

"When you are studying any matter, or considering any philosophy, ask yourself only what are the facts and what is the truth that the facts bear out. Never let yourself be diverted either by what you wish to believe, or by what you think would have beneficent social effects if it were believed"

— Bertrand Russell

Preface

This master thesis is written to fulfill the requirement of TPK4950-Reliability, Availability, Maintainability, and Safety, Master's Thesis. This thesis is a part of the two-years master's program in Mechanical and Industrial Engineering (MTP) at NTNU. The thesis is written during the spring semester in 2020.

The research work is the continuation of the specialization project written in the fall semester 2019 about *Predictive maintenance and digital twin*, where a comprehensive literature review was carried out, and a digital twin road map for predictive maintenance was proposed. This thesis is aimed to propose and demonstrate an architecture for the digital twin implemented in predictive maintenance. Meanwhile, state-of-the-art methods are implemented in this thesis to get a brief view of how digital twin works. Further work could be executed based on the architecture proposed.

The inspirations of this thesis are from the course: TPK4450 - Data-Driven Prognostics and Predictive Maintenance and PK8207 - Maintenance Optimization, together with the guidance from Jørn Vatn.

This report assumes that the reader has background and knowledge within the RAMS perspective and basic programming knowledge.

Trondheim,2020-06

Jinghao Wang

汪敬豪

Acknowledgment

I would like to express my sincere thanks to those who have contributed the relevant work to this thesis and who provide kindness help to me during this pandemic period. Particularly, I would like to express my thanks to those medical crews. They are risking their lives and fighting in the frontline against COVID-19 and recapture the lives from the grim reaper.

First of all, I would like to thank my supervisor Jørn Vatn, who weekly provides help and suggestions during the whole period. The knowledge he provided helps me to understand the topic more solidary.

Secondly, I would like to thank my family, who help me go through this period when I was suffering in the deep moods.

Furthermore, I would like to thank the RAMS group, Cuthbert Shang Wui Ng and Chuangxin lyu. They provide much help during this period.

Last but not least, I would like to express thanks to all the predicament I met, which make me stay stronger and not give up.

Jinghao Wang

汪敬豪

Executive Summary

With the development of industry 4.0, the maintenance strategy starts moving towards predictive maintenance to provide dynamic support to maintenance engineers. Meanwhile, the digitalization provides a fundamental cognition from the physical asset, which could help improve system performance and availability through digital simulation and optimization. The synchronization of the physical asset becomes a trend in the system diagnostics and prognostics. Therefore, to get the state of physical assets ahead of time, the concept of the digital twin is defined to duplicate the physical behavior into digital form. Meanwhile, the Cyber-physical system and Internet of Things provide a real-time data stream to the digital twin, which provides the ability to estimate system behavior dynamically.

A digital twin could provide a dynamic system state in the future, while predictive maintenance could provide support based on system state. Therefore, it is essential to integrate predictive maintenance in the digital twin. However, predictive maintenance and digital twin is a rather emerging concept, and there is no standardized document and practical cases in the literature. Hence, a literature review was conducted to get the requirements in digital twin and predictive maintenance. Within the requirements, a digital twin framework for predictive maintenance was proposed after the literature review.

In order to demonstrate the digital twin framework, a hypothetical system is proposed, where the concept of the system is based on wind farm maintenance. This system requires to get a dynamic maintenance schedule within the maintenance window through real-time and historical data. Thus, the digital twin model is established and integrated by a communication protocol, PHM models, and a decision model. In the PHM reference model, we analyze the historical data and get the main features to extract health indicators. The dataset contains 21 different monitoring signal data and three operational settings. The dataset is formatted in time series with the 'Cycle' time scale. According to historical data, we could establish an offline reference model. In this thesis, we present three states of art methods, neural network, $k - NN$ regression model, and Geometric Brownian motion model. (Noted that: In the Geometric

Brownian motion model, we need to set the threshold for the degradation, which we regard as the failure when health indicators exceed the threshold.) Then, the offline model would be uploaded to the 'Server' by the Socket communication protocol. When it comes to the prognostics, the real-time data could be streaming from the 'Client' to the 'Server' through the communication protocol. From the online prognostics, we could get the estimated RUL in real-time. Then, the RULs could be transferred to the decision model. The decision model in this thesis is based on a discrete event simulation model to provide dynamic decision support based on RUL, Cost per unit of time, and Spares in the inventory.

Through the previous process, we could conclude that the digital twin for predictive maintenance could follow the framework proposed in the literature review. Moreover, based on the model selected, the digital twin could provide dynamic decision support during real-time monitoring. However, for this hypothetical system, we do not have a standard evaluation of the digital twin performance. So, we compare the PHM method and the performance, which could provide vital information on the properties of these three methods and might be helpful when implemented in practice.

This thesis aims to bridge the gap in the digital twin implemented in predictive maintenance and demonstrate the architecture proposed within the fields of RAMS and data-driven methods. By applying such a framework, the digital twin could provide dynamic maintenance support and realize the predictive maintenance behavior.

Table of Contents

| | |
|---|-------------|
| PREFACE | II |
| ACKNOWLEDGMENT | III |
| EXECUTIVE SUMMARY | IV |
| LIST OF TABLES | VIII |
| LIST OF FIGURES | X |
| INTRODUCTION | 1 |
| 1.1 BACKGROUND | 1 |
| 1.2 OBJECTIVES | 3 |
| 1.3 APPROACHES | 4 |
| 1.4 LIMITATION | 4 |
| 1.5 OUTLINE | 5 |
| SYSTEM DESCRIPTION | 6 |
| 2.1 PHYSICAL SYSTEM..... | 6 |
| 2.2 DATASET DESCRIPTION | 7 |
| PREDICTIVE MAINTENANCE AND DIGITAL TWIN | 9 |
| 3.1 PREDICTIVE MAINTENANCE REQUIREMENTS..... | 9 |
| 3.2 CYBER-PHYSICAL SYSTEM AND DIGITAL TWIN | 11 |
| DATA ANALYSIS AND PROCESSING METHODS | 24 |
| 4.1 DIMENSION REDUCTION | 24 |
| 4.2 TIME SERIES DECOMPOSITION | 25 |
| 4.3 PATTERN RECOGNITION APPROACHES..... | 26 |
| DIGITAL TWIN FRAMEWORK AND ARCHITECTURE | 36 |
| 5.1 COMMUNICATION PROTOCOL..... | 36 |
| 5.2 PHM FRAME AND METHOD..... | 37 |
| 5.3 DECISION MAKING | 40 |
| DIGITAL TWIN OFFLINE MODEL | 41 |
| 6.1 DATA PRE-PROCESSING AND OFFLINE REFERENCE MODEL | 41 |
| 6.2 DATA PRE-PROCESSING | 41 |

| | |
|---|------------|
| ONLINE PROGNOSTICS AND DECISION MAKING..... | 79 |
| 7.1 ONLINE PROGNOSTICS | 79 |
| 7.2 PROGNOSTICS INFORMATION OF SIX MACHINES..... | 87 |
| 7.3 DECISION-MAKING MODEL | 87 |
| ALTERNATIVE ANALYSIS..... | 94 |
| DISCUSSION, CONCLUSIONS, AND RECOMMENDATIONS FOR FUTURE WORK..... | 96 |
| 9.1 DISCUSSION | 96 |
| 9.2 SUMMARY AND CONCLUSION | 98 |
| 9.3 FURTHER WORK | 100 |
| BIBLIOGRAPHY | 103 |
| APPENDIX A | 108 |
| ACRONYMS..... | 108 |
| APPENDIX B..... | 110 |
| ROADMAP FOR PREDICTIVE MAINTENANCE DIGITAL TWIN..... | 110 |
| APPENDIX C | 111 |
| PYTHON CODES OF DIGITAL TWIN ARCHITECTURE..... | 111 |
| C.1 COMMUNICATION PROTOCOL IN SOCKET | 111 |
| C.2 DATA PRE-PROCESSING AND ANALYSIS..... | 114 |
| C.3 K-NN PROGNOSTICS MODEL (INCLUDING OFFLINE REFERENCE MODEL AND ONLINE PROGNOSTICS MODEL)..... | 123 |
| C.4 NEURAL NETWORK MODEL (INCLUDING OFFLINE REFERENCE MODEL AND ONLINE PROGNOSTICS MODEL)..... | 131 |
| C.5 GBM ALGORITHM MODEL (INCLUDING OFFLINE REFERENCE MODEL AND ONLINE PROGNOSTICS MODEL)..... | 137 |

List of Tables

| | |
|---|----|
| TABEL 2.1 DATASET FEATURES DESCRIPTION..... | 7 |
| TABEL 2.2 SENSOR PROPERTIES OF THE PHM-08 DATASET | 8 |
| TABLE 6.1 THE DETAILED INFORMATION OF THE DATASET | 42 |
| TABLE 6.2 HIGH CORRELATION COEFFICIENT VALUES IN THE CORRELATION MAP | 50 |
| TABLE 6.3 THE VALUES OF STANDARD DEVIATION AND MEAN | 51 |
| TABLE 6.4 CORRELATION, DESCRIPTIVE STATISTICS, AND DISTRIBUTION OF SENSORS FOR ALL MACHINES | 53 |
| TABLE 6.5 LINEAR TREND FOR SENSORS | 57 |
| TABLE 6.6 PCA VARIANCE VALUES | 58 |
| TABLE 6.7 THE DATAFRAME OF HEALTH INDICATOR INFORMATION AND CORRESPONDING LIFETIME | 62 |
| TABLE 6.8 THE EXAMPLE OF THE TRAINING SET AND VALIDATION SET WITH INPUT VARIABLES AND OUTPUT VARIABLES | 63 |
| TABLE 6.9 THE K VALUES WITH CORRESPONDING RMSE..... | 64 |
| TABLE 6.10 FACTORS OF THE REGRESSION FITTING FOR ALL MACHINES..... | 67 |
| TABLE 6.11 HEALTH INDICATORS FOR 100 MACHINES | 69 |
| TABLE 6.12 THE NUMBER OF ΔL VALUES..... | 69 |
| TABLE 6.13 THE μ , σ , AND THE MEAN VALUES FOR EACH MACHINE | 70 |
| TABLE 6.14 DATAFRAME FOR DEEP LEARNING NEURAL NETWORK..... | 72 |
| TABLE 6.15 DATAFRAME AFTER NORMALIZATION | 73 |
| TABLE 6.16 TRAINING DATA SAMPLE FOR NEURAL NETWORK | 73 |
| TABLE 6.17 AN EXAMPLE OF TRAINING AND VALIDATION DATASET..... | 75 |
| TABLE 7. 1 THE MONITORING DATA OF A NEW MACHINE..... | 80 |
| TABLE 7. 2 THE MEAN VALUES AND STANDARD DEVIATIONS OF LIFETIME | 82 |
| TABLE 7. 3 THE MEAN VALUES AND STANDARD DEVIATIONS OF RUL..... | 83 |
| TABLE 7. 4 THE MEAN VALUES AND STANDARD DEVIATIONS OF RUL..... | 85 |
| TABLE 7. 5 DATASET AFTER NORMALIZATION..... | 86 |
| TABLE 7. 6 THE ESTIMATED RUL..... | 86 |
| TABLE 7. 7 ONLINE PROGNOSTICS FOR SIX MACHINES..... | 87 |
| TABLE 7. 8 THE LIFETIME INFORMATION ON CYCLE 50,100 AND 150..... | 91 |
| TABLE 7. 9 PARAMETER INFORMATION..... | 91 |
| TABLE 7.10 SCHEDULE TIME AND COST | 92 |

| | |
|---|----|
| TABLE 8. 1 R^2 AND MSE FOR ANN AND $k - NN$ | 95 |
| TABLE 9. 1 COMPARISON OF THREE PROGNOSTICS MODELS | 97 |

List of Figures

| | |
|--|----|
| FIGURE 3. 1 ILLUSTRATION OF THE CONNECTIONS IN CPS | 11 |
| FIGURE 3. 2 THE ILLUSTRATION OF IOT AND CPS SYSTEM (WENINGER, 2020) | 12 |
| FIGURE 3. 3 DIGITAL TWIN FRAMEWORK AND NECESSARY INFORMATION | 14 |
| FIGURE 3. 4 SYSTEM ANALYSIS METHODS AND CLASSIFICATION | 15 |
| FIGURE 3. 5 PROGNOSTICS AND HEALTH MANAGEMENT PROCESS..... | 17 |
| FIGURE 3. 6 THE ‘BATHTUB CURVE’ OF FAILURE RATE (CROARKIN ET AL., 2006)..... | 19 |
| FIGURE 3. 7 TREND AND STATES OF THE DEGRADATION LEVEL..... | 20 |
| FIGURE 3. 8 THE INFORMATION PROVIDED BY DIAGNOSTICS (FRANGOPOL, 2011) | 21 |
| FIGURE 4. 1 THE ILLUSTRATION OF PCA..... | 25 |
| FIGURE 4. 2 K-NN REGRESSION EXAMPLE..... | 27 |
| FIGURE 4. 3 FEEDFORWARD NEURAL NETWORK FRAMEWORK | 29 |
| FIGURE 4. 4 ILLUSTRATION OF BINARY STEP FUNCTION | 31 |
| FIGURE 4. 5 ILLUSTRATION OF THE SIGMOID ACTIVATION FUNCTION | 31 |
| FIGURE 4. 6 ILLUSTRATION OF THE HYPERBOLIC TANGENT FUNCTION | 32 |
| FIGURE 4. 7 ILLUSTRATION OF THE RECTIFIED LINEAR UNITS | 33 |
| FIGURE 5. 1 THE SOCKET COMMUNICATION WORKFLOW | 37 |
| FIGURE 5. 2 OVERALL SCHEME OF OFFLINE REFERENCE MODEL FLOW CHART..... | 39 |
| FIGURE 5. 3 PHM FLOW CHART AND PROCESS ILLUSTRATION..... | 40 |
| FIGURE 6.1 ILLUSTRATION OF THE RAW DATASET | 44 |
| FIGURE 6.2 ILLUSTRATION OF LABELED DATA..... | 44 |
| FIGURE 6.3 THE INFORMATION AND FEATURES OF ALL DATA..... | 45 |
| FIGURE 6.4 CORRELATION MAP ALL MONITORING DATA FOR ALL MACHINES..... | 48 |
| FIGURE 6.5 CORRELATION MAP ALL MONITORING DATA FOR MACHINE 1 | 49 |
| FIGURE 6.6 STATISTIC COUNTING FOR ALL MONITORING DATA..... | 50 |
| FIGURE 6.7 MEAN VALUES AND STANDARD DEVIATION FOR EACH MONITORING DATA | 51 |
| FIGURE 6. 8 DISTRIBUTIONS OF EACH MONITORING SIGNAL | 53 |
| FIGURE 6.9 REMAINING SENSOR DATA WITH THE LIFETIME FOR ALL OF THE MACHINES | 55 |
| FIGURE 6. 10 ILLUSTRATION OF THE SENSOR DATA WILL BE REMOVED | 56 |
| FIGURE 6. 11 ILLUSTRATION OF FITTED LINEAR TREND..... | 57 |
| FIGURE 6.12 ILLUSTRATION OF LINEAR TRENDS | 58 |
| FIGURE 6.13 ILLUSTRATION OF FIRST THREE PCA..... | 59 |
| FIGURE 6.14 TREND AND INFORMATION FOR HI CANDIDATE | 60 |
| FIGURE 6.15 TIME SERIES DECOMPOSITION FOR SENSOR 11 OF MACHINE 20..... | 61 |

| | |
|---|-----|
| FIGURE 6.16 HEALTH INDICATOR TREND OF ALL MACHINES | 61 |
| FIGURE 6.17 ILLUSTRATION OF RMSE..... | 65 |
| FIGURE 6.18 ILLUSTRATION OF THE VALIDATION PROCESS AND THE PERFORMANCE | 65 |
| FIGURE 6.19 AN EXAMPLE OF THE DETERIORATION OF THE MACHINE | 66 |
| FIGURE 6. 20 ILLUSTRATION OF REGRESSION CURVES | 67 |
| FIGURE 6.21 THE ILLUSTRATION OF THE ESTIMATED INCREMENT..... | 68 |
| FIGURE 6.22 AN EXAMPLE OF GBM WITH A CERTAIN μ AND σ | 71 |
| FIGURE 6.23 DENSITY DISTRIBUTION OF THE HEALTH INDICATOR FOR 100 MACHINES | 71 |
| FIGURE 6.24 GBM PATHS WITH 0.8 AS THE THRESHOLD | 72 |
| FIGURE 6.25 THE NEURAL NETWORK FRAMEWORK | 74 |
| FIGURE 6. 26 MODEL LOSS ILLUSTRATION DURING THE TRAINING PROCESS..... | 76 |
| FIGURE 6.27 ILLUSTRATION OF THIS NEURAL NETWORK | 77 |
| FIGURE 6. 28 UPLOADING OFFLINE REFERENCE MODEL TO THE ‘SERVER’ | 78 |
| FIGURE 7. 1 ILLUSTRATION OF RECEIVING MONITORING DATA | 79 |
| FIGURE 7.2 THE ILLUSTRATION OF SIGNAL EVOLVING..... | 81 |
| FIGURE 7. 3 THE PRIMARY TREND BY APPLYING TSD | 81 |
| FIGURE 7. 4 DISTRIBUTION AND ESTIMATION PROCESS | 82 |
| FIGURE 7. 5 MCS PROCESS | 85 |
| FIGURE 7. 6 THE ESTIMATION THROUGHOUT TIME STEPS | 86 |
| FIGURE 7. 7 AN EXAMPLE OF THE MAINTENANCE PROCESS | 91 |
| FIGURE 7. 8 COST PER UNIT TIME WITH THE CORRESPONDING CYCLE | 92 |
| FIGURE 7. 9 SPARES SPENDING ALONG WITH TIME/CYCLE..... | 92 |
| FIGURE 8. 1 ESTIMATIONS OF PHM08 DATASET WITH THREE METHODS..... | 95 |
| FIGURE 9.1 DIGITAL TWIN FRAMEWORK PRESENTED IN THIS THESIS..... | 100 |

Chapter 1

Introduction

In this chapter, the background is presented to explain the main scope of the digital twin in predictive maintenance. A case problem is formulated and described in this section to demonstrate the objectives of the scope. Besides, approaches and limitations in achieving the scope of objectives are presented. In the end, the structure of this thesis is presented.

1.1 Background

For industries, it is a challenging problem of how to improve work efficiency based on their needs and reduce the unnecessary cost of degradation and failure of production or equipment. A study by the Wall Street Journal and Emerson shows that 42% unscheduled downtime of equipment is because of the equipment failure, and unscheduled downtime costs \$50 billion every year in manufacturing. Hence, an efficient maintenance strategy becomes essential ([IMMERMAN, 2018](#)).

In the production process, manufacturing and maintenance planning are two separate processes; however, maintenance scheduling influences both manufacturing and failure probability. The maintenance during the manufacturing makes production unavailable. The idea of integrating different maintenance decisions with prognostics and all the resources is enabled to improve the productivity, efficiency, and availability of the whole process. ([Lu et al., 2007](#); [Liao et al., 2017](#)).

As we are entering the industry 4.0, the focus on maintenance is turning from preventive to predictive ([Wegener, 2019](#)). Predictive maintenance could provide a dynamic insight view of the maintenance strategy to help lower the maintenance cost, utilize equipment operation, and improve work efficiency ([Luo et al., 2003](#); [Mobley, 2002](#)). With the development of the network, collaboration, and automation systems, predictive

maintenance has become a new challenge from theory establishment to achievement.

Nowadays, sensors and IoT are widely used in the industries, which make real-time data easily to be accessed. Thus, industries start to formulate the physical assets into digital form, to get an insight of the system, which is also known as digitalization ([Vemuri, 2019](#)). Digitalization provides possibilities to achieve predictive maintenance.

The principle of digitalization is to integrate data from manufacture to supply chain from start to end. All physical information could be transferred into digital information and connected by the Cyber-Physical Systems (CPS) to make production, sales, and supplement smarter ([Tao et al., 2018a](#)). Digital twin, as an advanced digitized system, firstly is coined from the aerospace area, especially from NASA. The purpose of the digital twin is to build a “same” digitalization system as a physical asset to simulate the state of operation ([Glaessgen and Stargel, 2012](#)). With the improvement of productivity level and the increase of operational data, the implementation of digital transformation becomes more complicated. Now, Digital twin enabled virtual producing and process planning, which provides simulation in the future for different purposes with the synchronization of the current time ([Tao et al., 2018a](#)). These properties help digital twin organize several resources data and different systems and execute the optimized solutions during the operation via simulation ([Kritzinger et al., 2018](#)).

Within the perspective of the RAMS field, the primary focus raises to data-driven diagnostics, prognostics, and maintenance strategies in the digital twin. According to real-time data and system behaviors, the diagnostics can help to determine the system state and health condition at present. As the data updated, the health condition is also updated. From the diagnoses and the system condition changing overtime, prognostics can help to find the potential RUL of the system, which could help to decide the maintenance strategies. ([Lee et al., 2017](#))

Digitalization provides an initial frame and resources to build the digital twin. Meanwhile, predictive maintenance provides the possibility of making a dynamic maintenance decision through multiple sources of data. Some researches, such as Qiao et al., Qi et al., and Tao et al., started trying to link predictive maintenance with the digital twin together and establish a bridge from the smart devices and physical objects to digital objects. However, due to various definitions and understandings of the digital

twin, these authors have a different perspective on building a digital twin. Some of them only focus on data processing methods, not on the structure for predictive maintenance. ([Qiao et al., 2019](#); [Tao et al., 2018b](#); [Qi et al., 2018](#)). The digital twin should not only be a ‘data monster’ ([Boschert and Rosen, 2016](#)).

Problem formulation

Despite having a lot of digital twins exist, which can provide information about the system conditions and integrated some predictions or other perspectives, still, there are no standardized requirements on how to establish a digital twin related to predictive maintenance in the literature. Meanwhile, the standardized digital twin framework is still under development ([ISO/CD, 2019](#)). How to efficiently integrate the information and establish the predictive maintenance digital twin is still an open question. Furthermore, what information and methods are needed for a digital twin performing predictive maintenance? In the literature review of *digital twin and predictive maintenance*, an underlying architecture of the digital twin has been pointed out ([Appendix A](#)) ([Wang, 2019](#)). Therefore, there is a need to demonstrate digital twin architecture and select appropriate methods.

1.2 Objectives

The objective of this thesis is mainly to build a digital twin to achieve predictive maintenance and demonstrate it by a case study. The following sub-objectives are listed:

- Present the primary information for the case study;
- Present the architecture of digital twin and the relevant sub-systems to establish the digital twin;
- Perform the literature review of the relevant methods or systems to achieve each fundamental architecture in the digital twin;
- Propose state-of-the-art methods in the literature to establish the digital twin;
- Establish a digital twin based on the case study and relevant data.

1.3 Approaches

The approaches in this thesis include three parts, propose a hypothetical system and relevant data to support the objectives, literature review of prevalent systems and methods, and a case study to apply the explicit methods. The literature review will cover more on health management and prognostics management, and explicit state-of-the-art data-driven methods to get a deeper understanding. In order to achieve the objectives, we will follow the digital twin fundamental architecture proposed in Wang's literature review¹. Since the literature review of *Predictive maintenance and Digital twin* has already been investigated. Here we only propose the findings in the literature.

The relevant research platform used in the literature review were *Google Scholar*, *ORJA*, *IEEE transactions*, and *Research gate*. The algorithm support and platforms are *Python*, *Kaggle*, *Medium*, *GitHub*, and *MATLAB & Simulink*. Besides, some fundamental ideas are from course *PK8207 - Maintenance Optimization* and *TPK4450 - Data-Driven Prognostics and Predictive Maintenance*. The data applied in this thesis is from the open-source *PHM08* dataset. We only choose 'Training FD_001' and 'Testing FD_001'.

1.4 Limitation

In this thesis, since the case study is based on a hypothetical system, there might be some defects when describing the system and establish the digital twin. The whole concept of the system is based on wind farm maintenance. Some ideas are rather conceptual, which could bring difficulties to digital twin modelling.

The author of this thesis is in the RAMS field, not a specialist in programming and computer science field. Due to the limited skills, the model demonstrated may not fully achieve the expected functions or only be presented as a demo model. However, the primary function of the digital twin will be presented as much as possible. Meanwhile, due to the lack of a standard framework in the literature of digital twin for predictive maintenance, it hard to evaluate the justifiability in practical.

¹ This is a literature review on *Predictive maintenance and Digital twin*. The document explicitly presents the requirements for building the digital twin of predictive maintenance and architecture of digital twin.

1.5 Outline

The main structure of this thesis will be organized as follows:

- Chapter 1: Present the background and problem formulation for the topic, the objectives to be achieved, and relevant approaches and limitations for the thesis.
- Chapter 2: Present the hypothetical system, the main architecture and problem to analysis.
- Chapter 3: Introduce the requirements for predictive maintenance and digital twin. Present necessary systems to build the digital twin and digital twin architecture.
- Chapter 4: Introduce Data analysis and processing models; Present principles and mechanisms of each method explicitly.
- Chapter 5: Present the framework and architecture of the digital twin for the hypothetical system.
- Chapter 6: Present the digital twin Offline reference model with three different methods and illustrate the data processing procedure in detail.
- Chapter 7: Present the Online prognostics model and decision model; meanwhile, present the digital twin workflow.
- Chapter 8: Alternative analysis; analysis performance of three prognostics models.
- Chapter 9: Present the discussion and conclusions for this thesis, as well as recommendations for future work.
- Bibliography
- Appendix A: Presents acronyms relevant to this thesis.
- Appendix B: Presents Roadmap for predictive maintenance digital twin in the literature review
- Appendix C: Presents the programming codes for the work carried out.

Chapter 2

System description

In this Chapter, a hypothetical system is introduced. The purpose of this system is to demonstrate and connect the digital twin for predictive maintenance. This system simulates a factory, drawn from the wind farm, which requires long-term maintenance planning and specific maintenance windows ([Seyr and Muskulus, 2019](#)).

2.1 Physical system

There is an unmanned automation factory called ATF. Inside the ATF, there are several machines in this factory. Due to some reasons, the factory is located far away from the company. There is no possibility to access in time. The factory works 24h/ day, the real-time data of each machine is through the network passing to the company (the network always works). If there are some issues happen, the digital system can alert the crews to shut down or slow down the process.

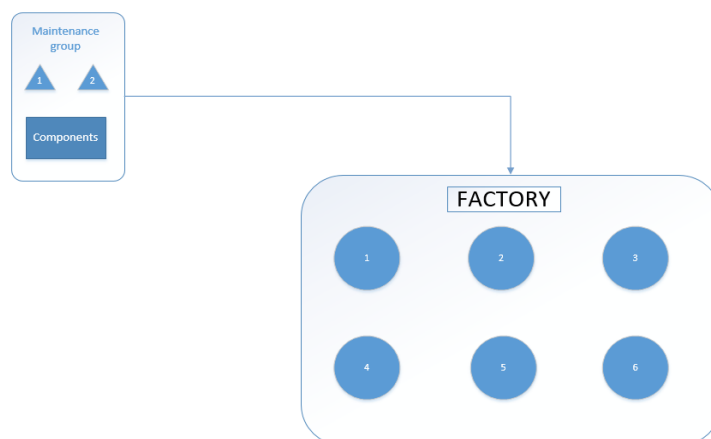


Figure 2.1: Illustration of automation factory

Due to the remote location of ATF, the maintenance group cannot fix the issue. Thus,

the maintenance is based on the schedule. Meanwhile, due to inaccessibility, maintenance only can be conducted in a specific period². During the maintenance, we assume the maintenance only needs one type of component. After repair, the machine is AS GOOD AS NEW. When it comes to the scheduled time, all the machines are maintained, regardless of their states of failure. There are two repairmen in the maintenance group. The repair time varying from 4-6 (cycles) depends on the machine state. In the system, we count time by cycles, not by hours.

2.2 Dataset description

The data set is originally from the Prognostics and Health Management PHM08 Challenge Data Set. This data set is generated by C-MAPSS (Commercial Modular Aero Propulsion System Simulation)([Saxena et al., 2008](#)). The original datasets have been pre-processed from the Kaggle website into 4 training datasets (shows in the following table), 4 testing datasets, and 4 evaluation data containing real RUL, which can be directly used in the data analysis.

Tabel 2.1 Dataset features description

| Data Set: | FD001 | FD002 | FD003 | FD004 |
|---------------------|-----------------------|-----------------------|--|--|
| Train trajectories: | 100 | 260 | 100 | 248 |
| Test trajectories: | 100 | 259 | 100 | 249 |
| Conditions: | ONE (Sea Level) | SIX | ONE (Sea Level) | SIX |
| Fault Modes: | ONE (HPC Degradation) | ONE (HPC Degradation) | TWO (HPC Degradation, Fan Degradation) | TWO (HPC Degradation, Fan Degradation) |

In this case study, we select ‘train_FD001’, ‘test_FD001’ and ‘RUL_FD001’ as our dataset, since there is only one Fault Mode. In addition, we only choose 6 test samples³ corresponding to the 6 machines in this hypothetical system. The reason we choose this

² In the wind farm maintenance, the available time slot is named as Maintenance window. In the following part, we call this period as Maintenance window. [TAVNER, P. 2012a. Offshore wind turbines: reliability, availability and maintenance, The Institution of Engineering and Technology.](#)

³ We choose 31,34,35, 68,81,82 as the samples. Since the degradation trends are similar, we could assume these machines as identical.

CHAPTER 2: SYSTEM DESCRIPTION

dataset is that the objective of the PHM08 is to predict the number of remaining operational cycles before failure in the test set.,which matches some of the objectives of this thesis. Inside the datasets from training and test, the given columns are as following:

- 1) unit (engine) number
- 2) time, in cycles
- 3) operational setting 1
- 4) operational setting 2
- 5) operational setting 3
- 6) 21 sensor monitoring data, shown in the following table:

Tabel 2. 2 Sensor properties of the PHM-08 dataset

| Symbol | Description | Unit of measure | Label |
|-----------|---------------------------------|-----------------|-------|
| T2 | Total temperature at fan inlet | °R | sen1 |
| T24 | Total temperature at LPC outlet | °R | sen2 |
| T30 | Total temperature at HPC outlet | °R | sen3 |
| T50 | Total temperature at LPT outlet | °R | sen4 |
| P2 | Pressure at fan inlet | psia | sen5 |
| P15 | Total pressure in bypass-duct | psia | sen6 |
| P30 | Total pressure at HPC outlet | psia | sen7 |
| Nf | Physical fan speed | rpm | sen8 |
| Nc | Physical core speed | rpm | sen9 |
| epr | Engine pressure ratio (P50/P2) | -- | sen10 |
| Ps30 | Static pressure at HPC outlet | psia | sen11 |
| phi | Ratio of fuel flow to Ps30 | pps/psi | sen12 |
| NRf | Corrected fan speed | rpm | sen13 |
| NRc | Corrected core speed | rpm | sen14 |
| BPR | Bypass Ratio | -- | sen15 |
| farB | Burner fuel-air ratio | -- | sen16 |
| htBleed | Bleed Enthalp | -- | sen17 |
| Nf_dmd | Demanded fan speed | rpm | sen18 |
| PCNfR_dmd | Demanded corrected fan speed | rpm | sen19 |
| W31 | HPT coolant bleed | lbm/s | sen20 |
| W32 | LPT coolant bleed | lbm/s | sen21 |

Chapter 3

Predictive maintenance and Digital twin

3.1 Predictive maintenance requirements

Due to the stochastic and dynamic behavior of the disturbance in the manufacturing process, maintenance planning becomes more critical. The concept of predictive maintenance is by analyzing relevant information and conducting the diagnosis to predict the potential failure or RUL of the equipment, as well as providing maintenance support dynamically ([Lee et al., 2017](#)).

Predictive maintenance could track the system condition by detection and indication during the operation. In principle, predictive maintenance optimizes the maintenance behavior to prevent unexpected maintenance, and lower the maintenance frequency and cost ([Mobley, 2002](#)). It is based on essential data from internal and external information to predict asset behaviors and scheduling maintenance strategy. Thus, a platform, such as a digital twin, is necessary to integrate different information to make maintenance schedules based on the prediction of assets condition and available resources.

Predictive maintenance relies on the actual condition of equipment, rather than average or expected life statistics, to predict when maintenance will be required. It means the predictive maintenance acquires not only the RUL prediction but dynamically provides different scenarios for the users to choose, based on alternative information. Therefore, a robust system and more information are necessary to realize predictive maintenance ([Cachada et al., 2018](#)).

In Wang's literature review, the following is the main points to realize predictive maintenance:

- Data gathering and pre-processing;
- Indicators selection and model training;

CHAPTER 3: PREDICTIVE MAINTENANCE AND DIGITAL TWIN

- Pre-detection and localization;
- Prediction and prognosis (including remain useful life prediction or fault prognostics);
- Decision making and actions

([Wang, 2019](#))

Within the scope, real-time monitoring is essential for predictive maintenance, which could provide internal real-time data and external information to evaluate and track system performance. Hence, a robust system that including communication and the digital system is critical for predictive maintenance ([Nguyen and Medjaher, 2019](#)).

In addition, to illustrate the predictive maintenance better, we give a brief example. We consider a critical component in a machine that is exposed to deterioration. To measure the deterioration, it is possible to install sensors that, in real-time, could monitor the behaviors of the critical component. The deterioration is considered to be governed by some stochastic loads. In order to measure the deterioration, we need an evaluation method, named as $X(t)$, to establish the bridge between the data and deterioration behavior. We assume we could observe $X(t)$ in real-time. From the point of time, we decide to change the component. Then we need to decide what time to choose to do the maintenance and take the lowest risk.

When we consider the predictive maintenance in this example, these questions should be considered⁴. $X(t)$ is univariate, could be questioned. Therefore, we need to look into how to select the feature to establish the $X(t)$ and the maintenance threshold. Meanwhile, how to monitor data and select the data-driven methods, since the data could be multi-dimensional. Besides, how to establish an objective function. Hence, there should be a digital twin system to support us in solving these questions efficiently.

⁴ The questions also could be presented as: the technical aspects for monitoring and acquiring real-time data, data-driven method and PHM process, and objective model for making decisions.

3.2 Cyber-physical system and digital twin

3.2.1 Communication system

Technically, Digital twin represents physical assets in a virtual form. As the physical assets operating, digital twin serves to simulate or estimate the state of the process. To make sure these two identical parts could be connected and synchronized, Digital twin should be able to communicate with the physical systems or multi-phase digital systems ([Wegener, 2009](#)). Cyber-physical system (CPS) and the Internet of things (IoT) enable communication between the digital twin and physical assets.

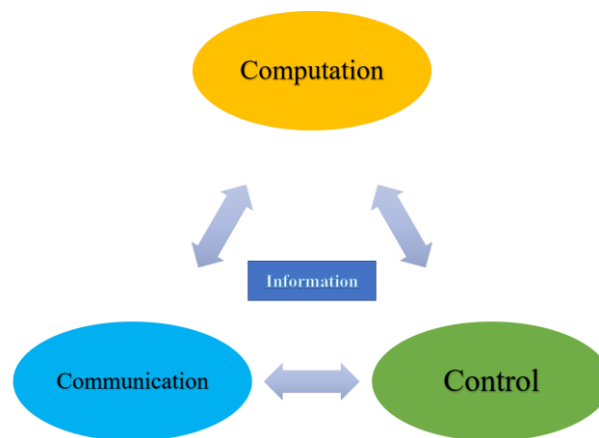


Figure 3. 1 Illustration of the connections in CPS

The Cyber-physical system (CPS) is a group combination of physical components and computational processes. The term 'cyber' and 'physical' are firmly connected and interacting. These two parts are tightly interwoven and continually interacting, which is not duplicated merely or united with each other ([Akkaya, 2016](#)). The Cyber-physical system (CPS) is not a reference model, which does not reflect any other applications. Within the scope of Industry 4.0, the Internet of things (IoT) is a successful implementation of CPSs, which connects different domains and provide a bridge ([Lee, 2015](#)).



Figure 3. 2 The illustration of IoT and CPS system ([Weninger, 2020](#))

Internet of things (IoT) provides interrelationship for physical machines and digital twin, smart devices, objects, even human. Through the unique identifiers (UIDs) of each object. IoT can transfer data over the network and remote control automatically and allow the information transferred from different places and different devices, which improve data accessibility and transmission efficiency.

The integration and connection of information make the environment can communicate with physical assets dynamically, which allows the terminal to provide an intelligent, dynamic decision. ([Akkaya, 2016](#))

3.2.2 Digital twin

The digital twin is designed to manage data, store, process, and communicate with physical systems and the environment. It is the crucial component to achieve system digitalization and real-time optimization. ([Söderberg et al., 2017](#))

The digital twin can cover various perspectives in different aspects. In the manufacturing field, a digital twin can connect essential information for prediction and simulation, which can be used to optimize the whole manufacturing procedure and activities ([Rosen et al., 2015](#)). The digital twin is not a new concept, and it is evolving from time to time. ([Wegener, 2019](#); [Bacidore, 2019](#)). For a digital twin, the most crucial element is data. The quantity and quality of data in some way could influence the accuracy of simulation digital twin. Also, the digital twin function is limited by the

data-driven method. What kind of information can be gathered from physical assets is another critical factor. In theory ([Grieves and Vickers, 2017](#)), the digital twin could represent the physical model. The feature implemented in digital could be decided by physical system analysis, which could influence the data fusion and prediction accuracy.

On the industry level, Digital twin could provide prognostics and health management (PHM), optimize decision making under uncertainty, and provide a series probabilistic model, which performs as a reward model to help correct predictive maintenance ([Rocchetta et al., 2019](#)). Because of the complexity of the system, a digital twin can help to separate the mission to different sub-systems, make whole processes more efficient, and avoid having an all integrated system, which might slow down the analysis procedure. ([Qiao et al., 2019](#))

3.2.3 Digital twin architecture

A digital twin can cover various perspectives in different aspects. In the manufacturing field, a digital twin can connect essential information for prediction and simulation, which can be used to optimize the whole manufacturing procedure and activities ([Rosen et al., 2015](#)). With a different perspective, the digital twin has different formations and requirements. In ([Wang, 2019](#)), the literature review shows that there are mainly seven requirements needed to build a digital twin for predictive maintenance:

- Physical system background and information
- Real-time data from equipment,
- Local data with lifecycle stage,
- Automatic self-updating/communication,
- Working status definition.
- Data-driven technologies,
- Decision and evaluation

They are highly suggested by literature and take a high percentage of all the requirements. Along with all the requirements, some of them have common parts with the PHM process. The PHM process can be regarded as a part of the digital twin, which could provide prognostics for the system ([Rocchetta et al., 2019](#)). So, to build the digital

twin, we need Physical system background and information, real-time data, historical data, prognostics and health management (PHM), and decision-making module.

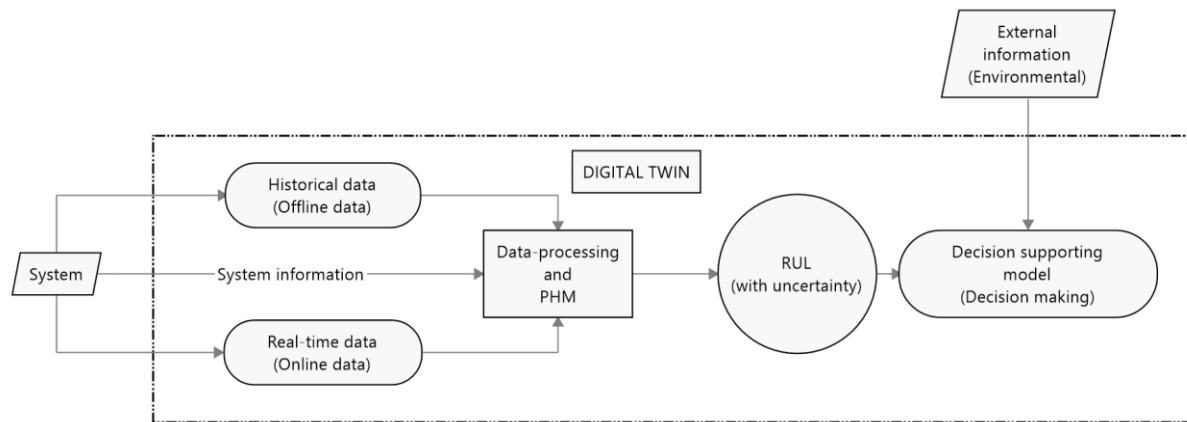


Figure 3. 3 Digital twin framework and necessary information

3.2.3.1 Physical system background and information

Nowadays, Industrial systems become more complex, and it is hard to transfer all of the system's physical processes to digital form. Hence, we need methods to extract the main feature or principal components from the physical system to represent the whole degradation ([Scheifele et al., 2019](#)).

In order to determine the critical components or sub-systems, one of the typical methodologies is to apply hazard analysis or risk assessment, which both belong to system analysis. System analysis can estimate the likelihood, cause, and consequences of a hazardous event or condition, which requires a good understanding of the system and proper methods. Typically, system analysis could be divided into two parts: qualitative and quantitative analysis, shown in [Figure 3. 4](#). Different methods provide different systems analysis approaches.

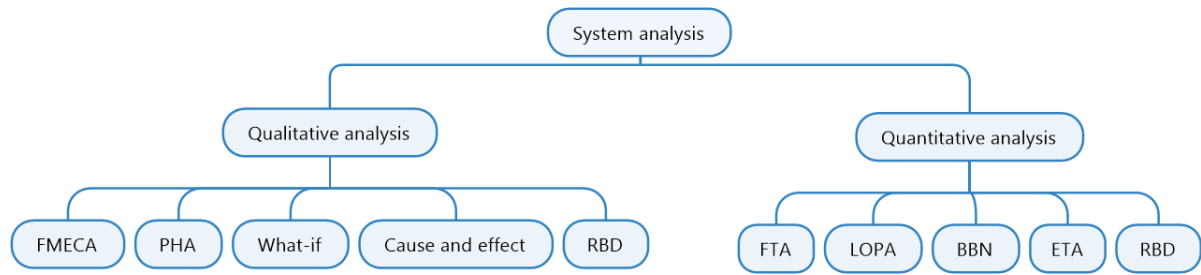


Figure 3. 4 System analysis methods and classification

Within these methods, FMECA and FTA are the primary approaches in determining critical sub-systems or components. FMECA analyzes as many components and subsystems as possible to identify the failure modes and effects. It documents the result of failure on the system, which can be regarded as the basis for inspection and monitoring. FTA represents the interrelationship between sub-systems and components related to the critical accident in the system. According to the likelihood of each critical event and components fail, we could decide which components or sub-system are more valuable to monitor.

The analysis of the physical system aims to understand the failure mode and risk of the system failure, which could help industries to avoid unnecessary cost and reduce the risk. (Bevilacqua et al., 2020). The failure mode and relevant physical information can help to determine the system degraded state, which could be adopted in establishing the health indicator for PHM (Atamuradov et al., 2017).

3.2.3.2 Data availability

In industries, sensors are wildly used in different processes. After monitoring the data, suitable software or method needs to be done to process these data. In the digital twin requirements, two kinds of data are needed: one is historical data, and the other is real-time data. In (Tao et al., 2019) and (Ding et al., 2019), historical data for the whole lifecycle stage is stored in the cloud to provide a reference model for the prognostics. In (Biesinger et al., 2019), the real-time data is gathered by sensors under MQTT (Message Queuing Telemetry Transport) and MSB (Media Stream Broadcast) protocol. In order to get the RUL prediction, the requirements of data in the following should be

considered ([Tao et al., 2018b](#)):

1. Historical data. The historical data can help to extract the main feature of the system, which includes failure mode, failure time, and system behavior. This information can indicate the potential failure of the same or similar system as a reference database.
2. Realtime data and labelling. The real-time data needs to be gathered in a fixed interval and transmitted to the digital twin. The data should be labelled with relevant physical meaning or sensor location, which is available for system component level analysis.

For some of the industries, because of the system and lack of historical data, usually conditional monitoring and mathematical model are used. The digital twin could process real-time data and give a probability of each state of the system or the probability of reaching the system healthy threshold. Then, based on prediction and real case, an evaluation for the digital twin would be conducted. Furthermore, if necessary, the improvement could be implemented to the digital twin and get better performance. For those who both have real-time data and historical data, the process would be more complicated. The historical data can be stored in the cloud or specific data storage space. The data processing could help to clean and process the redundant and missing data. After this, based on systematic analysis, some relevant features in historical data could be selected. By using mathematical or big data algorithms, a simulation digital twin model could be established, and historical data is for training the model. Then, based on the model, real-time data could be used to make a prediction. There could be two types of predictions, in which one is long-term, and the other is short term. For long-term prediction, it is usually for maintenance planning and resource arrangement. For short-term prediction, it is usually for an emergency, and it is more like conditional monitoring to get a short time alert for the unexpected situation.

([Wang, 2019](#))

3.2.3.3 Prognostics and health management

Prognostics and health management (PHM) is a modern engineering concept which provides an overview in the real-time assessment of system operational health state,

CHAPTER 3: PREDICTIVE MAINTENANCE AND DIGITAL TWIN

along with the prediction according to multiple information. PHM aims to reveal system failure trends, diagnostics, and prognostics to perform health management. Further, it can assist in providing maintenance suggestions or determine the optimal execution plan ([Atamuradov et al., 2017](#)). With the explosive development of digital information, PHM conveys data and information into connection with the health state, which provides a perceptual intuition of system properties. Due to PHM could conduct the prediction of remain useful life (RUL), it becomes an intermediary process in realizing predictive maintenance ([Kim et al., 2016](#)).

The main task of PHM is that a PHM system needs to consider three-stage: current system state estimation, estimate future state and remain useful life (RUL), and the impact of failure ([Atamuradov et al., 2017](#)). According to the task, the process of PHM includes data collection, diagnostics, prognostics, and health management. Data collection is to acquire the condition monitoring data and, according to data, extract some main features for the following process and establish an evaluation method. Diagnostics is to detect the system state at present based on the evaluation method. Prognostics is to get a predictive lifetime or RUL based on the monitoring data. Then, through all of the information obtained from previous, health management is to give a general maintenance plan. ([Atamuradov et al., 2017](#))

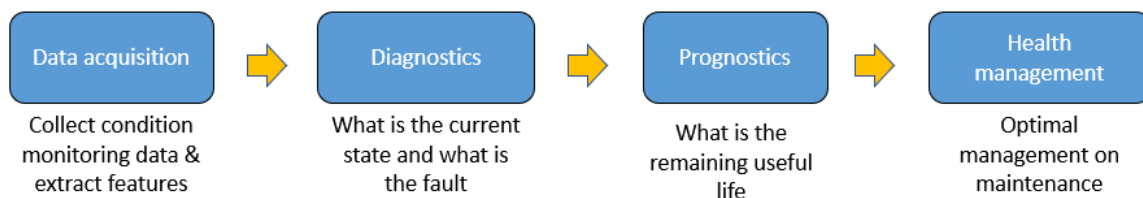


Figure 3. 5 Prognostics and health management process

Remaining useful of lifetime

The remaining useful of lifetime (RUL) is the length from the current time to the end of lifetime, which wildly applies in the PHM and maintenance schedule. RUL highly depends on the current system statue, operation environment, and relevant health condition ([Si et al., 2011b](#)). Usually, RUL follows some distribution, which is a time-dependent variable with a mean value. We define RUL (t_j) as a random variable of

CHAPTER 3: PREDICTIVE MAINTENANCE AND DIGITAL TWIN

remaining useful of lifetime at time t_j . The health indicator which denotes the health statue of the assets on degradation level, we define as $Y(t)$. During deterioration, the system may still be functional, but may not perform well after a threshold. We define S_L as an unacceptable failure limit. The RUL (t_j) is defined with the following formula:

$$RUL(t_j) = \inf\{h: Y(t_j + h) \in S_L | Y(t_j < L), Y(s)_{0 \leq s \leq t_j}\} \quad (3.1)$$

The definition requires:

1. To have $Y(t_j)$ as health indicator at time t_j ;
2. To have S_L as acceptance threshold;
3. Able to estimate $Y(t_j)$;
4. Able to predict $Y(t_j)$ at any time interval h . ([Barros, 2019](#); [Si et al., 2011b](#))

State identification

State identification is to detect and recognize the system state at present from condition monitoring data. There are a variety of factors could influence the system performance and cause degradation in operation. Even the newly assembled equipment has an early failure period. An example is the ‘Bathtub curve’, shown in [Figure 3. 6](#).

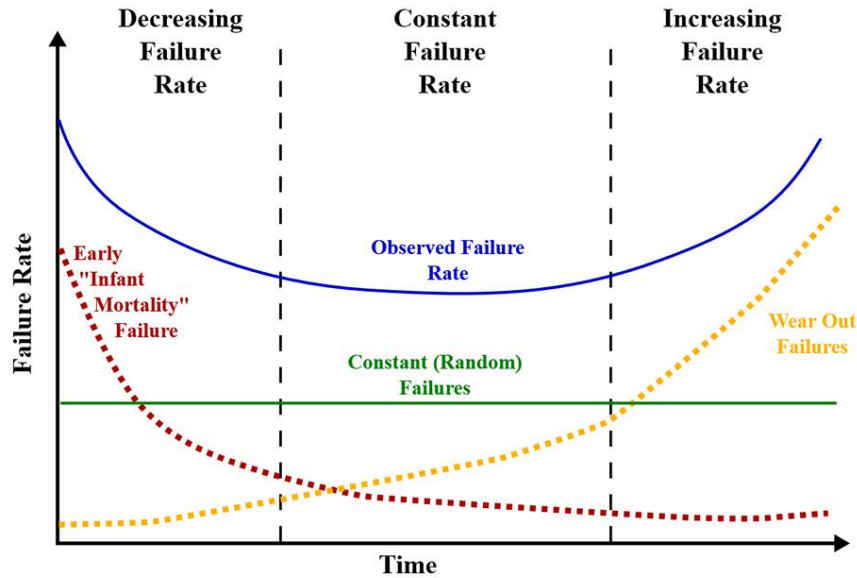


Figure 3. 6 The ‘Bathtub curve’ of failure rate (Croarkin et al., 2006).

Hence, a consistent application is needed to evaluate the health state of how degraded the system is. [Figure 3. 7](#) illustrates the trend and states of a degradation level. In the illustration, the system degraded states have two thresholds: one is alert, one is the alarm, three phases: healthy, degraded, failure. Phase 1, the system is in the health state, where the degradation level is from HI_0 to HI_1 . In this phase, the system is regarded in the health state. Phase 2, the system is slightly degraded, some of the failures start to show up, but the majority of failure is still not revealed. In this phase, the degradation level reaches the HI_1 -alert level. Phase 3 the system is severely degraded; the majority of failures are within this period. Moreover, it passed the alarm threshold – HI_2 , which means the system is going to fail.

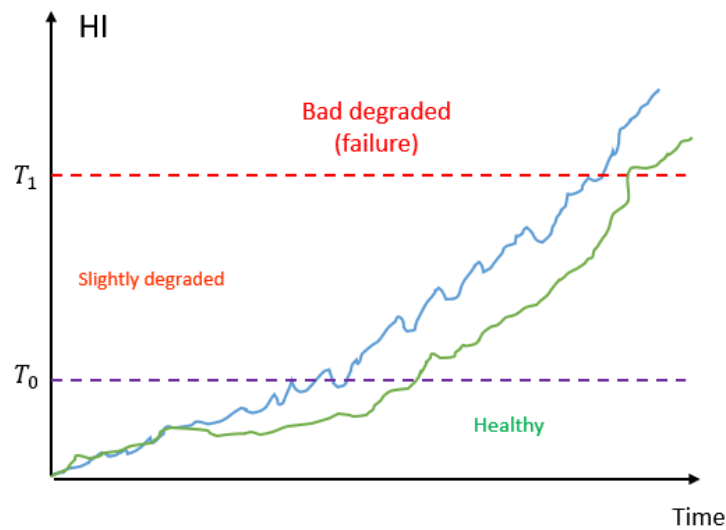


Figure 3. 7 Trend and states of the degradation level

In this concept, we need to decide the HI threshold in terms of categorizing the system degradation level. However, pre-set the threshold is a bit challenged, which needs a complete understanding of system behavior and operational information. The historical system data is essential here to address the system state, which provides a reference to set up the threshold ([ISO, 2019](#)).

Diagnostics

Diagnostics is a part of state identification, but with more aspects. It mainly focuses on faulty or failure with a high degraded level. Generally speaking, diagnostics performs high detection of faulty state when the system still can operate, and with the lower false alarm rate ([Atamuradov et al., 2017](#)). The main task of diagnostics is to 1. detect the fault, which indicates the abnormal performance of the system; 2. isolate the fault, which is to address the system component problem; 3. identify the fault, which is the root cause of this failure ([Janasak and Beshears, 2007](#)). Diagnostics can be classified into four categories:

1. General inspection;

The general inspection is primarily related to the sensory inspection and functional test, which is conducted on accessible equipment by operators. Usually, the frequency is

CHAPTER 3: PREDICTIVE MAINTENANCE AND DIGITAL TWIN

high, which might be hourly or daily. The general inspection can reveal 40% in the early stage of faulty.

2. Detailed inspection and detection;

Detailed inspection and detection are conducted by the expertise or maintenance group. During the operation, some of the tests cannot be carried out by operators, such as voltage examination, equipment decomposition.

3. Offline equipment evaluation;

Offline equipment evaluation is based on specific physical properties (temperature, vibration, sound, etc.) to determine the degradation of the system. To some extent, offline equipment evaluation is conducted with a schedule. It can be regarded as preventive maintenance, which does not reveal the failure or faulty directly.

4. Realtime data monitoring.

Realtime data monitoring is widely applied nowadays, which provides the opportunity for acquiring the functional feature online. From analyzing those features and dynamic updating, it can indicate the system statue and degradation state.

([Frangopol, 2011](#))

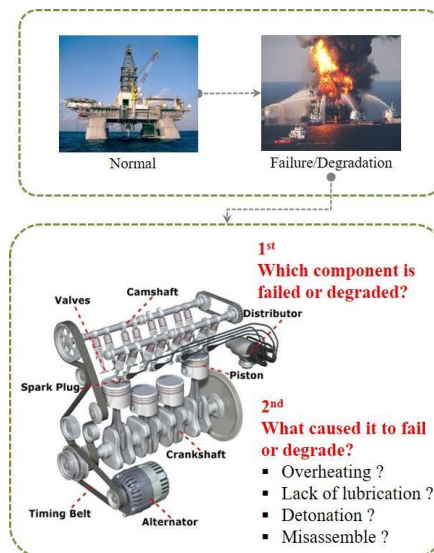


Figure 3. 8 The information provided by diagnostics ([Frangopol, 2011](#))

Diagnostics provides information about which components failed or degraded and what reason cause this situation, which helps to predict the further potential degradation or

failure. ([Janasak and Beshears, 2007](#))

Prognostics

Prognostics focuses on the prediction of the system state, in which the system is no longer functional, or it reaches the maintenance threshold. Typically, through comparison between the operational condition with the health indicator, the deviation shows the deterioration of the system with the time scale. Further, according to this mechanism, prognostics could provide a time estimation of future performance ([Pecht, 2009](#)). Within the scope, it is crucial to collect system information for the faulty and failure mode(including the abnormal signal, system behavior, and reason for the phenomenon) ([Atamuradov et al., 2017](#)).

There are three approaches to realize prognostics: 1.data-driven prognostics; 2.model-based prognostics(statistical approaches); 3.hybrid prognostics.([An et al., 2013](#))

Data-driven prognostics applies the historical data (training data) to determine the current state of the system and predict the future trend. The principle is only focusing on the data through the whole operation period and ignoring the system architecture and physical meaning. This method applies to the complex system, or the dataset is not suitable for model-based prognostics ([Barros, 2019](#); [An et al., 2013](#)). Artificial intelligence approaches and fuzzy logic are the typical methods of data-driven prognostics. The prediction uncertainty is estimated by a validation data set ([Barros, 2019](#)).

Model-based prognostics usually combines physical models and degradation level to estimate the RUL of the system. The model-based prognostics applies several physical considerations and system monitoring variables into a mathematical model. However, this process might reduce the accuracy of the degradation model, especially when the system becomes complex. So, the model-based prognostics typically perform on components level or damage propagation. ([Mosallam et al., 2013](#); [Barros, 2019](#)). Statistical approaches are the classical method of model-based prognostics, which can estimate the RUL along with probability. Classical model-based prognostics methods are trend models, time series, and stochastic processes. ([Barros, 2019](#))

Hybrid prognostics takes the advantages of both data-driven and model-based. In the practical, the data-driven method needs historical data (i.e., training dataset) to obtain

the degradation trend and threshold. However, some of the system failures are not frequent. It is hard to get several historical data. Hence, the hybrid prognostics method combines statistical approaches with data-driven methods ([Pecht and Jaai, 2010](#)).

Health indicator (Health index)

The implement of prognostic needs evaluation scale that identifies the health condition of the system operating condition. This evaluation scale is called a health indicator or health index. There are two kinds of health indicators: Physics Health Indicator (PHI) and Virtual health indicator (VHI). The PHI is related to the physical phenomenon, such as the vibration of bearing ([Mosallam et al., 2015](#)), and the temperature of the lithium-ion battery. This implement needs acquired signals related to system degradation level, which depends on system complexity and difficulty of decomposing the system while analyzing. Hence, the VHI is available for those failures which are not directly to physical phenomenon. Multiple sensors information and data sources can merge into one-dimension health indicators, such as linear weight ([Bai et al., 2014](#)).

3.2.3.4 Decision making

In practical, some maintenance schedule is not a short-time plan. Due to the availability of resources and accessibility for physical assets, it is necessary to have a decision model to help engineers make the maintenance schedule. According to the marketing, environmental, and resource information, the decision digital twin could not only help engineers balance the cost and plans, but also provide several decisions with a different probability. Meanwhile, the decision is dynamic, which means as more real-time data collected, the prediction would be more accurate, and the decision could be corrected along with the time ([Seyr and Muskulus, 2019](#)). The decision-making methods are various. In Liu et al. , the author states that the inventory of spare parts is one critical influence factor in predictive maintenance arrangement ([Liu et al., 2013](#); [Liu et al., 2018](#)). Bousdekis et al. propose a decision model based on economic loss when the system in different deterioration states ([Bousdekis et al., 2018](#)). In general, the decision model should be established based on real need and loss when the system failed.

Chapter 4

Data analysis and processing methods

In this chapter, the data analysis and processing methods are presented. The purpose of these methods is to analyze historical data to establish the offline reference model for the digital twin.

4.1 Dimension reduction

Data collected from monitoring usually contains multi-dimensional time-series signals. Thus, it is essential to compress the signals into one phase or select the most representative ones. The principal component analysis (*PCA*) can help to reduce the dimension ([Mosallam et al., 2016](#)). The mechanism of *PCA* relies on the linear algebra of several dimensions, which could transfer the data to a new coordinate system. The purpose of *PCA* is to find the most significant variance in the new coordinate system. The first principle is to choose the direction with the most significant variance in the original data. The second principle is to choose the orthogonal axis with the first principle component, along with the most significant variance in this coordinate system, and so on. The axis with the most significant variances is the principal component, as [Figure 4. 1](#) shows.

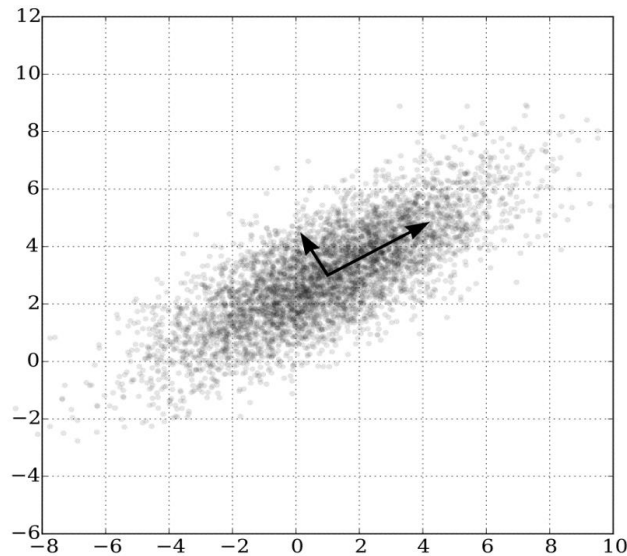


Figure 4. 1 The illustration of PCA

Assume, the data set is $X = \{x_1, x_2, x_3, \dots, x_n\}$, based on the eigenvalue decomposition covariance matrix or singular value decomposition (SVD):

1. Normalization or de-average;
2. Calculate the covariance matrix $\frac{1}{n}XX^T$;
3. Find the eigenvalue and eigenvector of the covariance matrix by using eigenvalue decomposition or SVD;
4. Sort the values from largest to smallest, and form a new matrix P
5. Convert the data in a new matrix from P and X , which is $Y = PX$

([Shlens, 2014](#))

After the dimension reduction, the multi-dimensional signal data could be merged into several principal components. The first principal component contains the most significant variance, which could represent the main feature of the dataset.

4.2 Time series decomposition

4.2.1 Time series patterns

The real-time data usually includes some signal errors and fluctuate. The disturbance

of the external environment and human interrupt usually makes sensor signal quality not that feasible to analyze directly. Generally, the real-time monitoring data is a time series, including a specified time step and corresponding values. There are three typical patterns in time series: Trend, Seasonal, and Cyclic. The trend is capable of a long-term increase or decrease in the data series. The trend does not need to be linear or specified functions. The seasonal pattern is that the time series follows a fixed frequency, such as daily, monthly, and yearly. It typically shows periodically trend. Cyclic applies when the data trend varying not follows a specified frequency, which seems more random.

4.2.2 Time series components

Real-time data could be a combination of those patterns. When it comes to the analysis, time series decomposition could split a real-time series into trend-cycle (trend), seasonal, and residuals, these three components. Let's S_t denote seasonal component, T_t is trend-cycle components, and R_t is the residual components, y_t is the value of the signal, at time t . The additive decomposition 4.1 and multiplicative decomposition 4.2 can be formulated as:

$$y_t = S_t + T_t + R_t \quad (4.1)$$

$$y_t = S_t \times T_t \times R_t \quad (4.2)$$

The additive is applicable when the magnitude of the variation does not fluctuate around the trend-cycle. Otherwise, it is better to use multiplicative decomposition.

[\(Hyndman and Athanasopoulos, 2018\)](#)

4.3 Pattern recognition approaches

Pattern recognition is a recognizing process that matches the information from input data to stored historical data. This method is widely applied in machine learning because of high efficiency ([Barros, 2019](#); [Pecht, 2010](#)). Pattern recognition can be done direct computation through machines, such as k-Nearest-Neighbor (k-NN), and Decision tree, or on bionics-related, such as the Artificial Neural Network (ANN) ([Venkatesan et al., 2019](#); [Jesan, 2004](#)). In this part, the neural network and k-NN are introduced.

4.3.1 K Nearest Neighbors

K Nearest Neighbors (k-NN) is a similarity approach according to entire training data and classification. Not like other methods, k-NN does not spend lots of time on training. The k-NN is a non-parametric method, which means the prediction only depends on the training data without learning any specific functions (Singh, 2018). Figure 4. 2 shows an example of $k - NN$ Regression

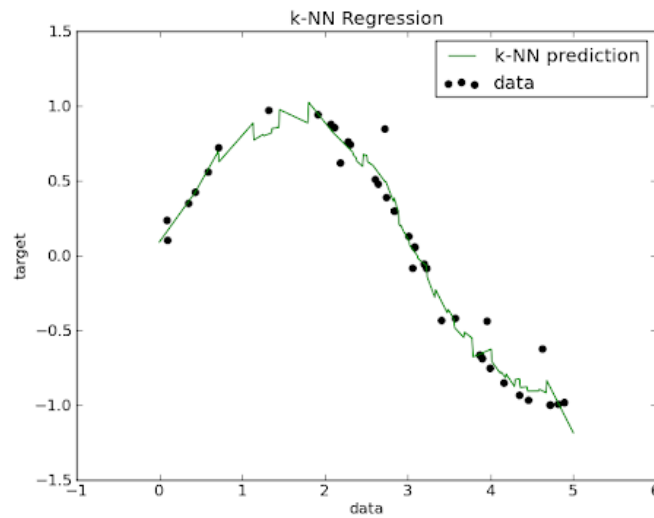


Figure 4. 2 k-NN regression example

4.3.1.1 k-NN prediction

The k-NN prediction is by searching for the k most similar neighbors, i.e., instances. For the distance measuring, three methods are widely used, Euclidean, Manhattan, Minkowski. The mathematical approaches and formulas of distance functions are showing in the following (Singh, 2018).

$$EuclideanDistance(a, b) = \sqrt{\sum_{i=1}^n (a_i - b_i)^2} \quad (4.3)$$

$$Manhattan(a, b) = \sqrt{\sum_{i=1}^n |a_i - b_i|^2} \quad (4.4)$$

$$Mkowski(a, b) = \sqrt[q]{\left(\sum_{i=1}^k (|a_i - b_i|)^q\right)} \quad (4.5)$$

These three methods are only valid for the continuous model. If there is NaN or null value in the model, it should be changed into a specific value.

4.3.2 Neural Network (ANN)

The artificial neural network is an information recognition and classification system, which the network through studying through input data to learn the mechanism and generate desired outputs. The linkage between input and output depends on how neural network architecture is built and what kind of activation function is chosen ([Atamuradov et al., 2017](#); [Venkatesan et al., 2019](#); [Jesan, 2004](#)). The neural network now has developed broad branches with multiple domains, such as the convolutional neural network (CNN), Long short-term memory (LSTM) ([Kim et al., 2016](#); [Pecht, 2009](#); [Atamuradov et al., 2017](#)).

4.3.2.1 Neural Network introduction

The primary component in the neural network is called nodes. Each node could be formulated into different functions. The nodes with the same function can be grouped, called layers. Typically, the layers can be categorized into three different types, which are the input layer, hidden layer, and output layer. In the NN, the input layer first collects the data from outside, then passes the data to the hidden layer(s). In the end, the data past from the hidden layer(s) to the output layer and output the data ([Kim et al., 2016](#); [Fortuner, 2019](#)).

[Figure 4. 3](#) shows a typical NN model with three layers. In this NN model, x_n , h_n and z represents the nodes respectively in the Input layer, Hidden layer, and Output layer.

Between each layer, Wx_{nn} and Wh_n represents weights, Bx_{nn} and Bz_n represents the biases.

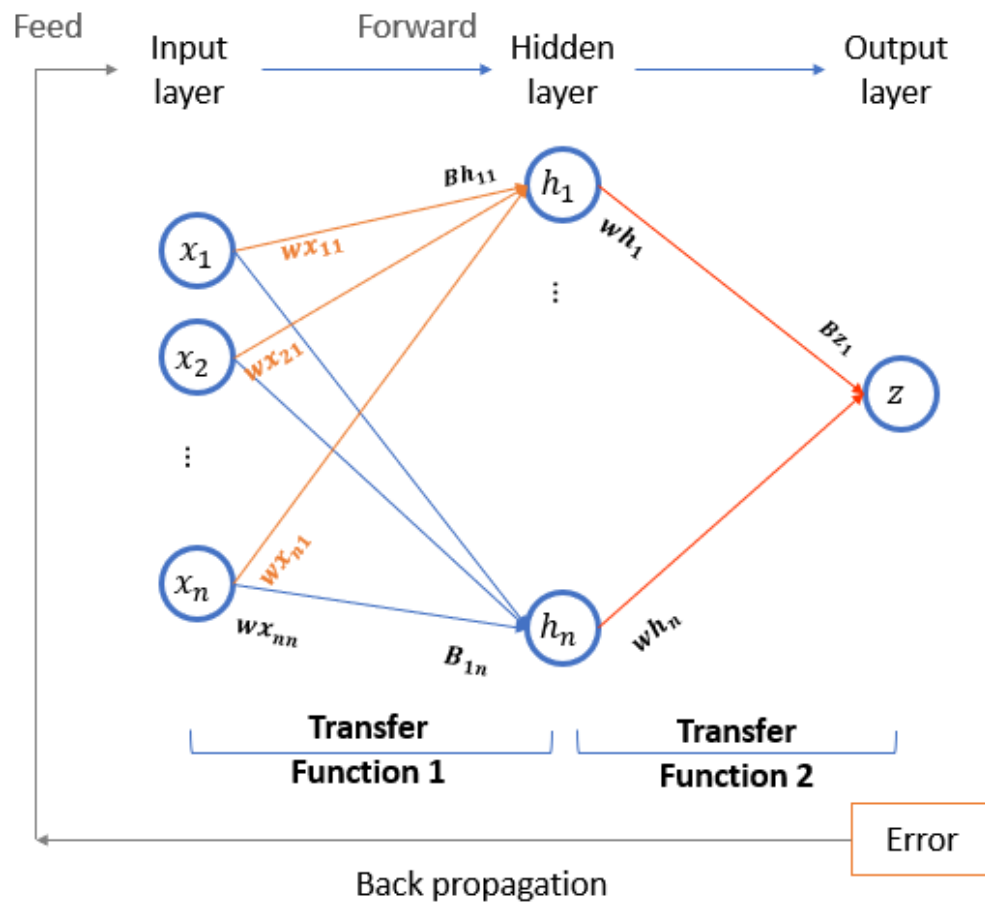


Figure 4. 3 Feedforward neural network framework

4.3.2.2 Neural Network mechanism

The primary process of NN is to estimate the value of W_s and B_s . From layer to layer, there are activation functions to determine the relations between input and output variables. In this case, the weights and biases from the input layer become the input of the activation function. The output of the activation function is distributed to hidden nodes in the hidden layer. Similarly, the hidden weights from the output of hidden nodes become the input of another activation function. After the activation function, the outputs are the final outputs in the output layer.

Typically, the weights and biases of the input nodes are formulated by a linear

combination, which is the input of the activation function in the next layer. The mathematical formula could be described as:

$$H = d_h(Wx_{nn} \cdot X + Bh_{nn}) \quad (4.6)$$

$$Z = d_z(Wh_n \cdot H + Bz_n) \quad (4.7)$$

Where: X is a matrix of training dataset with $i \times j$ data point, Wx_{nn} is the input weights for the activation function with $i \times k$ datapoints, Bh_{nn} is the hidden bias with $1 \times k$ vector, H is hidden nodes with a $k \times j$ matrix, d_h is the activation function between each layer, Wh_n is the hidden weight with $1 \times k$ vector, Bz_n is $1 \times j$ bias vector, and Z is the final output with j elements.

4.3.2.3 Activation function

The purpose of the activation function is to convert the input to the node and classify which node it should be located. After training, each node has a weight and bias. When the new feature comes to input, the activation function could decide which node should be activated or not. With the activation function, in principle, the neural network could learn and compute any function.

([SHARMA, 2017](#) ; [Missinglink.ai, 2020](#))

Typical activation Functions(commonly):

Binary step function

The binary step function is a classification function by evaluating the input with a specific threshold. The threshold could be modified and divided into several levels.

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \quad (4.8)$$

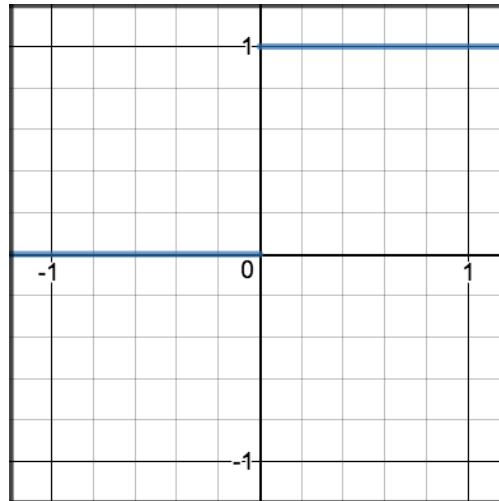


Figure 4. 4 Illustration of Binary step function

Sigmoid Activation Function

The sigmoid activation function is an ‘S’ shape curve with the limitation from 0 to 1. Thus, this function is used widely for prediction of probability. The shortcoming of this function is that the neural network will collapse if the input is a strong negative value.

$$s(z) = \frac{1}{1 + e^{-z}} \quad (4.9)$$

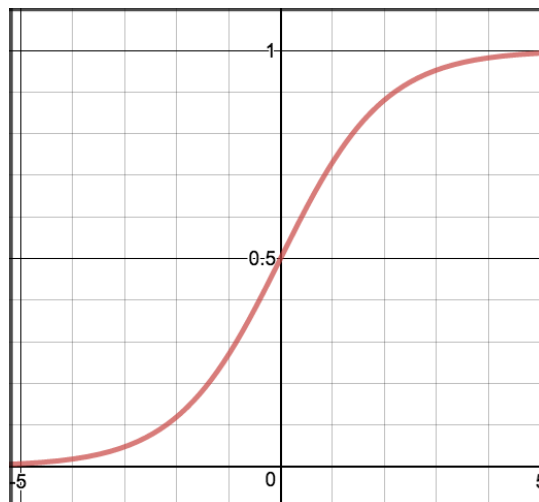


Figure 4. 5 Illustration of the Sigmoid Activation Function

Hyperbolic Tangent Function — (tanh)

Tanh is a non-linear function with real-valued range form (-1,1). Compared with Sigmoid function, Tanh can be used with strong negative input, and near-zero input

could compute near-zero output. It is more preferred than the sigmoid function.

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \quad (4.10)$$

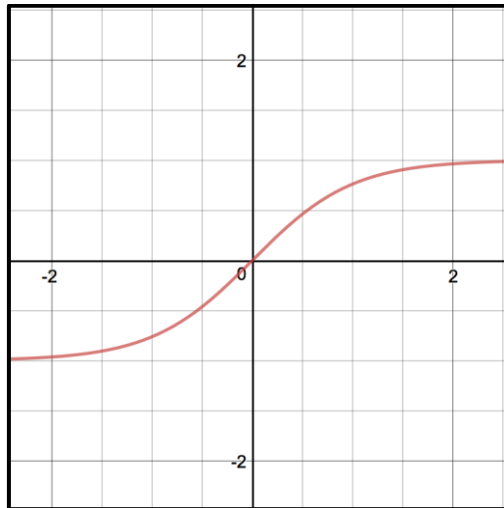


Figure 4. 6 Illustration of the Hyperbolic Tangent Function

Rectified Linear Units — (ReLU)

ReLU is a function with a range from $[0, +\infty)$, which is the most used function in the hidden layers. ReLU corrects the vanish of the gradient problem, but it could blow up the activation. Moreover, because of the lower limit, the gradients might die during the training. Leaky ReLU was introduced. The parameter ‘a’ usually is around 0.01.

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x > 0 \end{cases} \quad (4.11)$$

$$f(x) = \begin{cases} ax & \text{for } x < 0 \\ x & \text{for } x > 0 \end{cases} \quad (4.12)$$

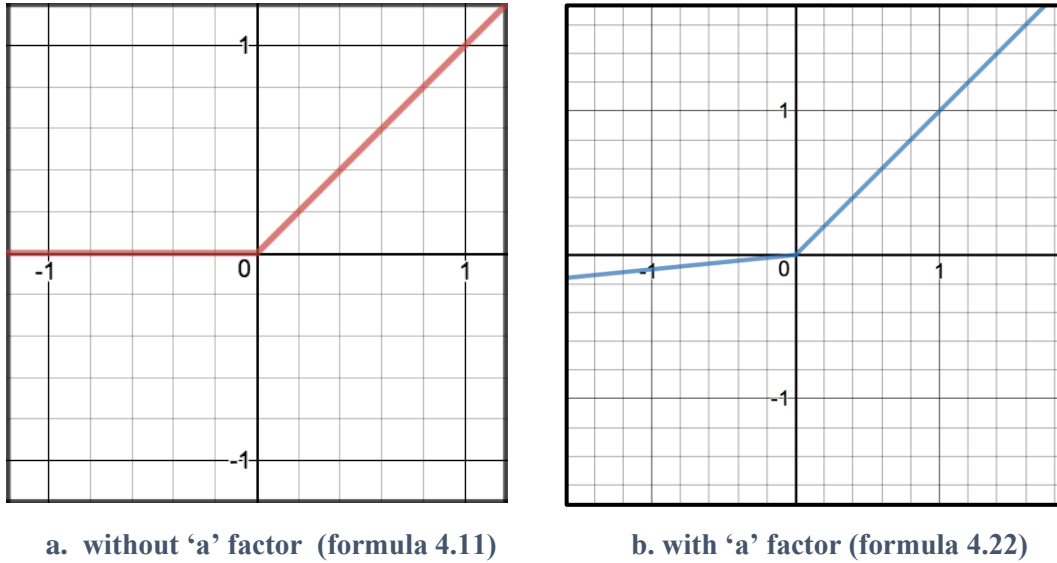


Figure 4. 7 Illustration of the Rectified Linear Units

4.3.2.4 The learning process and cost function

When the framework and activation function are selected, the next step is to train the neural network. During the training process, it is necessary to know when the training is complete and how successful it is. Thus, the cost function is established, which is based on the error value of estimation, such as mean squared error (MSE). The training process is to get the cost function as low as possible and update the weight, which also is called back-propagation.

([Chauhan, 2019](#).)

$$MSE = \frac{1}{j} \sum_{j=1}^j [y_j - z_j(W_x, B_h, W_h, B_z)]^2 \quad (4.13)$$

The optimizers are used to speed up the reduction of losses process and find the optimal training the model with different weights and biases. Among all of the optimizers, Adam spends less time and more efficiently, which provides an adjusted learning rate for each parameter. The mathematical principle works as follows:

$$v_{aw} = \beta_1 v_{aw} + (1 - \beta_1) \frac{\partial J}{\partial w} \quad (4.14)$$

$$S_{dW} = \beta_2 S_{dW} + (1 - \beta_2) \left(\frac{\partial J}{\partial W} \right)^2 \quad (4.15)$$

$$v_{dW}^{corrected} = \frac{v_{dW}}{1 - (\beta_1)t} \quad (4.16)$$

$$S_{dW}^{corrected} = \frac{S_{dW}}{1 - (\beta_1)t} \quad (4.17)$$

$$W = W - a \frac{v_{dW}^{corrected}}{\sqrt{S_{dW}^{corrected} + \varepsilon}} \quad (4.18)$$

The v_{dW} is the past gradients with an exponentially weighted average, while S_{dW} is corresponding to the square of the past gradients. $v_{dW}^{corrected}$ and $S_{dW}^{corrected}$ are the error correction of the bias. W is the weight matrix, and undated from average calculations.

([Doshi, Jan 13, 2019](#))

4.3.3 Stochastic process (model-based)

The stochastic process is prevalent to model the deterioration of components, where the system deterioration is non-monotonic and with some randomness ([Zhang et al., 2018](#); [Le Son et al., 2013](#)). The stochastic process can link to probability theory, which combines statistical-based data-driven with uncertainty and health indicator ([Zhang et al., 2018](#); [Si et al., 2011a](#)).

4.3.3.1 Brownian motion and Geometric Brownian motion

The Brownian motion is also called the Wiener process. It is a continuous-time stochastic process with a stationary trend and independent increment ([Le Son et al., 2013](#); [Si et al., 2011a](#)). The equation of one-dimension Brownian motion is defined as the following:

$$Y(t) = ut + \sigma W(t) \quad (4.19)$$

Where, $W(0) = 0, W(s) - W(t) \sim N(0, s - t), (for 0 < s < t)$. This is a Wiener process undulating around zero, where the drift is:

CHAPTER 4: DATA ANALYSIS AND PROCESSING METHODS

$$Y(t) - Y(s) \sim N(u(s - t), \sigma^2(t - s)) \quad (4.20)$$

Geometric Brownian motion (*GBM*) is also called exponential Brownian motion. The increment of *GBM* is related to the current value. The stochastic differential equation (*SDE*) is:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \quad (4.21)$$

Where the W_t is a Wiener process, dS_t is the increment of a short-time-interval dt , μ is the percentage drift, and σ is the percentage volatility.

In order to find the probability when the stochastic process passes the threshold of a prognostic process, it is necessary to calculate the probability density function of the deterioration. Thus, we define the following equation:

$$Y(t + dt) = Y(t) + S_{y,t,dt} \quad (4.22)$$

Where $Y(t + dt)$ is the deterioration from $Y(t)$ after time dt , $S_{y,t,dt}$ is the random quantity in dt . Let $g(s)$ be the *PDF* of $S_{y,t,dt}$, $f(y)$ be the *PDF* of $Y(t)$. After considering the threshold of deterioration L , we can get:

$$f(t|d + dt) = \int_{y-L}^{\infty} f(y - s|t) \cdot g(s) ds \quad (4.23)$$

The failure probability density function:

$$F_T(t) = Pr(T \leq t) = 1 - \int_{-\infty}^L f(y|t) dy \quad (4.24)$$

Chapter 5

Digital twin framework and architecture

In this chapter, a digital twin framework for predictive maintenance is presented. There are three parts in the digital twin, which are a communication protocol, PHM, including data-driven and prediction method, and strategic and decision making.

The digital system (digital twin) mainly divided into three parts:

- Communication protocols (models)
- Machine diagnostics and prognostics models
- Decision models

Communication protocols (models) perform communication between cloud storage or local storage and different machines. The communication includes data flow (inlet and outlet), data file transfer, and terminal control.

Machine diagnostics and prognostics models execute system diagnostics and prognostics algorithms and prediction of RUL. These processes are based on the historical data from the cloud storage or local storage and real-time data from machines. The predictions could pass to the decision models to execute further functions.

After the predicted RUL and some environmental information passed to decision models, the decision models could provide the cost of maintenance behavior and update in real-time joint with a schedule.

5.1 Communication protocol

The communication protocol is based on the Transmission control protocol (TCP) ([Foundation, 2020](#)). In this thesis, Sockets programming is used to present the connection and communication between the physical part and the digital part. The

CHAPTER 5: DIGITAL TWIN FRAMEWORK AND ARCHITECTURE

sockets are a client and server interaction, which typically has clients and servers on both sides. In sockets, the Client's machine could connect with the servers via IP address. The workflow of Sockets shows in the following:

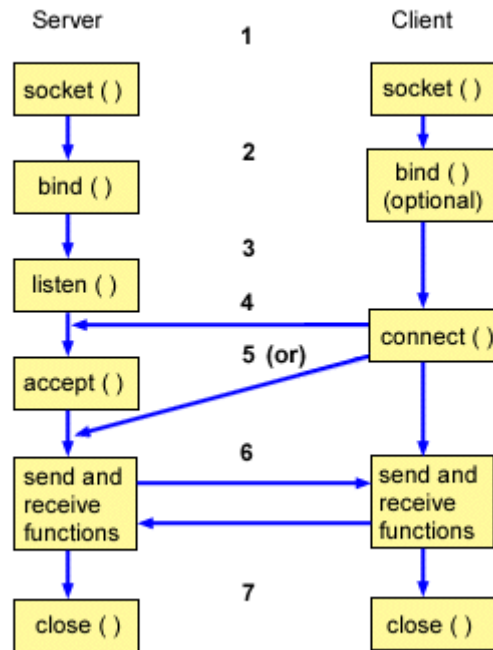


Figure 5. 1 The Socket communication workflow

The primary steps of setting up the Socket is:

1. The socket protocol firstly is set up in the server and client machine, which are connected by WLAN or LAN with descriptor;
2. The descriptor could provide a label to each Client with a bind () name to get access from the network;
3. While the server set up, the listen () API will indicate the request-listen (), from the Client. Then after approved- accept () by Sever, the connection is set up.
4. When the connection is established, the clients and server could transfer stream data by data transfer APIs, such as to send () and receive ().
5. When the stream finishes, the API close could stop the service for both sides.

5.2 PHM frame and method

The online and offline framework, according to Mosallam et al. ([Mosallam et al., 2016](#);

[Mosallam et al., 2015](#)), contains three different methods, which are 1) similarity-based prognostics based on the a. $k - NN$ approach⁵, 2) Data-driven prognostics, which are b. Deep-learning based on Artificial neural network (ANN) ([Raman and Hassanaly, 2019](#); [Venkatesan et al., 2019](#)), and c. Stochastic process ([Atamuradov et al., 2017](#); [Le Son et al., 2013](#); [Zhang et al., 2018](#)).

5.2.1 Offline reference model

The offline reference model is aimed to extract the trend and health indicator for the prognostics, which is emphasized in Chapter 3&4. The health indicator should be extracted from training data that has already stored in the cloud or local mirror ([Pecht and Jaai, 2010](#)). The historical data includes time-to-failure trajectories and multiple sensors or labeled data ([Kim et al., 2016](#)). The primary step of construct the offline reference follows the step below:

1. Data pre-processing: including data cleaning, check out missing value, find the features of the differently labeled data. The purpose of data pre-processing is to get a general impression of the whole dataset, minimize the prediction error caused by raw data.
2. Feature selection: Typically, for monitoring, not all of the data are informative from monitoring. Thus, it is essential to reduce the dimension of the dataset and compress the variables. Principal component analysis (PCA) introduced in Chapter 4, could merge the relevant data and get the merged or selected data set from the raw dataset ([Mosallam et al., 2016](#); [Mosallam et al., 2015](#)).
3. Trend extraction and noise elimination: The signal data generally is combined with noise, so it is essential to extract the primary trend of a time trajectory. In Chapter 4, the time-series component decomposition is introduced to extract the trend, which is presented in Hyndman and Athanasopoulos's book ([Hyndman and Athanasopoulos, 2018](#)).
4. Health indicator construction: After analysis of the primary trend, the health

⁵ The main process of k-NN model presented in this thesis follows main process of a case study in MATLAB. Further, the author of this thesis did some modifications according to the system. <https://se.mathworks.com/help/predmaint/ug/similarity-based-remaining-useful-life-estimation.html#SimilarityBasedRULExample-10>

CHAPTER 5: DIGITAL TWIN FRAMEWORK AND ARCHITECTURE

indicator could be selected as an evaluation standard. When real-time data collected from monitoring, there is a standard to classify the state of the system.

Noted that the ANN does not follow these steps, the primary process to establish the ANN will be discussed in the following chapters.

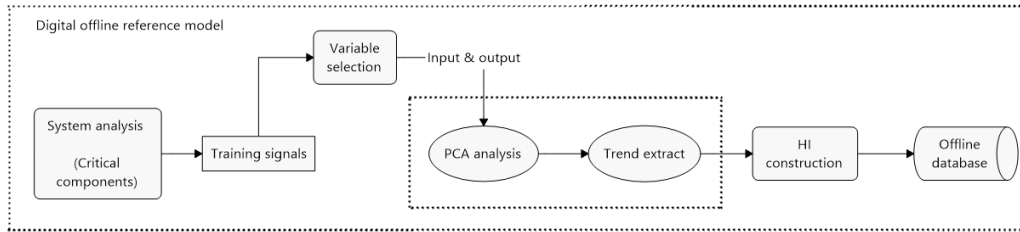


Figure 5.2 Overall scheme of offline reference model flow chart

5.2.2 Online prognostics model

In the online phase, new data is collected by sensors through the communication protocol, transferring to the data pre-process model. The data pre-process mode is to extract the same feature data as the reference model. Then according to the current data point, the prognostics processes are to perform RUL prediction. The prognostics can acquire through the model-based or data-driven method, which is introduced in Chapter 4. Then the estimated RUL will transfer through the decision-making model in real-time to help make maintenance decisions.

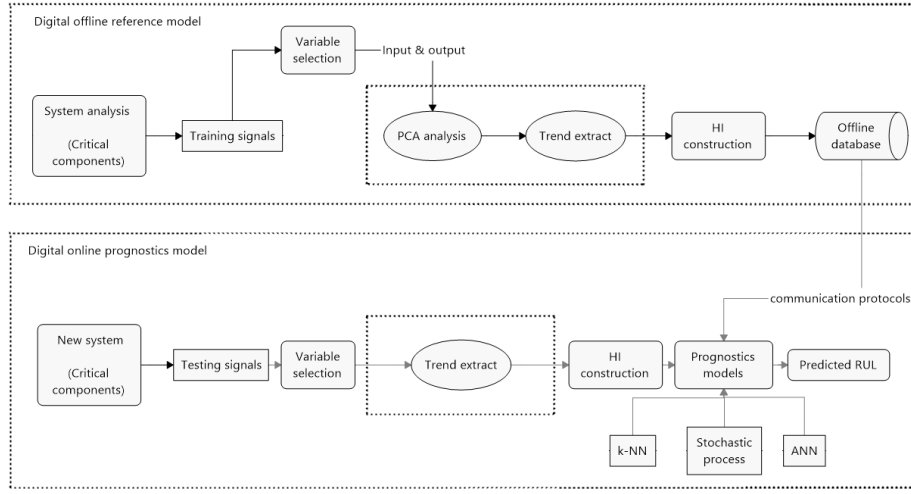


Figure 5. 3 PHM flow chart and process illustration

5.3 Decision making

The decision making is based on a simplified block replacement modeling (Formula 5.1) and static group maintenance (Formula 5.2). The basic idea is to calculate the cost per unit of time in order to find the best time slot for maintenance.

$$C_B(t_0) = \frac{c + kW(t_0)}{t_0} \quad (5.1)$$

Where: c is the preventive maintenance cost, k is the unplanned cost of system failure, $W()$ is the mean number of failures in the interval.

$$C(T_1, T_2, T_3, T_4 \dots, T_m) = \sum_{j=1}^m \frac{S}{T_j} + \sum_{j \in G_j} \left[\frac{C_i^p}{T_j} + C_i^u \lambda_{E,i}(T_i) \right] \quad (5.2)$$

Where, $C(T_1, T_2, T_3, T_4 \dots, T_m)$ is the cost at the time T_m , S is the set-up cost, G_j is a maintenance candidate group, C_i^p is the planned maintenance cost, C_i^u is the unplanned maintenance cost, $\lambda_{E,i}$ is an effective failure rate for component.

Chapter 6

Digital twin offline model

In this chapter, we are going to analyze the data from our system and establish the offline reference model. This process is the fundamental process in the digital twin establishment.

6.1 Data pre-processing and offline reference model

The following Chapter presents the establishment of a reference model for a digital twin, which provides an offline model to perform prognostics. The main idea of this offline reference model is based on a ‘Similarity-Based Remaining Useful Life Estimation’ published on MATLAB⁶ ([Wang et al., 2008](#)), a deep learning neural network, and the stochastic process, respectively. Besides, the open-source libraries and packages are presented to build this offline reference model.

6.2 Data pre-processing

Data pre-processing is aimed at process raw data and prepare the data for the next stage. It is not extracting features or data fusion. The data pre-processing could increase data quality and provide a better solution when it comes to analysis. ([Mosallam et al., 2015](#); [Mosallam et al., 2016](#)).

⁶. The main process of k-NN model presented in this thesis follows main process of a case study in MATLAB. <https://se.mathworks.com/help/predmaint/ug/similarity-based-remaining-useful-life-estimation.html#SimilarityBasedRULEExample-10>

6.2.1 Historical data description

As presented in Chapter 2, the historical datasets are uploaded to the cloud through the communication system. The machine starts operating from a healthy state and degraded during the operation. When the system runs to failure, the monitoring stops. There are 100 machines in the historical dataset, and each machine is independent identical operating during operation. The dataset includes 21 different monitoring signal data and three operational settings. Each row is taken by a time unit-cycle, which is regarded as the time scale. The columns correspond to 1. Machine serial number; 2. Time/Cycle; 3-5. Operational setting; 6-26. Sensor measurement.

Table 6.1 The detailed information of the dataset

| Symbol | Description | Unit of measure | Label |
|-----------|---------------------------------|-----------------|-------|
| T2 | Total temperature at fan inlet | °R | sen1 |
| T24 | Total temperature at LPC outlet | °R | sen2 |
| T30 | Total temperature at HPC outlet | °R | sen3 |
| T50 | Total temperature at LPT outlet | °R | sen4 |
| P2 | Pressure at fan inlet | psia | sen5 |
| P15 | Total pressure in bypass-duct | psia | sen6 |
| P30 | Total pressure at HPC outlet | psia | sen7 |
| Nf | Physical fan speed | rpm | sen8 |
| Nc | Physical core speed | rpm | sen9 |
| epr | Engine pressure ratio (P50/P2) | -- | sen10 |
| Ps30 | Static pressure at HPC outlet | psia | sen11 |
| phi | Ratio of fuel flow to Ps30 | pps/psi | sen12 |
| NRf | Corrected fan speed | rpm | sen13 |
| NRc | Corrected core speed | rpm | sen14 |
| BPR | Bypass Ratio | -- | sen15 |
| farB | Burner fuel-air ratio | -- | sen16 |
| htBleed | Bleed Enthalp | -- | sen17 |
| Nf_dmd | Demanded fan speed | rpm | sen18 |
| PCNfR_dmd | Demanded corrected fan speed | rpm | sen19 |
| W31 | HPT coolant bleed | lbm/s | sen20 |
| W32 | LPT coolant bleed | lbm/s | sen21 |

([Saxena et al., 2008](#))

6.2.2 Software and libraries for data pre-processing

The digital twin proposed is programmed in Python version 3.8. The algorithm will be present in Appendix C. Besides, there are extra libraries used to process the data;

- **Os:** Os function is to get access to the operating system dependent functionality. Usually, Os is related to file editing and transferring. In the digital twin model, Os performs read and updating files.
- **Pandas:** Pandas is mainly to structure data efficiently and intuitively. The data frame in Pandas contains two-dimensional and corresponding labels. For the Pandas data frame, the processing speed is faster than standard EXCEL and SQL in many cases.
- **Seaborn:** Seaborn is a Python visualization library based on matplotlib, which provides an API to apply for a statistical plot and integrated with Pandas data frame functionality.

6.2.3 Raw data pre-processing

A '.txt' format document collects the raw datasets. Based on the raw data structure and properties, the following steps need to be done for raw data pre-processing:

1. **Build data labels:** In the raw datasets, there is no index for each column and rows. In order to make data tight and easy to process in the following steps, it is necessary to add labels to the dataset;
2. **Diagnose data for cleaning:** In the monitoring, there could be some data missing and inconsistency. The missing value should be either replaced or removed.
3. **Data information:** After diagnosis, it is essential to get a general impression if how much data in the dataset, the data type, and anything wrong with the data frame established.
4. **Data category:** There are 100 machines data in the dataset, and each machine has the same label in columns. Thus, the data should be categorized by machine labels and time step.

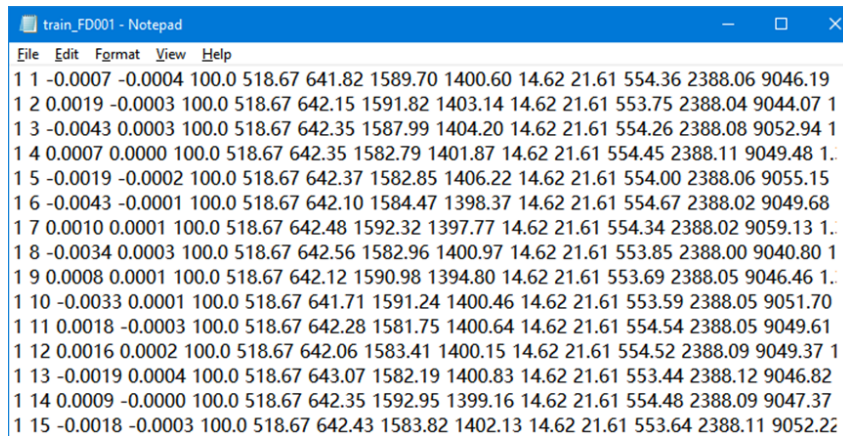


Figure 6.1 Illustration of the raw dataset

The raw data shows in [Figure 6.1](#). The label of corresponding data is added to the data frame. The data set is categorized by machine number and time series. The following figures show the total data frame of historical data and the data frame of the first machine.

| | time/cycle | op_cond_1 | op_cond_2 | op_cond_3 | ... | sn_18 | sn_19 | sn_20 | sn_21 |
|---------|------------|-----------|-----------|-----------|-----|-------|-------|-------|---------|
| machine | | | | | | | | | |
| 1 | 1 | -0.0007 | -0.0004 | 100.0 | ... | 2388 | 100.0 | 39.06 | 23.4190 |
| 1 | 2 | 0.0019 | -0.0003 | 100.0 | ... | 2388 | 100.0 | 39.00 | 23.4236 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 100 | 199 | -0.0011 | 0.0003 | 100.0 | ... | 2388 | 100.0 | 38.29 | 23.0640 |
| 100 | 200 | -0.0032 | -0.0005 | 100.0 | ... | 2388 | 100.0 | 38.37 | 23.0522 |

20631 rows × 25 columns

Figure 6.2 Illustration of labeled data

Missing data will influence the data processing procedure. [Figure 6.3](#) is the information of all data to count the missing value and invalid value. Besides, the memory usage of this data set is provided, which could help the company to decide the storage method.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20631 entries, 1 to 100
Data columns (total 25 columns):
time/cycle      20631 non-null int64
op_cond_1      20631 non-null float64
op_cond_2      20631 non-null float64
op_cond_3      20631 non-null float64
sn_1           20631 non-null float64
sn_2           20631 non-null float64
sn_3           20631 non-null float64
sn_4           20631 non-null float64
sn_5           20631 non-null float64
sn_6           20631 non-null float64
sn_7           20631 non-null float64
sn_8           20631 non-null float64
sn_9           20631 non-null float64
sn_10          20631 non-null float64
sn_11          20631 non-null float64
sn_12          20631 non-null float64
sn_13          20631 non-null float64
sn_14          20631 non-null float64
sn_15          20631 non-null float64
sn_16          20631 non-null float64
sn_17          20631 non-null int64
sn_18          20631 non-null int64
sn_19          20631 non-null float64
sn_20          20631 non-null float64
sn_21          20631 non-null float64
dtypes: float64(22), int64(3)
memory usage: 4.1 MB
```

There is no missing data or Null data

Figure 6.3 The information and features of all data

In this data set, there is no missing data or invalid data. Thus, the raw data set is prepared to continue further analysis.

6.2.4 Offline reference model establishing

6.2.4.1 Data fusion and extraction

In the monitoring, not all signals are capable of building a health indicator. The primary purpose is to select non-random relationships through all signals and get a *VHI* for the prognostics. To select such an indicator, the following steps and Python libraries are implemented:

Data fusion steps:

1. Variable selection: The variable selection is to filter the signals which do not relate to the degradation or has negligible influence. The most common method is to find Pearson's correlation coefficient- r_{xy} for all of the signals data.

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (6.1)$$

Where: x_i and y_i defines as two continues variables, i.e., signal data, \bar{x} and \bar{y} are the mean values of these variables. there is a strong relationship between x_i and y_i If r_{xy} is close to 1, vice versa. However, Pearson's correlation coefficient measures the linear correlation between two variables, which may not be efficient for the non-linear case. Hence, the statistic visualization should be performed as well, such as distribution, descriptive statistics.

2. Dimension reduction: In Chapter 4, the *PCA* method is introduced to implement in dimension reduction. The *PCA* could provide a compact data set for health indicator extract. ([Mosallam et al., 2015](#); [Mosallam et al., 2016](#))
3. Trend extraction: For a monitoring signal, due to the environmental disturbances and human activities, the signal might be varying and not efficient for health indictor formulating. In Chapter 4, the time series decomposition is introduced to reduce the noise error to obtain a clear and reasonable trend for health indicator establishment. ([Mosallam et al., 2015](#); [Hyndman and Athanasopoulos, 2018](#))

Python libraries:

1. Scikit-learn: Scikit-learn is an efficient package for data mining and data analysis. The priory functions in Scikit-learn are classification, regression, clustering, dimensionality reduction model selection, and pre-processing. Besides, Scikit-learn supports NumPy and SciPy, which means it has more compatibility and performs faster⁷.
2. Statsmodels: Statsmodels provides different functions for statistical model estimation and data exploration. The priory functions used in this thesis are Time series analysis and PCA. Statsmodels are based on NumPy, SciPy, and Matplotlib, which are advanced for statistical testing, modeling, and visualization.

The correlation heat map uses colored blocks to reveal the correlation coefficient, which

⁷ The methods and data processing precdure are following the packages in the websites below: <https://www.dataquest.io/blog/sci-kit-learn-tutorial/> ; <https://scikit-learn.org/stable/>

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

has the advantages of illustration intuition of the correlation coefficient. The correlation coefficient can help to find if there is a possibility to reduce the dimension of the historical data frame. [Figure 6.4](#) and [Figure 6.5](#), respectively show a correlation heat map of all monitoring data and one machine monitoring data.

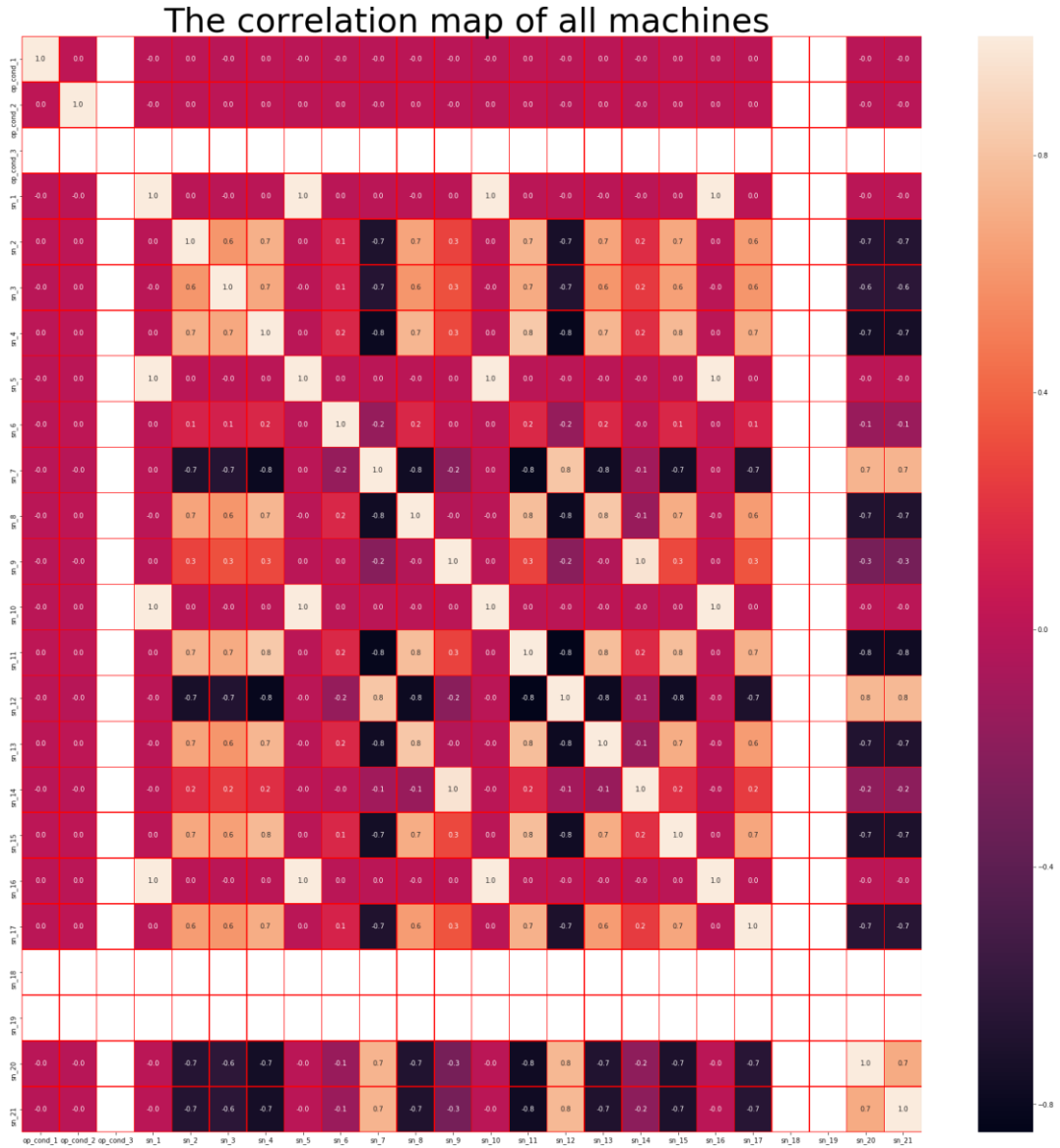


Figure 6.4 Correlation map all monitoring data for all machines

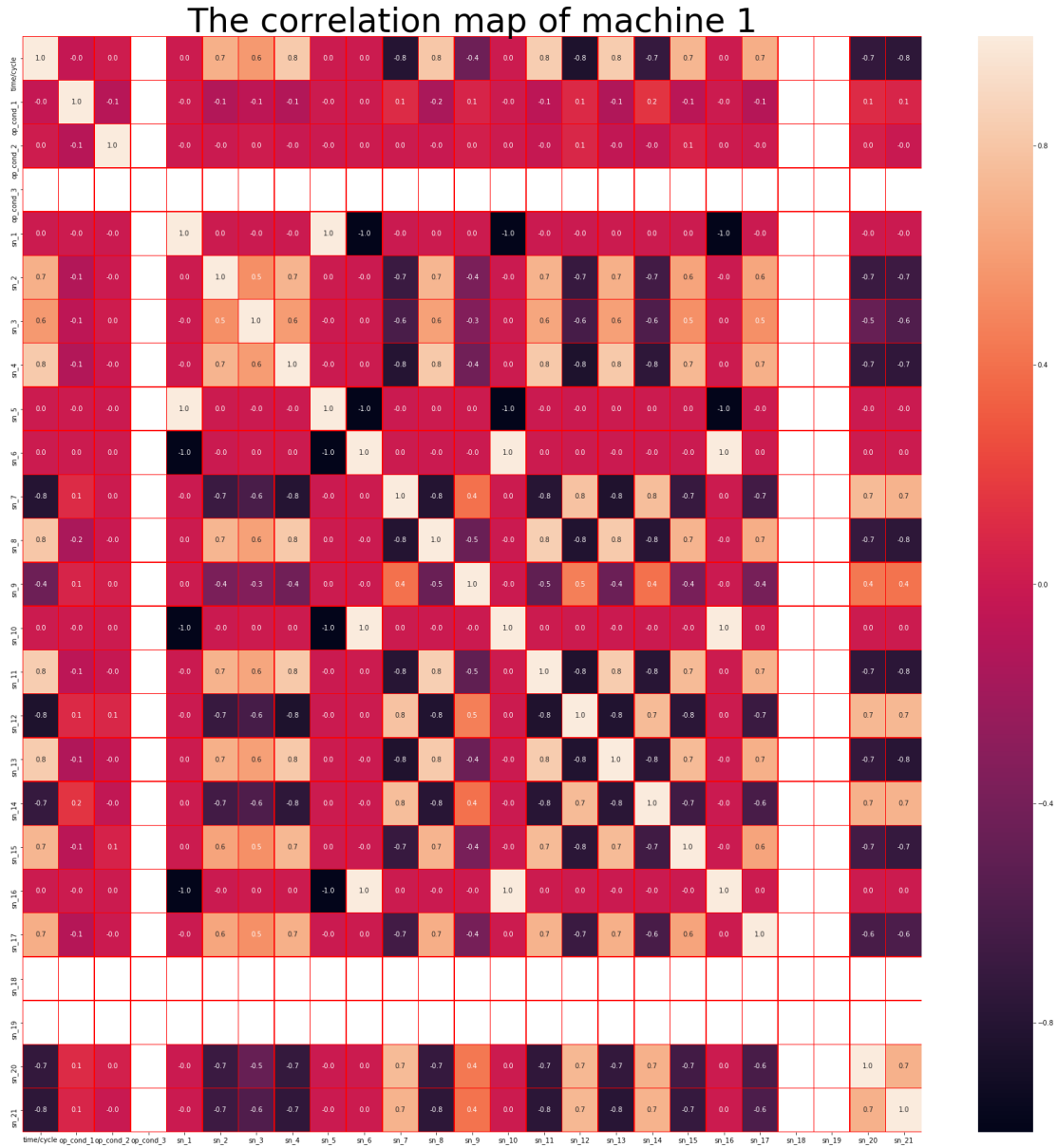


Figure 6.5 Correlation map all monitoring data for machine 1

In the figure, the color becomes darker, the higher relationship between the horizontal-axis variable and the vertical-axis variable. There are some missing values on Operation condition 3, Sensor 18, and Sensor 19. The reason might be that the value of these three does not change over time. Both from a single machine and all machine, it shows that some of the variables are highly correlated, which means those could be deleted or fused. The highly correlated pairs are shown in [Table 6. 2](#). Assuming when the correlation is higher than 0.9, we regard it as highly correlated.

Table 6. 2 High correlation coefficient values in the correlation map

| Sensor labels | Correlation coefficient |
|-------------------------|-------------------------|
| Sensor 1 and Sensor 5 | 1.0 |
| Sensor 1 and Sensor 10 | 1.0 |
| Sensor 1 and Sensor 16 | 1.0 |
| Sensor 5 and Sensor 10 | 1.0 |
| Sensor 5 and Sensor 16 | 1.0 |
| Sensor 9 and Sensor 14 | 0.96 |
| Sensor 10 and Sensor 16 | 1.0 |

In [Figure 6.6](#), the descriptive statistics show the information about 24 signal data. Some of the mean value and standard deviation (*std*) are rather small, which means the data of this signal is negligible.

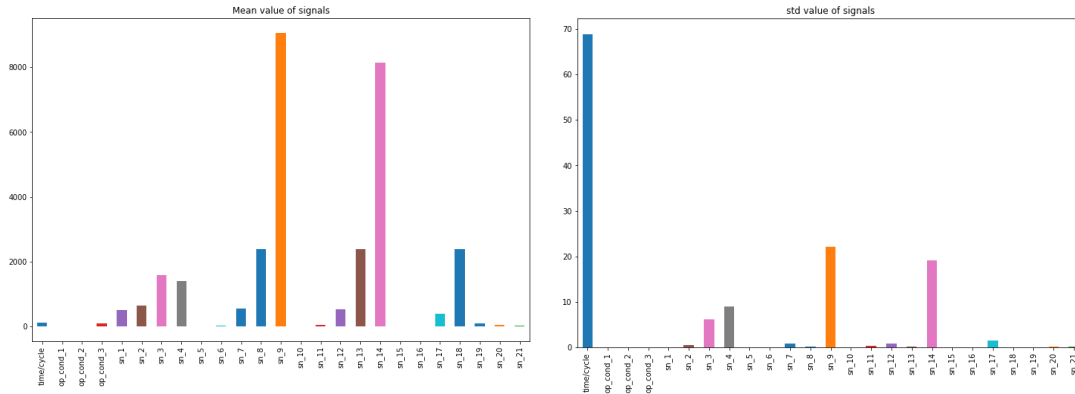
| | time/cycle | op_cond_1 | op_cond_2 | op_cond_3 | ... | sn_18 | sn_19 | sn_20 | sn_21 |
|--------------|--------------|--------------|--------------|-----------|-----|---------|---------|--------------|--------------|
| count | 20631.000000 | 20631.000000 | 20631.000000 | 20631.0 | ... | 20631.0 | 20631.0 | 20631.000000 | 20631.000000 |
| mean | 108.807862 | -0.000009 | 0.000002 | 100.0 | ... | 2388.0 | 100.0 | 38.816271 | 23.289705 |
| std | 68.880990 | 0.002187 | 0.000293 | 0.0 | ... | 0.0 | 0.0 | 0.180746 | 0.108251 |
| min | 1.000000 | -0.008700 | -0.000600 | 100.0 | ... | 2388.0 | 100.0 | 38.140000 | 22.894200 |
| 25% | 52.000000 | -0.001500 | -0.000200 | 100.0 | ... | 2388.0 | 100.0 | 38.700000 | 23.221800 |
| 50% | 104.000000 | 0.000000 | 0.000000 | 100.0 | ... | 2388.0 | 100.0 | 38.830000 | 23.297900 |
| 75% | 156.000000 | 0.001500 | 0.000300 | 100.0 | ... | 2388.0 | 100.0 | 38.950000 | 23.366800 |
| max | 362.000000 | 0.008700 | 0.000600 | 100.0 | ... | 2388.0 | 100.0 | 39.430000 | 23.618400 |

8 rows × 25 columns

Figure 6.6 Statistic counting for all monitoring data

To get a straightforward impression of the mean value and std in various signal data, we illustrate the mean and std values for each signal in [Figure 6.7, a.](#) and [b.](#)

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL



a. Mean values of signals

b. std values of signals

Figure 6.7 Mean values and standard deviation for each monitoring data

The illustrations indicate the variation and possible range of each signal. Some of the signals are not varying through the changing of time.

Table 6. 3 The values of Standard deviation and Mean

| | Standard deviation value | Mean value |
|------------------------|--------------------------|-------------|
| Operation condition 1 | 2.187313e-03 | -0.000009 |
| Operation condition 2 | 2.930621e-04 | 0.000002 |
| Operation condition 3* | 0.000000e+00 | 100.000000 |
| Sensor 1* | 6.537152e-11 | 518.670000 |
| Sensor 2 | 5.000533e-01 | 642.680934 |
| Sensor 3 | 6.131150e+00 | 1590.523119 |
| Sensor 4 | 9.000605e+00 | 1408.933782 |
| Sensor 5* | 3.394700e-12 | 14.620000 |
| Sensor 6 | 1.388985e-03 | 21.609803 |
| Sensor 7 | 8.850923e-01 | 553.367711 |
| Sensor 8 | 7.098548e-02 | 2388.096652 |
| Sensor 9 | 2.208288e+01 | 9065.242941 |
| Sensor 10* | 4.660829e-13 | 1.300000 |
| Sensor 11 | 2.670874e-01 | 47.541168 |
| Sensor 12 | 7.375534e-01 | 521.413470 |
| Sensor 13 | 7.191892e-02 | 2388.096152 |
| Sensor 14 | 1.907618e+01 | 8143.752722 |
| Sensor 15 | 3.750504e-02 | 8.442146 |
| Sensor 16* | 1.556432e-14 | 0.030000 |
| Sensor 17 | 1.548763e+00 | 393.210654 |
| Sensor 18* | 0.000000e+00 | 2388.000000 |

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

| | | |
|------------|--------------|------------|
| Sensor 19* | 0.000000e+00 | 100.000000 |
| Sensor 20 | 1.807464e-01 | 38.816271 |
| Sensor 21 | 1.082509e-01 | 23.289705 |

We assume that if the standard deviation is below $10e-10$, the sensor data could be regarded as a constant along with tie variation. In [Table 6. 3](#), the sensor label with ‘*’ could be regarded as a constant value, not change with time.

Thus, it is possible to filter out the none time-varying data to optimize the speed of processing. However, removing data from the historical data frame is risky, since the training and prognostics are all based on this data frame. In order to get a straightforward impression in which data could be deleted from the data frame, the following picture shows the distribution of sensor data.

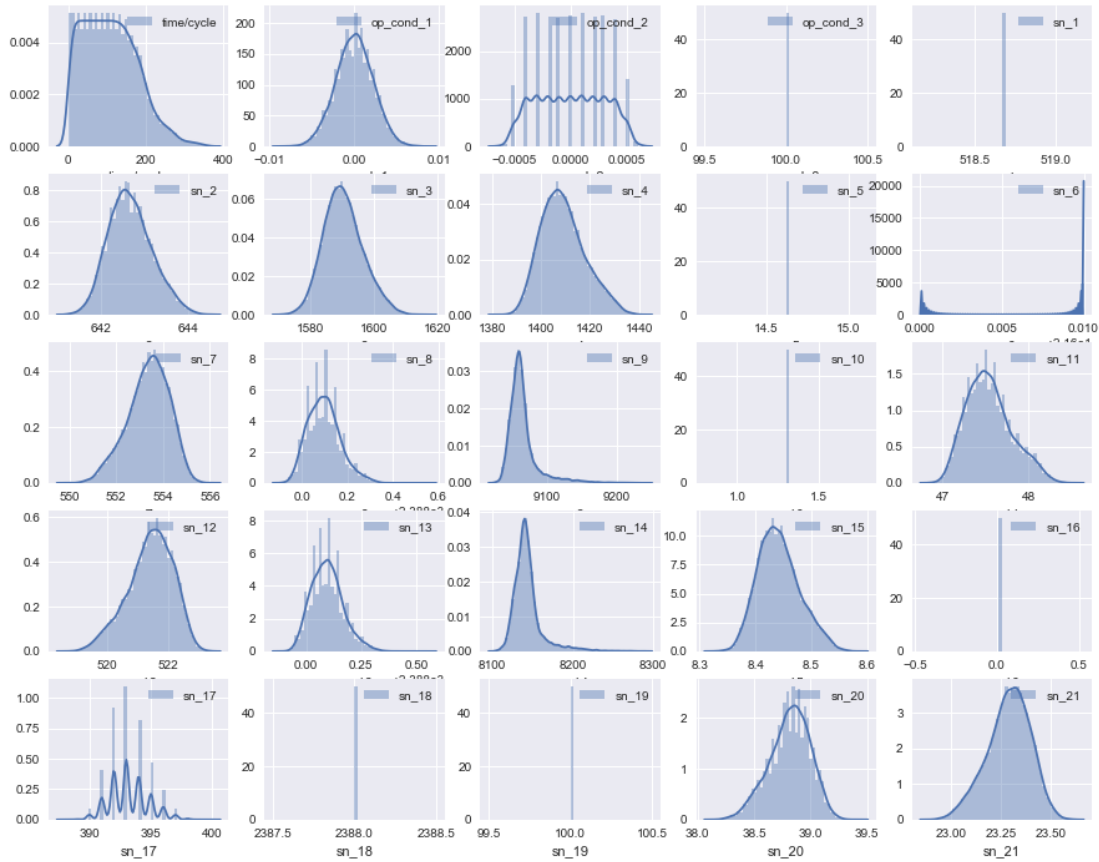


Figure 6. 8 Distributions of each monitoring signal

From this illustration, it is evident that some of the sensor data do not change along with time. According to correlation, descriptive statistics, and distribution of sensors for all of the machines in [Table 6. 4](#), the following sensors data could be considered as no effect on the model building, and these data could be removed from the data frame.

Table 6. 4 Correlation, descriptive statistics, and distribution of sensors for all machines

| | Correlation | Descriptive statistics | Distribution |
|---------------------|---|---|---|
| Data label | Sensor (1,5,10,16,18,19) Operation condition 3 | Sensor (1,5,10,16,18,19) Operation condition 3 | Sensor (1,5,6,10,16,18,19) Operation condition 3 |
| Data removal | Sensor (1,5,10,16,18,19); Operation condition 3 | | |

The Sensor 1,5,10,16,18,19 and Operation condition three are removed from the

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

original data frame since they do not change or has a negligible effect on the following analysis. The sensor 9 and 14 are highly related, which could be merged into one dimension in the following process. [Figure 6.9](#) illustrates sensor data with the lifetime for all of the machines after removing data. Some of the data shows a trend related to lifetime, which could be used in the health indicator construction. While some varying depends on different machines, such as sensor 9 and sensor 14, which could be removed from the data frame.

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

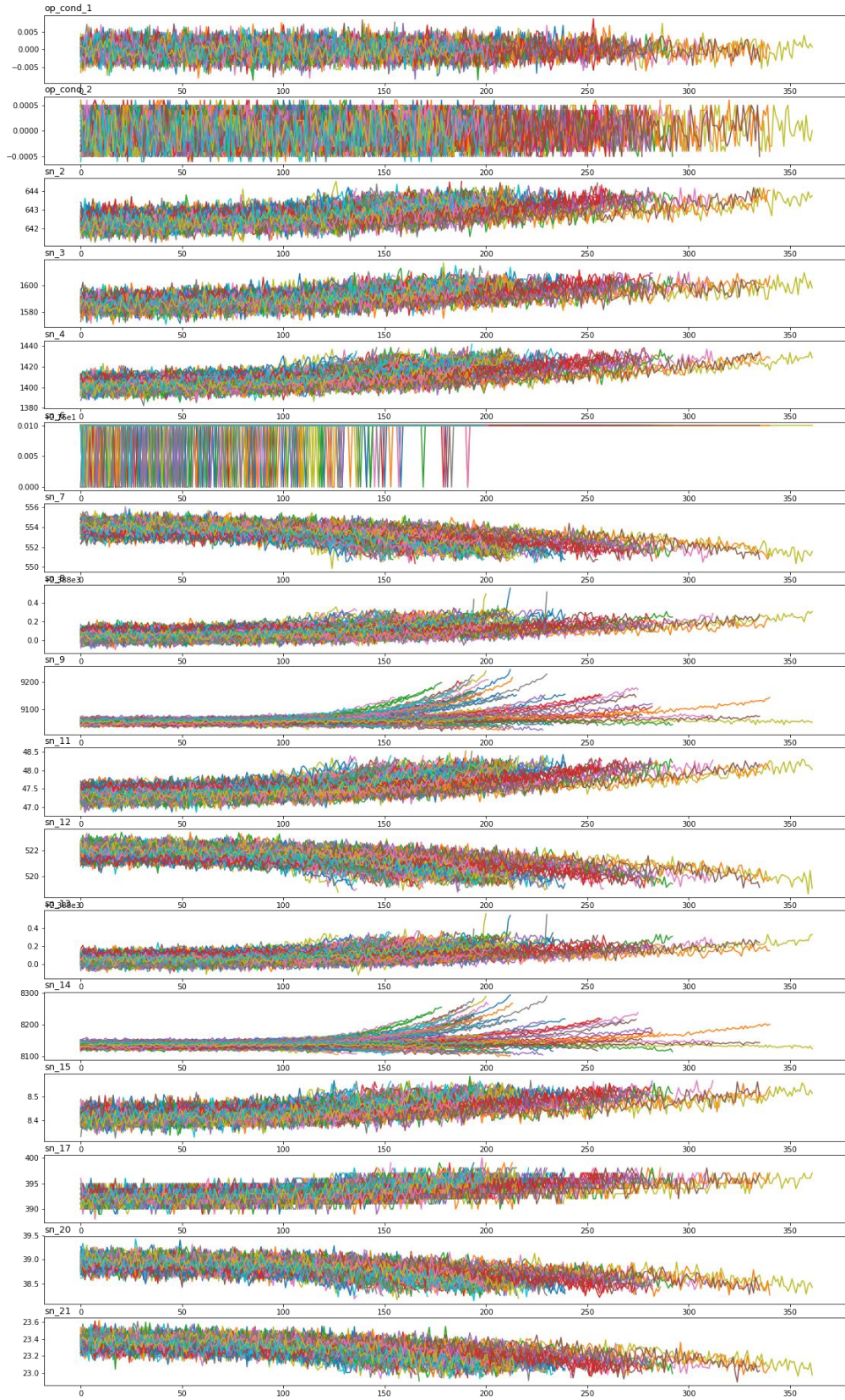


Figure 6.9 Remaining sensor data with the lifetime for all of the machines

6.2.4.2 Health indicator building

Trend extracting

From the illustration in [Figure 6. 10](#), operational condition 1 and 2 do not have a clear trend, and it shows a noise throughout the time scale. Sensor 9 and 14 also show trends. However, the trends depend on the machine, not on the deterioration level. In the following processes, these four signals could be discarded. Sensor 6 shows a discrete state, and some of the machines do have the signal for sensor 6. Hence it can be abandoned in the following analysis.

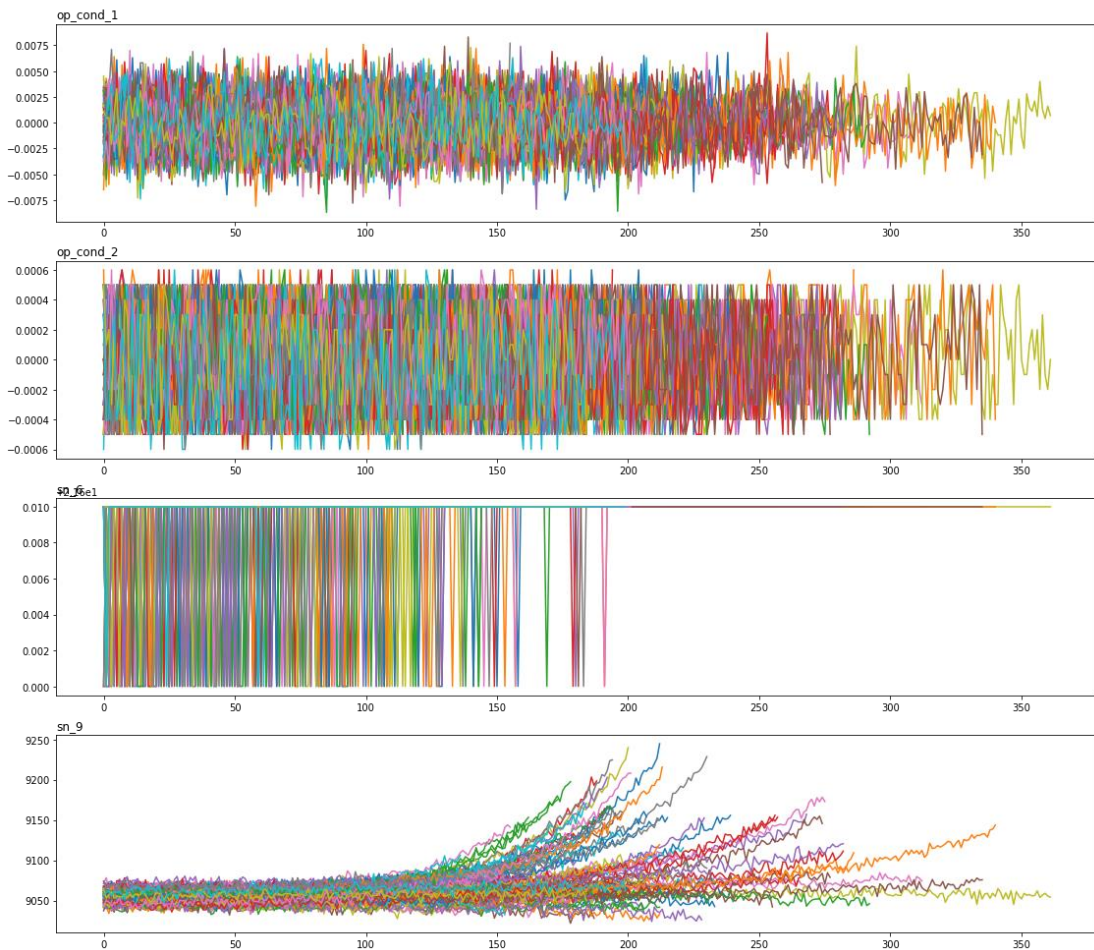


Figure 6. 10 Illustration of the sensor data will be removed

The linear slope could be an evaluation scale to acquire the most apparent trend throughout all the signals. [Figure 6. 11](#) shows machine 20 with the fitted linear trend.

Table 6. 5 shows the signal trends of all sensors and machines. From the illustration and table, all of the signals show an apparent trend, while different sensors might have a different trend.

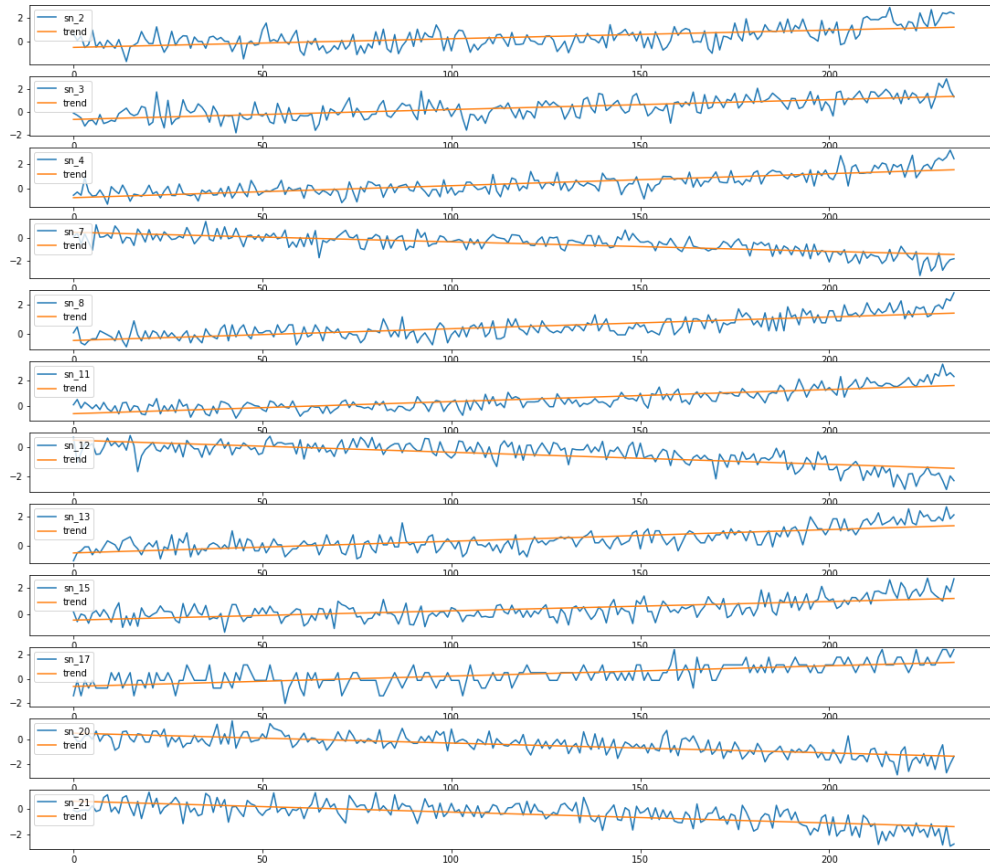


Figure 6. 11 Illustration of fitted Linear trend

Table 6. 5 Linear trend for sensors

| | sensor_2 | sensor_3 | sensor_4 | ... | sensor_17 | sensor_20 | sensor_21 |
|------------|----------|----------|----------|-----|-----------|-----------|-----------|
| 1 | 0.012146 | 0.009863 | 0.014113 | ... | 0.011758 | -0.012135 | -0.013337 |
| 2 | 0.010617 | 0.009526 | 0.010825 | ... | 0.009322 | -0.010204 | -0.009955 |
| 3 | 0.012130 | 0.014022 | 0.014576 | ... | 0.013707 | -0.012684 | -0.013178 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 98 | 0.017200 | 0.016361 | 0.019765 | ... | 0.015148 | -0.016696 | -0.017117 |
| 99 | 0.013823 | 0.011721 | 0.014742 | ... | 0.012696 | -0.012731 | -0.013448 |
| 100 | 0.011005 | 0.010159 | 0.012809 | ... | 0.010688 | -0.013099 | -0.011720 |

Throughout all of the linear trends, Figure 6.12 shows a primary trend. From the

illustration, the majority trends concentrate in one range, which indicates that all of the sensor trends could be merged into one dimension. In the following process, the PCA is to extract this feature.

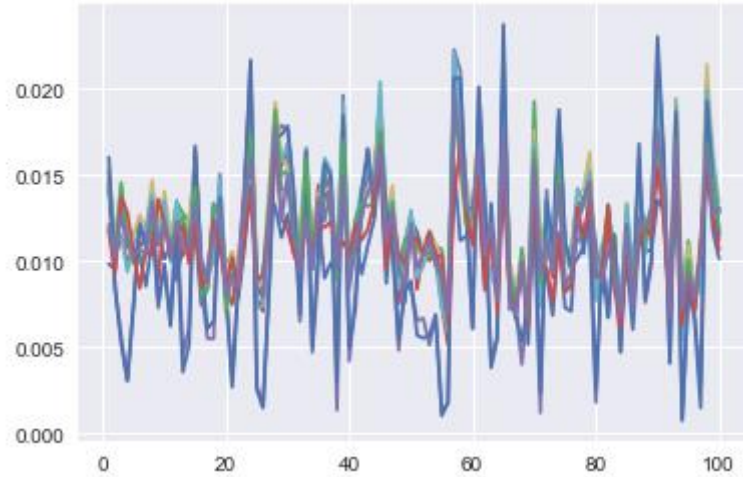


Figure 6.12 Illustration of linear trends

Dimension reduction

In ([Mosallam et al., 2015](#)), Mosallam et al. propose the principal component analysis method for variable compression. The raw data containing multiple sensors could be regarded as multi-dimensional data. The purpose of implementing PCA is to merge the multi-dimensional data into one-dimensional data space. The first principle component could represent the rest of the sensors, which contains the maximum of the variance ([Wold et al., 1987](#)).

The PCA calculation is through the Python package of Sklearn.decomposition. After PCA, the rank of the most apparent factors with variance ratio is listed in [Table 6.6](#).

Table 6.6 PCA variance values

| | | |
|------------|------------|------------|
| 0.74022293 | 0.04098082 | 0.0335441 |
| 0.0302408 | 0.02793587 | 0.02547959 |
| 0.0239673 | 0.01916276 | 0.01703485 |
| 0.01472286 | 0.01438374 | 0.01232441 |

The first principal component takes 74% of the total factors of influence. The first three principal components take higher than 90% representativeness of the 12-dimensional characteristics. The health indicator should be chosen from these three principal

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

components; however, after visualization in [Figure 6.13](#), the 2nd and 3rd are noisy throughout time, which are not accessible to extract the trend. Thus, the first principal component is taken as a health indicator.

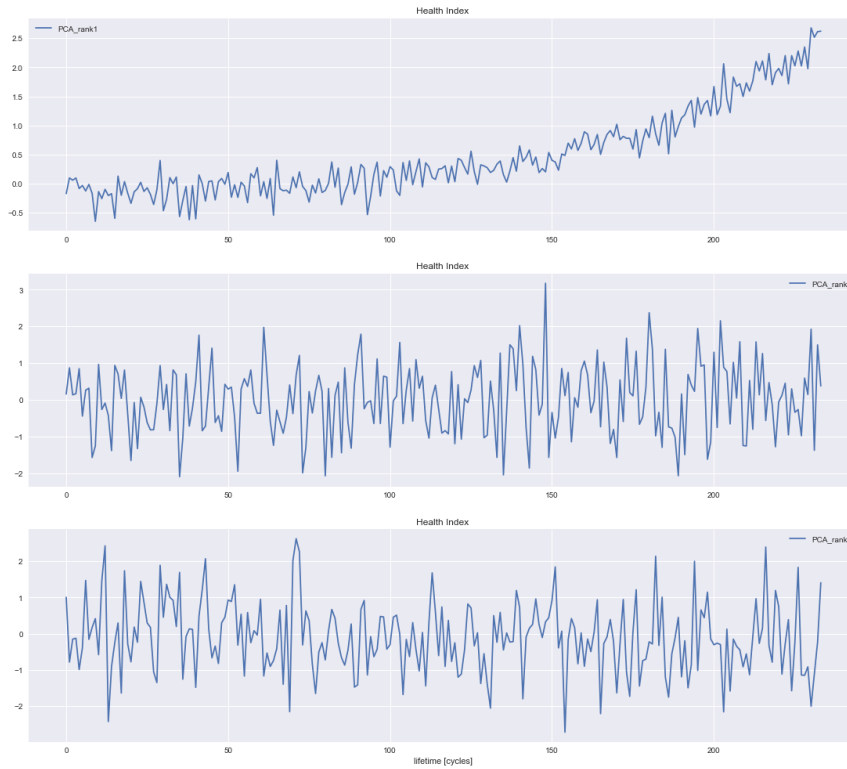
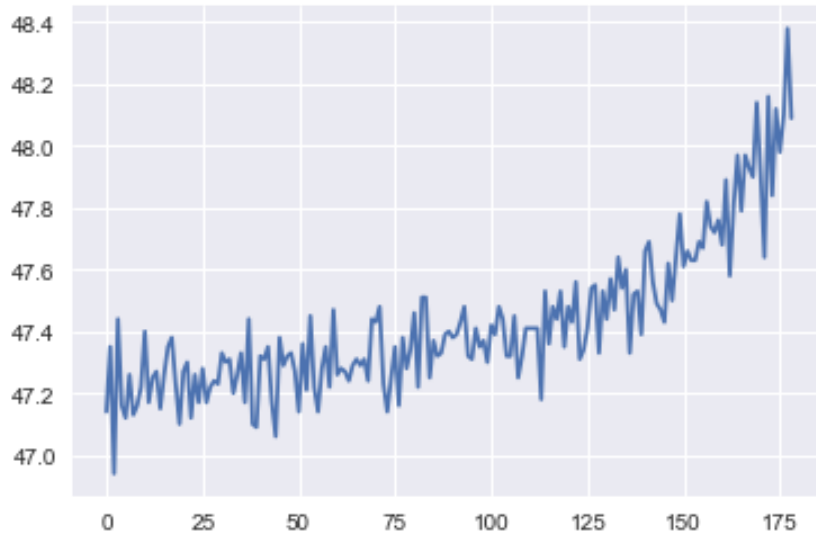


Figure 6.13 Illustration of first three PCA

The data put into PCA is standardized scaling. However, when it comes to the prognostics of real-time data, the scaling will change. Therefore, it is better to track back and using the original data for the health indicator establishing. By evaluation and weight extract, sensor 11 is the most evident; [Figure 6.14,b](#) shows all the rank of sensors.



a. Illustration of sensor 11 for machine 20

```
array(['sn_11', 'sn_4', 'sn_12', 'sn_7', 'sn_15', 'sn_21', 'sn_20',  
      'sn_17', 'sn_2', 'sn_3', 'sn_13', 'sn_8'], dtype=object)
```

b. The rank of all sensors according to the first principal component

Figure 6.14 Trend and information for HI candidate

In [Figure 6.14.a](#), the signal displays a noise around a stationary trend. The time series decomposition (TSD) is introduced in Chapter 4, which could help to extract the main trend of the noisy signal. After implement TSD, the trend, seasonal, and residual features could be acquired. [Figure 6.15](#) shows an example of machine 20.

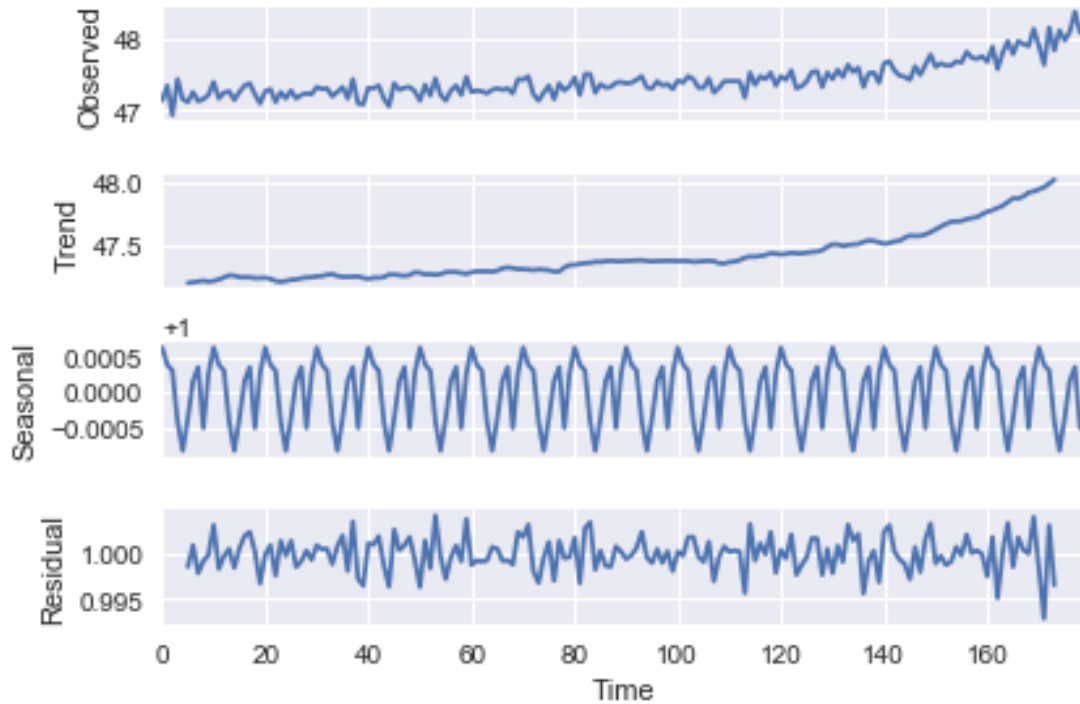


Figure 6.15 Time series decomposition for sensor 11 of machine 20

The TSD method is aimed to remove the stationary and regular noise to reveal the true trend of the signal. [Figure 6.16](#) shows the indicator trends of all the machines, after scaling the trend feature by subtracting the data from the minimum value of each machine.



Figure 6.16 Health indicator trend of all machines

Through previous steps, the health indicator data of each machine is collected and stored as data frame M , with corresponding input values X - health indicator and output values Y -lifetime.

Table 6.7 The dataframe of health indicator information and corresponding lifetime

| Label | Health indicator (X) | | | | | | Lifetime (Y) |
|-------|--------------------------|--------|--------|-----|--------|--------|------------------|
| | 0 | 1 | 2 | ... | 350 | 351 | |
| 1 | 0.0415 | 0.0100 | 0.0000 | ... | 0.9130 | 0.9130 | 191.0 |
| 2 | 0.0550 | 0.0635 | 0.0485 | ... | 1.0525 | 1.0525 | 286.0 |
| 3 | 0.0000 | 0.0040 | 0.0105 | ... | 0.8215 | 0.8215 | 178.0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 98 | 0.0000 | 0.0045 | 0.0290 | ... | 0.9470 | 0.9470 | 155.0 |
| 99 | 0.0520 | 0.0460 | 0.0435 | ... | 0.9345 | 0.9345 | 184.0 |
| 100 | 0.0680 | 0.0750 | 0.0820 | ... | 0.8035 | 0.8035 | 199.0 |

6.2.4.3 Offline reference model

The feature of deterioration is extracted from historical data in the data fusion. According to these historical data (M), the offline reference model could be established to evaluate the health state of a new machine in the future prognostics. In this section, three reference models are established according to the complexity and computational resource, which are similarity-based, deep learning neural network, and stochastic process.

Similarity-based model ($k - NN$ regression model)

The mechanism of the similarity-based model is based on k -NN regression ([Barros, 2019](#); [Wang et al., 2008](#)). The main steps for establishing the similarity-based reference model are:

a) Split the historical data frame:

To avoid the overfitting problem, we split the data frame (M_k) into two subsets, training and validation $\{M_k^T, M_k^V\}$. Meanwhile, M_k^V could help to determine the optimized k -value. The training data set takes 70% of the whole data frame. The

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

training data and validation data are:

$$M_k = \{M_k^T, M_k^V\}$$

each subset with the corresponding input and output are:

$$M_k^T = \{X_k^T, Y_k^T\}, M_k^V = \{X_k^V, Y_k^V\}$$

[Table 6.8](#) shows an example of the training set and validation set with input variables and output variables.

Table 6.8 The example of the training set and validation set with input variables and output variables

| | Label | Health indicator (X_k) | | | | | | Lifetime (Y_k) |
|--|-------|----------------------------|--------|--------|-----|--------|--------|--------------------|
| | | 0 | 1 | 2 | ... | 350 | 351 | |
| Training data set $\{X_k^T, Y_k^T\}$ | 6 | 0.0000 | 0.0215 | 0.0350 | ... | 0.9210 | 0.9210 | 258.0 |
| | 29 | 0.0475 | 0.0515 | 0.0450 | ... | 0.9225 | 0.9225 | 193.0 |
| | 55 | 0.0960 | 0.1015 | 0.1015 | ... | 0.5475 | 0.5475 | 274.0 |
| | ... | ... | ... | ... | ... | ... | ... | ... |
| | 47 | 0.1150 | 0.1040 | 0.0855 | ... | 0.7775 | 0.7775 | 230.0 |
| | 75 | 0.0210 | 0.0090 | 0.0000 | ... | 0.7370 | 0.7370 | 209.0 |
| | 80 | 0.0335 | 0.0380 | 0.0430 | ... | 0.7320 | 0.7320 | 239.0 |
| Validation data set $\{X_k^V, Y_k^V\}$ | 32 | 0.025 | 0.0185 | 0.0190 | ... | 0.9740 | 0.9740 | 199.0 |
| | 52 | 0.011 | 0.0000 | 0.0070 | ... | 0.6960 | 0.6960 | 194.0 |
| | 2 | 0.000 | 0.0040 | 0.0105 | ... | 0.8215 | 0.8215 | 178.0 |
| | ... | ... | ... | ... | ... | ... | ... | ... |
| | 68 | 0.078 | 0.0850 | 0.0875 | ... | 0.8605 | 0.8515 | 361.0 |
| | 10 | 0.023 | 0.0230 | 0.0115 | ... | 0.8740 | 0.8740 | 239.0 |
| | 19 | 0.093 | 0.0795 | 0.0670 | ... | 0.7065 | 0.7065 | 233.0 |

b) Set-up $k - NN$ regressor model:

In $k - NN$ regressor model, we use the Python package from sklearn. The basic parameter settings are:

- $n_neighbors$, will be optimized in step c)
- $weights = distance$, which allocates the weight by the inverse of 3 nearest neighbours distance;
- $p=2$, which applies Euclidean distance.

c) Identify optimized k by using root mean square error (RMSE):

The reference model is established by M_k^T . Then, we could get the estimated lifetime \widehat{Y}_k^V by fitting X_k^V to X_k^T and after, compare \widehat{Y}_k^V with true value Y_k^V by using RMSE. During the estimation, various k s are used to get different RMSEs. By selecting the corresponding k with the lowest RMSE, we could get the optimized k in the training process.

[Table 6.9](#) and [Figure 6.17](#) show the k values and the optimized k throughout previous steps. Among all of the k values, when k is equal to 3, the RMSE is the lowest. So, we choose 3 as k nearest neighbor.

Table 6.9 the k values with corresponding RMSE

| k-values | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|-----------------|----------|----------|----------|----------|----------|----------|----------|----------|----------|-----------|-----------|
| RMSE | 4.53 | 5.56 | 2.70 | 4.24 | 5.80 | 6.84 | 7.90 | 7.95 | 9.13 | 9.99 | 10.67 |

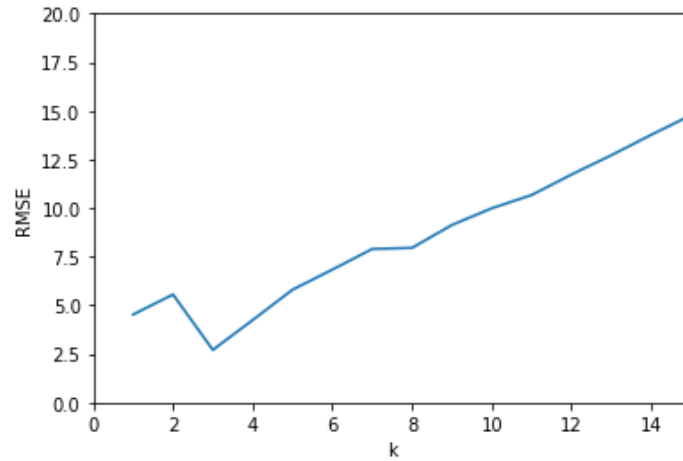
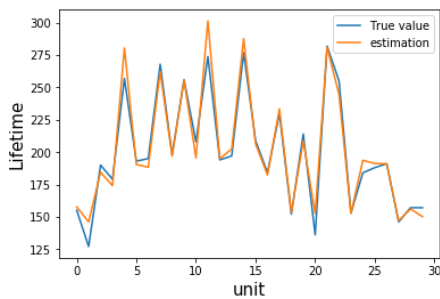


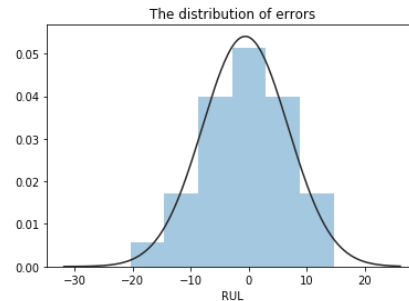
Figure 6.17 Illustration of RMSE

d) Validate the estimation and evaluate the model:

In this step, we change the n_neighbors to 3. Then, we fit X_k^V to X_k^T and get a new estimated lifetime, $Y_k^{V'}$. This procedure is achieved by ‘neigh.fit’ in the algorithm. The comparisons between true Y_k^V and estimated $Y_k^{V'}$ are shown in [Figure 6.18 a and b](#).



a. Comparison between the true value and estimation value



b. The distribution of errors

Figure 6.18 Illustration of the validation process and the performance

e) Find a suitable model:

In the validation process, the distribution of error could represent the performance of the $k - NN$ reference model. In order to obtain a relatively accurate model, we repeat the step a) and c) with the constant k value until the error on the validation data set is minimized.

When the previous processes are done, the $k - NN$ reference model could be upload to the ‘Server’ for further online prediction process.

Stochastic process

As mentioned in Chapter 4, the stochastic process is implemented in a non-monotonic case. Since the pre-processed data still have the feature of fluctuation, we consider the stochastic process to describe the characteristics of the deterioration respected to the operational environment.

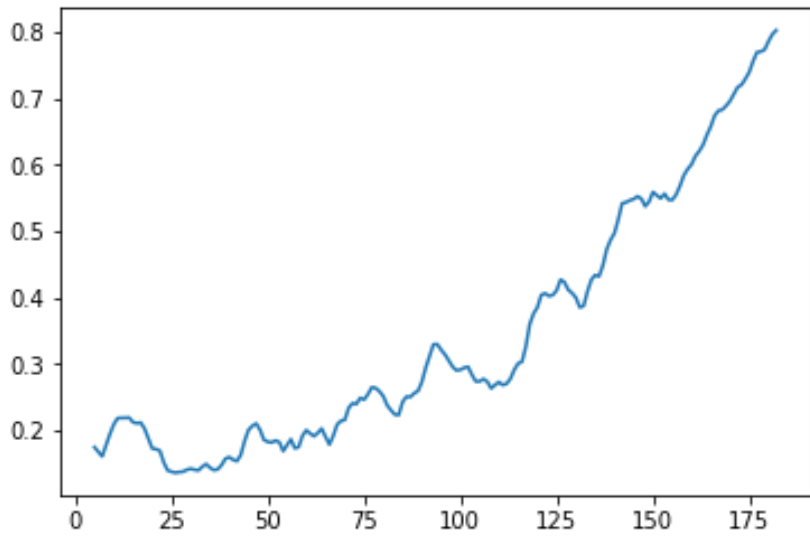


Figure 6.19 An example of the deterioration of the machine

[Figure 6.19](#) shows an example of the deterioration of the machine. The historical data has an exponential trend with some noise, which means the deterioration might have an accelerated feature with time ([Si et al., 2011a](#)), or depends on the previous degradation level ([Le Son et al., 2013](#)). Therefore, we could consider applying Geometric Brownian Motion (*GBM*) to this degradation process. Since HI_s shows an exponential trend, we fit an exponential model (Formula 6.1) proposed in Le Son et al. to have a general understanding of the deterioration evolving ([Le Son et al., 2013](#)).

$$y(t) = ae^{(-bt-c)} \tag{6.1}$$

Where a, b, c are regression factors, which are shown in table x; $y(t)$ is estimated

health indicator, and t is the lifetime of the machine.

Table 6.10 Factors of the regression fitting for all machines

| | a | b | c |
|-----|----------|-----------|----------|
| 1 | 0.277957 | -0.016010 | 1.834878 |
| 2 | 0.217378 | -0.012961 | 2.106209 |
| 3 | 0.208583 | -0.021156 | 2.384633 |
| ... | ... | ... | ... |
| 98 | 0.340329 | -0.021403 | 2.183700 |
| 99 | 0.251561 | -0.020031 | 2.334489 |
| 100 | 0.292956 | -0.016423 | 2.189216 |

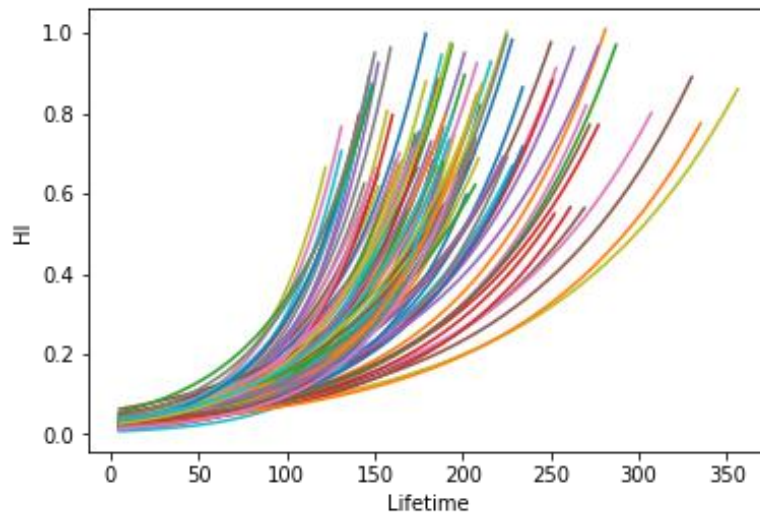


Figure 6. 20 Illustration of regression curves

Since we have acquired the exponential regression model, let us take the natural logarithm to find the increment of on fitted curve by formula 6.2.

$$\Delta y = \ln y(t + \Delta t) - \ln y(t) \tag{6.2}$$

Where: Δy is the increment in time Δt , $y(t)$ is the value of estimated HI at time t ; $y(t + \Delta t)$ is the value of HI at time $t + \Delta t$.

For each different curve, the Δy is the $-b$ factor in the regression, which represents an estimated increment. Therefore, we assume the increment fluctuates around some specific value in the historical data.

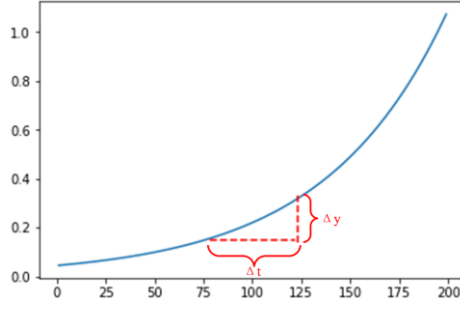


Figure 6.21 The illustration of the estimated increment

In Chapter 4, the SDE of GBM is introduced as:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t)$$

We could solve the SDE formula under Itô's interpretation ([Kobayashi et al., 2011](#)):

$$S(t) = S(0) \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W(t)\right) \quad (6.3)$$

Where: S_t are the values of health indicator at time t , S_0 is the initial value of health indicator, μ is the percentage drift, σ is the percentage volatility, $W(t)$ is Wiener Process or Standard Brownian Motion.

For estimating μ and σ , some of the authors by using Most Likelihood Estimation to get the parameters ([Kaise, 2012](#); [Park and Padgett, 2005](#); [Zhang et al., 2018](#)). Since the method is rather complicated and time-consuming, we use historical data to simplify this process([Chiang et al., 2015](#)).

In order to estimate the drift and volatility by historical data, we transfer formula 6.4 by taking the natural logarithm and get 6.4, 6.5 (Assume the time step is 1):

$$\ln S(t) = \ln S(0) + \left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W(t) \quad (6.4)$$

$$\ln S(t-1) = \ln S(0) + \left(\mu - \frac{\sigma^2}{2}\right)(t-1) + \sigma W(t-1) \quad (6.5)$$

We could obtain ΔL by subtracting 6.4 by 6.5:

$$\Delta L = \ln S(t) - \ln S(t-1) = \left(\mu - \frac{\sigma^2}{2}\right) + \sigma(W(t) - W(t-1)) \quad (6.6)$$

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

According to the previous assumption- ΔL is increment fluctuates around a mean value, and equation 6.4, ΔL follows a normal distribution:

$$\Delta L \sim N\left(\mu - \frac{\sigma^2}{2}, 2\sigma^2\right)$$

[Table 6.11](#) shows the historical dataset M_s with 100 machines.

Table 6.11 Health indicators for 100 machines

| Label | Health indicator $S(t)$ | | | | | | | |
|------------|-------------------------|--------|--------|--------|-----|--------|--------|--------|
| | 0 | 1 | 2 | 3 | ... | 349 | 350 | 351 |
| 1 | 0.0415 | 0.0100 | 0.0000 | 0.0210 | ... | 0.9130 | 0.9130 | 0.9130 |
| 2 | 0.0550 | 0.0635 | 0.0485 | 0.0400 | ... | 1.0525 | 1.0525 | 1.0525 |
| 3 | 0.0000 | 0.0040 | 0.0105 | 0.0175 | ... | 0.8215 | 0.8215 | 0.8215 |
| 4 | 0.0205 | 0.0215 | 0.0160 | 0.0155 | ... | 0.7390 | 0.7390 | 0.7390 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 97 | 0.0965 | 0.1020 | 0.0925 | 0.0945 | ... | 0.7195 | 0.7195 | 0.7195 |
| 98 | 0.0000 | 0.0045 | 0.0290 | 0.0555 | ... | 0.9470 | 0.9470 | 0.9470 |
| 99 | 0.0520 | 0.0460 | 0.0435 | 0.0305 | ... | 0.9345 | 0.9345 | 0.9345 |
| 100 | 0.0680 | 0.0750 | 0.0820 | 0.0860 | ... | 0.8035 | 0.8035 | 0.8035 |

Since we are going to take nature logarithm for all of the $S(t)$, we remove all the ‘0’ values and substitute them with the next closest values in the M_s . [Table 6.12](#) shows the value after the substitution and transforming into ΔL .

Table 6.12 The number of ΔL values

| Label | ΔL | | | | | | | |
|-------|------------|----------|----------|----------|-----|----------|----------|----------|
| | 0 | 1 | 2 | 3 | ... | 348 | 349 | 350 |
| 1 | -1.42311 | 0.741937 | 0 | 0.579818 | ... | 0.031508 | 0.029988 | 0.01768 |
| 2 | 0.143707 | -0.26948 | -0.19268 | -0.07796 | ... | 0.017813 | 0.019908 | 0.011948 |
| 3 | 0 | 0.965081 | 0.510826 | -0.72213 | ... | 0.023099 | 0.021263 | 0.037424 |
| 4 | 0.047628 | -0.29546 | -0.03175 | 0.478036 | ... | 0.04705 | 0.034247 | 0.026857 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 97 | 0.05543 | -0.09776 | 0.021391 | 0.031253 | ... | 0.037598 | 0.039788 | 0.020359 |
| 98 | 0 | 1.863218 | 0.649087 | 0.267204 | ... | 0.014558 | 0.020358 | 0.031102 |
| 99 | -0.1226 | -0.05588 | -0.35503 | -0.3973 | ... | 0.031476 | 0.029969 | 0.021634 |

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

| | | | | | | | | |
|-----|---------|----------|----------|----------|-----|----------|----------|----------|
| 100 | 0.09798 | 0.089231 | 0.047628 | 0.017291 | ... | -0.01302 | 0.013021 | 0.038698 |
|-----|---------|----------|----------|----------|-----|----------|----------|----------|

However, some of the original values in the dataset M_s are rather small, which could cause a large value by taking the natural logarithm. For example, in the dataset M_s , one of the values is $2.77556E-16$, where the ΔL is almost 33. It is not practical in this case. Therefore, in the ΔL dataset, we keep the values within the range $(-5,5)$ through investigating the dataset. Then, we could get the μ' and σ'^2 from ΔL dataset for each machine.

In the prognostics, the following formula is prepared for estimation:

$$S(t) = S(t - 1) + S(t - 1)\mu\Delta t + S(t - 1)\sigma W(0,1) \tag{6.7}$$

Thus, we need μ and σ . By applying

$$\sigma = \frac{\sigma'^2}{2} \tag{6.8}$$

and

$$\mu = \mu' + \frac{\sigma'^2}{4} \tag{6.9}$$

[Table 6.13](#) partially shows the μ and σ for each machine and the mean values.

Table 6.13 The μ , σ , and the mean values for each machine

| | 1 | 2 | 3 | 4 | ... | 99 | 100 | Mean |
|----------------------------|----------|----------|----------|----------|-----|----------|----------|----------|
| μ | 0.02347 | 0.026679 | 0.041537 | 0.020817 | ... | 0.070827 | 0.02685 | 0.032436 |
| σ | 0.111367 | 0.178142 | 0.131663 | 0.236773 | ... | 0.328432 | 0.164762 | 0.159177 |

[Figure 6.22](#) shows the GBM with $\mu = 0.02347$ and $\sigma = 0.111367$ with the initial $S(0) = 0.001$.

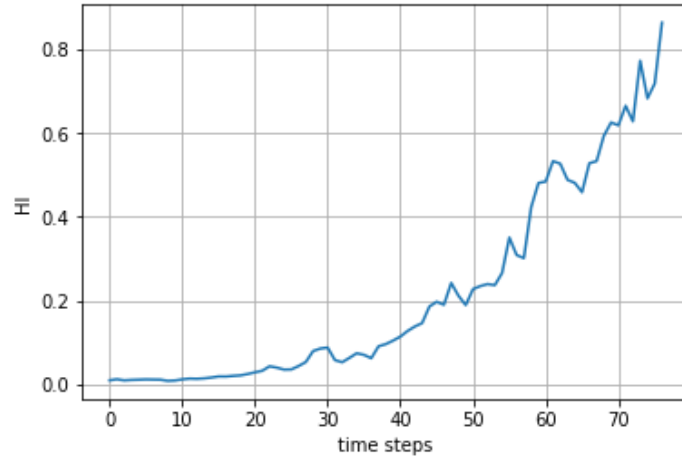


Figure 6.22 An example of GBM with a certain μ and σ

However, when it comes to prognostics, we only need one variable for each μ and σ . Meanwhile, at the beginning of the monitoring, there is no sufficient data for obtaining μ and σ . Hence, we take the mean value of μ s and the mean value of the σ s as the initial variable. When there are enough data, we could consider updating these two parameters by new monitoring data.

The stochastic process model is unlike the neural network model and $k - NN$ model. In the formula, the process could evolve to infinite if a μ and σ are set up. Therefore, when it comes to prognostics, we need to set a threshold to indicate that the machine is considered failed if the health indicator value passes this threshold. [Figure 6.23](#) is the density distribution of the health indicator when 100 machines reach the end of the lifetime.

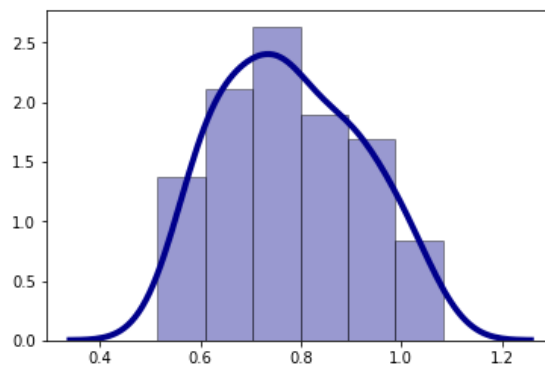


Figure 6.23 Density distribution of the health indicator for 100 machines

We set the threshold by the mean value of the health indicators, i.e., $S(t) = 0.8$ with an uncertainty variable, i.e., $\sigma = 0.137$. [Figure 6.24](#) shows ten paths when the simulation

stops after $S(t) > 0.8$ with $S(0) = 0.001$, $\mu = 0.032436$ and $\sigma = 0.111367$.

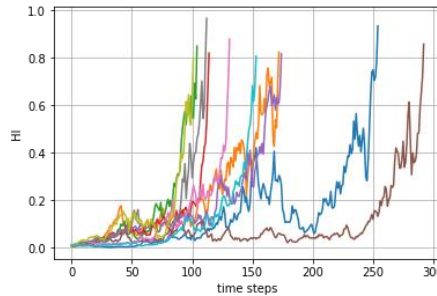


Figure 6.24 GBM paths with 0.8 as the threshold

Deep learning neural network

From the mechanism in Chapter 4, the trend model prognostics are one-dimensional process, while deep learning neural network supports multi-dimensional input (Hu et al., 2019; Raman and Hassanaly, 2019). Thus, in this section, we change the previous trend extraction process into a neural network for the reference model directly. The main framework of the neural network is under the TensorFlow platform. The necessary information of the neural network is introduced in Chapter 4. The steps of building a neural network are the following:

a) Data filtering:

In this part, the data filtering is aimed to remove the unchanged data and noise input, i.e., the operation condition 1-3 and sensor 1,5,6,10,16,18,19 are discarded from the data frame.

Table 6.14 Dataframe for deep learning neural network

| | time/cycle | Sensor 2 | Sensor 3 | Sensor 4 | ... | Sensor 17 | Sensor 20 |
|------------|------------|----------|----------|----------|-----|-----------|-----------|
| 1 | 1 | 641.82 | 1589.70 | ... | 392 | 39.06 | 23.4190 |
| 1 | 2 | 642.15 | 1591.82 | ... | 392 | 39.00 | 23.4236 |
| 1 | 3 | 642.35 | 1587.99 | ... | 390 | 38.95 | 23.3442 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 100 | 198 | 643.42 | 1602.46 | ... | 398 | 38.44 | 22.9333 |
| 100 | 199 | 643.23 | 1605.26 | ... | 395 | 38.29 | 23.0640 |
| 100 | 200 | 643.85 | 1600.38 | ... | 396 | 38.37 | 23.0522 |

b) Normalization

In the remaining data frame, the scaling of signals is different. Hence, we need to formulate all the signals in range (0,1). The normalization is implemented in the scaling, which follows equation 6.4.

$$X_d = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{6.10}$$

Table 6.15 Dataframe after normalization

| | Sensor 2 | Sensor 3 | Sensor 4 | ... | Sensor 17 | Sensor 20 | Sensor 21 |
|--------------|----------|----------|----------|-----|-----------|-----------|-----------|
| 0 | 0.183735 | 0.406802 | 0.309757 | ... | 0.333333 | 0.713178 | 0.724662 |
| 1 | 0.283133 | 0.453019 | 0.352633 | ... | 0.333333 | 0.666667 | 0.731014 |
| 2 | 0.343373 | 0.369523 | 0.370527 | ... | 0.166667 | 0.627907 | 0.621375 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 20628 | 0.665663 | 0.684979 | 0.775321 | ... | 0.833333 | 0.232558 | 0.053991 |
| 20629 | 0.608434 | 0.746021 | 0.747468 | ... | 0.583333 | 0.116279 | 0.234466 |
| 20630 | 0.795181 | 0.639634 | 0.842167 | ... | 0.666667 | 0.178295 | 0.218172 |

c) Define the input variables and output variables:

In the neural network, the input layer contains input dimensions. In this case, the input variables are the remaining 12-dimensional training data, and the output variable is the RUL values.

Table 6.16 Training data sample for neural network

| | Input data (X_d) | | | | | | Lifetime (Y_d) |
|----------|----------------------|----------|----------|-----|-----------|-----------|--------------------|
| | Sensor 2 | Sensor 3 | Sensor 4 | ... | Sensor 17 | Sensor 20 | |
| 0 | 0.183735 | 0.406802 | 0.309757 | ... | 0.333333 | 0.713178 | 191 |

| | | | | | | | |
|-------|----------|----------|----------|-----|----------|----------|-----|
| 1 | 0.283133 | 0.453019 | 0.352633 | ... | 0.333333 | 0.666667 | 190 |
| 2 | 0.343373 | 0.369523 | 0.370527 | ... | 0.166667 | 0.627907 | 189 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 20628 | 0.665663 | 0.684979 | 0.775321 | ... | 0.833333 | 0.232558 | 2 |
| 20629 | 0.608434 | 0.746021 | 0.747468 | ... | 0.583333 | 0.116279 | 1 |
| 20630 | 0.795181 | 0.639634 | 0.842167 | ... | 0.666667 | 0.178295 | 0 |

d) Define layers and nodes:

In this case, we consider establishing a four-layer neural network with two hidden layers, one input layer, and one output layer. The architecture of the neural network is shown in [Figure 6.25](#).

```

Model: "sequential_8"
-----
Layer (type)                Output Shape              Param #
-----
dense_29 (Dense)            (None, 14)                182
dense_30 (Dense)            (None, 1024)              15360
dense_31 (Dense)            (None, 128)               131200
dense_32 (Dense)            (None, 1)                 129
-----
Total params: 146,871
Trainable params: 146,871
Non-trainable params: 0
-----
    
```

Figure 6.25 The neural network framework

e) Activation and optimizer function:

In this case, we choose sigmoid function as activation function, which could indicate the properties of each machine. Moreover, for the optimizer function, we choose Adam with a learning rate of 0.0015.

f) Cross-validation:

To prevent the overfitting problem and evaluate the model, we split the total data frame

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

(M_d) into a training dataset M_d^T and a testing data set M_d^V . The training dataset takes 70% of the whole data frame. The training data and validation data are:

$$M_d = \{M_d^T, M_d^V\}$$

each subset with the corresponding input and output are:

$$M_d^T = \{X_d^T, Y_d^T\}, M_d^V = \{X_d^V, Y_d^V\}$$

The rule of splitting the training data set M_d^T and a testing data set M_d^V are realized by a Python function ‘*validation_split*’, which is a random quantity in each epoch.

[Table 6.17](#) is an example of a training and validation dataset with the corresponding input variables (X_d) and the output variable (Y_d).

Table 6.17 An example of training and validation dataset

| | | Input data (X_d) | | | | | | Lifetime (Y_d) |
|--|-------|----------------------|----------|----------|-----|-----------|-----------|--------------------|
| | | Sensor 2 | Sensor 3 | Sensor 4 | ... | Sensor 17 | Sensor 20 | |
| Training data set $\{X_d^T, Y_d^T\}$ | 3725 | 0.530120 | 0.454545 | 0.388758 | ... | 0.379845 | 0.623170 | 50.0 |
| | 4130 | 0.481928 | 0.509483 | 0.485652 | ... | 0.418605 | 0.321320 | 37.0 |
| | 14458 | 0.343373 | 0.324395 | 0.517218 | ... | 0.573643 | 0.661143 | 92.0 |
| | ... | ... | ... | ... | ... | ... | ... | ... |
| | 4227 | 0.325301 | 0.397646 | 0.325118 | ... | 0.620155 | 0.547501 | 135.0 |
| | 5572 | 0.331325 | 0.446697 | 0.440749 | ... | 0.565891 | 0.591964 | 57.0 |
| | 18814 | 0.463855 | 0.632003 | 0.649223 | ... | 0.558140 | 0.302403 | 41.0 |
| Validation data set $\{X_d^V, Y_d^V\}$ | 7810 | 0.825301 | 0.627207 | 0.799291 | ... | 0.302326 | 0.427644 | 15.0 |
| | 13428 | 0.590361 | 0.778068 | 0.916779 | ... | 0.294574 | 0.232809 | 3.0 |
| | 18683 | 0.319277 | 0.274907 | 0.469784 | ... | 0.496124 | 0.613781 | 172.0 |
| | ... | ... | ... | ... | ... | ... | ... | ... |

CHAPTER 6: DIGITAL TWIN OFFLINE MODEL

| | | | | | | | | |
|--|-------|----------|----------|----------|-----|----------|----------|------|
| | 1025 | 0.436747 | 0.316765 | 0.382512 | ... | 0.620155 | 0.657553 | 90.0 |
| | 16576 | 0.611446 | 0.608023 | 0.720628 | ... | 0.519380 | 0.256421 | 15.0 |
| | 16113 | 0.771084 | 0.608895 | 0.744092 | ... | 0.418605 | 0.191798 | 24.0 |

The mean square error is calculated in each epoch and used to evaluate the model. [Figure 6. 26](#) illustrates the model loss of training dataset and testing dataset. From the model loss, we could assume this model could be used for the estimation after 100 epochs.

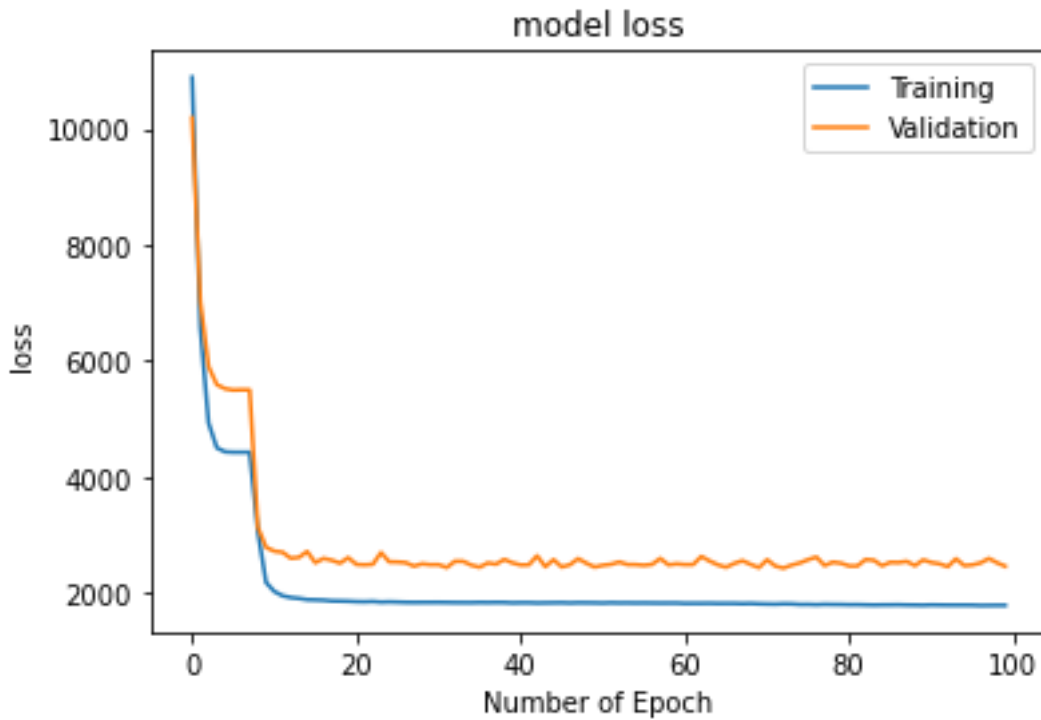


Figure 6. 26 Model loss illustration during the training process

When the monitoring data from a new machine is collected, it will be passed to this model and get the lifetime prediction. The illustration of this neural network is shown in [Figure 6.27](#).

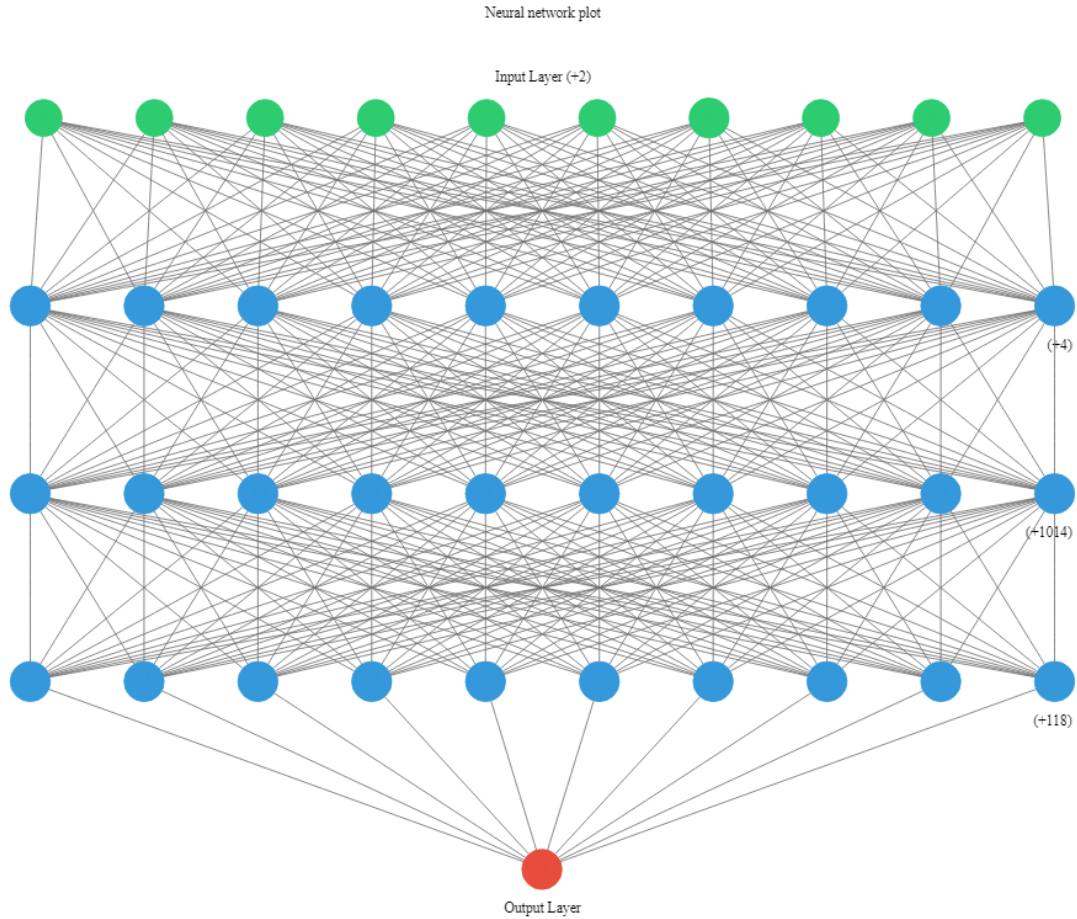


Figure 6.27 Illustration of this neural network

6.2.5 Upload the offline reference model

When the reference model is established, the whole frameworks could be uploaded to the ‘Server’ and store all of the information. The ‘Server’ could be a local mirror with the LAN network or on the ‘Cloud’.

For the model uploading, in the communication protocol, we need to insert the *IP* address to build the connection between ‘Server’ and ‘Client’. In this case, our *LAN IP* address is ‘192.168.137.1’.

[Figure 6. 28](#) shows the interface of request to upload the reference model to the ‘Server’, and the uploading process.

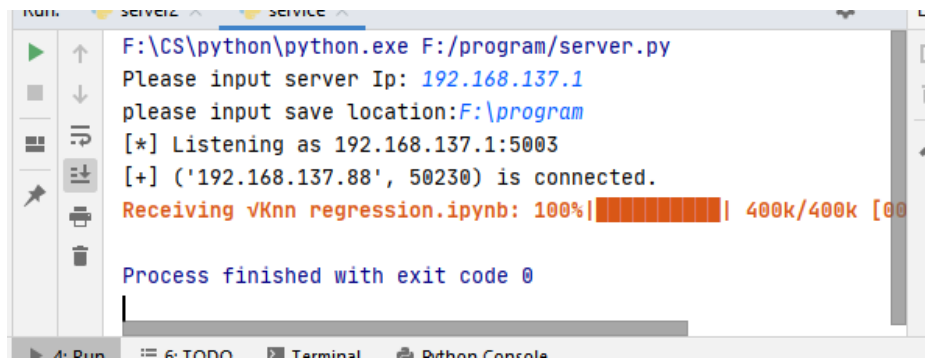
CHAPTER 6: DIGITAL TWIN OFFLINE MODEL



```
Run: server2 x service x
F:\CS\python\python.exe F:/program/server2.py
waiting the call
<socket.socket fd=520, family=AddressFamily.AF_INET, type=Socket
the call has comming
data: Data uploading request

Process finished with exit code -1
```

a. Uploading request from the 'Client'.



```
Run: server2 x service x
F:\CS\python\python.exe F:/program/server.py
Please input server Ip: 192.168.137.1
please input save location:F:\program
[*] Listening as 192.168.137.1:5003
[+] ('192.168.137.88', 50230) is connected.
Receiving vknn_regression.ipynb: 100%|██████████| 400k/400k [00
Process finished with exit code 0
```

b. Receiving offline reference model.

Figure 6. 28 Uploading offline reference model to the 'Server'.

Since we upload all of the offline reference models, we could pursue the following online prognostics section.

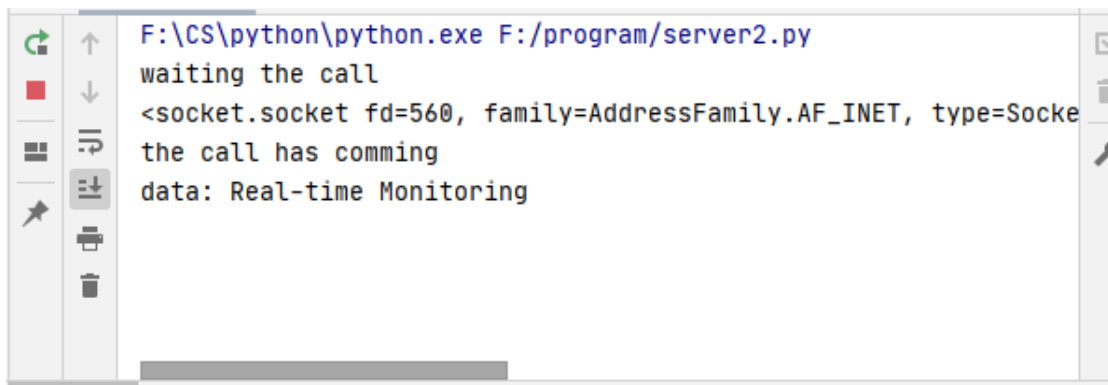
Chapter 7

Online prognostics and decision making

In this chapter, we are going to apply the reference model on the new dataset collected from a new machine and make decisions in real-time. The prognostics are based on three reference models established in Chapter 5. The real-time decision-making model is based on the formula in Chapter 4.

7.1 Online prognostics

In this section, the new sensor data are collected through the communication protocol. [Figure 7. 1](#) shows the process of transferring data from the ‘Client’ machine to ‘Server’. Every cycle, the monitoring data will be streamed from the machine (‘Client’) to the reference model (‘Server’). The algorithm operates and estimates the lifetime of the machine. Each cycle, the ‘Server’ could receive the monitoring data from the ‘Client’ machine.



```
F:\CS\python\python.exe F:/program/server2.py
waiting the call
<socket.socket fd=560, family=AddressFamily.AF_INET, type=Socket
the call has coming
data: Real-time Monitoring
```

Figure 7. 1 Illustration of receiving monitoring data

For the $k - NN$ and stochastic process model, we apply the same pre-processing procedure to the newly collected data, as shown in Chapter 6. In contrast, for the ANN model, we use the same input signals as in the reference model. In the following

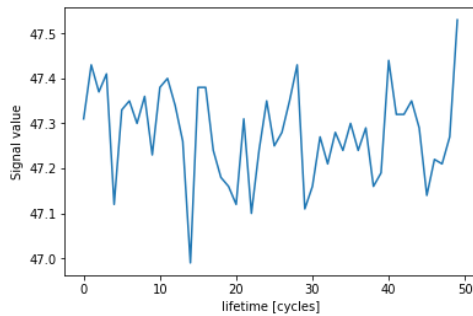
section, the prognostics are presented through three reference models. [Table 7.1](#) shows the monitoring data of a new machine, which is used to present as input parameters in the following sections.

Table 7.1 The monitoring data of a new machine

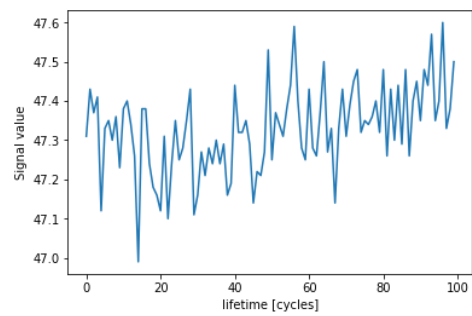
| | Sensor 2 | Sensor 3 | Sensor 4 | ... | Sensor 17 | Sensor 20 | Sensor 21 |
|------------|----------|----------|----------|-----|-----------|-----------|-----------|
| 1 | 642.75 | 1582.03 | 1392.43 | ... | 394 | 38.83 | 23.4048 |
| 2 | 642.02 | 1586.39 | 1398.94 | ... | 391 | 39.03 | 23.2768 |
| 3 | 642.36 | 1590.20 | 1405.66 | ... | 391 | 38.99 | 23.3628 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 194 | 643.30 | 1593.45 | 1426.21 | ... | 395 | 38.52 | 23.0864 |
| 195 | 643.57 | 1603.82 | 1426.44 | ... | 396 | 38.49 | 23.1562 |
| 196 | 643.31 | 1598.19 | 1420.66 | ... | 395 | 38.53 | 23.1105 |

7.1.1 Similarity-based model ($k - NN$ regression model):

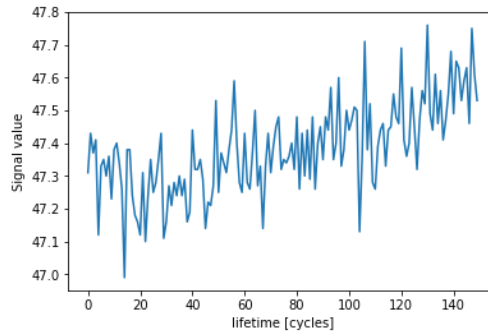
Since the $k - NN$ regression model is established according to sensor 11, we drop other parameters and only collect sensor 11 as the input signal in the prognostics. The data streams into the reference model in each cycle. [Figure 7.2\(a,b,c\)](#) shows the signal evolving in cycle 50,100 and 150 of one new machine.



a. Signal information within Cycle 50



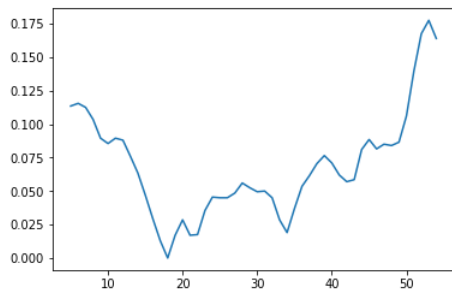
b. Signal information within Cycle 100



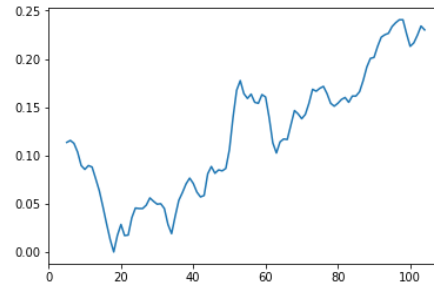
c. Signal information within Cycle 150

Figure 7.2 The illustration of signal evolving

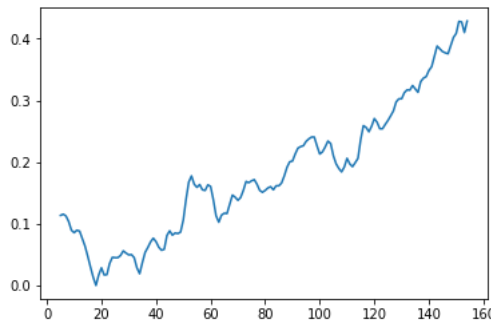
Figure 7. 3(a,b,c) shows that after we eliminate partial noise and extract the primary trend by *TSD*. Meanwhile, we scale the data from 0 by subtracting the minimum value of dataset as the new monitoring data N_k .



a. Signal information within Cycle 50



b. Signal information within Cycle 100



c. Signal information within Cycle 150

Figure 7. 3 the primary trend by applying TSD

After the pre-process of the new dataset N_k , we search from the $k - NN$ reference model to find similar growths with the new dataset N_k and get the estimated **lifetime** \hat{T} . This process achieved by Python $k - NN$ library, '*neigh.predict*'. As time goes by, we could obtain several estimated \hat{T}_i . However, the $k - NN$ could only get a specific value without uncertainty. To obtain uncertainty of the prediction, we propose to collect all of the estimations \hat{T}_i , and get their distributions. Figure 7. 4 shows the distributions

CHAPTER 7: ONLINE PROGNOSTICS AND DECISION MAKING

and estimation processes of \hat{T}_{50} , \hat{T}_{100} and \hat{T}_{150} . The dark blue points are estimated lifetime, the red curve is the real-time monitoring data, and the light blue points are the historical data. [Table 7.2](#) shows the mean values and standard deviations of \hat{T}_{50} , \hat{T}_{100} and \hat{T}_{150} .

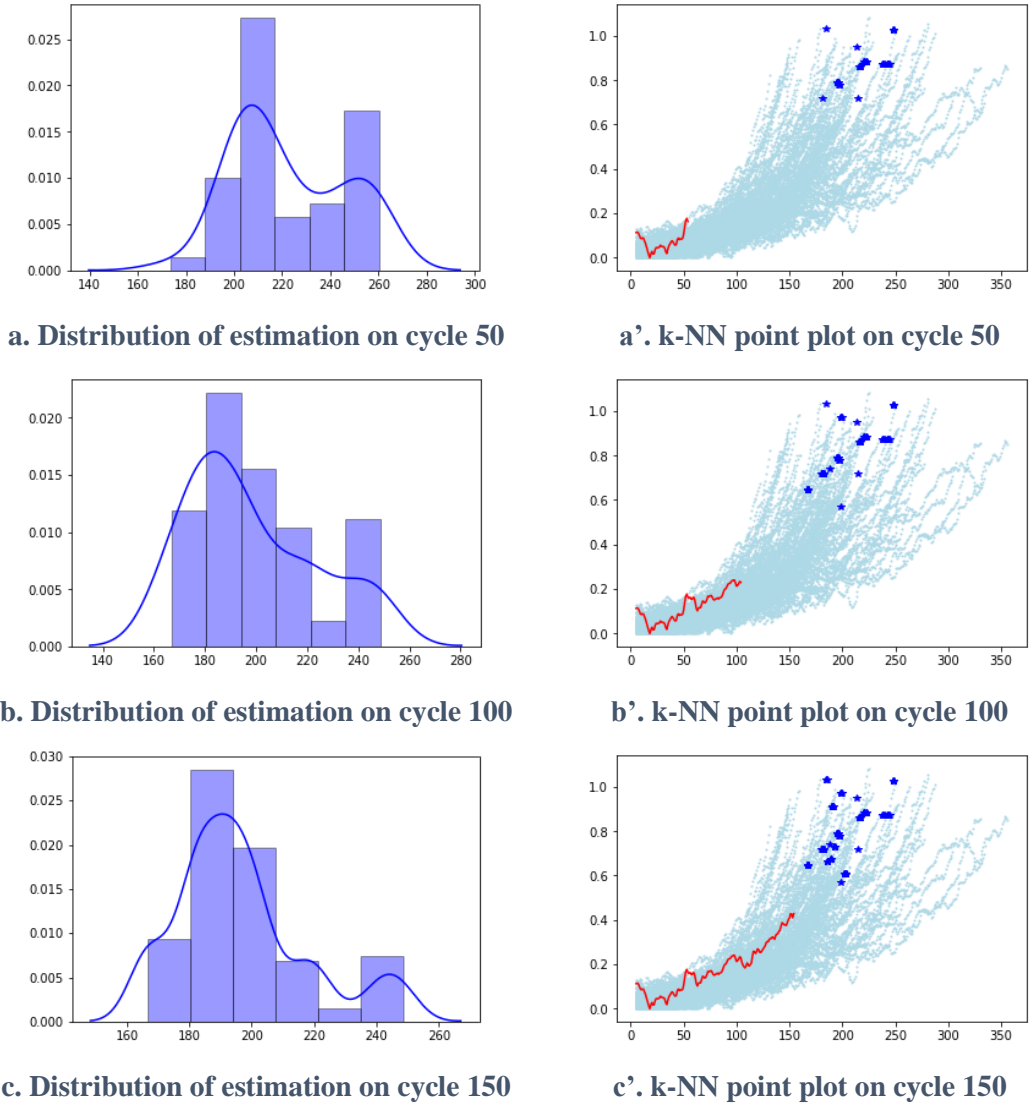


Figure 7.4 Distribution and estimation process

Table 7.2 The mean values and standard deviations of lifetime

| | \hat{T}_{50} | \hat{T}_{100} | \hat{T}_{150} |
|-------------|----------------|-----------------|-----------------|
| Mean | 217.40 | 198.32 | 196.61 |
| SD | 20.71 | 25.04 | 21.21 |

After the estimation time by time, we could obtain a series lifetime for the different machine through this process, and calculate the RUL- \hat{R}_t with mean value and std. Further, the estimated lifetime will be transferred into the decision model.

Table 7.3 The mean values and standard deviations of RUL

| | \hat{R}_{50} | \hat{R}_{100} | \hat{R}_{150} |
|------|----------------|-----------------|-----------------|
| Mean | 167.40 | 98.32 | 46.61 |
| SD | 20.71 | 25.04 | 21.21 |

7.1.2 Stochastic process model

The Stochastic process model is established by the primary trend of Sensor 11. Therefore, when the new data streams into the model, we only keep Sensor 11 and make the prognostics.

The raw data processing is the same as $k - NN$ regression model (here no longer repeat), and we denote the new dataset as N_s . The principal mechanism is following equation 6.7, where we consider the parameter $dt = 1$, $\mu = 0.032436$, and $\sigma = 0.159177$. The essential algorithm is shown in the following:

Algorithm 1 Geometric Brownian Motion

```

Input:  $S[0], I$ 
Output:  $S[t], T[t]$ 
for all  $t$  in range  $I$ :
     $rand = random\_normal(0,1)*0.159177$ 
     $S[t] = S[t-1]+0.032436*S[t-1]+rand*S[t-1]$ 
     $T[t] = t$ 
    If  $S[t]>0.8$ 
        break
end

```

Algorithm 1 only can provide one path of health indicator. Therefore, The Monte Carlo Simulation is implemented to generate several paths and get the uncertainty of estimation. Algorithm 2 shows the Monte Carlo process of collecting the lifetime variables:

Algorithm 2 Monte Carlo Simulation

```

Input:  $S[0], I, N$ 
Output:  $lifetime[]$ 
 $m = 0$ 
While  $m < N$ :
     $result = GBM(S[0], I)$ 
     $x = result[1]$ 
     $y = result[0]$ 
     $lifetime.append(x[-1])$ 
     $m += 1$ 
end

```

As proposed in $k - NN$ model, we estimate the **RUL**- \hat{R}_{50} , \hat{R}_{100} and \hat{R}_{150} in time/cycle 50, 100 and 150, with corresponding uncertainty parameters – standard deviation. Meanwhile, we update the μ and σ by the new data with the percentage of current health indicator value:

$$\mu' = \left(\frac{HI_t}{0.8}\right) * \mu + \left(1 - \frac{HI_t}{0.8}\right) * \mu_n \quad (7.1)$$

$$\sigma' = \left(\frac{HI_t}{0.8}\right) * \sigma + \left(1 - \frac{HI_t}{0.8}\right) * \sigma_n \quad (7.2)$$

Where μ' and σ' is the current value in the GBM, μ and σ are historical values, μ_n and σ_n is the new value obtained from the new dataset, HI_t is the current health indicator.

Here, the alternative method for updating μ' and σ' is applying Bayesian approaches ([Mosallam et al., 2015](#)). The Bayesian approach, in this case, is rather complicated. We simplify this process by taking the weight between the new parameter and the historical parameter as formula 7.1 and 7.2.

[Figure 7.5](#) shows the Monte Carlo simulation processes of \hat{R}_{50} , \hat{R}_{100} and \hat{R}_{150} .

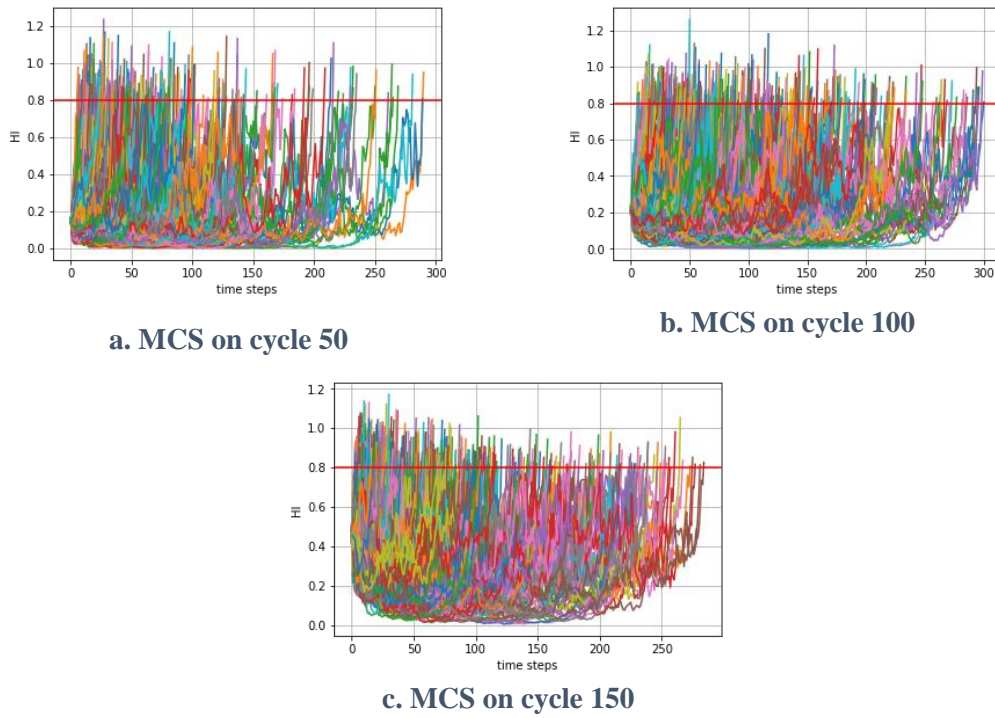


Figure 7. 5 MCS process

Table 7. 4 The mean values and standard deviations of RUL

| | \hat{R}_{50} | \hat{R}_{100} | \hat{R}_{150} |
|------|----------------|-----------------|-----------------|
| Mean | 123.762 | 99.77 | 44.91 |
| SD | 78.162 | 77.51 | 54.49 |

After, we could steam the estimation data and uncertainty in the decision model.

7.1.3 Deep learning neural network model

When we obtain the new data from the machine, we need to follow the same normalization procedure in the offline phase in Chapter 6 formula 6.10, since the data does not cover all the value. Therefore, we could obtain the following dataset N_d after normalization in [Table 7. 5](#).

Table 7. 5 Dataset after normalization

| | Sensor 2 | Sensor 3 | Sensor 4 | ... | Sensor 17 | Sensor 20 | Sensor 21 |
|-----|----------|----------|----------|-----|-----------|-----------|-----------|
| 1 | 0.511041 | 0.337315 | 0.164182 | ... | 0.625 | 0.472727 | 0.664402 |
| 2 | 0.280757 | 0.450532 | 0.297121 | ... | 0.250 | 0.654545 | 0.483227 |
| 3 | 0.388013 | 0.549468 | 0.434348 | ... | 0.250 | 0.618182 | 0.604954 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 193 | 0.684543 | 0.633861 | 0.853992 | ... | 0.750 | 0.190909 | 0.213730 |
| 194 | 0.769716 | 0.903142 | 0.858689 | ... | 0.875 | 0.163636 | 0.312527 |
| 195 | 0.687697 | 0.756946 | 0.740658 | ... | 0.750 | 0.200000 | 0.247841 |

At each cycle/time, we feed the data to the *NN* reference model, and then, there is an estimated RUL value- \hat{Y}_i by executing '*Sequential().predict()*'. [Table 7. 6](#) shows the partial estimated RUL values in each cycle. [Figure 7. 6](#) illustrates the estimation throughout time steps.

Table 7. 6 The estimated RUL

| | 1 | 2 | 3 | 4 | ... | 195 |
|-------------|--------|--------|--------|--------|-----|------|
| \hat{Y}_i | 135.07 | 139.41 | 134.99 | 137.80 | ... | 7.27 |

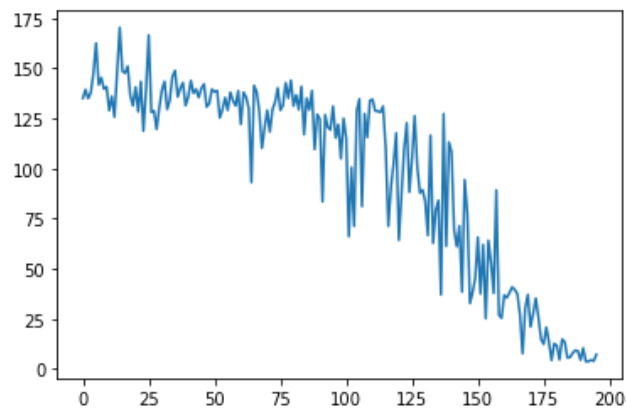


Figure 7. 6 The estimation throughout time steps

7.2 Prognostics information of six machines

The previous section presents the necessary processes for prognostics of one machine. Since we have six identical independent machines, we execute the same process for these six machines and get the RUL in time $t = 50, 100$ and 150 .

Table 7. 7 Online prognostics for six machines

| | | <i>k</i> – NN (RUL) | | <i>GBM</i> (RUL) | | Neural network (RUL) |
|-----------|---------|------------------------|-------|---------------------|--------|-------------------------|
| | | Mean | SD | Mean | SD | |
| Machine 1 | t = 50 | 167.4 | 20.71 | 123.762 | 78.162 | 148.67 |
| | t = 100 | 98.32 | 25.04 | 99.77 | 77.51 | 118.71 |
| | t = 150 | 46.61 | 21.21 | 44.91 | 54.49 | 63.37 |
| Machine 2 | t = 50 | 172.39 | 22.82 | 148.50 | 75.79 | 190.67 |
| | t = 100 | 130.83 | 35.85 | 107.59 | 70.18 | 148.71 |
| | t = 150 | 44.43 | 4.54 | 49.79 | 52.94 | 53.29 |
| Machine 3 | t = 50 | 151.51 | 17.70 | 117.70 | 65.18 | 146.03 |
| | t = 100 | 169.91 | 30.72 | 99.07 | 63.02 | 131.39 |
| | t = 150 | 83.76 | 14.47 | 65.31 | 53.16 | 50.27 |
| Machine 4 | t = 50 | 157.63 | 32.36 | 148.87 | 78.27 | 146.03 |
| | t = 100 | 97.83 | 17.60 | 67.71 | 61.06 | 100.25 |
| | t = 150 | 36.18 | 9.08 | 38.12 | 49.98 | 43.25 |
| Machine 5 | t = 50 | 136.73 | 21.12 | 119.40 | 65.57 | 137.55 |
| | t = 100 | 112.71 | 34.09 | 77.65 | 59.27 | 117.05 |
| | t = 150 | 67.98 | 13.68 | 53.08 | 48.30 | 56.06 |
| Machine 6 | t = 50 | 168.50 | 17.36 | 153.06 | 95.31 | 143.42 |
| | t = 100 | 109.65 | 15.47 | 84.26 | 80.58 | 96.40 |
| | t = 150 | 45.699 | 11.69 | 36.412 | 22.54 | 49.79 |

7.3 Decision-making model

In this section, the decision-making model is introduced to demonstrate the importance of decision-making in the predictive maintenance digital twin. The underlying mechanism is to follow equation 5.3 in Chapter 5. The decision model is aimed to find a suitable time slot for maintenance and pre-prepare the suitable service for the

maintenance process based on the cost per unit time.

$$C(t_0) = (S + \sum_{j \in G_j}^m I_j \times C_{pm} + \sum_{i \notin G_j}^m I_i \times C_u \times t) / t_0 \quad (7.3)$$

The decision model is established in the Python environment with Simpy package. Simpy is a discrete-event simulation framework powered by Python generator functions, which is a process-based simulation, especially for continuous-time simulation.⁸

The fundamental assumptions of this model are simulating the maintenance decision for the wind farm, which has a specific maintenance window due to the weather or other environmental issues. Moreover, the cost of maintenance is rather high if we miss a suitable maintenance window ([Tavner, 2012b](#)).

In this case, we assume the maintenance could only be conducted during the maintenance window. During each maintenance action, we replace the main component for all of the machines, and we assume that after the replacement, the machines are AGAN. To make the model more intuitive, we only consider one round and apply the mean lifetime without uncertainty in this model. In order to make the model more practical, we also consider modelling the repair time with randomness, repairman, and spare parts. The decision model is divided into two parts, one for factory modelling, one for optimization information.

The main architecture for this factory is shown in Algorithm 1:

⁸ The decision mode is based on the Simpy package and examples on the website <https://pypi.org/project/simpy/>

Algorithm 1 Machine farm

Input: The initial **cost** for prepare

Repair schedule

Maintenance **engineers**

Repair time

Machine lifetime

Output: Total **cost**, **spares**

Main:

Class: Machine farm

Function: Machine operation

If lifetime < repair schedule:

Failed time = lifetime

Repairing time = scheduled time

Cost for penalty = delay days * cost / day

else: (machine failed after scheduled time)

Repairing time = scheduled time

Function: **Repair**

Request **engineers**

Request **spares** ## components

Cost for **repair** = Repair time * cost/day

end

Algorithm 1 is to simulate the machine operational information. After we establish the machinery information, we could follow Algorithm 2 to get the maintenance time slot within the maintenance window.

Algorithm 2 Optimizer

Input: **Maintenance window**

Output: Total **cost per unit time**, **spares**

Main:

For **time_slot** in **Maintenance window**:

Algorithm 1

Per_Unit_Cost = Total cost/time_slot

end

Find_optimized **time** slot

Find **spares** used

end

Algorithm 2 is to find the optimized time slot within the maintenance window by collecting the cost information.

7.3.1 Decision model implement

For this model, we need the following assumptions:

- The Set-up cost (S), which indicates the costs for preparations, such as transportation, raw materials.
- Planned maintenance cost (C_{pm}), which indicates the costs of machine repair before machine failure.
- Unplanned maintenance cost (C_u), which is the cost after the machine failure.
- Repair time (t_r), in this case, the repair time is a random variable, which indicates the repair time could depend on the situation.
- Repair schedule (t_0), the set-up schedule to conduct maintenance activity.

The formula for this model is:

$$C(t_0) = (S + \sum_{j \in G_j}^m C_{pm} \times t_{rj} + \sum_{i \notin G_j}^n C_u \times (\Delta t + t_{ri})) / t_0 \quad (7.4)$$

Where, $C(t_0)$ is the cost/ unit time, S is the set-up cost, G_j is a non-failure machine group, C_{pm} is the preventive maintenance cost, C_u is the corrective maintenance cost, t_r is the repair cost, Δt is the interval between failure time and schedule time.

After we acquire the lifetime/RUL from the prognostics model, the information could be transferred into a real-time decision model through the communication protocol.

We use the mean value of the lifetime from prognostics model in cycle 50, 100, and 150 as the example to illustrate the process. The lifetime information is showed in [Table 7.8:](#)

CHAPTER 7: ONLINE PROGNOSTICS AND DECISION MAKING

Table 7. 8 The lifetime information on cycle 50,100 and 150

| | Machine 1 | Machine 2 | Machine 3 | Machine 4 | Machine 5 | Machine 6 |
|------------|------------------|------------------|------------------|------------------|------------------|------------------|
| 50 | 217 | 222 | 202 | 208 | 187 | 219 |
| 100 | 198 | 231 | 240 | 198 | 213 | 210 |
| 150 | 197 | 194 | 234 | 187 | 212 | 195 |

The parameters settings:

Table 7. 9 Parameter information

| Maintenance window | Set-up cost | Repair time | Repair engineers | C_{pm} | C_u |
|--------------------|-------------|---------------------------------|------------------|--------------|---------------|
| 160-260 | 30k \$ | Random variable from 4-6 cycles | 2 | 2 k \$/cycle | 10 k \$/cycle |

We set the repair limitation to 2, which means only two of the machines could be repaired at one time. [Figure 7. 7](#) shows an example when we repair all of the machines after failure in cycle 256.

```

233.00 machine failed
256.00 machine to be repaired
194.00 machine failed
256.00 machine to be repaired
196.00 machine failed
256.00 machine to be repaired
217.00 machine failed
256.00 machine to be repaired
190.00 machine failed
256.00 machine to be repaired
201.00 machine failed
256.00 machine to be repaired
262.00 repair complete
262.00 repair complete
268.00 repair complete
268.00 repair complete
274.00 repair complete
274.00 repair complete

```

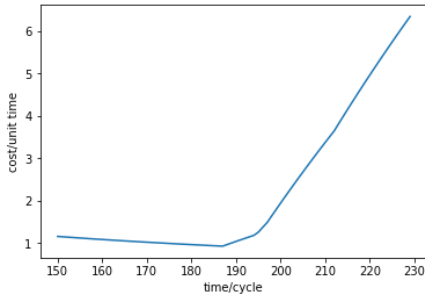
Figure 7. 7 An example of the maintenance process

After all of the parameters are set up, we could obtain a suitable time slot for doing maintenance and get the corresponding cost.

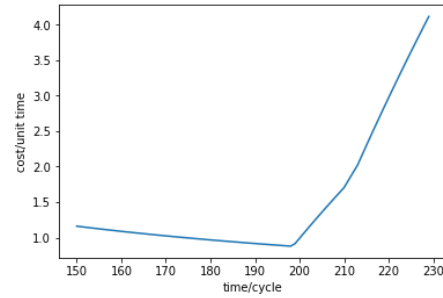
CHAPTER 7: ONLINE PROGNOSTICS AND DECISION MAKING

Table 7.10 Schedule time and cost

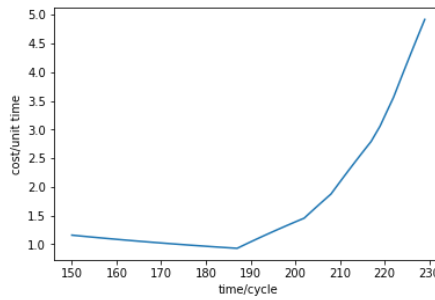
| Cycle | 50 | 100 | 150 |
|--------------------|------|------|------|
| Best schedule time | 187 | 198 | 187 |
| Cost | 0.9k | 0.8k | 0.9k |



a. Schedule time in cycle 50



b. Schedule time in cycle 100



c. Schedule time in cycle 150

Figure 7. 8 Cost per unit time with the corresponding cycle

From the simulation, we could know the suitable time is around cycle 198 during the process. After we set the schedule, the spares information in the inventory also could be illustrated. Here, we assume the spares are enough.

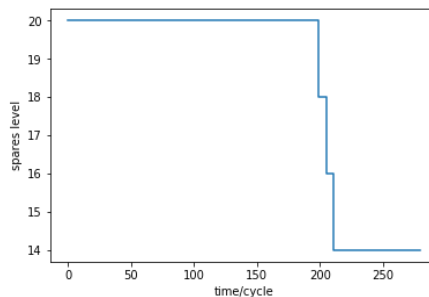


Figure 7. 9 Spares spending along with time/cycle

CHAPTER 7: ONLINE PROGNOSTICS AND DECISION MAKING

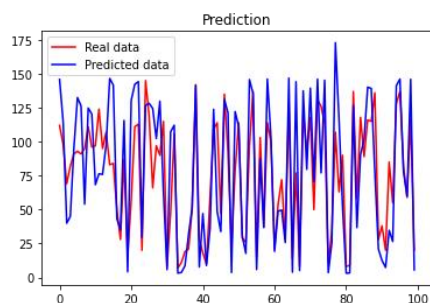
Further, we could add uncertainty and inventory information in this model to get the dynamic decision recommendations via different input schedules.

Chapter 8

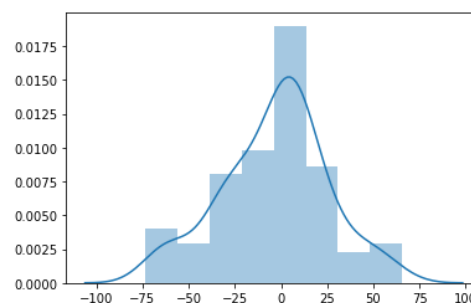
Alternative analysis

In the previous chapters, the digital twin model is presented, and the relevant theoretical background is introduced. However, this digital twin is based on a hypothetical system, and we do not have a robust standard to evaluate the result of this model.

In the previous part, we apply PHM 08 dataset and only select six engine data to perform the prognostics. In order to show the performance of these three models, i.e., $k - NN$ regression, stochastic process, and ANN , we apply the same process in Chapter 6 and 7 to all of the data in the PHM 08. Here, the figure shows the performance of the whole PHM08 test dataset in the following with corresponding error distribution. The table shows the accuracy R^2 score and Root mean square error.



a. Prediction and estimation on NN



a'. Error distribution

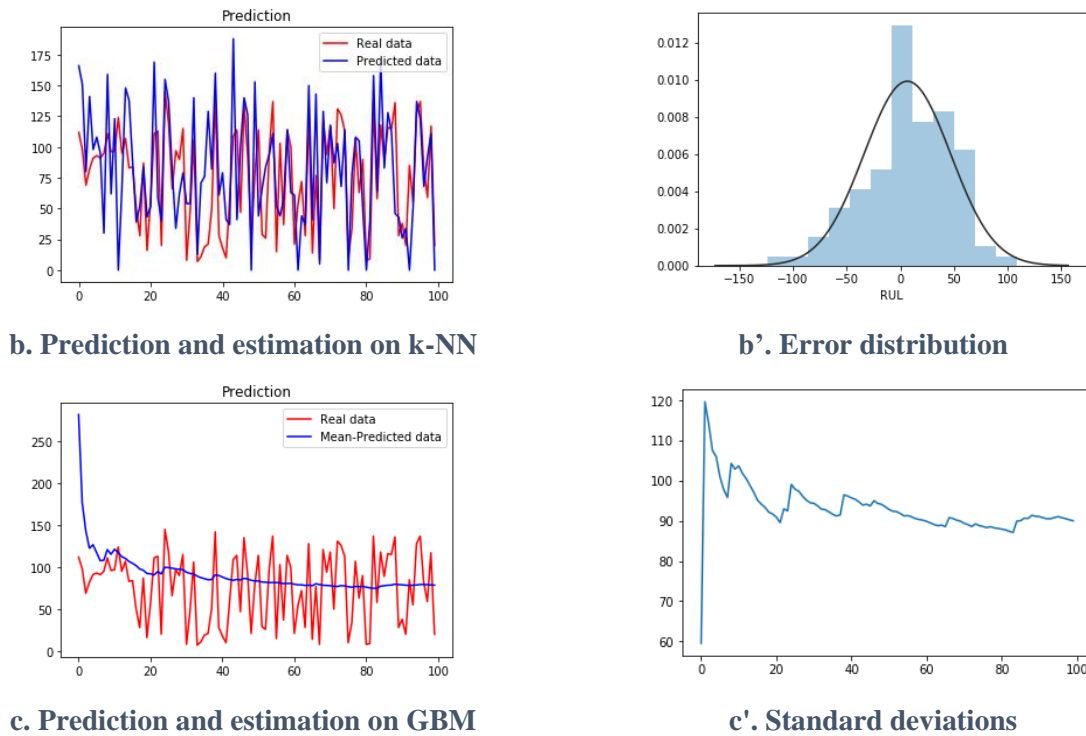


Figure 8. 1 Estimations of PHM08 dataset with three methods

Table 8. 1 R^2 and MSE for ANN and $k - NN$

| | R^2 | MSE |
|----------------------------|-------|-------|
| ANN | 0.51 | 29.10 |
| $k - NN$ | 0.26 | 45.77 |

The GBM process seems not robust⁹ than ANN and $k - NN$ in this process. The reason could be that the parameters are estimated by data directly, and the variances are enormous for each prediction. So, it is hard to get the exact number as the other two methods.

⁹ The ‘robust’ here means the performance between estimated mean values and the specific real RUL in this case is not so accurate than other two methods.

Chapter 9

Discussion, Conclusions, and Recommendations for Future Work

In this chapter, a discussion and conclusion are presented. In order to get improvement, the recommendations for further work also are listed here.

9.1 Discussion

In this thesis, the whole digital twin framework is connected by the Socket communication protocol model. This model is mainly in charge of file transferring and essential real-time communication. However, the Socket model in this thesis could not fully achieve automatic control in the case study, which passes the dataset or data file manually.

In Chapter 6, when extracting the health indicator, we use the *VHI* rather than *PHI*. As presented in Chapter 3, the *VHI* could capture the primary information. It could not combine all of the information from sensors. During the trend extracting process, it could lose some information, because we abandon some of the signal information in the offline model establishment procedure. Thus, a hybrid model could be considered to combine physical information by applying some relevant equations. For example, we could apply the thermal equation to combine some of the information. Then the data fusion could combine all of the information to approach the real connection by physical machine structure between sensors.

For the whole digital twin, we assume the machines are identical. However, after obtaining the primary trend in Chapter 4, the initial value of each machine is slightly different, which might cause by the operational information in the dataset. These

**CHAPTER 9: DISCUSSION, CONCLUSION, AND
RECOMMENDATIONS FOR FUTURE WORK**

differences could be one reason that the lifetime of the machines is varying that much at the end.

In the digital twin offline model, we propose three different models, i.e., $k - NN$ regression, stochastic process, and ANN . In the following, we are going to compare these three methods. Since the accuracy performance depends on further adjusting, the following comparison does not include accuracy performance.

Table 9. 1 Comparison of three prognostics models

| | Pros | Cons |
|----------|--|--|
| $k - NN$ | <ol style="list-style-type: none"> 1. Easy to apply; 2. Not time-consuming for training, because the prediction depends on the comparison between historical data and real-time data directly; 3. Fast processing if the data set is one-dimensional; 4. Initially, it could not obtain uncertainty. We propose to collect historical predictions to obtain uncertainty. So, it can provide uncertainty information. | <ol style="list-style-type: none"> 1. The prediction highly depends on historical data. If the historical data could cover all of the situations, the predictions could be unreliable; 2. For different predictions, k value could be a dispute, since the k selection based on existing data, not on real-time data; 3. Not suitable for a multi-dimensional data sample in this case. |
| GBM | <ol style="list-style-type: none"> 1. Suitable for one-dimensional data; 2. Possible to obtain uncertainty information and failure rate function; 3. Possible to update variables according to new data; 4. Does not need to cover all the aspect in the historical data; 5. Flexible; could use mathematic way to estimate parameters or by historical data directly; | <ol style="list-style-type: none"> 1. High expertise; Need to have a good understanding of the deterioration evolving and choose the correct model. 2. Time-consuming when applying stochastic model (GBM) 3. Hard to find proper parameters support multi-dimensional data 4. We need to define the failure threshold, which increases uncertainty. |

**CHAPTER 9: DISCUSSION, CONCLUSION, AND
RECOMMENDATIONS FOR FUTURE WORK**

| | | |
|------------|--|--|
| <i>ANN</i> | <ol style="list-style-type: none"> 1. Support multi-dimensional data; 2. Fast processing since we have the offline model; 3. Easy to apply; 4. Possible to acquire uncertainty information by Bayesian drop out; 5. Could update parameters when comes the new dataset; 6. It could be regarded as a universal proxy model if the training data covers most of the features. | <ol style="list-style-type: none"> 1. The prognostics depend on the historical data; 2. The training process is Time-consuming; 3. High professionalism; Need to have a good understanding of the mechanism of <i>ANN</i>; 4. Hard to get the best model, since there is serval different <i>ANNs</i>; 5. The prediction is only based on the latest data in this case, which increase the vulnerability; 6. Easy to hit overfitting problems. |
|------------|--|--|

In Chapter 8, we propose a decision model. To make the model directive and illustrative, we do not consider the uncertainty of the lifetime or RUL. This model is to present that what kind of information could a decision model provides since there is not much information from the literature review on how to formulate a decision model in the digital twin. This information is essential when it comes to long-term planning, such as windfarm maintenance and subsea equipment maintenance. The decision model is cost-based monitoring combined with spares, which could be adapted to net present value and inventory information. In practice, the deterioration of components is not identical. After adjusting and re-organizing, the decision-model could provide dynamic support for long-term planning.

9.2 Summary and Conclusion

The main objectives of this thesis are to propose a digital twin framework for predictive maintenance and demonstrate the framework by a case study.

From the literature review, there is not much information about a standardized methodology and process to build a digital twin for predictive maintenance. Inspired

CHAPTER 9: DISCUSSION, CONCLUSION, AND RECOMMENDATIONS FOR FUTURE WORK

by several standards, websites, and documentation, we propose a conceptual framework to illustrate the primary process for establishing a digital twin by integrating existing systems for industries in Chapter 3. The framework information includes but not limited to:

- System information collecting and system analysis
- Data availability, structure, and types
- Communication through different models
- PHM methods
- Decision making

In order to demonstrate the framework, a hypothetical system is introduced in Chapter 2. This system simulates a factory, drawn from the wind farm, which requires long-term maintenance planning and specific maintenance windows. This system could provide an insightful view of how to decide maintenance through primary factors dynamically.

Within this scope, the interior architecture of the digital twin presented in this thesis is a communication protocol, three state-of-the-art PHM methods, and a decision model. The three PHM methods are $k - NN$ similarity-based model, the Artificial Neural network model, and the Stochastic process, presented in Chapter 4. Inside the digital twin, the models are connected by a communication protocol-socket, which is aimed at establishing bridges through models and execute commands. In order to simulate the data streaming process, we divide the PHM process into two phases, offline reference, and online prognostics, respectively, to three different PHM methods. Meanwhile, the streaming data from machines are with noise and missing values. The pre-processing of the dataset is necessary before the offline model, which is presented in Chapter 4 and Chapter 6, to enhance the performance of PHM methods and get a robust prediction.

For offline reference and online prognostics, all three methods have unique characteristics both in model training and prognostics, as presented in Chapter 6 and 7. These characteristics could be an evaluation standard of the performance in the practical, such as the complexity, time spending, and accuracy. The comparison of these three methods is also discussed in Chapter 8 Discussion. The purpose of prognostics is

CHAPTER 9: DISCUSSION, CONCLUSION, AND RECOMMENDATIONS FOR FUTURE WORK

to get the RUL with uncertainty, which is the input of the decision model.

In this thesis, a simplified dynamic decision model is proposed in Chapter 7. For predictive maintenance, the decision model provides an evaluation standard of when to conduct maintenance. This standard is based on preparation, maintenance capacity, inventory, and the cost. The purpose of the decision model is not only to find a maintenance time but to integrate and allocate the resource and provide a dynamic strategy based on predicted and known events.

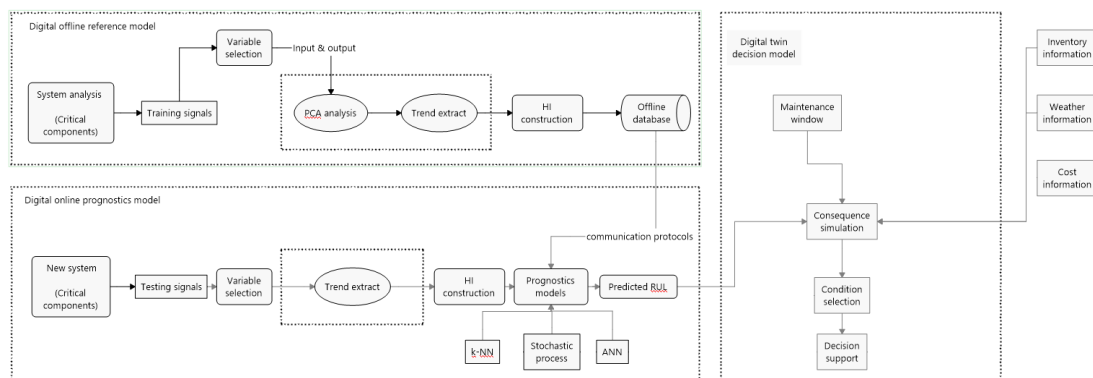


Figure 9.1 Digital twin framework presented in this thesis

As a whole, due to the limitations of heavy computational work, some of the models are not completed, only present a demo version. However, the integrated digital twin with predictive maintenance could demonstrate the framework and conceptual ideas (shown in [Figure 9.1](#)), such that it could be concluded that the main objectives for this master thesis are met.

9.3 Further Work

In this section, we are going to discuss the potential improvements and recommendations of the digital twin as guidance to implement predictive maintenance successfully in the future.

Remote control

In the digital twin proposed, the communication protocol is to establish a bridge

CHAPTER 9: DISCUSSION, CONCLUSION, AND RECOMMENDATIONS FOR FUTURE WORK

between different models and transferring data. With the prognostics model, the RUL information could be obtained. However, due to the environmental disturbance, the prognostics could not be that accurate. To avoid unnecessary loss, the communication model should send control information to the assets to adjust the working preference. Further, the model could pre-define the state of assets according to external information, such as bad weather and maintenance window time, to strive for extending the lifetime.

Prognostics and health management

In this thesis, we discussed three different methods. For these three methods, we could try to merge the Pros to get a hybrid model. In the prognostics, most of the input variable is sequential; meanwhile, In industries, some of the assets do not have much historical data. Recurrent Neural Networks (RNN) is aimed at time series analysis, which should perform better than the regular feed-forward neural network. Stochastic processes use ‘the percentage drift’ and ‘the percentage volatility’ to estimate the health indicator. Bayesian could help to update the parameters. There could be some hybrid neural network model to integrate these three features¹⁰, which could be a potential development.

Decision model

For predictive maintenance, the decision model could provide solutions to engineers and realize partial control of the assets. During the operation of the asset, one deterioration component may cause a chain effect and cause more damage to the machine. Thus, it is essential to lower or shut down the machine remotely or automatically to reduce the cost of maintenance by the decision model. Meanwhile, the primary mission of predictive maintenance is to overall plan the cost and available resources. Meanwhile, it could monitor the market and calculate the updated price of rental or purchase according to the net present value (NPV). In the future, the decision

¹⁰ The combination of these three features could be inside RNN, the activation function is related to ‘the percentage drift’ and ‘the percentage volatility’, and the parameters are updated by Bayesian model.

***CHAPTER 9: DISCUSSION, CONCLUSION, AND
RECOMMENDATIONS FOR FUTURE WORK***

model should act as a ‘brain’ in the whole digital twin. It could overall plan the maintenance schedule and allocate resources. Moreover, it could illustrate the consequence when engineers select different scenarios, such as postpone the maintenance schedule and partial maintenance. These improvements could make maintenance more flexible and smarter.

Bibliography

AKKAYA, I. 2016. *Data-driven cyber-physical systems via real-time stream analytics and machine learning*. UC Berkeley.

AN, D., CHOI, J. H. & KIM, N. H. Options for Prognostics Methods: A review of data-driven and physics-based prognostics. 54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2013. 1940.

ATAMURADOV, V., MEDJAHAR, K., DERSIN, P., LAMOUREUX, B. & ZERHOUNI, N. 2017. Prognostics and health management for maintenance practitioners-review, implementation and tools evaluation. *International Journal of Prognostics and Health Management*, 8, 1-31.

BACIDORE, M. 2019. *The all-knowing digital twin* [Online]. Control Design. Available: <https://www.controldesign.com/articles/2019/the-all-knowing-digital-twin/> [Accessed 11 2019].

BAI, G., WANG, P., HU, C. & PECHT, M. 2014. A generic model-free approach for lithium-ion battery health management. *Applied Energy*, 135, 247-260.

BARROS, A. 2019. *Data Driven Prognostic and Predictive Maintenance*. Norwegian University of Science and Technology.

BEVILACQUA, M., BOTTANI, E., CIARAPICA, F. E., COSTANTINO, F., DI DONATO, L., FERRARO, A., MAZZUTO, G., MONTERIÙ, A., NARDINI, G. & ORTENZI, M. 2020. Digital Twin Reference Model Development to Prevent Operators' Risk in Process Plants. *Sustainability*, 12, 1088.

BIESINGER, F., MEIKE, D., KRAß, B. & WEYRICH, M. 2019. A digital twin for production planning based on cyber-physical systems: A Case Study for a Cyber-Physical System-Based Creation of a Digital Twin. *Procedia CIRP*, 79, 355-360.

BOSCHERT, S. & ROSEN, R. 2016. Digital twin—the simulation aspect. *Mechatronic Futures*. Springer.

CACHADA, A., BARBOSA, J., LEITÃO, P., GCRALDCS, C. A., DEUSDADO, L., COSTA, J., TEIXEIRA, C., TEIXEIRA, J., MOREIRA, A. H. & MOREIRA, P. M. Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture. 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 2018. IEEE, 139-146.

CHAUHAN, N. S. 2019 *Introduction to Artificial Neural Networks(ANN)* [Online]. Available: <https://towardsdatascience.com/introduction-to-artificial-neural-networks-ann-1aea15775ef9> [Accessed].

CHIANG, J. Y., LIO, Y. & TSAI, T. R. 2015. Degradation tests using geometric Brownian motion process for lumen degradation data. *Quality and Reliability Engineering International*, 31, 1797-1806.

CROARKIN, C., TOBIAS, P., FILLIBEN, J., HEMBREE, B. & GUTHRIE, W. 2006.

- NIST/SEMATECH e-handbook of statistical methods. *NIST/SEMATECH*, July. Available online: <http://www.itl.nist.gov/div898/handbook>.
- DING, K., CHAN, F. T. S., ZHANG, X., ZHOU, G. & ZHANG, F. 2019. Defining a Digital Twin-based Cyber-Physical Production System for autonomous manufacturing in smart shop floors. *International Journal of Production Research*, 57, 6315-6334.
- DOSHI, S. Jan 13, 2019. *Various Optimization Algorithms For Training Neural Network* [Online]. Available: <https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6> [Accessed].
- FORTUNER, B. 2019. *Machine learning glossary* [Online]. Available: <http://ml-cheatsheet.readthedocs.io> [Accessed].
- FOUNDATION, P. S. 2020. *Python - Network Programming* [Online]. Available: https://www.tutorialspoint.com/python/python_networking.htm [Accessed].
- FRANGOPOL, D. M. 2011. Life-cycle performance, management, and optimisation of structural systems under uncertainty: accomplishments and challenges 1. *Structure and Infrastructure Engineering*, 7, 389-413.
- GLAESSGEN, E. & STARGEL, D. The digital twin paradigm for future NASA and US Air Force vehicles. 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 20th AIAA/ASME/AHS Adaptive Structures Conference 14th AIAA, 2012. 1818.
- GRIEVES, M. & VICKERS, J. 2017. Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems. *Transdisciplinary perspectives on complex systems*. Springer.
- HU, C., YOUN, B. D. & WANG, P. 2019. *Engineering design under uncertainty and health prognostics*, Springer.
- HYNDMAN, R. J. & ATHANASOPOULOS, G. 2018. *Forecasting: principles and practice*, OTexts.
- IMMERMAN, G. 2018. *The Impact of Predictive Maintenance on Manufacturing* [Online]. Available: <https://www.machinemetrics.com/blog/the-impact-of-predictive-maintenance-on-manufacturing> [Accessed].
- ISO, -. 2019. 13374-1: 2003 Condition Monitoring and Diagnostics of Machines—Data Processing. *Communication and Presentation—Part, 1*.
- ISO/CD, -. 2019. *Digital Twin manufacturing framework — Part 1: Overview and general principles* [Online]. Available: <https://www.iso.org/standard/75066.html> [Accessed].
- JANASAK, K. M. & BESHEARS, R. R. Diagnostics to Prognostics-A product availability technology evolution. 2007 Annual Reliability and Maintainability Symposium, 2007. IEEE, 113-118.
- JESAN, J. P. 2004. The neural approach to pattern recognition. *Ubiquity*, 2004, 2.
- KAISE, T. Reliability Analysis for Degradation Data Based on Stochastic Process Models and Bayesian Estimations. Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications, 2012. The ISCIE Symposium on Stochastic Systems Theory and Its Applications, 246-250.
- KIM, N.-H., AN, D. & CHOI, J.-H. 2016. *Prognostics and health management of engineering*

- systems: An introduction*, springer.
- KOBAYASHI, H., MARK, B. L. & TURIN, W. 2011. *Probability, random processes, and statistical analysis: applications to communications, signal processing, queueing theory and mathematical finance*, Cambridge University Press.
- KRITZINGER, W., KARNER, M., TRAAR, G., HENJES, J. & SIHN, W. 2018. Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51, 1016-1022.
- LE SON, K., FOULADIRAD, M., BARROS, A., LEVRAT, E. & IUNG, B. 2013. Remaining useful life estimation based on stochastic deterioration models: A comparative study. *Reliability Engineering & System Safety*, 112, 165-175.
- LEE, C., CAO, Y. & NG, K. H. 2017. Big data analytics for predictive maintenance strategies. *Supply Chain Management in the Big Data Era*. IGI Global.
- LEE, E. A. 2015. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15, 4837-4869.
- LIAO, W., CHEN, M. & YANG, X. 2017. Joint optimization of preventive maintenance and production scheduling for parallel machines system. *Journal of Intelligent & Fuzzy Systems*, 32, 913-923.
- LIU, Q., DONG, M. & CHEN, F. 2018. Single-machine-based joint optimization of predictive maintenance planning and production scheduling. *Robotics and Computer-Integrated Manufacturing*, 51, 238-247.
- LIU, Q., DONG, M. & PENG, Y. 2013. A dynamic predictive maintenance model considering spare parts inventory based on hidden semi-Markov model. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 227, 2090-2103.
- LU, S., TU, Y. C. & LU, H. 2007. Predictive condition-based maintenance for continuously deteriorating systems. *Quality and Reliability Engineering International*, 23, 71-81.
- LUO, J., NAMBURU, M., PATTIPATI, K., QIAO, L., KAWAMOTO, M. & CHIGUSA, S. Model-based prognostic techniques [maintenance applications]. Proceedings AUTOTESTCON 2003. IEEE Systems Readiness Technology Conference., 2003. IEEE, 330-340.
- MISSINGLINK.AI. 2020. *7 Types of Neural Network Activation Functions: How to Choose?* [Online]. Available: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/> [Accessed].
- MOBLEY, R. K. 2002. *An introduction to predictive maintenance*, Elsevier.
- MOSALLAM, A., MEDJAHHER, K. & ZERHOUNI, N. 2013. Nonparametric time series modelling for industrial prognostics and health management. *The International Journal of Advanced Manufacturing Technology*, 69, 1685-1699.
- MOSALLAM, A., MEDJAHHER, K. & ZERHOUNI, N. Component based data-driven prognostics for complex systems: Methodology and applications. 2015 First International Conference on Reliability Systems Engineering (ICRSE), 2015. IEEE, 1-7.
- MOSALLAM, A., MEDJAHHER, K. & ZERHOUNI, N. 2016. Data-driven prognostic method

- based on Bayesian approaches for direct remaining useful life prediction. *Journal of Intelligent Manufacturing*, 27, 1037-1048.
- NGUYEN, K. T. & MEDJAHHER, K. 2019. A new dynamic predictive maintenance framework using deep learning for failure prognostics. *Reliability Engineering & System Safety*, 188, 251-262.
- PARK, C. & PADGETT, W. 2005. Accelerated degradation models for failure based on geometric Brownian motion and gamma processes. *Lifetime Data Analysis*, 11, 511-527.
- PECHT, M. 2009. Prognostics and health management of electronics. *Encyclopedia of structural health monitoring*.
- PECHT, M. & JAAI, R. 2010. A prognostics and health management roadmap for information and electronics-rich systems. *Microelectronics Reliability*, 50, 317-323.
- PECHT, M. G. 2010. A prognostics and health management roadmap for information and electronics-rich systems. *IEICE ESS Fundamentals Review*, 3, 4_25-4_32.
- QI, Q., TAO, F., ZUO, Y. & ZHAO, D. 2018. Digital twin service towards smart manufacturing. *Procedia CIRP*, 72, 237-242.
- QIAO, Q., WANG, J., YE, L. & GAO, R. X. 2019. Digital Twin for Machining Tool Condition Prediction. *Procedia CIRP*, 81, 1388-1393.
- RAMAN, V. & HASSANALY, M. 2019. Emerging trends in numerical simulations of combustion systems. *Proceedings of the Combustion Institute*, 37, 2073-2089.
- ROCCHETTA, R., BELLANI, L., COMPARE, M., ZIO, E. & PATELLI, E. 2019. A reinforcement learning framework for optimal operation and maintenance of power grids. *Applied energy*, 241, 291-301.
- ROSEN, R., VON WICHERT, G., LO, G. & BETTENHAUSEN, K. D. 2015. About the importance of autonomy and digital twins for the future of manufacturing. *IFAC-PapersOnLine*, 48, 567-572.
- SAXENA, A., GOEBEL, K., SIMON, D. & EKLUND, N. Damage propagation modeling for aircraft engine run-to-failure simulation. 2008 international conference on prognostics and health management, 2008. IEEE, 1-9.
- SCHEIFELE, C., VERL, A. & RIEDEL, O. 2019. Real-time co-simulation for the virtual commissioning of production systems. *Procedia CIRP*, 79, 397-402.
- SEYR, H. & MUSKULUS, M. 2019. Decision Support Models for Operations and Maintenance for Offshore Wind Farms: A Review. *Applied Sciences*, 9, 278.
- SHARMA, S. 2017 *Activation Functions in Neural Networks* [Online]. Available: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6> [Accessed].
- SHLENS, J. 2014. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*.
- SI, X.-S., HU, C.-H., CHEN, M.-Y. & WANG, W. An adaptive and nonlinear drift-based Wiener process for remaining useful life estimation. 2011 Prognostics and System Health Management Conference, 2011a. IEEE, 1-5.
- SI, X.-S., WANG, W., HU, C.-H. & ZHOU, D.-H. 2011b. Remaining useful life estimation—

- review on the statistical data driven approaches. *European journal of operational research*, 213, 1-14.
- SINGH, A. 2018. A Practical Introduction to K-Nearest Neighbors Algorithm for Regression (with Python code). Accès à <https://www.analyticsvidhya.com/blog/2018/08/k-nearestneighbor-introduction-regression-python>.
- SÖDERBERG, R., WÄRMEFJORD, K., CARLSON, J. S. & LINDKVIST, L. 2017. Toward a Digital Twin for real-time geometry assurance in individualized production. *CIRP Annals*, 66, 137-140.
- TAO, F., CHENG, J., QI, Q., ZHANG, M., ZHANG, H. & SUI, F. 2018a. Digital twin-driven product design, manufacturing and service with big data. *The International Journal of Advanced Manufacturing Technology*, 94, 3563-3576.
- TAO, F., QI, Q., LIU, A. & KUSIAK, A. 2018b. Data-driven smart manufacturing. *Journal of Manufacturing Systems*, 48, 157-169.
- TAO, F., SUI, F., LIU, A., QI, Q., ZHANG, M., SONG, B., GUO, Z., LU, S. C. Y. & NEE, A. Y. C. 2019. Digital twin-driven product design framework. *International Journal of Production Research*, 57, 3935-3953.
- TAVNER, P. 2012a. *Offshore wind turbines: reliability, availability and maintenance*, The Institution of Engineering and Technology.
- TAVNER, P. 2012b. *Offshore wind turbines: reliability, availability and maintenance*, IET.
- VEMURI, S. 2019. *Why Is Predictive Maintenance Important?* [Online]. Available: <https://www.digitaldoughnut.com/articles/2018/january/why-is-predictive-maintenance-important> [Accessed].
- VENKATESAN, S., MANICKAVASAGAM, K., TENGENKAI, N. & VIJAYALAKSHMI, N. 2019. Health monitoring and prognosis of electric vehicle motor using intelligent-digital twin. *IET Electric Power Applications*, 13, 1328-1335.
- WANG, J. 2019. Predictive maintenance and Digital twins. TPK4550 Specialization project.
- WANG, T., YU, J., SIEGEL, D. & LEE, J. A similarity-based prognostics approach for remaining useful life estimation of engineered systems. 2008 international conference on prognostics and health management, 2008. IEEE, 1-6.
- WEGENER, P. 2019. GERMAN STANDARDIZATION ROADMAP Industrie 4.0 Version 3. *DIN e*, 2018.
- WENINGER, R. 2020. *What Do the Next Five Years Hold For the IoT?* [Online]. Innovation & Technology Business School. [Accessed 5.20 2020].
- WOLD, S., ESBENSEN, K. & GELADI, P. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2, 37-52.
- ZHANG, L., MU, Z. & SUN, C. 2018. Remaining useful life prediction for lithium-ion batteries based on exponential model and particle filter. *IEEE Access*, 6, 17729-17740.

Appendix A

Acronyms

| | |
|--------------|---|
| ANN | Artificial Neural Network |
| ATF | Automation factory |
| CNN | convolutional neural network |
| CPS | Cyber-Physical Systems |
| FMECA | Failure mode, effects, and criticality analysis |
| FTA | Fault tree analysis |
| GBM | Geometric Brownian motion |
| HI | Health indicator |
| IoT | internet of things |
| IP | Internet Protocol address |
| k-NN | k-Nearest-Neighbor |
| LSTM | Long short-term memory |
| MQTT | Message Queuing Telemetry Transport |
| MSB | Media Stream Broadcast |
| MSE | Mean squared error |
| NPV | Net present value |
| PCA | Principal component analysis |
| PDF | Probability density function |
| PHI | Physics Health Indicator |

PHM prognostics and health management

RAMS Reliability availability maintainability and safety

RNN Recurrent Neural Networks

RUL Remaining useful if life

SDE Stochastic differential equation

SVD Singular value decomposition

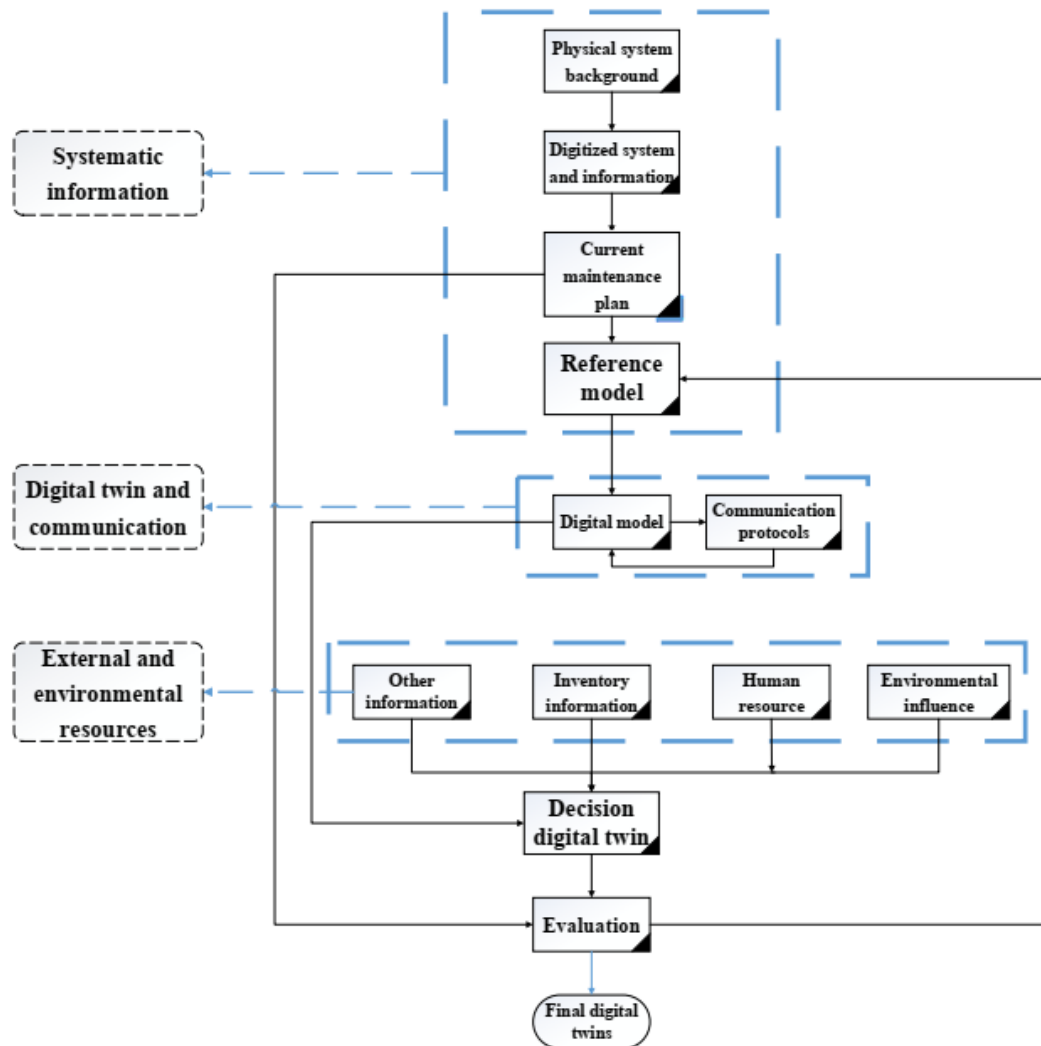
TCP Transmission control protocol

TSC Time series components

VHI Virtual health indicator

Appendix B

Roadmap for Predictive maintenance Digital twin



Appendix C

Python codes of digital twin architecture

C.1 Communication protocol in Socket

C1.1 Socket listening file transferring 'Server'

```
1. """
2. file: recv.py
3. socket service
4. """
5.
6. import socket
7. import tqdm
8. import os
9.
10. # device's IP address
11. SERVER_HOST = input("Please input server Ip: ")
12. SERVER_PORT = 5003
13. # receive 4096 bytes each time
14. BUFFER_SIZE = 4096
15. SEPARATOR = "<SEPARATOR>"
16. # create the server socket
17. # TCP socket
18. s = socket.socket()
19. # bind the socket to our local address
20. s.bind((SERVER_HOST, SERVER_PORT))
21. # enabling our server to accept connections
22. # 5 here is the number of unaccepted connections that
23. # the system will allow before refusing new connections
24. s.listen(5)
25. os.chdir(input("please input save location:"))
26. print(f"[*] Listening as {SERVER_HOST}:{SERVER_PORT}")
27. # accept connection if there is any
28. client_socket, address = s.accept()
29. # if below code is executed, that means the sender is connected
30. print(f"[+] is connected.")
31. # receive the file infos
32. # receive using client socket, not server socket
33. received = client_socket.recv(BUFFER_SIZE).decode()
34. filename, filesize = received.split(SEPARATOR)
35. # remove absolute path if there is
36. filename = os.path.basename(filename)
37. # convert to integer
38. filesize = int(filesize)
39. # start receiving the file from the socket
40. # and writing to the file stream
41. progress = tqdm.tqdm(range(filesize), f"Receiving {filename}", unit="B", uni
    t_scale=True, unit_divisor=1024)
```

```

42. with open(filename, "wb") as f:
43.     for _ in progress:
44.         # read 1024 bytes from the socket (receive)
45.         bytes_read = client_socket.recv(BUFFER_SIZE)
46.         if not bytes_read:
47.             # nothing is received
48.             # file transmitting is done
49.             break
50.         # write to the file the bytes we just received
51.         f.write(bytes_read)
52.         # update the progress bar
53.         progress.update(len(bytes_read))
54.
55.
56. # close the client socket
57. client_socket.close()
58. # close the server socket
59. s.close()

```

C1.2 Socket listening file transferring 'Client'

```

1. import socket
2. import tqdm
3. import os
4.
5. SEPARATOR = "<SEPARATOR>"
6. BUFFER_SIZE = 4096 # send 4096 bytes each time step
7.
8. # the ip address or hostname of the server, the receiver
9. host = input("Please input server ip:")
10. # the port, let's use 5001
11. port = 5003
12. # the name of file we want to send, make sure it exists
13. filename = "D:\programming\Knn regression.ipynb"
14. # get the file size
15. filesize = os.path.getsize(filename)
16.
17. # create the client socket
18. s = socket.socket()
19.
20. print(f"[+] Connecting to :{port}")
21. s.connect((host, port))
22. print("[+] Connected.")
23.
24. # send the filename and filesize
25. s.send(f"{filename}{SEPARATOR}{filesize}".encode())
26.
27. # start sending the file
28. progress = tqdm.tqdm(range(filesize), f"Sending {filename}", unit="B", unit_
    scale=True, unit_divisor=1024)
29. with open(filename, "rb") as f:
30.     for _ in progress:
31.         # read the bytes from the file
32.         bytes_read = f.read(BUFFER_SIZE)
33.         if not bytes_read:
34.             # file transmitting is done
35.             break

```

```

36.     # we use sendall to assure transimission in
37.     # busy networks
38.     s.sendall(bytes_read)
39.     # update the progress bar
40.     progress.update(len(bytes_read))
41. # close the socket
42. s.close()

```

C1.3 Socket monitoring file transferring ‘Server’

```

1.  '''
2.  server:
3.  '''
4.  import socket
5.  server = socket.socket()
6.  server.bind(("192.168.137.1",5001)) #port and ip
7.  server.listen(5) # listing, mostly have five connection
8.  print('waiting the call')
9.  while True:
10.     conn,addr = server.accept() # waiting for caling
11.     print(conn)
12.     print('the call has comming')
13.     while True:
14.         data = conn.recv(1024)
15.         if not data :
16.             print('this user is end,exit!\n next user')
17.             break
18.         print('data:',data.decode())
19.         conn.send(data.upper())

```

C1.4 Socket monitoring file transferring ‘Client’

```

1.  import socket
2.  import os
3.
4.  SEPARATOR = "<SEPARATOR>"
5.  BUFFER_SIZE = 4096 # send 4096 bytes each time step
6.
7.  # the ip address or hostname of the server, the receiver
8.  host = input("Please input server ip:")
9.  # the port, let's use 5001
10. port = 5001
11. # the name of file we want to send, make sure it exists
12.
13.
14. # create the client socket
15. s = socket.socket()
16.
17. print(f"[+] Connecting to {host}:{port}")
18. s.connect((host, port))

```

```

19. print("[+] Connected.")
20.
21.
22. while True:
23.     msg = input("message:").strip()
24.
25.     if len(msg) == 0:
26.         continue
27.
28.     s.send(msg.encode(encoding='utf-8')) # not allowed empty message
29.
30.     data = s.recv(1024)
31.     print(data.decode())
32.
33. s.close()

```

C.2 Data pre-processing and analysis

```

1. # coding: utf-8
2.
3. # In[125]:
4.
5. # package import
6.
7. import pandas as pd
8. import os
9. import numpy as np
10. import matplotlib.pyplot as plt
11. import seaborn as sns
12.
13. import importlib
14. from sklearn.linear_model import LinearRegression
15. from sklearn import preprocessing
16. from sklearn.preprocessing import StandardScaler
17. from sklearn.decomposition import PCA
18. # from sklearn.feature_selection import f_regression
19. from sklearn.linear_model import LogisticRegression
20. from scipy.signal import savgol_filter
21. from scipy.optimize import curve_fit
22. from sklearn.model_selection import TimeSeriesSplit
23. from sklearn.metrics.pairwise import euclidean_distances
24. get_ipython().run_line_magic('matplotlib', 'inline')
25.
26. # In[76]:
27.
28. ## data import
29. dirname = os.getcwd()
30. data_pth_train = os.path.join(dirname, 'training', 'train_FD001.txt')
31. data_pth_rul = os.path.join(dirname, 'rul', 'RUL_FD001.txt')
32. # column names for the dataset
33. # op_cond refers to operational condition, sn: sensor
34. col_name = ['machine', 'time/cycle', 'op_cond_1', 'op_cond_2', 'op_cond_3']
35. col_name = col_name + ['sn_{}'.format(s + 1) for s in range(21)]
36.
37. # In[77]:

```

```

38.
39. # load data into sensor df
40. # notice index_col=0 is used so that the data for each separate engine can b
    e obtained easily
41. #(engine columns is just a group of data)
42. df_train = pd.read_csv(data_ptn_train, header=None, names=col_name,delim_whi
    tspace=True,index_col=0)
43. TrueRUL= pd.read_csv(data_ptn_rul, sep = "\n", header = None)
44.
45. # In[78]:
46.
47. TrueRUL.columns= ['RUL']
48. TrueRUL.head()
49.
50. # In[79]:
51.
52. # Take a look at data
53. #df_train.head()
54. #pd.set_option('max_rows', 10,'max_columns',8)
55.
56. # In[80]:
57.
58. df_train
59.
60. # so basically there are 218 engines in the data set
61.
62. # In[81]:
63.
64. df_train.info()
65.
66. # There is no missing data or Null data
67.
68. # In[82]:
69.
70. df_train.describe()
71.
72. # In[83]:
73.
74. # look at the mean of all columns
75. df_train.std().plot.bar(figsize=(12,8));
76. plt.title('std value of signals')
77.
78. # In[84]:
79.
80. df_train.mean()
81.
82. # In[85]:
83.
84. df_train_data1 = df_train.loc[1,"time/cycle" : "sn_21"]
85. df_train_data1.info()
86. #pd.set_option('max_rows', 5,'max_columns',8)
87. # In[86]:
88. df_train_data1
89. # In[87]:
90. df_train_data1.describe()
91. # In[88]:
92. df_train_data1.mean().plot.bar(figsize=(12,8));
93. # Examine Correlation Between Columns (linear)
94. # In[89]:
95. #For all
96. #heatmap to view correlation between independent variables
97. correlations = df_train.drop('time/cycle',axis = 1).corr()
98. #corr_map = sns.diverging_palette(220, 10, as_cmap=True)

```

```

99. f, ax = plt.subplots(figsize=(30,30))
100.     sns.heatmap(correlations, annot=True, linewidths=0.5,linecolor="red",
    fmt= '.1f',ax=ax)
101.     plt.title('The correlation map of all machines ',color = 'black',font
    size = 50)
102.     plt.show()
103.     # In[90]:
104.     # correaltion for engine 1
105.     #engine_num=1
106.     f,ax = plt.subplots(figsize=(30, 30))
107.     sns.heatmap(df_train_data1.corr(), annot=True, linewidths=0.5,linecol
    or="red", fmt= '.1f',ax=ax)
108.     plt.title('The correlation map of machine 1',color = 'black',fontsize
    = 50)
109.     plt.show()
110.     # R value in the figure
111.     # From these two figure , we could say there are alot variables have
    correlations and in order to find the correlation part
112.     # only consider the positive values
113.     # and the totally value figure is not that different compared with th
    e first machine
114.     #
115.     # Then categorize the highly correlation pairs
116.     # In[91]:
117.
118.     def find_corr_pairs(corr,thrsh):
119.
120.         """
121.         find high correlation column pairs in df
122.         =====
123.         input:
124.         corr - (df)- correlation matrix generated by pandas
125.         thrsh - (float) threshold value to consider correlation as high s
    o that it is included in the output
126.         output:
127.         high_corr_pairs - (list) list of tuples of the two-
    column names and their correlation. corr> thrsh
128.         """
129.         high_corr_pairs = []
130.         # same as input 'corr' but the upper -
    triangle half of the matrix is zeros ( for convenience only)
131.         corr_diag = pd.DataFrame(np.tril(corr.values), columns=corr.columns,
    index = corr.index)
132.
133.         # check the correlation between every pair of columns in the cor
    r and keeps the high ones
134.         for col_num , col in enumerate(corr_diag):
135.             col_corr=corr_diag[col].iloc[col_num+1:] # this slicing ensur
    es ignoring self_corr and duplicates due to symmetry
136.             # bool mask for pairs with high corr with col
137.             mask_pairs = col_corr.apply(lambda x: abs(x))>thrsh
138.             idx_pairs=col_corr[mask_pairs].index
139.
140.             # create list of high corr pairs
141.             for idx , corr in zip(idx_pairs,col_corr[mask_pairs].values):
142.
143.                 high_corr_pairs.append((col, idx, corr))
144.
145.         return high_corr_pairs
146.     # In[92]:
147.
148.     corr_pairs=find_corr_pairs(correlations,0.9)

```

```

149.     for c in corr_pairs:
150.         print(c)
151.         # the value above are correlated with others , however, we need to check if they really correlated.
152.         # then we will plot the signal PDF for each sensor.
153.         #
154.         # plot variables distribution
155.         # In[93]:
156.         def plot_distribution(df, engine_num=None):
157.             '''plot all non trivial measurements and states'''
158.
159.             cols = df.columns
160.             n_cols = min(len(cols), 5)
161.             n_rows = int(np.ceil(len(cols) / n_cols))
162.
163.             sns.set()
164.             fig, axes = plt.subplots(n_rows, n_cols, figsize=(15,12))
165.             axes = axes.flatten()
166.             if engine_num != None:
167.                 fig.suptitle('distributions for Machines #: {}'.format(engine
168. _num))
169.                 df_plot = df.loc[engine_num]
170.             else:
171.                 fig.suptitle('distributions for all Machines')
172.                 df_plot = df
173.             for col, ax in zip(cols, axes):
174.                 ax=sns.distplot(df_plot[col], ax=ax, label=col)
175.                 ax.legend(loc=1)
176.                 # labels(col, "p", ax)
177.             return fig
178.         # In[94]:
179.         fig=plot_distribution(df_train)
180.         # In[95]:
181.
182.         df_train_std = df_train.drop('time/cycle',axis=1)
183.
184.         df_train_std.std().plot.bar(figsize=(12,8));
185.
186.         # In[96]:
187.         df_train_std.std()
188.         # dropping insignificant variables
189.         # remove the variable in the dataset
190.         # In[97]:
191.         variable_remove=[ col for col in df_train.columns if (df_train[col].s
192. td() <= .0001) ]
193.         print('columns to be removed from analysis since they do not change w
194. ith time \n',variable_remove)
195.         # In[98]:
196.         df_train.drop(columns=variable_remove,axis=1,inplace=True)
197.         # In[99]:
198.         def plot_ts(df, engine_num):
199.             """
200.             plot time history of for specific engine
201.             =====
202.             input:
203.             df - (df) Dataframe you wish to plot the time series for its columns
204.             engine_num - (int) engine number to selector
205.             """
206.             # prepare the dataframe for plotting
207.             ts = df.loc[engine_num].copy() # df for the needed engine

```



```

207.         time = ts['time/cycle']
208.         ts.drop(labels=['time/cycle'],axis=1,inplace=True)
209.         cols = ts.columns
210.
211.         # plotting
212.         fig, axes = plt.subplots(len(cols), 1, figsize=(19,17))
213.         for col, ax in zip(cols, axes):
214.             ax.plot(time,ts[col],label=col)
215.             ax.legend(loc=2)
216.
217.         # figure title
218.         fig.suptitle('Engine #: {}'.format(engine_num))
219.
220.     # In[100]:
221.     fig_signal1= plot_ts(df_train, 1)
222.     # In[101]:
223.     # add RUL to each engine based on time column,
224.     # notice that RUL is negative quantity here to make 0 as the end of 1
    ife for all engines
225.     for id in df_train.index.unique():
226.         df_train.loc[id,'RUL'] = df_train.loc[id]['time/cycle'].apply(lam
    bda x: x-df_train.loc[id]['time/cycle'].min())
227.     # In[102]:
228.     def plot_ts_all(df):
229.         """
230.         plot time history of for all engines in the data
231.         =====
232.         input:
233.         df - (df) Dataframe you wish to plot the time series for its col
    umns
234.         """
235.
236.         # prepare the dataframe for plotting
237.         ts = df.copy() # df for the needed engine
238.         ts.drop(labels=['time/cycle'],axis=1,inplace=True)
239.
240.         cols = ts.columns
241.         # plotting
242.         fig, axes = plt.subplots(len(cols)-1, 1, figsize=(19,17))
243.         for col, ax in zip(cols, axes):
244.             if col == 'RUL':
245.                 continue
246.
247.             fontdict = {'fontsize': 12}
248.             ax.set_title(col,loc='left',fontdict=fontdict)
249.             for engine_id in ts.index.unique():
250.                 time = ts.loc[engine_id,'RUL']
251.                 ax.plot(time,ts.loc[engine_id,col],label=col)
252.
253.             # figure title
254.
255.         return fig
256.     # In[103]:
257.     fig=plot_ts_all(df_train)
258.     fig.suptitle('All Machine Time Series \n each line is different machi
    ne response\n x-axis is lifetime')
259.     # In[60]:
260.
261.     df_train.drop(['sn_2','sn_3' , 'sn_4' , 'sn_7','sn_8','sn_11','sn_12',
    'sn_13','sn_15','sn_17','sn_20','sn_21'],axis=1,inplace=True)
262.     # In[104]:
263.     fig=plot_ts_all(df_train)

```

```

264.     fig.suptitle('All Machine Time Series \n each line is different machi
ne response\n x-axis is lifetime')
265.     # In[105]:
266.     df_train.drop(['op_cond_1','op_cond_2' , 'sn_6' , 'sn_9' , 'sn_14'],axi
s=1,inplace=True)
267.     df_train.shape
268.     # In[143]:
269.     for id in df_train.index.unique():
270.         df_train.loc[id,'RUL'] = df_train.loc[id]['time/cycle'].apply(lam
bda x: x-df_train.loc[id]['time/cycle'].min())
271.         # positive RUL for each engine
272.
273.         # get all sensors
274.         raw_columns = df_train.columns.values[1:-1]
275.         raw_sensors = df_train[raw_columns].values # as numpy array
276.         raw_columns
277.
278.         # In[107]:
279.         standard_scale = StandardScaler()
280.         standard_sensors = standard_scale.fit_transform(raw_sensors)
281.         # In[108]:
282.         lin_model =LinearRegression()
283.         engine_num=20
284.         x = df_train.loc[engine_num,'RUL'].values
285.         # In[109]:
286.         row_name=df_train.loc[engine_num].iloc[-1].name
287.         row_sl=df_train.index.get_loc(row_name) # row slice to get numpy inde
x
288.         y=standard_sensors[row_sl] # sensor values for the specific engine
289.         x.reshape(-1, 1).shape # add dimension, to fit the line
290.         x.shape
291.         lin_model.fit(x.reshape(-1, 1),y)
292.
293.         # In[110]:
294.
295.         lin_model.coef_[:,0].shape
296.         # how many line has been fitted
297.
298.         # In[111]:
299.
300.         lin_model.score(x.reshape(-1, 1),y)
301.         # Return the coefficient of determination R^2 of the prediction.
302.
303.         # In[112]:
304.
305.         y_hat = lin_model.predict(x.reshape(-1, 1))
306.
307.         # In[113]:
308.
309.         # plotting
310.         time = df_train.loc[engine_num,'RUL']
311.         cols = df_train.columns[1:-1]
312.         fig, axes = plt.subplots(len(cols), 1, figsize=(19,17))
313.         for col, ax in zip(range(standard_sensors.shape[1]), axes):
314.             ax.plot(time,standard_sensors[row_sl,col],label=col+1)
315.             ax.plot(time,y_hat[:,col],label='trend')
316.             ax.legend(loc=2)
317.
318.         # In[114]:
319.         def lin_slopes(sensors,df,engine_num):
320.             """
321.             gives slopes of a teh tred lines for each sesnor
322.             =====

```

```

323.         input:
324.         sensors - (ndarray) numpy array of standardized signals ( rows: -
RUL columns, various signals)
325.         engine_num - (int) engine number to selector
326.         df - (df) data frame of data
327.         output:
328.         slopes -
(ndarray) numpy array of slopes rows: slope of each signal linear trend line

329.         """
330.         model = LinearRegression()
331.         x = df.loc[engine_num,'RUL'].values
332.         row_name=df.loc[engine_num].iloc[-1].name
333.         row_sl=df.index.get_loc(row_name) # row slice to get numpy index

334.         y=sensors[row_sl] # sensor values for the specific engine
335.         model.fit(x.reshape(-1, 1),y)
336.         slopes=model.coef_[0]
337.         return slopes
338.
339.         # In[115]:
340.         # finding slopes for all engines
341.         engines=df_train.index.unique().values
342.         slopes = np.empty((standard_sensors.shape[1],len(engines)))
343.         for i,engine in enumerate(engines):
344.             slopes[:,i] = lin_slopes(standard_sensors,df_train,engine)
345.
346.         # In[116]:
347.         slopes_df = pd.DataFrame(slopes.T,index=engines,columns =raw_columns)

348.         slopes_df
349.         # In[117]:
350.         for sn in slopes_df.columns[1:-1]:
351.             plt.plot(abs(slopes_df[sn]))
352.         plt.show()
353.
354.         # engine 20
355.         # From the pic above, we can find that , mainly there are two signal
of slope can reveal the potential trend of
356.         # all sensors , so in the following , we are going to use PCA to extr
act the main feature of several signals.
357.         #
358.         # first, we are going to get the slopes of each linear model.
359.
360.         # In[118]:
361.         slope_order_idx=np.argsort(np.abs(slopes.mean(axis=1)))[::-1]
362.         raw_columns[slope_order_idx]
363.         # In[119]:
364.         # PCA with all sensors engine 20
365.         # get first impression of how many PCA it might have
366.         pca = PCA()
367.         # In[120]:
368.         num_high_slopes = 6
369.         pca_high_n_components=3
370.         sensors_high_trend=standard_sensors[:,slope_order_idx[0:num_high_slop
es]]
371.         pca_high = PCA(pca_high_n_components,whiten=True)
372.         pca_high.fit(sensors_high_trend)
373.         # In[121]:
374.         pca_high.explained_variance_ratio_
375.         # In[128]:
376.         sensors_pca=pca_high.transform(sensors_high_trend)
377.         sensors_pca

```

```

378. # In[122]:
379. # create a dictionary with engine slices
380. engines=df_train.index.unique().values # engine numbers
381. engine_slices = dict()# key is engine number, value is a slice that g
    ives numpy index for the data that pertains to an engine
382.
383. for i,engine_num in enumerate(engines):
384.     row_name=df_train.loc[engine_num].iloc[-1].name
385.     row_sl=df_train.index.get_loc(row_name) # row slice to get numpy
    index
386.     engine_slices[engine_num]=row_sl
387. # In[136]:
388. # create RUL vector
389. RUL = np.empty(len(engines))
390.
391. for i,engine_num in enumerate(engines):
392.     RUL[i]=df_train.loc[engine_num]['RUL'].max()
393.
394. # In[137]:
395. # fit a model to get fused sensor
396. #Multiple Linear Regression
397. HI_linear = LinearRegression()
398. data_scaler = preprocessing.MinMaxScaler()
399. preprocessing.MinMaxScaler()
400. # In[144]:
401. engine_num=20
402. engine_sensors=sensors_pca[engine_slices[engine_num],:]
403. RUL_engine = df_train.loc[engine_num]['RUL'].values
404. # In[145]:
405. y1_engine = engine_sensors[:,0]# sn_11
406. #y1_engine = engine_sensors[:,0]# sn_11
407. y2_engine = engine_sensors[:,1]
408. y3_engine = engine_sensors[:,2]
409. x1_engine = RUL_engine = df_train.loc[engine_num]['RUL']
410. x2_engine = RUL_engine = df_train.loc[engine_num]['RUL'] #df_op.loc[e
    ngine_num]['op_cond_1']
411. x3_engine = RUL_engine = df_train.loc[engine_num]['RUL']#df_op.loc[e
    ngine_num]['op_cond_2']
412. fig, axes = plt.subplots(3, 1, figsize=(19,17))
413. plt.subplot(311)
414. plt.plot(x1_engine,y1_engine,label='PCA_rank1')
415. plt.title('Health Index ')
416.
417. plt.legend()
418. plt.subplot(312)
419. plt.plot(x2_engine,y2_engine,label='PCA_rank2')
420. plt.title('Health Index ')
421.
422. plt.legend()
423. plt.subplot(313)
424. plt.plot(x3_engine,y3_engine,label='PCA_rank3')
425. plt.legend()
426. plt.title('Health Index ')
427. plt.xlabel('lifetime [cycles]')
428.
429. plt.legend()
430. # In[146]:
431. op_column = ['sn_11']
432. df_op = df_train[op_column]
433.
434. # In[147]:
435. engine_num=3
436. #engine_sensors=sensors_pca[engine_slices[engine_num],:]

```

```

437.     RUL_engine = df_train.loc[engine_num]['RUL'].values
438.     engine_sensors_o= df_op.values[engine_slices[engine_num],:]
439.     # In[148]:
440.     plt.plot(RUL_engine,engine_sensors_o[:,0],label='x1')
441.     # In[149]:
442.     from pandas import Series
443.     from matplotlib import pyplot
444.     from statsmodels.tsa.seasonal import seasonal_decompose
445.     result = seasonal_decompose(engine_sensors_o[:,0], model='multiplicat
ive', freq=10)#
446.     trend= result.trend
447.     plt.plot(trend)
448.     pyplot.show()
449.     # In[150]:
450.
451.     result.plot()
452.     pyplot.show()
453.
454.     # In[151]:
455.
456.     y1_engine = trend
457.     x1_engine = df_train.loc[engine_num]['RUL']
458.     plt.plot(x1_engine,y1_engine,label='x1')
459.     # In[152]:
460.     x_tran= engine_sensors_o[:,0]-
np.min(engine_sensors_o[:,0])+0.000001#
461.     result = seasonal_decompose(x_tran, model='multiplicative', freq=10)#
462.
463.     trend= result.trend
464.     plt.plot(trend)
465.     pyplot.show()
466.     # In[153]:
467.     HI_x= np.delete(RUL_engine,np.argwhere(np.isnan(trend)))
468.     transfer_y = trend[~np.isnan(trend)]
469.     plt.plot(HI_x,transfer_y)
470.     # In[155]:
471.     df_curvefitting = pd.DataFrame()
472.     df_midfitting = pd.DataFrame()
473.     # In[156]:
474.     for engine_num in engines:
475.         engine_sensors_o= df_op.values[engine_slices[engine_num],:]
476.         x_tran= engine_sensors_o[:,0]-
np.min(engine_sensors_o[:,0])+0.000001
477.         RUL_engine = df_train.loc[engine_num]['RUL'].values
478.         result = seasonal_decompose(x_tran, model='multiplicative', freq=
10)#
479.         trend= result.trend
480.         HI_x= np.delete(RUL_engine,np.argwhere(np.isnan(trend)))
481.         transfer_y = trend[~np.isnan(trend)]
482.         HI_y= transfer_y-np.min(transfer_y)
483.         #df_midfitting = pd.DataFrame({'RUL': HI_x, 'HI': HI_y}, columns=
['RUL', 'HI'])
484.
485.         df_midfitting = pd.DataFrame(data=HI_y.reshape(1,-1))
486.         df_curvefitting=df_curvefitting.append(df_midfitting)
487.         #plt.scatter(HI_x,HI_y,label='raw')
488.         plt.plot(HI_x,HI_y,label='exp_fitted')
489.         #plt.axhline(y=1.0, linestyle='-',lw=2)
490.         plt.title('Health Index (HI)')
491.         plt.xlabel('RUL [cycles]')
492.         plt.ylabel('HI [-]')

```

C.3 k-NN prognostics model (Including offline reference model and online prognostics model)

```
1. # coding: utf-8
2.
3. # In[1]:
4.
5.
6. # package import
7.
8. import pandas as pd
9. import os
10. import numpy as np
11. import matplotlib.pyplot as plt
12. import seaborn as sns
13. import importlib
14. from sklearn.linear_model import LinearRegression
15. from sklearn.preprocessing import StandardScaler
16. from sklearn.decomposition import PCA
17. # from sklearn.feature_selection import f_regression
18. from sklearn.linear_model import LogisticRegression
19. from scipy.signal import savgol_filter
20. from scipy.optimize import curve_fit
21. from sklearn.model_selection import TimeSeriesSplit
22. from sklearn.metrics.pairwise import euclidean_distances
23. from statsmodels.tsa.holtwinters import ExponentialSmoothing , HoltWintersRe
    sults
24. from sklearn import preprocessing
25.
26. get_ipython().run_line_magic('matplotlib', 'inline')
27.
28. # In[2]:
29.
30. ## data import
31. dirname = os.getcwd()
32. data_pth_train = os.path.join(dirname, 'training', 'train_FD001.txt')
33. data_pth_rul = os.path.join(dirname, 'rul', 'RUL_FD001.txt')
34. # column names for the dataset
35. # op_cond refers to operational condition, sn: sensor
36. col_name = ['engine', 'time/cycle', 'op_cond_1', 'op_cond_2', 'op_cond_3']
37. col_name = col_name + ['sn_{}'.format(s + 1) for s in range(21)]
38. df_train = pd.read_csv(data_pth_train, header=None, names=col_name, delim_whi
    tespace=True, index_col=0)
39.
40. # In[3]:
41.
42. variable_remove=[ col for col in df_train.columns if (df_train[col].std() <=
    .0001) ]
43.
44. print('columns to be removed from analysis since they do not change with tim
    e \n', variable_remove)
45.
46. # In[4]:
47.
48. df_train.drop(columns=variable_remove,axis=1,inplace=True)
49.
50. op_column = ['sn_11']
51. df_op = df_train[op_column]
```

```

52.
53. # In[5]:
54.
55. for id in df_train.index.unique():
56.     df_train.loc[id, 'RUL'] = df_train.loc[id]['time/cycle'].apply(lambda x:
57.         x-df_train.loc[id]['time/cycle'].min())
58. # positive RUL for each engine
59. # In[6]:
60.
61. # get all sensors
62. raw_columns = df_train.columns.values[1:-1]
63. raw_sensors = df_train[raw_columns].values # as numpy array
64.
65. # In[7]:
66.
67. # create a dictionary with engine slices
68. engines=df_train.index.unique().values # engine numbers
69. engine_slices = dict()# key is engine number, value is a slice that gives nu
70.     mpy index for the data that pertains to an engine
71. for i,engine_num in enumerate(engines):
72.     row_name=df_train.loc[engine_num].iloc[-1].name
73.     row_sl=df_train.index.get_loc(row_name) # row slice to get numpy index
74.     engine_slices[engine_num]=row_sl
75. # In[8]:
76. RUL = np.empty(len(engines))
77. for i,engine_num in enumerate(engines):
78.     RUL[i]=df_train.loc[engine_num]['RUL'].max()
79. # In[9]:
80. engine_num=1
81. #engine_sensors=sensors_pca[engine_slices[engine_num],:]
82. RUL_engine = df_train.loc[engine_num]['RUL'].values
83. engine_sensors_o= df_op.values[engine_slices[engine_num],:]
84. # In[10]:
85. plt.plot(RUL_engine,engine_sensors_o[:,0],label='x1')
86. # In[11]:
87. from pandas import Series
88. from matplotlib import pyplot
89. from statsmodels.tsa.seasonal import seasonal_decompose
90. result = seasonal_decompose(engine_sensors_o[:,0], model='multiplicative', f
91.     req=10)#
92. trend= result.trend
93. plt.plot(trend)
94. pyplot.show()
95. # In[12]:
96. x_tran= engine_sensors_o[:,0]-np.min(engine_sensors_o[:,0])+0.000001#
97. result = seasonal_decompose(x_tran, model='multiplicative', freq=10)#
98. trend= result.trend
99. plt.plot(trend)
100.     pyplot.show()
101.     # In[13]:
102.     HI_x= np.delete(RUL_engine,np.argwhere(np.isnan(trend)))
103.     transfer_y = trend[~np.isnan(trend)]
104.     #transfer_y= np.nan_to_num(trend)
105.     #HI_linear.fit(HI_x,HI_y)
106.     #print('Coefficients: \n', HI_linear.coef_)
107.     plt.plot(HI_x,transfer_y)
108.
109.     # In[14]:
110.

```

```

111.     df_curvefitting = pd.DataFrame()
112.     df_midfitting = pd.DataFrame()
113.     for engine_num in engines:
114.         engine_sensors_o= df_op.values[engine_slices[engine_num],:]
115.         x_tran= engine_sensors_o[:,0]-
116.         np.min(engine_sensors_o[:,0])+0.000001
117.         RUL_engine = df_train.loc[engine_num]['RUL'].values
118.         result = seasonal_decompose(x_tran, model='multiplicative', freq=
119.         10)#
120.         trend= result.trend
121.         HI_x= np.delete(RUL_engine,np.argmax(np.isnan(trend)))
122.         transfer_y = trend[~np.isnan(trend)]
123.         HI_y= transfer_y-np.min(transfer_y)
124.         #df_midfitting = pd.DataFrame({'RUL': HI_x, 'HI': HI_y}, columns=
125.         ['RUL', 'HI'])
126.         df_midfitting = pd.DataFrame(data=HI_y.reshape(1,-1))
127.         df_curvefitting=df_curvefitting.append(df_midfitting)
128.     # In[15]:
129.
130.     df_curvefitting=df_curvefitting.reset_index()
131.     df_curvefitting.drop(['index'],axis=1,inplace=True)#
132.     #df_curvefitting.interpolate(method='nearest', limit_direction='forwa
133.     rd')
134.     # In[16]:
135.
136.     df_curvefitting=df_curvefitting.interpolate(axis=1)
137.
138.     # In[17]:
139.
140.     pd.set_option('max_rows', 9,'max_columns',9)
141.
142.     # In[18]:
143.
144.     df_curvefitting
145.
146.     # In[19]:
147.
148.     df_curvefitting['RUL'] = RUL.reshape(-1,1)
149.
150.     # In[20]:
151.     data_pth_test = os.path.join(dirname, 'test','test_FD001.txt')
152.     data_pth_rul = os.path.join(dirname, 'rul','RUL_FD001.txt')
153.     # column names for the dataset
154.     # op_cond refers to operational condition, sn: sensor
155.     col_name = ['engine', 'time/cycle', 'op_cond_1', 'op_cond_2', 'op_con
156.     d_3']
157.     col_name = col_name + ['sn_{}'.format(s + 1) for s in range(21)]
158.     df_test = pd.read_csv(data_pth_test, header=None, names=col_name,deli
159.     m_whitespace=True,index_col=0)
160.
161.     # In[21]:
162.
163.     df_op_t = df_test[op_column]
164.     df_RUL_TRUE= pd.read_csv(data_pth_rul,header=None)
165.     df_RUL_TRUE.columns= ['RUL']
166.
167.     # In[22]:
168.
169.     for id_t in df_test.index.unique():

```



```

168.         df_test.loc[id_t, 'RUL'] = df_test.loc[id_t]['time/cycle'].apply(1
ambda x: x-df_test.loc[id_t]['time/cycle']).min())
169.
170.     # In[23]:
171.
172.     engines_t=df_test.index.unique().values # engine numbers
173.     engine_slices_t = dict()# key is engine number, value is a slice that
gives numpy index for the data that pertains to an engine
174.     for i_t,engine_num_t in enumerate(engines_t):
175.         row_name_t=df_test.loc[engine_num_t].iloc[-1].name
176.         row_sl_t=df_test.index.get_loc(row_name_t) # row slice to get num
py index
177.         engine_slices_t[engine_num_t]=row_sl_t
178.
179.     # In[24]:
180.
181.     RUL_t = np.empty(len(engines_t))
182.     for i_t,engine_num_t in enumerate(engines_t):
183.         RUL_t[i_t]=df_test.loc[engine_num_t]['RUL'].max()
184.
185.     # In[25]:
186.
187.     engines_t=df_test.index.unique().values # engine numbers
188.     engine_slices_t = dict()
189.     for i_t,engine_num_t in enumerate(engines_t):
190.         row_name_t=df_test.loc[engine_num_t].iloc[-1].name
191.         row_sl_t=df_test.index.get_loc(row_name_t) # row slice to get num
py index
192.         engine_slices_t[engine_num_t]=row_sl_t
193.
194.     # In[105]:
195.
196.     engine_num_t=20
197.     RUL_engine_t = df_test.loc[engine_num_t]['RUL'].values
198.     engine_sensors_t= df_op_t.values[engine_slices_t[engine_num_t],:]
199.
200.     # In[106]:
201.
202.     y1_engine_t = engine_sensors_t# sn_11
203.     x1_engine_t = df_test.loc[engine_num_t]['RUL']
204.     #df_op.loc[engine_num]['op_cond_2']
205.     #plt.subplots(figsize=(19,10))
206.
207.     plt.plot(x1_engine_t[0:150],y1_engine_t[0:150],label='x1')
208.
209.
210.     plt.xlabel('lifetime [cycles]')
211.     plt.ylabel('Signal value')
212.     plt.show();
213.
214.     # In[107]:
215.
216.     result_t = seasonal_decompose(engine_sensors_t[:,0], model='multiplic
ative', freq=10)#
217.     trend_t= result_t.trend
218.     plt.plot(trend_t)
219.     pyplot.show()
220.
221.     # In[108]:
222.
223.     HI_x_t= np.delete(RUL_engine_t,np.argwhere(np.isnan(trend_t)))
224.     transfer_y_t = trend_t[~np.isnan(trend_t)]
225.     HI_y_t= transfer_y_t-np.min(transfer_y_t)

```

```

226. #transfer_y= np.nan_to_num(trend)
227. #HI_linear.fit(HI_x,HI_y)
228. #print('Coefficients: \n', HI_linear.coef_)
229. plt.plot(HI_x_t,HI_y_t)
230.
231. # In[109]:
232.
233. pd.DataFrame(data=HI_y_t)
234.
235. # In[110]:
236.
237. from sklearn.neighbors import KNeighborsRegressor
238. from sklearn.neighbors import KNeighborsClassifier
239. from sklearn.model_selection import train_test_split
240. #import required packages
241. from sklearn import neighbors
242. from sklearn.metrics import mean_squared_error
243. from math import sqrt
244. import matplotlib.pyplot as plt
245. get_ipython().run_line_magic('matplotlib', 'inline')
246. train , test = train_test_split(df_curvefitting, test_size = 0.3)
247. x_train = train.drop('RUL', axis=1)
248. y_train = train['RUL']
249.
250. x_test = test.drop('RUL', axis = 1)
251. y_test = test['RUL']
252. np.random.seed()
253. train , test = train_test_split(df_curvefitting, test_size = 0.3)
254.
255. # In[111]:
256.
257. model = neighbors.KNeighborsRegressor(n_neighbors = 3, weights='distance',p=2)
258. model.fit(x_train, y_train) #fit the model
259. pred=model.predict(x_test) #make prediction on test set
260. error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
261.
262. # In[112]:
263.
264. plt.plot(y_test.values,label='True value')
265. plt.plot(pred,label='estimation')
266. plt.xlabel('unit', fontsize = 15)
267. plt.ylabel('Lifetime', fontsize = 15)
268. plt.legend()
269. plt.show()
270.
271. # In[113]:
272.
273. fig, ax = plt.subplots()
274. ax.scatter(pred,y_test.values, edgecolors=(0, 0, 0))
275. ax.plot([y_test.min(),y_test.max()], [y_test.min(), y_test.max()], 'k
--', lw = 2)
276. ax.set_xlabel('Predicted RUL', fontsize = 18)
277. ax.set_ylabel('Actual RUL', fontsize = 18)
278. ax.tick_params(axis='both', which='major', labelsize = 14)
279. ax.set_title( 'Predicted Lifetime vs Actual Lifetime', fontsize = 20)
280.
281. plt.show()
282. # In[114]:
283.
284. print(np.mean(y_test-pred))
285. print(np.std(y_test-pred))

```

```

286.
287.     # In[115]:
288.
289.     from scipy.stats import norm
290.     sns.distplot(y_test-pred,fit=norm,kde=False)
291.     plt.title('The distribution of errors')
292.
293.     # In[116]:
294.
295.     timelimit=150
296.
297.     # In[117]:
298.
299.     neigh = neighbors.KNeighborsRegressor(n_neighbors = 3,weights='distan
300. ce',algorithm='auto')
301.     #neigh.fit(df_curvefitting.iloc[:,0:len(HI_y_t)],np.ravel(RUL))
302.     neigh.fit(df_curvefitting.iloc[:,0:timelimit],np.ravel(RUL).reshape(-
303. 1,1))
304.     pred_t=neigh.predict(HI_y_t.reshape(1,-1)[:0:timelimit])
305.     print(pred_t)
306.
307.     # In[118]:
308.
309.     lifetime=[]
310.     for t in range (1,timelimit):
311.         neigh.fit(df_curvefitting.iloc[:,0:t],np.ravel(RUL).reshape(-
312. 1,1))
313.         pred_t=neigh.predict(HI_y_t.reshape(1,-1)[:0:t])
314.         lifetime.append(pred_t.item())
315.     lifetime_new = np.array(lifetime)[-51:-1]
316.
317.     # In[119]:
318.
319.     lifetime_new
320.
321.     # In[120]:
322.
323.     pred_t.item()
324.
325.     # In[121]:
326.
327.     #pred_t=neigh.predict(HI_y_t.reshape(1,-1))
328.
329.     # In[122]:
330.
331.     from scipy.stats import norm
332.     sns.distplot(lifetime_new, kde=True,
333.                 bins=int(6), color = 'blue',
334.                 hist_kws={'edgecolor':'black'},
335.                 )#kde_kws={'linewidth': 2}
336.
337.     # In[123]:
338.
339.     pred_t
340.
341.     # In[124]:
342.
343.     print(np.array(lifetime_new).mean())
344.     print(np.array(lifetime_new).std())
345.

```

```

346.     # In[125]:
347.
348.     np.array(lifetime_new).mean()-len(HI_y_t)
349.
350.     # In[126]:
351.
352.     np.array(lifetime_new).mean()
353.
354.     # In[127]:
355.
356.     HI_vector = []
357.     #Initial_vector = []
358.     for engine_num in engines:#
359.         engine_sensors_o= df_op.values[engine_slices[engine_num],:]
360.         x_tran= engine_sensors_o[:,0]-
np.min(engine_sensors_o[:,0])+0.000001#
361.         RUL_engine = df_train.loc[engine_num]['RUL'].values
362.         result = seasonal_decompose(x_tran, model='multiplicative', freq=
10)#
363.         trend= result.trend
364.         HI_x= np.delete(RUL_engine,np.argmax(np.isnan(trend)))
365.         transfer_y = trend[~np.isnan(trend)]
366.         HI_y= transfer_y-np.min(transfer_y)
367.         HI_vector.append(np.max(HI_y))
368.         #Initial_vector.append(HI_y[0])
369.         #plt.scatter(HI_x,HI_y,label='raw'
370.
371.         #plt.axhline(0.15, color='r', linestyle='-')
372.
373.     plt.show()
374.
375.     # In[128]:
376.
377.     final_pd=pd.DataFrame(data=HI_vector)
378.     final_pd[1]=np.array(RUL).reshape(-1,1)
379.
380.     # In[129]:
381.
382.     from scipy import spatial
383.     HIpoint=[]
384.
385.     A=np.array(final_pd[1]).reshape(-1, 1)
386.     for t in range (1,51):
387.         neigh.fit(df_curvefitting.iloc[:,0:t],np.ravel(RUL).reshape(-
1,1))
388.         pred_t=neigh.predict(HI_y_t.reshape(1,-1)[:,0:t])
389.         pt=[pred_t.item()]
390.         A[spatial.KDTree(A).query(pt)[1]]
391.         distance,index = spatial.KDTree(A).query(pt)
392.         HIpoint.append(final_pd.iloc[index][0])
393.
394.
395.     # In[130]:
396.
397.     for engine_num in engines:
398.         engine_sensors_o= df_op.values[engine_slices[engine_num],:]
399.         x_tran= engine_sensors_o[:,0]-
np.min(engine_sensors_o[:,0])+0.000001
400.         RUL_engine = df_train.loc[engine_num]['RUL'].values
401.         result = seasonal_decompose(x_tran, model='multiplicative', freq=
10)#
402.         trend= result.trend
403.         HI_x= np.delete(RUL_engine,np.argmax(np.isnan(trend)))

```

```

404.         transfer_y = trend[~np.isnan(trend)]
405.         HI_y= transfer_y-np.min(transfer_y)
406.         plt.scatter(HI_x,HI_y,s=1,c='lightblue')
407.         #plt.plot(HI_x,HI_y,label='exp_fitted')
408.         #plt.axhline(y=1.0, linestyle='-',lw=2)
409.         #plt.title('Health Index (HI)')
410.         #plt.xlabel('RUL [cycles]')
411.         #plt.ylabel('HI [-]')
412.
413.         plt.plot(HI_x_t[0:timelimit],HI_y_t[0:timelimit],c='r')
414.         plt.plot(lifetime_new,HIpoint,'*',c='b')
415.         #plt.plot(lifetime)
416.         plt.show()
417.
418.         # with open("C:\\Users\\wjh62\\Desktop\\data.txt","w") as f:
419.         #         np.savetxt(f,final_pd, delimiter=",")
420.
421.         # with open("C:\\Users\\wjh62\\Desktop\\data.txt","w") as d:
422.         #         np.savetxt(d,lifetime, delimiter=",")
423.
424.         # In[52]:
425.
426.         pd_prediction = []
427.         pd_prediction_mid=[]
428.
429.         for engine_num_t in engines_t:
430.             engine_sensors_t= df_op.values[engine_slices_t[engine_num_t],:]
431.             x_tran_t= engine_sensors_t[:,0]-
np.min(engine_sensors_t[:,0])+0.000001
432.             result_t = seasonal_decompose(x_tran_t, model='multiplicative', f
req=10)#
433.             trend_t= result_t.trend
434.             RUL_engine_t= df_test.loc[engine_num_t]['RUL'].values
435.             HI_x_t= np.delete(RUL_engine_t,np.argwhere(np.isnan(trend_t)))
436.             transfer_y_t = trend_t[~np.isnan(trend_t)]
437.             HI_y_t= transfer_y_t-np.min(transfer_y_t)
438.             neigh.fit(df_curvefitting.iloc[:,0:len(HI_y_t)],np.ravel(RUL))
439.             pred_t=np.int(neigh.predict(HI_y_t.reshape(1,-1)))
440.             #pd_prediction_mid = pd.DataFrame(nt(pred_t))
441.             pd_prediction.append(int(pred_t))
442.             pd_prediction_mid.append(pred_t-len(HI_y_t))#
443.
444.
445.         # In[53]:
446.
447.         plt.plot(df_RUL_TRUE, color = 'red', label = 'Real data')
448.         plt.plot(pd_prediction_mid, color = 'blue', label = 'Predicted data')
449.         #*400
450.         plt.title('Prediction')
451.         plt.legend()
452.         plt.show()
453.         # In[54]:
454.
455.
456.         neigh = neighbors.KNeighborsRegressor(n_neighbors = 3,weights='distan
ce',algorithm='auto')
457.
458.         # In[55]:
459.
460.         sns.distplot((pd_prediction_mid-
df_RUL_TRUE['RUL']), fit=norm, kde=False)
461.

```

```

462.     # In[56]:
463.
464.     fig, ax = plt.subplots()
465.     ax.scatter(pd_prediction_mid,df_RUL_TRUE, edgecolors=(0, 0, 0))
466.     ax.plot([df_RUL_TRUE.min(),df_RUL_TRUE.max()], [df_RUL_TRUE.min(), df
_RUL_TRUE.max()], 'k--', lw = 2)
467.     ax.set_xlabel('Predicted RUL', fontsize = 18)
468.     ax.set_ylabel('Actual RUL', fontsize = 18)
469.     ax.tick_params(axis='both', which='major', labelsize = 14)
470.     ax.set_title( 'Predicted RUL vs Actual RUL', fontsize = 20)
471.     plt.show()
472.
473.     # In[57]:
474.
475.     from sklearn.metrics import r2_score, mean_squared_error
476.     rmse_test = np.sqrt(mean_squared_error(pd_prediction_mid,df_RUL_TRUE)
)
477.     r2_test = r2_score(pd_prediction_mid,df_RUL_TRUE)
478.     print("The model performance for the test set")
479.     print("-----")
480.     print("RMSE of test set is {}".format(rmse_test))
481.     print("R2 score of test set is {}".format(r2_test))

```

C.4 Neural network model (Including offline reference model and online prognostics model)

```

1. # coding: utf-8
2.
3. # In[1]:
4.
5.
6. # package import
7.
8. import pandas as pd
9. import os
10. import numpy as np
11. import matplotlib.pyplot as plt
12. import seaborn as sns
13.
14. import importlib
15. from sklearn.linear_model import LinearRegression
16. from sklearn.preprocessing import StandardScaler
17. from sklearn.decomposition import PCA
18. # from sklearn.feature_selection import f_regression
19. from sklearn.linear_model import LogisticRegression
20. from scipy.signal import savgol_filter
21. from scipy.optimize import curve_fit
22. from sklearn.model_selection import TimeSeriesSplit
23. from sklearn.metrics.pairwise import euclidean_distances
24. from statsmodels.tsa.holtwinters import ExponentialSmoothing , HoltWintersRe
sults
25. from sklearn import preprocessing
26.
27. get_ipython().run_line_magic('matplotlib', 'inline')
28.
29.
30. # In[2]:

```

```

31.
32. dirname = os.getcwd()
33. data_pth_train = os.path.join(dirname, 'training', 'train_FD001.txt')
34. # column names for the dataset
35. # op_cond refers to operational condition, sn: sensor
36. col_name = ['engine', 'time/cycle', 'op_cond_1', 'op_cond_2', 'op_cond_3']
37. col_name = col_name + ['sn_{}'.format(s + 1) for s in range(21)]
38. df_train = pd.read_csv(data_pth_train, header=None, names=col_name, delim_whitespace=True, index_col=0)
39.
40. # In[3]:
41.
42. data_pth_test = os.path.join(dirname, 'test', 'test_FD001.txt')
43. data_pth_rul = os.path.join(dirname, 'rul', 'RUL_FD001.txt')
44. col_name = ['engine', 'time/cycle', 'op_cond_1', 'op_cond_2', 'op_cond_3']
45. col_name = col_name + ['sn_{}'.format(s + 1) for s in range(21)]
46. df_test = pd.read_csv(data_pth_test, header=None, names=col_name, delim_whitespace=True, index_col=0)
47. TrueRUL= pd.read_csv(data_pth_rul, sep = "\n", header = None)
48.
49. # In[4]:
50.
51. variable_remove=[ col for col in df_train.columns if (df_train[col].std() <=
    .0001*df_train[col].mean()) & (df_train[col].nunique() <=4) ]
52.
53. print('columns to be removed from analysis since they do not change with time \n', variable_remove)
54.
55. # In[5]:
56.
57. df_train.drop(columns=variable_remove, axis=1, inplace=True)
58.
59. # In[6]:
60.
61. df_test.drop(columns=variable_remove, axis=1, inplace=True)
62.
63. # In[7]:
64.
65. df_test.drop(['op_cond_1', 'op_cond_2', 'sn_9', 'sn_14'], axis=1, inplace=True)
66.
67. #df_test.drop(['op_cond_1', 'op_cond_2'], axis=1, inplace=True)
68. df_test.shape
69.
70. # In[8]:
71.
72. df_train.drop(['op_cond_1', 'op_cond_2', 'sn_9', 'sn_14'], axis=1, inplace=True)
73. #df_train.drop(['op_cond_1', 'op_cond_2'], axis=1, inplace=True)
74. df_train.shape
75.
76. # In[9]:
77.
78. for id in df_train.index.unique():
79.     df_train.loc[id, 'RUL'] = df_train.loc[id]['time/cycle'].apply(lambda x:
    df_train.loc[id]['time/cycle'].max()-x)
80. # positive RUL for each engine
81.
82. for id_t in df_test.index.unique():
83.     df_test.loc[id_t, 'RUL'] = df_test.loc[id_t]['time/cycle'].apply(lambda x
    : x-df_test.loc[id_t]['time/cycle'].min()+1)
84.
85. # In[10]:
86.

```

```

87. # get all sensors
88. raw_columns = df_train.columns.values[1:-1]
89. raw_sensors = df_train[raw_columns].values # as numpy array
90. raw_columns
91.
92. raw_columns_t = df_test.columns.values[1:-1]
93. raw_sensors_t = df_test[raw_columns_t].values # as numpy array
94. raw_columns_t
95.
96. # In[11]:
97.
98. engines=df_train.index.unique().values # engine numbers
99. engine_slices = dict()# key is engine number, value is a slice that gives nu
    mpy index for the data that pertains to an engine
100.
101.     for i,engine_num in enumerate(engines):
102.         row_name=df_train.loc[engine_num].iloc[-1].name
103.         row_sl=df_train.index.get_loc(row_name) # row slice to get numpy
    index
104.         engine_slices[engine_num]=row_sl
105.
106.
107.     engines_t=df_test.index.unique().values # engine numbers
108.     engine_slices_t = dict()# key is engine number, value is a slice that
    gives numpy index for the data that pertains to an engine
109.
110.     for i_t,engine_num_t in enumerate(engines_t):
111.         row_name_t=df_test.loc[engine_num_t].iloc[-1].name
112.         row_sl_t=df_test.index.get_loc(row_name_t) # row slice to get num
    py index
113.         engine_slices_t[engine_num_t]=row_sl_t
114.
115.     # In[12]:
116.
117.     # create RUL vector
118.     RUL = np.empty(len(engines))
119.
120.     for i,engine_num in enumerate(engines):
121.         RUL[i]=df_train.loc[engine_num]['RUL'].max()
122.
123.
124.     RUL_t = np.empty(len(engines_t))
125.
126.     for i_t,engine_num_t in enumerate(engines_t):
127.         RUL_t[i_t]=df_test.loc[engine_num_t]['RUL'].max()
128.
129.     # In[13]:
130.
131.
132.     # normalization
133.     engine_sensors=(raw_sensors-
    raw_sensors.min(axis=0))/(raw_sensors.max(axis=0)-
    raw_sensors.min
134.     (axis=0))
135.
136.
137.     # In[14]:
138.
139.     cycle_RUL= (df_train['RUL'].values)
140.     cycle_RUL_t= (df_test['RUL'].values)
141.
142.     # In[15]:
143.

```



```

144.     x_train= engine_sensors
145.     y_train = cycle_RUL.reshape(-1,1)
146.
147.     # In[164]:
148.
149.     from keras.layers import Dense, Dropout, LSTM, Activation
150.     from keras.models import Sequential
151.     from tensorflow.keras import layers
152.     from tensorflow.keras import activations
153.
154.     # Initialising the ANN
155.     ann = Sequential()
156.
157.     # Adding the input layer and the first hidden layer
158.     ann.add(Dense(14, activation=activations.sigmoid, input_dim = 12))
159.     #ann.add(Dropout(0.3))
160.     #ann.add(LSTM(input_shape=(100, 12),units=100,return_sequences=True))
161.
162.     # Adding the second hidden layer
163.     ann.add(Dense(units = 400, activation=activations.softplus)) #softsig
164.     #ann.add(Dropout(0.3))
165.     # Adding the third hidden layer
166.     ann.add(Dense(units = 128, activation=activations. sigmoid )) #
167.     # Adding the third hidden layer
168.     # Adding the output layer
169.     ann.add(Dense(units = 1))
170.
171.     #model.add(Dense(1))
172.
173.
174.     # In[165]:
175.     ann.summary()
176.
177.
178.     # In[166]:
179.
180.     import keras
181.
182.     # In[167]:
183.
184.     # Compiling the ANN
185.     ann.compile(optimizer = keras.optimizers.Adam(learning_rate=0.0015),
186.     loss = 'mean_squared_error', metrics=['mae'])#adamax
187.
188.     history=ann.fit(x_train, y_train,validation_split=0.3, epochs=100
189.     ,verbose=2 )#batch_size = 1,
190.
191.     #compling the Artificial neural network
192.     #classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy',
193.     metrics = ['accuracy'])
194.
195.     #fitting the training setvalidation_split=0.17,
196.     #classifier.fit(x_train, y_train, batch_size = 10, nb_epoch = 100)
197.
198.     # In[168]:
199.
200.     #plt.axis([0, 100, 0, 0.1])
201.     plt.plot(history.history['loss'])
202.     plt.plot(history.history['val_loss'])
203.     plt.title('model loss')
204.     plt.ylabel('loss')

```

```

203.     plt.xlabel('Number of Epoch')
204.     plt.legend(['Training', 'Validation'], loc='upper right')
205.
206.     plt.show()
207.
208.     # In[169]:
209.
210.     get_ipython().system('pip3 install ann_visualizer')
211.     get_ipython().system('pip install graphviz')
212.     from ann_visualizer.visualize import ann_viz
213.     from graphviz import Source
214.
215.     ann_viz(ann, view=True, title="Neural network plot")
216.     Source.from_file('network.gv')
217.
218.     # In[170]:
219.
220.
221.     #normalization
222.     engine_sensors_t=(raw_sensors_t-
223.     raw_sensors_t.min(axis=0))/(raw_sensors_t.max(axis=0)-
224.     raw_sensors_t.m
225.     in(axis=0))
226.
227.     # In[171]:
228.
229.     engine_num_t = 31
230.
231.     x_predict_31=engine_sensors_t[engine_slices_t[engine_num_t],:]
232.
233.     # In[172]:
234.
235.     x_predict_31
236.
237.     # In[173]:
238.
239.     prediction_ann = ann.predict(x_predict_31)
240.
241.     # In[174]:
242.
243.     prediction_ann
244.
245.     # In[175]:
246.
247.     plt.plot(prediction_ann)
248.
249.     # In[176]:
250.
251.     df_e=pd.DataFrame(data=df_test.index.values,columns=['machine'])
252.     df_r=pd.DataFrame(data= engine_sensors_t)
253.     predict_sensor=df_r.join(df_e)
254.     predict_sensor=predict_sensor.set_index(['machine'])
255.
256.     # In[177]:
257.
258.     df_pre = pd.DataFrame()
259.     for j in engines_t:
260.         df_mid = predict_sensor.loc[j].iloc[-1:]
261.         df_pre = df_pre.append(df_mid)
262.         #return df_mid
263.
264.     # In[178]:

```

```

264.
265.     Truepre_t=ann.predict(df_pre)
266.
267.     # In[179]:
268.
269.     plt.plot(TrueRUL, color = 'red', label = 'Real data')
270.     plt.plot(Truepre_t, color = 'blue', label = 'Predicted data')#*400
271.     plt.title('Prediction')
272.     plt.legend()
273.     plt.show()
274.
275.     # In[185]:
276.
277.     fig, ax = plt.subplots()
278.     ax.scatter(TrueRUL,Truepre_t, edgecolors=(0, 0, 0))
279.     ax.plot([TrueRUL.min(), TrueRUL.max()], [TrueRUL.min(), TrueRUL.max()
280. ], 'k--', lw = 2)
281.     ax.set_xlabel('Predicted RUL', fontsize = 18)
282.     ax.set_ylabel('Actual RUL', fontsize = 18)
283.     ax.tick_params(axis='both', which='major', labelsize = 14)
284.     ax.set_title( 'Neural Network Predicted RUL vs Actual RUL', fontsize
285. = 20)
286.     plt.show()
287.
288.     # In[181]:
289.     error = TrueRUL-Truepre_t
290.     ax=sns.distplot(error,bins=8)
291.
292.     # In[182]:
293.     from sklearn.metrics import r2_score, mean_squared_error
294.
295.     from sklearn.model_selection import train_test_split
296.
297.     rmse_test = np.sqrt(mean_squared_error(TrueRUL, Truepre_t))
298.     r2_test = r2_score(TrueRUL, Truepre_t)
299.
300.
301.     print("The model performance for the test set")
302.     print("-----")
303.     print("RMSE of test set is {}".format(rmse_test))
304.     print("R2 score of test set is {}".format(r2_test))
305.
306.     # In[183]:
307.
308.     def score(estimated_RUL,true_RUL):
309.
310.         error = np.array(estimated_RUL)-np.array(true_RUL)
311.         error = np.array(error)
312.         S1 = np.heaviside(error,0)* (np.exp(error/10)-
313. 1) # positive errors
314.         S2 = np.heaviside(-1*error,0)* (np.exp(-1*error/13)-
315. 1) # negative errors
316.         S = S1 + S2
317.
318.         return S.mean()
319.
320.     # In[184]:
321.     score(Truepre_t,TrueRUL)

```

C.5 GBM algorithm model (Including offline reference model and online prognostics model)

```
1. # coding: utf-8
2.
3. # In[4]:
4.
5.
6. # package import
7.
8. import pandas as pd
9. import os
10. import numpy as np
11. import matplotlib.pyplot as plt
12. import seaborn as sns
13.
14. import importlib
15. from sklearn.linear_model import LinearRegression
16. from sklearn.preprocessing import StandardScaler
17. from sklearn.decomposition import PCA
18. # from sklearn.feature_selection import f_regression
19. from sklearn.linear_model import LogisticRegression
20. from scipy.signal import savgol_filter
21. from scipy.optimize import curve_fit
22. from sklearn.model_selection import TimeSeriesSplit
23. from sklearn.metrics.pairwise import euclidean_distances
24. from statsmodels.tsa.holtwinters import ExponentialSmoothing, HoltWintersResults
25. from sklearn import preprocessing
26.
27. get_ipython().run_line_magic('matplotlib', 'inline')
28.
29.
30. # import math
31. # x = np.arange(1,200)
32. # def y(t):
33. #     y= 0.27796*np.exp(0.01601*t-1.83488)
34. #     return y
35. # plt.plot(x,y(x))
36. # plt.axhline(0.15, color='r', linestyle='--',xmin=0.4, xmax=0.6)
37. # plt.axvline(123, color='r', linestyle='--',ymin=0.13, ymax=0.3)
38.
39. # In[114]:
40.
41.
42. dirname = os.getcwd()
43. data_pth_test = os.path.join(dirname, 'test', 'test_FD001.txt')
44. data_pth_rul = os.path.join(dirname, 'rul', 'RUL_FD001.txt')
45. # column names for the dataset
46. # op_cond refers to operational condition, sn: sensor
47. col_name = ['engine', 'time/cycle', 'op_cond_1', 'op_cond_2', 'op_cond_3']
48. col_name = col_name + ['sn_{}'.format(s + 1) for s in range(21)]
49. df_test = pd.read_csv(data_pth_test, header=None, names=col_name,delim_whitespace=True,index_col=0)
50. TrueRUL= pd.read_csv(data_pth_rul, sep = "\n", header = None)
51.
52.
53. # In[115]:
```

```

54.
55.
56. TrueRUL.columns= ['RUL']
57. TrueRUL.head()
58.
59.
60. # In[6]:
61.
62.
63. op_column = ['sn_11']
64.
65. df_op_t = df_test[op_column]
66.
67.
68. # In[7]:
69.
70.
71. for id_t in df_test.index.unique():
72.     df_test.loc[id_t,'RUL'] = df_test.loc[id_t]['time/cycle'].apply(lambda x
    : x-df_test.loc[id_t]['time/cycle'].min())
73.
74.
75. # In[8]:
76.
77.
78. engines_t=df_test.index.unique().values # engine numbers
79. engine_slices_t = dict()
80. for i_t,engine_num_t in enumerate(engines_t):
81.     row_name_t=df_test.loc[engine_num_t].iloc[-1].name
82.     row_sl_t=df_test.index.get_loc(row_name_t) # row slice to get numpy inde
    x
83.     engine_slices_t[engine_num_t]=row_sl_t
84.
85.
86. # In[9]:
87.
88.
89. df_mid = pd.DataFrame()
90. df_final = pd.DataFrame()
91.
92.
93. # In[10]:
94.
95.
96. from statsmodels.tsa.seasonal import seasonal_decompose
97. from matplotlib import pyplot
98. engine_num_t=82
99. RUL_engine_t = df_test.loc[engine_num_t]['RUL'].values
100.     engine_sensors_t= df_op_t.values[engine_slices_t[engine_num_t],:]
101.
102.
103.     # In[11]:
104.
105.
106.     y1_engine_t = engine_sensors_t# sn_11
107.
108.     x1_engine_t = df_test.loc[engine_num_t]['RUL']
109.     #df_op.loc[engine_num]['op_cond_2']
110.     #plt.subplots(figsize=(19,10))
111.
112.     plt.plot(x1_engine_t[0:150],y1_engine_t[0:150],label='x1')
113.
114.

```

```

115.     plt.xlabel('lifetime [cycles]')
116.     plt.ylabel('Signal value')
117.     plt.show();
118.
119.
120.     # In[12]:
121.
122.
123.     result_t = seasonal_decompose(engine_sensors_t, model='multiplicative
', freq=10)#
124.     trend_t= result_t.trend
125.     plt.plot(trend_t)
126.     pyplot.show()
127.
128.
129.     # In[13]:
130.
131.
132.     HI_x_t= np.delete(RUL_engine_t,np.argwhere(np.isnan(trend_t)))
133.     transfer_y_t = trend_t[~np.isnan(trend_t)]
134.     HI_y_t= transfer_y_t-np.min(transfer_y_t)
135.     #transfer_y= np.nan_to_num(trend)
136.     #HI_linear.fit(HI_x,HI_y)
137.     plt.plot(HI_x_t,HI_y_t)
138.
139.
140.     # In[14]:
141.
142.
143.     df_mid = pd.DataFrame(data=HI_y_t.reshape(1, -1))
144.     df_final=df_final.append(df_mid)
145.
146.
147.     # In[15]:
148.
149.
150.     HI_y_t[49]
151.
152.
153.     # In[16]:
154.
155.
156.
157.     def GBM(Y0, M,mu,sigma):
158.         dt = 1
159.         Y = np.zeros((I + 1), np.float64)
160.         T = np.zeros((I + 1), np.float64)
161.         Y[0] = Y0
162.         T[0] = 0.0001
163.
164.         for t in range(1, I + 1):
165.
166.             rand = np.random.normal(0, 0.159177*(Y0/0.8)+(1-
Y0/0.8)*sigma)#*0.1+0.9*0.1607)0.1*+0.9*0.0104847
167.             Y[t] = Y[t-1]+(0.032436*(Y0/0.8)+(1-Y0/0.8)*mu)*Y[t-
1]*dt+rand*Y[t-1]
168.             T[t] = t
169.
170.             if Y[t]>0.8:
171.                 break
172.
173.         return Y[Y != 0],T[T != 0]
174.

```

```

175.
176.
177.
178.     # In[17]:
179.
180.
181.     end_life=[]
182.
183.     I=300
184.     Y0 = 0.717
185.
186.
187.     mu = 0.015205587
188.
189.
190.     sigma =0.134393064
191.
192.
193.
194.     m=0
195.     while m < 1000:
196.         result = GBM(Y0,I,mu,sigma)
197.         x=result[1]
198.         y=result[0]
199.         end_life.append(x[-1])
200.         m+=1
201.         if y[-1] > 0.8:
202.             plt.plot(x,y)
203.             plt.grid(True)
204.             plt.xlabel('time steps')
205.             plt.ylabel('HI')
206.     plt.axhline(0.8, color='r', linestyle='-')
207.
208.
209.     # In[18]:
210.
211.
212.     np.mean(end_life)
213.
214.
215.     # In[19]:
216.
217.
218.     np.std(end_life)
219.
220.
221.     # In[20]:
222.
223.
224.     with open("C:\\Users\\wjh62\\Desktop\\data.txt","w") as f:
225.         np.savetxt(f,df_final, delimiter=",")
226.
227.
228.     # In[77]:
229.
230.
231.     HI_test = []
232.     HI_test_mid=[]
233.
234.     df_prediction = pd.DataFrame()
235.     df_midfitting = pd.DataFrame()
236.
237.     for engine_num_t in engines_t:

```

```

238.         engine_sensors_t= df_op_t.values[engine_slices_t[engine_num_t],:]
239.         x_tran_t= engine_sensors_t[:,0]-
np.min(engine_sensors_t[:,0])+0.000001
240.         result_t = seasonal_decompose(x_tran_t, model='multiplicative', f
req=10)#
241.         trend_t= result_t.trend
242.         RUL_engine_t= df_test.loc[engine_num_t]['RUL'].values
243.         HI_x_t= np.delete(RUL_engine_t,np.argwhere(np.isnan(trend_t)))
244.         transfer_y_t = trend_t[~np.isnan(trend_t)]
245.         HI_y_t= transfer_y_t-np.min(transfer_y_t)
246.         HI_test_mid = HI_y_t[-1]
247.         HI_test.append(HI_test_mid)
248.         df_midfitting = pd.DataFrame(data=HI_y_t.reshape(1,-1))
249.         df_prediction=df_prediction.append(df_midfitting)
250.
251.
252.
253.         # In[34]:
254.
255.
256.         with open("C:\\Users\\wjh62\\Desktop\\data.txt","w") as f:
257.             np.savetxt(f,df_prediction, delimiter=",")
258.
259.
260.         # In[87]:
261.
262.
263.         df_sigma.iloc[a].name
264.
265.
266.         # In[90]:
267.
268.
269.         end_life=[]
270.
271.         a=17
272.
273.
274.         I=300
275.         Y0 =HI_test[a]
276.
277.         mu =df_mu.iloc[a].name
278.
279.         sigma =df_sigma.iloc[a].name
280.
281.
282.         m=0
283.         while m < 2500:
284.             result = GBM(Y0,I,mu,sigma)
285.             x=result[1]
286.             y=result[0]
287.             end_life.append(x[-1])
288.             m+=1
289.             if y[-1] > 0.8:
290.                 plt.plot(x,y)
291.                 plt.grid(True)
292.                 plt.xlabel('time steps')
293.                 plt.ylabel('HI')
294.                 plt.axhline(0.8, color='r', linestyle='-')
295.                 print(np.mean(end_life))
296.                 print(np.std(end_life))
297.

```



```

298.
299.     # In[57]:
300.
301.
302.     data_path_mu = os.path.join(dirname, 'test', 'mu.txt')
303.     data_path_sigma = os.path.join(dirname, 'test', 'sigma.txt')
304.     df_mu = pd.read_csv(data_path_mu, header=None, delim_whitespace=True, index_col=0)
305.     df_sigma=pd.read_csv(data_path_sigma, header=None, delim_whitespace=True, index_col=0)
306.
307.
308.     # In[117]:
309.
310.
311.     end_life=[]
312.     end_mean=[]
313.     end_std=[]
314.     for i in range(0,100):
315.         I=300
316.         Y0 =HI_test[i]
317.
318.         mu =df_mu.iloc[i].name
319.
320.         sigma =df_sigma.iloc[i].name
321.
322.
323.         m=0
324.         while m < 2500:
325.             result = GBM(Y0,I,mu,sigma)
326.             x=result[1]
327.             y=result[0]
328.             end_life.append(x[-1])
329.             m+=1
330.             end_mean.append(np.mean(end_life))
331.             end_std.append(np.std(end_life))
332.             print(np.mean(end_life))
333.             print(np.std(end_life))
334.
335.
336.     # In[122]:
337.
338.
339.     plt.plot(TrueRUL, color = 'red', label = 'Real data')
340.     plt.plot(end_mean, color = 'blue', label = 'Mean-
Predicted data')**400
341.     plt.title('Prediction')
342.     plt.legend()
343.     plt.show()
344.
345.
346.     # In[120]:
347.
348.
349.     from sklearn.metrics import r2_score, mean_squared_error
350.     rmse_test = np.sqrt(mean_squared_error(TrueRUL, end_mean))
351.     r2_test = r2_score(TrueRUL, end_mean)
352.
353.
354.
355.     print("The model performance for the test set")
356.     print("-----")
357.     print("RMSE of test set is {}".format(rmse_test))

```

```
358.     print("R2 score of test set is {}".format(r2_test))
359.
360.
361.     # In[121]:
362.
363.
364.     plt.plot(end_std)
```