection and diagnosis
ML	Machine learning
DL	Deep learning
LSTM	Long short-term memory
CNN	Convolutional neural networks
ANN	Artificial neural networks
RNN	Recurrent neural networks
RF	Random forests
CART	Classification and regression trees
SVM	Support vector machines
LR	Logistic regression
LDA	Linear discriminant analysis
KNN	k-nearest neighbour
NB	Naïve Bayesian
PCA	Principal component analysis
FI	Feature importance

Chapter 1: Introduction

1. Motivation

Mechanical machineries within industry 4.0 are considered a vital part of the industrial operation. Hence, they play a tremendous role in the production and manufacturing processes. Due to their major importance in the production line, mechanical devices are usually placed in tough locations and dangerous environments, which make them susceptive to the occurrence of various faults and malfunctions. Nowadays, the industrial applications are getting more complicated and scalable than ever, which contributed tremendously to the complexity of fault detection in mechanical machineries, as well as making those tasks quite challenging [1].

The study in [2] indicated that 70-90% of the incidents associated to the industrial operations are caused by human workers or operators. Consequently, the need for computer-aided diagnosis emerged, to ensure highly accurate fault detection, prediction, and diagnosis of systems with extreme complexities. Moreover, computer-aided diagnosis for mechanical machines may also contribute to the speed and precision of the recovery actions deployment required following the fault appearance.

The main goal of fault detection in mechanical machinery is to capture the anomalies accurately as soon as they manifest, to ensure deploying the necessary maintenance procedures, and to dodge economical, humanitarian, and environmental tragedies. Creating a solid fault detection and diagnosis systems, not only contribute to reducing the risk, and providing safety to human operators and the environment. But also, they play a major role in cutting down the costs related to unnecessary maintenance. Thereafter, fault detection and diagnosis in mechanical devices placed in complex systems like the industrial ones is always a hot research topic.

Automated FDD algorithms and systems are usually dependent on the training and analysis of datasets, in which they are extracted from numerous sensors attached to the industrial equipment and its components. Those sensors continuously send signals essential to monitor each component of the mechanical machine. In other words, sensor readings are the modalities, or the source of raw data associated to automated FDD systems. The health of these sensors is the key to monitor those components properly, which leads to accurate component diagnosis results. Although these sensors are substantial for computer-aided diagnosis, they are mostly ridiculously cheap and perform under extreme environmental conditions due to their attachment to the machinery device. Therefore, sensors in mechanical machines are prone to malfunctions and variety of faults themselves.

Smart FDD systems should monitor both the mechanical devices and their components, as well as the sensors' health status of those who are responsible of reporting the health indications of mechanical components. Hence, it is undeniably important to establish sensor FDD systems along with the component FDD ones when monitoring industrial operations.

One of the most essential mechanical equipment for industrial processes are hydraulic systems. The hydraulic system's data applied to this study is gathered from a hydraulic test rig. A test rig can be defined as a piece of mechanical devices that is mainly utilized to assess, evaluate, and test the capacities and performance of other mechanical machines, or just certain components of them. Test rigs can be called by various terminologies including test bench or test pay, and testing station. Test rigs are common in a wide range of industrial fields from hydraulic systems to aerospace. They literally have a vast scope of testing methods, and analytical parameters such as, manual, cyclical, brake and burst testing.

Hydraulic systems obtained from a hydraulic test rig are the focus of this dissertation, due to their importance and limited FDD resources in the past decade, in which a comprehensive FDD system of both component and sensor faults are included.

2. Problem Statement

As mentioned earlier, this work represents a thorough analysis and application of both sensor and component faults in hydraulic systems manifested by hydraulic test rigs. To ensure comprehension, three main perspectives of FDD in hydraulic systems are researched and analysed in this dissertation, which includes: (1) creating comprehensive FDD algorithms to cover sensor and component faults in hydraulic systems. (2) Establishing a new feature engineering method applicable to FDD in hydraulic systems. And finally, (3) creating dynamic diagnostic rules to aid the automation of existing model-based FDD methods in hydraulic systems. In which each aspect has its own problem statements and research questions. Therefore, the problem statement is divided into three main parts corresponding to each section in the thesis.

A. Problem Statement 1: Creating Comprehensive FDD Algorithms to Cover Sensor and Component Faults in Hydraulic Systems

Hydraulic systems in the industry require a non-stop monitoring, supervision and evaluation of the sensors' readings connected to each component of the system, most especially when they are prone to failure. In reality, sensor faults tend to be ungeneralised and highly system-specific with a wide range of characteristics and possibilities that vary even within the same system. That is the reason sensor fault detection is usually tackled manually by the system's expert who acquire the knowledge of each sensor and its characteristics, its healthy indictors, and parameters. Automated sensor fault FDD has attracted a lot of research and industrial attention in recent years. The accuracy and efficiency of automated FDD for Hydraulic systems is entirely dependent on the reliability of the sensors and their readings, since they are the raw data fed into the FDD systems. In other words, their health and validity determine the overall reliability and accuracy of the overall FDD system.

On a different note, hydraulic systems' components are also extremely prone to failure. To address this issue, it is necessary to develop sensor FDD algorithms along with the component FDD ones, to be able to distinguish the source of the behavioural deviation of the system, whether it is actually a deviation in a part of the hydraulic system or a malfunctioned sensor with a misleading reading.

When the data trained and utilized to build the FDD model is extracted from a hydraulic test rig, it has -by definition- observed labels and conditions, since the main goal of using hydraulic test rigs is to test and evaluate the condition of the system under experimentation. Therefore, the data-driven FDD methods of interest are the supervised approaches. Supervised ML approaches for FDD in hydraulic systems have shown significant and highly accurate results in the literature. However, most classification methods fail to capture new faults or rare occurrences beyond their trained labels, which makes it a huge drawback of supervised ML methods of FDD comparing to clustering or unsupervised ones, which urge the need to combine those methods to other ML and DL branches of data-driven diagnosis.

B. Problem Statement 2: Establishing A New Feature Engineering Method Applicable to FDD in Hydraulic Systems

Human brains can only visualize and imagine three-dimensional spaces. Data with larger dimensions outpaces the human capacity to understand and manually analyse such data. In accordance with the human incapacity to deal with high volume data, data mining is brought to light to discover highly dimensional patterns in this large data, offering new solutions to visualize, analyse and process the big influx of information. Although, machine learning offers a lot of algorithms and methods for big data analysis, finding relevant and non-redundant attributes in data that contains hundreds or thousands of attributes can be challenging.

It is important to mention that feature selection is performed on the data before it is fed to the FDD algorithms. Thus, the quality of feature selection reflects the expected outcome of the FDD algorithm and its precision. Irrelevant features in the input data may decrease the accuracy of the machine learning model learnt especially when they are many, because they can create an accumulative amount of deviation from the correct patterns. However, redundant features are the features that do not carry new information necessary for learning. So that, redundant features may not affect the learning process and its accuracy, but indeed will increase the computational cost required for performing such tasks [3]. Therefore, when the dimensionality of the data is high without investigation and analysis of their redundancy and relevance, this would weaken the quality of the data used for learning, increase its computational cost, and diminish its accuracy. Moreover, besides redundant, and irrelevant features, noisy features can also contribute to degrading the training performance of various learning algorithms.

Feature selection in data mining is the process of choosing a subset of the overall features (variables) in the feature space, by sacrificing the ones carrying little valuable information, unnecessarily redundant ones, and noisy features [4]. Feature selection is most appropriate for multivariate datasets owning to their nature of numerous numbers of attributes and samples. Some examples of datasets require feature selection due to their complexity and size demands are text, genetic information, imaging modalities and time-series data. Moreover, to achieve FDD in univariate systems, univariate time-series data can also benefit from feature selection, but only when it is structured in a window format, where the feature selection method is rather window compression, to select the most representative time points in the window. This side of feature selection is usually ignored by researchers, specifically when applying FDD in a sliding-window format.

Although, one might think the deployment of feature selection in any FDD task is completely optional, incorporating this step has a great potential to add-up many advantages. For instance, reducing the number of features can effectively improve the ability of data understanding and visualization, decrease the computational power and storage requirements needed, noticeably fasten the training and testing times. Finally, increasing the accuracy of the FDD model using smaller inputs and resources. Wherefore, feature selection is a substantial step to eliminate undesirable features, by selecting the ones that are more relevant, non-noisy and non-redundant [5] and [4].

Most of the feature selection algorithms utilized to aid FDD systems proposed in the literature are classification-based techniques [6], [7] and [8], where these methods are dependent on the presence of clear classes or labels to perform the feature selection accordingly. In the past few years, a new cluster-based also known as unsupervised feature selection algorithm emerged. Unsupervised feature selection methods work by grouping objects in various groups based on their similarity, where better clusters are the ones with higher within-cluster similarities and lower between-clusters similarities [9] and [10]. This group of feature selection methodology fits more to the unpredicted nature of data used for FDD in real life. One of the most famous clustering algorithms in data mining is *k*-means clustering. *k*-means depends on dividing the data between *k* main clusters, where the intra-similarity within the cluster and inter-similarity between clusters is measured using silhouette value measurement.

Although, the literature is enriched with numerous supervised learning feature selection methods, it is still scarce when it comes to unsupervised ones and it needs more investigation and research in this field especially when applied prior to data-driven FDD systems.

Recently, feature selection applied to FDD in mechanical systems research interest drastically shifted to unsupervised methods, mostly because of their strength in identifying relevant features without the need of existing class labels. *k*-means clustering is one of the most famous clustering algorithms deployed for feature selection due to its simplicity, the mount of existing research already done to accurately select various parameters of the algorithm. i.e., selecting the best *k* and initializing the seeds. Moreover, *k*-means is relatively easier to evaluate comparing to other clustering algorithms, since it has many clear measures to evaluate the quality of the clustering process such as the silhouette measure.

C. Problem Statement 3: Creating Dynamic Diagnostic Rules to Aid the Automation of Existing Model-Based FDD in Hydraulic Systems

Model-based approaches for FDD tend to have a handful of disadvantages such as, the lack of dynamicity and generality, since they exhibit static knowledge for a specific domain stored in the model. The lack of or absence in handling sudden or novel fault occurrences (hence they are not pre-stored in the reference model), and the inability to automatically detect, fill or update the system gaps. The lack of credibility in knowledge acquisition because it is completely dependent on experts' reliability. And finally, the impossibility to learn from misdiagnosis and fault occurrence overtime.

Although, data-driven approaches might offer dynamic and general-domain diagnostic alternatives comparing to their model-based counterparts, they tend to have their share of challenges. i.e., the dependency on the data in case of poor data collection or tending to invalid sources, the dependency on possessing certain skills to apply data-driven processing and analysis methods, storing data necessary for learning and testing is resource and security expensive, and the additional expenses related to the needed supplementary hardware purchases and regular maintenance.

According to the mentioned drawbacks of each approach, it is essential to establish a hybrid approach that combines the positive sides of each one and eliminates as many as possible of their limitations. In order to, provide a clear architecture that may aid existing model-based approaches for FDD in hydraulic systems to get dynamic and automated.

The algorithm in [11] demonstrates a model-based component FDD method based on using diagnostic graphs created by static/constant diagnostic rules extracted from semantic ontology. In other words, the system model represented by the Ontology [12] is fed directly with the expert knowledge, and later used to generate diagnostic graphs that links between various symptoms and their faults. The created graphs using the model-based approach alone are lacking the dynamicity and the generality, where they are only applicable to a certain system or model that they were created upon. Thus, a more general and dynamic approach is needed.

Creating dynamic diagnostic graphs using data-driven approaches such as Random Forest (RF) can be beneficial. However, because of their dynamic nature, these models are hard to use in structured or distributed systems without following some guidelines, graphs, or clear steps. Furthermore, data-driven approaches require more time and resources to process and store the needed data. Thus, a strong necessity to create a general domain, dynamic but structured enough algorithm, to guarantee general-domain application and the possibility of distributed computing if needed. Additionally, a decrease in the time and resource complexity constraints required by online data-driven approaches.

3. Research Questions

The following are the research questions investigated and answered in this dissertation:

1- How to create a supervised FDD method in hydraulic systems that is comprehensive enough to cover both sensor and component faults?
2- How to overcome the drawback of supervised FDD approaches in detecting and diagnosing the occurrence of rare faults in both sensor and component faults located in hydraulic systems?
3- How to create an unsupervised feature engineering method that can be applicable for both sensor and component FDD in hydraulic systems?
4- How to automate/dynamize the diagnostic rules in existing model-based approaches?

4. Our Contribution

A. Contribution 1: Sensor and Component FDD for Hydraulic Systems Using Combined LSTM Autoencoder Detector and Diagnostic Classifiers

In chapter 6, a comprehensive FDD approach for hydraulic systems is proposed, where an additional step is added in advance to the diagnosis using the classification phase, to overcome the weaknesses of supervised diagnostic approaches in capturing rare and beyond the existing labels faults.

To overcome this challenge, in this section the detection and diagnosis phase are performed separately. Where the detection phase is done by applying a LSTM autoencoder to detect rare and unprecedented faults. Followed by the diagnosis phase using the ML and DL classifiers to analyse the nature of the captured faults in phase one. This approach has been already created in the literature; however, our work is beyond the state-of-the-art by the following: (1) It is the first time this schema is applied to hydraulic test rigs data. (2) This work was applied to both sensor and component faults, in two thorough separate experiments. (3) We applied a unique architecture of the LSTM autoencoder utilized for the detection phase. (4) In the detection phase represented by the autoencoder, we presented a new criterion to calculate the deviation between the predicted signal and the input one, which is proven more effective to the traditional method in computing more accurate diagnostic thresholds. (5) In the fault diagnosis phase proposed by the classification, we provided a full comparison results between numerous ML and DL classifiers of different functionality and technique. (6) In the same phase, we also provided a behavioural analysis of each ML and DL classifier with a bunch of time-domain feature selection methods, to support further research in the future by mapping each classifier to their best or least suitable time-domain feature, to achieve either component or sensor FDD in hydraulic systems.

B. Contribution 2: Unsupervised Feature Selection using Recursive *k*-Means Silhouette Elimination (R*k*SE): A Two-Scenario Case Study for Fault Classification of High-Dimensional Sensor Data

Chapter 5 introduces a new feature selection algorithm that depends on recursively clustering the data into *k* groups using *k*-means clustering, then calculating the silhouette value for each member of the individual cluster to decide which feature is the representative for the rest of the cluster, and which are going to be re-clustered for further analysis.

For this section, the following are contributed to this thesis: (1) an in-depth review for feature selection algorithms based on *k*-means clustering is structured in related-work chapter, chapter 4. In addition, (2) a new taxonomy for feature selection algorithms using *k*-means clustering is presented. Furthermore, (3) a new feature selection algorithm based on the deployment of *k*-means and silhouette measure in an iterative fashion called "Recursive *k*-Means Silhouette Elimination (R*k*SE)" is introduced, tested, and validated. (4) R*k*SE is compared to various feature selection and extraction algorithms when applied prior to component and sensor fault classification using a bunch of ML and DL classifiers, performed in two separate experiments.

C. Contribution 3: A Hybrid Approach: Dynamic Diagnostic Graphs for Sensor Systems Generated by Online Hyperparameter Tuned Random Forest

In chapter 7, a novel architecture to dynamize fixed-ruled model-based diagnostic systems, by the application of RF as a hybrid approach between model-based and data-driven diagnostic approaches.

On one hand, this work demonstrates a unique architecture to deploy RF in FDD beyond its ordinary application as a data-driven methodology. Usually, RF is used as a classifier, feature selection, and when certain adaptations are made, RF can also be used for unsupervised learning. However, the literature is lacking the use of RF for model-based FDD or hybrid approaches beyond the data-driven combination ones.

On the other hand, in this work, a development and extension of the work in [11] is proposed, by offering a dynamic and general domain approach, with the possibility of generating dynamic diagnostic rules using RF.

5. Our Publications

- **Journals Submitted, Accepted and Published**
    - Mallak and M. Fathi, "A Hybrid Approach: Dynamic Diagnostic Rules for Sensor Systems in Industry 4.0 Generated by Online Hyperparameter Tuned Random Forest," *Sci*, vol. 2, no. 3, Art. no. 3, Sep. 2020, doi: 10.3390/sci2030061.
    - A. Mallak and M. Fathi, "Sensor and Component Fault Detection and Diagnosis for Hydraulic Machinery Integrating LSTM Autoencoder

Detector and Diagnostic Classifiers," Sensors, vol. 21, no. 2, Art. no. 2, Jan. 2021, doi: 10.3390/s21020433.

- **Journals in progress of publishing**
  - Mallak and M. Fathi, "Unsupervised Feature Selection Using Recursive k-Means Silhouette Elimination (RkSE): A Two-Scenario Case Study for Fault Classification of High-Dimensional Sensor Data," Aug. 2020, doi: 10.20944/preprints202008.0254.v1.

- **Conferences**
  - Mallak, C. Weber, M. Fathi, and A. Holland, "Active diagnosis automotive ontology for distributed embedded systems," in *2017 IEEE European Technology and Engineering Management Summit (E-TEMS)*, Oct. 2017, pp. 1–6, doi: 10.1109/E-TEMS.2017.8244219.
  - Behravan, A. Mallak, R. Obermaisser, D. H. Basavegowda, C. Weber, and M. Fathi, "Fault injection framework for fault diagnosis based on machine learning in heating and demand-controlled ventilation systems," in *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Dec. 2017, pp. 0273–0279, doi: 10.1109/KBEI.2017.8324986.
  - Mallak, A. Behravan, C. Weber, M. Fathi, and R. Obermaisser, "A Graph-Based Sensor Fault Detection and Diagnosis for Demand-Controlled Ventilation Systems Extracted from a Semantic Ontology," in *2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES)*, Jun. 2018, pp. 000377–000382, doi: 10.1109/INES.2018.8523895.
  - Mallak, A. Sonnad, and M. Fathi, "SenGen: A Two-Phase Dynamic Simulation and Toolbox of an Indoor Mobile Wireless Sensor Network for Sensor Monitoring and Dataset Generation," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Dec. 2019, pp. 1190–1195, doi: 10.1109/CSCI49370.2019.00224.

- **Posters**
  - Mallak, M. Fathi, A Dynamic Sensor Fault Detection and Identification System in IoT using MATLAB and Simulink, MATLAB Expo 2019, Munich, Germany, July 2019

- **Awards**

- Best poster award, MATLAB EXPO 2019: A Dynamic Sensor Fault Detection and Identification System in IoT using MATLAB and Simulink.

6. Structure of the Dissertation

The rest of the dissertation is structured as follows: in the following chapter (chapter 3), a brief yet extensive explanation of all the technical terminologies and definitions of the background used in this dissertation. Chapter 4 showcases all the state-of-the-art related research and application to the created methods explained in the next chapters. Chapter 5 represents an explanation of our innovative feature selection and window compression algorithm called R*k*SE. R*k*SE has been tested in two different experiments once as a feature selection to aid the diagnosis of component faults using multi-variate datasets, while the other experiment shows the abilities of R*k*SE as a window compression method when applied to a univariate dataset in a window structure to aid sensor fault diagnosis. Chapter 6 represents a joint approach between LSTM autoencoder and diagnostic classifiers to achieve both sensor and component faults in hydraulic systems. In chapter 7, RF is applied in a hybrid approach between model-based and data-driven approaches, beyond its traditional method of application as a data-driven or hybrid between data-driven approaches. Chapter 8 is where the conclusion of our methodologies represented in the previous three chapters is discussed, as well as adding some ideas for further research in the future.

Chapter 2: Conceptual and Theoretical Foundation

1. The Fourth Industrial Revolution

The fourth industrial revolution also known as industry 4.0 [13] is defined as the continuous dynamizing and automation of regular industrial and technological manufacturing applications invented in the previous three industrial revolutions. It is worth to spot the light on the main technologies created Industry 4.0, which can be concluded by the expansion in Machine-to-machine communication (M2M), and the invention of the internet of things (IoT). The emergence of Industry 4.0 has a tremendous influence on the shape of the industry nowadays, where engaging this technology improved the automation, built enhanced communication between industrial machines and components, improved dynamic and real-time monitoring and control, fully automated fault detection and diagnosis systems that do not have the need for human operators or control.

Below is a brief explanation of all the industrial revolutions that played a valuable role in the creation of the fourth industrial revolution we recognise today.

- **First industrial revolution:** In the 18th century the invention of steam power and mechanical manufacturing is the core of Industry 1.0.
- **Second industrial revolution:** The industry 2.0 era has emerged in the 19th century, due to the discovery of electricity and direct current.

- **Third industrial revolution:** This revolutionary discovery took place in the 70s of the 20th century. The main highlight of industry 3.0 is the growing technological computers and the creation of programmable memories, which insured partial automation in industrial processes.
- **Fourth revolution industry:** Finally, industry 4.0 is where all the previous revolutions are joined together with IoT, M2M, cyber physical systems and cloud-based technologies…etc, where all computers, components and humans communicate together via wired or wireless networks, to provide full automation, communication, and control between various industrial components over the production line.

2. Hydraulic Systems Overview

Hydraulic systems [14] are defined by the systems that are dependent on the utilization of pressurised fluids to generate a driving force or an incentive power necessary to power-up various industrial tools and components such as, motors and fans. Simply, the fluid motion due to temperature and pressure variations functions as a driving force to perform various industrial tasks and run mechanical devices.

The generated power by the hydraulic mechanism is enormous, which is the main motive of deploying such systems in heavy machineries. i.e., bulldozers, backhoes, loaders, cranes, and log splitters.

Applying the energy preservation laws and pascal law of pressure, it is noticeable that the created power by the pressurized liquid is transmitted and undiminished. According to Pascal's law [15], the principal law of hydraulics: "A pressure change occurring anywhere in a confined incompressible fluid is transmitted throughout the fluid such that the same change occurs everywhere.". Thus, by controlling the fluid temperature and the amplitude and direction of the enforced pressure, the power generated is recycled and circulated over all the components connected to the hydraulic system. In other words, the hydraulically generated power is not only massive to lift extremely heavy loads, but also can perform repetitive tasks easily.

The schema represented in the figure below is a paradigm of hydraulic systems with their basic components.

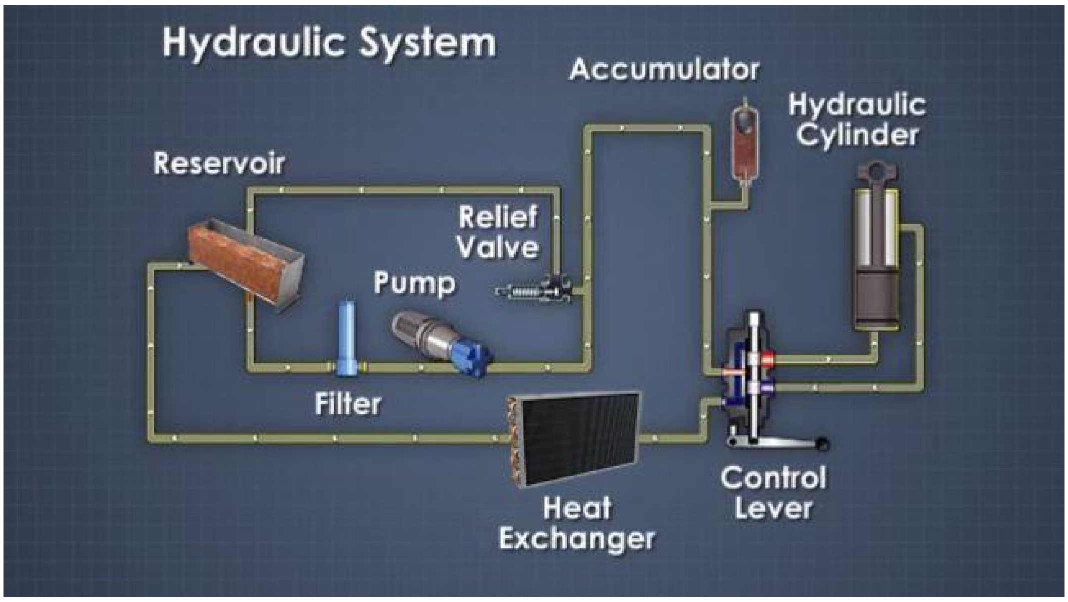


Figure 1 A Hydraulic System Paradigm Containing the Basic Components [16].

The main basic components in any hydraulic circuit are explained below:

- **Hydraulic pumps:** the pump is the mechanical part responsible for pumping the fluid to the rest of the hydraulic circuit components. Pumps act as a vacuum to push the liquid out of the reservoir through the filter back to the pump.

- **Hydraulic cylinders:** a hydraulic cylinder also known as the hydraulic motor, is the component responsible of generating unidirectional strokes that lead to establishing a unidirectional force. In other words, the cylinder is the mechanical actuator in the hydraulic circuit. Moreover, hydraulic cylinders and pumps can be also used as one unit in the hydraulic circuit called "hydraulic transmission".

- **Control valves:** there are several types of valves placed in various locations of the hydraulic circuit, but what they all have in common is their functionality of directing the fluid to the designated actuator or cylinder.

- **Hydraulic fluids:** some of the hydraulic circuits run by water. However, most of hydraulic systems are fuelled by hydraulic fluids. Hydraulic fluids also known as "tractor fluid" that mainly contains petroleum oil and other added fluids to fit the purpose and restrictions of the containing hydraulic circuit.

- **Filters:** this component cleans and filter the fluid coming from reservoir, by removing the unnecessary dirt, particles and contamination resulted from the other mechanical components.

- **Reservoir:** this part main functionality is to contain the excessive fluids continuously formed by the pressure and temperature differences, which leads to fluid expansion or sometimes leaks.

- **Accumulators:** this component is essential to reserve the generated energy to ensure the preservation necessary to perform repetitive motions. Accumulators store the energy applying a certain gas under pressure.

3. Fault Types and Classifications

In the last century, the world has witnessed an industrial and technological revolution that caused the rapid spread of nonlinear, complex sensor systems in various domains and applications. These systems are literally everywhere; in aircrafts, transportation like cars and ships, computing systems such as computers, laptops, smart phones, embedded systems, and in many industrial applications such as factories, chemical reactors and nuclear power plants and many more countless examples. As long as these complex systems continue to function properly, they play a major role in providing help, comfort and assistance to our daily lives and they are even considered a necessity to the current structure of modern societies. But what if these systems fail? What are the consequences and risks followed by such failures?

Faults in complex sensor systems can be defined as unexpected events that may occur at a certain point of time, that might trigger bigger events or a series of other unexpected events. Isermann and Balle [17] defined faults as:
A fault is an unauthorized, permitted or allowed deviation of one or more of the system's parameters, characteristics, behaviours or patterns from the normal or standard state of the system.

Based on the nature of these systems being non-linear, dynamic, and having complex relationships between its components, it is extremely complicated to predict faults in such systems. Faults consequences fall in a spectrum that ranges from harmless, ignorable faults to extremely disastrous ones that could lead to major economical and human catastrophes. There are many examples throughout the years of incidents caused by faults in complex systems which were so severe and almost caused human and nature extinction. An example of such incidents was the explosion of the nuclear reactor at Chernobyl, Ukraine, in 1986. Many civilians died immediately by the explosion, and many others died shortly after, affected by the radiations emitted from the power plant which caused them serious health issues including various types of cancer. The reason behind the explosion was a fault in one of the internal units that provided the system with trajectory and altitude information. Chernobyl till this day still is uninhabitable and so dangerous to visit, it still is highly contaminated by radio-active waste in the soil, water and air, and it is extremely biohazardous to all biological creatures in the planet for the next couple of hundreds or even thousands of years.

The real question is: is there anything that could have been done to avoid such catastrophic incidents? In most incidents in history, it is not possible to avoid the occurrence of such incidents. However, it is possible to predict the causing failures, detect them as soon as they occur, then take quick recovery actions to minimize the severity of their consequences. Which leads to the significance of applying Fault Detection and Diagnosis (FDD) technologies to ensure the reliability and safety of complex systems.

Before diving in this topic, it is highly recommended to understand the distinction between different technical terminologies related to FDD. The following definitions are described in more detail in [17]:

- **Fault:** is an unauthorized, permitted or allowed deviation of one or more of the system's parameters, characteristics, behaviours, or patterns from the normal or standard state of the system.
- **Failure:** a failure is when the system loses the ability to perform one or more of its required functions permanently.
- **Symptom:** an observable change in the system's quantifiable parameters from the normal state, which can point to the existence of a fault, that might lead to another fault or a system failure.
- **Fault Detection:** is the ability to detect the fault presence in the system and the point of time this presence occurred.
- **Fault Isolation:** is the ability to isolate the detected fault(s), by finding its type, location, and time. This process usually follows the fault detection process.
- **Fault Identification:** is the ability to identify the detected and isolated faults by identifying their size and time-variant behaviour. This process follows fault isolation.

- **Fault Diagnosis:** is the ability to create static or dynamic relationships between symptoms and their connected faults, that altogether might cause the occurrence of system failure.
- **Monitoring:** is a real-time operation of continuously observing the system and its behaviour, by recording its vital information that might indicate an existence of a fault.

In the literature, faults can be classified into three main categories based on the location of the fault itself in the containing system as: sensor faults, actuator faults and component faults [18].

3.1 Actuator Faults

Are the faults occur in actuation units and appears as a partial or complete malfunction of the actuation control. In other words, the actuators could be faulty when they fail to perform the actuation function i.e., stuck actuator. A complete fault in actuators can appear as a result of a burning wire, a cut, leakage, breakage or a presence of an actual physical object holding back the actuator preventing it from controlling the system's behaviour.

3.2 Sensor Faults

Sensor faults are the faults represented by the sensors and their readings. Usually, these faults are noticed when the sensors are producing incorrect readings due to a physical fault in the sensor itself, broken wires, or a malfunction in the communication channels between the sensors and the controlling unit, or the change of the sensor's reading could be an indicator (symptom) of a component or system fault. Industrial systems contain hundreds of sensors attached in different locations of the system, wired or wireless, stationary, or mobile, to continuously measure some key variables of these systems in real time. The data generated from sensors is considered a rich source of information from the analytical perspective, since this type of data has a vast majority of unique patterns and worthwhile characteristics. Moreover, any sudden changes of these sensors' readings, or the appearance of any unexpected patterns that goes without notice, can lead to a major risk and serious consequences.

A. Sensor Faults Taxonomy

There are several existing classifications and categorical descriptions of sensor faults. The most sound and interesting one, is the comprehensive study conducted in 2009 [19], where an extensive approach was taken to provide a clear definition of each fault, their potential cause(s), the observed duration of each fault in time-series datasets and the effect each fault carries on the sensed data.

In accordance with the mentioned research, sensor faults can be divided into two main categories based on how they get observed, and where the fault shows its symptoms. The two groups are: faults based on system view and faults based on data centric view.

The first broad category is system-centric faults, where the fault influences the system, and its symptoms can be observed by the system's behaviour. The system here is meant to be the sensor itself and its physical nature. The system related sensor faults are usually caused by the following: (1) the sensor calibration such as offset, gain, and drift faults. (2) low sensor battery, (3) data clipping, which occurs when the sensor reaches its minimum or maximum reading and cannot sense beyond its defined capacity, (4) connectors and hardware problems and (5) environmental factors as of radical changes beyond the sensor range of readings, which leads to clipping.

Calibration Faults: Calibration is a tricky task when applied to sensors. If the calibration is not done with caution and expertise, it might lead to serious consequences on the generated sensor data. The following are the main calibration related sensor faults and their definition.

- Offset or bias fault: This fault occurs when the data is offset or shifted by a constant value called the bias. This fault can be easily injected by adding a constant value or an offset to the input data. Bias faults are hard to diagnose since the input data keeps containing similar patterns to healthy data but shifted in value.

- Gain faults: gain fault represents the amplification of the input signal, or simply multiply the input signal with a constant value.

- Drift faults: in these faults the accuracy or the performance of the input signal is drifting away from the expected healthy performance. This kind of fault can be injected by adding the input signal to a function of change such as a polynomial.

The second category is the data-centric sensor faults, where the sensors deviation can be observed in the data, while not necessarily having a symptom or cause noticeable on the physical aspect of the sensor. The data-centric faults are summarized by; outliers, spike, stuck-at or constant, and noise.

- Outliers are one of the most common faults to occur in sensor data. An outlier as a sensor fault can be defined as an input sensor signal or an isolated duration that shows a significant deviation from the expected recorded patterns. The reasons behind this kind of faults are unknown and varies each time.

- Spikes: is a type of sensor faults that can be diagnosed by the data observation, in which the rate of change over a limited duration of time is observed to be higher than the expected rate of change. This type of faults is not necessarily returning to normal phase. The difference between the spike and outlier faults, is that spike faults should be observed in multiple readings or data samples, unlike outliers where they only show effect in one isolated sample or entry. The main reasons behind spikes faults are expected lower battery levels of the sensor, or a malfunction in its connectors or the hardware.

- Stuck-at faults: stuck-at faults also known as constant faults are defined as a sequence of consecutive entries that have no variance over a relatively long period of time. In other words, the sensor is stuck at some value for longer periods than usual. The sensor can be stuck at zero, hence is called constant-zero. As well as, stuck at the highest or lowest possible values of the sensor, which are called constant-high and constant-low, respectively. The main reasons behind constant faults can range from neglectable faults due to clipping (sudden environmental changes that cause the sensor to falsely point at its maximum or minimum reading range) a low battery level, or more serious causes connected to the sensor connectors or hardware malfunction.

- High noise: normally any sensor data exhibits a small amount of noise. However, when the data contains more noise or variance than usual, it might be a sign of faulty sensors. This fault can be connected to lower battery levels, or the occurrence of actual hardware malfunction in the physical sensor.

3.3 Component Faults

Component faults are the faults appear in different components of a complex system that are not considered a sensor or an actuator. These faults are the most common among all the fault types, and they can vary drastically from a system to another or a domain of application to another. When component faults occur, they change the observed behaviour of the system and its expected outcomes and results.

4. Fault Detection and Diagnosis (FDD)

4.1 FDD Definition

Fault Detection and Diagnosis (FDD) is the process of finding odd, extraordinary, or unusual patterns in the given data, comparing to the patterns it usually forms in the regular or healthy state. These irregular patterns are most commonly called faults, anomalies or outliers [20].

In the last decade, FDD has been an interesting topic for many researchers applied in a wide range of applicational domains. Due to its enormous significance to provide the needed safety, security, and reliability in many industrial systems. As well as, the vital role it plays in the fast detection of abnormalities and faulty patterns, which is essential in many industrial systems, especially the ones with harsh or highly restricted environments, systems that are prone to malicious attacks, sensor systems that contain fault-prone sensors or the sensors' reading might be faulty or unusual. As a result, many FDD systems are developed for a specific domain, while others offer a more generic solution.

4.2 FDD Classification

FDD approaches can be divided into three main categories; model-based and data-driven approaches, as Venkatasubramanian et al explained in [21], [22] and [23].

Model-based approaches uses a physical system or a simulated model of the system to form a prior-knowledge of this system, its components, symptoms, faults, and failures. The stored knowledge is used as a reference to identify faults, where they can be spotted easily by observing the difference between this reference knowledge and the current applicational measurements.

Data-driven approaches might not offer any knowledge of the physical system or a modelled simulation. However, they tend to extract data-driven knowledge by analysing the system recorded data and applying various methods to find hidden patterns and relationships that describe the unknown system and its behaviour. Although these approaches do not provide a good insight of the system and its processes, these approaches are extracted from the data (unlike model-based methods that are dependent on fixed rules and rigid knowledge) is considered a dynamic, general-domain solution.

Hybrid approaches can be formed by introducing a combination of methods of the same group or different groups. i.e., a combination between two data-driven approaches to form a new hybrid one or establishing a bridge between a specific method in data-driven and model-based to finally produce a hybrid offspring method between entirely different branches.

Model-based and data-driven approaches each can be divided into two main classes based on the methodology they use to approach the fault detection problem: Quantitative models and qualitative models.

- **Quantitative models:** Are the models that uses the relationships between the systems' symptoms, faults, parameters, and components either statically or dynamically to describe the system and its behaviour quantitively using ***mathematical formulas*** and ***statistical approaches***.

- **Qualitative models:** Are the models that uses the relationships between the systems' symptoms, faults, parameters, and components either statically or dynamically to describe the system and its behaviour ***qualitatively using causalities, graphs and if-then rules***.

A. Model-Based FDD

- Quantitative Model-Based FDD

The quantitative approach for model-based FDD is based on deriving mathematical formulas and relationships from the physical or simulation model to describe the natural behaviour of the underlying system. These explicitly derived mathematical knowledge is used to detect and diagnose the intended system faults. The quantitative information derived from the physical system and represented by mathematical equations can be extracted from the system by either observing the behaviour of the system in the time-domain, parameters estimation using the knowledge of the system experts and their understanding of the created physical system or finally, by applying parity space methods.

- Qualitative Model-Based FDD

Another common approach for model-based FDD is by applying qualitative methods. These methods do not rely on mathematical formulas to describe the normal state of the physical system. Instead, they create a rule-based diagnostic model derived from the physical system's prior-knowledge. These rules can be extracted by a direct contact with the system expert to build and exchange knowledge about the system's components and rules between them. The verbally transmitted expert-knowledge is normally documented and summarized in some sort of a database such as ontologies, either with a semantic representation or simple if-then statements. The most famous qualitative model-based FDD methods are digraph methods, fault trees and qualitative physics approaches.

A majorly important type of model-based systems are expert systems which tend to use rule-based, empirical reasoning and functional reasoning to inference a series of rules necessary to make the diagnosis. These systems are suitable to diagnose complex systems that need a lot of human expertise to offer deep understanding and provide resolutions.

The main characteristic of expert systems is the deep association between the fundamental given concepts of the system, and the inferred knowledge. This relationship is a perfect example of a causality, also known as the cause-effect relationships based on establishing logical conclusions that bridge the initial premises and the diagnostic conclusions. In Addition, deploying heuristics into these systems can result in forming heuristic-based expert systems, which tends to contribute to solving many of the limitations of the regular expert systems but still is domain-specific and lacks the generality in its approach.

The downsides of using the traditional cause-effect methods represented in expert systems include the following:

- The lack of dynamicity and generality; static knowledge for a specific domain.
- The lack, or absence in handling sudden or novel occurrences.

- Unable to detect, fill and update the knowledgebase/system gaps.
- The lack of credibility in knowledge acquisition because it is completely dependent on experts' reliability.
- The impossibility to learn from misdiagnosis and errors them.

B. Data-Driven FDD

- **Quantitative Data-Driven FDD**

Data-driven quantitative approaches use the process history data and the systems measured data to extract mathematical formulas that defines hidden patterns or relationships between different parameters and features of the physical model, without having any prior-knowledge of this model. The mathematical representation of the data knowledge can be extracted in two different fashions; one is a statistical approach, and the other is a black-box approach.

In statistical approaches, the mathematical model parameters are estimated taking into account the physical system harmony and principles, these parameters are often obtained from the measured data generated from the system using various statistical and pattern recognition approaches, such as supervised, unsupervised, semi-supervised and ensemble-learning methods.

The black-box models formulate the system using non-physical parameters or characteristics of the system. In other words, the mathematical formulas extracted by black-box approaches do not reflect or represent the actual physical system or its components, that is why such models are identified by a "black-box". An example of black-box data-driven FDD methods are shallow neural networks like Artificial Neural Network (ANN), fuzzy logic and deep neural networks.

- **Qualitative Data-Driven FDD**

Previously, there were three major methods to extract qualitative knowledge from the historical information or the system's measured data, which are expert systems, knowledge-bases and Qualitative Trend Analysis (QTA) [24]. However, recent studies [23] and [25] added these three approaches under qualitative model-based branch, since such methods include creating a model of the system and its diagnostic rules, either inferred or learned. Which makes these methods more suitable to be categorised under model-base rather than data-driven.

In conclusion, data-driven approaches offer a lot of advantages over other types of diagnosis approaches, including their ability to transform the data into less complex and lower-dimensional forms using feature selection and extraction approaches. Moreover, data-driven approaches are of a dynamic nature. Hence, they can be an effective alternative to general-domain FDD approaches with high level of dynamicity and ability to identify and learn new faults and abnormalities directly from the data.

Although data-driven approaches show a dynamic behaviour, the decision made using such methods are not guaranteed to be more accurate and they have the tendency to consume more time, computational and storage resources comparing to their model-based counterparts. Additionally, data-driven approaches are useless when ones do not possess the fundamental skills in machine learning, big data analysis and statistical knowledge. However, the main disadvantage of methods relying on this schema lays in the concept of data-driven approaches itself, where they are a hundred percent dependent on the quality of the data. Which results in producing and storing a huge amount of data, that is not only a resource expensive but also dangerous, since some data has a sensitive content and storing them makes such an attractive hub for cyberthefts and anomalous attacks.

The following is a sum-up of data-driven cons:

- The dependency on the data
- The dependency on possessing certain skills to apply those methods.
- Storing data is resource and security expensive.
- Additional expenses related to the need of additional hardware and maintenance.

An overview of the classifications of FDD methods influenced by the chart visualized in 2003 by Venkatasubramanian et al [21], and Skliros et al in 2018 [23]. The chart is extended and updated to meet the rapid development in FDD methods and techniques in the past ten years as shown in Figure 2.

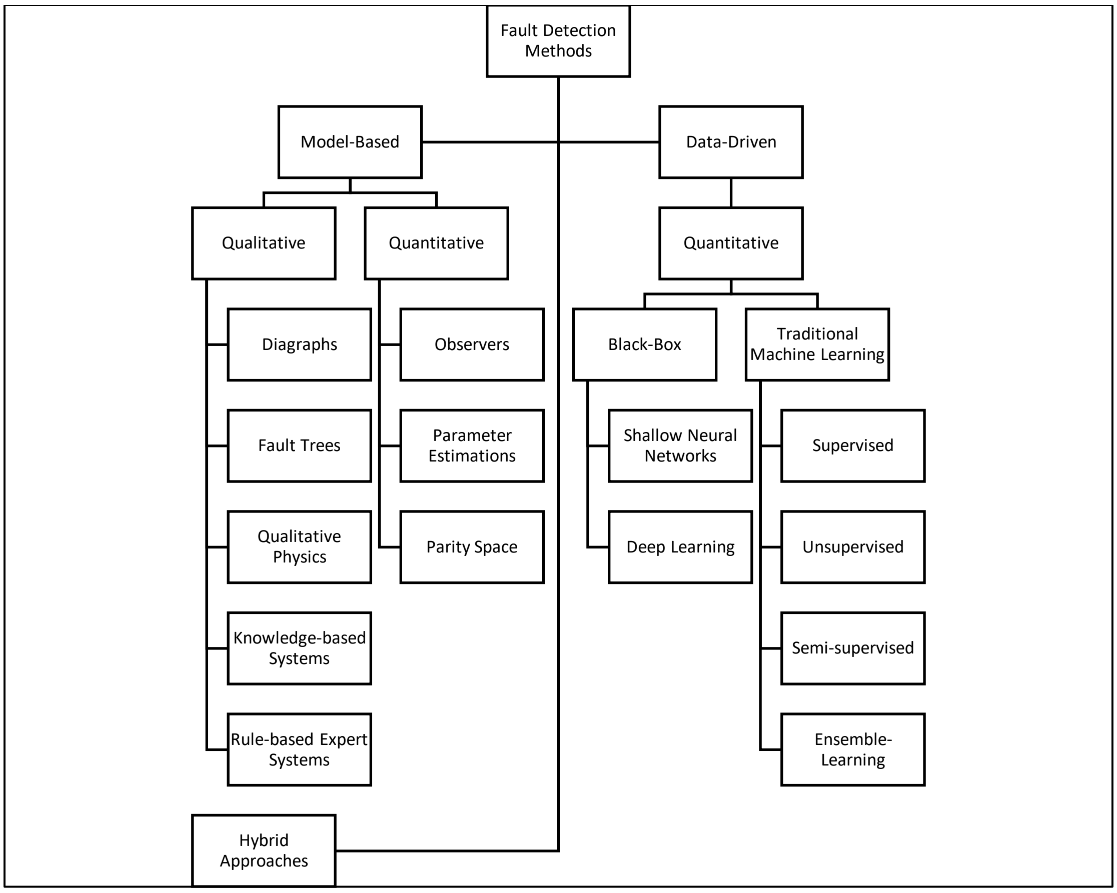


Figure 2 Classifications of FDD Methods

5. Machine Learning Algorithms Taxonomy

Machine Learning (ML) is a subset of Artificial intelligence (AI) that is concerned in the algorithmic structure of computer procedures, which continuously improve due to learning or experience [26]. Commonly speaking, ML is fundamentally a process of establishing mathematical and statistical models using a portion of the input data used for training, named "training data". These models are expected to make decisions when tested with further data based on the patterns learned from the training process without the aid of pre-defined programs or rules.

ML approaches can be classified into four main categories based on their mechanism of learning the patterns and decision making, as well as the availability of the labels associated to the sample data or not. Based on this, ML algorithms can be classified into supervised, unsupervised, semi-supervised and reinforcement learning. As shown in Figure 3

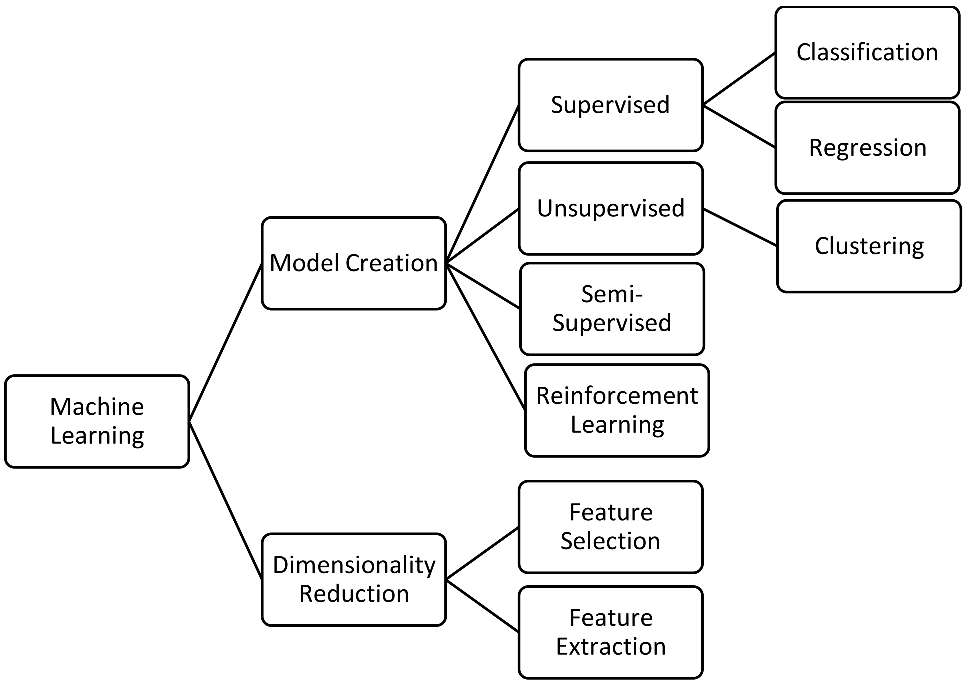


Figure 3 Machine Learning Algorithms' Types

Supervised Learning: is the type of machine learning where the data samples giving for training are fully paired with their label or category that describes the problem. These types of models try to map the relationship between the samples' features dependencies to their target label to make the final decision.

The labels or target variables in supervised learning can be in applied two forms: (1) discrete values in which each value represent a category or class. This representation is used in classification algorithms. (2) Continuous target values or a value within a range. This type of label representation is use in regression problems, where the algorithm is expected to result a number within a specific range.

Some famous examples of supervised learning algorithms are: SVM, KNN, NB, CART, Linear Regression and many more.

Unsupervised Learning: is the type of ML that lacks the class labels or target variables as in supervised methods. These approaches function by analysing the relationships and connections between the samples themselves, then create clusters or groups where the samples of similar behaviours are gathered together. Unsupervised ML methods are extremely important to analyse, provide better understanding of the data, and derive meaningful relationships between the samples. An example of unsupervised learning algorithms is *k*-means clustering, Hierarchical clustering, Self-Organising Maps (SOM) and many more.

Semi-supervised ML: is the type of ML that lays in between the supervised and the unsupervised approaches, in which the data is partially containing the target variable information. It is worth to be mentioned here that this type of ML approaches is quite common in real-life scenarios. Since the data labelling process is very time and effort consuming and usually is manually done by a human observer exhaustively recording the status of an observed process. These methods can be used in classification or clustering tasks based on the end-goal of the learning problem in hand.

Reinforcement Learning: reinforcement ML has a different mechanism of learning comparing to the last three ML types. The Reinforcement learning is based on the continuous interaction between the training model and the environment to provide a reward system, in which it the decision making ensures the best reward and the least risk. Reinforcement learning is the closest scenario to how humans learn based on actual environmental interactions and experience, which makes this type ideal for robotics and robot learning. The following workflow chart shows how reinforcement learning methods organize the interaction between agents and the surrounding environment. One of the most well-known reinforcement learning methods is Q-learning and temporal difference.

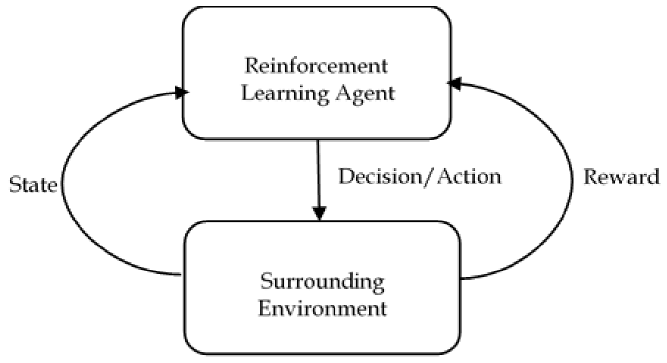


Figure 4 Reinforcement Learning Mechanism

In this work, we are mainly concentrating on classification ML algorithms to achieve FDD. In addition, to unsupervised feature selection algorithms which might enhance the efficiency of FDD in hydraulic systems. Accordingly, the literature of the methods mentioned bellow are coherent to the content of this dissertation and reflects all the methods that are encountered within this work.

6. Feature Selection Literature

Feature selection categories are determined based on two main characteristics. (1) The existence of samples' labels or classes, and (2) the search strategy used to select the features [27].

First, according to the label availability, feature selection methods can be classified into; supervised, semi-supervise and unsupervised feature selection methods [28].

Supervised feature selection algorithms depend on the existence of labels to efficiently chose the most discriminant and informative features, by effectively classifying the samples between the classes using the given labels. However, when only some of the data samples contain labels, while the rest are unlabelled, in this case the feature selection is called semi-supervised. Semi-supervised algorithms are capable of extracting feature importance from both labelled and unlabelled instances. Generally, semi-supervised feature selection relies on the construction of correlation matrix between features and determine the valuable ones based on their similarity degree to the formed matrix [29]and [30]. The most spontaneous, natural, and common form of data is the unlabelled form, where the records are stored from their source naturally without being combined with any observations, or any sort of grouping or notations. Due to the absence of guidelines and clues, unsupervised feature selection is considered by far the most difficult type comparing to the former two [31]. The related work to this method is explain in detail in related work section.

Second, another way to determine the feature selection method used is by the process strategy which can be divided into filter, wrapper, and embedded methods.

In most practices, filter methods are applied as a pre-processing stage prior to the actual feature selection using wrapping methods. The features are selected by performing various statistical tests to measure the correlation between each feature and which are more relevant. Some examples of statistical tests to measure the correlation coefficient are, Pearson's correlation, Fisher transformation coefficient also known as F-test, Linear Discriminant Analysis (LDA), Chi-square, and many more. Filter methods utilize the shape of the data to determine the most valuable features. More specifically, by applying a certain condition, methods, or criteria to rank the features, then order them in a descending order based on the rank calculated while selecting the features highest in the order to represent the rest. Some examples of the filtering approach is reliefF [32], information gain [33], and F-statistic [34].

Wrapper methods rely on the continuous selection of various subsets of features from the feature space and utilize them each to train a machine learning model and infer which subsets to choose and which to eliminate according to the resulting performance of the model. In other words, wrapper methods do not rely on the shape of the data as the filter methods, instead they use machine learning to select the features with the best accuracy when running the machine learning model. Although, this kind of methods conclude the feature selection process in a search problem, these methods are very computationally expensive.

Wrapper methods are known for applying various machine learning and data mining algorithms to select features such as, the utilization of SVM to create SVM Recursive Feature Elimination (SVMRFE), which is considered one of the most useful feature selection algorithms when using microarrays and genetic data [35].

Wrapper methods can be divided into forward, backward, and recursive feature elimination. Below is the detailed explanation of each:

- **Forward Selection:** is the wrapper feature selection method in which ones iteratively keep adding new features to the model, while starting the selection process with no features at all. These methods are initialized with a null feature set and keep adding new ones each iteration until convergence is reached. The most common convergence criterion is when the model has the highest performance the way it is without any additional feature from the feature space.
- **Backward Elimination:** this method follows the exact opposite strategy of the forward selection. In this case, the feature subset is initialized with the whole subsets in the feature space, followed by iteratively eliminating them until the performance of the model is the best without any further elimination.
- **Recursive Feature Elimination:** these methods are remarkably similar in the mechanism to greedy search. They iteratively select a different

subset of the features, run the model, and keep tracks of the best and worst performance features at each iteration. Each iteration, the construction of the next model is dependent on the features left from the previous iteration. Finally, a full rank list of the features is created and accordingly the selection or elimination is decided.

Finally, embedded feature selection methods or sometimes called hybrid methods, are a combination between filter and wrapper methods, where they construct their own built-in feature selection using both the shape of the data or the statistical analysis of it, along with applying machine learning models. A popular example of embedded methods for feature selection is the application of LASSO and RIDGE regression by constructing internal criteria functions to contribute to feature selection and reduce overfitting. Figure 5, shows the classification of feature selection methods according to the research in [27].

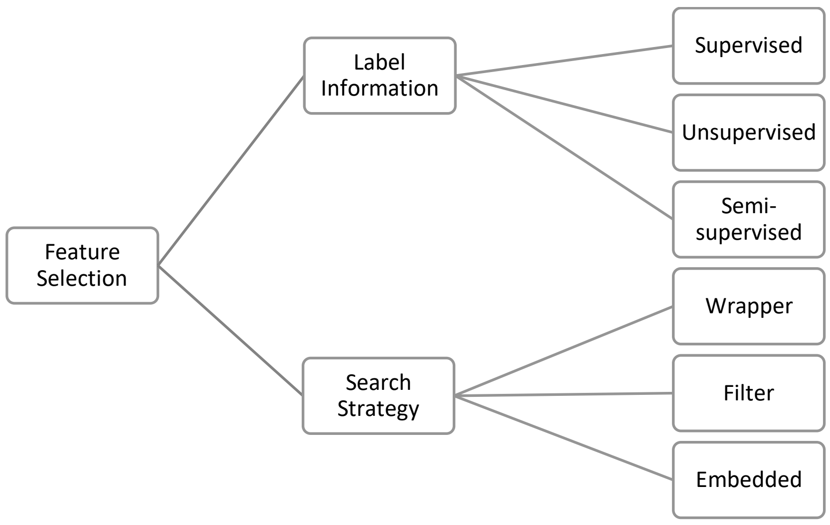


Figure 5 Feature Selection Classification [27].

The following table describes the advantages and disadvantages of wrapper, filter and hybrid (embedded) approaches, as described in [36].

Table 1 Filter, Wrapper and Embedded Feature Selection Methods Pros and Cons Comparison.

Method	Pros	Cons
Filter	• Scalability • Speed • Independent from the clustering process • Permits parallel computation	• Has limited interaction with the clustering algorithm.
Wrapper	• Can Create a model Representation of feature dependencies. • Provides interaction with the clustering algorithm used for selection.	• Prone to overfitting. • Computationally expensive. • Method-specific: depends on the clustering algorithm applied
Embedded	• Provides interaction with the clustering algorithm used for selection. • Less computational time complexity comparing to Wrapper methods. • Can Create a model Representation of feature dependencies.	• Method-specific: depends on the clustering algorithm applied

7. *k*-means Clustering Literature

k-means clustering algorithm was first created back in 1967 by James MacQueen, the detailed article is shown in [37], where data is divided into *k* number of clusters based on their connectivity and pattern. This method is optimal to find hidden patterns in unlabelled data.

The basic flow of any process must contain inputs, outputs, and the mechanism of the process itself for *k*-means clustering algorithm. First, the inputs expected are the unlabelled dataset *D*, and a predefined number of clusters *k*. *k* can be chosen randomly or computed using various methods. In a recent study [38] various methods are introduced to calculate the most optimal number of clusters in *k*-means algorithm. The most common method to calculate *k* is using elbow method. The elbow method will be used later in the feature selection algorithm proposed in this thesis, so that it is essential to have some overview of the method and how it works. The fundamental idea of the elbow method is to calculate the sum of squared errors (SSE) for various *k* values iteratively, while the best *k* is represented by the *k* value with the first sudden drop in SSE value, in such way that it looks like an elbow when plotting *k* and the distortion of SSE.

Second, the process in which *k*-means clustering work as the following: (1) randomly select *k* number of samples from the dataset *D*. These *k* samples are considered the seeds of the algorithm which can be chosen randomly or following some specific algorithms to initialize the seeds. (2) The chosen seeds will perform as the initial centroids of the *k* clusters, in which the distance between all the instances and these centroids is calculated. Each instance is grouped in the nearest centroid's cluster (minimum centroid distance). (3) Perform iterative or repetitive centroids selection and distance calculations to optimize the centroids locations, to eventually have the best centroids locations that ensures better clustering. The regular equation used to calculate the new centroid each iteration is explained below:

$$C_{i\_new} = \left(\frac{1}{N_i}\right)\sum_{i}^{N_i} x_i \qquad \text{Eq. 1}$$

Where N_i is the number of members or instances in the cluster *i,* and the calculation of C_{i_new} is simply computed by finding the mean point or object of each cluster.

There are two main conditions to revoke the iteration and get the final centroids positions. Either by calculating the mean distance for each cluster iteratively and stop when the cluster has the minimum mean distance between the centroid of this cluster and all the cluster members, or when a maximum number of iterations is reached.

Finally, *k*-means output is expected to be a *k* number of clusters, where each cluster has a centroid C_i and various members of the object forming the dataset *D* based on their connectivity or distance to their final cluster.

The study in [39] demonstrates four different distance measures that can be used to calculate the distance between the objects and the centroids within each iteration of processing *k*-means. The following bullet points explain the most popular three distance measures that used in *k*-means clustering algorithm.

- **Euclidean distance:** is the distance computed by subtracting the corresponding coordinates of two objects, and then measure the square root for the squared value of the individual subtraction, as shown in the equation below:

$$D_{xy} = \sqrt{\sum_{f=1}^{m} \left(X_{if} - X_{jf}\right)^2} \qquad \text{Eq. 2}$$

Where m is the total number of dimensions in both vectors $X_i$and X_j.

- **Manhattan Distance:** is the distance calculated by measuring the absolute value of the difference between two objects.

$$D_{xy} = \left|X_{if} - X_{jf}\right| \qquad \text{Eq. 3}$$

- **Chebychev Distance:** this kind of distance measurement computes the Manhattan distance for each two corresponding dimensions in two objects individually, and then take the maximum dimension distance as the overall distance representing the two vector objects.

$$D_{xy} = max_f\left|X_{if} - X_{jf}\right| \qquad \text{Eq. 4}$$

Generally, k-means for feature selection methods work by trying to cluster the features as samples, as they initially selecting k random number of features from the feature space to perform as the k clusters' centres and repeatedly measure the similarity between the selected centroids and their objects.

7.1 Silhouette Value Literature

Clustering in general, is the process of defining groups of objects. In a way that, in each group objects tend to be like one another, and different from other objects located in other clusters or groups. How to evaluate these clusters? A high-quality cluster usually have a high intra-cluster similarity value, and low inter-cluster value, which means the objects contained in this cluster are highly similar and connected to one another, and distinctively distant from other objects in other clusters. In other words, the cluster should be well-defined to be considered a high-quality cluster.

There are many similarity measures to compute the intra-cluster and inter-cluster similarity values. Such as, silhouette value [40].

The silhouette value for an object i in one of the k clusters computed using k-means clustering can be calculated as the following equation:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \qquad \text{Eq. 5}$$

Silhouette value can also be written in a function formula as the following:

$$S_i = \begin{cases} 1 - \frac{a_i}{b_i}, & if\ a_i < b_i \\ 0, & if\ a_i = b_i \\ \frac{a_i}{b_i} - 1, & if\ a_i > b_i \end{cases} \qquad \text{Eq. 6}$$

Where a_i is the average distance between the object i, and all other objects that belongs to the same cluster. b_i is the minimum average distance between the object i and all other object in all neighbouring clusters.

S_i is the silhouette value of the object i. Based on the values of a_i and b_i using one of the equations above.

S_i is a straightforward approach to know the similarity or dissimilarity measures between the object i and its group. If S_i is close to the positive end, positive one, that means this object is highly similar to its cluster and appropriately grouped. When S_i is close to the negative end or negative one, then i is not appropriate to its original cluster, and the similarity between i and its neighbouring clusters are higher, so i should have been clustered in the neighbouring group instead. An S_i close to zero means that this object is located in between two main clusters and does not belong to any.

8. Relevant ML Classification Algorithms

8.1 Logistic Regression (LR)

To understand LR, it is necessary to distinguish it from Linear Regression. Linear Regression is suitable for ML tasks where the outcome is expected to be a range of values, or a continuous measurement such as, students grades prediction, the amount of rainfall in a specific time of the year, the expected earnings of a production line…etc. Nevertheless, logistic regression [41] [42] results a discrete value of a clear category mostly suitable for binary classifications (i.e. student fail or pass, the weather is rainy or not, a production line have earnings or not)

Logistic regression applies a transformation function to calculate the probability in which each targeted occurrence is measured, and then accordingly decide which group to classify the sample within. This function has an S shaped-curve and is called the "logistic function" which is where logistic regression gets its name from.

An example of how logistic regression works, assume a binary classification scenario where $Y = \{0,1\}$ and the input data has m number of samples $X = \{x_1, x_2, x_3 \dots x_m\}$ and the relationship with the classes is linear. The traditional equation for logistic regression is shown below:

$$l = log_b \frac{p}{1-p} \qquad \text{Eq. 7}$$
$$= \beta_0 + \beta_1 x_1 + \beta_2 x_2 \dots + \beta_m x_m$$

And

$$p = \frac{1}{(1 + e^{-X\beta})} = \frac{1}{(1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 \dots + \beta_m x_m)})} \qquad \text{Eq. 8}$$

The results of l will determine the value of y. If $l > 0$ then $y = 1$ (instance is in the positive class). When $l\ is\ otherwise$ then $y = 0$ (instance is in the negative class).

8.2 Naïve Bayes (NB)

NB is a well-known supervised, statistical machine learning approach. NB is dependent on the probability calculation of the next event occurrence, given the probability of another influential event that has already happened. The basic idea of NB classifier is derived from Bayes theorem [43], where the prior knowledge is used to calculate the probability of the hypothesis being true.

Bayes' theorem in a mathematical formula is shown below:

$$P(h|d) = (P(d|h)\ P(h))/P(d) \qquad \text{Eq. 9}$$

Where:

- *$P(h|d)$* is the conditional probability of the hypothesis event h in condition of the probability shown in the data d.

- *$P(d|h)$* the conditional probability of the data d that makes the hypothesis h true.

- *$P(d)$* the probability of the data d irrespective to the hypothesis condition whether it is true or false.

- *$P(h)$* the probability of the occurrence of hypothesis h irrespectively with the observations stored in the data d.

As a final note about NB. The word "Naïve" is added to the algorithm name to indicate the naïve assumption that this theorem is built upon, which assumes that the variables in d are independent from each other, which is highly unrealistic in real-world scenarios.

8.3 K-Nearest Neighbors (KNN)

KNN [44] is a classification and regression machine learning algorithm, which has a unique technique of using the whole dataset without splitting the data into training and testing required by traditional supervised approaches.

KNN uses the instances or samples in the data as a reference, when a new instance is given, KNN tries to find the most similar K points or instances to the new point. When KNN is used for a classification task, the label of the new point is determined by the majority class label of the chosen nearest K points. For regression problems using KNN, the mean value of the K instances is used to compute the regression outcomes.

KNN parameters such as K and the similarity measure used are predefined by the user. There are a variety of similarity measures which basically work by calculating the distance between the new instance and each point in the dataset. The distance measures can be Euclidean, hamming distance and many others.

8.4 Linear Discriminant Analysis (LDA)

LDA is a machine learning algorithm used for classification and feature selection purposes based by the application of statistical concepts. More specifically, LDA applies Fischer linear discriminant [45] statistical approach to compute the linear combinations for the features in the given dataset, which provide the best separation between the labelled classes or events.

It is worth to be mentioned that both PCA and LDA functions by finding the linear combinations of features. However, they both have different criteria to choose the hyperplanes. In PCA, the hyperplanes are chosen to maximize the variance or difference between the data points in the feature space without taking the knowledge of the points' labels or classes into consideration, as well as the fact that the chosen PCs must be orthogonal to one another. On the other hand, LDA chooses the hyperplanes to maximize the similarities between points of the same class or label to provide the best separation possible from other classes in the dataset. LDA's hyperplanes chosen are not necessarily orthogonal to one another as in PCA.

8.5 Support Vector Machine (SVM)

SVM [46] is a non-statistical machine learning approach that can be applied to solve classification and regression problems. SVM works by attempting to find the optimal hyperplanes that offers the best separation between the labelled classes in the feature space. The optimal separation between the classes is determined by choosing the hyperplanes that has the furthest distance between the data points and the hyperplane. The gap distance between the closest instance on both sides of the hyperplane is known as "functional margin". The margin should be at least equal to one. However, in some cases some tolerances can be applied and margins with less than one distance can be accepted to optimize the SVM performance.

Although, SVM works by finding linear combinations of hyperplanes, it can also be used to solve non-linear problems by applying the "kernel trick" which makes SVM extremely relevant to the nature of real-life problems, and universally applicable in many fields and applications such as, fault detection and classification.

When plotting the data points in the feature space, sometimes it is even visible by the naked eye that the separation hyperplane between various classes is not linear. The kernel trick comes to overcome this challenge by projecting the data points with non-linear separation hyperplanes into a higher dimensional space, where the classes appear to be linearly separated by a linear hyperplane. There are various types of kernels based on the shape of the decision boundary most optimal to separate the data classes, i.e., Fischer kernel, polynomial kernel, Gaussian kernel, Radial Basis Function (RBF) kernel, and many more.

8.6 Decision Trees

Decision trees in data mining are a commonly used supervised technique to solve classification and regression problems. Where a set of observations and their labels or classes are already known and used to make various predictions [47]. In data mining, decision tree algorithm is divided into two main types: classification and regression trees. In 1984, Breiman et al [48] combined the two types together under the same category using the term Classification And Regression Tree (CART).

Decision trees are called this way because they are visualized in a tree structure, in which is created by recursively splitting the training dataset from top to bottom, forming the first level node of depth zero called the "root", followed by going down the tree forming higher depths and continuously splitting into successor children nodes. The splitting process is determined using different rules that determine the impurity of a certain node, and upon the selection of the splitting criteria [49].

8.7 Random Forests (RF)

- From Bootstrap Aggregation (Bagging) To Random Forests

Furthermore, there are some algorithms classified under ensemble learning category that allow the possibility of creating multiple different trees over the same dataset, to contribute to minimizing the over-fitting problem decision trees usually suffer from, especially when the sample size provided is relatively small. The two main types of such ensemble methods are boosted ensemble trees [50] [51] and bootstrap aggregated or bagged trees [50] [52].

Boosted trees are a sequential type of ensemble decision trees, where the optimal shape of the tree is established incrementally by adjusting the tree continuously based on the arrival of new instances. The most famous boosted trees algorithm is AdaBoost method.

Bootstrap aggregated or bagged trees are a parallel type of ensemble decision trees, generates multiple numbers of decision trees concurrently, by resampling the training dataset with replacement. The final prediction for such methods is made by voting the results of the created trees altogether. What is worth to be mentioned is that random forests are an example of a Bootstrap aggregating methods to optimize the traditional decision trees methodology [52].

Random forests are an optimization algorithm of decision trees, under the ensemble learning sub-category, intended to perform different tasks such as, classification, regression, and many others. The core of this algorithm relies on creating multitude of parallel decision trees based on dividing the feature space each time and deploying the chosen sub-space to form the tree of choice. The prediction decision of random forests is made by majority vote of all the separately created trees. Random forests started as "stochastic discrimination" approach created by Eugene Kleinberg [8]. Which was inspired by the formula created by Tin Kam Ho [53] to deploy the understanding of random subspaces and how to use them in a practical approach. Recently, random forest algorithm is trademarked by Leo Breiman and Adele Cutler owned by Minitab, Inc in 2019 [54]. The registered algorithm represents an extension of the formula introduced by Ho [53] and the "Bagging" idea created by Breiman [52] and [48].

Bagging algorithm whole idea depends on randomly choosing a subset of the original training set with placement, to perform an **S** number of classification or regression tasks, to finally make the overall decision of the performed task using all the learners created. Generally, the trees created by the bagging algorithm alone tend to be highly correlated and, in most cases, the same tree is being generated for multiple of times. Due to, simply, training multiple trees over the same dataset with placement that can easily generate high correlation between the formed estimators. The best way to introduce some sort of de-correlation between the trained trees is by feeding the algorithms different datasets. A new dataset can be formed from the original dataset by using the random subspace algorithm [55] to not only randomly choose the data points, but also concurrently pick randomly a feature from the feature space, to act as a new splitting point. Random forests use random subspace method to de-correlate the trees formed using the bagging method alone.

Random subspace algorithm is highly identical to bootstrap aggregation in many ways. The only difference is that in random subspace the features are the subject of bagging and they are considered as the "predictors" or "random variables" that would be sampled with replacement to create predictions for each learner. Thus, random subspace is also known as attribute bagging [56] or feature bagging. Random forest algorithm is a combination of bootstrap aggregation to sample the training dataset, and random subspace algorithm necessary to sample the features, to create splitting points that results in generating multiple estimators with high level of distinction and accuracy.

9. Relevant DL Literature

9.1 From Neural Networks to DL

Shallow and deep neural networks are defined as a machine learning algorithm inspired by the brain neurons functionality and logical processing flow. Neural networks were used in mathematics since 1943 [57], and the fundamental algorithm kept on developing, changing and getting more complicated until this day. Neural networks can solve supervised, unsupervised, semi-supervised and reinforcement learning problems.

In the following sections, a gradual demonstration of many deep learning techniques starting from the simplest form of neural networks also known as "the perceptron", reaching to various complicated deep learning schemas.

- Perceptron and Multi-layer Perceptron

The perceptron [58] represents the neuron or the basic building block of the neural network. The perceptron uses a simple mathematical operation calculated by the dot product between or sum of the inputs and their corresponding weights, which contributes to its decision making or output acquisition process. The perceptron consists of three main layers: input layer, one hidden layer and an output layer. The figure below, Figure 6 shows a perceptron with six inputs. As shown in the figure, the perceptron can only make binary decisions either 1 or 0. Furthermore, the perceptron has only one hidden layer, where each input is multiplied to its weight in a dot product operation and added to the rest of multiplications in sum of product operation. In other words, if we consider a perceptron with six inputs, the hidden layer value is calculated using $\sum_{i=1}^{6}(w_i\, x_i) + bias$. The bias is w_0 stored in the hidden layer. The output layer is where the classification decisions are made. The output layer also known as the activation function layer that can hold various types of activation functions which accordingly the classification result is selected. In perceptrons the activation function is a simple step function. The function below shows the activation function or the content of the output layer in the simplest neural network form as a perceptron:

$$f(x) = \begin{cases} 1 \;\; if \;\; \sum_{i=1}^{6}(w_i\, x_i) + b \geq 0 \\ 0 \;\; if \;\; \sum_{i=1}^{6}(w_i\, x_i) + b < 0 \end{cases} \qquad \text{Eq. 10}$$

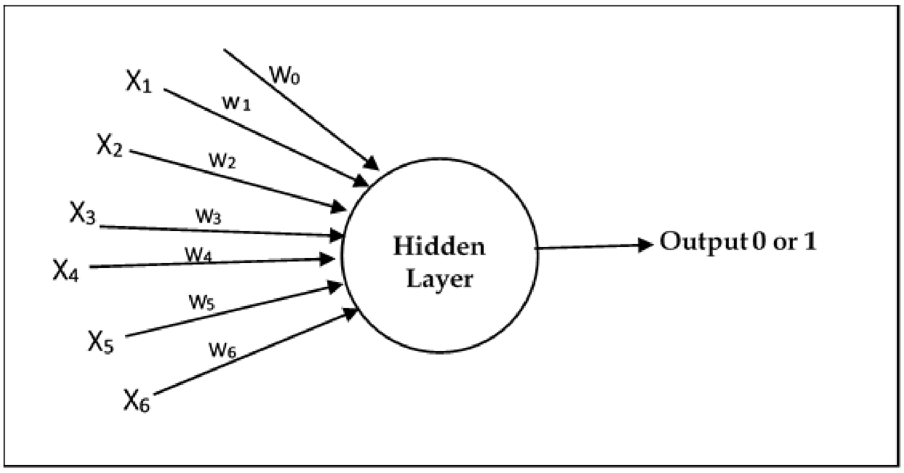


Figure 6 Example of a Perceptron in Neural Networks.

Before explaining any further, it is essential to clear the difference between the perceptron, multilayer perceptron and deep learning or deep neural network.

As explained before, the perceptron is the building block of any neural network and its main distinctive feature is the possession of only one hidden layer with only one neuron within. Although, multilayer perceptron has one hidden layer as in simple perceptron, it has various neuron within the hidden layer, each with their own vector of weights. The size of the weight vector in each neuron within the hidden layer is the number of the inputs plus one because of the additional value of w_0 the bias for each neuron in the hidden layer. Finally, deep learning is a chain of multiple hidden layers of multilayer perceptron. In other words, a deep neural network consists of many hidden layers connected sequentially to one another, where each has many neurons of different weight vectors. The output of the previous hidden layer contributes as the input layer of the next hidden layer. The overall output in the output layer is calculated by the sum of product between the outcomes of all neurons in the last hidden layer that act as inputs, and they will be multiplied to their corresponding weights using dot product. The figure below explains the difference between the perceptron, multilayer perceptron, and deep neural networks assuming the input layer has only two inputs. Meanwhile, w_0 in each layer represent the bias of this layer, W_{ij} shows the indices of the weights in a multi-layer perceptron when i is the input index and j is the index neuron connected to that input. In deep neural networks the weights on each edge are determined with at least three indices based on the shape of the neural network. W_{ijk} is how to identify weights of each edge in deep neural networks, where i and j are explained earlier, k represents the index of the hidden layer that the input neuron is connected to with the edge W_{ijk}. Each layer, the choice of the right weights for the edges is bounded to direction of the edge. The neuron that the edge is coming out of is always considered the input for this edge, even if this neuron located in a hidden layer. While the destination neuron is the hidden layer neuron for this operation. It is crucial to define the direction of the operation since not all neural operations are feed forward processes, but it can also be backward process as in the backpropagation neural networks [59]. WO_i is the weight connecting the last hidden layer to the output layer. The dot product between the outcome of the last hidden neuron with its corresponding output weight will be then added to the rest of the dot products calculated from all neurons in the final hidden layer, which all together will be sent to the output layer to be judged by the activation function content and criteria before the final classification, regression or re-enforcement decision is finally made.

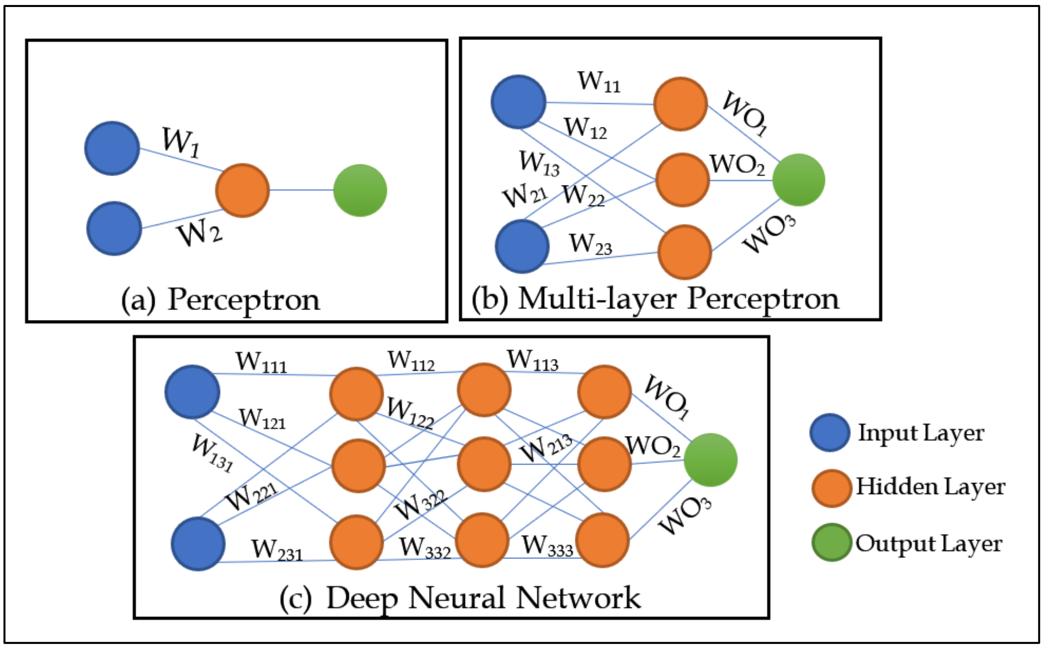


Figure 7 Comparison Between Perceptron, Multi-layer Perceptron and Deep Neural Network.

- Activation Functions

Activation functions are defined as a mathematical equation or a series of equations in which they are the output determinant in a perceptron or a multi-layer neural network. Choosing the suitable activation function has a critical impact in boosting the accuracy and enhancing the training process of the neural network no matter how simple or complex these networks are. Generally, it is highly advisable to experiment with various activation functions to ensure most effective results. Moreover, during the training stage tuning the activation function as a hyperparameter of the neural network can also contribute to the overall efficiency and accuracy.

The following are some examples of the most common activation functions:

- The Rectified Linear Unit (ReLU) function [60]: one of the famous activation functions that works efficiently with positive inputs, and it lacks precision with negative or positive approaching to zero input values. Some other activation functions are created based on ReLU such as, the leaky and the parametric ReLU.

- The Tanh function [61]: tanh is hyperbolic function that works better with centred or strongly on the edges positive or negative inputs. Which means it has better results with strong positives, strong negatives, and mean values to fit better with the hyperbolic plane shape.
- The sigmoid function: this function is distinctive for its S-shape curve or so called the sigmoid curve. The sigmoid function rule is derived from the logistic function operation and equation [62]. The sigmoid creates smooth gradient to find the optimal output between zero and one.
- Other well-known functions: Softmax [63] and Swish[64].

9.2 Convolutional Neural Network (CNN)

CNN [65] is a form of deep neural networks mostly applied to various imaging modalities, i.e. medical images, regular grayscale or RGB images and videos. CNN is often related to computer vision applications such as, images and video classification, image captioning and metadata production, facial recognition, object, and posture recognition, and medical image processing. CNN can also be applied to non-imaging applications as in time-series prediction and various natural language processing applications. CNN possesses many advantages over traditional machine learning approaches when dealing with images and image classification, that is because of its ability to internally select the most valuable features directly from the input images, without the need to manually apply feature engineering. However, with non-imaging inputs it requires more research to compare the feature engineering capabilities of CNN and the available ones in the literature.

- **CNN Architecture**

CNN has two main phases: feature engineering phase and classification phase. The feature engineering phase consists of two main layers: convolution and pooling layers. The classification phase includes a fully connected deep neural network schema that is trained using the feature vector that was created by flattening the feature matrix extracted from the earlier phase of the implicit feature engineering in CNN. The following figure, Figure 8 shows an example of a CNN network, while later each layer will be explained in comprehensive detail.

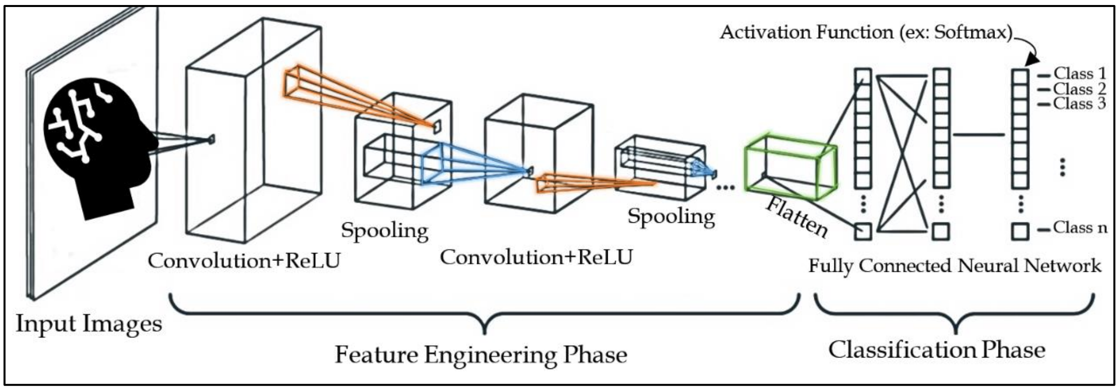


Figure 8 An Example of a Convolutional Neural Network Framework for Image Classification [66].

- **Convolution Layer**

The name CNN is created based on the convolution layer because it is the core of the CNN and its main building block. Moreover, the heaviest computational load occurs within this layer.

The kernel in CNN is defined by a matrix that is smaller in size of the original input image but have more depth. It is like applying masks or filters to images in image processing. If the image has RGB layers, the kernel would be able to perceive the three dimensions in the depth but smaller area of the x and y coordinates of the original image.

In the convolution layer the mask or the kernel will spatially slide over the input image, each time a dot product operation occurs between the visible part of the image under the kernel and the kernel. Then the kernel will slide to the next position of the image to calculate the next position's dot product value, the sliding size is called the stride. The convolution between the image and the filter or the kernel is called feature map or activation map. Let us assume the image matrix has the size $\mathrm{h} \times w \times d$, the kernel size is $\mathrm{h}_K \times w_K \times d_K$ the output after the convolution will have this size: $(\mathrm{h} - h_K + 1) \times (w - w_K + 1) \times 1$ assuming that the stride is equal to one sliding unit each time.

In some cases, the kernel does not fit the image properly, in this case a new concept called padding is used. Padding indicates adding values to pad the non-fitting cells between the kernel and the image. Padding can be applied by zero-padding which -by definition- means filling the extra cells with zeros, or another padding mechanism called valid padding, which drops the non-fitting part of the image, while only contributing the valid parts of the image.

In Figure 8, it is noticeable that ReLU is applied to the mapped features after the convolution process. ReLU implies adding non-linearity to the convolution operation. Since real-world problems and challenges are mostly non-linear, so by applying ReLU to the convoluted results it is more reflecting to the reality. The output of ReLU is calculated by: $f(x) = max\ (0, x)$, x is the value of each cell in the feature map. Many of the activation functions mentioned earlier can be applied to the convolution layer instead of ReLU such as, sigmoid and tanh functions.

- Pooling Layer:

Pooling layer is meant to reduce the spatial size of the feature map produced by the convolution process followed by non-linearity addition. Basically, pooling creates a rectangle that also slides across the resultant selected features and select only one value of the triangle to represent the whole one. The most popular types of pooling are max pooling, average pooling, sum pooling and L2 norm pooling of the rectangular neighbours.

The convolution layer and pooling layer should always follow one another in this order. However, this process can be repeated as many times as possible, especially if the original image is highly dimensional and requires adding additional feature engineering steps.

Before the selected feature matrix is feed into the classification phase, it is necessary to flatten the matrix representing the image while being feature selected and dimensionally reduced into a single vector, in which each cell of the vector is feed as an input to the fully connected neural network.

The rest of the CNN architecture represented by the classification phase is completely like the explanation provided in the neural network section mentioned earlier. Where all the activation functions also known as non-linearity layer are explained in detail.

9.3 Recurrent Neural Network (RNN)

RNN [67] is a type of neural networks specialized in sequential or time-series data such as, videos, audio, sensors' readings, text and the data record of a series of dependent events overtime. RNN elementary neuron functions by accepting a sequence of input variables each time, while keep tracking or remembering the previous input in the sequence, assuming that the previous occurrences could carry valuable information, in which can lead or enhance the process in the current step.

9.4 Long Short-term Memory (LSTM)

LSTM [68] is a type of deep artificial neural networks that follows RNN architecture. Unlike traditional neural network with feedforward process flow, LSTM provides a connection between the current point and the previous ones. Hence, LSTM has the ability to process sequential datasets such as speech, audio, and time-series data in general. Speech recognition, fault detection and handwriting detection are some common applications of LSTM.

A LSTM unit is highly similar to RNN in which both propagates forward the input sequences to learn dependencies from previous periods. However, the internal structure of LSTM is more sophisticated than the regular RNN cell, which offers various gates or regulators, that provide the selectivity between keeping or forgetting certain sequences rather than accepting all the memories from previous cells blindly.

LSTM is applicable to various machine learning problems over time-series data, i.e., classification, one point or a sequence prediction and reinforcement learning. Generally, a LSTM unit consists of the cell and three more gates: one input, one output and one forget gates. The cell is responsible of remembering the states of inputs over various arbitrary periods of time. The remaining gates basically regulate the information flow from the beginning of the cell until the decision is made at the end of the cell.

More specifically, LSTM units has various architectures. However, the most common components among them are the *cell*, which represents the memory of the LSTM unit, and three *regulators* also known as gates that include *input gate*, *output gate*, and *forget gate*. It is essential to point out that not all LSTM composed of all the three gates combined, for example the LSTM architecture called Gated Recurrent Units (GRUs) does not compose of output gate. Moreover, some LSTM architectures may contain additional gates to regulate the data flow in and out the LSTM unit as needed.

The cell is responsible of pointing out the dependencies between the input elements or given sequences. The memorized dependencies determine the importance of the current memory to the next LSTM connected unit. The input gate controls the new sequences flowing into the LSTM unit. Whilst the forget gate determines which input sequences to forget and which to remain the LSTM unit. Finally, the output gate controls the values computed and used in the output activation function at the end of a LSTM unit. The activation function used in LSTM is the sigmoid function [69]. Furthermore, tanh [61] activation function is also used internally to compute different gates and intermediate stages in the LSTM unit.

The following figure explains the common architecture of LSTM units, the information flow and the main function and operations within. As shown in Figure 9, The forget gate F_t uses the summation of previous hidden state H_{t-1} also known as the previous unit prediction, and the input sequence X_t passed through a sigmoid function to determine if the information derived from the input sequence based on the previous knowledge is worthy of keeping or better to forget. As it is known for sigmoid functions, the output is between zero and one. If the forget gate results a value closer to one that means store the information. However, if closer to zero forget the processed information.

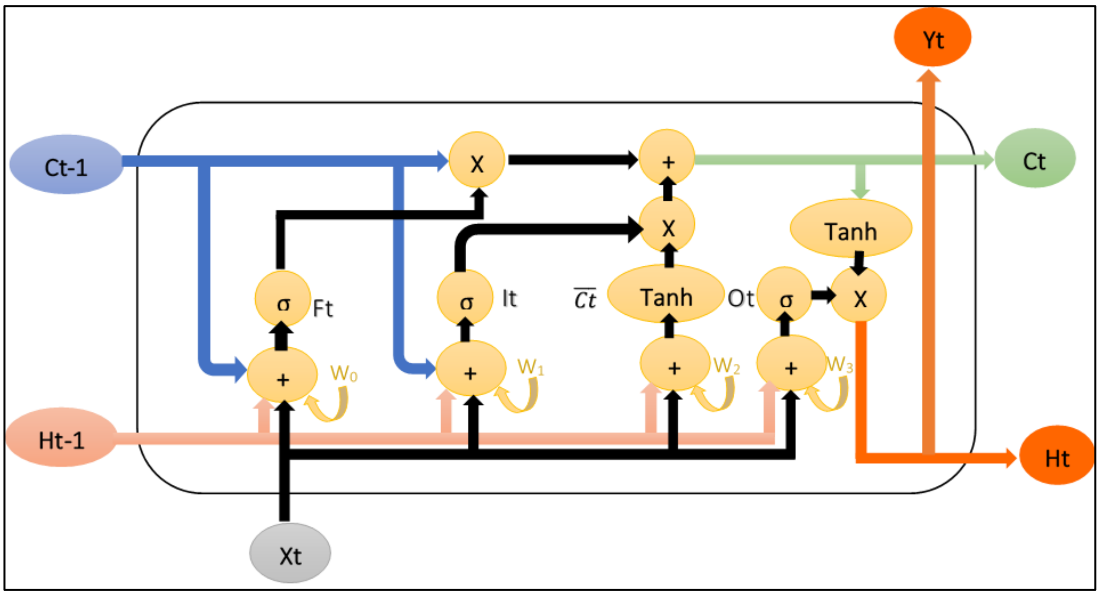


Figure 9 LSTM Cell Common Architecture

Input gate I_t is crucial to provide important parameters to update the current cell state C_t. First, the summation of the previous hidden state H_{t-1} and the input sequence X_t is passed into a sigmoid function to determine which input information is important (sigmoid close to one), or not important hence ignore (sigmoid close to zero). There is an intermediate stage for the current state called $\bar{C}_t$ can be calculated by passing the sum H_{t-1} and X_t into a tanh function, which contributes to normalizing the sum of product between -1 and 1.

The cell state C_t can be calculated by two steps: (a) the element-wise multiplication between the previous memory or previous cell state C_{t-1} , and the forget gate F_t calculated. To determine which previous memories to keep or to forget in case it gets multiplied by a zero from the forget gate. (b) The element-wise addition of the values computed in (a) and the newly added important information from the input gate I_t, plus the intermediate current memories $\bar{C}_t$ calculated by the tanh function.

At last, the computation of the output gate O_t is in order. The output gate is the determinant of the current hidden state H_t that would be passed to the next LSTM unit as the previous hidden state H_{t-1}. The hidden state H_t of the last LSTM unit connected represents the output of the overall LSTM network that is used for predictions and decision making. The output gate O_t value is calculated by passing the summation of the previous hidden state H_{t-1} and the input state X_t into a sigmoid function. The value of H_t is calculated by multiplying O_t resulted from the output gate by the outcome measured by passing the current cell memories C_t into a tanh function.

The descriptive information derived from Figure 9 and explained in detail above, can be comprehended and easier to understand by equations. Thus, the following are the equations for the calculations required by each gate in the LSTM unit. Each gate has different weight vector, but for the sake of simplicity they were denoted by W and U. The symbols subscripted under the weights represents the gate the weight vector belong to. Where f is for forget gate, C for cell state, i and O are for input and output gates weight vectors, respectively.

$$F_t = \sigma\,(X_t \times U_f + H_{t-1} \times W_f) \qquad \text{Eq. 11}$$

$$I_t = \sigma\,(X_t \times U_i + H_{t-1} \times W_i) \qquad \text{Eq. 12}$$

$$\bar{C}_t = tanh\,(X_t \times U_c + H_{t-1} \times W_c) \qquad \text{Eq. 13}$$

$$C_t = \sigma\,(C_{t-1} \times F_t + \bar{C}_t \times I_t) \qquad \text{Eq. 14}$$

$$O_t = \sigma\,(X_t \times U_O + H_{t-1} \times W_O) \qquad \text{Eq. 15}$$

$$H_t = tahn\,(C_t) \times O_t \qquad \text{Eq. 16}$$

9.5 Encoder-Decoder and Autoencoders

Encoder-decoder [70] is a machine learning approach also known as Seq2seq, where it is required to predict a sequence of observations by learning patterns of other sequences of different size, or even different nature from the target sequence. Encoder-decoders are widely used in image captioning, summarization of texts, natural language translation and many more. The first Seq2seq algorithm was developed by google to achieve machine translation. Seq2seq rely on turning one sequence of inputs into another sequence of outputs using RNN and mostly LSTM and GRU. Encoder-decoders construct of an encoder that turns each input sequence a hidden vector. A decoder that receives the hidden vector encoded by the encoder and reverse it back to an output item of the same nature of the output category or group.

Encoder
Hidden Vector
Decoder

Figure 10 Encoder-Decoder General Structure

Previously, encoder-decoder framework was explained. To rephrase it, decoder-encoders are a type of RNN functions which maps a sequence from an input space, to another sequence from an entirely different input/ feature space, and probably both sequences own variable sizes as well. i.e., in image captioning, the input is an image of a certain size, shape and extracted features. While the caption is a text owning entirely variant feature space and sequence size than the images, they describe them. Another example is in natural language translation, when the input sequence is an English sentence, while the output is a French sentence for example.

On the other hand, autoencoders [71] are a special type of encoder-decoders that map two sequences to one another, while the target and source are both of a similar feature space, nature, and size. As an illustration when mapping one English sentence to another English sentence of the same size. Furthermore, autoencoders are most popular in sequence reconstruction, which means learning previous sequences to reconstruct the next or missing sequences of the dataset. Finally, both encoder-decoders and autoencoders can be applied for various types of deep learning algorithms such as, CNN, RNN. LSTM and GRU.

10. Other Relevant Literature

10.1 Principal Component Analysis (PCA)

PCA [72] is a dimensionality reduction technique falls under feature extraction category. The concept of PCA is dependent on projecting the data perpendicularly on hyperplanes that possess the most variance of the data also called "principal components (PCs)". Many data points have similar projections on the principal component hyperplanes, which provides a way to reduce the number of features by using their projections instead.

The chosen components are the linear combination of the data features which provides the most variance, as well as each component is orthogonal to one another to eliminate the chance of any correlation between them (correlation= 0). Hence, to ensure avoiding the data redundancy as possible.

The 1st PC carries the maximum variance where it has the greatest number of data points, followed by the 2nd PC that has the remaining points that did not correlate with the 1st PC. The selection of further PCs like the 3rd and 4th…etc is determined by the same technique as the 2nd PC, demonstrated by finding the maximum variance in the remaining uncorrelated data points left from the previous PC.

10.2 Feature Importance (FI)

FI is a feature selection method created based on RF. In the context of RF, *Mean Decrease Accuracy (MDA)* or *permutation importance* or *feature importance* [52] and [73], of a variable X_n to predict Y of classes is computed by the summation of the Gini impurities of X_n for all the nodes d where X_n is present and used. Followed by the mean of the impurity decrease metric of all the trees D in the forest. The following equation comprehend the concept of feature importance using RF.

$$FI(X_n) = \frac{1}{no.Trees} \sum_{no.Trees} \sum_{d \in D : v(s_d) = X_n} Gini\_Impurity(X_n) \qquad \text{Eq. 17}$$

Where X_n is the feature of interest. $v(s_d)$ is the feature/variable used to split s_d.

11. Data Collection and Generation

11.1 Condition Monitoring of Hydraulic Systems Dataset

This dataset [74] represents real measurements of multivariate, time-series sensors, placed in a hydraulic test rig. The purpose intended for the data collection is to monitor and assess the hydraulic system health condition.

The outcome of this experiment yielded a success of collecting sensor data of various system health degrees of different components of the hydraulic system, such as the cooler, valve, pump, and accumulator.

The system consists of six pressure, four temperature, two volume sensors and one vibration sensor which all possess a constant cycle of 60 seconds. Each cycle the sensors are collected, while the condition of the four main hydraulic components; cooler, valve, pump and accumulator are monitored and observed. The component health ranges from completely healthy to totally damaged, and each condition degree is decoded into a numerical value to facilitate the application of statistical and data mining approaches, as shown in the following table, Table 2 .

Table 2 Hydraulic System Fault Degrees and Their Codes

Cooler condition		Valve condition		Internal pump leakage		Internal pump leakage	
3	close to total failure	100	optimal switching behaviour	0	no leakage	130	optimal pressure
20	reduced efficiency	90	small lag	1	weak leakage	115	slightly reduced pressure
100	full efficiency	80	severe lag	2	severe leakage	100	severely reduced pressure
		73	close to total failure				

This dataset has been used by many researchers to perform sensor fault monitoring. i.e. constant, shift, bias and peak. [2]. Moreover, this dataset is also beneficial to perform component or system FDD such as the application researched in [76]. As well as, being applicable for creating and testing feature extraction and selection algorithms in [77].

Chapter 3: Relevant Related Work

1. Supervised ML Approaches for FDD in Mechanical Machinery

Parts of this research have been used in our published research in [78]. Please note that all rights and copy rights have been reserved to the MDPI publisher.

Griffin et al. [79] proposes an approach that mimics real-industrial processes, by investigating two main machine processes and their faults at the same time: (1) grinding: an example of grinding faults are chatters and grinding burns. (2) Hole making: the faults that can be associated to hole making are drill tool onset faults, drilling malfunctions and tool gradual wear. The approach shows a combination of neural networks and CART to provide a robust classification. A Makino A55 machine is calipered to be used in both the grinding and the drilling experiments. The work in [80] developed a fault detection and monitoring algorithm for spur gears based on decision trees. The vibration signals extracted from the spur gear go first through feature selection applying the time-domain features. i.e., sum, mean, skewness, minimum, maximum, and so on. Then the selected features are fed in the CART model. In [81] an improved CART algorithm to achieve fault diagnosis in refrigerant flow systems is introduced. The results conducted from the improved CART are compared to the regular CART, RF and Generalized Boosted Regression (GBR). The improved CART has shown better results in comparison of the previously mentioned methods. Additionally, the work in [82] provided an intelligent approach of applying rotation forests ensemble of C4.4 CART, to achieve fault diagnosis in wind turbines. Seven sensor readings are used to test and validate the improved model without any feature engineering required.

For the last few decades, RF has been used widely to perform FDD and monitoring applied in various fields and applications, such as industrial systems. The literature demonstrates several techniques to apply RF for the purpose of outlier detection, either exclusively or incorporated with other algorithms to form some sort of a hybrid approach aimed to fulfil an intended research or applicational purpose. The most common methodology of deploying RF is as a classifier. RF is intended to achieve an optimized, supervised, and structured resolution for labelled problems, which is proven to have more accurate results comparing to many other supervised machine learning algorithms. In [83] RF is compared to numerous classifiers of different functionality to overcome two occurring sensor faults in Wireless Sensor Networks (WSNs), which are spike fault and data loss fault. This study represents an elaborated comparison between RF, SVM, Stochastic Gradient Descent (SGD), Multilayer Perceptron (MLP), CNN, and Probabilistic Neural Network (PNN). Using Detection Accuracy (DA), Matthews Correlation Coefficients (MCC), True Positive Rate (TPR), and F1-score as the comparison criteria that determine the overall rank of each method. As a result, RF is proven to have the highest rank of all the above classifiers in WSN's sensor fault classification. In addition, another study in [84] showed similar results in proving the superiority in performance of RF in the field of WSN, but this time while detecting four different sensor faults; gain, offset, constant and out of range faults.

In [85], another example of using RF in a solo fashion to achieve FDD in industrial sensor systems applied to unmanned aircraft vehicle. This study deployed a brilliant interpretation of RF and feature importance, to extract a weighted similarity metric based on the data priority represented by RF. The induced similarity measure is then used to perform FDD.

RF can also be used combined to different approaches instead of using directly as a classifier to achieve FDD in industrial systems. Usually, any hybrid approach is originated to optimize the individual forming methods combined, or to establish a customized solution that fulfils additional system goals or requirements. In [86], a hybrid approach is established to detect faults of rolling bearings, which if left undetected can lead to major consequences in the performance of the rotating machine. This hybrid approach combines Wavelet Packet Decomposition (WPD) method to extract new enhanced features from the bearing vibration signal provided from n number of sensors, using signal-to-noise ratio and Mean Square Error (MSE). Followed by the step of mutual, dimensionless index construction, which will be fed to the fault database and contribute as the data necessary to train and test the RF model. Moreover, another example for a hybrid FDD approach using RF is [87], This method demonstrates the effect of combining genetic algorithm and RF to increase the classification accuracy of the FDD process of an induction motor.

A parallel RF cloud-based approach in [88] is introduced, to predict and monitor the wear of dry milling tool operations. 28 Statistical features are extracted from the row data collected from various channels of the dry milling tool such as, cutting force, acoustic emission, and vibration. Furthermore, an RF algorithm for predictive maintenance in wind turbines at real-time is proposed in [89]. A new approach of deep RF fusion in [90] is utilized to enhance the fault diagnosis process in gearboxes using two feature modalities: (1) acoustic emission sensors and (2) vibrational signals extracted from accelerometers. The best features from both modalities are selected using wavelet packet transform.

Beyond the intensive use of RF in industrial sensor systems, RF can be used in a smaller range, for many reasons and purposes exceeding the industry. One of the common applications of RF is in the medical field using sensing modalities. In [91] a recent study shows an application of RF to reduce the fallacious clinical alarms. i.e., the Arrhythmia alarms. In case of false Arrhythmia alarms occurrences, that may lead to elevation in the patient and staff stress level, as well as causing unnecessary pressure on the intensive care staff. According to the study, the application of RF detects the true from the faulty calls has significantly reduced the number of false calls concerning five main types of arrhythmia. In this recent work [92], RF is used in a hybrid fashion with Feedforward Neural Network (FNN) to investigate the relationship(s) among multi-modal signals, extracted from electrochemiluminescence (ECL) sensor located in a smartphone and the concentration of $Ru(bpy)_3^{2+}$ luminophore and its electrochemical data. Establishing such correlation is essential for building optimized and cheaper diagnostic devices. Understanding the hidden relationships between each modality may lead to creating diagnostic rules, which can be used for FDD in later stages. Thus, this study is included with the application of RF in FDD related work.

The work presented in [93] demonstrates a component fault detection using kernel-SVM, applied to auxiliary marine diesel engine. The kernel applied is multi-variate gaussian kernel. The work presented in [94] focuses on the utilization of multi-class classification using SVM classifier, applied on the field of semiconductor manufacturing to achieve predictive maintenance. The entire feature space included 31 features, and six main features are extracted using time-domain statistical and mathematical calculations such as, maximum, minimum, mean, variance, skewness, and kurtosis. In [95] a data-driven approach to predict component faults in air craft systems using SVMs is proposed. Where the prediction process is used to perform maintenance when needed. The created method is called Auto-Regressive Moving Average (ARMA). Six different classifiers are compared within ARMA's interior architecture to determine the best fitted algorithm for the problem, including

KNN, Generalized Linear Regression (GLR), ANN, RF and SVM. The application of SVM within ARMA showed best results comparing to the rest of the classifiers. The data gathered for this work is collected from a real industrial operation of an aircraft engine that suffers from critical valve malfunctions. Followed by applying PCA to extract the optimal features prior the classification. Moreover, the research in [96] demonstrates the application of multi-class SVM to build a diagnostic model of component faults in rolling bearings. The vibration signals are used as the input signals, followed by the application of time-domain feature selection to extract the appropriate features. A deep SVM for multi-class classification approach is researched in [97]. The word "deep" in this context indicates the usage of various types of features including the fault diagnosis process. Deep SVM is applied to detect component faults in gearboxes using various homologous features, where their time, frequency and wavelet natures are extracted from the original data. LR is widely used for FDD in mechanical machinery. The following related work shows several research applications using LR. The research proposed by Li *et al.* in [98] introduces an FDD approach that combines LR with acoustic emission to ensure the reliability of various cutting tools during the manufacturing process. The aim of this research goes beyond FDD of cutting tools, but also monitors the tear and wear of these tools and estimates the best time to perform maintenance. The feature selection method applied in this research is wavelet packet decomposition. The work shown in [99] represent a monitoring and prognosis system for gas circulator units applying a joint approach of LR and linear SVM L1-regularizer. This method implicitly selects the most distinctive features. Thus, no other external feature engineering methods are applied. Pandya *et al.* in [100] introduced an FDD approach of rolling bearing devices, in which the application of multinomial LR is applied and its effectiveness is compared to SVM and ANN. Multinomial LR is proven to have the best accuracy results comparing to both SVM and ANN. The feature engineering method applied to this research prior to the classification is wavelet packet decomposition. In a similar note, Caesarendra *et al.* [101] focuses on building a machine degradation analysis model using bearing run-to-failure datasets. The proposed approach is a combination of LR and Relevance Vector Machines (RVM). LR is used to detect the degradation status, and its results are used as labels for degradation probability estimation in the following step performed by RVM. An FDD approach applied on micro-piercing process is developed in [102]. The proposed approach represents an online vibration-based monitoring and FDD system using LR. LR is applied to the selected features to achieve fault detection and monitoring at run-time. Statistical feature engineering method is applied to select the most optimal features extracted in both time and frequency domains. Moreover, a fault detection and prediction approach using dynamic

LR applied to rolling bearings is introduced [103]. The approach is tested and validated on PRONOSTIA dataset [104]. Similarly, the work in [105] LDA classifier is applied to achieve sensor fault identification in hydraulic systems. A simulated hydraulic system benchmark is used to extract the sensor readings, which then are statistically engineered to extract time-domain features such as, mean, variance, skewness, and kurtosis. Before finally applying them to the classification model for training and later for testing. To sum up, the following table compares the previously mentioned ML supervised approaches to achieve FDD in mechanical machinery in the past decade.

Table 3 ML Supervised Approaches for FDD in Mechanical Machinery for the Past Decade [78].

Classifier	Reference	Mechanical Equipment	Feature Engineering	Fault Type/Purpose	Dataset
CART	[79]	A Makino A55 machine for grinding and hole making	-----	Component faults associated to grinding and hole making processes.	Dataset extracted from Makino A55 machine
	[80]	Spur gears	Time-domain statistical features	Component faults in spur gears.	Fault simulator.
	[81]	Refrigerant flow systems	-----	Component faults in Refrigerant flow systems.	Real commercial buildings, and a VRF system.
	[82]	Wind turbines	------	Component fault diagnosis in wind turbines.	From a physical test bed.
	[106]	Electropneumatic brakes.	-----	Isolate sensor faults in electropneumatic brakes.	An actual Locomotive electro-pneumatic brake (DK-2).
RF	[88]	Dry milling tool operations	Time-domain statistical features	Predict and monitor the wear of dry milling tools.	Obtained from this paper [107]
	[89]	Wind turbines	------	Predictive maintenance in wind turbines at real-time.	From actual wind turbine within 2 years.
	[90]	Gearboxes	------	Component fault diagnosis in gearboxes	From a simulator.
	[86]	Rolling bearings	Wavelet Packet Decomposition (WPD)	Component fault diagnosis in rolling bearings.	From an actual system.

	[85]	Unmanned aircraft vehicle	FI	Component FDD in unmanned aircraft vehicle	From a Physical aircraft vehicle
SVM	[93]	Diesel engine	------	Component faults in diesel engines	Data extracted from auxiliary marine diesel engine.
	[94]	Semiconductor manufacturing	Time-domain statistical features	Component faults in semiconductors	From implanter tool
	[95]	Aircraft engine	PCA	Component fault prediction and maintenance in airlines	From real aircraft engine valve.
	[96]	Rolling bearing	Time-domain statistical features	Component faults in rolling bearings	From six test bearings.
	[97]	Gearbox	Time, frequency, and wavelet domain features	Component faults detection in gearboxes	Gathered from UPS.
LR	[98]	Cutting tools	wavelet packet decomposition	Wear and tear evaluation of components in cutting tools.	Dongyu machine and tool CMV-850A centre
	[99]	Gas circulator units.	-----	Component faults in gas circulators estimation.	EDF energy
	[100]	Rotating bearings	Wavelet packet decomposition	Component faults in bearings to estimate their degradation.	A test bearing rig
	[101]	Bearings	-----	Component faults in bearings to estimate their degradation.	MATLAB simulation
	[102]	Micro-piercing Process	Statistical feature engineering	Component faults in micro-piercing devices detection.	Readings of an actual machine.
	[103]	Bearings	-----	Component faults detection in bearings	PRONOSTIA dataset [104].
LDA	[105]	Hydraulic systems	Time-domain statistical features	Sensor and component fault identification in hydraulic systems.	A simulated hydraulic system benchmark.

2. Autoencoder Approaches for FDD in Mechanical Machinery

Parts of this research have been used in our published research in [78]. Please note that all rights and copy rights have been reserved to the MDPI publisher.

The work in [108] shows a combined approach to achieve component fault detection and diagnosis of rare events occurring in chemical factories. The proposed method joints LSTM autoencoder as the detection phase, followed by the diagnosis phase using LSTM classifier. This approach is used to detect and diagnose faults of the Tennessee Eastman benchmark [109], which represent a dataset extracted from a simulator of actual chemical processes that includes various components: reactors, condensers, vapor-liquids… and so on. In the detection phase, the sequence comparison between the reconstructed sequence and the given one is achieved by applying the traditional signal difference. In the diagnosis phase, no feature selection or extraction approach is used prior to the classification using LSTM classifiers. Moreover, a solo comparison to CNN is made, but no comparisons with other DL or ML classifiers are conducted.

Lu et al. [110] introduced a novel autoencoder called Stacked Denoised Autoencoder (SDA) that is used to detect component faults in rotary machineries. The method is applied to a dataset extracted from a physical simulation of a bearing test-rig. SDA implicitly feature engineer the data, which is compared to PCA and regular stacked autoencoders (SAE). Moreover, the classification results provided by SDA are then compared to SAE, SVM, RF and regular autoencoders.

The work proposed in [111] shows a novel approach of creating a new type of autoencoders, in which it combines stacked autoencoders and LSTM network. The work is separated into two-phases: (1) feature transformation using LSTM stacked autoencoders. (2) Apply LSTM for fault identification. The proposed method focuses on detecting injected component faults to a Bently Nevada Rotor Kit RK3, which is designed to physically simulate rotating equipment and its conditions. The raw vibrational signals are directly collected from the RK3 kit, then Wavelet Packet Decomposition (WPD) method is used to select features in both time-domain and frequency-domain, to ensure a wide investigation in both domains, followed by transforming the selected features using the stacked autoencoders in account to their mean square error calculated, which helps in generating a threshold for each feature. Finally, the fault detection accuracy for each feature is validated using five-fold cross validation after classification using KNN method. No comparisons of other feature selection methods to WPD, or additional classifiers besides KNN are used in the mentioned work.

According to [112] a component fault diagnosis system of rolling bearings using stacked autoencoders is introduced, as well as compared to two other deep learning schemas: (1) deep Boltzmann machines and (2) deep belief networks. Four experiments are conducted using various data pre-processing schemas using time-domain, frequency-domain, and time-frequency domain.

As stated in [113] a deep autoencoder is developed to diagnose vibration signals in both gearboxes and electrical locomotive roller bearings. The novel approach proposed consist of two steps: (1) the design of the deep loss function in the autoencoder using maximum correntropy. (2) Applying artificial fish swam algorithm to optimize the autoencoder's parameters and its ability to extract valuable features.

Similarly, the approach proposed in [114] demonstrates a new method of combining wavelet transform and stacked autoencoders, to diagnose faults occurring in roller bearing systems.

Furthermore, a deep autoencoder in [115] is used to develop the quality of feature fusion, which contributes in aiding the diagnosis of faults in rotating machinery. The applied autoencoder is a collaboration between denoising autoencoders and contractive autoencoders, where the deeply extracted features from both methods separately are then fused together using Locality Preserving Projection (LPP). The fused features are then applied to SoftMax function to train the diagnosis process.

In addition, another architecture of sparse autoencoders is performed in [116] to monitor and diagnose the component faults in motors and air compressors. The application of regular ML classifiers such as SVM requires intensive understanding and expertise in feature engineering. Thus, the application of autoencoders can massively facilitate the feature engineering process and perhaps outperform the regular feature engineering approaches. For that matter, sparse autoencoders are compared to other ML fault diagnosis methods such as, SVM and SoftMax regressor to classify faults in motors and air compressors.

Accordingly, in [117] a multivariant fault diagnosis and health monitoring approach in rotating machines is introduced. This method is called "SAE-DBN" as a combination of a two-layered Sparse Autoencoder (SAE) to perform data fusion between the features of multi-sensors followed by the application of Deep Belief Networks (DBN) for the diagnosis.

In [118] another approach using sparse autoencoders is proposed. The method is applied to induction motors monitoring and fault diagnosis purposes.

The autoencoder application in [119] shows an ensemble, and deep approach of autoencoders designated to fault diagnosis in rolling bearings. Various activation functions are deployed at the same time, to create multiple autoencoders that are going to be combined later using a novel strategy.

Finally, the work in [120] investigates fault detection and feature extraction schema for motors using an autoencoding schema of RNN networks. The explained schema for fault classification is applied directly on time-domain vibrational data then compared to the results conducted by a two-layered ANN model. On a different note, the feature selection capacities of the RNN autoencoder was compared to PCA and LDA for dimensionality reduction. The vibrational signals used in this work were obtained form an actual motor positioned with different accelerometers in various locations.

The table demonstrated below is created to conclude all the autoencoding FDD approaches in mechanical machinery performed in the past decade.

Table 4 Autoencoding-Based Methods for FDD in Mechanical Machinery [78].

Reference	Autoencoding Method	Mechanical Equipment	Fault Type/ Purpose	Dataset
[108]	LSTM autoencoder+ LSTM classifier	Chemical reactor	Component faults of Tennessee Eastman benchmark.	Tennessee Eastman benchmark[109].
[111]	Stacked autoencoder LSTM + KNN	Rotating equipment	Injected component faults to a physical simulation	Data collected from Bently Nevada Rotor Kit RK3 to simulate rotating device.
[110]	Stacked denoised autoencoder	Rotary machinery	Component faults in a bearing test-rig	Data extracted from physical bearing test-rig.
[112]	Stacked deep autoencoders	Rolling bearings	Component faults in rolling bearings.	Gathered from UPS.
[113]	Another architecture deep autoencoder	Gearboxes and electrical locomotive roller bearings	Component faults in rolling bearings and electrical locomotive.	From a physical test rig.
[114]	Wavelet transform +	Roller bearing systems	Component faults in rolling bearings.	From case western reserve university (CWRU) [121].

	stacked autoencoders			
[115]	Another architecture of deep autoencoders	Rotating machinery	Component faults in rotating machinery	Physical rotor fault test, CWRU [121] and NASA datasets [122].
[116]	Another architecture of sparse autoencoders	Motors and air compressors	Component faults in motors and air compressors	Actual air compressor and motor
[117]	SAE-DBN (sparse autoencoder + deep belief networks)	Rotating machines	Component faults in rotating machinery	Extracted from an experimental system.
[118]	Another architecture of sparse autoencoders	Induction motors	Component faults in induction motors	Fault simulator.
[119]	Ensemble deep autoencoder	Rolling bearings	Component fault diagnosis in rolling bearings	CWRU [121].
[123]	Another architecture of stacked autoencoders	Hydraulic pumps	Detect component faults in hydraulic pumps	Hydraulic pump of type axial piston pump (25MCY14-1B).
[120]	Autoencoding schema of RNN networks	Motors	Component fault detection and feature extraction in motors	Physical motor.

3. *k*-means for Feature Selection Related Work

Parts of this research have been used in our published research in [124]. Please note that all rights and copy rights have been reserved to the pre-print publisher.

On one hand, the literature is rich with review research papers related to feature selection methods. However, the vast majority of these review papers are focused on supervised and semi-supervised methods. The research in [28] and [27] represent a thorough analysis of various supervised and semi-supervised algorithms, along with a quick glance at few unsupervised techniques for feature selection. In [125] an inclusive research is done investigating various semi-supervised techniques in various fields and applications. Finally, the work in [126] introduce a new perspective for supervised feature selection methods, including more recent studies and different taxonomies comparing to the ones described in the latter papers.

On the other hand, a few research studies concentrated their efforts to analyse unsupervised methods for feature selection such as, the work in [36] where they pointed out the lack of survey research in this area, and offered a detailed analysis of numerous unsupervised methods along with summarising their advantages and disadvantages, as well as an experimental comparisons between them. The work in [127] narrowed down the scope of the research in [36] and instead, it focuses specifically on clustering algorithms for feature selection providing various clustering techniques for generic, text, streaming and linked data. Moreover, they finalized their review with some challenges that clustering algorithms for feature selection witness and elaborated with some suggestions to overcome the proposed challenges.

In this review, we narrowed down the scope even more, to include clustering feature selection algorithms using *k*-means clustering alone. This work is essential since *k*-means clustering for feature selection has already a huge amount of literature with different strategies and mechanisms, which creates the need to add some structure and taxonomy for this influx of studies, to facilitate navigating through them, as well as building up new literature following the legitimate path.

According the literature in the past decade, it is prominent that *k*-means for feature selection can divided into the following main categories based on their clustering strategy and the included mechanisms.

(1) *k*-mean hybrid approaches: which includes a combination between *k*-means and other wrapper feature selection or filter feature selection methods. (2) *k*-means based on feature weighting or ranking: this group depends on assigning some weights to the features and rank them accordingly to measure their relevance, followed by choosing the highest ranked features as the selected ones. (3) *k*-means with correlation measures: this method uses the similarity measures between features as the decision criteria. (4) Sparse k-means Feature Selection methods.

Figure 11, shows the four main categories of *k*-means for unsupervised feature selection proposed in this literature review. Based on the acquired knowledge and understanding of the work in the literature.

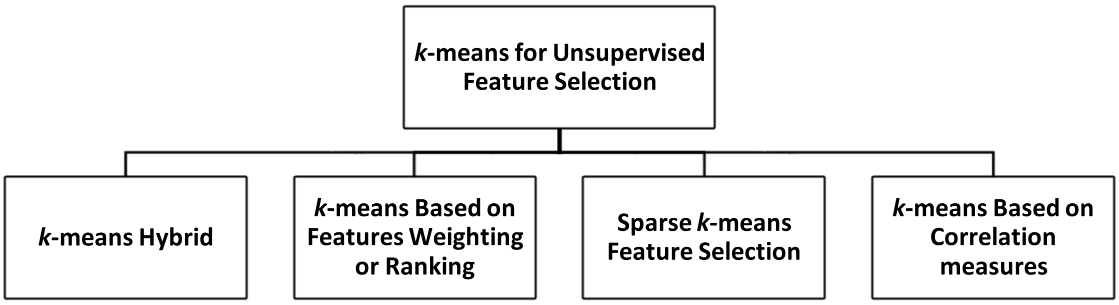

k-means for Unsupervised Feature Selection
k-means Hybrid
k-means Based on Features Weighting or Ranking
Sparse *k*-means Feature Selection
k-means Based on Correlation measures

Figure 11 k-means for Unsupervised Feature Selection Proposed Taxonomy [124].

The following section represents a full explanation of each sub-category of the previously mentioned taxonomy. A comprehensive analysis and overview of numerous related works for each category is also explained.

3.1 *k*-means Hybrid Approaches for Feature Selection

The work in [128] is an example of feature ranking methods, which can also be grouped as hybrid approach between filter and wrapper methods. The filter stage used in this algorithm is introduced in [5]. In addition, the wrapper method included is used to determine the separability criterion following the algorithm explained in [31]. This method consists of two main stages: filter and wrapper. In the filter stage an entropy elimination calculation technique is used, where the process is initialized with a full dataset, and then the features are being eliminated individually while the entropy is being computed during the elimination process. As a result, a list of features and their entropy is formed and can be sorted, which allows the features with higher entropy values to be excluded. The second phase is the wrapper phase, during this stage *k*-means clustering is applied to the remaining feature, and the cluster separability criteria used is the scatter separability. Finally, the feature subset with the highest scatter separability is selected.

In [129] another hybrid approach is introduced. Evolutionary Local Selection Algorithm (ELSA) is an unsupervised feature selection algorithm that computes the number of clusters *k* and the feature subsets, by using the combination of *k*-means clustering and expectation maximisation embedded to

Gaussian Mixture clustering. The quality of the cluster is determined by three main criteria based on the maximum likelihood, the separation criterion, and the cluster cohesion. ELSA is validated using numerous synthetic and real-world datasets.

In [130] a hybrid approach between wrapper and filter methods is proposed. This method contains two main phases: the first is the wrapper phase which starts by applying *k*-means clustering to the input dataset using an upper range of cluster numbers specified by the user, followed by applying simplified silhouette measurement as the separation criterion, then the feature subsets with higher silhouette value is selected. Note that this method is not a feature ranking *k*-means algorithm even though silhouette criterion is applied, because silhouette criterion here is only an intermediate stage for the feature selection and part of the first phase only, which is not directly contributing to the final decision of the selection process. The second phase for this algorithm uses Bayesian network as the filter approach to select the best feature subsets. Moreover, this method generates Bayesian networks in the form of directed graphs of the nodes representing the features selected, and the edges connecting them are the relationships between features.

3.2 *k*-means Based on Features Weighting or Ranking

The general idea of *k*-means for feature selection based on feature weighting begins with clustering the dataset into *k* main clusters. Followed by, using variations of strategies to assign weights to each feature or a features' subsets in some literature, in a way that the feature or subset of features that minimizes the inter-cluster distance and maximizes the intra-cluster distance is assigned higher ranks or weights. The type of measurements or process responsible for assigning weights or ranks to a feature or a groups of features during clustering are called 'clustering criteria' [131]. Although, the literature introduced numerous clustering criteria, the oldest and most common ones are the silhouette criterion [40] and Davies-Bouldin index (DB) [132], where they contributed as base methods for modern weighting criteria nowadays.

The work in [133] showcases a feature selection method based on the application of *k*-means along with Fisher ratio. Fisher ratio is used as the clustering criterion that reduces the ratio between the mean intra-cluster to the mean inter-cluster dispersion. Several clustering attempts using different feature subspaces is generated, while the ones with the smallest Fisher ratio is chosen to as the final subspace of features.

In a similar way, the feature weighting method proposed in [134] uses the feature intra-cluster variance to measure the weights of each feature within its containing cluster.

Hruscka and Covoes [135] introduced Simplified Silhouette Sequential Forward Selection (SS-SFS) approach for feature selection. As the name of the algorithm implies the simplified silhouette measure is used as the clustering criterion to determine the quality of a feature subset. The algorithm starts with partitioning the data into various feature subsets, followed by applying *k*-means clustering to each feature subset. The simplified silhouette criterion is computed to each performed *k*-means and the subsets with best silhouette measures are selected. SS-SFS depends on forward selection of the features, which make it the key difference between SS-SFS and our proposed method in this work which operates in an iterative manner.

In [136] a method called Entropy Weighting *k*-means (EWKM) is introduced to reduce the intra-cluster distortion and increase negative entropy throughout the clustering. EWKM weighting criteria depends on the computation of weight entropy in *k*-means objective function. Additionally, this method allows subspace clustering.

In [131] a new method for unsupervised clustering criterion is introduced, where it solves two main challenges in *k*-means clustering methods; obtaining the optimal partitioning, and applying ranks for features to perform feature selection. The proposed method is applied to *k*-means clustering to choose the best partitioning according to the intra and inter cluster inertia scores. The inertia scores are created by building scatter matrices from each cluster's partition, and then based on the minimization-maximisation of the created matrices a ranking score for each cluster partition is established. Eventually, all the partitions and their ranks are added to a search space for the application of a proper searching algorithm necessary for optimal partitioning.

The recent work proposed in [137] the authors introduced a ranking pipeline that includes *k*-means and various statistical approaches such as, signal-to-noise ratio, t-statistics and significance analysis to rank the features in a highly dimensional microarray. This method is also considered a hybrid *k*-means approach that combines wrapper methods represented by *k*-means, as well as filter methods represented by the statistical analysis.

3.3 Sparse k-means Feature Selection

The research done in [138] explains the definition of sparse learning specialized in clustering algorithms for dimensionality reduction. One way to describe sparse learning in k-means is a form of matrix decomposition that yields the matrix A as a lower dimensional and more relevant partition of the original dataset X .Where X is a matrix of $n \times p$ size and it can be approximately decomposed to the matrices A and B, following the formula: $X \approx AB$, As A is a $n \times q$ size, and B is $q \times p$ matrix, known that $q \ll p$. Eventually, the clustering can be formed using the lower dimensional decomposition matrix A instead of the whole dataset X.

In the last decade, Witten and Tibshirani [138] proposed a revolutionary framework for feature selection by introducing the concept of sparse clustering. They implicitly combined k-means algorithm with $l_1 - norm$ of Lasso-type as the feature selection contribution. The mechanism incorporated in this work for feature selection was introduced before in [139] as a technique for choosing the optimal k or number of clusters during k-means application. This technique is called gap statistics and it was included in this method to compute l instead, which refers to the number of features selected.

Embedded Unsupervised Feature Selection (EUFS) [140] proposes a new idea of embedding the feature selection process within the clustering algorithm by the deployment of sparse learning. In this work, k-means is used to initialize two essential matrices for the EUFS algorithm: matrix U the cluster indicator where $\mathrm{U} \in R^{Nxk}$, and matrix V the feature weights where $\mathrm{V} \in R^{dxk}$. EUFS applies $l_{2,1} - norm$ as a loss function to minimize the inaccuracies during the reconstruction of the dataset X where $\mathrm{X} \in R^{NXd}$ and the feature selection over the latent feature matrix V. EUFS is validated over six different real-world datasets from various fields of applications. Note that N, d, and k represent the number of samples, the number of dimensions or features and the number of clusters, respectively. In [141] the research done is based on the novelty algorithm introduced in [140]. This method adopts a similar analogy to EUFS explained earlier. However, the recent work in [141] uses Frobenius-norm as the loss function. Moreover, this method represents an iterative approach of sparse learning where k-means is executed iteratively until the convergence criteria is met.

3.4 *k*-means Based on Correlation Measures

In [142], a new perspective for feature selection using *k*-means is introduced, where a correlation measure between clusters is the selection or elimination criterion. The correlation measure is used to improve the quality of the feature subsets to be clustered using *k*-means. This method provides an elimination possibility of both irrelevant features using *k*-means, and redundant features using the correlation measure applied to each cluster. This method is validated by solving a classification problem using Naïve Bays classifier, applied on microarray and text datasets. Additionally, the work in [143] successfully integrated correlation-based *k*-means clustering to improve the accuracy of the computer-aided diagnosis specified with cardiovascular diseases. The following table analysis proposes a visual overview of the mentioned related work in a chronological order within each sub-category. Showing the datasets used for each literature and their validation approach used.

Table 5 *k*-means Unsupervised Feature Selection Related-Work [124].

Clustering Approach	Literature	Database Used for Validation	Validation Method
k-means Hybrid Approaches	[128]	Synthetic datasets UCI machine learning repository.	Feature ranking impurity
	[129]	Real datasets synthetic datasets(Wisconsin Prognostic Breast Cancer (WPBC) data [144]).	F-score for accuracy.
	[130]	Synthetic dataset UCI machine learning repository. (congress, ionosphere, pima diabetes and wine) [145].	Class error
k-means Based on Features Weighting or Ranking	[133]	UCI machine learning repository (heart, adult and Australian datasets [146])	Precision/recall evaluations
	[134]	Synthetic dataset UCI machine learning repository. (heart diseases data and the Australian credit card data)	Rand index evaluation [147].

	[135]	Synthetic dataset (same data used in [31]) UCI machine learning repository.(Bio1, Bio2..Bio5, yeast galactose dataset [148],)	Class error
	[136]	Synthetic datasets UCI machine learning repository (text data)	Entropy F-Score
	[131]	Synthetic datasets (generated using the framework in [149])	Recall Precision F-Score
	[137]	Benchmark microarray datasets (DLBCL [150], prostate, lymphoma [151], breast cancer [152])	Accuracy Error Rate Precision Sensitivity Specificity
Sparse k-means Feature Selection	[138]	Human breast tumour dataset [153]. Single Nucleotide Polymorphism (SNP) data	----------------------
	[140]	Mass Spectrometry (MS) dataset. Two microarrays of prostate cancer genes. Two face image datasets. One object image dataset.	Accuracy Normalize Mutation Information (NMI)
	[141]	Object image dataset (COIL202). Spoken letter recognition dataset (Isolet12). Cancer dataset (LUNG2). Handwritten digit dataset (USPS2) Face image datasets (AT&T3 and UMIST4)	Accuracy
k-means Based on Correlation measures	[24]	12 text and microarray Datasets.	Classification accuracy
	[143]	Heart dataset of children born with intrauterine growth restriction (IUGR) UCI machine learning repository: "CORONARY" a cardiovascular problems dataset [146].	Correlation measures.

Chapter 4: Unsupervised Feature Selection Using Recursive *k*-Means Silhouette Elimination (R*k*SE): A Two-Scenario Case Study for Fault Classification of High-Dimensional Sensor Data

Parts of this chapter have been used in our published pre-print in [124]. Please note that all rights and copy rights have been reserved to the pre-print publisher.

1. Chapter Overview

Feature selection is a crucial step to overcome the curse of dimensionality problem in data mining. This chapter proposes Recursive *k*-means Silhouette Elimination (R*k*SE) as a new unsupervised feature selection algorithm to reduce dimensionality in univariate and multivariate time-series datasets. Where *k*-means clustering is applied recursively to select the cluster representative features according to a unique application of silhouette measure for each cluster as the feature selection or elimination criteria. The proposed method is evaluated on a Hydraulic test rig multi sensor reading in two different fashions; (1) reduce the dimensionality in a component fault multivariate classification problem using various classifiers of different functionalities. (2) Classification of univariate injected sensor faults in a sliding window scenario, where the proposed method is used as a window compression method, to reduce the window dimensionality by selecting the best time points in a sliding window.

In both experiments, the classifiers used are: LR, LDA, KNN, CART, NB, SVM and finally, RF. Moreover, the results are validated for each classifier separately using 10-fold cross validation technique. As well as, compared to the results when the classification is pulled directly with no feature selection applied, and to another well-known feature selection and extraction techniques, which are FI and PCA, respectively. The experimental results and observations in the two comprehensive experiments demonstrated in this work reveal the capabilities and accuracy of the proposed method.

2. Recursive *k*-means Silhouette Elimination (R*k*SE): Method Overview

Recursive *k*-means Silhouette Elimination (R*k*SE): is a dimensionality reduction technique for high dimensional data of various types such as, large time-series datasets, microarrays, text, images and so on. The idea behind R*k*SE method is similar to any ordinary cluster-based unsupervised feature selection method, where they treat features as objects or samples, and it is required to cluster them into groups based on a computed similarity measure, or with the aid of data mining by applying a suitable clustering method. R*k*SE keeps recursively applying *k*-means clustering to group the features with similar patterns in the same cluster, while applying silhouette criteria iteratively as the selection condition. Start the feature selection with collecting the features that are higher than some user-defined threshold or tolerance value. This threshold represents the strength of the connection between the cluster and the individual features located within, represented by the silhouette measure. The highest selected thresholds, the more connected the feature should be, to be selected, and the more iterations required to complete the feature selection process. Thereafter, the features with the highest silhouette criteria of each cluster are selected to represent the whole cluster. Within each cluster, neglect all the features higher than the threshold other than the selected highest silhouette feature. Since the feature with highest silhouette value in the cluster is the one connected the most to this cluster, and the rest of the features within the cluster are either highly connected to the cluster centre (the ones with high silhouette criterion) or weakly connected to the centre (the ones with lower silhouette criterion). The highly connected features are similar to each other, hence they are all strongly connected to the same cluster centroid, and by selecting only one of them, particularly the highest silhouette above the threshold, to represent the high pack is only fair and necessary to eliminate the redundant cluster similar features. However, the weakly connected features within the cluster (silhouette lower than selected threshold) are following slightly to highly different patterns than their connected clusters, and these connections can be affiliated to other cluster or other centroids within the same cluster. That is why, these features should be accumulated from all the clusters and stored in a matrix for remaining features, followed by aggregating them, re-cluster them all together and compute the silhouette over again. This process keeps repeating recursively between clustering (dividing), silhouette calculation, feature selection (highest silhouette above threshold of each cluster), elimination (silhouette above threshold of each cluster other than the highest) and aggregation (lower than threshold of each cluster) until all features are either selected or eliminated. In other words, the recursion is convergence when the amount features in the remaining features matrix is empty or null. Let us Assume that $\boldsymbol{X} \in \mathbb{R}^{\boldsymbol{d} \times \boldsymbol{N}}$ where $\boldsymbol{d}$ is the number of features or dimensions needed to be clustered, and $\boldsymbol{N}$ is the value of each feature $\boldsymbol{d}$ through the samples or selected subset of the samples. Set the threshold $\boldsymbol{\sigma}$ to any desired percentage where $\mathbf{0} < \boldsymbol{\sigma} < \mathbf{1}$. The higher the threshold, the more features to be selected, and the number of iterations or re-clustering before reaching convergence is increased. Moreover, the quality of the feature selection is directly proportional to the threshold $\boldsymbol{\sigma}$ selected. When $\boldsymbol{\sigma} \to \mathbf{1}$ the max number of features $< \boldsymbol{N}$ are selected, and the accuracy of the feature selection is maximized. However, the computational cost and time will rise dramatically in comparison to lower thresholds, due to the increase of the iteration count for the process repetition. It is crucial to identify some matrices required during the feature selection. $\mathbf{X}_{\mathbf{remain}} \in \mathbb{R}^{\boldsymbol{r} \times \boldsymbol{N}}$ where $\boldsymbol{r}$ is the remain features from past iterations that has not yet been eliminated or selected but require re-clustering to make the choice accordingly. $\mathbf{X}_{\mathbf{remain}}$ contains the features from all the cluster aggregated, which did not satisfy the condition $\boldsymbol{S}_{\boldsymbol{i}} \geq \boldsymbol{\sigma}$ in the previous iteration, as well as they showed weak connection to their current cluster, so re-clustering is inevitable to find another more connected pattern in the feature space. $\mathbf{X}_{\mathbf{selected}} \in \mathbb{R}^{\boldsymbol{s} \times \boldsymbol{N}}$ where $\boldsymbol{s}$ is the number of features selected. The selected features are only the features with the highest silhouette criteria $\boldsymbol{max}\ (\boldsymbol{S}_{\boldsymbol{i}})|_{\boldsymbol{S}_{\boldsymbol{i}} \epsilon \boldsymbol{C}_{\boldsymbol{k}}}$ that is also fulfilling the selection criteria $\boldsymbol{S}_{\boldsymbol{i}} \geq \boldsymbol{\sigma}$ within each cluster $\boldsymbol{C}_{\boldsymbol{k}}$ collected recursively throughout the iterations after the aggregation and re-clustering of each phase. Which make the final condition for choosing the feature is $(\mathbf{max}(\boldsymbol{S}_{\boldsymbol{i}}) \geq \boldsymbol{\sigma})|_{\boldsymbol{S}_{\boldsymbol{i}} \epsilon \boldsymbol{C}_{\boldsymbol{k}}}$. Finally, $\mathbf{X}_{\mathbf{eliminated}} \in \mathbb{R}^{\boldsymbol{e} \times \boldsymbol{N}}$ where $\boldsymbol{e}$ is the number of features eliminated that has the size of $\boldsymbol{e} < \boldsymbol{d}$. The features added to the $\mathbf{X}_{\mathbf{eliminated}}$ matrix are the redundant ones within each cluster collected iteratively throughout the iterations. More specifically, the eliminated features represent the ones that did belong to their representing cluster following that exact iteration. However, they have higher than threshold silhouette value $(\boldsymbol{S}_{\boldsymbol{i}} \geq \boldsymbol{\sigma})|_{\boldsymbol{S}_{\boldsymbol{i}} \epsilon \boldsymbol{C}_{\boldsymbol{k}}}$, but not high enough to represent the whole similar features in the cluster. Which means $((\boldsymbol{S}_{\boldsymbol{i}} \geq \boldsymbol{\sigma}) \cap (\boldsymbol{S}_{\boldsymbol{i}} < \mathbf{max}(\boldsymbol{S}_{\boldsymbol{i}})))|_{\boldsymbol{S}_{\boldsymbol{i}} \epsilon \boldsymbol{C}_{\boldsymbol{k}}}$. Eliminating those features even though they possess high intra-cluster relation can massively reduce the features redundancy. Furthermore, another reason to escape the algorithm is when it refuses to reach convergence for a pre-defined number of iterations, where $\mathbf{X_{remain}}$ keeps constant and fixed for many iterations and no more possible re-clustering that provides the sufficient requested threshold is possible. In this case, all the features in $\mathbf{X_{remain}}$ will be added to $\mathbf{X_{eliminated}}$, which ensures $\mathbf{X_{remain}}$ to have null content, that provokes the completion of the algorithm by reaching convergence. To develop more precise explanation of the feature selection proposed, the below pseudo code to R*k*SE is introduced.

R*k*SE Pseudo Code

```
1. Initialisation of important matrices and parameters:
$X \in \mathbb{R}^{d \times N}$
$\sigma = user\_defined \;\; 0 < \sigma < 1$
$\mathbf{X_{remain}} = X$
$\mathbf{X_{eliminated}} = \emptyset$
$\mathbf{X_{selected}} = \emptyset$
2. Apply k-means clustering using $\mathbf{X_{remain}}$ (Choose the optimal k using the
   elbow method prior to k-means application)
3. Calculate $S_i$ for each element within each cluster following the equation
   below:
$S_i = \frac{b_i - a_i}{\max(a_i, b_i)}$
4. Some features will be selected, eliminated, or remain for re-clustering
   based on the following value of $S_i$ within each cluster $C_k$ separately.
$S_i = \begin{cases} \mathbf{X_{remain}} \leftarrow i, & if\ S_i < \sigma \\ \mathbf{X_{eliminated}} \leftarrow i, & if\ (S_i \geq \sigma) and\ (S_i < \max S_i) \\ \mathbf{X_{selected}} \leftarrow i, & if\ (S_i \geq \sigma) and\ (S_i = \max S_i) \end{cases}$

5. Remove $\mathbf{X_{eliminated}}$ and $\mathbf{X_{selected}}$ from $\mathbf{X_{remain}}$
6. Check if $\mathbf{X_{remain}}$ is empty

if $\mathbf{X_{remain}} = \emptyset$ then
Convergence achieved; feature selection is complete.
 Selected features are stored in $\mathbf{X_{selected}}$
else
Repeat from step 2
```

To sum up, R*k*SE represents an iterative, unsupervised, silhouette-based, *k*-means clustering feature selection algorithm. Although, R*k*SE has plenty of advantaged and contributions that exceed the methods mentioned in related-work section, it also has limitations that we hope to eliminate in the future.

Based on the iterative and unsupervised nature of R*k*SE, it can include the advantages of both methodology groups and their disadvantages. The following table shows the pros and cons of R*k*SE.

Table 6 R*k*SE Pros and Cons

Method	Pros	Cons
R*k*SE	- Can create a model representation of feature dependencies. (iterative advantage) - Feature selection and clustering are made concurrently in one single operation. (iterative advantage) - Unsupervised feature selection method, no labels required. - Simple, robust, and low computational and time cost: due to the exclusive application of *k*-means and silhouette criteria, which provides simplicity and reduce time and computational complexity - User-interactive: allows the user to choose the value of the threshold which give the freedom of choice, which can change the algorithm drastically. - Introducing a new concept of using *k*-means and silhouette measure in a recursive manner, instead of the common forward and backward approaches.	- Prone to overfitting. (iterative algorithms disadvantage) - The choice of the threshold can drastically affect the quality of the selection, which cannot be guaranteed since σ is user selected. - The accuracy is a little compromised because the algorithm focuses on the relationship between the feature and the cluster, rather than the relationships between features.

3. Analysis and Experimental Results

In this section, R*k*SE method is analysed, evaluated, and tested when applied on two main experiments.

The first experiment aims to study the effect of R*k*SE in feature selection for univariate time-series data in a shape of windows with a defined length. R*k*SE when applied to experiment one, is supposed to act as a time-series compression method that chooses the most informative time points in each

sliding window and eliminates the redundant and less important time points per window. (selection of optimal time points in one modality type)
The second experiment explores the potential of R*k*SE applied to a multivariate time-series dataset without the sliding window application. In this experiment, RKSE is expected to choose the most informative features within all the time points. i.e., in a dataset of different sensors readings, R*k*SE is expected to choose the best sensors as the representative features. (selection of best modalities among variations of them during various time frames).
For the sake of validating the results of the two mentioned experiments, it is essential to explain the main methodologies followed to validate unsupervised feature selection methods. The following points describe the main categories for unsupervised feature selection validation techniques as researched in [36].

1- **Feature selection evaluation by classification accuracy:** in this method the selected features using the feature selection method subject of evaluation are used to test a classification problem using one of the common supervised classifiers such as, SVM, KNN or NB. Then, the classification accuracy or the error rate is measured, and compared to the classification results of the entire dataset prior to classification.
2- **Feature selection evaluation using clustering criteria:** In this case the results of a clustering task such as, *k*-means clustering is evaluated by using one of the clustering qualities measures. i.e., Normalized Mutual Information and Clustering Accuracy.

In this work, the evaluation method used is the classification accuracy approach since the dataset available for evaluation has already included the labels. Moreover, the fault classification experiments are done in more detail in chapter 6, against PCA as a feature extraction method, and FI and manually selected time-domain features as the supervised feature selection method. Therefore, it will provide a wide range of comparisons between R*k*SE and other approaches besides the original dataset prior to feature selection. The dataset used for the following experiments is the hydraulic test rig dataset. This dataset is described in full detail Chapter 2. In addition, it was pre-processed differently to fit two scenarios of classification schemas; one to fulfil sensor FDD using a fault injection scenario, while the other is processed for component FDD using the real-time measured faults in the test rig.

Before we start demonstrating the two experiments, it is necessary to explain the procedure in deciding the optimal number of features to select for each method such as PCA and FI. The optimal number of features selected for FI are the ones guaranteeing the best accuracy when performing the classification task. In chapter 7, FI is used as the feature selection method prior to the hybrid RF approach, and the experimental results of that chapter shows the fundamental steps of experimentally choosing the number of features when using FI. On the other hand, when extracting features using PCA, it is essential to calculate as many PCs as possible, then compute the variance of each PC separately. The variance usually is an indication of the availability of variant data points, which indicates the existence of information. Data that is rich in variance (no redundancy) is a healthy data that carries a significant amount of information for classification. When plotting the variance value of each PC, the optimal number of PCs is the one carrying the most variance, before the variance gets almost constant when moving to further PCs. Figure 12 shows process of selecting the optimal PC based on the variance changes. As shown the best number of PCs is five, before the variance gets almost constant in the next PCs.

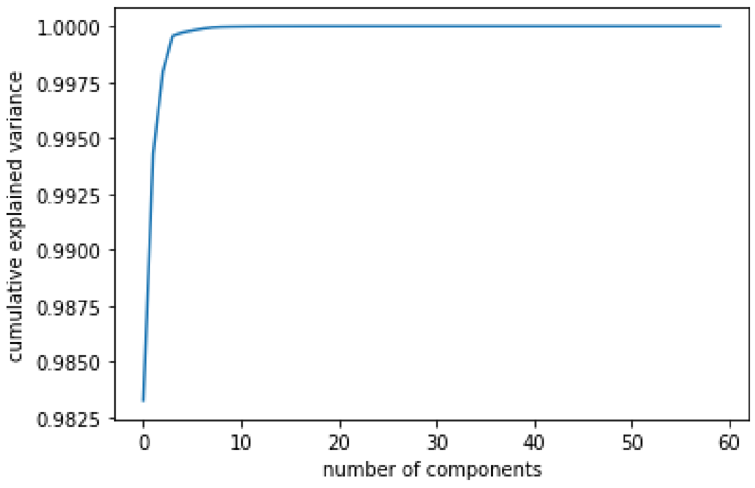
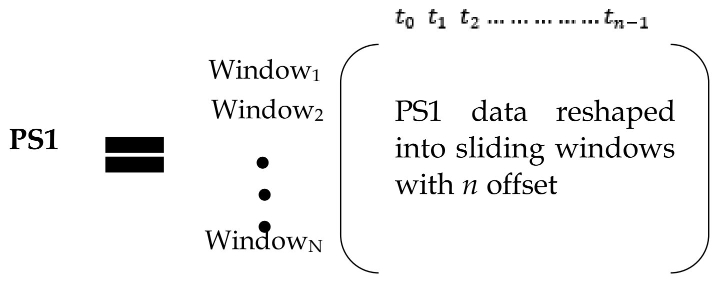


Figure 12 Choosing the Optimal PCs or Features in PCA.

3.1 Experiment One: R*k*SE for Univariate Time-Series Feature Selection within a Window

In this experiment, the sensor PS1 reading from the hydraulic test rig dataset is used for the purpose of sensor FDD using a classification schema. Four main types of faults are injected in the PS1 data such as, constant fault (constant high, low and zero), gain fault and bias or offset fault, which makes the resultant PS1 data containing four different labels of faults along with the healthy readings. The pre-processed PS1 data that has 28,882 readings that are captured in a second basis, is reshaped into a 7210 sliding windows with 60 seconds length worth of PS1 readings with zero intersection points between each window or offset/delay equals the window size *n*, as shown below:


$t_0\ t_1\ t_2 \dots\dots\dots\dots\dots t_{n-1}$
PS1
Window₁
Window₂
Window_N
PS1 data reshaped into sliding windows with *n* offset

First of all, as described earlier R*k*SE has a user-specified threshold that effects the number of features selected and the accuracy of selection process. Thus, the following table shows the number of features selected, the execution time for R*k*SE and the number of re-clustering iterations required until reached convergence when changing the threshold between 0.1 until approaching the highest threshold of one.

Table 7 The Effect of Threshold on Number of Features Selected, Number of Iterations Required and Execution Time in Milliseconds [124].

σ	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.95	0.98
No. Features Selected	3	5	8	8	8	9	10	11	20	46	60
Exe.Time (msec)	76.2	76.99	77.58	77.22	77.18	77.07	77.00	77.45	78.03	80.69	87.14
No. Iterations	2	2	3	3	3	4	4	7	8	8	10

As shown in the table, the increase of the threshold selected increases the number of features selected, as well as increasing the time and computational complexity of the algorithm by increasing the number of iterations required until reaching the convergence criteria of empty $X_{remain.}$ The funnel chart below emphasizes the relationship inferred above. Notice that when the threshold is 0.98≈ 1 all features in X were selected as if no feature selection is applied in the first place.

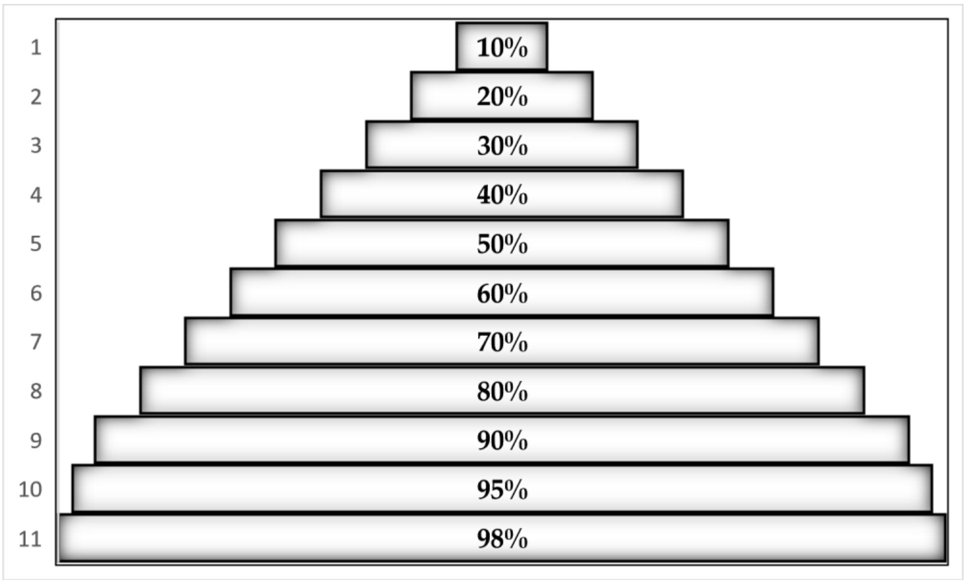


Figure 13 Funnel Graph Describing the Directly Proportional Relationship Between the Threshold and the Features Selected [124].

To evaluate the performance of the feature selection algorithm applied on the PS1 data, the features selected by R*k*SE are used to classify the data into healthy and faulty. The performance of R*k*SE is compared to the original dataset without feature selection (number of features is 60), and then compared to the features extracted with PCA, and selected with FI.
The following graph shows the mean accuracy of 10-folds when applying 10-fold cross validation technique evaluating the performance of R*k*SE over various classifiers; LR, LDA, KNN, CART, NB, SVM and RF. Illustrated in Figure 14, the results of the mean 10-fold accuracy for the classifiers applying different number of selected features to show a trade-off between the accuracy and the number of features selected. The fault classification schema is similar to the one introduced fully in chapter 6, but using data with various health and fault statuses instead of just applying faulty data as in chapter 6 experiment.

As shown, the increase in the number of features selected followed by an increase of classification accuracy for all the features. However, when the threshold is 95% the feature selected are 46 features out of 60 overalls, the 46 features offered higher classification accuracy than the original dataset. Which implies that R*k*SE has successfully reduced the dimensionality of a univariate time-series dataset in a sliding window scenario with even an increase of the mean accuracy for most of the classifiers applied. Figure 15 shows the relationship between the classifier accuracy and the threshold applied.

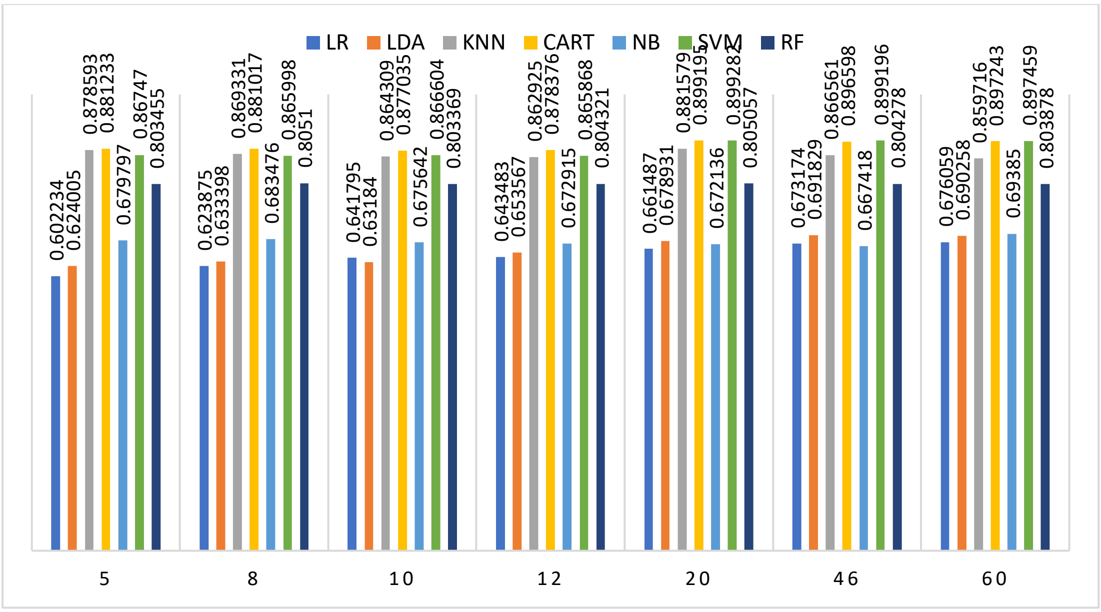


Figure 14 Feature Number and Mean Accuracy Comparisons Applied for Various Classifiers [124].

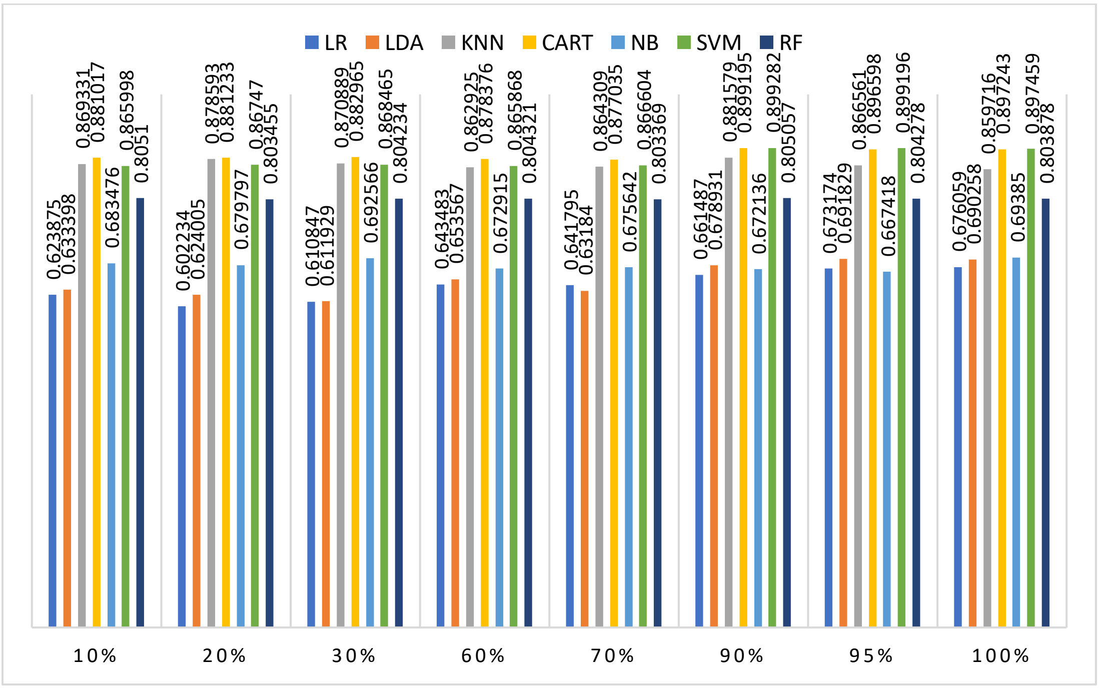
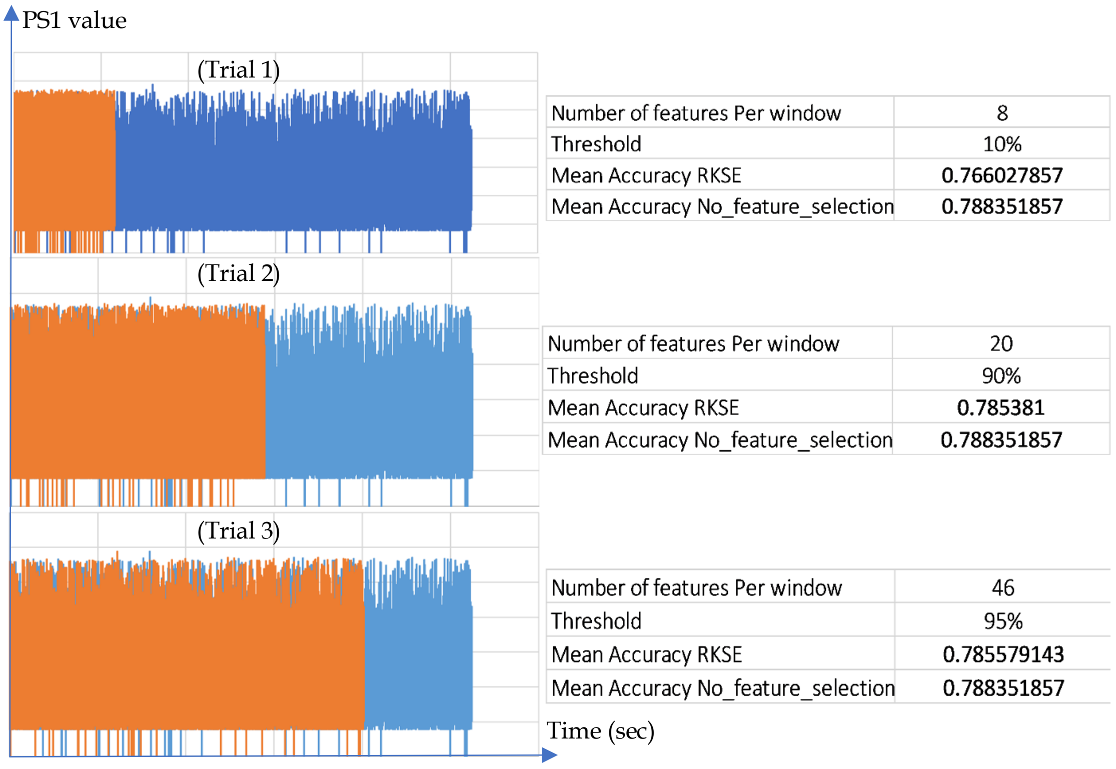


Figure 15 Threshold and Mean Accuracy Comparisons Applied for Various Classifiers [124].

Figure 16 demonstrates the ability of R*k*SE in reducing the size of the training time-series data, by selecting small number of features per window instead of taking all the window size for classification. The orange signal is the PS1 after feature selection, while the blue signal is the original PS1 signal without feature selection laying underneath the orange signal. As a conclusion, R*k*SE noticeably reduced the size of the training dataset without compromising the accuracy of the classification even with smallest threshold applied. Furthermore, some other feature selection and extraction methods are applied to the classification experiment shown. PCA is applied and provided only five features extracted to from each sliding window to represent the whole window for classification. Not surprisingly, PCA succeeded to maintain the average accuracy while applying the minimal number of features. While the feature selection methods tested FI and R*k*SE has smaller average accuracy when five features are selected; 0.706 and 0.762 respectively, PCA has the mean average of all classifiers applied of 0.784 which is slightly higher than both FI and R*k*SE. Moreover, both FI and R*k*SE reached their full accuracy potential when the number of features selected are 0.787 and 0.785 respectively, which shows little improvement of the accuracy when using only five features with PCA with the result of 0.784.



Number of features Per window	8
Threshold	10%
Mean Accuracy RKSE	**0.766027857**
Mean Accuracy No_feature_selection	**0.788351857**

Number of features Per window	20
Threshold	90%
Mean Accuracy RKSE	**0.785381**
Mean Accuracy No_feature_selection	**0.788351857**

Number of features Per window	46
Threshold	95%
Mean Accuracy RKSE	**0.785579143**
Mean Accuracy No_feature_selection	**0.788351857**

Figure 16 The Effect of R*k*SE in Minimizing the Size of the Original Signal While Keeping the Accuracy [124].

The following figure, Figure 17 shows a detailed comparison for all the classifiers used when each method; FI, R*k*SE and PCA reach their highest accuracy with 45 features for FI and R*k*SE comparing to only five features using PCA.

In conclusion, R*k*SE has proven its potential to accurately select features in univariate time-series datasets in a window fashion, where the features are the time points in the window. R*k*SE has shown better results comparing to without the application of any feature selection at all. Moreover, when R*k*SE is compared to another supervised feature selection method (FI), R*k*SE has also shown slight improvements using the same number of selected features. However, feature extraction methods such as PCA, offered comparable accuracy with little features included. This observation could lead to the conclusion that feature extraction methods such as PCA, are more effective when applied to univariate time-series dimensionality reduction problems within a sliding window.

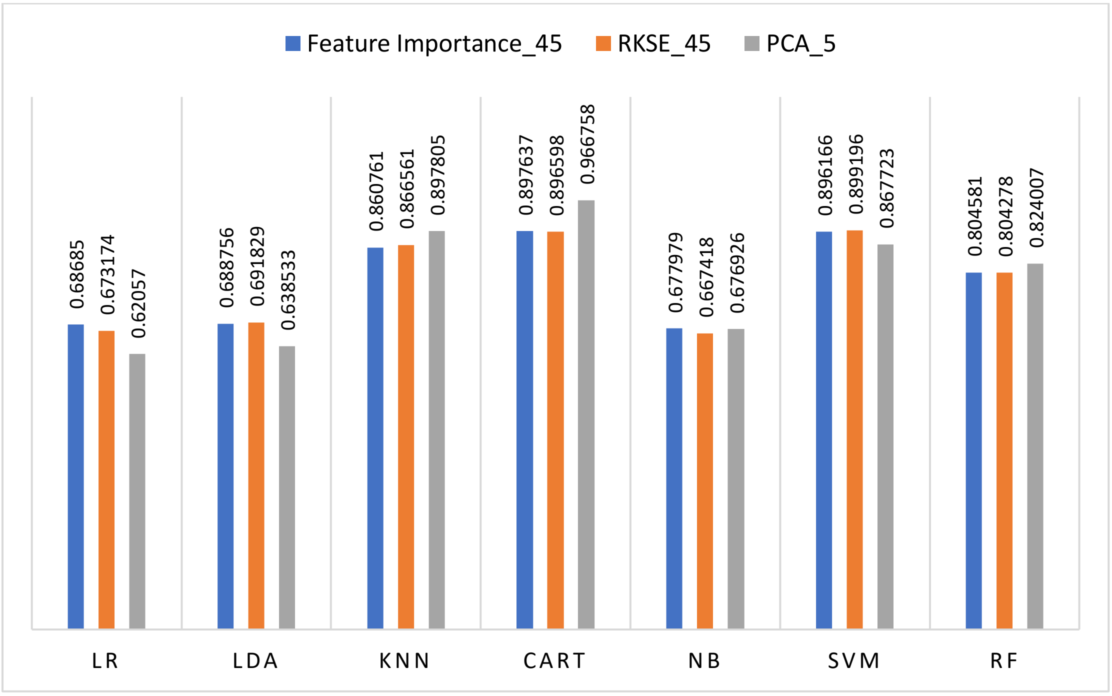


Figure 17 FI, R*k*SE and PCA Performance Evaluation with Best Number of Features [124].

3.2 Experiment Two: R*k*SE for Multivariate Time-Series Feature Selection without a Window

In this experiment, the hydraulic test rig dataset is used for component fault classification based on the classification of eleven main sensors: PS1-PS6, TS1-

TS4 and VS1. The data is processed in a way that include the fully efficient samples as the healthy form, while the full failure of the cooler, valve, pump, and heater are used to represent the rest of the faulty samples.

The overall goal of this experiment is to investigate the potential of R*k*SE for selecting the most important sensors for the component fault classification challenge. When applying R*k*SE to this experiment, it is crucial to make sure that the features (sensors) are located on the row of the dataset as they are the subject to be clustered iteratively. When applying R*k*SE to the multivariate hydraulic test rig dataset, it showed a good performance comparing to when applying the entire eleven-dimensional dataset for classification. When the threshold is set to 0.20 the number of features selected are four, with 0.90 threshold five features are selected, 0.95 with six features, and finally with 0.98 threshold nine features out of eleven are selected. Figure 18, shows the results of all the classifiers applying different threshold when using R*k*SE. Hence, different number of features are compared.

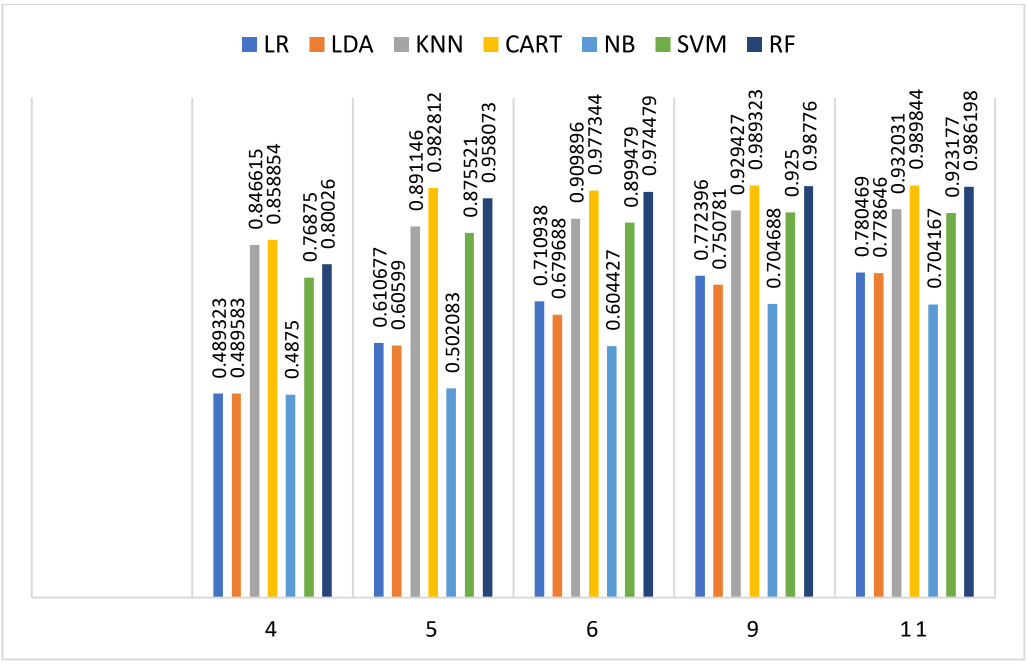


Figure 18 R*k*SE of Various Threshold Values Applied to Different Classifiers [124].

Figure 19, shows the average accuracy of all the classifiers mean accuracy at a certain number of features used for classification. It is obviously noticeable that R*k*SE of thresholds 0.90 and 0.95 with six and nine number of selected features respectively, has shown comparable results to the fully sized dataset of eleven features with lower dimensionality applied.

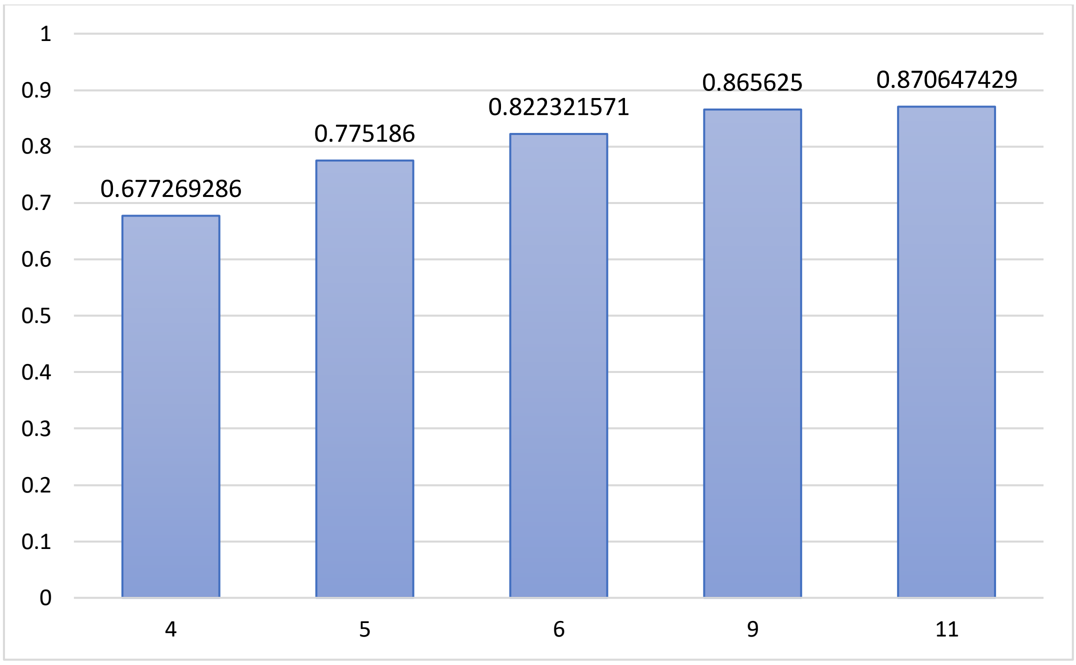


Figure 19 Average Accuracy of All Classifiers for Different Feature Numbers [124].

Similarly, when comparing the classification results between PCA and FI as done previously in experiment one, PCA achieves its highest average accuracy when the features extracted are the first five PCs with 0.8478 average accuracy. Comparing to the FI that showed the highest recorded performance using only four most important features, with the average accuracy of 0.9638. Finally, when applying R*k*SE the highest accuracy is accomplished when nine features are selected, that makes its average accuracy among all classifiers is 0.8656. Figure 20 provides a visual explanation of the previous observation.

To sum up, PCA has shown a steady performance when applying various classifiers of different functionality and mechanism for validation. Additionally, the steady performance of PCA was not affected by the type or shape of the input dataset, whether the classification problem is univariate or multivariate, sliding windows or not and, sensor or component faults. PCA remained consistence in its high performance which proves the theory of the suitability of feature extraction methods over feature selection when applying to time-series computations.

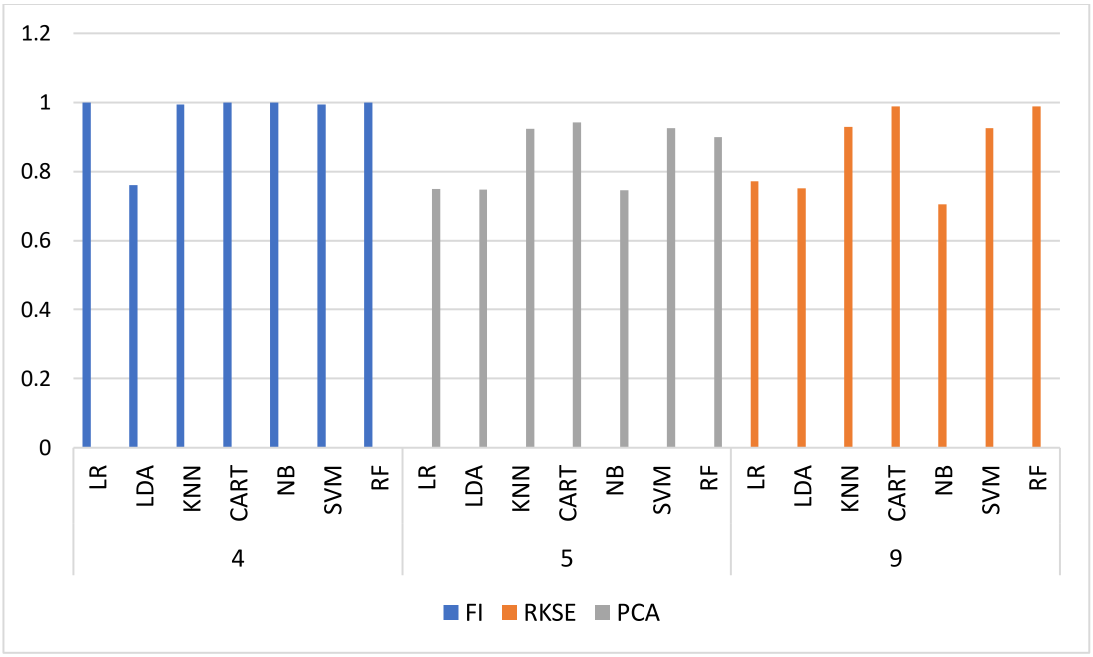


Figure 20 FI, R*k*SE and PCA Comparisons.

Back to the feature selection methods evaluated in this chapter. R*k*SE and FI are two wrapper feature selection algorithms of two different nature. R*k*SE is unsupervised based on *k*-means and the silhouette value, while FI is a supervised method that requires the availability of class labels to apply RF as the criteria used to compute the performance of each feature. R*k*SE showed slightly better overall performance when applied to the univariate sliding window data structure, comparing to the FI results when the same number of features are selected. Although R*k*SE increased the average accuracy over FI in the first experiment, FI flipped the turns and showed better accuracy than R*k*SE with only four features selected versus nine in R*k*SE. The reason behind this remark is that FI uses the knowledge of the samples' labels, which allows more heuristic approach that keeps improving as an optimization of RF.

Chapter 5: Sensor and Component FDD for Hydraulic Systems using Combined LSTM Autoencoder Detector and Diagnosis Classifiers

Please note that parts of this chapter have been published by us in [78], while all copy rights and rights are reserved to the MDPI publisher.

1. Chapter Overview

In this chapter, an FDD method is constructed based on integrating fault detection using LSTM autoencoders, and fault diagnosis applying various supervised ML and DL approaches. The detection and diagnosis processes are done separately to ensure detecting rare fault occurrences in time-series data applied on hydraulic systems.

In the fault detection phase, the LSTM autoencoder is trained using the fully efficient inputs of the dataset, which represents the healthy form of the training data. Eventually, the autoencoding model is trained to reconstruct the healthy version of the input data at any given point of time. Comparing the reconstructed healthy signal with the given one, provides an indication of any fault presence. The more the given signal is identical to the healthy reconstructed one, the most likely it is a healthy signal and lacks the presence of anomalies, and vice versa.

The faults or deviations captured in the first phase (detection phase), are then used to train the classification model in the fault diagnosis stage. This phase consists of two major processes: feature engineering analysis, and ML and DL model investigation. The engineered features are fed into various ML and DL approaches to determine the best approach to diagnose component and sensor faults. In this phase, all the training data are faulty inputs in contrary of the previous phase.

Otherwise speaking, in this chapter a fault detection and classification applying healthy signal reconstruction using LSTM Multi-step Forecasting, combined to a fault classification schema is introduced, where a comprehensive application of various classification ML and DL approaches is shown and compared.

This chapter is inspired by the recent research in [108], and represents a more extensive and improved version of the mentioned research. The former FDD method developed in [108] shows an integration between two separate phases; the detection phase using LSTM autoencoders, and the diagnosis phase represented by the LSTM classifier. They applied the proposed FDD system on the dataset generated by Tennessee Eastman benchmark [109].

In this work, we adopted the same idea of separating the detection and diagnosis phases to ensure better performance in capturing rare occurrences or events. However, the following improvements and distinctions between the former research and our current work is explained below:

- In the former work, they only investigated one type of faults, which is component faults. However, our research has conducted two separate experiments applying the same analogy, but one to detect and diagnose sensor faults while the other do the same but to component faults.

- Both component FDD and sensor FDD experiments conducted are applied on a hydraulic test rig dataset. The component faults are already labelled and provided by the dataset. Meanwhile, the sensor faults on were injected in the data following a data fault injection schema.

- In the former work, the detection phase represented by the LSTM healthy signal reconstruction, the similarity measure chosen to check the relevance between the constructed signal and the input signal is a simple signal difference. However, in this work we provided a more accurate measurement to find the similarity between the signals, applying Pearson’s autocorrelation to calculate the similarity, followed by calculating the signal difference by simply taking the subtraction between the full correlation of positive one and the calculated autocorrelation.

- The former approach directly applied LSTM as a classifier to follow up with the detection phase represented by the autoencoder. They compared the result with CNN classifier. However, they have not conducted any thorough comparisons with any other traditional ML approaches. In this work, we conducted the detection phase to investigate various ML and DL approaches to decide the most accurate algorithm for the joint FDD approach.

- The former work did not focus on feature engineering prior the classification phase. In this work, we applied different feature engineering approaches such as, PCA, FI, manually extracted time-domain features and a R*k*SE. This step provides essential comparisons between four common feature engineering methods of different nature, which shines a light on a guide that clearly shows the effect of each features on the target classifiers. Thus, researchers using fault classification as a fault diagnosis mechanism could rely on the comparisons we provided as a base for further research on FDD for hydraulic rigs, to know which features work the best/worst for each ML and DL classifiers.

2. Hydraulic System FDD Overview

In this experiment the data used is collected from a condition monitoring of a hydraulic test rig, which is designated to test a hydraulic system. Thereafter, the data is used to conduct two main experiments, one to analyse the provided component faults at the total failure stage. The other experiment is concerned with sensor faults, where the fault injection takes place to successfully inject three main types of sensor faults; constant fault that covers constant low, constant high and constant zero faults, as well as gain faults and bias or offset faults. These injected faults along with the healthy readings would eventually act as pre-defined classes necessary for the fault classification and the healthy signal reconstruction learning, respectively.

Figure 21, shows the two main experiments to detect and diagnose a variety of sensor faults and severe component failures in the hydraulic test rig readings, as an example of industry 4.0 hydraulic system's data. The method followed to perform the FDD consists of two main separately conducted processes; the first is the fault detection using the LSTM healthy reconstruction schema. The second process is involved to diagnose the faults being detected during the detection phase, the diagnosis process is a classification algorithm using a variety of features and classifiers.

In this chapter, two comprehensive experiments were conducted to guarantee performing fault detection and diagnosis for each component and sensor faults in the hydraulic system tested using the hydraulic test rig.

The two experiments start mutually with applying the necessary data pre-processing steps, to ensure removing the unnecessary noise in the input signals, and to arrange the inputs in a way suitable for the LSTM autoencoder. Please note that the data applied, and its pre-processing differs between the sensor FDD experiment and the component one. Moreover, the data used for both experiments are filtered and organized differently. The data description and organisation for both experiments will be described in detail in the experimental results section of this chapter.

For Sensor FDD, the available dataset of a hydraulic test rig described in the data description chapter, lacks the presence of any sensor faults. Which necessities the need of injecting various sensor faults into the filtered and pre-processed data. The choice of which types of sensors faults to inject is decided upon convenience and necessity. For example, stuck-at or constant sensor fault was chosen to be injected in the data due to its simplicity to apply such an effect on different periods of time comparing to other data-centric sensor faults. i.e., outliers or spikes that are not easily predicted or frequently occurring, or even possessing a regular pattern in which they could be injected in the data. Gain and bias faults are an example of system-centric faults, which -by definition- are complicated to diagnose relying on the data alone, that is why these sensor faults are significant to study and apply algorithms with high accuracies to diagnose, as well as both mentioned faults have a clear definition and pattern that makes it easier to inject them to the dataset.

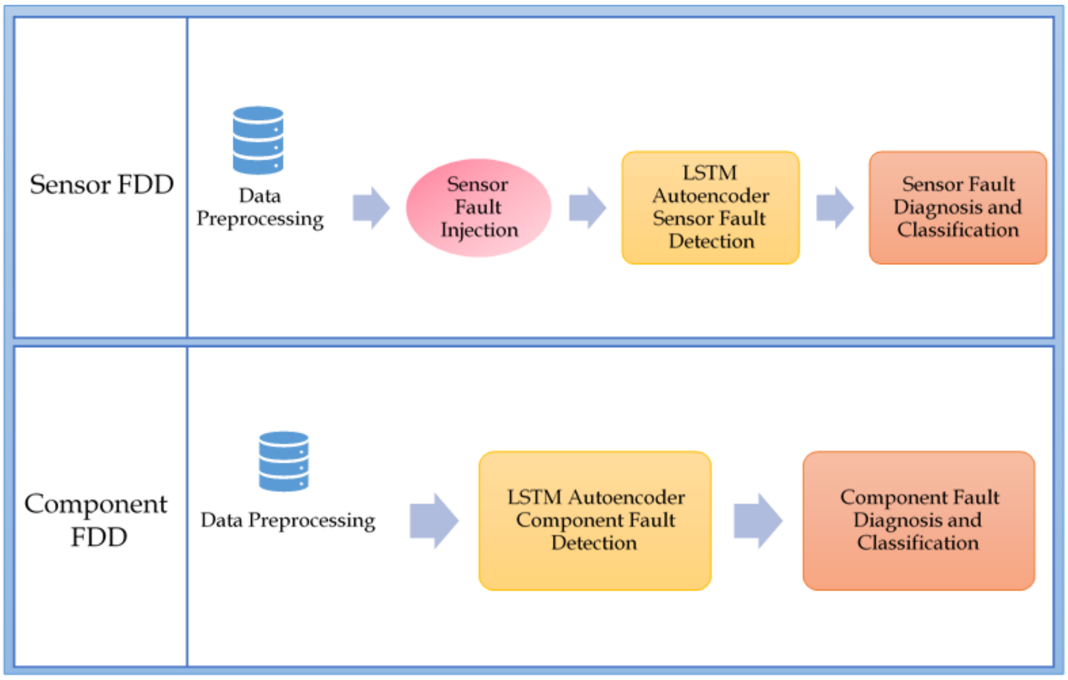

Sensor FDD
Data Preprocessing
Sensor Fault Injection
LSTM Autoencoder Sensor Fault Detection
Sensor Fault Diagnosis and Classification
Component FDD
Data Preprocessing
LSTM Autoencoder Component Fault Detection
Component Fault Diagnosis and Classification

Figure 21 An Overview of the Two Experiments to Achieve FDD in Hydraulic Test Rigs for both Sensor and Component Faults [78].

For the component FDD experiments, the data selected are the ones with full efficiency to be fed in the detection phase demonstrated by the LSTM autoencoder. However, the faults that are proceeded to the diagnosis stage are the ones representing total failure in the hydraulic test rig which are, cooler total failure, valve total failure, pump severe leakage and hydraulic accumulator total failure.

In both experiments the detection phase is demonstrated by the LSTM autoencoder to reconstruct the healthy form of the input signal when the FDD is conducted and tested. The LSTM autoencoder is trained using healthy sensor data that shows full efficiency in both experiments. However, the data formulation and organisation is different between the two experiments, since the signal subject of reconstruction for sensor FDD is a window of 60 seconds values corresponding to each sensor separately in the hydraulic test rig, but in the component FDD the healthy signal used for training is organised without sliding windows, while each reading represents the values of all eleven sensors at this particular point of time, and how they altogether contributed in diagnosing the failure.

In the diagnosis phase for both experiments, the faulty data of both systems are fed separately into a classification process. The choice of traditional ML classifiers for this experiment is dependent on the selection of various supervised learning methods of entirely different functionality and mechanism as possible, to provide a broad and comprehensive analysis. The classifiers used in this experiment are LDA, LR, KNN, CART, NB, SVM and finally RF. The DL methods chosen for this experiment are CNN and LSTM both applied in an interesting manner. The comparison includes the application of the chosen ML and DL approaches using different features extracted or selected via numerous feature extraction and selection methods such as, manually extracting time domain features for each sliding window, applying Principal PCA directly using the raw multivariate time domain sensor data without dimensionality reduction, as well as using our developed R*k*SE feature selection and dimensionality reduction algorithm. R*k*SE results comparisons will be explained in comprehensive detail in the following chapter. Moreover, the results of each ML and DL method using three different feature extraction schemas are shown.

Finally, the trained models and saved thresholds from each experiment can be easily used to achieve run-time predictions of new samples at real-time. The FDD prediction at run-time can be done by the following: (1) the detection: a) predict the healthy reconstructed sample to the new sample using the trained model of LSTM autoencoder. b) Compare the reconstructed sample and its original form by applying the suitable sequence difference. c) Compare the calculated sequence difference to the threshold computed during the offline training stage, if the difference is greater than the threshold then a fault has been detected.

When fault is detected, it needs to be passed to the next stage of fault diagnosis. (2) Fault diagnosis: this step is done by passing the new sample that has been detected as faulty, into the chosen trained classifier. After taking into consideration choosing the best features and the most optimal classifiers based on the comprehensive training and comparisons done previously in the model training offline phase.

In this section, only an overview of FDD process proposed is explained without comprehension. However, in the next section each experiment will be explained in exhaustive details.

3. Analysis and Experimental Results

3.1 Experiment One: Sensor FDD Using the Joint LSTM Autoencoder and Classifier Approach

In this section FDD of sensor faults in hydraulic test rigs using a joint approach between healthy signal reconstruction to detect sensor faults, followed by fault classification to diagnose the selected sensor faults is introduced, analyzed, and discussed.

The following subsections elucidate each step of the described approach applied on sensor faults and showcase their results. Figure 22 shows the steps included in experiment one, where below each step is elaborated in comprehensive detail.

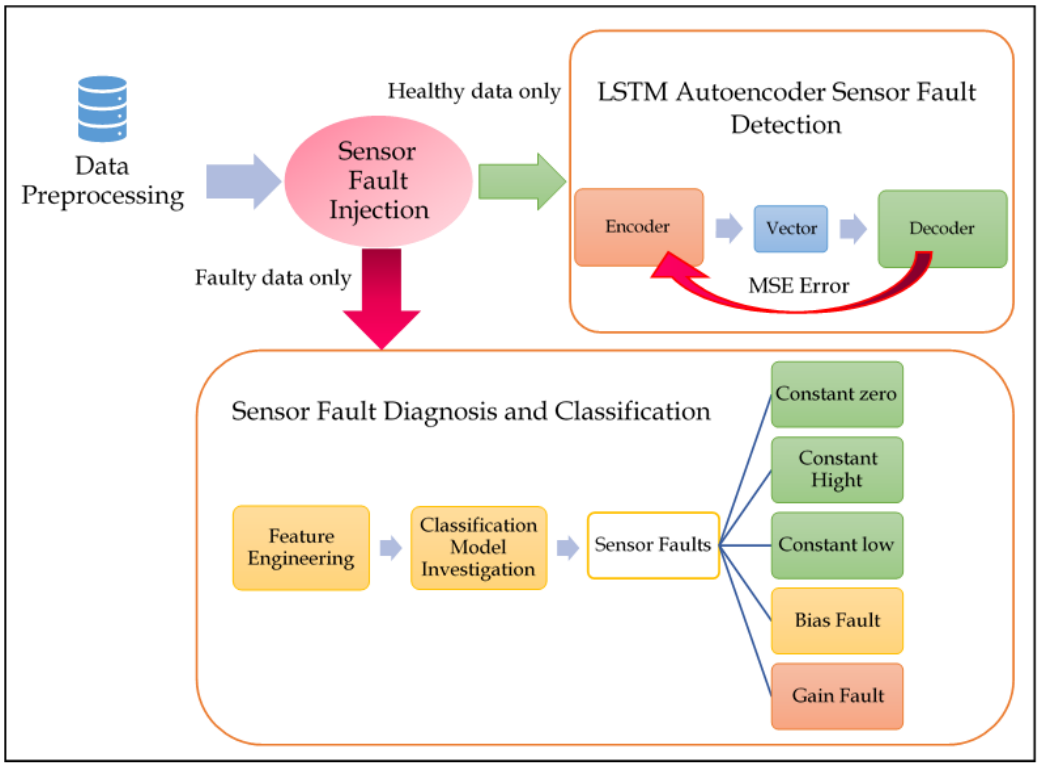


Figure 22 Sensor FDD Comprehensive Framework [78].

A. Data Pre-Processing and Organization

It is worth to be mentioned that the preprocessing step is applied twice for each phase in a different manner. The pre-processing and data structure required for the LSTM autoencoder in the detection phase differ from the data structure used for the diagnosis via classification phase. For this purpose, the preprocessing necessary for each phase is explained under the corresponding operation.

B. Sensor Fault Injection Schema

The dataset used for the sensor fault detection and diagnosis is the hydraulic test rig dataset explained in the data generation and collection chapter. The data provided a wide range of component faults varies from slightly damaged to total failure. However, the dataset did not provide any sensor faults. Thus, it is essential to inject sensor faults to build the sensor FDD model.

Although the sensor FDD architecture navigated in this chapter is meant for multivariate time-series data, for simplicity reasons one sensor only is considered to show results for the sensor FDD process.

Sensor PS1 (the first pressure sensor) is used to showcase the sensor FDD results during the fault injection, sensor FDD while using LSTM autoencoder and the sensor detection classification results.

The faults chosen to be injected are: (1) stuck-at: three main types of stuck at faults has been injected due to the fact that stuck-at or constant faults are the most common form of data-centric faults, and it shows a mighty severity of the sensor condition. Moreover, constant faults are extremely easy to inject. Consider the input sensor signal is $x(t)$ then the constant fault can be injected easily by following $x'(t) = c$ where c is a constant number representing the stationary condition of the sensor. Three main types of constant faults are added. constant zero when the sensor is stuck at zero, constant high when the sensor is stuck at the highest value in the window, and constant low stuck at the lowest point of the sensor readings during the observed window. We randomly injected 40 windows of size 60 seconds with constant zero fault, 7210 windows of size 60 readings of PS1 are injected with high and low constant faults, which make the overall number of windows injected with stuck-at fault is 7250 windows. (2) Gain fault and (3) bias or offset faults. these faults are a type of system-centric faults; hence it is hard to observe their pattern through sensor signal's observation alone. So that, these faults are significant to study and build ML approaches to dynamically detect and diagnose them.

Furthermore, both faults have a clear pattern that makes it easy to inject these types of faults in the data. Gain fault also known as amplification, where the original signal $x(t)$ is amplified with a constant w; $x'(t) = x(t) * w$. To inject this fault, randomly selected amplification number between 0.3-1.3 are selected each time, to regenerate the magnified fault signal. 7210 samples of 60 PS1 sensor readings are injected with randomly chosen gain values. Bias or offset fault is another example of calibration system-centric fault, where the original signal is shifted with a constant value. Consider the original signal is $x(t)$ then the manipulated with offset signal is $x'(t) = x(t) + b$ where b is the constant number representing the bias or offset added to the signal. b value can be too small and hard to notice or observe, or too large and hard to ignore. As a result, it is essential to inject both cases of b. To achieve this, 3480 windows of size 60 were injected with a random number between 0.1-1 to represent the too tiny bias category, while the remaining 3730 windows of size 60, were injected with the comparatively larger biases that are randomly chosen between 1.1-50. Finally, the overall PS1 sensor data prepared after the fault inject process, possesses many windows of size 60 readings that consists of the following: (1) 7210 windows representing fully efficient windows as an example of healthy windows. (2) 7250 windows of constant faults (zero, high, low). (3) 7120 windows of gain fault. (4) 7210 windows of bias faults (low bias, high bias). Figure 23, Figure 24 and Figure 25 show a small part of a signal with each type of faults injected.

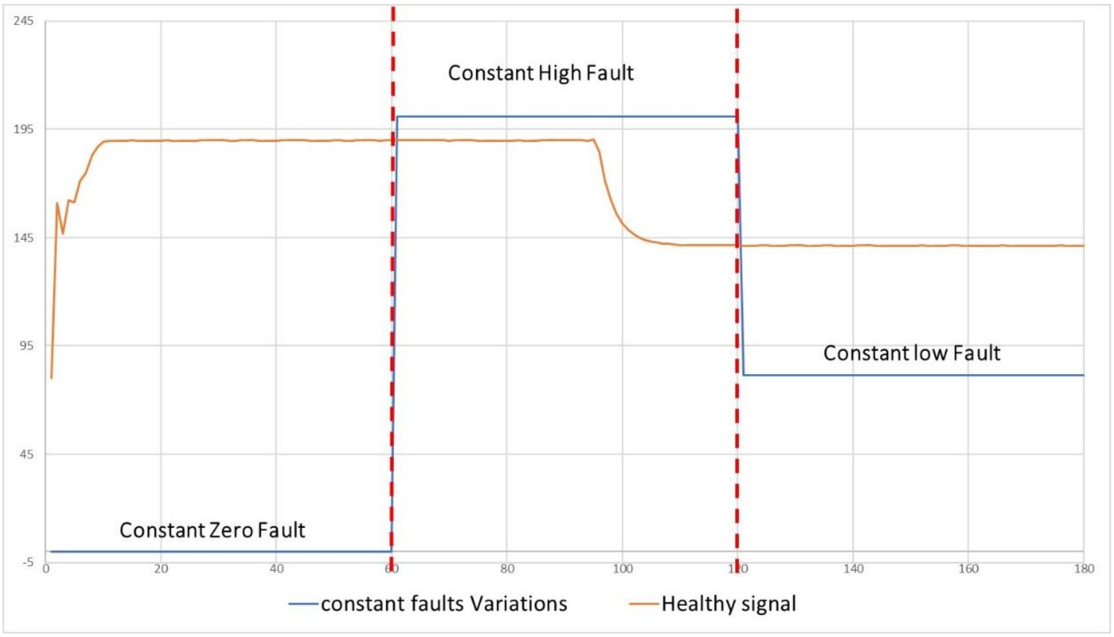


Figure 23 Constant Faults Verses Healthy Signal.

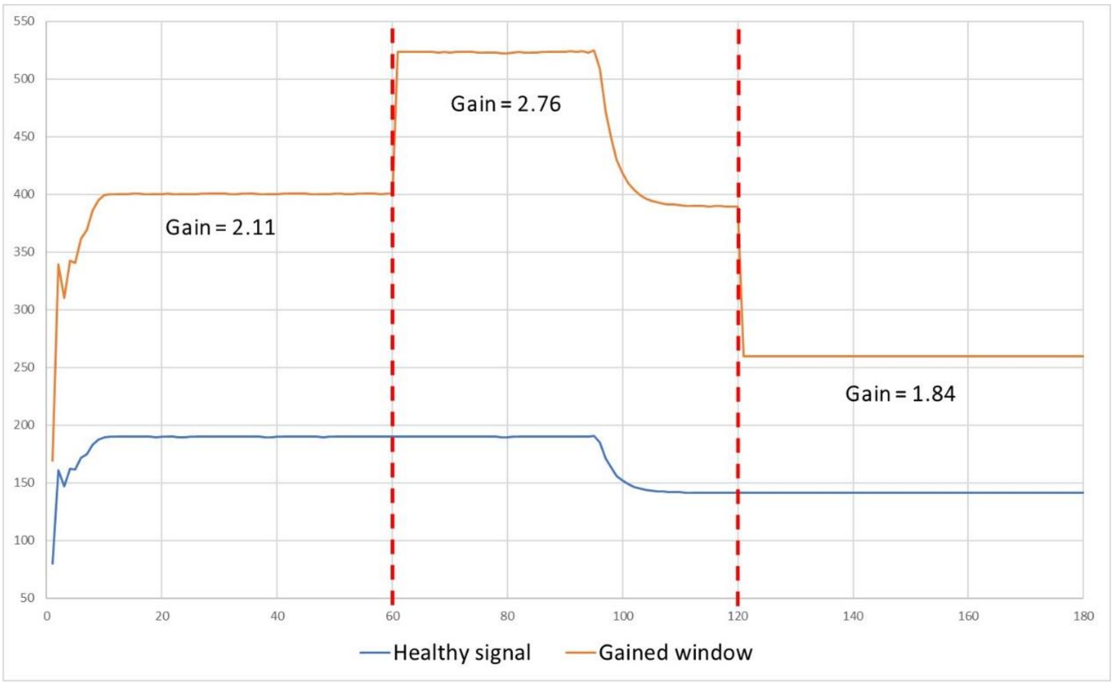


Figure 24 Gain Faults Verses Healthy Signal.

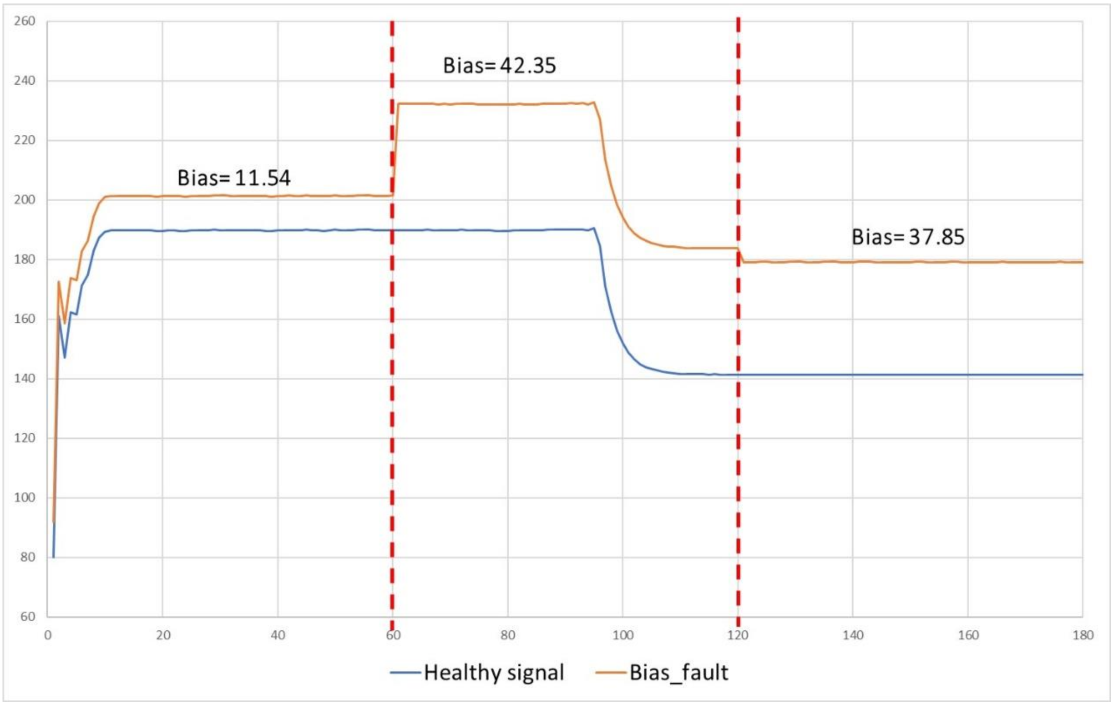


Figure 25 Bias Faults Verses Healthy Signal.

C. Sensor Fault Detection: Healthy Signal Reconstruction Using LSTM Autoencoder

This step is applied to feed the healthy windows of sensor PS1 into the LSTM autoencoder to be able to reconstruct the healthy form of an input signal for prediction that will be necessary to identify the faulty patterns by how much they are deviated from the healthy reconstructed one.
The steps below showcase the main steps to achieve sensor fault detection using LSTM autoencoder collaborated with the application of Pearson's autocorrelation.

(1) Define the problem. (2) Design the neural network suitable to solve the problem. (3) Pre-process the healthy PS1 data in a form useful for the designed neural network. (4) Compile the designed model with a fit compiler and libraries such as, Keras in Python. (5) Train and then validate the compiled model by calculating the necessary metrics. i.e., Mean Square Error (MSE). (6) Use the validated model to make predictions. (7) Find a suitable metric to measure the deviation between given signal and reconstructed one using the validated model. (8) Find the suitable threshold for the found metric. (9) Detect the faulty occurrences based on the calculated threshold.
The validated model can make predictions by reconstructing the healthy form of the given signal, which means that the outcome of the autoencoder is a signal, not a measurement to decide upon the status of the given signal to be healthy or faulty. The state-of-the-art research work suggest using the amplitude of the signal difference between the healthy reconstructed signal and the given signal, then define a threshold for the accepted amount of difference as the criteria to decide whether the signal is healthy or not. However, in this work we suggested a new way to calculate the signal difference using the Pearson's autocorrelation and then derive the suitable threshold from the difference extracted from the correlation.

In the following bullet points a full explanation supported by the experimental results in LSTM autoencoder to achieve sensor fault detection.

- Design LSTM Autoencoder for Sensor Signal Reconstruction

To achieve the problem under investigation, the desired neural network should be able to perform sequence to sequence predictions. Hence, the input sequence is sliding window of the sensor PS1 and the reconstructed signal is from the same nature of the input sequence, as well as they are both having the same size of 60. Then the encoder-decoder type required to fit the problem is an autoencoder. The choice of LSTM as a type of DL algorithm is because its tendency to learn the hidden dependencies between many time points at once, which make LSTM one of the most suitable forms of DL when it comes to time-series data, especially sequence to sequence (seq2seq) operations.

The LSTM autoencoder created for this experiment, has only one batch of LSTM sequences. This batch is designed to be sequential in direction and nature, which means the input layer is directly connected to the hidden layers, then the hidden layer is connected to the output layer. The LSTM hidden layer consists of a hundred hidden LSTM neurons. The activation function applied for the designed DL model is ReLU, based on a try and error validation. The hidden layer is chosen to be fully connected by adding the dense layer of output equal to the overall output expected from the LSTM model. The optimizer chosen for the LSTM layer is Adam optimization algorithm.

- **Data Preprocessing Prior to LSTM Autoencoder for Sensor Signal Reconstruction**

In order to utilize the healthy windows of PS1 for LSTM use, it must go under a heavy pre-processing and structuring to fit the LSTM criteria. The pre-processing and restructuring including the following: (1) Flatten the data into a vector. (2) Normalize the flattened data between zero and one to be able to use in LSTM. (3) Create the target sequence $y(t)$ to reconstruct this is the most important step of all, which determines what to learn and what to predict. In our case, the input sequence is a sliding window of size 60, while the target sequence is the next sliding window. The shift or sliding step is assumed to be only one step to guarantee higher model accuracy, which means if the input point is $x(0)$ then the target point used to train the prediction model is $y(0) = x(1)$. So that in general, $y(t) = x(t+1)$ (4) divide the flattened normalized vectors of $x(t)$ and $y(t)$ between training and testing samples, where the training windows are the 80% selected from the overall data, while the remaining 20% is divided equally between testing and prediction. (5) convert the flat, normalized vectors of $x(t)$ and $y(t)$ into a two-dimensional array of (number of samples, window size) shape. (6) convert the training and testing 2D tensor samples into a 3D tensor suitable to use in LSTM. LSTM units in Keras only accept the training and testing data in a 3D tensor shape following the size of (number of samples, time points, number of features). Where z-axis or pages or axis 0 is the number of samples, axis 1 or rows is the number of time points to store in the memory of LSTM and learn their dependencies, and finally axis 2 or columns represents the number of features inserted in the data. The $x(train)$ used in this experiment has the size of (11191, 60, 1), where 11191 of samples that has the size of (60,1) which is corresponding to one window of 60 only healthy readings of PS1.

- Train, Validate and Test the Designed LSTM Autoencoder Model

The previously designed LSTM model is trained and validated using the intensely pre-processed healthy data of PS1. In this experiment, the LSTM parameters are set to one hundred epochs and verbose equal one.

The validation results of the LSTM healthy signal reconstruction using the formulated testing data at the last epoch (number one hundred) has the errors MSE and MAE 0.000039871 and 0.0029, respectively, which both are considered exceedingly small loss values. Figure 26, shows a randomly selected window from the test samples compared to its reconstructed window using the trained and validated LSTM autoencoder. As shown the figure, the resemblance between the original, healthy window and the reconstructed window is extremely high, which provides another proof of efficiency besides the low error rate.

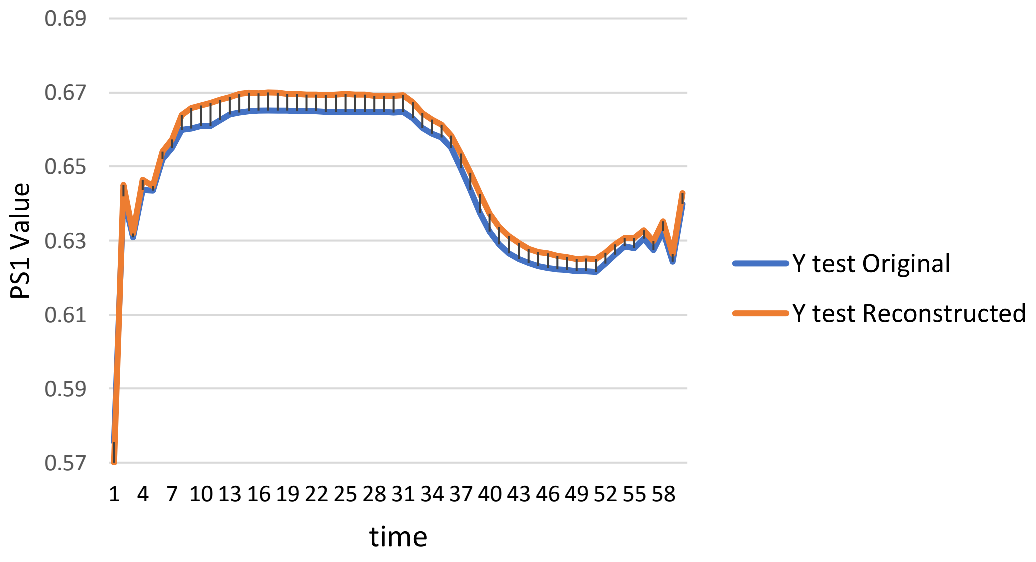


Figure 26 A Comparison Between a Randomly Chosen Testing Window and its Corresponding Reconstructed Signal.

- **Determine the Best Signal Difference Metric and its Threshold:**

After training, evaluating, and testing the LSTM autoencoder model, it is time to start making fault detection decisions aided by the model, but the question arises, how to detect faults based on the quality of the reconstructed signal? Which brings up another important question: How to determine the fault detection threshold?

Taking a glance at the state-of-the-art methods helps answering the previous questions, as such the study researched in [154]. The approach they have applied is highly similar to our approach by having separate phases for both detection using signal reconstruction, and diagnosis applying fault classification. To find the difference between the predicted sequence and the input sequence, they used signal difference that can be easily calculated by taking the amplitude of the subtraction operation between the two sequences ($z(t) = |x(t) - x'(t)|$). Although using signal difference has shown accurate results, we propose a different signal similarity measure that showed more accuracy and performance when comparing to signal difference for fault detection.

The threshold determines what is faulty or healthy based on the value of the signal difference. If the value is higher than the designated threshold then the reconstructed signal is considered faulty, or else it is healthy. The threshold is best measured by creating a pool of various threshold values between the minimum and maximum values of the calculated signal difference. Followed by making the fault detection decisions on the prediction samples, based on each one of the thresholds in the pool. For each threshold in the pool, check if the prediction's sample signal difference is higher than the threshold to be considered faulty, while lower is detected to be healthy. Finally, calculate the precision, recall, f1-score, and accuracy for all the prediction results made by each threshold in the pool. The choice of the right threshold for the sensor fault detection, is made by choosing the threshold that guarantees the best precision to recall trade-off, also known as f1-score. In this experiment, a prediction dataset (500 windows of size 60 readings of PS1) that consists of various healthy and faulty samples, is used to determine the fault detection accuracy using the LSTM autoencoder sequence reconstruction. The 500 windows reconstructed using LSTM were then each compared to the original sequence to show how much they were deviated from the original window regarding their health status. The comparisons between the reconstructed windows and the original ones are made utilizing two main metrics: (1) signal difference: $z(t) = |x(t) - x'(t)|$. (2) our new metric using the complement of Pearson's autocorrelation.

Pearson's autocorrelation can be calculated using the formula shown below:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \qquad \text{Eq. 18}$$

Where r_{xy} is the correlation between to vectors x, y. Furthermore, x *and* y are expected to possess the same length of n. Where the autocorrelation measures the similarity between two sequences, while subtracting the measured similarity from the highest possible value of resemblance (+1) represents another way of calculating the difference between two sequences. $(z(t) = 1 - (Correlation))$

The table below, Table 8 shows the signal difference values, and the signal difference based on the correlation values, for the first five entries of the prediction samples after being compared to their reconstructed version using LSTM.

Table 8 The First Five Samples and Their Signal Difference when Compared to the Reconstructed Signal, Calculated in Two Different Fashions

Difference Metric/samples	1	2	3	4	5
Signal Difference: $\mathbf{z(t) = \lvert x(t) - x'(t) \rvert}$	0.0806	0.6880	0.8279	0.8485	0.6538
Pearson's Correlation: $\mathbf{r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}}$	0.4281	-1	0.1784	-0.5772	-1
Signal difference= $\mathbf{z(t) = 1 - (Correlation)}$	0.5718	2	0.8215	1.5772	2

The tables demonstrated below shows some of the threshold values selected to find the optimal threshold necessary for the sensor fault detection, corresponding to their precision, recall, f1-score, and accuracy.

Table 9 Signal Difference Thresholds and Their Metrics [78].

THRESHOLD	**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**1.1**
PRECISION	0.78	0.76	0.74	0.69	0.65	0.79
RECALL	0.89	0.77	0.69	0.54	0.33	0.2
F1-SCORE	0.83	0.76	0.71	0.61	0.43	0.32
ACCURACY	0.71	0.62	0.55	0.44	0.32	0.32

Table 10 Signal Difference Using the Correlation and Their Metrics [78].

THRESHOLD	0.1	0.3	0.5	0.7	0.9	1.1	1.3
PRECISION	0.79	0.8	0.81	0.83	0.84	0.85	0.86
RECALL	0.95	0.85	0.82	0.72	0.64	0.55	0.5
F1-SCORE	0.86	0.82	0.81	0.77	0.73	0.66	0.63
ACCURACY	0.76	0.71	0.7	0.66	0.61	0.56	0.53

Based on the values shown in Table 9 and Table 10. The optimal threshold based on each signal difference metric can be easily detected by choosing the threshold that provided the best precision, recall trade-off. It is apparent that the threshold of 0.3 is the optimal threshold when using the regular signal difference metric, and the accuracy of the LSTM autoencoding sensor fault detection when using the optimal threshold of 0.3 is 0.62. On the other hand, the optimal threshold when using the signal difference based on the correlation is 0.5, and the accuracy of the sensor detection given the optimal threshold is 0.71.

As visualized in Figure 27 and Figure 28. The optimal threshold is the one providing the best precision, recall trade-off also known as f1-score. The optimal threshold can be easily observed as the intersection point between the three metrics mentioned previously. Based on the visualization in Figure 27, the threshold selected is 0.3, which provide the detection with 0.62 accuracy. Compared to the intersection point shown in Figure 28, where the threshold is chosen 0.5, and the corresponding accuracy is observed higher at 0.71.

This concludes the accuracy of the proposed signal difference measure comparing to the traditional one, to achieve fault detection using signal reconstruction technique.

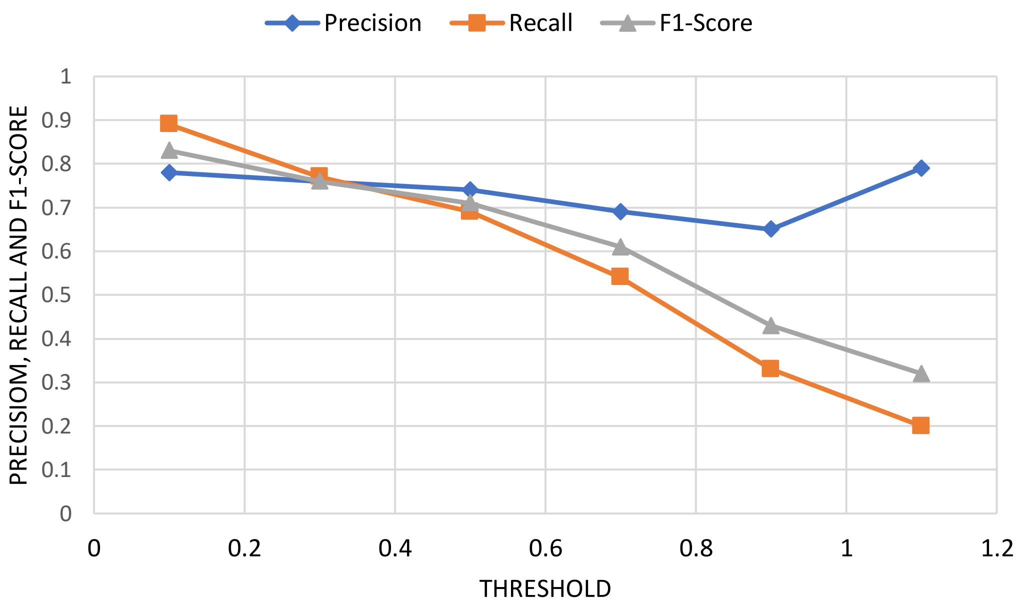


Figure 27 Optimal Threshold Selection Using Regular Signal Difference [78].

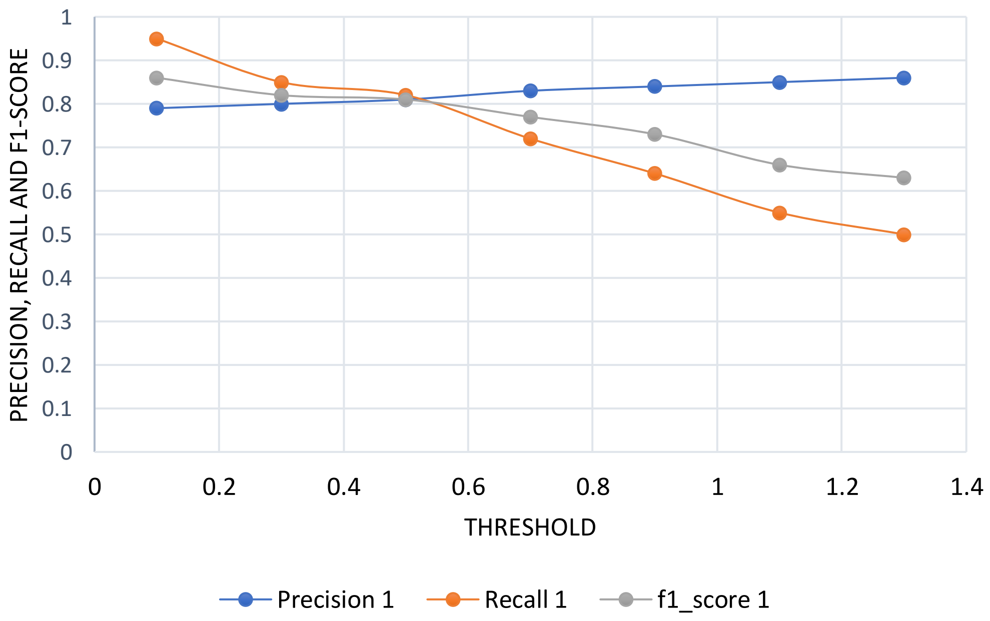


Figure 28 Optimal Threshold Selection Using Signal Difference Based on Correlation [78].

D. Sensor Fault Diagnosis: Classification Schema

In this section, the experimental results conducting sensor fault diagnosis using a variety of supervised learning algorithms is demonstrated. As shown in Figure 22, the second phase following the detection of existing anomalies is applying the necessary means to diagnose their nature. The detected faults by the previous phase using the LSTM autoencoder, are then fed into a fault classification schema to determine the type and nature of the detected faults. In other words, to perform the fault diagnosis only faulty data is classified.

PS1 sensor data is pre-processed, and then restructured into a two-dimensional matrix of 28882 windows of size 60. Three main faults existing in the training data provided to the classifiers; constant faults of three different types, gain fault, and bias or offset faults. The fault injection schema and its details are explained earlier in this experiment.

In this work, the classification results are compared when various feature engineering approaches are applied. The feature engineering approaches used for this section are: PCA, FI, manually extracted time-domain features, and new cluster-based feature selection method called R*k*SE. Feature selection or extraction when applied to univariate datasets in a shape of sliding windows, is simply considered as a window compression method, to minimize the size of the readings provided by each window and select the features with most contribution to the learning process. Therefore, the time and complexity constraints of the ML or DL models can be managed and minimized with smarter choice of features. Various ML and DL classifiers has been trained, validated, and tested, individually with the selected features using a diversity of feature engineering approaches. Then their results are documented and compared. The ML approaches used are LR, LDA, KNN, CART, NB, SVM and RF. The DL approaches selected to perform the classification tasks are CNN and LSTM.

The following experimental results tackle each of the feature engineering process and their results, when fed into the mentioned above ML and DL classifiers, to eventually achieve the FDD for sensor faults, using PS1 sensor as an example of sensors in the hydraulic test rig system.

The feature selection section for PS1 as a window compression method, is thoroughly explained, as well as all the mentioned methods are explained in detail in the previous chapter (chapter 5).

The parameters selection for each classifier was chosen by trial and error, to ensure the highest possible accuracy when validating using 10-fold cross validation over the original data without any feature engineering applied. The table shown below describes the applied ML classifiers and their corresponding parameters using Scikit in Python. Furthermore, the mean accuracy for the 10-folds and the standard deviation corresponding to the 10-folds is calculated.

Table 11 ML Classifiers and Their Parameters in Scikit Python.

ML Classifier	Parameters	Mean Accuracy	Standard Deviation
LR	solver='liblinear', multi_class='ovr'	0.69	0.02
LDA	no parameters	0.71	0.01
KNN	no parameters	0.87	0.01
CART	no parameters	0.90	0.01
NB	no parameters	0.71	0.01
SVM	gamma='auto'	0.91	0.01
RF	n_estimators=1000, max_depth=5, random_state=0	0.82	0.01

The CNN classifier has the parameters verbose, epochs and batch size of zero, hundred and 20, respectively. The parameters are chosen by try and error to provide the designed deep neural network with the highest 10-fold classification accuracy. The CNN applied is designed as sequential model (input, hidden layers, and output). The CNN convolutional layer applied here is a 1-D layer since the training dataset is a time-series data and of a one-dimensional nature, unlike the usual application of CNN where the data is typically of two-dimensional shape such as, images. CNN design included 6 one-dimensional convolutional layers of filter equals to 64 and kernel size of one. The kernel size that shows the length of the convolution window/masking window required for the convolution is selected as one. The number of convolution layers is added to guarantee highest possible accuracy, and by try and error it is set to 6 layers of 1-D convolutional layers. The activation function within the created layers is ReLU. The next process following each convolution layer is the pooling layer, in this work the pooling function is selected as the maximum pooling, which indicates selecting the maximum entry in the kernel during the pooling phase. Two fully connected layers are added following the pooling phase, one of size hundred and their activation function is ReLU. The second fully connected layer has three outputs to match the number of classification outputs/faults designated for the training, while its activation function is selected as SoftMax. Finally, the CNN optimizer chosen is Adam optimizer. The LSTM model designed for classification differs from the one used in the previous step, as this model is a classifier while the previous LSTM model is an autoencoder designed to solve multi-regression problems not classification. Only one batch of LSTM neurons is used, this batch has hundred sequential hidden layers or neurons. The layers are fully connected using a dense layer of size one hundred with the activation function ReLU, which is connected to another fully connected dense layer of size three (to match the number of outputs expected from the LSTM model), with the activation function SoftMax. The LSTM classifier parameters are represented by verbose, epochs and batch size which are equal to zero, 10 and 20, respectively.

Here are the classification results when using number of ML and DL classifiers when fed with only faulty data, to perform PS1 fault diagnosis and classification. In Table 12, nine ML and DL classifiers are trained separately using five different features at a time, which are selected/extracted using PCA, manually extracted time-domain features, FI, R*k*SE. As well as the entire faulty dataset without any feature selection. The number of features without feature selection is equivalent to the window size of 60. Four features are extracted using PCA, 45 features are selected using FI, compared to 46 features selected using R*k*SE. Finally, four time-domain features extracted from each window, represented by the mean, variance, standard deviation, and signal to noise ration. The number of features selected by each method is the one with the highest fault classification mean accuracy.

Table 12 Classification Accuracy of Different ML and DL Approaches Using Various Feature Engineering Methods [78].

Classifier	no feature selection	PCA	Time-Domain Features	FI	RKSE
LR	0.6911	0.6356	0.2425	0.7019	0.6882
LDA	0.7053	0.6535	0.7859	0.7038	0.7068
KNN	0.8747	0.9128	0.9625	0.8758	0.8816
CART	0.8972	0.9818	0.9951	0.9126	0.9116
NB	0.7089	0.6919	0.4821	0.6930	0.6824
SVM	0.9125	0.8827	0.7298	0.9112	0.9142
RF	0.8189	0.8390	0.9402	0.8196	0.8193
CNN	0.8773	0.8486	0.7575	0.8385	0.8562
LSTM	0.8352	0.9568	0.9684	0.7278	0.7499
Mean Feature Accuracy	0.81	0.82	0.76	0.80	0.80

When observing each row in respect to each feature engineering method, it can clearly show the feature engineering approach giving the highest or lowest 10-fold mean accuracy corresponding to each classification method. The mean feature accuracy row shows the overall accuracy for each feature engineering approach in respect to all ML and DL classifiers combined. It is shown that PCA has the highest mean feature engineering accuracy when applied to the nine classifiers, which proves the consistency of PCA and its validity with different classification techniques. It is also obvious that the time-domain features selected are the one providing highest accuracy to many of the ML and DL classifiers, which are LDA, KNN, CART, RF and LSTM. However, this feature selection technique does not provide consistency in the accuracy results, since LR and NB classifiers for example show exceptionally low performances when applying the four selected time-domain features, comparing to the rest of feature engineering methods. This explains why time-domain features result in lower overall mean feature accuracy comparing to PCA, even though more classifiers have the maximum accuracy when applying time-domain features.

The selection of the suitable feature engineering method is highly dependent on each classifier type and its functionality. The table above has the purpose of investigating the behavioral changes of some of the most common ML and DL classifiers in respect to various commonly used feature engineering methods with time-series datasets. Furthermore, finding the best pair of features and classification approach which provides the most optimal accuracy-complexity trade-off when performing sensor fault detection is the number one aim of these comparisons. As a result, the highest measured sensor fault detection combination is when applying CART using time-domain features, followed by LSTM, KNN and RF using the same extracted features.

Finally, the sensor fault detection for new windows of PS1 (size 60) at run-time, can be easily performed by applying the new window into the trained, LSTM autoencoder explained in full details previously in this experiment. Followed by the reconstructed healthy signal of the new window will be compared to the new window by calculating the signal difference using the Pearson's correlation. If the difference is bigger than the previously trained threshold, then a fault is detected and needs to pass to the next stage of fault diagnosis using the chosen trained classifier, by feeding the new window into the chosen trained classification model after applying the necessary feature engineering modifications. In our case, extracting the time-domain features such as, mean, variance, standard deviation, and signal to noise ratio to the signal. Then, the created vector of size four (four features extracted) is used to make predictions using the trained CART, LSTM or RF classifiers, as they showed better results than others when combined to the chosen features.

3.2 Experiment Two: Component FDD Using the Joint LSTM Autoencoder and Classifier Approach

In this experiment, component faults existing in the hydraulic test rig are detected and diagnosed using a unique approach, in which the detection and diagnosis stages are done separately to ensure more accurate detection of rare occurrences. Figure 29, shows the framework of this experiment.

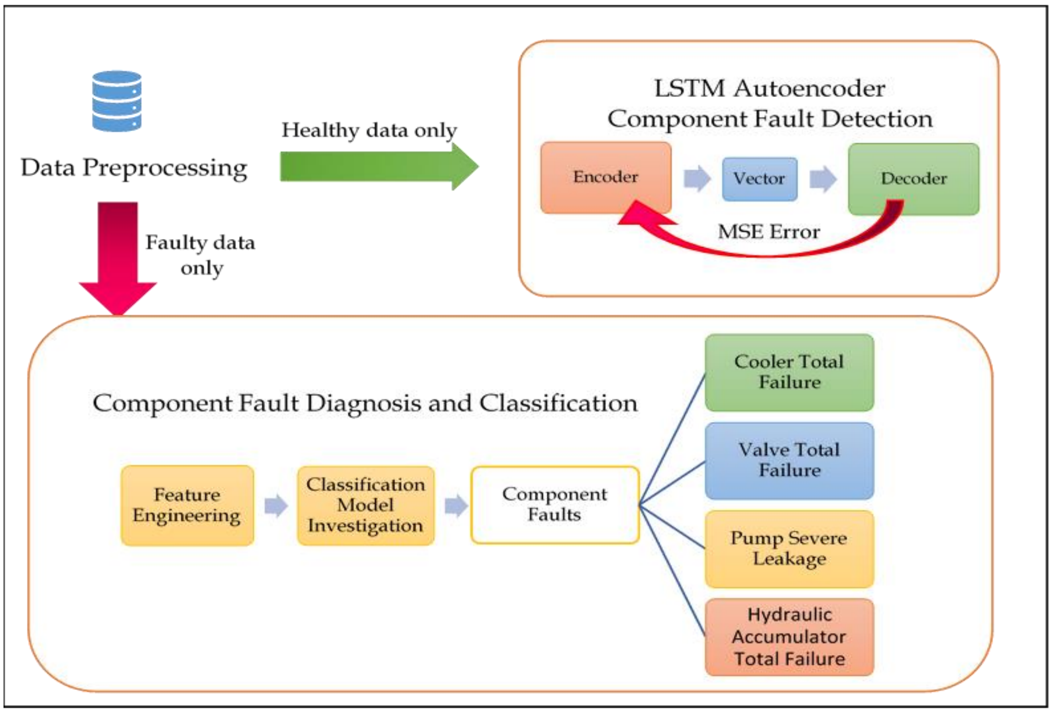


Figure 29 Component FDD Comprehensive Framework [78].

The same steps and parameters created in experiment one (sensor FDD) are repeated in this experiment, excluding the data pre-processing and structuring, and the fault injection schema. The pre-processing step here differs from the previous experiment, since this time it is a multi-variate autoencoding and classification problem, without the application of sliding windows. Moreover, no fault injection is required in this experiment because the component faults studied are already available in the hydraulic test rig dataset used for this experiment.

A. Data Pre-Processing and Organization

In this section, the data used is the hydraulic test rig dataset of eleven sensors, which indicates that this experiment is a multi-variate FDD experiment. The healthy data is applied for the detection as the first step of the FDD system represented by the LSTM autoencoder. While the faulty data containing four main component faults; cooler, value and hydraulic accumulator total failures, and pump severe leakage fault, are used in the second stage represented by the fault diagnosis using the supervised ML and DL methods.

In both stages, the data is organized as a 2D matrix of samples and the features expressed by the eleven sensors and their readings at different time points.

B. Component Fault Detection: LSTM Autoencoder

This section has the same procedure explained in experiment one for fault detection in sensors. The LSTM autoencoder for fault detection stage has the following main steps: (1) design the LSTM autoencoder to fit the problem. (2) Prepare the data into a form acceptable in the LSTM. (3) Train and validate the LSTM autoencoder using healthy data only and calculate the MSE and other error metrics. (4) Predict the samples that contain fault and healthy readings. (5) Calculate signal difference between the original samples and the predicted samples, using the regular difference and the Pearson's correlation one, to establish accuracy comparisons. (6) Find the best threshold of sequence difference to ensure the best trade-off between precision, recall and f1-score. (7) Make component fault detection decisions using the trained, validated LSTM autoencoding model, and their calculated sequence difference compared to the computed threshold.

For training the designed model, 1438 samples of the eleven sensors' reading are used to train and validate the model. The data should be normalized between zero and one, as well as converted to a three-dimensional tensor format (samples, time points for LSTM to remember, number of features) before applying to the LSTM model. The LSTM model designed for component FDD detection is an autoencoder of a sequential hundred hidden LSTM neurons using the activation function ReLU, then a fully connected dense layer of size equal to the number of sensors or features is added. The dense layer contributes to improving the accuracy of the LSTM model, as well as making sure to the LSTM model generates outcomes equal in size with the designated input signal. Finally, the optimizer applied for the LSTM model is Adam. The training parameters of the LSTM autoencoder are epochs= 100 and batch size= 30.

After training and validating the designed model over a hundred epochs, the MSE error of the last epoch is 0.00057 and the MAE is equal to 0.0096. Both error metrics are exceedingly small, which is a high indication of the validity and accuracy of the created model in reconstructing healthy input sequences.

To select the optimal threshold corresponding to the allowed sequence difference between the original sequence and the reconstructed one resulted from the LSTM autoencoder. 4800 samples of size eleven are predicted using the autoencoder, to reconstruct 4800 healthy versions of the prediction samples. The signal difference using between each of the corresponding sequences in the original and reconstructed sequences are computed using (1) the traditional signal subtraction to find the signal difference as a vector, then find the magnitude of this vector. (2) The sequence difference using (1-Pearson's autocorrelation) as a measurement created in this work and proposed to be more accurate measurement for fault detection than the traditional signal subtraction.

To avoid repeating the explanation of each signal difference methods, we will jump right through the results and their comparisons.

A pool of candidate threshold values is created, then the labels of the 4800 prediction samples were obtained based on each threshold in the pool, if the value of the signal difference is higher than the threshold a fault is considered to be detected, thus label 1, else label 0. The precision, recall, and f1-score are computed to each threshold in the pool based on the generated labels and the original labels of the given prediction samples.

When applying signal difference using the Pearson's autocorrelation, the pool of chosen thresholds between the minimum and maximum observed values are shown in Table 13. Furthermore, for each chosen threshold the accuracy, precision, recall and f1-score are computed. Figure 30, illustrates the process of choosing the component fault detection threshold based on the precision, recall and f1-score trade-off shown in the table below. As clearly shown in Figure 30, the selected threshold is the intersection between the three curves, which is approximately equal 0.0007 and the accuracy observed for this threshold value is 0.71.

Table 13 The Thresholds of Pearson's Correlation Difference and Their Corresponding Fault Detection Accuracy, Precision, Recall and F1-Score [78].

THRESHOLD	**0.0001**	**0.0003**	**0.0005**	**0.0007**	**0.0009**	**0.001**	**0.003**	**0.005**
PRECISION	0.63	0.73	0.77	0.78	0.79	0.81	0.95	0.93
RECALL	1	0.79	0.74	0.74	0.72	0.68	0.25	0.14
F1-SCORE	0.77	0.76	0.75	0.76	0.75	0.74	0.39	0.24
ACCURACY	0.63	0.69	0.7	0.71	0.7	0.69	0.52	0.46

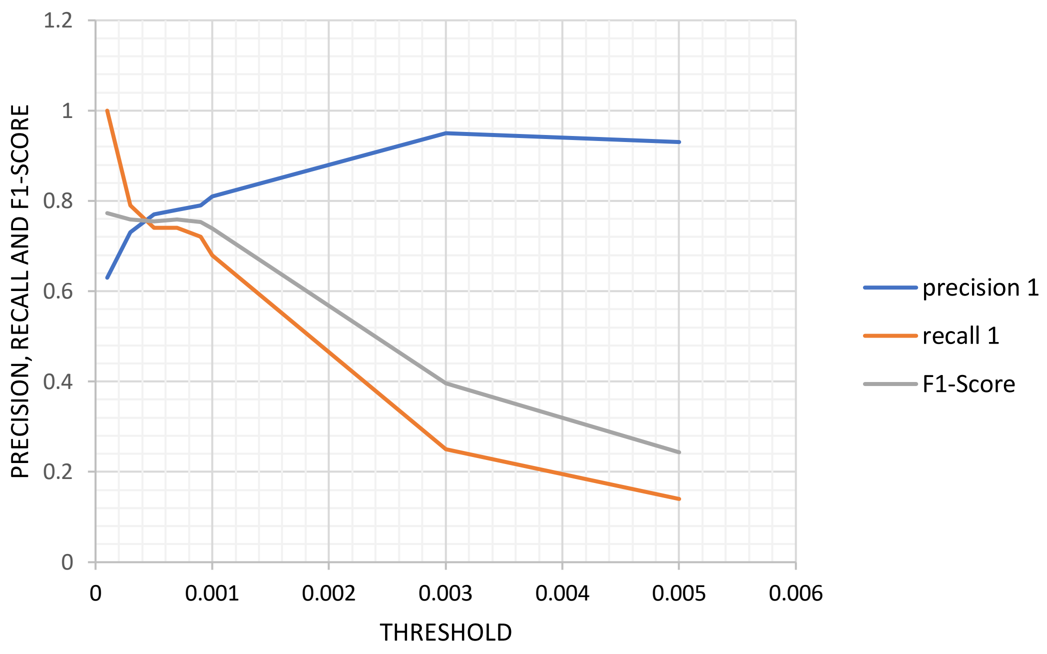


Figure 30 Precision, Recall and F1-Score Trade-Off for Threshold Selection Using Pearson's Correlation Difference [78].

On the other hand, Table 14 and Figure 31 shows the pool of threshold and their metrics when using the traditional subtraction as signal difference method is shown.

Table 14 The Thresholds of Subtraction Difference and Their Corresponding Fault Detection Accuracy, Precision, Recall and F1-Score [78].

CANDIDATE THRESHOLDS	0.02	0.03	0.05	0.07	0.09	0.1
PRECISION	0.63	0.73	0.77	0.92	0.92	0.89
RECALL	0.99	0.76	0.66	0.26	0.15	0.1
F1-SCORE	0.77	0.74	0.71	0.41	0.26	0.18
ACCURACY	0.63	0.69	0.66	0.52	0.46	0.43

The intersection between the three curves in Figure 31, shows that the optimal threshold for component fault detection using the sequence subtraction is approximately 0.03, with the fault detection accuracy of 0.69. The optimal accuracy using signal subtraction of 0.69 is less than the measured one using the optimal threshold computed by Pearson’s correlation of 0.71.

As a result, when comparing the accuracy of the optimal thresholds selected using two different signal deviation measurements, which are the autocorrelation complement and the traditional subtraction. It is clearly shown that the proposed method using Pearson's correlation guarantees higher component and sensor faults detection accuracies, comparing to its commonly used traditional subtraction counterpart, based on the measured comparisons in experiment one and two.

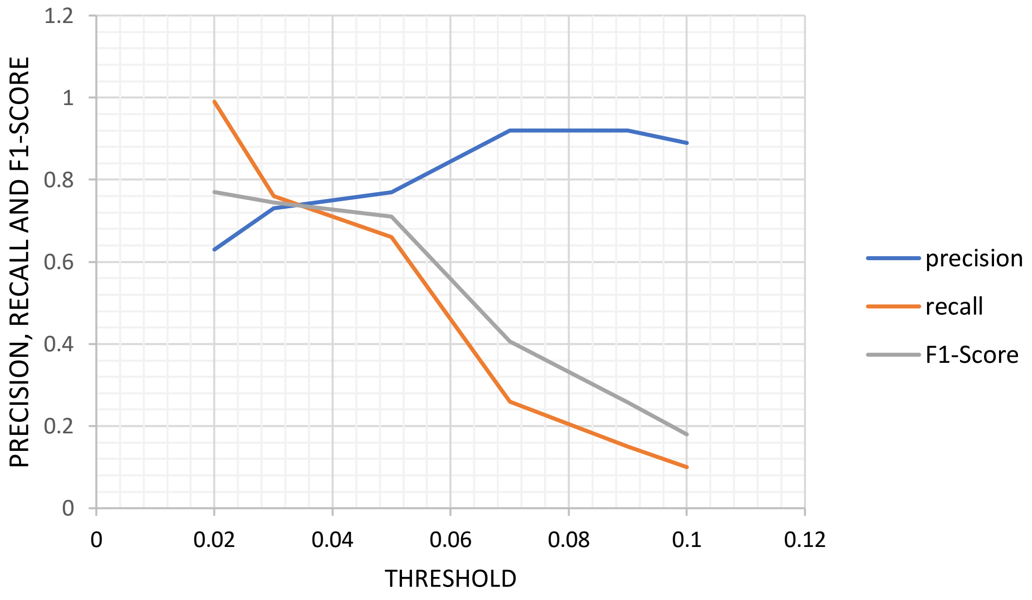


Figure 31 Precision, Recall and F1-Score Trade-Off for Threshold Selection Using Signal Subtraction Difference [78].

C. Component Fault Diagnosis: Classification Schema

In this section, the feature engineering methods compared are FI, PCA and R*k*SE. The time-domain extracted features applied for multi-variate time series sequences without the application of sliding windows, are expected to have lower accuracy values regardless the ML or DL classifier used. Hence, it does not make sense to compute the mean, standard deviation, and variance to a sample of readings extracted from sensors of different nature. However, the time-domain features were extracted and applied to all the classifiers anyways, to prove the point mentioned earlier.

Based on the feature engineering thorough analysis in chapter 5, the optimal number for each feature engineering method is computed. The overall number of features in each sample is eleven, corresponding to each sensor in the hydraulic test rig dataset explained in chapter 2. The optimal features for FI, PCA, time-domain features and R*k*SE are four, five, four and nine, respectively.

The accuracies computed for all the ML and DL classifiers are results of dividing the fault data with component faults, into training and testing data, with percentages of 80% to 20% of the faulty data, respectively. Followed by applying 10-fold cross validation technique for each classifier separately.

The parameters and optimizers for each ML method used in this section, are identical to the ones used in the sensor FDD experiment earlier in this chapter and shown in Table 11. Moreover, some minor changes in the CNN and LSTM classifiers' design and parameters have been made comparing to the previous experiment.

CNN design consists of only one 1D CNN layer of filter size 64, and kernel size of one. The layer is sequential which means the input layer is directly connected to the hidden layer(s) that is connected to the output layer. The activation function used is ReLU. Followed by the pooling layer that has pooling size of one and applies maximum pooling as the pooling function. Finally, a fully connected dense layer of size equivalent to the number of expected outputs, with SoftMax activation function is created to make the classification process for the extracted features during the convolutional and pooling layers. The CNN optimizer used is Adam, as a stochastic gradient descent approach to optimize the network.

The LSTM classifier applied has only one LSTM batch with two hundred hidden neurons that are sequential in order and nature. Followed by a fully connected dense layer of SoftMax activation function, and naturally Adam is the applied optimizer for the LSTM as well. The verbose, epochs and batch size parameters are applied by testing various values and their effect on the classification accuracy, and they are set to zero, 10 and 20, accordingly.

The table below comprehend the component fault classification results when trained by faulty hydraulic test rig data, applying various ML and DL approaches using numerous feature engineering methods.

Table 15 Component Fault Diagnosis Using Various Feature Engineering and Classification Approaches [78].

METHOD NAME	FI	PCA	RKSE	TIME-DOMAIN FEATURES	NO FEATURE SELECTION
LR	0.9962	0.7300	0.7823	0.37599	0.6832
LDA	0.7634	0.7490	0.7528	0.370521	0.7031
KNN	0.9940	0.9229	0.9320	0.831458	0.8677
CART	0.9932	0.9435	0.9912	0.928594	0.6849
NB	0.9924	0.7510	0.7122	0.39526	0.9035
SVM	0.9859	0.9337	0.9310	0.833281	0.8139
RF	0.9930	0.9013	0.9910	0.871042	0.9042
CNN	0.7343	0.8427	0.7343	0.3971	0.7385
LSTM	0.7375	0.8770	0.7375	0.3981	0.73124
MEAN ACCURACY	0.910	0.850	0.840	0.600	0.781

As shown in Table 15, feature selection approaches work better than extraction ones such as, PCA and time-domain features when dealing with traditional ML classifiers, to classify multi-variate time series datasets. On the one hand, FI is consistently showing highest accuracy results comparing to the rest of the feature engineering methods when dealing with traditional ML classifiers. Followed by R*k*SE, that is yet slightly lower accuracy than FI for traditional ML approaches, but it shows consistency in all ML classifiers. On the other hand, PCA has shown the highest accuracy when applying DL classification algorithms comparing to other feature selection approaches. While FI and R*k*SE are neck to neck when it comes to the classification accuracy using the selected DL approaches. Finally, even though time-domain features were the most accurate ones when applied to sliding-windows for univariate classification as shown in experiment one. As spotted earlier in this experiment, when it comes to extracting time-domain features from multi-variate datasets without applying sliding windows, this feature extraction method is proven weaker than the rest of the approaches, when combined to regardless ML or DL classifiers.

In another side of comparison, KNN, CART, RF and SVM showed great consistency in high accuracy results no matter what feature engineering method is applied, including time-domain features regardless its weakness. CNN and LSTM had lower accuracies comparing to the traditional ML approaches mentioned earlier, and their accuracies drops radically when time-domain features are applied.

To conclude experiment two, it is important to know how to apply the trained models and saved parameters from experiment two at run-time, to make new real-time predictions. The input vector for prediction should have one readings of each sensor of the eleven sensors used for the training process during the offline or training phase. (1) fault detection: fault detection for new samples can be done by; feeding the new sample into the trained and validated LSTM autoencoder model to reconstruct the healthy form of the sequence. Then the reconstructed sequence and the original one is compared by calculating the signal difference using Pearson's autocorrelation. Finally, with the signal difference calculated is above or below the trained threshold, this will determine the existence of faults or not. (2) Fault diagnosis: in case a fault is being detecting using the detection step. The fault should be diagnosed by applying the necessary feature engineering approaches, then feeding the processed new sample to the highest accuracy ML or DL trained classifier, suitable to the features chosen. In our case, based on the results shown in Table 15, it is more accurate to use RF a combined with FI to guarantee better accuracy and complexity trade-off.

Chapter 6: A Hybrid Approach: Dynamic Diagnostic Rules for Hydraulic Systems in Industry 4.0 Generated by Online Hyperparameter Tuned Random Forest

Please note that parts of this chapter have been published under our own article in [155], and all copy rights and other rights are reserved to the MDPI publisher.

1. Chapter Overview

In this chapter, a hybrid component FDD approach for a hydraulic system extracted from a hydraulic test rig is established and analysed, to provide a hybrid schema that combines the advantages and eliminates the drawbacks of both model-based and data-driven methods of diagnosis. Moreover, it shines light on a new utilization of RF together with model-based diagnosis, beyond its ordinary data-driven application. RF is trained and hyperparameter tuned using three-fold cross validation over a random grid of parameters using random search, to finally generate diagnostic graphs as the dynamic, data-driven part of this system. This is followed by translating those graphs into model-based rules in the form of if-else statements, SQL queries or semantic queries such as SPARQL depending on the type of the model-based system available, in order to feed the dynamic rules into a structured model essential for further diagnosis. The RF hyperparameters are consistently updated online using the newly generated sensor data to maintain the dynamicity and accuracy of the generated graphs and rules thereafter. The architecture of the proposed method is demonstrated in a comprehensive manner, and the dynamic rules extraction phase is applied using a case study on condition monitoring of a hydraulic test rig using time-series multivariate sensor readings. Furthermore, the proposed method contributes to providing a way of dynamizing already existing model-based FDD systems containing fixed rules, by automating the rules and replacing them with the dynamic, RF generated ones. This work is an optimization of the model-based FDD approach using semantic ontologies described in [11]. In order to offer a hybrid-approach that is dynamic, and less complex than the one relying on model-based FDD for component faults in industrial systems.

2. System Model Overview

The proposed system scenario in Figure 32 represents a possible method to merge RF as a data-driven approach for industrial FDD and model-based FDD approaches into a final hybrid approach, which possesses the powerful features of both approaches. This technique eliminates the main drawbacks of each approach individually such as, the lack of dynamicity and response to sudden occurrences in case of traditional model-based FDD. As well as, providing, dynamic diagnostic rules that contribute massively to reducing the diagnostic time and computational needed resources, comparing to their online data-driven counterparts. The figure below shows the two main diagnosis phases used in this research; data-driven and model-based, and how these two methods are combined into a new improved approach.

The following is a comprehensive explanation of each phase:

A. Data-Driven Phase

This phase consists of multiple internal steps essential to learn the best possible dynamic diagnostic rules using random forest algorithm. Below, each step is discussed in an elaborate manner:

- Multivariate Time-Series Dataset

In this work, the dataset used is a multivariate, time-series dataset, of six pressure, four temperature, two volume sensors and one vibration sensor which all possess a constant cycle of 60 seconds, placed in a hydraulic test rig to monitor its condition over time. For more details about this dataset and its previous applications, please refer the data collection and generation section, in this chapter.

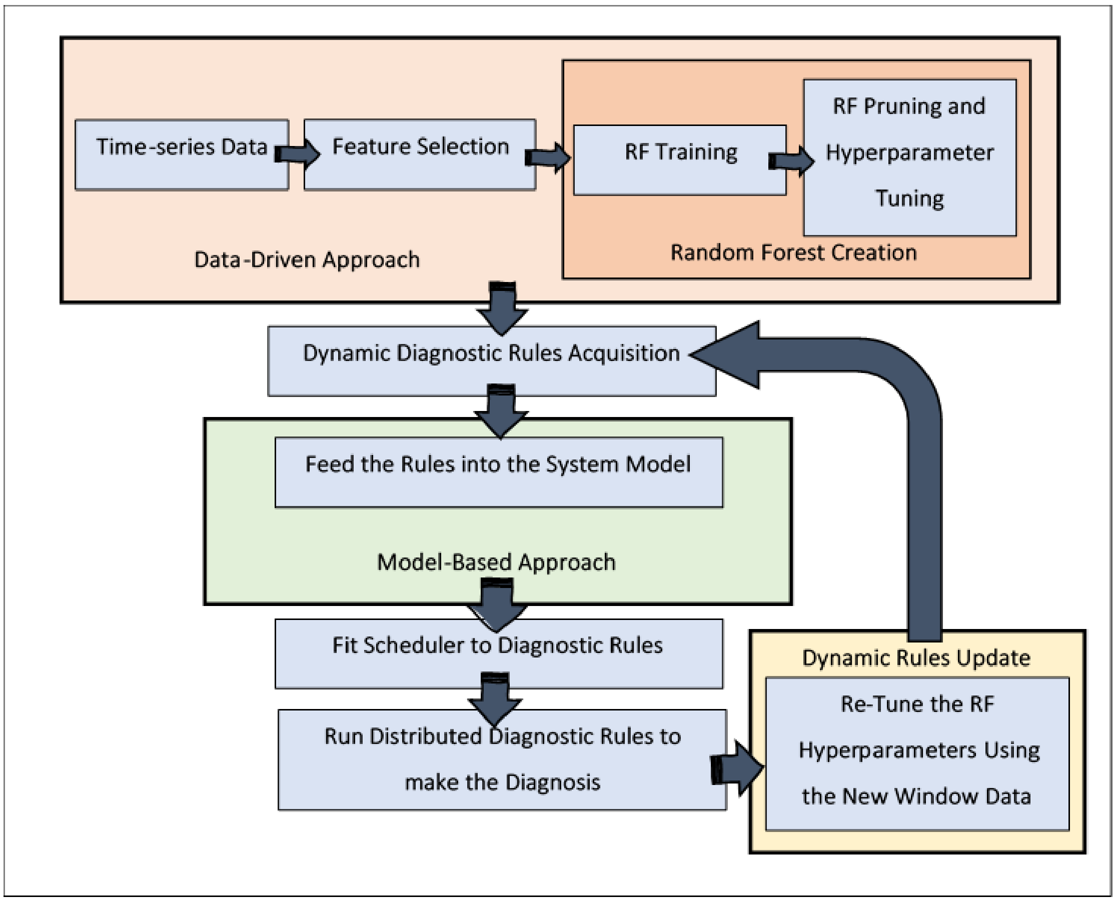

Time-series Data
Feature Selection
RF Training
RF Pruning and Hyperparameter Tuning
Random Forest Creation
Data-Driven Approach
Dynamic Diagnostic Rules Acquisition
Feed the Rules into the System Model
Model-Based Approach
Fit Scheduler to Diagnostic Rules
Run Distributed Diagnostic Rules to make the Diagnosis
Dynamic Rules Update
Re-Tune the RF Hyperparameters Using the New Window Data

Figure 32 Hybrid-based FDD System Overview [155].

- Feature Selection

Complex sensor industrial systems often have hundreds or even thousands of sensors connected, simultaneously transmitting sensors' reading data crucial to monitor and control those systems. Each of which is considered a feature for analysis and model training. Thus, creating diagnostic models that only include valuable features is a necessity.

Implementing a model with less but more meaningful features have a significant impact on the overall system. First, the diagnostic model become simpler to analyse and interpret when fewer elements are included. Second, by eliminating some features of the dataset, the data would be less scattered hence less variant, which can lead to reducing overfitting. Finally, the main reason behind feature selection is generally to reduce the time and computational costs required to train the model.

In practice, RF algorithm can be applied to carry out feature selection as well. Simply because the features are implicitly ranked based on their impurity during the formation of each decision-tree creating the forest. In other words, when top-down traversing a tree in RF, the nodes toward the top happened to have the largest impurity decrease metric, also known as Gini Impurity (GI), comparing to the nodes at the bottom. Thus, by determining a particular impurity decrease threshold, it is possible to prune the tree below this tolerance, in order to establish a subset of the most fitting or important features.

The data-driven FDD method implemented in this work is RF. In intention of reducing the computational cost as possible, RF is also used to perform feature selection using what is known as *feature importance* or *permutation importance* [52] and [73]. Since, GI calculations are already measured during the RF training process, and only a slightly bit of additional computations are required to complete the feature selection process.

The most popular implementation of feature importance provided by RF, is the Python library Scikit-learn where a pre-defined function feature_importances_ is directly executed given the learned RF model. However, a team of data scientists at the University of San Francisco pointed out some bugs associated to this function and implemented an alternative to generate more accurate feature importance results in [156].

- Hyper-Parameter Optimization

The foremost goal of any machine learning algorithm is to minimize the expected loss as possible. To achieve this, it is consequential to deploy some optimization equations to select the optimal values for some, or all the hyperparameters of the machine learning algorithm of focus.

RF algorithm has plenty of hyperparameters. On one hand, some are implemented on the overall forest level such as, the number of subjects randomly drawn from the dataset to form each tree, the choice of with or without replacement regarding the samples selection and most importantly is the number of trees in the RF. On the other hand, some hyperparameters are on a tree level, which control the shape of each tree in RF. i.e., the number of features drawn for each split, the selection of splitting rules, the depth of each tree and many others. These parameters are typically selected by the user. Consequently, creating a method to efficiently select these hyperparameters can influence the performance versus the cost of RF significantly. In addition, the recent research done in [157] emphasizes the significance of hyperparameter optimization specifically for RF parameters, as well as providing deep comparisons between numerous tuning and optimization mechanisms and software.

One of the key tuning strategies for RF, is using searching algorithms to look for optimal parameters in a pool or grid of selected ones. Search techniques differ in their way of pool or grid creation, based on the mechanism applied to choose the successful candidates forming the bag of options. Some searching strategies use all the possibilities available as candidates to be exhaustively investigated, one by one to select the optimal choice, as in grid search algorithm. However, in random search, the bag of candidates are drawn randomly from the overall existing possibilities, which is not only a precious asset for reducing the search complexity, but also studies have proved that random search produces better accuracy scores for parameters optimization comparing to grid search [158].

Random search refers to a group of searching algorithms that rely on randomness or pseudo-random generators as the core of their function. This method is also known as a Monte Carlo, a stochastic or metaheuristic algorithms. Random search is beneficial for various global optimization problems, structured or ill-structured over discrete or continuous variables [159]. Below is the algorithm describing the flow-work of a typical random search algorithm.

Algorithm Random Search Algorithm

```
Let $RF$ is the cost function to be optimized or minimized. $C$ is a candidate solution in the search-space $R^n$.
Select termination condition $TC$. i.e., specific fitness measure achieved, or max number of iterations reached, and so on.

Initialize C. $C = Random\ position \in R^n$
$For\ TC$ :
Randomly choose another position $C_{new}$ from the radius surrounding $C$ (the radius of the hypersphere surrounding C)
$if\ RF(C_{new}) < RF(C)\ then$
$C = C_{new}$
```

In practice, Scikit-learn library for Python machine learning provides a method; RandomizedSearchCV which can be provoked by creating a range for each hyperparameters subject of optimization. By calling RandomizedSearchCV method over the predefined range, random search is performed to randomly select a candidate grid of possibilities within the range, then applying K-fold cross validation technique over the created grid. For additional examples about this method, refer to [160].

B. Model-based Phase

This phase represents a clear model of the system, whether it is an actual physical model, a simulation, a knowledgebase semantically connecting the system component together, or a relational database. Based on the system model nature, the extracted nested, conditional rules from the RF are transmitted to a suitable form. i.e., in knowledge-based systems such as ontologies, the rules are converted into SPARQL [161] semantic queries, a regular SQL queries in case the system model is represented by a relational database, or in a simpler fashion use the extracted rules as a small conditional code, which runs every diagnostic window to perform the diagnosis. This phase is crucial to minimize the online diagnostic time and computational power needed, comparing to provoking the testing RF algorithm over and over for each sliding window.

C. Dynamic Rules Update Phase

In this phase, the new time-series data generated by the system for a certain number of sliding windows, is stored and used to update the originally created RF, by performing the hyperparameter tuning again to find out if any alteration of the RF parameters could reduce the size of the overall RF and increase the accuracy at the same time. The new updates selection or rejection decision is highly dependent on the accuracy of the newly tuned RF.

3. Experimental Results

In this experiment, RF is used following the steps in the data-driven flowchart in Figure 32, to generate dynamic diagnostic rules to diagnose and monitor the health of a hydraulic system tested by a hydraulic test rig. Provided in the dataset, each component condition ranges between full efficiency, reduced efficiency and close to total failure. In this experiment, for the sake of simplicity, the healthy state is represented by the full efficient cycles, and the failures are represented by the cycles where the component is close to failure, while the partial failure state is being excluded. Based on the previous fault description, there are four types of total failure in four different components to monitor; cooler total failure state, valve close to total failure, internal pump has a severe leakage and hydraulic accumulator close to total failure. Table 16 explains the definition of each fault chosen for this experiment and some example cycles that contains each fault.

Table 16 Hydraulic Test Rig Chosen Faults and Their Full Description [155]**..**

Status	Status Description	Example Cycle no.
Healthy	All components are healthy and in full efficiency mode.	1788, 1789,1790
Cooler Fault	Cooler has a total failure, and the rest of the components are fully efficient.	1,2,3
Valve Fault	Valve close to total failure, and the rest of the components are fully efficient.	1759, 1760, 1761
Pump Fault	Internal pump has severe leakage, and the rest of the components are fully efficient.	1675, 1706, 1707
Accumulator Fault	Hydraulic Accumulator close to total failure, and the rest of the components are fully efficient.	1465, 1466, 1467

The hydraulic system described in this experiment contains eleven sensor readings of three types of sensors located in different components of the hydraulic test rig. Six Pressure sensors labelled as PS1, PS2 up to PS6, four temperature sensors TS1-TS4, and finally one vibrational sensor labelled as VS1. The readings of all the eleven sensors from various cycles containing the five different statuses shown in the table above is collected in one labelled dataset of eleven features necessary to perform RF training and analysis.

As mentioned earlier, the selection of RF as the classification method in this work is done after carefully comparing the results of RF with respect to other famous classifiers. The supervised machine learning methods shown above along with RF are used to perform a multi-class classification task, to classify the hydraulic test rig faults described in Table 2. The following table, Table 3, shows the classification results after performing multi-class classification using different classifiers. It is demonstrated clearly that CART and RF have elevated accuracy compared to the rest of the approaches. However, RF is an optimization of CART, which overcomes its tendency to form overfitted relationships with the training dataset.

Table 17 RF Accuracy Results Comparison to Some Other Classifiers for Hydraulic Test Rig Fault Classification [155]..

Method	LR	LDA	KNN	CART	NB	SVM	RF
Accuracy	0.780469	0.778646	0.932031	0.989844	0.704167	0.923177	0.985198

Non-zero feature importance method is used to neglect the features with less impact of the learning process, by concentrating only on the features that contribute more to the model accuracy. The table below shows the importance of each sensor to the RF model calculated using Eq. 17.

Table 18 shows the calculated importance of each one of the 11 sensors. There are a variety of options by which these importance values are analysed and evaluated to achieve feature selection. One can pick the highest importance feature alone to represent all the features, or the highest three, highest six or just the non-zero ones to represent the whole pack. However, the most convincing approach is testing all the possibilities and making a logical accuracy versus complexity trade off. For each selected feature(s) scenario, the RF accuracies and the time complexity given $O(T \log n)$ equation are calculated, where T is the number of trees in the RF and n is the size of the input data used for training, assuming that the number of trees, T, is constant for all the feature trials. As such, the time complexity is a factor of input data size, represented by the number of features included without sacrificing much or any of the model accuracy.

Table 18 Feature Importance Calculated for Each Sensor Feature [155].

Sensor label	PS1	PS2	PS3	PS4	PS5	PS6	TS1	TS2	TS3	TS4	VS1
Importance X100 (%)	0	0	0	29.689	16.036	14.118	6.446	8.977	8.920	9.509	2.875

In Table 19, four different RF model training experiments are conducted to find out the best number of features required to train an RF on a hundred decision tree. In the first trial, the most important feature PS4 is used alone to train the RF model. The second trial used the top three important features PS4, PS5 and PS6. The third trial applied the highest six features. Finally, only the non-zero features are selected to train the RF model. For all the above four experiments, the random forest has fixed hyperparameters which were randomly chosen as one hundred trees and the maximum depth of five. Furthermore, 10-fold cross validation technique is used to compute each trial's accuracy.

Table 19 RF Accuracies Using Different Features Based on Their Importance [155].

Trial Description	No. features	RF Accuracy
No feature selection	11	0.985
Highest Feature	1	0.707
Highest Three	3	0.977
Highest Six	6	0.986
Non-zero importance	8	0.981

For this experiment, the highest six features are used for the training process since these features provided the best accuracy among all trials and shown lower time and space accuracy comparing to the training using eleven and eight features, respectively. The following figure shows how the time complexity $O(T \log n)$ and space complexity $O(n)$ for the RF are directly proportional to the number of features used. It is crucial to emphasize that the amount of accuracy sacrificed, and the added complexity tolerances are totally dependent on the system used and one's preferences. i.e., some other researchers would use the highest three features with 0.977 accuracy, if ones are willing to lose more accuracy in expense of the dramatic drop in both time and space complexities.

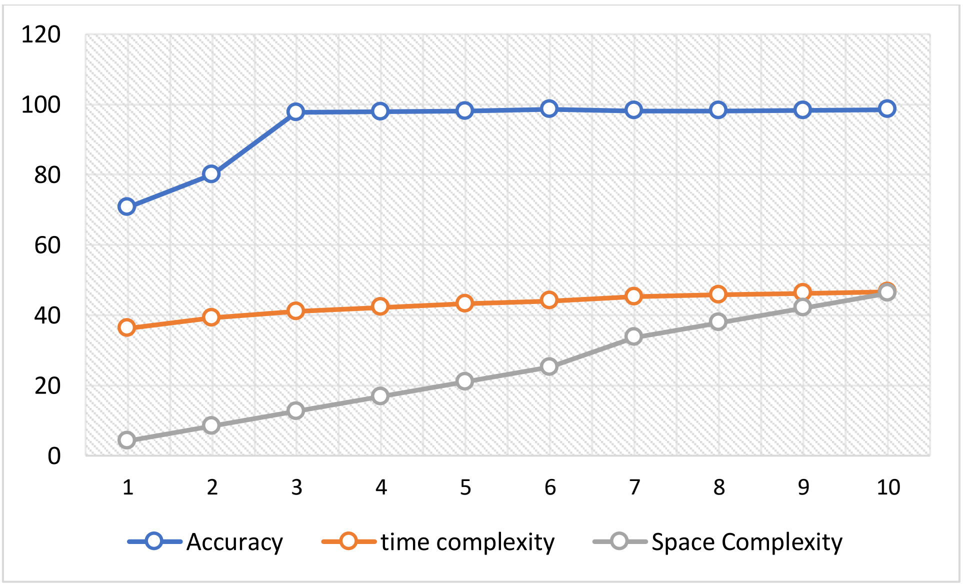


Figure 33 RF Accuracy, Time, and Space Complexities in Respect of No. of Features [155].

The next step is tuning the hyperparameters of the RF applied on the dimensionally reduced dataset of the selected six features: PS4, PS5, PS6, TS2, TS3 and TS4. The hyperparameters subject of tuning in this experiment are the number of trees in the RF, the maximum depth of the tree, minimum number of samples required to split a node and minimum number of samples required to form a leaf node. As the purpose behind RF creation in this research is to establish a set of base rules for fault diagnosis, the main hyperparameter of focus is the number of trees to ensure lessening the complexity as possible.

A random grid of hyperparameters is created by applying random search over a pre-defined range for each parameter separately. i.e., the number of trees is pre-defined to range between one and one thousand, and only a hundred possibility is selected from the range to form the random grid for this hyperparameter. Followed by RF training using one of the randomly selected pair of features at a time. The selection is validated using 3-fold cross validation to calculate the accuracy of the RF model over a particular set of hyperparameters. A hundred set of randomly selected parameters are used to create the grid, which means 300 RF model training has been successfully executed considering 3-fold cross validation over the hundred set of possibilities in the grid. Finally, the set of hyperparameters with the highest accuracy when applying 3-fold cross validation is the one selected to generate the diagnostic rules.

The RF model trained after applying feature selection with randomly chosen parameters yielded an accuracy of 0.9865 using a hundred decision trees forming the RF with maximum depth of five. However, the best hyperparameter tuned using cross validation over random search grid improved the accuracy with 0.32 to reach 0.9865 . As well as, using only 49 trees in total instead of one hundred over the same depth, which has dramatically decreased the complexity and size of the generated RF rules while increasing the accuracy and speed of the diagnosis.

The best hyperparameters selected from the grid has 49 number of estimators, minimum sample split of two, minimum leaf samples of one and maximum depth of 83. It is worth to be mentioned that the accuracy of the RF using all the best hyperparameters increased the accuracy to 0.99, but this slight rise in the accuracy is not worth it, especially when it is compared to the massive increase in the time and size of each tree in the RF, due to the large increase in the maximum depth. Figure 36 shows the one of the decision trees in the RF after feature selection and hyperparameter tuning.

Each tree in the RF can be translated into a set of nested if-else statements of rules. These rules can be executed in a distributed fashion when a proper scheduler is applied. Moreover, the formed dynamic, distributed rules from the RF can be fed inside various system models to generate a hybrid approach out of the data-driven and the model-based ones. The dynamic rules extracted from the RF can be used as they are, converted to SQL queries if the model is a relational database or a SPARQL queries if the system model is represented by a semantic knowledgebase such as ontologies. The diagnostic rules can be extracted from the RF dynamically, using a few lines of code in Python language. For the sake of simplicity, Figure 35 shows the tree in Figure 36 pruned in a way that only the positive part of the condition after the root node is remaining, connected to a series of nested if statements showing how this part of the tree is translated into clear dynamic rules. The work in [11] provided a graph-based FDD system for industrial systems using a model-based approach, based on creating a knowledge-base of the system under diagnosis, such as ontologies. Followed by manually feeding a set of static diagnostic rules created by the system expert, into the ontology, in a way that forms a causation relationship between the system sensors and the faults they lead to. In this work, we propose creating dynamic rules using RF, extracting these rules, and feeding them into the ontology instead of the expert rules that are static, unreliable, and unverifiable. Furthermore, the extracted diagnostic rules using RF can be applied in a variety of forms to fit the model expressing the system.

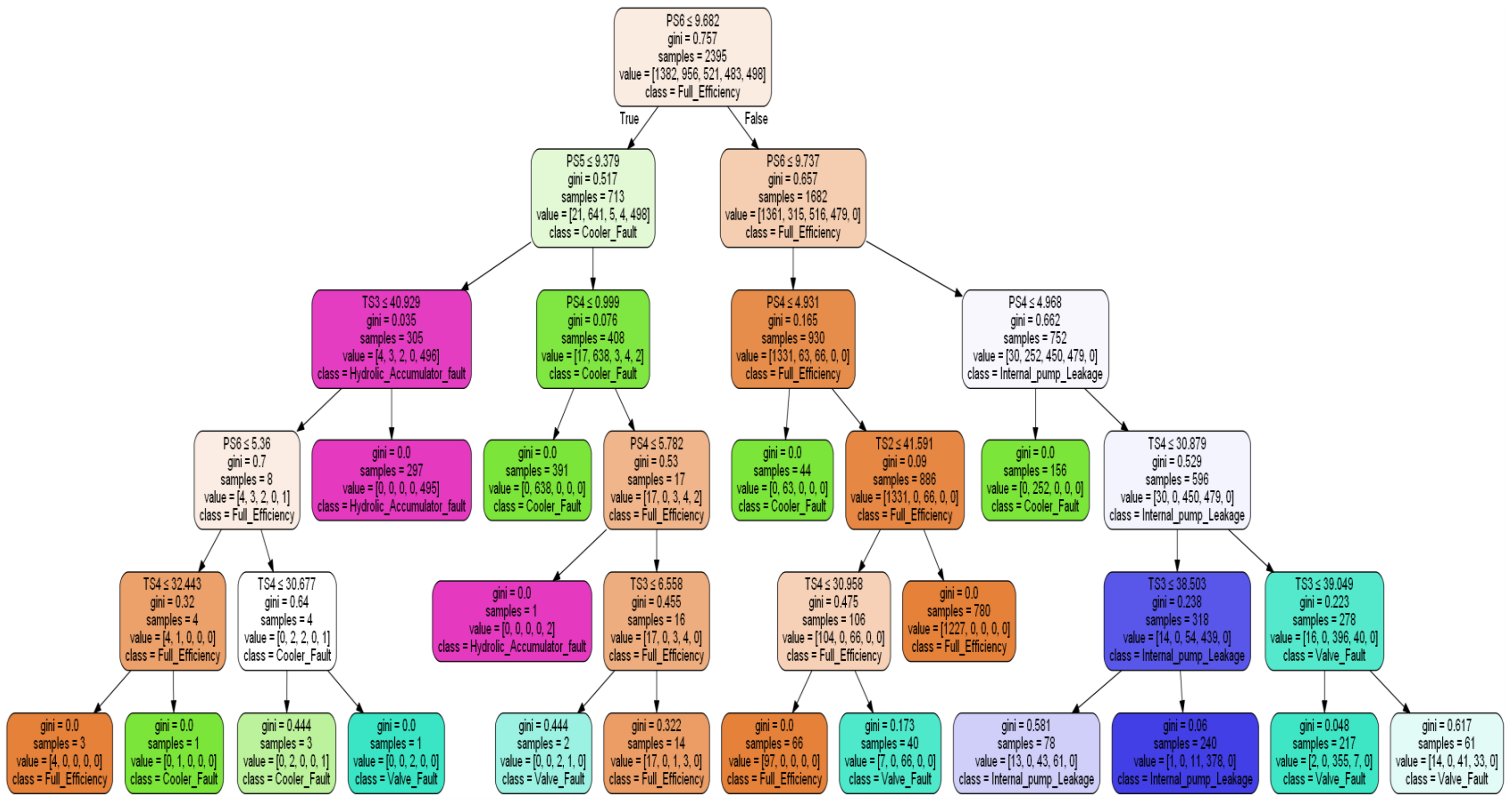
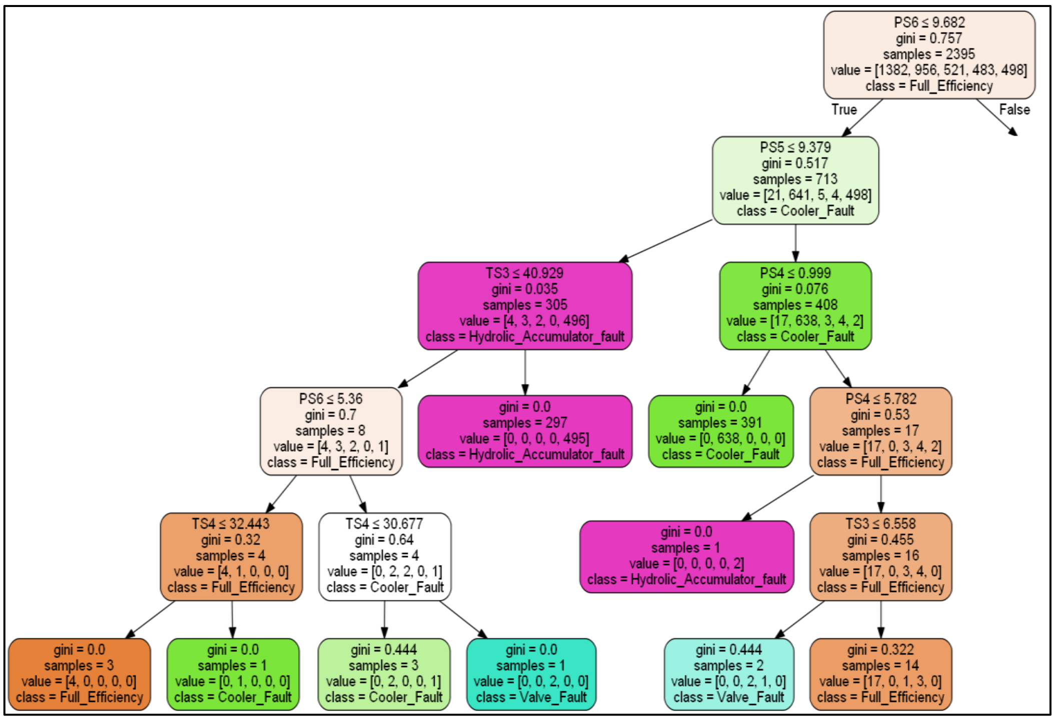

PS6 ≤ 9.682 gini = 0.757 samples = 2395 value = [1382, 956, 521, 483, 498] class = Full_Efficiency
True
False
PS5 ≤ 9.379 gini = 0.517 samples = 713 value = [21, 641, 5, 4, 498] class = Cooler_Fault
PS6 ≤ 9.737 gini = 0.657 samples = 1682 value = [1361, 315, 516, 479, 0] class = Full_Efficiency
TS3 ≤ 40.929 gini = 0.035 samples = 305 value = [4, 3, 2, 0, 496] class = Hydrolic_Accumulator_fault
PS4 ≤ 0.999 gini = 0.076 samples = 408 value = [17, 638, 3, 4, 2] class = Cooler_Fault
PS4 ≤ 4.931 gini = 0.165 samples = 930 value = [1331, 63, 66, 0, 0] class = Full_Efficiency
PS4 ≤ 4.968 gini = 0.662 samples = 752 value = [30, 252, 450, 479, 0] class = Internal_pump_Leakage
PS6 ≤ 5.36 gini = 0.7 samples = 8 value = [4, 3, 2, 0, 1] class = Full_Efficiency
gini = 0.0 samples = 297 value = [0, 0, 0, 0, 495] class = Hydrolic_Accumulator_fault
gini = 0.0 samples = 391 value = [0, 638, 0, 0, 0] class = Cooler_Fault
PS4 ≤ 5.782 gini = 0.53 samples = 17 value = [17, 0, 3, 4, 2] class = Full_Efficiency
gini = 0.0 samples = 44 value = [0, 63, 0, 0, 0] class = Cooler_Fault
TS2 ≤ 41.591 gini = 0.09 samples = 886 value = [1331, 0, 66, 0, 0] class = Full_Efficiency
gini = 0.0 samples = 156 value = [0, 252, 0, 0, 0] class = Cooler_Fault
TS4 ≤ 30.879 gini = 0.529 samples = 596 value = [30, 0, 450, 479, 0] class = Internal_pump_Leakage
TS4 ≤ 32.443 gini = 0.32 samples = 4 value = [4, 1, 0, 0, 0] class = Full_Efficiency
TS4 ≤ 30.677 gini = 0.64 samples = 4 value = [0, 2, 2, 0, 1] class = Cooler_Fault
gini = 0.0 samples = 1 value = [0, 0, 0, 0, 2] class = Hydrolic_Accumulator_fault
TS3 ≤ 6.558 gini = 0.455 samples = 16 value = [17, 0, 3, 4, 0] class = Full_Efficiency
TS4 ≤ 30.958 gini = 0.475 samples = 106 value = [104, 0, 66, 0, 0] class = Full_Efficiency
gini = 0.0 samples = 780 value = [1227, 0, 0, 0, 0] class = Full_Efficiency
TS3 ≤ 38.503 gini = 0.238 samples = 318 value = [14, 0, 54, 439, 0] class = Internal_pump_Leakage
TS3 ≤ 39.049 gini = 0.223 samples = 278 value = [16, 0, 396, 40, 0] class = Valve_Fault
gini = 0.0 samples = 3 value = [4, 0, 0, 0, 0] class = Full_Efficiency
gini = 0.0 samples = 1 value = [0, 1, 0, 0, 0] class = Cooler_Fault
gini = 0.444 samples = 3 value = [0, 2, 0, 0, 1] class = Cooler_Fault
gini = 0.0 samples = 1 value = [0, 0, 2, 0, 0] class = Valve_Fault
gini = 0.444 samples = 2 value = [0, 0, 2, 1, 0] class = Valve_Fault
gini = 0.322 samples = 14 value = [17, 0, 1, 3, 0] class = Full_Efficiency
gini = 0.0 samples = 66 value = [97, 0, 0, 0, 0] class = Full_Efficiency
gini = 0.173 samples = 40 value = [7, 0, 66, 0, 0] class = Valve_Fault
gini = 0.581 samples = 78 value = [13, 0, 43, 61, 0] class = Internal_pump_Leakage
gini = 0.06 samples = 240 value = [1, 0, 11, 378, 0] class = Internal_pump_Leakage
gini = 0.048 samples = 217 value = [2, 0, 355, 7, 0] class = Valve_Fault
gini = 0.617 samples = 61 value = [14, 0, 41, 33, 0] class = Valve_Fault

Figure 34 One of the Decision Trees after Feature Selection and Hyperparameter Tuning [155].



```
if PS6 <= 9.682436466217041:
  if PS5 <= 9.378981590270996:
    if TS3 <= 40.929338455200195:
      if PS6 <= 5.359611511230469:
        if TS4 <= 32.44318962097168:
          return [[4. 0. 0. 0. 0.]]
        else:  # if TS4 > 32.44318962097168
          return [[0. 1. 0. 0. 0.]]
      else:  # if PS6 > 5.359611511230469
        if TS4 <= 30.676508903503418:
            return [[0. 2. 0. 0. 1.]]
        else:  # if TS4 > 30.676508903503418
          return [[0. 0. 2. 0. 0.]]
    else:  # if TS3 > 40.929338455200195
      return [[0.    0.    0.    0. 495.]]
  else:  # if PS5 > 9.378981590270996
    if PS4 <= 0.9992818832397461:
      return [[0. 638.    0.    0.    0.]]
    else:  # if PS4 > 0.9992818832397461
      if PS4 <= 5.781721115112305:
        return [[0. 0. 0. 0. 2.]]
      else:  # if PS4 > 5.781721115112305
        if TS3 <= 6.558143854141235:
          return [[0. 0. 2. 1. 0.]]
        else:  # if TS3 > 6.558143854141235
          return [[17.  0.  1.  3.  0.]]
```

Figure 35 Diagnostic Rules Extraction from Parts of a Tree in the RF [155].

Figure 37 showcases the extracted rules from the optimized, hyperparameter tuned RF, and how these rules are transformed to various forms and types to match the nature of the system model. As mentioned before in this chapter, the diagnostic rules can be translated into SQL queries in case a relational database is the system model, or SPARQL queries if a semantic knowledge-base, such as ontologies [12] are used to represent the system. The rules extracted from each tree may be scheduled separately, or all together with the trees forming the RF.

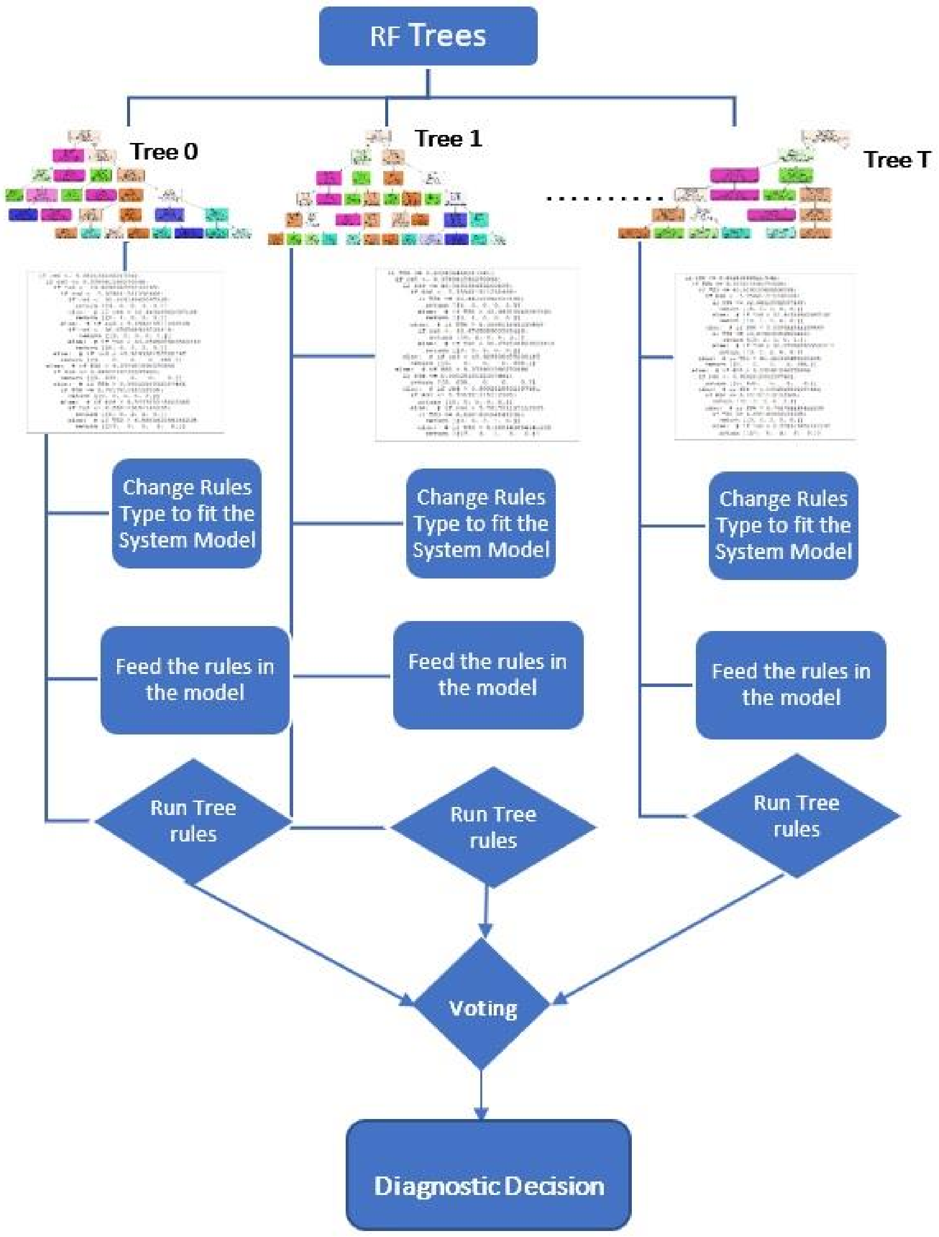

RF Trees
Tree 0
Tree 1
Tree T
Change Rules Type to fit the System Model
Change Rules Type to fit the System Model
Change Rules Type to fit the System Model
Feed the rules in the model
Feed the rules in the model
Feed the rules in the model
Run Tree rules
Run Tree rules
Run Tree rules
Voting
Diagnostic Decision

Figure 37 A Hybrid RF Approach Between Data-driven and Model-based Approaches [155].

Chapter 7: Conclusions and Future Work

This work demonstrated a full research and experimental analysis regarding FDD in hydraulic systems extracted from a hydraulic test rig. We made sure to tackle all the steps in data-driven FDD methods starting from feature engineering to establishing innovative intelligent models.

The contributions of this dissertation are summarized by the three main aspects, in which each one of them contributes to solving many major research problems and challenges connected to FDD in hydraulic systems. Below the conclusion and future adaptations for each contribution provided in this thesis.

A. R*k*SE Conclusions

In this work Recursive *k*-means Silhouette Elimination (R*k*SE) is proposed, tested, and validated. RkSE is a new unsupervised feature selection algorithm with a novel idea of applying *k*-means clustering and silhouette measure beyond its ordinary backward or forward selection style, but as a recursive acquisition approach with the application of a user-defined threshold, that plays a major role in the uniqueness of this approach.
R*k*SE can be applied for various applications. However, in this work R*k*SE is applied to reduce dimensionality in univariate and multivariate sensor time-series datasets. For future work, R*k*SE will be applied on numerous datasets of different applications and fields such as, genetics, microarrays, text, images and so on. Furthermore, this work focuses on the performance of R*k*SE on univariate and multi-variate time-series datasets. i.e., sensor data.

R*k*SE is evaluated on a real measured data of a hydraulic test rig and validated to solve a classical fault classification problem in two different experiments: (1) R*k*SE is used in a univariate time-series dataset, to classify sensor faults in a sliding window format. When R*k*SE is applied on sliding windows, its function indicates the ability to select the best quality time points to represent the whole window. In other words, R*k*SE can successfully be used as a signal compression method when, applied to time-series data in a window format. (2) In a multivariate time-series data without the application of sliding windows, R*k*SE is applied to select the best modality of features to represent the whole dataset.
In addition, this work structures a comprehensive review survey of feature selection algorithms based on k-means clustering, existing in the literature. Which yielded the creation of a new taxonomy for k-means clustering feature selection methods.
The results of the two extensive experiments proves the uniqueness and efficiency of R*k*SE method for time-series datasets in a univariate or multivariate format.
To sum up, R*k*SE represents an iterative, unsupervised, silhouette-based, k-means clustering feature selection algorithm. Although, RkSE has plenty of advantaged and contributions that exceed the methods R*k*SE in related-work section, it also has limitations that we hope to eliminate in the future. The following points show the strengths and limitations of R*k*SE method for feature selection based on its functionality and workflow.
R*k*SE advantages are the following:

- Can Create a model representation of feature dependencies (Iterative algorithm advantage).
- Feature selection and clustering are made concurrently in one single operation. (Iterative Advantage)
- Unsupervised Feature Selection method, no labels required.
- Simple, robust, and low computational and time costs: Due to the exclusive application of *k*-means and silhouette criteria, which provides simplicity and reduce time and computational complexity
- User-interactive: allows the user to choose the value of the threshold which give the freedom to select the best number of features which provide the option to loosen the accuracy, or the complexity constrains.
- Introducing a new concept of using *k*-means and silhouette measure in a recursive manner, instead of the common forward and backward approaches.

The limitations of R*k*SE are as follows:

- Prone to overfitting. (iterative algorithms disadvantage)

- The choice of the threshold if not chosen probably can drastically affect the quality of the selection, which cannot be guaranteed since σ is user selected.
- The accuracy is a little compromised because the algorithm focuses on the relationship between the feature and the cluster, rather than the relationships between features.

B. Hydraulic Systems FDD Model: A Joint Approach Between a LSTM Autoencoder Detector and Classification Diagnosis:

In this section a two-staged FDD approach is proposed. Where the detection and diagnosis are separated into two stages to guarantee detecting rare occurrences and events. The detection process is represented by a LSTM autoencoder that learns only from healthy observations, in an attempt to reconstruct the healthy version of the given sequences, sensor window readings or multi-sensors readings. Followed by the comparison between the given sequences and their healthy reconstructed version, to measure the deviation from the healthy state and the given sequences, which is vital to detect the existence of faults and malfunctions when this deviation exceeds a certain, learned threshold. The fault diagnosis is represented by a classification process that can be a ML or DL algorithm, which is trained using only the faulty observations captured by the detection stage using LSTM autoencoder.
The proposed approach is beyond the state-of-the-art methods by the following:

- The proposed method is applied into two entirely different experiments, with different data pre-processing, acquisition and structuring, different DL algorithmic designs, and above all to detect two different fault types: sensor faults and component faults. The methods proposed in the literature only focuses on one fault type, either component faults or sensor faults. However, it is rarely seen that any work shows comprehension in detecting or diagnosing different fault types at once.

- In the detection phase using LSTM autoencoder, some changes are made from the existing related work. The most important addition is using the sequence difference calculated by subtracting the Pearson's autocorrelation from one. The detection results using the complement of Pearson's autocorrelation comparing to the traditional signal difference measure applied in the state-of-the-art research is proved experimentally. In experiment one to detect sensor faults, the detection accuracy using signal difference is observed as 62%, comparing to the detection accuracy when applying our proposed measurement of signal difference, the detection accuracy is observed as 71%. Moreover, in experiment two for component fault detection, the accuracies observed using traditional signal difference, and the proposed one are 69% and 71%, respectively. The results of the detection phase in the two conducted experiments prove the superiority of the proposed signal difference measurement comparing to the traditional subtraction of signals to provide signal difference.

- Investigating various feature engineering approaches and pairing them with numerous ML and DL methods, to determine the most suitable feature engineering method to classifiers of different functionality and design. Furthermore, this pairing gives the opportunity to see how each classifier reacts with different feature engineering methods of different procedure, which would help future work researchers to select the best match pair or avoid the worst pair for both data structures, windows univariate or no window multi-variate. For example, in experiment one when dealing with sensor faults in a sliding window data structure, it was noticeable that the chosen time-domain features shown the highest diagnosis accuracy of almost all the classifiers. i.e., LDA, KNN, CART, RF and LSTM. Although the mean accuracy of all classifiers using PCA was computed the highest of 82%, which is justified by the consistency PCA shows with all the classifiers regardless their functionality. However, time-domain extracted features show extremely low diagnosis accuracies when applied to some classifiers such as LR and NB with the detection accuracies of only 24.25% and 48.21%, respectively. Which explains why the time-domain extracted features is not the highest mean accuracy even though it provides the highest accuracy to the majority of the supervised methods. On the other hand, in experiment two when component faults were classified using multi-variate sensor's readings without the application of sliding windows, FI showed the highest diagnosis accuracy for all the ML classifiers, and PCA shows the highest diagnosis accuracy when combined to DL such as, CNN and LSTM.

- In the related work, the diagnosis phase is represented by some chosen type of classifiers combined with a chosen set of features, without any analysis or investigation in respect of other classifiers or features. In this work, after careful experimental observations and calculations, the appropriate features and their suitable classifier is used to represent the diagnosis phase for our algorithm. In experiment one (univariate sliding window structure) the diagnosis phase is chosen using the time-domain extracted features combined with either CART or LSTM classifiers with the diagnosis accuracies of 99.51% and 96.84%. However, in experiment two, when dealing with multi-variate features without the application of sliding windows, FI combined with almost all ML classifiers showed extremely high accuracies exceeding 98%. Thus, RF combined with FI is the best combo used to perform multi-variate diagnosis, especially when FI can be done implicitly during the RF training stage based on FI nature, which can help in reducing time and computational complexities.

In conclusion, the proposed approach is used for the first time in the field of industry 4.0 especially when applied to hydraulic rigs. Although, the proposed work in this chapter has multiple changes and valuable improvements from the state-of-the art, there is always a place for improvements and further expansions. For future work, it can be a good challenge to improve experiment one, by designing a LSTM autoencoder that can learn multiple sequences of multiple windows that belongs to different sensors at a time applying one LSTM autoencoding model. For example, in experiment one only one sensor at a time (PS1) is used in a sliding window format to train the LSTM autoencoder to predict/reconstruct the healthy window of the given PS1 window of size 60. In the current approach we must train the LSTM autoencoder for each sensor separately to learn how to reconstruct the healthy window of each. However, A multi-variate approach, multi-sequences, multi-sensor deep neural network design would be a sophisticated approach in the future, where maybe a number of LSTM autoencoder batches can be connected together sequentially or in parallel so each can train on different sensor window sequences. Furthermore, the feature engineering methods applied to the diagnosis phase are all in time-domain. Investigating frequency-domain or the combination of time and frequency domains, such as applying Wavelet Coefficient Packer Decomposition (WPD) would add different perspective for the future of FDD in mechanical equipment's.

C. A Hybrid Approach to Generate Dynamic Diagnostic Rules Based on RF:

In this work, the architecture of a hybrid FDD method, between model-based and data-driven approaches to achieve FDD for component faults, is introduced. In this hybrid method, the data-driven part is represented by an optimized and hyperparameter tuned RF, in order to generate dynamic, diagnostic graphs that are later converted into a set of diagnostic rules and fed into a pre-defined system diagnostic model, acting as the model-based part of the proposed FDD system. The proposed approach provides a dynamic solution, unlike other model-based FDD approaches. Additionally, there is the option to apply distributed computing to the diagnostic graphs and extracted rules, which can reduce time and resource complexity compared to traditional data-driven approaches. Moreover, the proposed method introduces a new methodology to approach RF in a model-based fashion, beyond its exclusive, ordinary application as a data-driven approach.

The data-driven part of this system is experimentally applied and analyzed using multivariate time-series sensor data collected from an actual hydraulic test rig. The applied data-driven part includes the RF creation, RF feature selection using non-zero feature importance, and RF pruning and hyperparameter tuning using three-fold cross validation on a grid of variables, selected using random search. Furthermore, the diagnostic rules in the form of nested if-else statements are practically extracted from RF as the diagnostic graph of this approach. The extracted rules can be converted into various forms and shapes depending on the nature of the system model that is the subject of integration.

This work has successfully provided an extension and development of the model-based FDD approach introduced in [11], where the previous system is domain-specific, and can only be applied to the model described in the ontology. It also contains rigid knowledge preserved in the ontology as a set of semantic rules, but later translated into static, domain-specific, application-specific set of diagnostic rules with constrained reliability to the system's expert. On the other hand, our proposed method provided dynamic and reliable diagnostic rules, with the combined advantages of both the data-driven and model-based approaches.

The proposed FDD system offers a vast number of advantages and new insights. However, there is always room for improvements. Thus, some additional work and further modifications of the proposed system can be applied in the future. In this work, the traditional RF algorithm is applied to serve as the dynamic rule generator in both the offline learning and the online update stage, using the newly arrived sensor reading. Instead, considering online RF methods in the first place, such as Mondrian forests [162] or online incremental RF (described in [163]), may reduce the training and update time. Moreover, in future work a full application of this approach will be introduced and examined in a practical study, where the extracted rules are converted into SPARQL queries to fit the ontology designed of the chosen system. Furthermore, a proper scheduler will be chosen to demonstrate the possibility of distributed computing using the extracted diagnostic rules at run-time.

References

[1] R.-E. Precup, P. Angelov, B. S. J. Costa, and M. Sayed-Mouchaweh, "An overview on fault diagnosis and nature-inspired optimal control of industrial process applications," *Computers in Industry*, vol. 74, pp. 75–94, Dec. 2015, doi: 10.1016/j.compind.2015.03.001.

[2] P. Wang and C. Guo, "Based on the coal mine ' s essential safety management system of safety accident cause analysis," 2013.

[3] G. H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, New Brunswick, NJ, USA, Jul. 1994, pp. 121–129, Accessed: May 20, 2020. [Online].

[4] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, no. null, pp. 1157–1182, Mar. 2003.

[5] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis*, vol. 1, no. 3, pp. 131–156, Dec. 1997, doi: 10.3233/IDA-1997-1302.

[6] H. Fröhlich, O. Chapelle, and B. Schölkopf, "Feature selection for support vector machines using genetic algorithms," *Int. J. Artif. Intell. Tools*, vol. 13, no. 04, pp. 791–800, Dec. 2004, doi: 10.1142/S0218213004001818.

[7] S.-W. Lin, K.-C. Ying, C.-Y. Lee, and Z.-J. Lee, "An intelligent algorithm with feature selection and decision rules applied to anomaly intrusion detection," *Applied Soft Computing*, vol. 12, no. 10, pp. 3285–3290, Oct. 2012, doi: 10.1016/j.asoc.2012.05.004.

[8] D. K. Bhattacharyya and J. K. Kalita, *Network Anomaly Detection: A Machine Learning Perspective*. Chapman & Hall/CRC, 2013.

[9] P. Arabie and L. J. Hubert, "An overview of combinatorial data analysis," in *Clustering and Classification*, 0 vols., WORLD SCIENTIFIC, 1996, pp. 5–63.

[10] S. Kasim, S. Deris, and R. M. Othman, "Multi-stage filtering for improving confidence level and determining dominant clusters in clustering algorithms of gene expression data," *Comput. Biol. Med.*, vol. 43, no. 9, pp. 1120–1133, Sep. 2013, doi: 10.1016/j.compbiomed.2013.05.011.

[11] A. Mallak, A. Behravan, C. Weber, M. Fathi, and R. Obermaisser, "A Graph-Based Sensor Fault Detection and Diagnosis for Demand-Controlled Ventilation Systems Extracted from a Semantic Ontology," in *2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES)*, Jun. 2018, pp. 000377–000382, doi: 10.1109/INES.2018.8523895.

[12] "Definition of ONTOLOGY." https://www.merriam-webster.com/dictionary/ontology (accessed Aug. 11, 2019).

[13] A. Rojko, "Industry 4.0 Concept: Background and Overview," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 11, no. 5, Art. no. 5, Jul. 2017.

[14] R. Doddannavar, A. Barnard, and J. Ganesh, *Practical Hydraulic Systems: Operation and Troubleshooting for Engineers and Technicians*. Elsevier, 2005.

[15] "Pascal's principle | Definition, Example, & Facts," *Encyclopedia Britannica*. https://www.britannica.com/science/Pascals-principle (accessed Oct. 22, 2020).

[16] "What Is a Hydraulic System? Definition, Design, and Components | Convergence Training," *Convergence Training Blog*, Jul. 23, 2017. https://www.convergencetraining.com/blog/what-is-a-hydraulic-system-definition-design-and-components (accessed Oct. 22, 2020).

[17] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control Engineering Practice*, vol. 5, no. 5, pp. 709–719, May 1997, doi: 10.1016/S0967-0661(97)00053-1.

[18] S. K. Kanev, "Robust fault-tolerant control," 2004.

[19] K. Ni *et al.*, "Sensor network data fault types," *ACM Trans. Sen. Netw.*, vol. 5, no. 3, p. 25:1-25:29, Jun. 2009, doi: 10.1145/1525856.1525863.

[20] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009, doi: 10.1145/1541880.1541882.

[21] V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part I: Quantitative model-based methods," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 293–311, Mar. 2003, doi: 10.1016/S0098-1354(02)00160-6.

[22] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies," *Computers & Chemical Engineering*, vol. 27, pp. 313–326, 2003, doi: 10.1016/S0098-1354(02)00161-8.

[23] C. Skliros, M. E. Miguez, A. Fakhre, and I. Jennions, "A review of model based and data-driven methods targeting hardware systems diagnostics," *Diagnostyka*, vol. 20, no. 1, pp. 3–21, Nov. 2018, doi: 10.29354/diag/99603.

[24] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, "A review of process fault detection and diagnosis: Part III: Process history based methods," *Computers & Chemical Engineering*, vol. 27, no. 3, pp. 327–346, Mar. 2003, doi: 10.1016/S0098-1354(02)00162-X.

[25] A. Poongodai and S. Bhuvaneswari, "AI Technique in Diagnostics and Prognostics," *IJCA Proceedings on National Conference on Future Computing 2013*, vol. NCFC, no. 1, pp. 1–4, Sep. 2013.

[26] "Machine Learning textbook." http://www.cs.cmu.edu/~tom/mlbook.html (accessed Jul. 09, 2020).

[27] J. Miao and L. Niu, "A Survey on Feature Selection," *Procedia Computer Science*, vol. 91, pp. 919–926, Jan. 2016, doi: 10.1016/j.procs.2016.07.111.

[28] J. C. Ang, A. Mirzal, H. Haron, and H. N. A. Hamed, "Supervised, Unsupervised, and Semi-Supervised Feature Selection: A Review on Gene Selection," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13, no. 5, pp. 971–989, Sep. 2016, doi: 10.1109/TCBB.2015.2478454.

[29] Z. Zhao and H. Liu, "Semi-supervised Feature Selection via Spectral Analysis," in *Proceedings of the 2007 SIAM International Conference on Data Mining*, 0 vols., Society for Industrial and Applied Mathematics, 2007, pp. 641–646.

[30] Q. Cheng, H. Zhou, and J. Cheng, "The Fisher-Markov Selector: Fast Selecting Maximally Separable Feature Subset for Multiclass Classification with Applications to High-Dimensional Data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 6, pp. 1217–1233, Jun. 2011, doi: 10.1109/TPAMI.2010.195.

[31] J. G. Dy and C. E. Brodley, "Feature Selection for Unsupervised Learning," *J. Mach. Learn. Res.*, vol. 5, pp. 845–889, Dec. 2004.

[32] K. Kira and L. A. Rendell, "A Practical Approach to Feature Selection," in *Machine Learning Proceedings 1992*, D. Sleeman and P. Edwards, Eds. San Francisco (CA): Morgan Kaufmann, 1992, pp. 249–256.

[33] L. E. Raileanu and K. Stoffel, "Theoretical Comparison between the Gini Index and Information Gain Criteria," *Annals of Mathematics and Artificial Intelligence*, vol. 41, no. 1, pp. 77–93, May 2004, doi: 10.1023/B:AMAI.0000018580.96245.c6.

[34] C. Ding and H. Peng, “Minimum redundancy feature selection from microarray gene expression data,” in *Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003*, Aug. 2003, pp. 523–528, doi: 10.1109/CSB.2003.1227396.

[35] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene Selection for Cancer Classification using Support Vector Machines,” *Machine Learning*, vol. 46, no. 1, pp. 389–422, Jan. 2002, doi: 10.1023/A:1012487302797.

[36] S. Solorio-Fernández, J. A. Carrasco-Ochoa, and J. Fco. Martínez-Trinidad, “A review of unsupervised feature selection methods,” *Artif Intell Rev*, vol. 53, no. 2, pp. 907–948, Feb. 2020, doi: 10.1007/s10462-019-09682-y.

[37] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” presented at the Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, 1967, Accessed: Jun. 06, 2020. [Online]. Available: https://projecteuclid.org/euclid.bsmsp/1200512992.

[38] C. Yuan and H. Yang, “Research on K-Value Selection Method of K-Means Clustering Algorithm,” *J — Multidisciplinary Scientific Journal*, vol. 2, no. 2, Art. no. 2, Jun. 2019, doi: 10.3390/j2020016.

[39] A. Singh, A. Yadav, A. E. Block, A. Rana, E. Block, and G. Floor, *K-means with Three different Distance Metrics*. .

[40] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, Nov. 1987, doi: 10.1016/0377-0427(87)90125-7.

[41] S. H. Walker and D. B. Duncan, “Estimation of the Probability of an Event as a Function of Several Independent Variables,” *Biometrika*, vol. 54, no. 1/2, pp. 167–179, 1967, doi: 10.2307/2333860.

[42] J. Tolles and W. J. Meurer, “Logistic Regression: Relating Patient Characteristics to Outcomes,” *JAMA*, vol. 316, no. 5, pp. 533–534, Aug. 2016, doi: 10.1001/jama.2016.7653.

[43] J. Joyce, “Bayes’ Theorem,” Jun. 2003, Accessed: Jul. 17, 2020. [Online]. Available: https://plato.stanford.edu/archives/spr2019/entries/bayes-theorem/.

[44] N. S. Altman, “An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,” *The American Statistician*, vol. 46, no. 3, pp. 175–185, Aug. 1992, doi: 10.1080/00031305.1992.10475879.

[45] R. A. Fisher, “The Use of Multiple Measurements in Taxonomic Problems,” *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936, doi: 10.1111/j.1469-1809.1936.tb02137.x.

[46] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach Learn*, vol. 20, no. 3, pp. 273–297, Sep. 1995, doi: 10.1007/BF00994018.

[47] L. Rokach and O. Maimon, *Data mining with decision trees: theory and applications*, Second edition. Hackensack, New Jersey: World Scientific, 2015.

[48] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, “Classification and Regression Trees,” 1983, doi: 10.2307/2530946.

[49] S. Shalev-Shwartz and S. Ben-David, “Understanding Machine Learning: From Theory to Algorithms,” 2014.

[50] B. Anderson, *Pattern Recognition: An introduction*. Scientific e-Resources, 2019.

[51] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*, 2nd ed. New York: Springer-Verlag, 2009.

[52] L. Breiman, “Bagging predictors,” *Mach Learn*, vol. 24, no. 2, pp. 123–140, Aug. 1996, doi: 10.1007/BF00058655.

[53] Tin Kam Ho, “Random decision forests,” in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Aug. 1995, vol. 1, pp. 278–282 vol.1, doi: 10.1109/ICDAR.1995.598994.

[54] “RANDOM FORESTS Trademark of MINITAB, LLC - Registration Number 3185828 - Serial Number 78642027 :: Justia Trademarks.” http://trademarks.justia.com/786/42/random-78642027.html (accessed Jan. 18, 2020).

[55] Tin Kam Ho, “The random subspace method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, Aug. 1998, doi: 10.1109/34.709601.

[56] R. Bryll, R. Gutierrez-Osuna, and F. Quek, “Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets,” *Pattern Recognition*, vol. 36, no. 6, pp. 1291–1302, Jun. 2003, doi: 10.1016/S0031-3203(02)00121-8.

[57] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: 10.1007/BF02478259.

[58] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," *Psychological Review*, pp. 65–386, 1958.

[59] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception (Vol. 2): computation, learning, architectures*, USA: Harcourt Brace & Co., 1992, pp. 65–93.

[60] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Madison, WI, USA, Jun. 2010, pp. 807–814, Accessed: Aug. 23, 2020. [Online].

[61] "Hyperbolic functions - Encyclopedia of Mathematics." https://encyclopediaofmath.org/index.php?title=Hyperbolic_functions (accessed Aug. 23, 2020).

[62] S. International Workshop on Artificial Neural Networks (1995 : Torremolinos, *From natural to artificial neural computation : International Workshop on Artificial Neural Networks, Malaga-Torremolinos, Spain, June 7-9, 1995 : proceedings*. Berlin ; New York : Springer-Verlag, 1995.

[63] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[64] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," *arXiv:1710.05941 [cs]*, Oct. 2017, Accessed: Aug. 23, 2020. [Online]. Available: http://arxiv.org/abs/1710.05941.

[65] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*. Cambridge/Mass.: MIT Press, 1969.

[66] ICTperspectives | Taliawebs, "Introducing Deep learning with Matlab," Accessed: Jan. 12, 2021. [Online]. Available: https://www.slideshare.net/ICTperspectives/introducing-deep-learning-with-matlab.

[67] L. Medsker, L. C. Jain, and L. C. Jain, *Recurrent Neural Networks : Design and Applications*. CRC Press, 1999.

[68] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[69] S. Narayan, "The generalized sigmoid activation function: Competitive supervised learning," *Information Sciences*, vol. 99, no. 1, pp. 69–82, Jun. 1997, doi: 10.1016/S0020-0255(96)00200-9.

[70] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *arXiv:1409.3215 [cs]*, Dec. 2014, Accessed: Sep. 09, 2020. [Online]. Available: http://arxiv.org/abs/1409.3215.

[71] "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, MITP, 1987, pp. 318–362.

[72] K. P. F.R.S, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, Nov. 1901, doi: 10.1080/14786440109462720.

[73] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding variable importances in forests of randomized trees," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 431–439.

[74] "UCI Machine Learning Repository: Citation Policy." https://archive.ics.uci.edu/ml/citation_policy.html (accessed Feb. 04, 2020).

[75] N. Helwig, E. Pignanelli, and A. Schütze, "D8.1 - Detecting and Compensating Sensor Faults in a Hydraulic Condition Monitoring System," *Proceedings SENSOR 2015*, pp. 641–646, May 2015, doi: http://dx.doi.org/10.5162/sensor2015/D8.1.

[76] N. Helwig, E. Pignanelli, and A. Schütze, "Condition monitoring of a complex hydraulic system using multivariate statistics," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, May 2015, pp. 210–215, doi: 10.1109/I2MTC.2015.7151267.

[77] T. Schneider, N. Helwig, and A. Schütze, "Automatic feature extraction and selection for classification of cyclical time series data," *tm - Technisches Messen*, vol. 84, no. 3, pp. 198–206, 2017, doi: 10.1515/teme-2016-0072.

[78] A. Mallak and M. Fathi, "Sensor and Component Fault Detection and Diagnosis for Hydraulic Machinery Integrating LSTM Autoencoder Detector and Diagnostic Classifiers," *MDPI Sensors*, vol. 21, no. 2, Art. no. 2, Jan. 2021, doi: 10.3390/s21020433.

[79] J. M. Griffin, A. J. Doberti, V. Hernández, N. A. Miranda, and M. A. Vélez, "Multiple classification of the force and acceleration signals extracted during multiple machine processes: part 1 intelligent classification from an anomaly perspective," *Int J Adv Manuf Technol*, vol. 93, no. 1, pp. 811–823, Oct. 2017, doi: 10.1007/s00170-017-0320-3.

[80] A. Krishnakumari, A. Elayaperumal, M. Saravanan, and C. Arvindan, "Fault diagnostics of spur gear using decision tree and fuzzy classifier," *Int J Adv Manuf Technol*, vol. 89, no. 9, pp. 3487–3494, Apr. 2017, doi: 10.1007/s00170-016-9307-8.

[81] G. Li *et al.*, "An improved decision tree-based fault diagnosis method for practical variable refrigerant flow system using virtual sensor-based fault indicators," *Applied Thermal Engineering*, vol. 129, pp. 1292–1303, Jan. 2018, doi: 10.1016/j.applthermaleng.2017.10.013.

[82] P. Santos, J. Maudes, and A. Bustillo, "Identifying maximum imbalance in datasets for fault diagnosis of gearboxes," *J Intell Manuf*, vol. 29, no. 2, pp. 333–351, Feb. 2018, doi: 10.1007/s10845-015-1110-0.

[83] Z. Noshad *et al.*, "Fault Detection in Wireless Sensor Networks through the Random Forest Classifier," *Sensors*, vol. 19, no. 7, p. 1568, Jan. 2019, doi: 10.3390/s19071568.

[84] S. Zidi, T. Moulahi, and B. Alaya, "Fault Detection in Wireless Sensor Networks Through SVM Classifier," *IEEE Sensors Journal*, 2018, doi: 10.1109/JSEN.2017.2771226.

[85] S. Lee, W. Park, and S. Jung, "Fault Detection of Aircraft System with Random Forest Algorithm and Similarity Measure," *The Scientific World Journal*, 2014. https://www.hindawi.com/journals/tswj/2014/727359/ (accessed Aug. 26, 2019).

[86] Z. Wang, Q. Zhang, J. Xiong, M. Xiao, G. Sun, and J. He, "Fault Diagnosis of a Rolling Bearing Using Wavelet Packet Denoising and Random Forests," *IEEE Sensors Journal*, vol. 17, no. 17, pp. 5581–5588, Sep. 2017, doi: 10.1109/JSEN.2017.2726011.

[87] B.-S. Yang, X. Di, and T. Han, "Random forests classifier for machine fault diagnosis," *J Mech Sci Technol*, vol. 22, no. 9, pp. 1716–1725, Sep. 2008, doi: 10.1007/s12206-008-0603-6.

[88] D. Wu, C. Jennings, J. Terpenny, and S. Kumara, "Cloud-based machine learning for predictive analytics: Tool wear prediction in milling," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec. 2016, pp. 2062–2069, doi: 10.1109/BigData.2016.7840831.

[89] M. Canizo, E. Onieva, A. Conde, S. Charramendieta, and S. Trujillo, "Real-time predictive maintenance for wind turbines using Big Data frameworks," in *2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, Jun. 2017, pp. 70–77, doi: 10.1109/ICPHM.2017.7998308.

[90] C. Li, R.-V. Sanchez, G. Zurita, M. Cerrada, D. Cabrera, and R. E. Vásquez, “Gearbox fault diagnosis based on deep random forest fusion of acoustic and vibratory signals,” *Mechanical Systems and Signal Processing*, vol. 76–77, pp. 283–293, Aug. 2016, doi: 10.1016/j.ymssp.2016.02.007.

[91] K. Gajowniczek, I. Grzegorczyk, and T. Ząbkowski, “Reducing False Arrhythmia Alarms Using Different Methods of Probability and Class Assignment in Random Forest Learning Methods,” *Sensors*, vol. 19, no. 7, p. 1588, Jan. 2019, doi: 10.3390/s19071588.

[92] E. Ccopa Rivera *et al.*, “Data-Driven Modeling of Smartphone-Based Electrochemiluminescence Sensor Data Using Artificial Intelligence,” *Sensors*, vol. 20, no. 3, p. 625, Jan. 2020, doi: 10.3390/s20030625.

[93] A. Diez-Olivan, J. A. Pagan, N. L. D. Khoa, R. Sanz, and B. Sierra, “Kernel-based support vector machines for automated health status assessment in monitoring sensor data,” *Int J Adv Manuf Technol*, vol. 95, no. 1, pp. 327–340, Mar. 2018, doi: 10.1007/s00170-017-1204-2.

[94] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, “Machine Learning for Predictive Maintenance: A Multiple Classifier Approach,” *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 812–820, Jun. 2015, doi: 10.1109/TII.2014.2349359.

[95] M. Baptista, S. Sankararaman, Ivo. P. de Medeiros, C. Nascimento, H. Prendinger, and E. M. P. Henriques, “Forecasting fault events for predictive maintenance using data-driven techniques and ARMA modeling,” *Computers & Industrial Engineering*, vol. 115, pp. 41–53, Jan. 2018, doi: 10.1016/j.cie.2017.10.033.

[96] O. R. Seryasat, M. Aliyari shoorehdeli, F. Honarvar, and A. Rahmani, “Multi-fault diagnosis of ball bearing based on features extracted from time-domain and multi-class support vector machine(MSVM),” in *2010 IEEE International Conference on Systems, Man and Cybernetics*, Oct. 2010, pp. 4300–4303, doi: 10.1109/ICSMC.2010.5642390.

[97] C. Li, R.-V. Sanchez, G. Zurita, M. Cerrada, D. Cabrera, and R. E. Vásquez, “Multimodal deep support vector classification with homologous features and its application to gearbox fault diagnosis,” *Neurocomputing*, vol. 168, pp. 119–127, Nov. 2015, doi: 10.1016/j.neucom.2015.06.008.

[98] H. Li, Y. Wang, P. Zhao, X. Zhang, and P. Zhou, “Cutting tool operational reliability prediction based on acoustic emission and logistic regression model,” *J Intell Manuf*, vol. 26, no. 5, pp. 923–931, Oct. 2015, doi: 10.1007/s10845-014-0941-4.

[99] J. J. A. Costello, G. M. West, and S. D. J. McArthur, "Machine Learning Model for Event-Based Prognostics in Gas Circulator Condition Monitoring," *IEEE Transactions on Reliability*, vol. 66, no. 4, pp. 1048–1057, Dec. 2017, doi: 10.1109/TR.2017.2727489.

[100] D. H. Pandya, S. H. Upadhyay, and S. P. Harsha, "Fault diagnosis of rolling element bearing by using multinomial logistic regression and wavelet packet transform," *Soft Comput*, vol. 18, no. 2, pp. 255–266, Feb. 2014, doi: 10.1007/s00500-013-1055-1.

[101] W. Caesarendra, A. Widodo, and B.-S. Yang, "Application of relevance vector machine and logistic regression for machine degradation assessment," *Mechanical Systems and Signal Processing*, vol. 24, no. 4, pp. 1161–1171, May 2010, doi: 10.1016/j.ymssp.2009.10.011.

[102] T.-L. Wu, D. Y. Sari, B.-T. Lin, and C.-W. Chang, "Monitoring of punch failure in micro-piercing process based on vibratory signal and logistic regression," *Int J Adv Manuf Technol*, vol. 93, no. 5, pp. 2447–2458, Nov. 2017, doi: 10.1007/s00170-017-0701-7.

[103] W. Ahmad, S. A. Khan, M. M. M. Islam, and J.-M. Kim, "A reliable technique for remaining useful life estimation of rolling element bearings using dynamic regression models," *Reliability Engineering & System Safety*, vol. 184, pp. 67–76, Apr. 2019, doi: 10.1016/j.ress.2018.02.003.

[104] Meosis, "PRONOSTIA › Formatronics," *Patrick Nectoux Formatronics*. http://www.formatronics.fr/en/pronostia-en.html (accessed Oct. 18, 2020).

[105] N. Helwig, E. Pignanelli, and A. Schütze, "D8.1 - Detecting and Compensating Sensor Faults in a Hydraulic Condition Monitoring System," *Proceedings SENSOR 2015*, pp. 641–646, May 2015, doi: http://dx.doi.org/10.5162/sensor2015/D8.1.

[106] Y. Lu *et al.*, "A Data-Based Approach for Sensor Fault Detection and Diagnosis of Electro-Pneumatic Brake," in *2019 IEEE International Conference on Prognostics and Health Management (ICPHM)*, Jun. 2019, pp. 1–6, doi: 10.1109/ICPHM.2019.8819443.

[107] "Fuzzy Neural Network Modelling for Tool Wear Estimation in Dry Milling Operation | PHM Society." https://www.phmsociety.org/node/79 (accessed Oct. 19, 2020).

[108] P. Park, P. D. Marco, H. Shin, and J. Bang, "Fault Detection and Diagnosis Using Combined Autoencoder and Long Short-Term Memory Network," *Sensors (Basel)*, vol. 19, no. 21, Oct. 2019, doi: 10.3390/s19214612.

[109] xiaolu chen, "Tennessee Eastman simulation dataset." IEEE, Jun. 09, 2019, Accessed: Oct. 11, 2020. [Online]. Available: https://ieee-dataport.org/documents/tennessee-eastman-simulation-dataset.

[110] C. Lu, Z.-Y. Wang, W.-L. Qin, and J. Ma, "Fault diagnosis of rotary machinery components using a stacked denoising autoencoder-based health state identification," *Signal Processing*, vol. 130, pp. 377–388, Jan. 2017, doi: 10.1016/j.sigpro.2016.07.028.

[111] Z. Li, J. Li, Y. Wang, and K. Wang, "A deep learning approach for anomaly detection based on SAE and LSTM in mechanical equipment," *Int J Adv Manuf Technol*, vol. 103, no. 1, pp. 499–510, Jul. 2019, doi: 10.1007/s00170-019-03557-w.

[112] Z. Chen, S. Deng, X. Chen, C. Li, R.-V. Sanchez, and H. Qin, "Deep neural networks-based rolling bearing fault diagnosis," *Microelectronics Reliability*, vol. 75, pp. 327–333, Aug. 2017, doi: 10.1016/j.microrel.2017.03.006.

[113] H. Shao, H. Jiang, H. Zhao, and F. Wang, "A novel deep autoencoder feature learning method for rotating machinery fault diagnosis," *Mechanical Systems and Signal Processing*, vol. 95, pp. 187–204, Oct. 2017, doi: 10.1016/j.ymssp.2017.03.034.

[114] T. Junbo, L. Weining, A. Juneng, and W. Xueqian, "Fault diagnosis method study in roller bearing based on wavelet transform and stacked auto-encoder," in *The 27th Chinese Control and Decision Conference (2015 CCDC)*, May 2015, pp. 4608–4613, doi: 10.1109/CCDC.2015.7162738.

[115] H. Shao, H. Jiang, F. Wang, and H. Zhao, "An enhancement deep feature fusion method for rotating machinery fault diagnosis," *Knowledge-Based Systems*, vol. 119, pp. 200–220, Mar. 2017, doi: 10.1016/j.knosys.2016.12.012.

[116] N. K. Verma, V. K. Gupta, M. Sharma, and R. K. Sevakula, "Intelligent condition based monitoring of rotating machines using sparse auto-encoders," in *2013 IEEE Conference on Prognostics and Health Management (PHM)*, Jun. 2013, pp. 1–7, doi: 10.1109/ICPHM.2013.6621447.

[117] Z. Chen and W. Li, "Multisensor Feature Fusion for Bearing Fault Diagnosis Using Sparse Autoencoder and Deep Belief Network," *IEEE Transactions on Instrumentation and Measurement*, vol. 66, no. 7, pp. 1693–1702, Jul. 2017, doi: 10.1109/TIM.2017.2669947.

[118] W. Sun, S. Shao, R. Zhao, R. Yan, X. Zhang, and X. Chen, "A sparse auto-encoder-based deep neural network approach for induction motor faults classification," *Measurement*, vol. 89, pp. 171–178, Jul. 2016, doi: 10.1016/j.measurement.2016.04.007.

[119] H. Shao, H. Jiang, Y. Lin, and X. Li, "A novel method for intelligent fault diagnosis of rolling bearings using ensemble deep auto-encoders," *Mechanical Systems and Signal Processing*, vol. 102, pp. 278–297, Mar. 2018, doi: 10.1016/j.ymssp.2017.09.026.

[120] Y. Huang, C.-H. Chen, and C.-J. Huang, "Motor Fault Detection and Feature Extraction Using RNN-Based Variational Autoencoder," *IEEE Access*, vol. 7, pp. 139086–139096, 2019, doi: 10.1109/ACCESS.2019.2940769.

[121] "Welcome to the Case Western Reserve University Bearing Data Center Website | Bearing Data Center." https://csegroups.case.edu/bearingdatacenter/pages/welcome-case-western-reserve-university-bearing-data-center-website (accessed Oct. 19, 2020).

[122] "NASA | Open Data | NASA Open Data Portal." https://nasa.github.io/data-nasa-gov-frontpage/ (accessed Oct. 19, 2020).

[123] Z. Hui-jie, R. Ting, W. Xin-qing, Z. You, and F. Hu-sheng, "Fault diagnosis of hydraulic pump based on stacked autoencoders," *2015 12th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, 2015, doi: 10.1109/ICEMI.2015.7494195.

[124] A. Mallak and M. Fathi, "Unsupervised Feature Selection Using Recursive k-Means Silhouette Elimination (RkSE): A Two-Scenario Case Study for Fault Classification of High-Dimensional Sensor Data," Aug. 2020, doi: 10.20944/preprints202008.0254.v1.

[125] R. Sheikhpour, M. A. Sarram, S. Gharaghani, and M. A. Z. Chahooki, "A Survey on semi-supervised feature selection methods," *Pattern Recognition*, vol. 64, pp. 141–158, Apr. 2017, doi: 10.1016/j.patcog.2016.11.003.

[126] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, Jul. 2018, doi: 10.1016/j.neucom.2017.11.077.

[127] S. Alelyani, J. Tang, and H. Liu, "Feature Selection for Clustering: A Review," 2013, doi: 10.1201/9781315373515-2.

[128] M. Dash and H. Liu, "Feature Selection for Clustering," in *Knowledge Discovery and Data Mining. Current Issues and New Applications*, Berlin, Heidelberg, 2000, pp. 110–121, doi: 10.1007/3-540-45571-X_13.

[129] Y. Kim, W. N. Street, and F. Menczer, "Evolutionary model selection in unsupervised learning," *Intell. Data Anal.*, vol. 6, no. 6, pp. 531–556, Dec. 2002.

[130] E. R. Hruschka, E. R. Hruschka, T. F. Covoes, and N. F. F. Ebecken, "Feature selection for clustering problems: a hybrid algorithm that iterates between k-means and a Bayesian filter," in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, Nov. 2005, p. 6 pp.-, doi: 10.1109/ICHIS.2005.42.

[131] M. Breaban and H. Luchian, "A unifying criterion for unsupervised clustering and feature selection," *Pattern Recognition*, vol. 44, no. 4, pp. 854–865, Apr. 2011, doi: 10.1016/j.patcog.2010.10.006.

[132] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, Apr. 1979, doi: 10.1109/TPAMI.1979.4766909.

[133] D. S. Modha and W. S. Spangler, "Feature Weighting in k-Means Clustering," *Machine Learning*, vol. 52, no. 3, pp. 217–237, Sep. 2003, doi: 10.1023/A:1024016609528.

[134] J. Z. Huang, M. K. Ng, H. Rong, and Z. Li, "Automated variable weighting in k-means type clustering," *IEEE Trans Pattern Anal Mach Intell*, vol. 27, no. 5, pp. 657–668, May 2005, doi: 10.1109/TPAMI.2005.95.

[135] E. R. Hruschka and T. F. Covoes, "Feature Selection for Cluster Analysis: an Approach Based on the Simplified Silhouette Criterion," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, Nov. 2005, vol. 1, pp. 32–38, doi: 10.1109/CIMCA.2005.1631238.

[136] L. Jing, M. K. Ng, and J. Z. Huang, "An Entropy Weighting k-Means Algorithm for Subspace Clustering of High-Dimensional Sparse Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 8, pp. 1026–1041, Aug. 2007, doi: 10.1109/TKDE.2007.1048.

[137] B. Sahu, S. Dehuri, and A. K. Jagadev, "Feature selection model based on clustering and ranking in pipeline for microarray data," *Informatics in Medicine Unlocked*, vol. 9, pp. 107–122, Jan. 2017, doi: 10.1016/j.imu.2017.07.004.

[138] D. M. Witten and R. Tibshirani, "A framework for feature selection in clustering," *J Am Stat Assoc*, vol. 105, no. 490, pp. 713–726, Jun. 2010, doi: 10.1198/jasa.2010.tm09415.

[139] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 63, no. 2, pp. 411–423, 2001, doi: 10.1111/1467-9868.00293.

[140] S. Wang, J. Tang, and H. Liu, "Embedded unsupervised feature selection," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, Texas, Jan. 2015, pp. 470–476, Accessed: Jun. 11, 2020. [Online].

[141] J. Guo, Y. Guo, X. Kong, and R. He, "Unsupervised feature selection with ordinal locality," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*, Jul. 2017, pp. 1213–1218, doi: 10.1109/ICME.2017.8019357.

[142] S. Chormunge and S. Jena, "Correlation based feature selection with clustering for high dimensional data," *Journal of Electrical Systems and Information Technology*, vol. 5, no. 3, pp. 542–549, Dec. 2018, doi: 10.1016/j.jesit.2017.06.004.

[143] A. Wosiak and D. Zakrzewska, "Integrating Correlation-Based Feature Selection and Clustering for Improved Cardiovascular Disease Diagnosis," *Complexity*, Oct. 14, 2018. https://www.hindawi.com/journals/complexity/2018/2520706/ (accessed Jun. 12, 2020).

[144] O. L. Mangasarian, W. N. Street, and W. H. Wolberg, "Breast Cancer Diagnosis and Prognosis Via Linear Programming," *Operations Research*, vol. 43, no. 4, pp. 570–577, Aug. 1995, doi: 10.1287/opre.43.4.570.

[145] "Donald Bren School of Information and Computer Sciences @ University of California, Irvine." https://www.ics.uci.edu/ (accessed Jun. 15, 2020).

[146] "UCI Machine Learning Repository." http://archive.ics.uci.edu/ml/index.php (accessed Jun. 15, 2020).

[147] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. USA: Prentice-Hall, Inc., 1988.

[148] K. Y. Yeung, M. Medvedovic, and R. E. Bumgarner, "Clustering gene-expression data with repeated measurements," *Genome Biology*, vol. 4, no. 5, p. R34, Apr. 2003, doi: 10.1186/gb-2003-4-5-r34.

[149] J. Handl and J. Knowles, "Improvements to the scalability of multiobjective clustering," in *2005 IEEE Congress on Evolutionary Computation*, Sep. 2005, vol. 3, pp. 2372-2379 Vol. 3, doi: 10.1109/CEC.2005.1554990.

[150] "Broad Institute," *Broad Institute*. https://www.broadinstitute.org/ (accessed Jun. 15, 2020).

[151] "Microarray Datasets." http://csse.szu.edu.cn/staff/zhuzx/Datasets.html (accessed Jun. 15, 2020).

[152] "Microarray." http://ico2s.org/datasets/microarray.html (accessed Jun. 15, 2020).

[153] "Molecular Portraits > Download." http://genome-www.stanford.edu/breast_cancer/molecularportraits/download.shtml (accessed Jun. 15, 2020).

[154] P. Park, P. D. Marco, H. Shin, and J. Bang, "Fault Detection and Diagnosis Using Combined Autoencoder and Long Short-Term Memory Network," *Sensors (Basel)*, vol. 19, no. 21, Oct. 2019, doi: 10.3390/s19214612.

[155] A. Mallak and M. Fathi, "A Hybrid Approach: Dynamic Diagnostic Rules for Sensor Systems in Industry 4.0 Generated by Online Hyperparameter Tuned Random Forest," *Sci*, vol. 2, no. 4, Art. no. 4, Dec. 2020, doi: 10.3390/sci2040075.

[156] "Beware Default Random Forest Importances." http://explained.ai/decision-tree-viz/index.html (accessed Feb. 08, 2020).

[157] P. Probst, M. Wright, and A.-L. Boulesteix, "Hyperparameters and Tuning Strategies for Random Forest," *WIREs Data Mining Knowl Discov*, vol. 9, no. 3, May 2019, doi: 10.1002/widm.1301.

[158] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, Feb. 2012.

[159] Z. B. Zabinsky, "Random Search Algorithms," in *Wiley Encyclopedia of Operations Research and Management Science*, American Cancer Society, 2011.

[160] "sklearn.model_selection.RandomizedSearchCV — scikit-learn 0.22.1 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html (accessed Feb. 09, 2020).

[161] "SPARQL," *Wikipedia*. Jan. 29, 2020, Accessed: Feb. 14, 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SPARQL&oldid=938150653.

[162] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian Forests: Efficient Online Random Forests," *arXiv:1406.2673 [cs, stat]*, Feb. 2015, Accessed: Mar. 10, 2020. [Online]. Available: http://arxiv.org/abs/1406.2673.

[163] O. Hassab Elgawi and O. Hasegawa, "Online incremental random forests," in *2007 International Conference on Machine Vision*, Dec. 2007, pp. 102–106, doi: 10.1109/ICMV.2007.4469281.

[164] J. P. Martins, "Review: Michael R. Genesereth, Nils J. Nilsson, Logical Foundations of Artificial Intelligence," *J. Symbolic Logic*, vol. 55, no. 3, pp. 1304–1307, Sep. 1990.

[165] "Definition of ONTOLOGY." https://www.merriam-webster.com/dictionary/ontology (accessed Mar. 30, 2017).

[166] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?," *International Journal of Human-Computer Studies*, vol. 43, no. 5, pp. 907–928, Nov. 1995, doi: 10.1006/ijhc.1995.1081.

[167] I. Niles and A. Pease, "Towards a Standard Upper Ontology," in *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, New York, NY, USA, 2001, pp. 2–9, doi: 10.1145/505168.505170.

[168] M. Uschold and M. Gruninger, "Ontologies: principles, methods and applications," *The Knowledge Engineering Review*, vol. 11, no. 2, pp. 93–136, Jun. 1996, doi: 10.1017/S0269888900007797.

[169] A. Mallak, C. Weber, M. Fathi, and A. Holland, "Active diagnosis automotive ontology for distributed embedded systems," in *2017 IEEE European Technology and Engineering Management Summit (E-TEMS)*, Oct. 2017, pp. 1–6, doi: 10.1109/E-TEMS.2017.8244219.

[170] "protégé." https://protege.stanford.edu/ (accessed Aug. 11, 2019).

[171] L. W. Lacy, "Owl: Representing Information Using the Web Ontology Language," 2005.

[172] I. M. M. El Emary and S. Ramakrishnan, *Wireless Sensor Networks: From Theory to Applications*. Boca Raton, FL, USA: CRC Press, Inc., 2013.

[173] M. A. Alsheikh, S. Lin, D. Niyato, and H. Tan, "Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1996–2018, Fourthquarter 2014, doi: 10.1109/COMST.2014.2320099.

[174] E. Jovanov, A. Milenkovic, C. Otto, and P. C. de Groen, "A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation," *J Neuroeng Rehabil*, vol. 2, no. 1, p. 6, Mar. 2005, doi: 10.1186/1743-0003-2-6.

[175] E. Jovanov *et al.*, "A WBAN System for Ambulatory Monitoring of Physical Activity and Health Status: Applications and Challenges," *Conf Proc IEEE Eng Med Biol Soc*, vol. 4, pp. 3810–3813, 2005, doi: 10.1109/IEMBS.2005.1615290.

[176] M. O. Adepeju and A. Evans, "A dynamic microsimulation framework for generating synthetic spatiotemporal crime patterns," *GISRUK 2018 Proceedings*, Mar. 07, 2018. http://eprints.whiterose.ac.uk/128602/ (accessed Aug. 24, 2019).

[177] Q. I. Ali, A. Abdulmaowjod, and H. M. Mohammed, "Simulation amp; performance study of wireless sensor network (WSN) using MATLAB," in *2010 1st International Conference on Energy, Power and Control (EPC-IQ)*, Nov. 2010, pp. 307–314.

[178] M. N. Jambli, M. I. Bandan, K. S. Pillay, and S. M. Suhaili, "An Analytical Study of LEACH Routing Protocol for Wireless Sensor Network," in *2018 IEEE Conference on Wireless Sensors (ICWiSe)*, Nov. 2018, pp. 44–49, doi: 10.1109/ICWISE.2018.8633291.

[179] "Weather archive Jena." https://kaggle.com/pankrzysiu/weather-archive-jena (accessed Aug. 13, 2019).

[180] "Intel Lab Data." http://db.csail.mit.edu/labdata/labdata.html (accessed Aug. 13, 2019).

[181] "Moist Air Source Domain - MATLAB & Simulink - MathWorks United Kingdom." https://uk.mathworks.com/help/physmod/simscape/lang/moist-air-source-domain.html (accessed Aug. 13, 2019).

[182] G. Bajaj, R. Agarwal, P. Singh, N. Georgantas, and V. Issarny, "A study of existing Ontologies in the IoT-domain," *arXiv:1707.00112 [cs]*, Jul. 2017, Accessed: Sep. 01, 2019. [Online]. Available: http://arxiv.org/abs/1707.00112.

[183] A. Behravan, R. Obermaisser, and A. Nasari, "Thermal dynamic modeling and simulation of a heating system for a multi-zone office building equipped with demand controlled ventilation using MATLAB/Simulink," in *2017 International Conference on Circuits, System and Simulation (ICCSS)*, Jul. 2017, pp. 103–108, doi: 10.1109/CIRSYSSIM.2017.8023191.

[184] A. Behravan, R. Obermaisser, D. H. Basavegowda, and S. Meckel, "Automatic model-based fault detection and diagnosis using diagnostic directed acyclic graph for a demand-controlled ventilation and heating system in Simulink," in *2018 Annual IEEE International Systems Conference (SysCon)*, Apr. 2018, pp. 1–7, doi: 10.1109/SYSCON.2018.8369614.

Appendix A: Ontology and Ontology Design

In order to enhance the system ability for detecting faults, isolating them, and deciding the most suitable recovery action during the shortest possible completion period at runtime. It is necessary to build a precise and appropriate knowledge-based model that can serve the diagnosis and the fixing tasks efficiently and accurately. In this section, the literature of ontologies is described and explained as an example of knowledge-based systems, which are used in this work as a part of the model-based, component fault detection and diagnosis section to extract static diagnostic graphs.

Conceptualization is the foundation of any formally represented knowledge. conceptualization is needed to represent all the fundamental terms in a knowledge-base, including the objects, concepts and all the entities that are created to express an area of interest and all the connecting relationships between them [164].

Knowledge can be specified either implicitly or explicitly. In this study, ontologies are our knowledge-based form of interest. The term ontology itself is originally inherited from philosophy, where the ontology is a symbol of existence, the nature of being and reality, as well as the categories of beings and the connections between them. Historically, ontologies are categorized as a sub-major of metaphysics [165].

In the field of knowledge-based systems, ontologies formalize their concept from the studies in philosophy. What exists in our domain is exactly what will be included in the ontology. The set of entities that exist in our domain is called *the universe of discourse*. Thus, we can define the ontology for a knowledge-based system as a set of fundamental terms, which constructs the entities in the universe of discourse. e.g., classes, relations or other objects, and with natural languages, we can describe their names and the formal axioms and constraints between them.

Ontology representations can be either simple with just a few classes, hierarchy and relations, or extremely complicated with nested hierarchies, classes or multi-connected relations. Sophisticated ontologies tend to be easier to understand and learn by non-expert individuals. The main drawback of large ontologies is the lack of computational cost efficiency, due to the time and complexity of both querying and reasoning for large systems that can be costly to compute especially with a very complex structure or large number of instances. Hence, the main concern in designing our automotive ontology was to keep it simple but detailed enough to guarantee an accurate presentation of all the modelled facts in the reality. Additionally, simplifying the model plays an important role in simplifying the description of the raw data such as sensors types, values and thresholds, as well as being able to keep periodical history records of all the sensors values, which would benefit the quality of the diagnosis and learning in the future.

Ontology design is usually done as a collaborative task between a team of knowledge-engineers and domain experts, e.g. car technicians, IoT experts…etc. The main goal behind the ontology development process is to create a model that offers a common understanding between people of different areas of expertise and backgrounds [166].

In the last few decades, many studies about ontology development, design and methodologies have been proposed. The main steps of creating an ontology can be extracted from these studies as follows [167]:

- **Define the domain of interest:**

It is important to identify and define the domain that is needed to be modeled, the scope of this domain and the main purposes behind the creation of such domains. To ensure a better knowledge of the domain, it is advisable to answer some basic questions about the system. Such as:

What will the domain of the ontology cover?

What is the purpose of the ontology?

What type of questions are the ontology expected to answer?

Who is the target user of the ontology?

The answer to these questions might vary during the process of ontology design, but at least answering these questions would limit the scope of the ontology and direct it to a desired direction.

- **Search for existing ontologies of the same domain:**

Sometimes it is hard and time consuming to build an ontology from scratch, especially when the knowledge-based expert has no clue about the domain, or in case of the absence of field experts in the ontology development process. In this case, it is crucial to study existing ontologies of the same domain, to create a general idea about the domain of interest as well as the possibility to scale and adjust the existing ontologies to meet the desired goal ontology.

- **List keywords and important domain terms:**

It is very useful to write down all the terms related to the domain, in particular, the ones that answer the basic questions mentioned earlier.

- **Create classes and hierarchy:**

There are many approaches that describe the development of class hierarchy, we can limit them into three main approaches [168]: a top-down, a bottom-up and a combination approach. Simply the main difference between these approaches is the direction of the development process, which is very clear as the approaches names indicate; top-down for example, indicates the design process that starts with creating the top or super-classes with the most general details till the most down subclasses with the most detailed information.

- **Define class properties and data properties:**

The structured hierarchy will not provide an enough level of details, so that adding class properties and data properties will provide additional sources of clarity to the ontology and it will help to answer the basic domain questions.

- **Define the facets or constraints connected to each class and property value if needed:**

Many facets can be added regarding the value type, the allowed values and the number of allowed values that is what called 'Cardinality' and any other values that might benefit the description of the slot.

- **Define individuals:**

Define individuals for each class, subclass and all the components added, then connect these individuals with the defined relationships between them and describe their property/data values if needed.

Appendix B: Active Diagnosis and Repair Automotive (ADRA) Ontology

The active diagnosis and repair automotive system (ADRA) is a computing system, which describes the collaboration between knowledge-based systems and embedded systems to achieve an active diagnosis and repair for automotive failures at runtime [169].

Table 20 shows the main signs and symptoms associated with sensors used in the described system. The sensors, symptoms, failures and recovery actions are stored within a database. The data was collected based on the fusion of expertise between automotive and knowledge-based experts. It contains the following main attributes: sensor type, sensed object attribute (the sensed object is a 'car'), location of the sensor in the sensed object, possible symptoms associated with the sensor, the sensor's current value, sensor threshold value, external conditions and the main failures and the recovery actions performed to fix the failures. Symptoms might lead to other symptoms and the succession of connected symptoms represents a graph.

Table 20 Selected Sensors and their Corresponding Signs/Symptoms

Sensors	Symptoms/signs
Oxygen Sensor	"Check engine" light on, Bad gas mileage, Rough engine
Crankshaft Position Sensor	Issues starting the vehicle, "Check engine" light on
Camshaft Position Sensor	"Check engine" light on, Vehicle won't start

Sensors	Symptoms/signs
Knock Sensor	Loss fuel mileage, Trouble in acceleration.
Throttle Position Sensor	"Check engine" light on, Trouble in acceleration.
Pressure Sensor	Gas smell after starting the engine, Excessive fuel consumption, Trouble in acceleration.

Figure 38 shows the overall ADRA system workflow. The detection process begins with collecting the values of the embedded diagnostic sensors in the vehicle each specified period of time. Then the values are temporarily stored in a database. Stored values can be used to serve a learning process which is planned to be developed and performed in a latter project stage to improve the diagnosis of new symptoms and failures. The sensors are embedded into different components of the vehicle, such as the engine, accelerating pedals, oil and water tanks and many more.

The detection of prospective symptoms can be done by continuously monitoring and temporarily storing the instant values of these sensors and then continuously comparing them to the set threshold values at runtime. The threshold values are not the only conditions to complete the symptoms and failures. Other factors, as external, environmental factors, can include:

- **Weather conditions:** weather conditions can include the presence or absence of rain, snow, wind and dust and many other weather-related conditions.

- **Season:** incorporating the seasons might be necessary. Different seasons can trigger different changes which can lead to symptoms and failures, as well as some symptoms or failures may have a higher possibility to occur in some seasons comparing to others. So that, adding up the season attribute might help us to discover new relationships between the season and the symptoms and accidents appear in that season in particular.

- **Driving style:** the driving style might affect the sensor's reading and its' average lifetime. For example, the driver drives continuously, or he makes many stops, what is the average speed he usually drives in, does he drive on rough roads? Many other driving styles can be added to the driving style attribute, which might help in the learning process to answer unanswered questions and to find hidden, indirect relationships between symptoms, failures and each attribute.

Vehicles are continuously in motion. As a result, the sensors reading might change dramatically within short intervals. Thus, it is essential to address time in the diagnostic knowledge management system. Integrating additionally time as a semantic connection between individuals in future implementations will increase the importance of attributes and properties for an efficient reasoning.

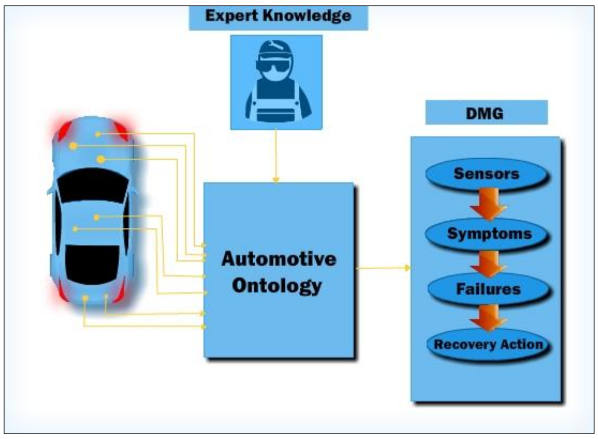


Figure 38 Active Diagnosis and Repair Automotive System Workflow

Vehicle failure diagnosis and repair is a complex decision-making process handled by mechanical and electrical technicians in order to detect failures based on observing signs and symptoms of the vehicle. Using the diagnostic ontology as the knowledge representation method to define specific symptoms for failures, can give additional support and facilitate the fault detection process for technicians. We developed an active diagnosis and repair automotive ontology using the open-source editor protégé [170].

The ontology consists of six main classes the *Sensors* class is divided into four subclasses. Each subclass shows what type of information the sensor can measure, such as temperature, pressure, speed, level and others. The *SensedObjects* class contains the possible sensed objects the ontology can diagnose - cars, trains, bicycles…etc,. This class makes the ontology scalable to different systems supported by one ontology. Each sensed object is divided into sub-classes that represent the components or parts of this sensed object, which can contain a diagnostic sensor. The *Symptoms* class describes the main symptoms that can lead to failures. The *Failures* class is a class representing all the possible failures in the sensed object. *RecoveryActions* represents the main recovery actions for the failures, addressed in the *Failures* class. Finally, the *ExternalConditions* class contains all the external factors that can play a hidden role in causing the symptoms.

The main relations in the diagnostic and repair ontology are described below:

- *CanCause(ExternalConditions, Symptoms)*: this relation states that the *ExternalCondition* that can cause a certain Symptom.

- *CanLead(Sensors, Symptoms)*: this relation connects the sensors class as the domain and the symptoms as the range, this relation connects the sensor(s) that can lead to a specific symptom(s).

- *CanLeadTo(Symptoms, Symptoms)*: some symptoms can cause the occurrence of other symptoms, so this recursive relation is to connect some symptoms to other symptoms, in order to form a sequence of causing and leading symptoms which influence the occurrence of failures.

- *IsCause(Symptoms, Failures)*: this relation states the symptoms that are responsible for specific failures.

- *IsLocatedIn(Sensors, SensedObjects)*: *IsLocatedIn* relation connects the *Sensors* and the *SensedObjects* to show in which *SensedObject* and component of this object in particular this sensor is located in. Furthermore, this relation is helpful in case of using the same sensor type in different *SensedObjects* or different parts of the same sensed object.

- *IsRecovered(Failures, RecoveryActions)*: this relation is about the *Failures* and their corresponding *RecoveryActions*.

- *Recover(RecoveryActions, Failures)*: this relation is the inverse of the *IsRecovered* relation.

The diagnosis ontology is scripted in the Web Ontology Language (OWL) [171] which guarantees a better storage, exchange and readability for the ontology. It is crucial to understand that the ontology is not a static dictionary of terms, but a semantic model of the domain and the diagnostic frame and will later on offer more dynamic features, using machine learning and text mining techniques as described earlier in this paper.

Figure 39 represents the main concepts of the ontology along with the relationships between them, as well as the subclasses of each concept and their data properties. In favor of space, not the full spectrum of instances is shown.

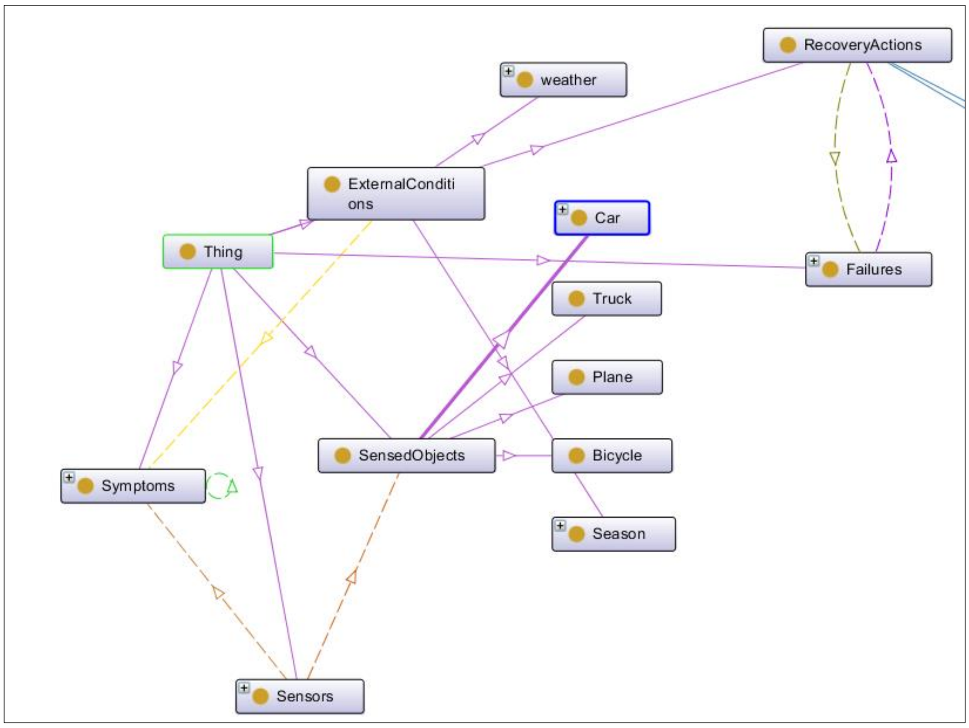


Figure 39 Active Diagnosis and Repair Automotive Ontology Structure [169].

Figure 40 shows the class hierarchy, the parent classes and the subclasses are inherited top-down. As addressed, the current version of the ontology is a starting point. The ontology is a core ontology that will be enhanced with additional dynamic solutions in future works.

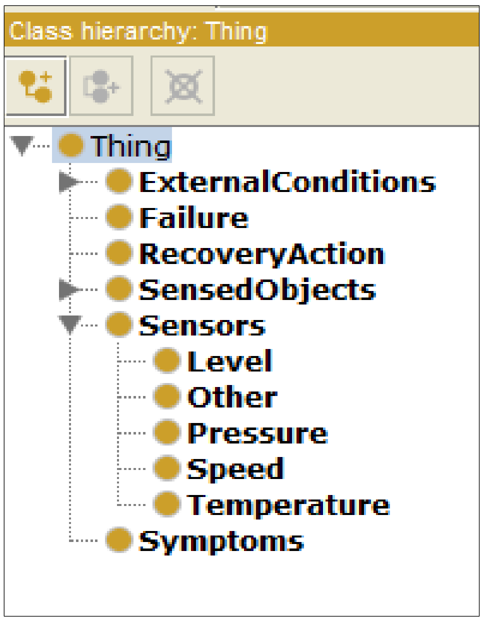


Figure 40 Active Diagnosis and Repair Automotive Ontology Class Hierarchy [169].

Appendix C: SenGen: A Two-Phase Dynamic Simulation and Toolbox for Sensor Datasets and Case-Study Generation in Mobile Wireless Sensor Networks (MWSN)

In this section, a MWSN simulated model and toolbox “SenGen” is introduced. The toolbox is used to create a customized MWSN as an example of an IoT system, where the system architecture and the relationships between its components is used to create the entities and semantic rules of the IoT ontology.

2.1 Why SenGen for Mobile Wireless Sensor Network (MWSN)?

Mobile Wireless Sensor Network (MWSN) is a group of non-stationary or mobile transducers enabled with wireless communication channels, designated to sense the environment where they are placed, in order to accomplish some intended system goal(s). The sensor nodes gather the sensed information and transfer it to a centralized backend unit called “Sink” or “base stations”. The sensor nodes can measure different modalities, depending on the nature of the system’s application and purpose, like pressure, temperature, light, speed, humidity and so on. WSNs was first applied during the world war period to survey enemy movements in the battlefield. Nowadays, MWSNs are applied in a vast majority of different fields and applications, such as industrial monitoring, home surveillance, habitat monitoring, traffic control and many more applications [172].

MWSN is an emerging field due to its flexibility, adaptability with dynamic topologies and the possibility of deploying anywhere. Due to its dynamic nature, when it comes to sensing data, MWSN tend to generate dynamic datasets as well. Sometimes, the exported sensor data is required to be altered to fulfil different system requirements, such as system scaling, change and calibration, as well as generating customized data for various research purposes. This need of customization in sensor data generation in MWSNs is highly dependent on system experts' intervention to change the physical or simulated systems upon required.

MWSNs study and research is a wide subject, which still has a lot of research problems, questions and optimizations that need to be investigated and researched. The sensor data generated from such networks is highly dimensional with multiple features and high complexity, which makes it a resource expensive, exhaustive, and extremely complicated to perform many tasks manually, such as system monitoring, maintenance, fault diagnosis and prediction tasks. Thus, the need for applying deep learning and other machine learning approaches to resolve such challenges is highly required [173]. To train and build such machine learning models, a huge amount of data is required, and not only that, this data should be customized enough to represent the research challenge in hand.

The possibilities, challenges and changes in MWSNs are endless. User-customized test cases are necessary for each particular research about MWSNs. Finding the right case-study for building a machine learning model is a very difficult challenge. It would be a lot easier, if some user-specific datasets were available. Where these datasets can fit any research questions and challenges, and also can be changed easily, without any pre-knowledge or expertise of the physical or simulated MWSNs model to generate and alter the data exported. If this possibility was not available, the researcher would be forced to conduct a lot of study and research to learn how to establish the preferred system, either using some simulation tool or building an actual physical system first. And then have to proceed with the generated data to solve the original intended research questions, which will add up a lot of time, effort and pressure on the researcher or customer.

The collaboration between machine learning and MWSNs is relatively recent. That is why, it is a huge challenge to find available sensor data for research purposes, and it is almost impossible to find platforms or benchmarks to provide customized ones.

In this section, a two-phase dynamic simulation toolbox -so-called ‘SenGen’- for sensor data and sensor case-study generation in MWSNs is presented and tested, where a full simulation of an indoor MWSN system is established using Simulink. Then a Graphical User Interface (GUI) is created with MATLAB, to overall perform as a dynamic toolbox for sensor data generation in MWSNs. Moreover, a common MWSNs challenge is tackled in the next chapter; sensor fault detection and identification using the data generated by SenGen in a distributed fashion, to showcase the capabilities of the provided solution to be customized, reliable and even extended to endless possibilities for researching sensors in MWSNs and overcome the challenge of the scarcity of these kinds of data generation platforms for MWSNs.

2.2 Related Work to SenGen

There are some tools available online that are successfully done creating dynamic data generation platforms, which will serve the purpose. However, these tools tend to cover different field of research and study, which is not related to MWSNs in particular [174], [175] and [176].

Furthermore, a simulated WSN model using Simulink/MATLAB is created in [177]. However, this model does not cover sensor mobility options and does not provide dynamicity, or support model alteration and modifications. Finally, there are some fixed WSN sensor datasets available for download online, but these are by far the least dynamic option, since the simulated model or the physical system is not available, that eliminates any possibility for modifications even when the system experts are available and have full knowledge of the created system .

To provide MWSN sensor data that is user-specific, we have developed a MATLAB GUI platform where users of different levels of expertise can customize the model. By having the possibility to add various number of sensors, of different types, in different locations and various patterns of motion. Also, different types of faults can be injected to any selected sensor(s) at any point of time during the simulation, to provide the possibility in researching anomaly detection and sensor fault detection and identification, along with a visualization representation of the whole system. This user-friendly GUI is built based on a pre-simulated Simulink model where a system of stationary sensors is placed in an indoor environment. The simulated data is linked with mobility and a routing protocol in MATLAB environment and finally presented to the user through an executable GUI.

Due to the continuous motion of the sensor nodes in MWSNs, the topology of these networks is in simultaneous change and the routing algorithm must handle the fast changes in the topology effectively. The routing from source node to sink could happen either by direct hopping (single hop) or through multiple hops using the in-between nodes from the source node to sink. Unai, Ugatiz and Caralos presented a unique technique for routing in MWSNs [177] following multi-hop technique where on real-time basis, the shortest path between is calculated at the source node and the data is communicated accordingly. In such scenarios, the energy of all the mediating sensor nodes is consumed for data transfer. This lowers the battery life of each nodes in-between which lowers the longevity of the entire system. Since battery life of sensor nodes is important in MWSNs, the routing could also be done using LEACH (Low Energy Adaptive Cluster Technology) [178]. LEACH is a hybrid method where a group of sensors form a cluster and the center-most sensor becomes a transmitter for all the cluster members to transfer data to sink. Though this method seems like a multi-hop scenario, it is not! Because the transmitter node acts as a primary sink and do not sense the surrounding as far as it acts as a transmitter. On following so, only two nodes consume energy; the source node and the cluster head node. During data transfer, the cluster head is comparatively more active than cluster members hence its battery drains out sooner. But, by rotating the role of clusters among the cluster heads, the total energy consumption is minimized.

2.3 SenGen System Overview

SenGen consists of two fundamental phases. In the first phase, an in-door MWSN system is created using Simulink/MATLAB environment to establish the foundation design of the model, which in a later phase can be altered to dynamically meet the user-specifications and requirements. The simulated model, represented by phase one will generate informative data for each sensor that consist of the sensors' readings, their locations in the simulated system, their mobility patterns, and then apply LEACH routing protocol based on their previously mentioned information. The second phase represents the dynamic factor of this software by developing a Graphical User Interface (GUI) in order to provide flexibility and the possibility to externally tweak the basis simulation model created in phase one, without the need to understand or to deal with unnecessary technical details indicated within the simulation model.

- **Phase 1: Indoor Simulation Model Creation**

The first step to start with in creating a simulated model using Simulink, is to find an input signal that is sufficient for the end system goal and requirements. In this simulated model, the input signal is used as a reference signal, which will act as the simulation environment parameters that will automatically provide the required parameters placed in the pipeline react accordingly.

The aim of this simulation is to create an indoor MWSN environment that can sense and collect environmentally related information of any indoor system in a natural base, without any artificial heating, air conditioning or ventilation applied. Such as, an office building, a hospital, an apartment…etc. The data generated from such models, can then be used later for energy research purposes or smart houses technology, sensor fault detection and identification and many other research related areas.

The sensors needed in the indoor system were atmospheric sensors. Hence, a weather data is the best choice to be used as one of the reference signals in the simulated system. The weather data of Jena (Austria) [179] had a suitable data with the psychometric details around Jena recorded for every 10 minutes over a period of 10 years. The environmental parameters of the simulation model must change more frequently, to match the real-time nature intended in this model. So that, this dataset is up-sampled using anti-aliasing filters which distribute the data between two given data-points. In our case, we choose the data to change for every single second hence the original data is re-sampled with a ratio of 600:1.

Since an open atmospheric environment cannot be modelled in Simulink because it is really hard to simulate the unpredictable weather changes, that varies a lot from one region to another. Hence, an indoor system is created, which consists of several closed rooms that built in a way it imitates actual, naturally ventilated closed environments. Each room design was inspired by the Intel Berkeley research lab [180]. The lab measurements are taken as references for each room modelled in our system since the lab also contains WSN that are stationary and fixed locations, in specific locations of the simulated system as shown in *Figure 41*. The Berkeley lab consists of 54 sensors distributed across the lab of size *50mx40m*. Figure 42 shows a top-view design of the indoor system simulated using Simulink where the air flow source, the total number of rooms, doors and the fixed locations of the routers and their measurements are clarified.

The air source is created using 'Moist air' library provided by Simulink [181], a virtual system is built consisting of three similar building blocks functioning parallel by drawing moist air from the system gateway. The moist air flow represents the natural ventilation provided in the indoor system. The gateway has a cross-sectional area of 0.75 m^2 which corresponds to the size of the door of Intel lab from which the moist air enters, and each building block has flow area of *0.25 m^2*. It comprises of a heat exchanger to modify the temperature and humidity levels of moist air, pipe block which models the addition of humidity and carbon-di-oxide into the system along a length of *5 m*, a variable local restriction element modelling the pressure loss in the flow and finally the moist air goes down to sink/reservoir. In *Figure 43*, the architecture of the air flow source is shown.

Sensor blocks are placed after each component to sense psychometric parameters. The variable parameters for all flow modifying blocks is initialized with random values to add up more realistic changes to the model. The architecture of the simulation model building block, or in other words one room of the indoor system created in Simulink is shown in *Figure 44*. The sensed outputs are exported to MATLAB environment using "simout" block and are saved for a future reference.

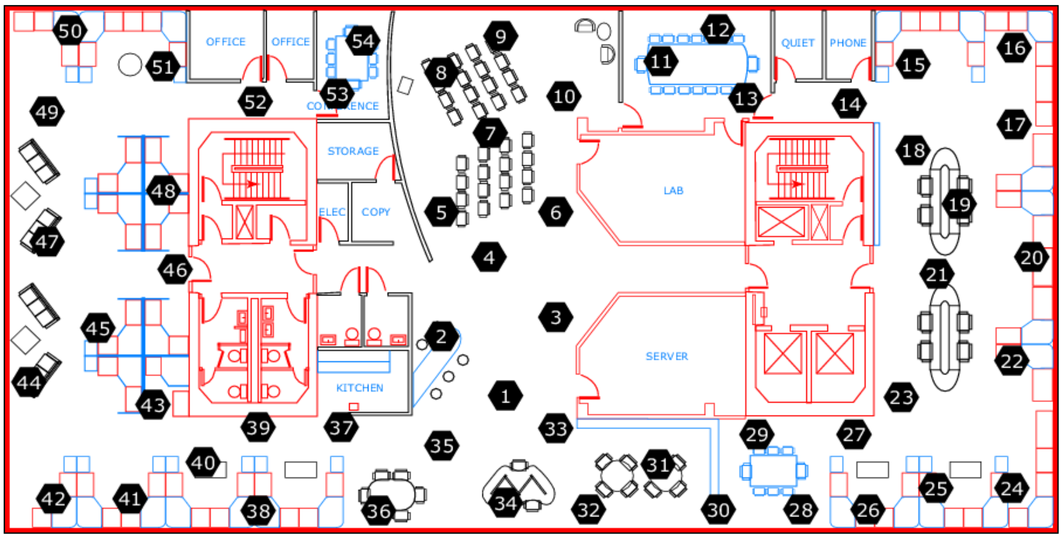


Figure 41 Sensor Distribution of Intel Berkeley Lab [180].

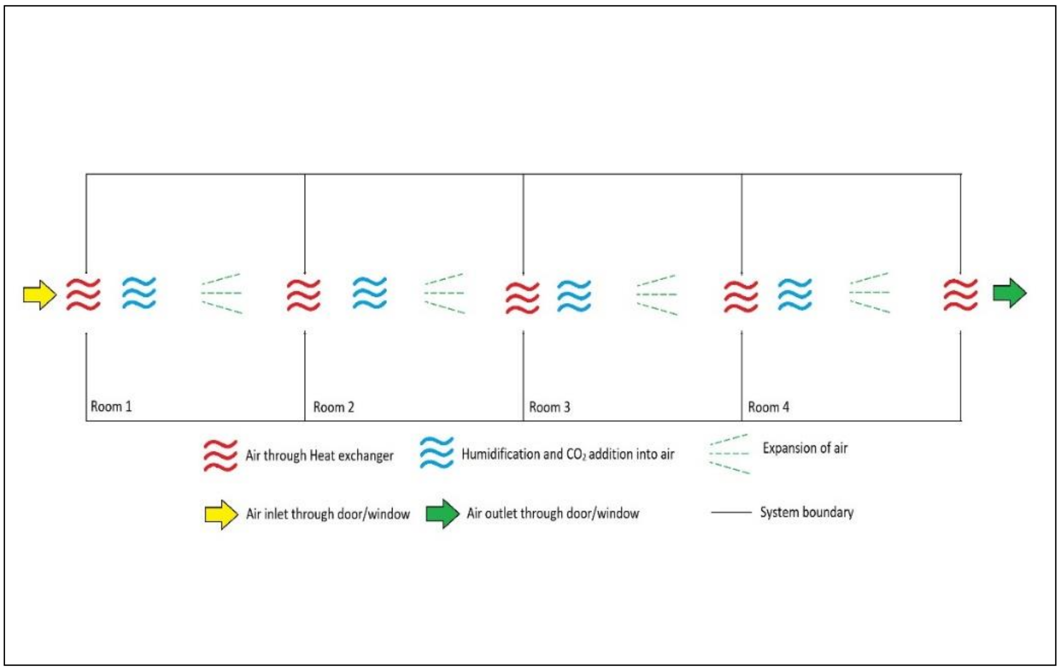

Room 1
Room 2
Room 3
Room 4
Air through Heat exchanger
Humidification and CO₂ addition into air
Expansion of air
Air inlet through door/window
Air outlet through door/window
System boundary

Figure 42 Top-view Indoor System Design.

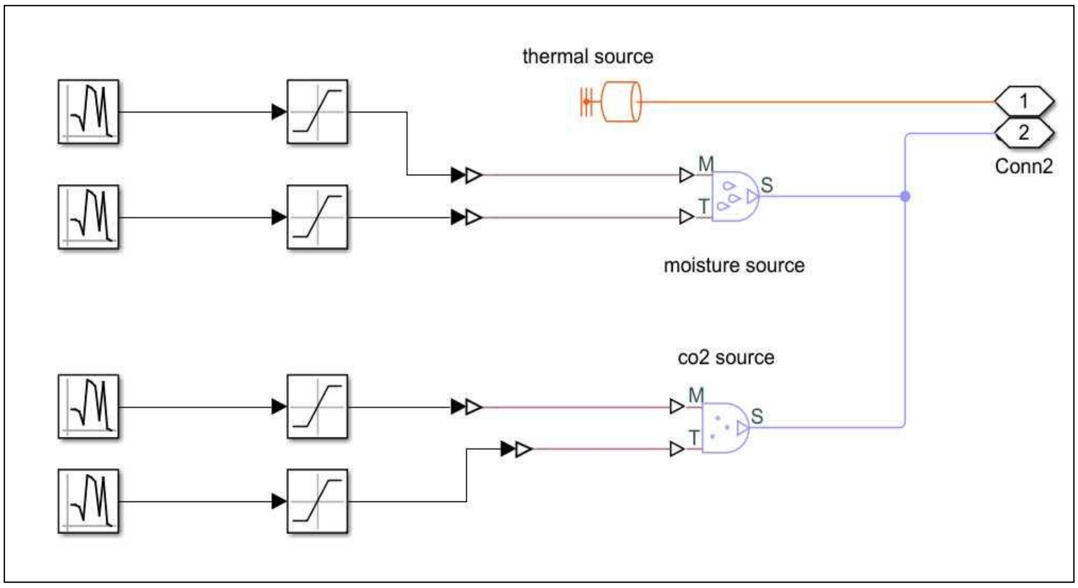


Figure 43 Air Flow Architecture.

Previously, in this paper, how to create the simulated model with fixed or stationary sensors was mentioned. To convert this WSN into a MWSN it is necessary to add some mobility patterns and scenarios for the sensors to use as their motion. For this purpose, a football players' position data [14] is used to extract realistic patterns of movements, by normalizing it to the size of the lab instead of the football field. This motion pattern is provided as a pre-defined one which the user can immediately use. However, the user can also apply their own mobility pattern(s) using the GUI as explained in the next section.

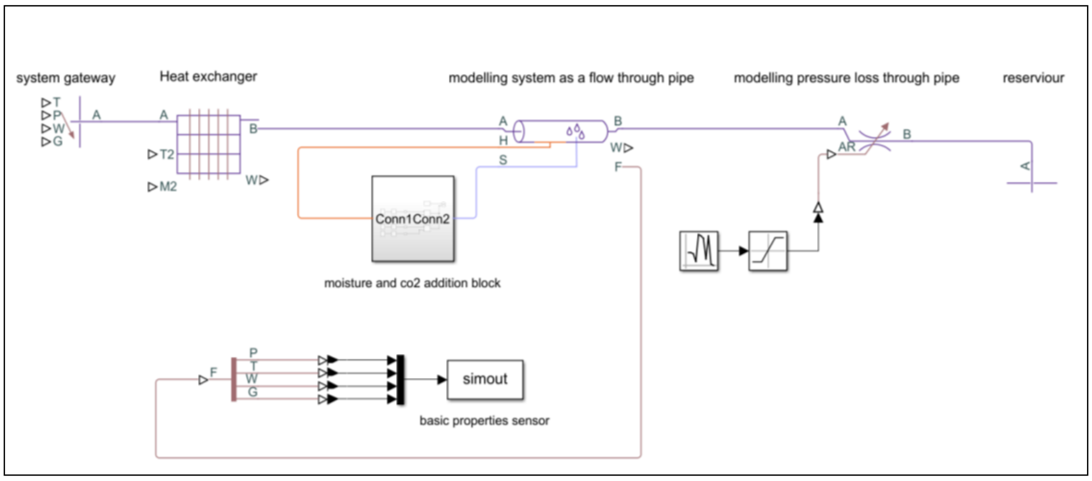

system gateway
Heat exchanger
modelling system as a flow through pipe
modelling pressure loss through pipe
reserviour
Conn1 Conn2
moisture and co2 addition block
simout
basic properties sensor

Figure 44 Build Block of Simulink Model

In order to ensure more system stability, a sliding window of user-defined length is introduced, to process the information each sliding window instead of each second. In other words, if the sensor changes location so fast, the new location will only be considered if the sensor is there for a length of a sliding window. During this period the sensor nodes communicate the information to routers and then move on with intermittent motion.

- **Phase 2: Dynamic GUI Creation**

To give the user the ability and flexibility to control the process of establishing a customized MWSN system, as well as providing an animated MWSN view, a GUI is created where the user can observe and change the system easily in a straightforward fashion.

The following steps represent a user quick manual, of how to use the GUI to benefit the customized data generation, visualization and extraction:

2 User Input Panel:

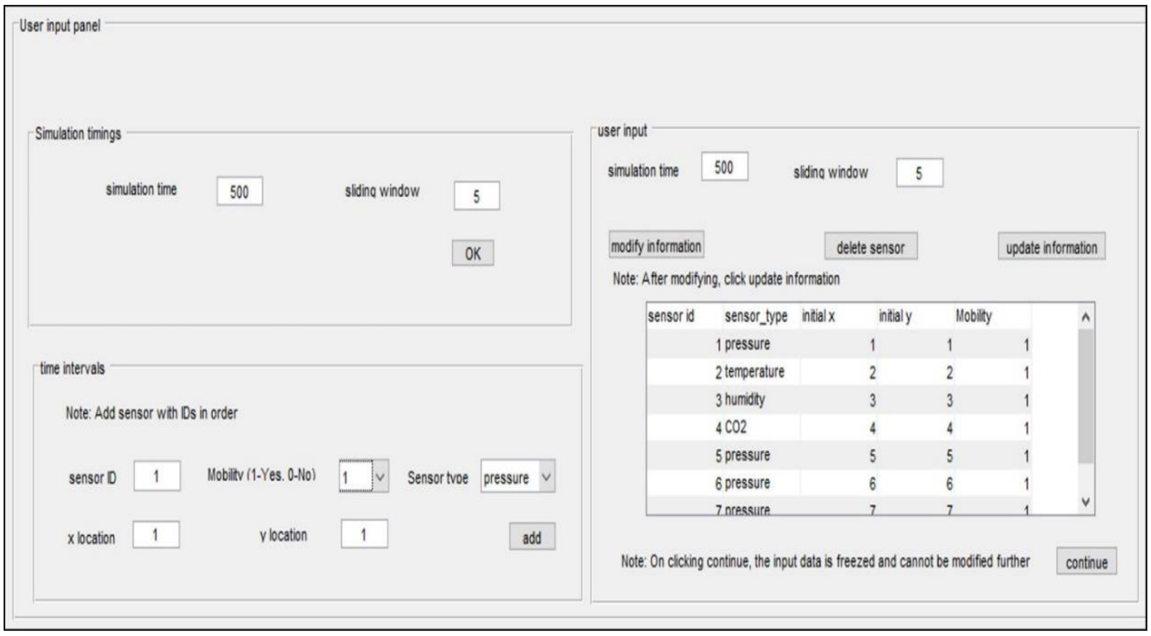
This Panel is shown in Figure 46. The user input panel is the first GUI window appears to the user, which is divided into three main sections; Simulation timings, time intervals and user input. In the simulation timing section, the user can enter the desired overall simulation time and sliding window size in seconds. In the time intervals part of the user input panel, the sensor addition process occurs that allows the user to add different sensors by providing their sensor ID, the initial x and y locations, sensor type and mobility patterns. This GUI offered a pre-defined mobility scenario and it will be activated when choosing 1 in the drop-down list under the mobility. The user can also upload a matrix of their own to perform as a mobility pattern to provide more mobility options. After adding the desired information of the sensor, press 'Add sensor' button to enable the addition of more sensors to fit the desired system requirements. It must be noted that the number of sensor nodes are limited to the number of sensor nodes placed in simulation model. The user input section located in the right side of the user panel is used to enable checking the added sensors, taking a final look at them and their information and delete or modify any sensor(s), to make sure the added information matches the wanted design before moving to the next panel and start the real-time simulation. Press 'continue' to start the real-time simulation of the inserted sensors.

3 Animated Real-time Panel:

When pressing 'start', the sensor nodes (blue) start to move following the mobility pattern chosen before. The routers (red) being stationery. The visualization will continue till it reaches the simulation time defined in the previous panel. The user can pause the simulation to make observations anytime during the simulation.

During the pause mode, the user has flexibility to inject any kind of sensor faults; bias, drift faults, constant or freezing faults...etc, to any sensor(s) on that particular instant by simply providing their sensor ID. For simplicity reasons, the faults types added are High, Low and Constant. Figure 45 shows the GUI panel for visualization and fault injection. The fault injection process is very crucial, especially for those who work with anomaly detection or sensor fault detection and identification, providing them with clear system fault labels which are customized in any given point of time is beneficial and highly necessary.

When the simulation ends, the updated MWSN data would be exported automatically in excel format, or press 'extract data in excel format' button to export the excel file anytime during the simulation.


User input panel

Simulation timings

simulation time 500 sliding window 5 OK

time intervals

Note: Add sensor with IDs in order

sensor ID 1 Mobility (1-Yes, 0-No) 1 Sensor type pressure

x location 1 y location 1 add

user input

simulation time 500 sliding window 5

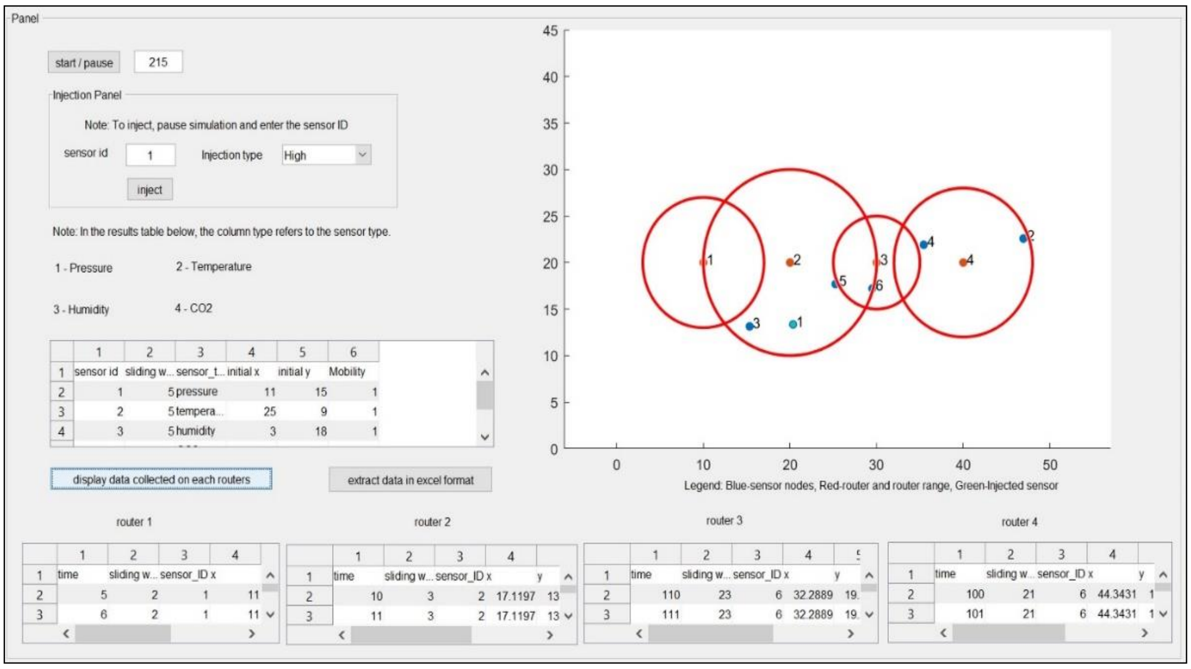
modify information delete sensor update information

Note: After modifying, click update information

sensor id	sensor_type	initial x	initial y	Mobility
1	pressure	1	1	1
2	temperature	2	2	1
3	humidity	3	3	1
4	CO2	4	4	1
5	pressure	5	5	1
6	pressure	6	6	1
7	pressure	7	7	1

Note: On clicking continue, the input data is freezed and cannot be modified further continue

Figure 46 Toolbox: User Input Pane



Panel

start / pause 215

Injection Panel

Note: To inject, pause simulation and enter the sensor ID

sensor id 1 Injection type High

inject

Note: In the results table below, the column type refers to the sensor type.

1 - Pressure 2 - Temperature

3 - Humidity 4 - CO2

	1	2	3	4	5	6
1	sensor id	sliding w...	sensor_t...	initial x	initial y	Mobility
2	1	5	pressure	11	15	1
3	2	5	tempera...	25	9	1
4	3	5	humidity	3	18	1

display data collected on each routers extract data in excel format

Legend: Blue-sensor nodes, Red-router and router range, Green-Injected sensor

router 1

	1	2	3	4
1	time	sliding w...	sensor_ID	x
2	5	2	1	11
3	6	2	1	11

router 2

	1	2	3	4	
1	time	sliding w...	sensor_ID	x	y
2	10	3	2	17.1197	13
3	11	3	2	17.1197	13

router 3

	1	2	3	4	5
1	time	sliding w...	sensor_ID	x	y
2	110	23	6	32.2889	19.
3	111	23	6	32.2889	19.

router 4

	1	2	3	4	
1	time	sliding w...	sensor_ID	x	y
2	100	21	6	44.3431	1
3	101	21	6	44.3431	1

Figure 45 Toolbox: Animated Real-Time Panel

The sensor data can be extracted anytime during the simulation in a sequential or bulk fashion where the values and information of all sensors in the system, located in the range of all routers is extracted. Also, the sensor data can be extracted in a distributed manner, where each router extract the data of the sensors in its range separately from other routers.

2.4 Data Generation and Visualization using SenGen

To experiment the simulated system in phase one and its outcomes. A test experiment is set by following the SenGen Toolbox step by step, to create five sensors in user-specific x, y locations of the indoor system, and each one of these sensors can measure one of the following weather parameters; pressure, temperature, CO_2 or humidity, that are chosen manually by the user while using the Toolbox.

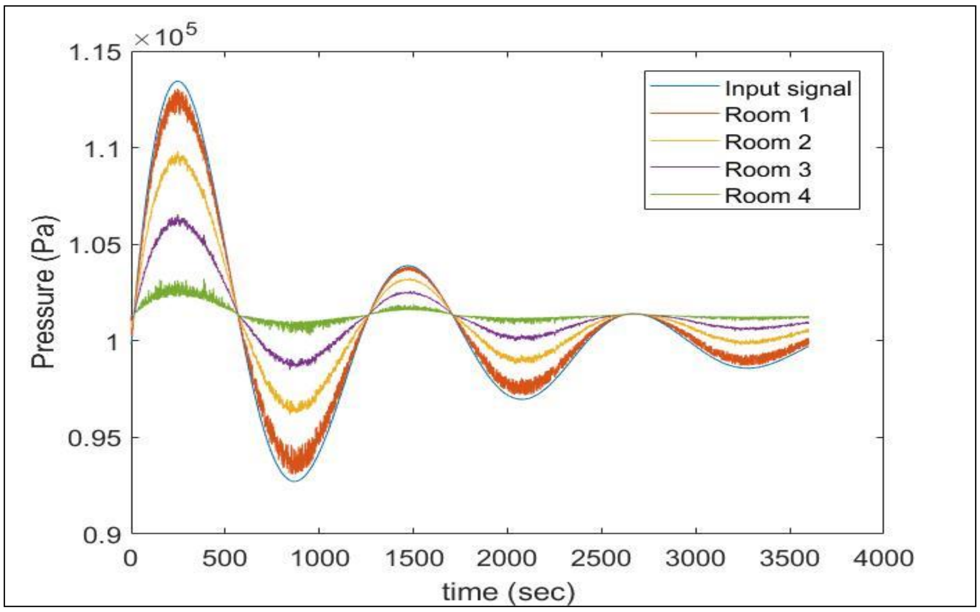


Figure 47 Pressure Signals Comparison Throughout the Indoor System with Time

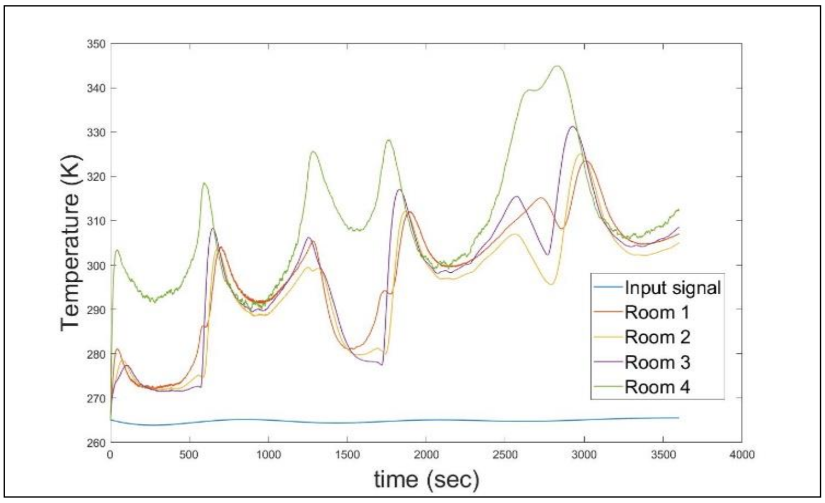


Figure 48 Temperature Signals Comparison Throughout the Indoor System with Time

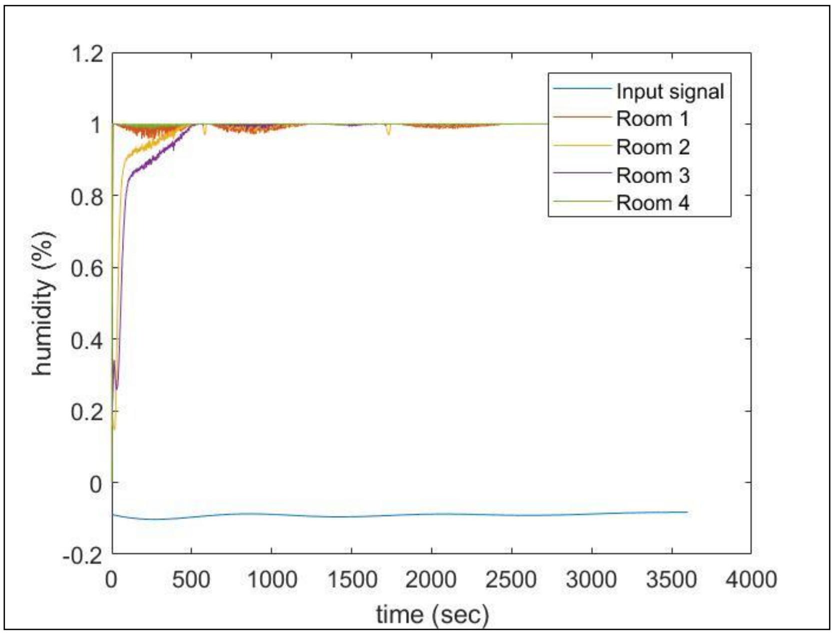


Figure 49 *Humidity Signals Comparison Throughout the Indoor System with Time*

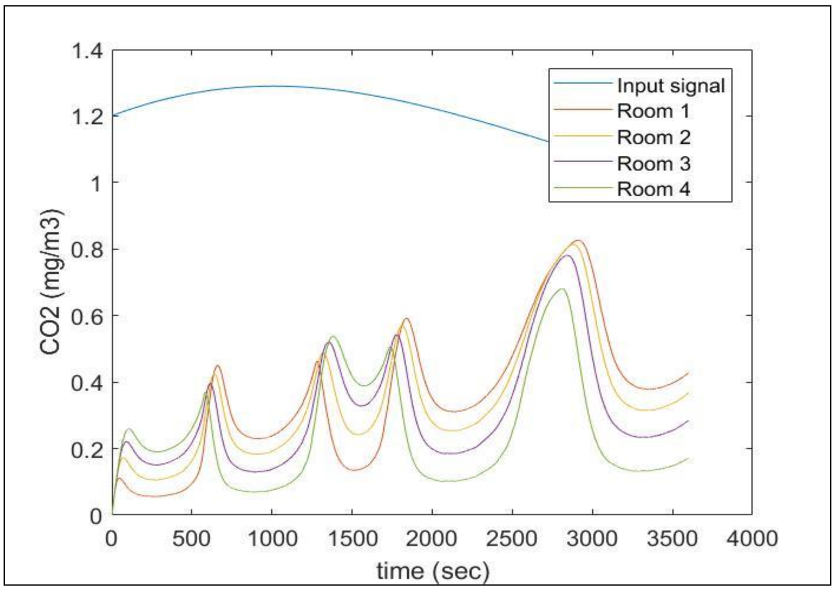


Figure 50 CO$_2$ Signals Comparison Throughout the Indoor System with Time.

Appendix D: A Model-Based Approach: A Graph-Based FDD for IoT Systems Extracted from A Semantic Ontology

In this chapter a full explanation of the model-based, component fault detection and diagnosis for IoT systems is introduced. This approach is applied on a Mobile Wireless Sensor Network (MWSN) created using SenGen toolbox.

In this work, the graph-based fault detection and diagnosis approach based on a semantic ontology proposed in [11] is adopted and applied to the IoT domain. The proposed methodology consists of the following main steps: 1) creating a simulated model or an actual physical system to provide a clear definition of the system and its components. 2) A semantic model represented by the ontology, to transfer the understanding of the system into a clear semantic representation of the system's components and the relationships between them. And finally, 3) A novel diagnostic directed graph is extracted from the ontology to offer more automation to the diagnosis and lessen the complexity of the system, by providing a clear graph of the decision-making process.

IoT Ontology: Design and Application

IoT-based devices are witnessing an increase of application in a vast majority of domains for their capabilities of generating sensor data and establishing a solid perception of the real world. Sensor data has a heterogenous nature that implicates several interoperability issues when applying to general-domain applications. In other words, it is inconvenient and highly complicated to apply one's domain sensor data to investigate another domain due to the multi-modal and heterogenous nature of sensor data. To overcome this challenge, it is essential to introduce semantic interpretations of the sensors, their containing systems and the relationships between them. An ontology as a semantic representation of knowledge has been proven to be effective to represent IoT systems and regulate the IoT-based sensor data collection and offering a comprehensive explanation of the system and its requirements [182].

FDD is an essential yet complicated and required a lot of precision step in IoT systems of various applications. Ontology-based FDD methodologies and algorithms plays an important rule in the diagnostic process especially in sensor-based applications, since they tend to be more complicated and heterogenous than other domains. Therefore, ontologies provide such systems with a semantic, expressive platform for knowledge-creation, knowledge-sharing and even re-application of the existing knowledge by adopting the represented deep semantics within the ontology structure. Ontology-based FDD has gained a lot of research focus in the past decade due to its semantic advantage over other FDD methods [3]. Although ontology-based FDD methods are solving a good amount of challenges in different domains, they tend to be a static form of diagnosis that is limited to the pre-defined semantics, knowledge and relationships stored within the ontology. In [4] is an example of an ontology of an automotive system where many sensors are used to diagnose a punch of common faults in automotive systems. Moving to another domain, a Demand Control Ventilation (DCV) ontology is introduced to tackle some faults in an office building depending on the sensors' readings in various zones of the simulated model [1].

The IoT ontology proposed in this work is created as a generic IoT domain ontology, where various examples of IoT systems can be added as an individual of the main entity of the ontology, then all the aspects of data properties, relationships and individuals can be added to give a comprehensive representation of this application. In this work, a Mobile Wireless Sensor Network (MWSN) is created using SenGen simulation and toolbox mentioned in earlier chapters and used as an example of IoT system to be described in the ontology. The MWSN created is a customized model of an indoor system that contains four rooms and two routers located in room one and two. Each room has two sensors; CO_2 and Temperature sensors that are fully mobile within the range of their containing rooms. Figure 51 shows the MWSN architecture stored in the IoT ontology.

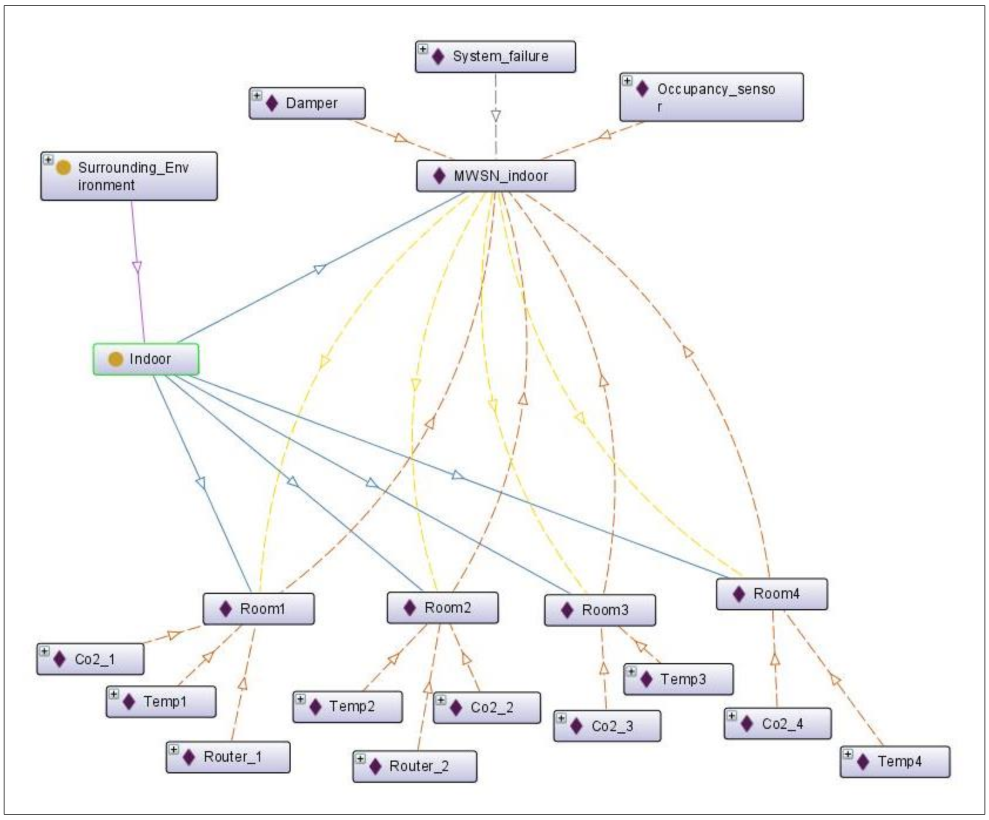


Figure 51 MWSN Architecture in the IoT Ontology

The IoT ontology consists of four main classes; *Sensory_and_Control*, *Surrounding_Environment*, Symptoms and Failures. The *Sensory_and_Control* entity describes the devices, sensors and other components of an IoT system, and in this work, it is divided into two main sub-classes of *Sensing_units* and *Routing_units*. The *Surrounding_Environment* class is added to describe the nature of the surroundings containing the IoT system, which has two main sub-classes *Indoor* and *outdoor*. The *Symptoms* class represents a set of individuals that are servings as symptoms that might lead to certain failure(s). Finally, the *Failures* class describes the instances of failures that might happen in a certain system and is connected to one or a series of symptoms stored in the *Symptoms* class. Figure 52 shows a visual representation of the class hierarchy of the IoT ontology described above.

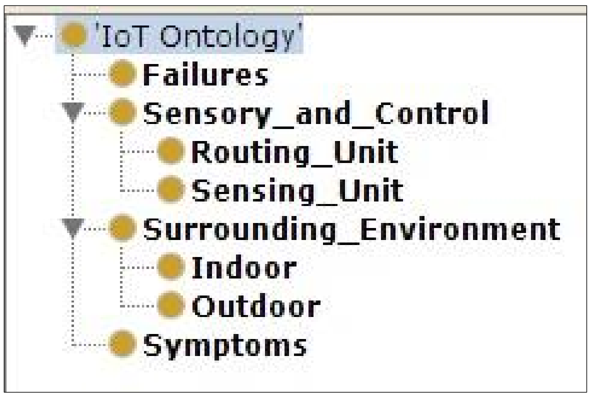

'IoT Ontology'
Failures
Sensory_and_Control
Routing_Unit
Sensing_Unit
Surrounding_Environment
Indoor
Outdoor
Symptoms

Figure 52 IoT Ontology Entities and Sub-Entities Hierarchy

The four entities form the IoT ontology must have some semantic representation of the relationships between themselves and their neighboring entities. The following bullet points are the main object properties or relations applied to the ontology, in the form of *relation_name(domain(from), Range(to))* :

- *Causes(Symptoms, Failures)*: this object property connects the symptoms to the failures by showing that symptoms causes failures.
- *Consist_of(Surrounding_Environment, Surrounding_Environment):* this relation is added to show that individuals in the *Surrounding_Environment* class can consist of other components from the same class. e.g. the instance *MWSN_indoor* of the indoor sub-class *consist-of* the individual *Room1* in the same sub-class of *Surrounding_Environment.*
- *Detected_By(Failures, Sensory_and_Control):* This relationship implies that individuals from the *Failures is Detected_By* individuals of *Sensory_and_Control class.*
- *Lead_to(Symptoms, Symptoms):* some *Symptoms* can *lead_to* other symptoms that is the reason behind adding this relationship.
- *Located_in(Sensory_and_Control, Surrounding_Environment):* the sensors, routers, devices..etc are all located in the individuals in the *Surrounding_Environment.*

Figure 53 shows a graph describing the ontology of its main entities and the relationships between those entities.

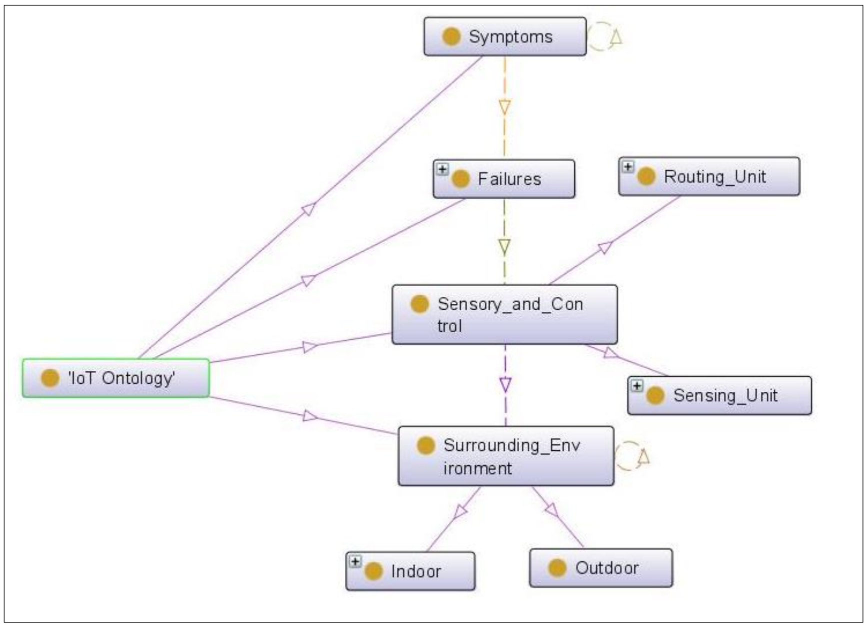


Figure 53 IoT Ontology Entities and Relationships

The diagnosis ontology is scripted in the Web Ontology Language (OWL) [171] which guarantees a better storage, exchange and readability for the ontology.

Figure 54 shows a graphical representation of the IoT ontology and the instances of a small one case study of the MWSN model. The model chosen does not have a lot of instances or complicated relationships. However, the graph is comprehensive and complicated enough to give a hint of how big and complex these semantic graphs can reach if more case-studies and instances are added.

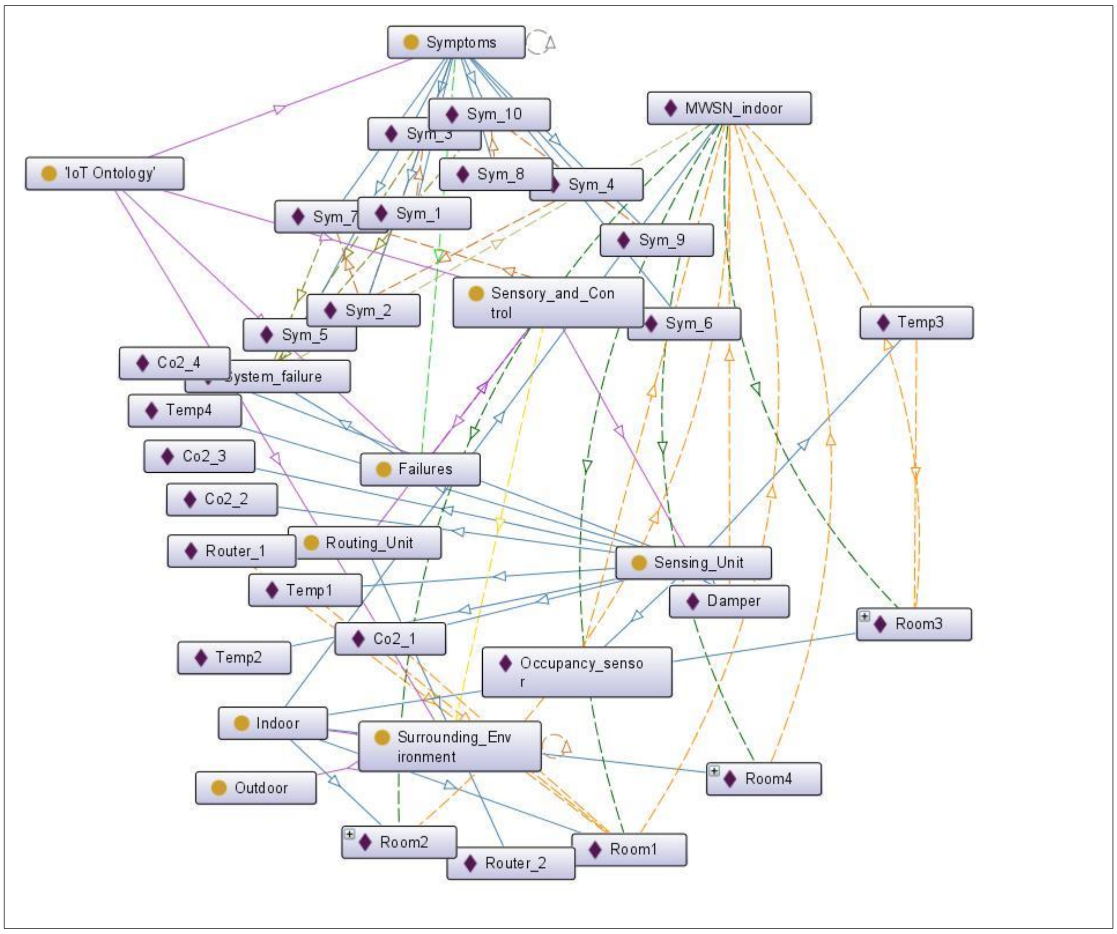


Figure 54 IoT Ontologies Main Classes and Some Individuals and Their Relationships.

A Graph-Based Sensor Fault Detection and Diagnosis

In this step, the semantic information provided by the domain experts, and stored in the IoT ontology are translated into a directed graph. Some might underestimate the importance of this stage for the rule-based diagnostic approaches, where such methods rely on IF-Then statements most of the time, which makes it so much time and effort inefficient. By representing these rules using a clear graph, the time and effort required for the detection and diagnosis process is reduced dramatically, as well as using the graph representation of the diagnostic information provides a clear and easy platform, which can be used by any individual including experts and non-experts of the domain system or the ontology. Mapping the semantic information from the system ontology to the diagnostic graph is done manually by the knowledge-based experts, to ensure that all the needed information was translated precisely and completely.

The diagnostic information and rules represented by a set of symptoms are added to the ontology as instances of the *Symptoms* class. These symptoms are created based on the system expert knowledge and expertise, where then fed to the ontology and translated into semantic keywords of relationships. The diagnosis rules implemented in this ontology are adopted from an existing demand control ventilation simulated model in [183]. Figure 55 shows that by adding the symptoms and their semantic representations to the IoT ontology it was a lot easier to extract diagnostic graphs based on the semantic graphical representation of the symptoms, their leading symptoms and causing failures.

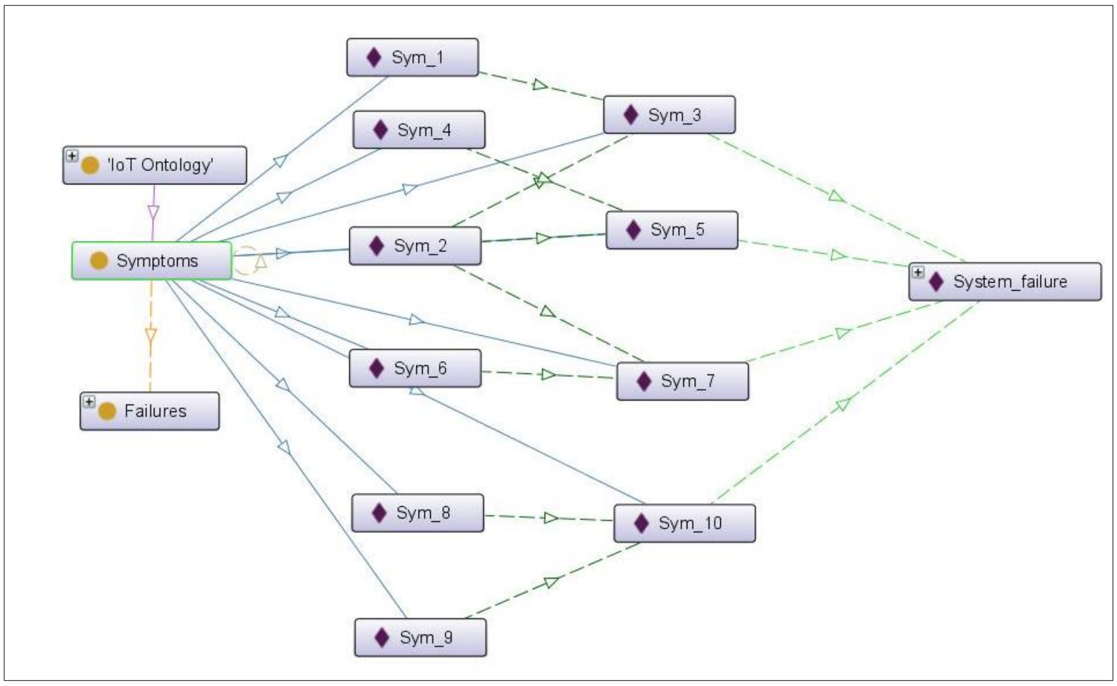


Figure 55 Expert-Rules Semantics Visualisation

Table 21 shows the main symptoms of the component failure connected to CO_2 sensor, and how these symptoms are nested to one another. Each sensor and actuator added to the system has its own symptoms table. These symptoms are provided by the embedded systems team and modelled into the ontology by the knowledge-based experts to form the final graph. For more information about the symptoms for the sensors and actuators used in this study, check [184].

Table 21. Symptoms Associated to CO_2 Sensor Failure

Symptom	Description
Sym 1	CO_2 >= 651
Sym 2	Damper is Healthy
Sym 3	Sym1 AND Sym2
Sym 4	CO_2 <= 549
Sym 5	Sym4 AND Sym2
Sym 6	CO_2_Change < (0.045 * Occupants no)
Sym 7	Sym6 AND Sym2
Sym 8	Damper_Open_Time > 1200
Sym 9	CO_2 >= 600
Sym 10	Sym8 AND Sym9

Figure 56 shows the diagnostic directed graph for the CO_2 sensor as an example of sensory and actuation effect on component/system faults. This graph is duplicated as a sub-graph for each room or corridor that contains a CO_2 sensor in it.

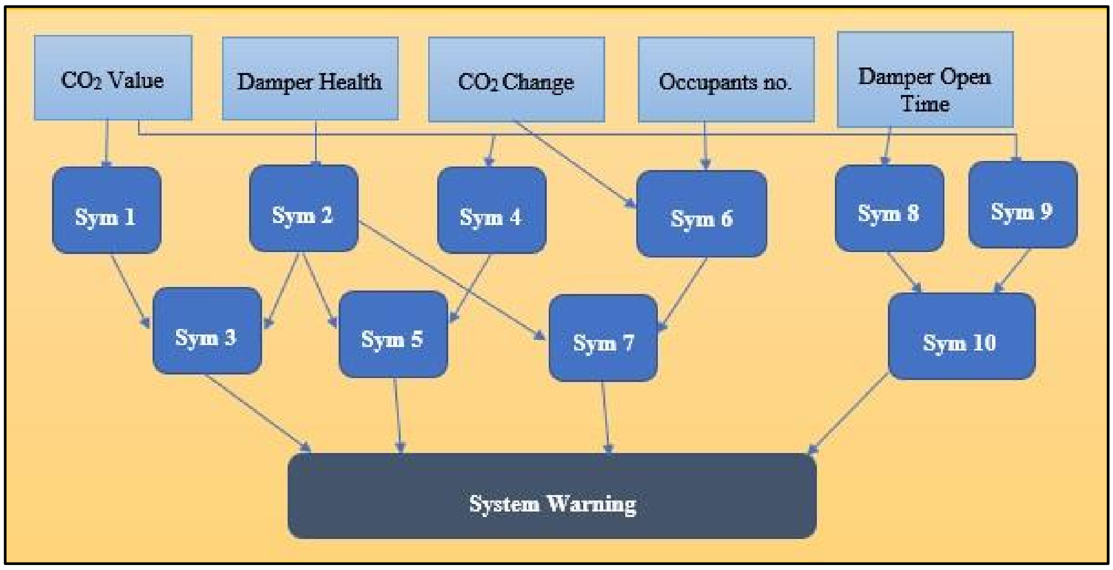


Figure 56 CO_2 Sensor Diagnostic Directed Graph

The directed graph created in this work, shows the connection between the diagnostic features extracted from the inserted instances in the ontology and their data properties. These features are connected to some corresponding symptoms, extracted from the semantic relationships added to the ontology.

The added symptoms can lead to another symptom, or directly cause the component failure.

A single diagnostic graph is created for each sensory and control device stored in the ontology, that is located in each and every component of the IoT ontology, regardless in which component or location it was created Thus, if each room has four sensory and control devices that might have a failure, and their information stored in the ontology, then the diagnostic graph of this room, contains four main sub-graphs connected to each sensory device. Keep in mind that our MWSN simulated and used to create the ontology is indoor and has four rooms, which means the overall diagnostic graph represents all the sub-graphs of each sensory device from all the indoor rooms and corridors. As a result, the overall diagnostic directed graph can be so complex and computationally challenging for bigger systems. To overcome this challenge, the integration of date-driven approaches to support this expert-based graph representation is needed. Applying machine learning as an example of the data-driven techniques will provide more dynamic solutions to this graph, by adding the possibility to prune, add some branches or learn the values and thresholds stored in each feature and symptom node in the diagnostic graph.