Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Application of Federated Learning in Predictive Maintenance to Predict Remaining Useful Life

Author:
Zhan Liang, Chan

Supervisors:
Dr. Pedro Baiz
Dr. Eric Topham

2nd Marker:
Dr. Thomas Lancaster

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 2021

Contents

Abstract

Data have and will continue to be at the centre of Prognostics and Health Management systems in training efficient machine learning models to predict remaining useful life of critical equipment. For example, remaining time to the next degradation in turbofan engines can be confidently predicted such that proactive maintenance can be scheduled to minimise downtime.

However, the data available to train such models - though large in quantity - are privately sensitive in nature and often exist in silos. Hence, access to these data remains a key hurdle to unlock their true potential in improving operational efficiency and reducing resource wastage.

To tackle this, the concept of federated learning was introduced in recent years to leverage the vast pool of data to train predictive models whilst preserving privacy. In this report, we show how federated learning can be practically applied using open-source packages in Python to predict the remaining useful life of turbofan engines.

Specifically, we applied two separate federated learning packages - FATE and dc_federated - and demonstrated the impact of balanced and imbalanced data sets on the performance of these models. We compared the results with baseline centralised models to show that despite the privacy preserving nature of the federated models, they are able to offer comparable predictive performance as centralised models.

Acknowledgement

I would like to thank my supervisors Dr. Pedro Baiz and Dr. Eric Topham for their continued guidance throughout the project. Their support and professional guidance have been a key contributing factor to the successful completion of this report.

I would also like to express my gratitude to the wider team at The Data Analysis Bureau (T-DAB) and the innovation sandbox, particularly Hamid Khandahari, for their time in whatever issues I have faced in the past 3 months.

Finally, I would like to thank my family and my lovely wife for being so understanding and patient with the countless grunts when my code failed in the middle of the night.

This project would not have been possible without any of them in what has been a tremendously difficult period for everyone and the world.

Chapter 1

Introduction

1.1 Motivation

1.1.1 Predictive Maintenance

The main concept of Predictive Maintenance (PdM) is the constant monitoring of key operational indicators of equipment or machinery and using data gathered to estimate its Remaining Useful Life (RUL) [1]. This allows for just-in-time maintenance which reduces unnecessary production downtime due to excessive maintenance. Inefficiencies and wastage are also minimised as machinery are maintained in an optimal condition before any significant degradation affects production quality.

To enable predictive maintenance, data are often collected throughout the manufacturing process using sensors attached to machines which communicate with a central server to transmit critical measurements such as humidity, temperature, power consumption, air pressure and air flow [2]. These sensors are part of a network of interconnected devices also known as Industrial Internet of Things (IIoT).

IIoT aim to create a web of connected smart machinery capable of recording large amount of data. These data will then be fed into a series of analytic pipeline with the ultimate goal of optimising key PdM performance indicators such as minimising wastage and increasing throughput [3] via maximising equipment uptime.

The use of IIoT for PdM is not new and have been executed with great success. Niyonambaza et al. [4] proposed a PdM structure implemented using IIoT to predict the impending failure of medical equipment in Rwandan hospitals which often practice corrective maintenance. Such maintenance regime happens after a failure has occurred which led to long patient backlogs. Their implementation was able to predict impending failure with an accuracy exceeding 90%. Zeki et al. [5] provided a further review of successful applications of PdM using machine learning techniques covering equipment from wind turbines [6], photovoltaic panels [7] to turbofan engines [8].

1.1.2 Constraints

Availability of data

Whilst these applications often utilise machine learning techniques to great success in predicting RUL, we observed that these have been carried out assuming that data are easily accessible and in sufficient volume.

In reality, there exists few small and medium-sized enterprises (SMEs) that possess the expertise and scale to monitor, extract and process big data. In the proposed PdM structure for hospitals in Rwanda [4], data used to train the machine learning model was collected from the largest referral hospital in Rwanda. Smaller hospitals with fewer equipment may not have the volume of data necessary to train the same model. The benefits of PdM would therefore be limited for these hospitals.

Heterogeneity of data

The distributed nature of the world we live in now means that data will no longer exist in a sufficiently homogeneous large pool to enable centralised model training. This was a problem faced by Google when attempting to perform Next-word predictions [9]. Users' data are always stored locally on device without a common homogeneous pool to train a viable model.

Although public mobile text data sets can be used to train a model, the author noted that there will be a mismatch in distribution between the training and population's distribution as the local training data are not IID.

Privacy

Even if data are sufficiently voluminous and highly accessible for model training, privacy remains a key hurdle and concern when sharing data. The applications of PdM discussed in Chapter 1.1.1 were always performed using a centralised model, which requires full unrestricted access to the underlying training data.

For manufacturers, this presents practical challenges as data may contain sensitive manufacturing parameters that translate into real world competitive advantage. Hospitals are also reluctant to share data on medical equipment as these could compromise patient confidentiality, attracting privacy lawsuits which outweigh the benefits of PdM.

Data privacy regulations have been tightening globally, with the introduction of China's Cybersecurity Law (CSL) [10] in 2017, Europe's General Data Protection Regulation (GDPR) [11] in 2018 and India's Personal Data Protection Bill (PDP Bill) [12] in 2019. This general trend will only continue in future as more data are collected in all aspects of our lives. Collaboration between individual data owners (individual or corporate) is therefore limited to the extent that data privacy can be preserved, without leaving the country or device on which it is stored.

1.1.3 Federated Learning

A recent development that will overcome these issues is Federated Learning, a privacy preserving machine learning technique. Introduced by Google in 2016 [13], federated optimisation or federated learning as a technique was utilised to shift from a centrally trained model to a shared model that is locally trained, such as on mobile devices, also known as a node or worker, illustrated in Figure 1.1.

Each worker trains a local model on its own local data before the locally trained model parameters are aggregated on a central server. The raw training data is never accessed or communicated beyond each worker at any point in time. This technique enhances privacy, alleviates collaborators' concerns over privacy preservation and offers the benefits of a scaled up machine learning model.

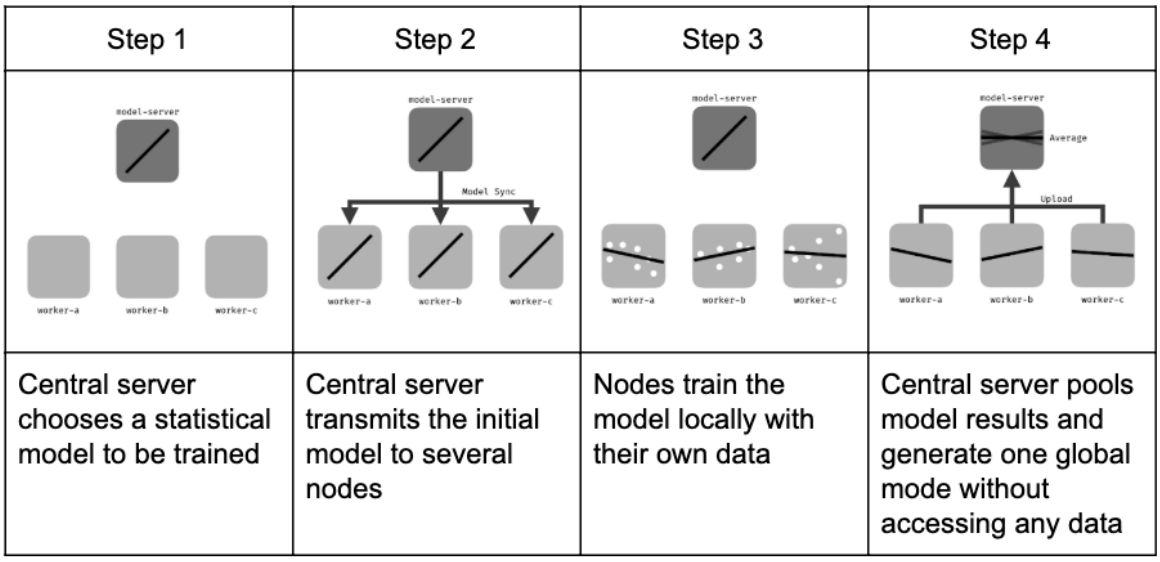

Figure 1.1: Key steps in federaeted learning [14]

The application of federated learning techniques to failure prediction in manufacturing is a relatively recent advancement in the last two years. Silveria et al. [15] applied federated learning techniques to predict springback for sheet steel materials whilst Ge et al. [16] applied similar techniques to predict failure in a Bosch production line.

It is interesting to note that both author's applications of horizontal federated learning techniques required the individually distributed data sets to have homogeneous feature space, which is also what we explored in this report. Both papers simulated a locally distributed environment by arbitrarily splitting the sampled data into different subsets, each representing a local private environment.

Finally, although federated learning techniques were overlaid on a range of machine learning models, it was used to predict the occurrence of a failure event rather than estimating RUL in the context of predictive maintenance and TTE prediction.

1.1.4 Time-to-Event Prediction

Time-to-Event (TTE) prediction models are a central component of PdM and are often used in manufacturing processes to predict the impending failure of a component, also known as the event of interest. As a branch of survival analysis, the main goal is to estimate the probability of the event occurring based on the derived event-time distribution given data observed to date.

The application of traditional survival analysis statistical methods to preempt failure is not new and its benefits are known with respect to manufacturing processes that adopts a proactive maintenance regime [17]. Under this regime, maintenance and replacement schedules can be optimised around predicted time to failure or RUL of critical components. This minimises downtime and wastage of materials by minimising the probability of defects occurring in raw materials.

The emergence of big data and advances in machine learning techniques for survival analysis further augment these traditional techniques [18]. In recent years, machine learning algorithms have been used to predict failure during electric resistance welded tube manufacturing [19] as well as tool wear prediction [20].

This report therefore aims to further the work of Silveria et al. [15] and Ge et al. [16] by applying TTE prediction models and survival analysis to predict the RUL of turbofan engines using a turbofan engine degradation simulation data set published by NASA [21] within a federated environment.

We note that Rosero et al. [22] performed a similar analysis to predict RUL via federated learning on a similar turbofan engine data set. In their work, a single federated neural network was modelled and they investigated the impact of different number of workers with a balanced data set.

We propose to extend their work by employing a different federated learning algorithm - Gradient Boosted Decision Tree - as well as investigate the impact of a balanced and imbalanced split of data between three and five workers in the federated process. We will also set up our modelling pipeline using open-source federated learning libraries to guide future real world deployment of federated learning pipeline.

1.2 Aims and Objectives

This report aims to contribute to the practical application of federated learning techniques in a PdM context by investigating the remaining time until the occurrence of a specified event, under a simulated federated environment. This report will accomplish this aim by performing the following sequentially:

1. Perform an Exploratory Data Analysis (EDA) on the NASA turbofan engine degradation simulation data set FD001.

2. Perform feature engineering to extract underlying time series trends to enhance the performance of selected machine learning models. The post-processed data set will then be split into three and five subsets of data to simulate data generated from individual workers for the purpose of applying federated learning models.

3. Apply TTE prediction models on a centralised basis, without splitting the data set. This set of centralised models is introduced in Chapter 2.1. Each model will then serve as a benchmark to compare against the selected federated learning model in the next step.

4. Apply horizontal federated learning approach on the selected models from step 2. The federated models will then be trained in a simulated local environment without actually distributing and aggregating the models remotely as in a real-world setting. The data set from step 2 will be split into balanced and imbalanced subsets to represent different number of workers.

5. Compare and evaluate the performance of the centralised and horizontal federated models with benchmark models.

6. Evaluate and highlight issues arising from potential deployment, including testing the accuracy of the trained model, processing bottlenecks, privacy concerns and communication failures.

7. Propose direction for future research.

1.3 Contributions

This report contributes to the practical application of federated learning in TTE prediction by deploying publicly available federated learning packages to a commonly studied PdM data set. The exact contributions are as follow:

1. Show that the publicly available federated learning packages FATE [23] and dc_federated [24] can be practically deployed across multiple machine learning algorithms, with minimal impact on model performance.

2. Show the impact of a balanced/imbalanced data set divided into three and five workers on model performance for both federated Gradient Boosted Decision Tree (GBDT) and Neural Network (NN).

3. Compare and contrast two established open-source federated learning packages - PySyft and FATE - with a focus on the difficulties in its practical deployment.

4. Highlight the key challenges and considerations for practitioners when deploying a federated learning framework, particularly in a production environment.

5. Provide specific directions for future research.

1.4 Outcome

This report evaluated the performance of four centralised and two federated models in the prediction of RUL for 100 engines in the NASA FD001 data set. The centralised models served as a benchmark of model performance when all data is available for training while each federated model is trained separately on a balanced and imbalanced split of FD001.

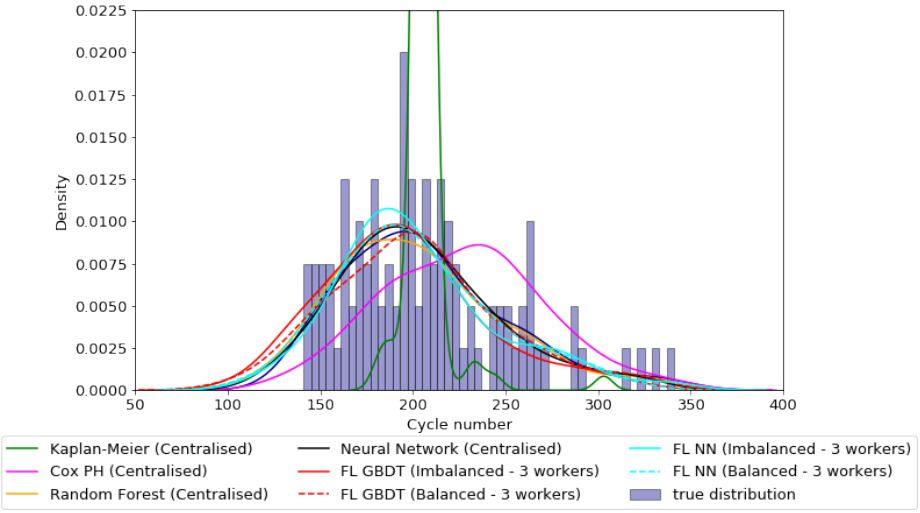

Figure 1.2: Distribution of failure cycles in each of the models

Figure 1.2 shows a summary of the distribution of model prediction across the 100 engines in the test set while Table 6.1 shows the test RMSE obtained by each of these models.

Kaplan-Meier (Centralised) Prediction based on the traditional Kaplan-Meier survival analysis by estimating a survival curve before deriving the mean survival time for the entire population based on pre-processed training data.

Cox PH (Centralised) Prediction based on the traditional Cox Proportional Hazards survival model by deriving the relationship between RUL and log-partial hazard based on pre-processed training data.

Random Forest (Centralised) Prediction based on the random forest machine learning model by performing a regression analysis on RUL using post processed training data.

Neural Network (Centralised) Prediction based on a neural network machine learning architecture with 3 hidden layers by performing a regression analysis on RUL using post processed training data.

FL GBDT Prediction based on a Gradient Boosted Decision Tree model by performing a regression analysis on RUL in a federated environment using post processed training

data. This model is trained across three and five individual workers using either a balanced or imbalanced split of the training data.

FL NN Prediction based on a neural network model by performing a regression analysis on RUL in a federated environment using post processed training data. This model is trained across three and five individual workers using either a balanced or imbalanced split of the training data.

The eventual finding from our experiment is that the federated models performed in line with centralised machine learning models despite segregation of training data into either balance or imbalanced splits. However, we note that the federated GBDT model performed better on a balanced data set than the federated NN model on both three and five workers in the federated environment.

In addition, traditional survival analysis techniques performed poorly on a centralised basis and was not well supported nor justified in implementing an equivalent federated learning model.

1.5 Outline

The following summarises the outline of the remainder of this report:

Chapter 2: Background This chapter provides the necessary background to survival analysis and federated learning. It introduces key statistical and machine learning algorithm to tackle the problem of predicting RUL and a summary of federated learning algorithms.

Chapter 3: Data In this chapter, we present an exploratory data analysis of the FD001 data set used throughout this report. It includes the background to the data set, why it was chosen and provides an intuitive technical understanding of each feature. More importantly, it elaborates on the feature engineering steps taken which is critical to model performance.

Chapter 4: Experimental Setup This chapter presents the setup on which the selected centralised and federated learning models are trained. Specifically, it introduces multiple federated learning frameworks and provides readers with an understanding of the key components of open source federated learning packages. This chapter ends with an in-depth elaboration of the FATE federated framework used in this report.

Chapter 5: Model Architecture Next, we introduce both the selected centralised and federated models in detail, including the choice of hyperparameters and model structure. Key hyperparameters for the federated learning framework are briefly touched upon.

Chapter 6: Results and Evaluation In this chapter, we present the results of both centralised and federated model results based on selected performance metrics. This set of results is compared against benchmark models.

Chapter 7: Conclusions and Future Work Finally, we conclude this report by summarising our findings and provide directions for future research.

1.6 Professional and Ethical Considerations

This project relies primarily on the publicly available NASA Turbofan Engine Degradation Simulation Data Set [21]. This report and the data it relies upon does not rely on personal data contributed by either individual or corporate participants.

The data set relied upon was provided by the Prognostics CoE at NASA Ames under the Creative Commons Public Domain Dedication. Copying, modification and distribution of the data under both personal and commercial purposes without requesting for permission is also authorised under this domain.

In line with "Examination and Assessments: Academic Integrity" published by Imperial College (version 4 - section 6.3, updated September 2018), we have also provided relevant citations where applicable throughout the report. This includes citation to other research and findings relied upon, technologies used as well as online resources and code base referenced.

Chapter 2

Background

2.1 Time-to-Event Prediction

As a branch of survival analysis, the main goal of TTE prediction is to estimate the probability of the event of interest occurring based on the derived event-time distribution given data observed to date. This is often performed using traditional statistical methods according to a survey by Wang et al. [18] although recent advancements in machine learning techniques have introduced complementary methods for survival analysis. The full taxonomy of methods available in survival analysis is shown in Figure 2.1 [18].

2.1.1 Statistical Methods

Here, we will provide a primer on well established statistical methods for survival analysis. These methods can be generally categorised as non-parametric, semi-parametric and parametric methods.

2.1.1.1 Kaplan-Meier Estimator

The most commonly used non-parametric survival analysis is the Kaplan-Meier estimator or the Product-Limit estimator [25] which is used to estimate the survival function $S(t)$ based on observed data up to time t without assuming any underlying parametric distribution.

For a set of N instances, let T_1, T_2, ... , T_N be the set of independent, identically distributed variables where T_j $(j \leq N)$is the random time when the event of interest occurred. This event of interest could either be when the instance was censored or that the event has occurred. Censoring occurs when the instance is no longer observed within the time interval of the data or that the event of interest occurred either before or after this time interval. The survival function $S(t)$ can then be defined

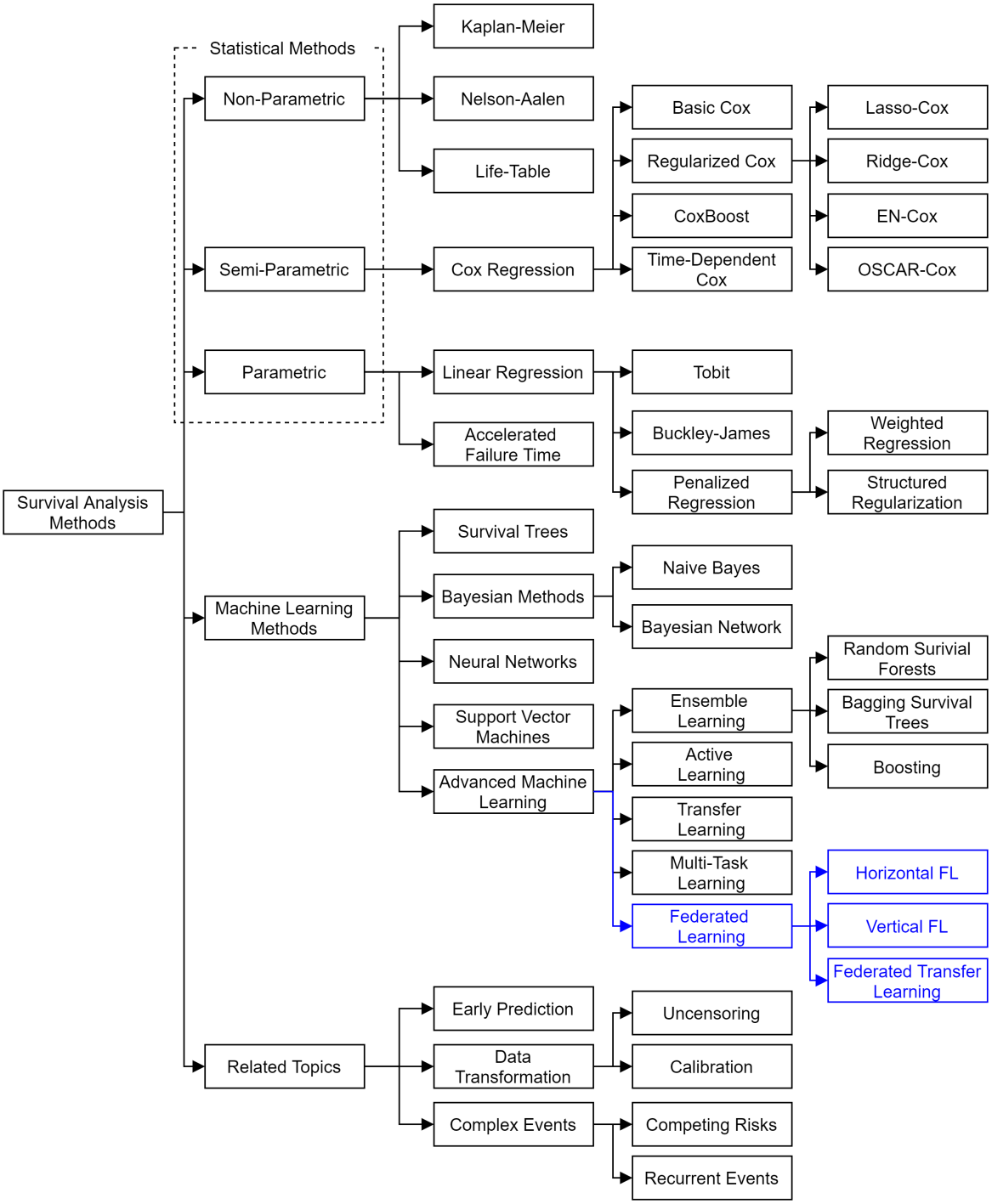

Figure 2.1: Taxonomy of methods developed for survival analysis [18], with the addition of Federated Learning

as:

$$S(t) = Prob(T > t|T > t-1)Prob(T > t-1) \tag{2.1}$$

which intuitively translates to the probability of an instance surviving beyond the current time period t, given that it has already survived up to time $t-1$. Continuing from Formula 2.1, we can obtain a recursive relationship of $S(t)$ as:

$$S(t) = p(t)S(t-1) \tag{2.2}$$

where $p(t) = 1 - Prob(T = t|T \geq t)$ which represents the proportion of instances surviving in the time interval t. Based on this recursive relationship, the Kaplan-Meier estimator at any given time t can then be derived as:

$$S(t) = \prod_{k=0}^{t} p(k) \tag{2.3}$$

Whilst the Kaplan-Meier estimator is simple to implement, it has multiple drawbacks in the context of failure or RUL prediction. The first of which is its limited application for multivariate analysis [26], where failure prediction is often interested in the correlation between multiple variables such as temperature, pressure and coolant volume and how it affects the probability of failure.

Another major pitfall is the decrease in accuracy of survival estimates as the proportion of censored data increases. This effect is compounded for a small dataset, since the survival estimate is now derived empirically from a smaller volume of data and could be easily distorted by outliers [27].

Finally, the Kaplan-Meier estimator does not lend itself well to analysing the impact of continuous variables [28] on survival rate, which is necessary when analyzing manufacturing data. Comparison of survival estimates under Kaplan-Meier is usually performed using the log rank test [27] which requires survival estimates to be placed in discrete buckets.

2.1.1.2 Cox Proportional Hazard Model

The Cox proportional hazard (Cox PH) model [29] is the most common model used for semi-parametric survival analysis [18]. The main idea supporting the Cox model is that all instances in the analysis have the same hazard function, which is defined as

$$h(t) = \lim_{\Delta t \to 0} \frac{Prob(t < T \leq t + \Delta t|T > t)}{\Delta t} = \frac{f(t)}{S(t)} \tag{2.4}$$

where $S(t)$ is defined in Formula 2.1. Since $f(t)$ is the probability of the instance T encountering the event in the period t to $t + dt$ given it has survived up to t and $S(t)$ is the probability of survival up to t, the hazard function represents the instantaneous rate at which the event occurs at time t. Note that $f(t)$ is also equal to $1 - S(t)$.

The Cox PH model can then be defined by modelling the hazard function $h(t|X = \boldsymbol{x})$ for a given set of covariates, which is represented as a vector $\boldsymbol{x}$ with β being the corresponding coefficient vector of x:

$$h(t|X = \boldsymbol{x}) = h_0(t)exp(\boldsymbol{x}^T\beta) \tag{2.5}$$

$h_0(t)$ is the baseline hazard function, that is used to compute the hazard function for all instances. The Cox PH model is a semi-parametric model because the underlying distribution of $h_0(t)$ is not required to be known to estimate β. The predicted hazard function is then the product of $h_0(t)$ and exponential function of the linear combination of the covariates and its coefficient. The effect of this linear combination stays constant throughout time. The cumulative hazard function $H(t)$ for the same covariate x can then be defined as:

$$H(t|x) = \int_0^t h(u|x)du = \int_0^t \frac{f(u|x)}{S(u|x)}du = \int_0^t \frac{1 - S(u|x)}{S(u|x)}du = -logS(t|x). \tag{2.6}$$

This further allow us to derive the conditional survival function as:

$$S(t|x) = exp(-H(t|x)) = S_0(t)^{exp(x^T\beta)} \tag{2.7}$$

where $S_0(t)$ is the baseline survival function. The estimation of the coefficient of covariates given a set of covariates can be performed by maximising the partial likelihood function which can be done trivially using statistical packages in Python.

2.1.1.3 Regression Model

Parametric models, as the name suggests, requires full parameterisation of the underlying distribution of survival data. All instances in the data is assumed to follow this distribution, which could be from a family of commonly known distributions such as normal, exponential, Weibull, logistic and log-normal. Even though knowledge of the underlying distribution allows better intepretability of results, this is also a key disadvantage as it may not always be possible to fit a good distribution to the data.

2.1.2 Supervised Machine Learning Methods

In this section, we will provide a primer on application of machine learning methods for survival analysis. Similar to statistical methods, machine learning methods can be categorised into four main types: supervised, semi-supervised, unsupervised and reinforcement learning.

However, we will focus on supervised learning where we provide labelled training data to the algorithm which will then map the combination of covariates to a discrete or continuous output. The latter is particularly relevant in our context where we aim to predict RUL which is a continuous target variable.

2.1.2.1 Random Survival Forest

The first of the two machine learning methods is the Random Survival Forest (RSF), first introduced by Ishwaran et al. [30], which leverages the Random Forest (RF) method proposed by Breiman [31].

Decision Trees

Fundamentally, both RF and RSF are extensions of decision tree, which splits an initial dataset into binary or multiple subsets of data (known as nodes) based on a splitting criteria [32]. It then recursively splits these nodes based on the same splitting criteria until the terminating condition is met. The end of the tree is then known as the leaf node.

Several metrics exist to derive the optimal splitting rule. This includes maximising the information gain prior and post split, minimising the Gini impurity [33] in each child node or variance reduction. For each categorical attribute, the algorithm would perform a split at each possible discrete partition, assess the performance of the split based on the selected metric and select the split point which maximises (or minimises) that metric. To do the same for a continuous attribute, the attribute has to be discretised beforehand either by selecting a fixed value or discretising the residuals after fitting a regression model on the data [34].

This process would continue recursively for each child node produced by the splitting rule, until a terminating condition is met. This termination condition could be met when all instances in a node belong to the same category, when there are no attributes left to split, when the maximum depth of the tree has been reached or when the minimum number of instances in the node is met [35]. In the last two cases, the maximum tree depth and minimum number of instances in a node are known as hyperparameters of the tree.

A test object can then be classified by applying the split rules sequentially to its attributes and counting the majority label of the instances in the leaf node in which the test object eventually landed in. For a continuous target variable, the output would usually be the average or median value of the instances in the leaf node of the trained tree. Finally, pruning is employed to deal with the problem of overfitting [36], where the tree conforms well to training data but performs badly on unseen test data.

Random Forest

Random Forest is then a combination or ensemble of decision trees, constructed by introducing randomisation in two ways. First, the parent node of each tree is generated through a process known as bagging [37], where new datasets are generated by randomly sampling with replacement from the original dataset. Second, an optimal split rule is chosen from a random subset of attributes, instead of selecting from the full set of attributes as in a decision tree.

As a result, multiple trees can be generated from the original dataset and classification can be performed by taking the majority vote across all the trees generated. For a continuous target variable, the output will usually be the average value [35] of the target variable across the selected leaf nodes in the ensemble.

Random Survival Forest

RSF extends RF by basing the splitting criterion on the survival function, where each node is split into the children node such that it maximises the difference in survival function between each child. Survival function here is obtained from the Kaplan-Meier estimator as defined in equation 2.1.

Hence, successive recursive splits of each node will lead to increase within-node homogeneity and eventually the leaf node will comprise of instances with similar survival.

RSF can be implemented by utilising one of four commonly used split rules - log-rank splitting rule [38], conservation-of-events splitting rule, log-rank score rule or random log-rank splitting rule. According to findings by Ishwaran et al. [30], RSF implemented using log-rank splitting rule and log-rank score rule results in the lowest prediction error.

2.1.2.2 Artificial Neural Network

Application of Artificial Neural Network (ANN) in the context of survival analysis has been well researched in various fields, such as in the medical field [39] [40] and in finance [41]. According to Wang et al. [18], there are three main methods that neural network based techniques can be adopted for survival analysis:

1. Employing a neural network to directly predict the survival time as the output given a set of inputs.

2. Employing a neural network to directly predict the survival probability as the output given the survival status of the inputs.

3. Proportional hazards neural network [42] which replaces the linear combination of the covariates and its coefficient in Formula 2.5 with the output of a non-linear neural network, maintaining the proportionality constraint in the original Cox PH model.

For the purpose of this report, we are interested in the application of neural network to survival analysis by directly predicting the RUL as the output given a set of inputs. This allows a comparable basis with the external benchmark models as well as when federated learning techniques are applied subsequently.

2.2 Federated Learning

2.2.1 Overview

The TTE prediction models introduced in Chapter 2.1 are usually executed as a centralised model where the data are assumed to originate from a single trusted source. This is often not realistic, as real-world data tend to reside in individual walled off workers with strict data privacy protection rules that prevent them from sharing data externally. In this scenario, centralised models are limited to the extent that these data are available for training.

First introduced by Google in 2016 [13], federated learning offers the advantage of access to these private data residing in individual clients without actually sharing it with external workers. McMahan et al. [43] and Cheng et al. [44] have shown that federated learning can offer comparable predictive performance versus centralised models.

The general idea underlying the federated learning framework can be summarised in four steps [44]:

1. A current global model is downloaded to the client side from the server.
2. Each client updates the global model with respect to its locally stored data. The local data is never shared with any external worker.
3. Model updates are sent back to the server by each client. This includes model parameters such as model weights for neural networks and information gain metric for decision trees respectively. These updates are encrypted before communicated to the server.
4. Server decrypts and aggregates information from each client to compute an updated new global model. The cycle then repeats from step 1 if the model has not converged. Else, the federated process ends here.

Federated learning can be categorised into three main types: horizontal federated learning, vertical federated learning and federated transfer learning [45]. Horizontal federated learning is applicable when the data across clients share the same feature space but have different individual labelled instances in the data as shown in Figure 2.2.

On the contrary, vertical federated learning is applicable when the data across clients share the same set of instance but with different feature space. Federated transfer learning is then used in the scenario when both instances and feature space between data sets of clients differ. In this report, we are only interested in the application of horizontal federated learning as the feature space of data set obtain from turbofan engines will be the same across each engine.

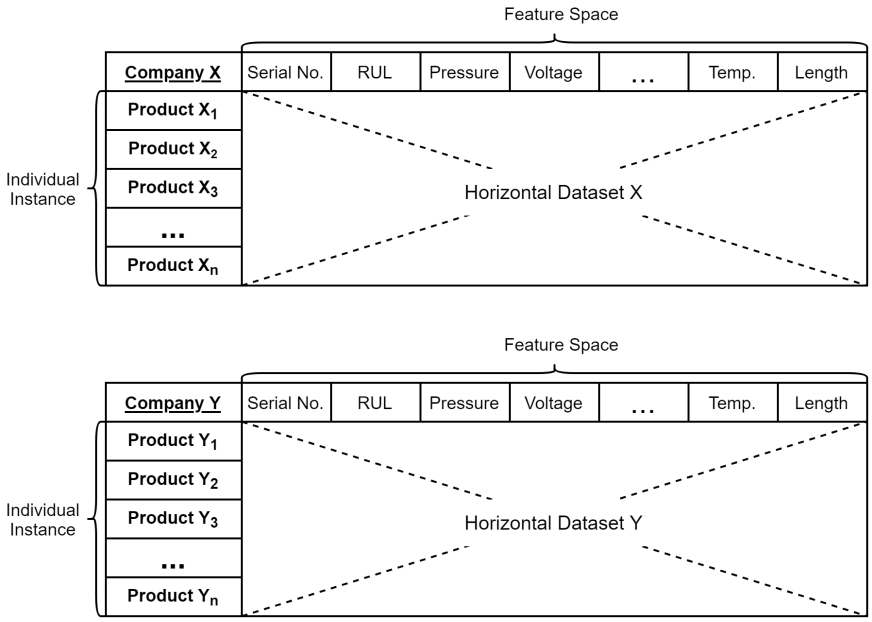

Figure 2.2: An example of a dataset used in horizontal federated learning

2.2.2 Federated Gradient Boosted Decision Tree

The first of the two federated techniques is the Federated Gradient Boosted Decision Tree (GBDT) which is supported by the Federated AI Technology Enabler (FATE) open-source Python package, an open-source project initiated by WeBank's AI Department [23]. We discuss the use of these packages in greater detail in Chapter 4.

Federated Forest

Federated forest was first proposed by Liu et al. [46] as a federated alternative of random forest proposed by Breiman [31], focusing on vertically partitioned datasets, where Liu et al. has shown through a series of experiments that their proposed algorithm produces a similar level of predictive accuracy as the centralised random forest equivalent.

Although the exact application of federated forest to vertically partitioned datasets is not relevant in the context of this report, Vaidya et al. [47] has further shown that application of a similar privacy-preserving random decision tree algorithm to horizontally partitioned datasets yields comparable accuracy to a centralised model.

However, it was noted that the result came with significant computational costs and according to Kholod et al. [48], the only available package present to perform federated decision tree is provided by FATE.

Federated Gradient Boosted Decision Tree

FATE currently supports multiple federated equivalent of commonly used machine learning algorithms such as tree-based and neural network algorithms. Specifically, FATE's Homo SecureBoost API trains a global GBDT on data with the same set of feature space held locally and privately by individual workers. Homo SecureBoost relies on the histogram-based split finding algorithm [49] to find the optimal split rule for a particular node and aggregates them in five steps:

1. Each client computes a histogram of gradients for each feature k, considering only a subset of values in each feature as potential split points
2. For each histogram created, the server performs secure aggregation by adding a different random number for each client such that it cancels out when aggregated. This allows the server to obtain the global sum while maintaining privacy
3. With the complete set of updates from each client, the server can then derive the global optimal split rule and communicate it back to the clients. The global optimal split is selected by maximising/minimising the selected metric such as information gain or Gini impurity
4. Clients will receive the optimal split rule and construct the next layer in the tree. Each client will never have information on the individual split rule proposed by another client. If terminating conditions are not met, the cycle will repeat from step 1
5. If terminating conditions are met such as maximal tree depth or minimum instance in a node, the fitting process is complete

2.2.3 Federated Neural Network

The Artificial Neural Network set up in Chapter 2.1.2.2 can similarly be adopted for federated learning. This is again supported by numerous federated learning packages such as PySyft [50], FATE [23] and dc_federated [24], both of which supports PyTorch, a deep learning machine learning library commonly used for building neural network architecture.

Federated Averaging

The core of a federated neural network is its aggregation algorithm which instructs how each local neural network model parameter gets aggregated at the server while preserving privacy. In this report, we rely primarily on the $FederatedAveraging$ (FedAvg) algorithm first introduced by McMahan et al. [43] which was described as a combination of stochastic gradient descent (SGD) performed locally at each client level before aggregating and averaged by a common server. The full FedAvg algorithm is shown in Algorithm 1.

Algorithm 1 $FederatedAveraging$ [43] K is the number of workers indexed by k, B is the local batch size, E is the number of local epochs and η is the learning rate

1: **Server executes:**
2: initialise w_0
3: **for** each round t = 1, 2, ... **do**
4: $m \leftarrow max(C \cdot K, 1)$
5: $S_t \leftarrow$ (random subset of m clients)
6: **for** each client $k \in S_t$ in parallel **do**
7: $w_{t+1}^k \leftarrow ClientUpdate(k, w_t)$
8: **end for**
9: $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
10: **end for**
11:
12: **ClientUpdate(k, w):** ▷ Run on client k
13: $\beta \leftarrow$ (split into batches of Size B)
14: **for** each local epoch i from 1 to E **do**
15: **for** batch $b \in \beta$ **do**
16: $w \leftarrow w - \eta \nabla \ell(w; b)$
17: **end for**
18: **end for**
19: return w to server

Federated Averaging works in the following steps:

- **Line 1** - A central server containing the global model initialises a set of weights w_0 to be communicated to individual client, if no prior training has been conducted. Else, the current set of global model weights w_t is sent to each client
- **Line 7** - Using the newly received weights, each client trains the local model using data residing locally within the client via stochastic gradient descent in line 16. This generates a set of local model gradients
- **Line 9** - The set of local model gradients are then communicated to a central server in line 19, without ever sending the underlying raw data. The server aggregates the gradients via FedAvg algorithm in line 9 which takes the weighted average of the gradients and uses it to update the global model weights
- **Line 3** - This cycle will continue until the global model converges

The application of federated artificial neural network can be adopted for any general neural network architecture which makes it a suitable candidate for the federated neural network architecture introduced in Chapter 5.

Chapter 3

Data

3.1 Overview

This report relies primarily on the turbofan engine degradation simulation data set produced by Saxena et al. [51], originally used as data in the 2008 Prognostics and Health Management (PHM) data competition. This data was generated using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) tool [52], which is an engine simulation software used to simulate large commercial turbofan engine with 90,000 lb of thrust.

The simulation can be carried out under three possible operating conditions: (i) Altitudes ranging from sea level to 40,000 ft, (ii) Air speed ranging from Mach 0 to Mach 0.9 and (iii) Sea level temperatures from approximately 15 to 40 degree Celsius.

The simulator can receive a maximum of 14 inputs such as fuel flow and efficiency/flow modifiers for each of the critical components in the engine. These components include the Fan, High-Pressure Turbine (HPT), High-Pressure Compressor (HPC), Low-Pressure Turbine (LPT) and Low-Pressure Compressor (LPC). A simplified diagram of the engine simulated is shown in Figure 3.1.

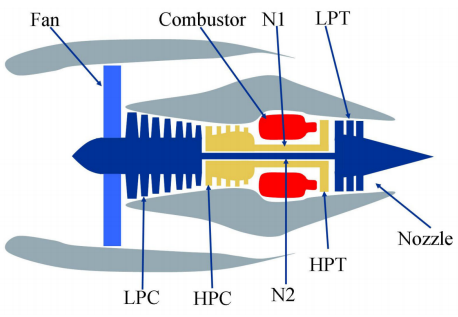

Figure 3.1: Simplified diagram of the simulated engine [52]

Based on the inputs provided, C-MAPSS models the engine degradation process and generates a total of 58 different sensor outputs of which 21 were provided in the simulation data set. For each of the sensors, a reading is captured for every unit of time defined as a cycle. A description of the sensor output is shown in Table 3.1.

Sensor No.	Symbol	Description	Units
1	T2	Total temperature at fan inlet	°R
2	T24	Total temperature at LPC outlet	°R
3	T30	Total temperature at HPC outlet	°R
4	T50	Total temperature at LPT outlet	°R
5	P2	Pressure at fan inlet	psia
6	P15	Total pressure in bypass-duct	psia
7	P30	Total pressure at HPC outlet	psia
8	Nf	Physical fan speed	rpm
9	Nc	Physical core speed	rpm
10	epr	Engine pressure ratio (P50/P2)	-
11	Ps30	Static pressure at HPC outlet	psia
12	phi	Ratio of fuel flow to Ps30	pps/psi
13	NRf	Corrected fan speed	rpm
14	NRc	Corrected core speed	rpm
15	BPR	Bypass Ratio	-
16	farB	Burner fuel-air ratio	-
17	htBleed	Bleed Enthalpy	-
18	Nf_dmd	Demanded fan speed	rpm
19	PCNfR_dmd	Demanded corrected fan speed	rpm
20	W31	HPT coolant bleet	lbm/s
21	W32	LPT coolant bleed	lbm/s

Table 3.1: Summary of sensor outputs from C-MAPSS [52]

Four data sets are provided from this simulation, each representing a number of engines simulated under different combination of operating conditions and fault modes. For example, data set FD001 contains 100 engines simulated at sea level with only one possible failure event (HPC degradation) while data set FD004 contains 248 engines simulated under six different operating conditions with 2 unique failure event (HPC degradation and Fan degradation). Table 3.2 shows a summary of each data set.

Data Set	FD001	FD002	FD003	FD004
No. of engines in training set	100	260	100	248
No. of engines in testing set	100	259	100	249
Number of operating conditions	1	6	1	6
Number of failure events	1	1	2	2

Table 3.2: Summary of sensor outputs from C-MAPSS [52]

Engines in all data sets are simulated starting from an initial operating condition at cycle 1 with a maximum RUL, with the unit of time defined in operating cycles of the engine. The initial condition differs between engines with varying extent of wear and manufacturing variability. Three additional fields are provided in each data set, representing the different initial operating settings of each engine. Hence, each engine will encounter a failure event at different cycle in time.

Each data set is made up of a training and test subsets. Engines in the training set are simulated until the point of failure while engines in the test set may be censored before a failure event is encountered. The true RUL for both the train and test data are also provided.

This report will focus on FD001 for both the centralised and federated models primarily because it has only one single failure event. Restricting to a single failure event would allow us to investigate the impact on performance of a balanced versus imbalanced data set on federated models. The existence of two or more failure events may mean that a balanced data set is in fact imbalanced as one worker may be inclined towards a particular failure event versus another worker.

3.2 Exploratory Data Analysis

FD001 contains 100 engines, each with its own time series development of the degradation process, documented by 21 different sensor readings. As seen in Figure 3.2, the average cycle when a failure event occurs is approximately at the 206th cycle. There is no clear correlation between engine numbers and its useful life.

Since the unit of time is defined in terms of cycle, the RUL for an engine at the maximum cycle will be 0. RUL is therefore defined in terms of the number of remaining cycles until a failure event occurs.

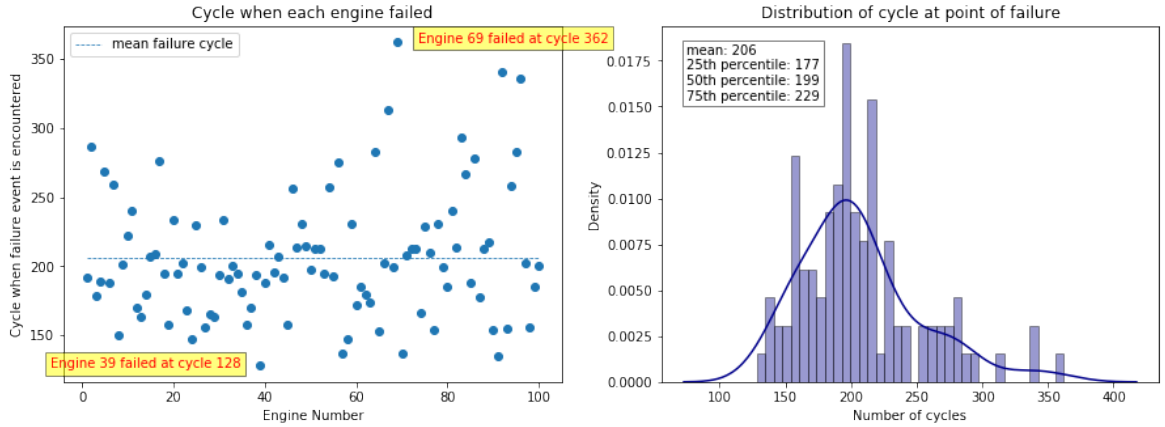

Figure 3.2: Scatter plot of cycle at failure event (left) and distribution of cycle at is encountered (right) in FD001 training data

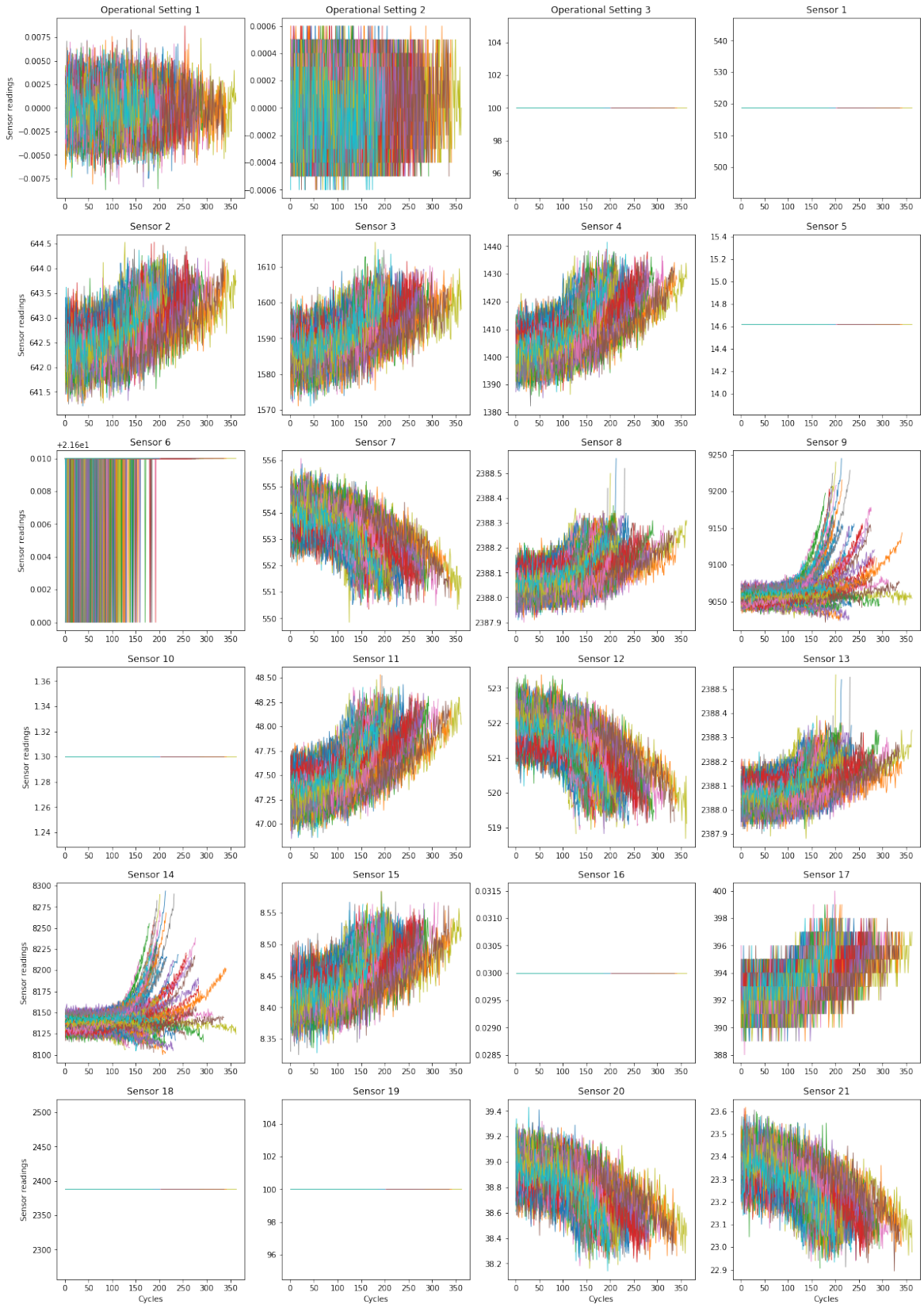

Figure 3.3: Operational and sensor readings for all 100 engines in FD001 training data

3.2.1 Operational Settings

Six different operating conditions were simulated across the four data sets but only one was applicable to FD001. As can be seen from Figure 3.3, these operational settings offer little to differentiate between an engine that is nearing the end of its RUL versus one that is fully functional in the near to medium term. The standard deviation for operational settings 1 and 2 is 0.002187 and 0.000293 which is insignificant and fluctuates wildly across cycles. Finally, operational setting 2 is a constant, providing no useful information for RUL prediction.

3.2.2 Sensor Readings

Similarly, we can see from Figure 3.3 that sensors 1, 5, 10, 16, 18 and 19 hold constant values that are not useful in RUL prediction. Sensor 6 fluctuates slightly within a tight bound with no discernible trend across engines.

For the remaining sensors, we can see that there is an obvious upward or downward trend as engines in the data set reach the end of their RUL. In particular, there is a slight divergence in readings for sensor 9 and 14 toward the end of the cycles. It should be noted that sensors 9 and 14 both measure the core speed of the engine as seen in Table 3.1.

3.3 Feature Engineering

In order to improve performance of the centralised model, a series of feature engineering step is performed to extract the underlying trend in the data. These steps were first proposed by Xin Chen et al. [53] and adopted with minimal changes. The main advantage of these steps is in their simplicity as they are reliant on established methods such as z-score normalisation and trend regression. The extracted features also do not lose their interpretability and relevance to the final prediction. Both these characteristics make it easy for troubleshooting and porting over to a federated environment.

3.3.1 Selection of Sensors

While Xin Chen et al. [53] opted to select relevant sensors by using Lasso regression, we have chosen to rely on the Mann-Kendall trend test [54]. The Mann-Kendall trend test is a simple non-parametric test to identify significant monotonically decreasing or increasing trends in time series data. The obvious advantage to using the Mann-Kendall test as opposed to Lasso regression or other methodologies test for trend is the ease of interpretation and implementation.

The use of Lasso regression requires additional parameterization of the L1-regularisation coefficient. In addition, Jun Wu et al. [55] relied on a composite selection criteria

(CSC) which scored the data based on a combination of its derived monotonicity and correlation, while Chuang Chen [56] relied on a combination of correlation and consistency indicators. A summary of selected sensors for each of these papers is shown in Table 3.3 which shows a general consensus.

This test is performed using the pyMannKendall [57] python package which does a pairwise comparison of a sensor's mid-range reading at cycle T with all other readings subsequent to cycle T. Let $x_1, x_2, ..., x_n$ represent data points in an ordered time series where x_j represents a sensor reading at time j. The Mann-Kendall statistic (S) is then given by:

$$S = \sum_{k=1}^{n-1} \sum_{j=k+1}^{n} sign(x_j - x_k) \tag{3.1}$$

where

$$sign(x_j - x_k) = \begin{cases} 1, & \text{if } x_j - x_k > 0 \\ 0, & \text{if } x_j - x_k = 0 \\ \text{-1}, & \text{if } x_j - x_k < 0 \end{cases} \tag{3.2}$$

The result of the Mann-Kendall test is shown in Figure 3.4 where sensors 7, 12, 20 and 21 returned a high negative value which is indicative of a monotonically decreasing trend. Sensors 17, 3, 13, 8, 2, 15, 4, 11 returned a high positive value which indicates the opposite. Similarly, sensors 14, 9 and 6 showed a slight increasing trend of a smaller magnitude. This result makes intuitive sense as it aligns with the general observation in Figure 3.3.

Thus, 12 sensors - 2, 3, 4, 7, 8, 11, 12, 13, 15, 17, 20 and 21 - are selected based on the Mann-Kendall test to be used in predicting RUL in FD001. The selected sensors are compared with results from other authors in Table 3.3 which shows a general consensus.

Source of comparison	Selected sensors	Total
Xin Chen et al. [53]	2, 3, 4, 7, 8, 11, 12, 13, 15, 17, 20, 21	12
Jun Wu et al. [55]	2, 3, 4, 7, 8, 9, 11, 12, 13, 15, 17, 21	14
Chuang Chen et al. [56]	2, 4, 7, 11, 12, 15, 17, 20, 21	9
Final selection	2, 3, 4, 7, 8, 11, 12, 13, 15, 17, 20, 21	12

Table 3.3: Comparison of selected sensors in FD001

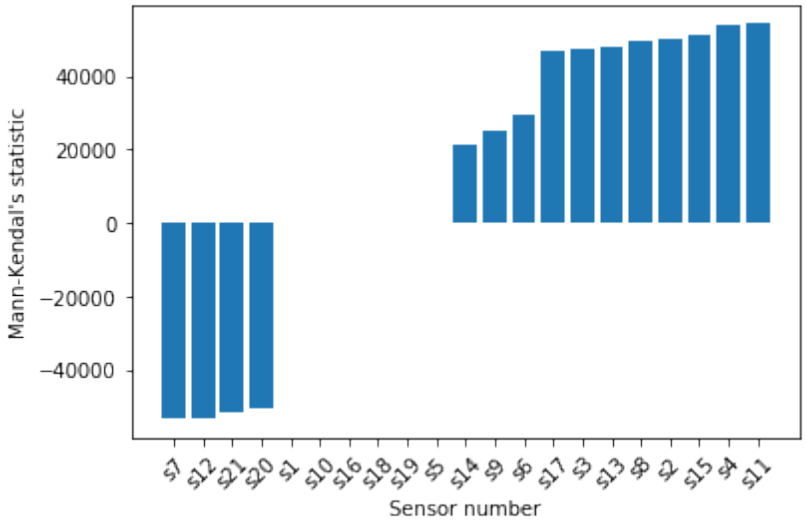

Figure 3.4: Mann-Kendall statistic for each sensor

3.3.2 Data Normalization and Polynomial Fitting

The selected sensor data are then normalised using Z-score normalisation to scale all sensors' data to the same range. Again, let $x_1, x_2, ..., x_n$ represent data points in an ordered time series where x_j represents a sensor reading at time j. Z-score normalisation is applied to x_i such that the normalised score $x_i{}'$ is defined as:

$$x_i{}' = \frac{x_i - \overline{X}}{stdev(X)} \tag{3.3}$$

where $\overline{X}$ represents the mean of X and $stdev(X)$ is the standard deviation of X.

Next, a 3-degree polynomial is fitted to the normalised sensor data as performed by [53]. This helps to reduce the noise in the data, preserving the core trend as shown in Figure 3.5.

3.3.3 Data censoring

It was shown in Figure 3.2 that the minimum number of cycles before the first failure occurs is 128. This suggests that a steady state exists where all engines are expected to operate optimally before the degradation process begins. This is visualised in Figure 3.2 where sensor readings are generally stable in its steady state before taking on a monotonically increasing or decreasing trend. Hence, RUL is capped at a pre-defined 150 cycle since predictions of RUL above this level may be spurious.

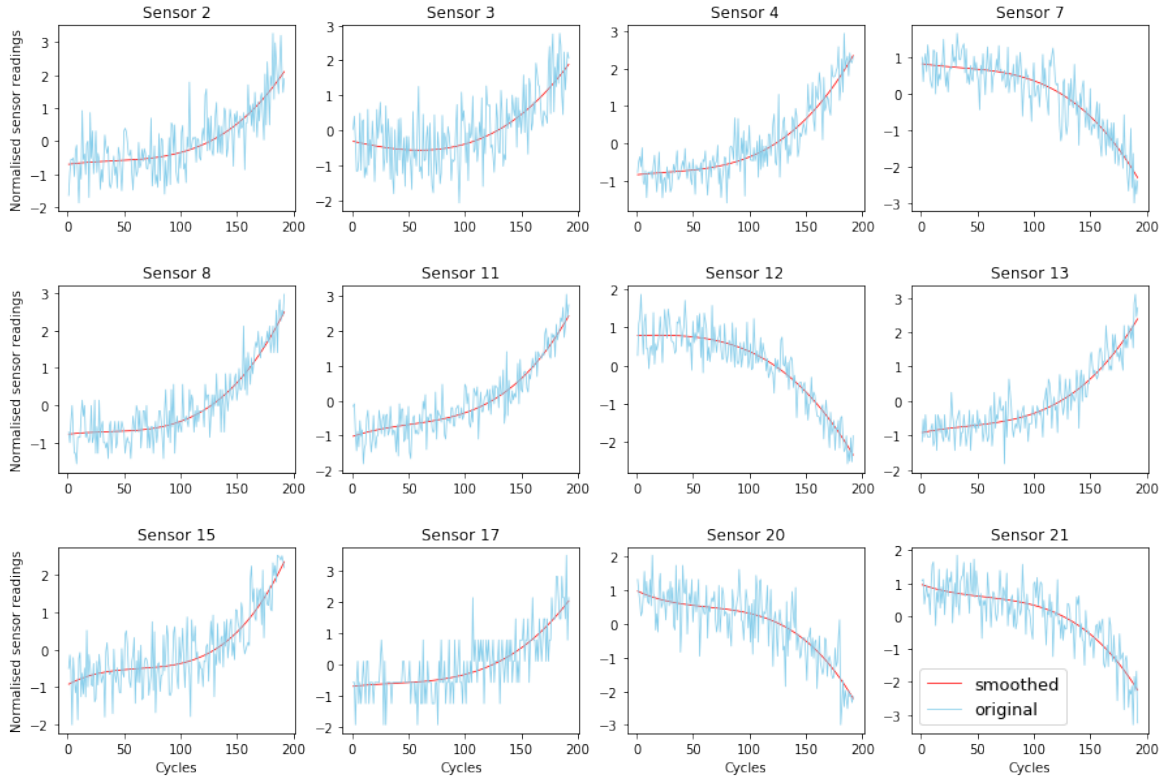

Figure 3.5: Smoothed normalised versus original normalised sensor data for Engine 1

Artificial right censoring of the training data is also performed which removes all sensor readings belonging to engines with a RUL exceeding a certain point. This simulates a realistic data set where time series data may not always be captured up to the point of failure. In this case, any sensor readings in the training data that has run beyond its 200th [58] cycle will be dropped from the training data set.

Whilst this would remove degradation trends beyond the 200th cycle, it would mean giving more weight to earlier degradation trend and lower fluctuations arising from the few engines with extended longevity. This is evident from Figure 3.3 where degradation trends beyond the 200th cycle are more spread out.

3.3.4 Feature Extraction

The re-engineered data is now prime for feature extraction. As proposed by Xin Chen et al. [53], two sets of feature - mean and trend - will be extracted per sensor data. However, sensor data for each engine must first be sliced into individual windows before the features can be extracted from each window.

This slicing mechanism is dependent on the variable length L which defines the size of each window and stride K, which defines the amount of overlap between consecutive windows.

Thus, for an ordered time series with n data points as defined for equation 3.1, the number of windows is formally defined as:

$$W_{L,K} = \lfloor \frac{n - L + 1}{K} \rfloor \tag{3.4}$$

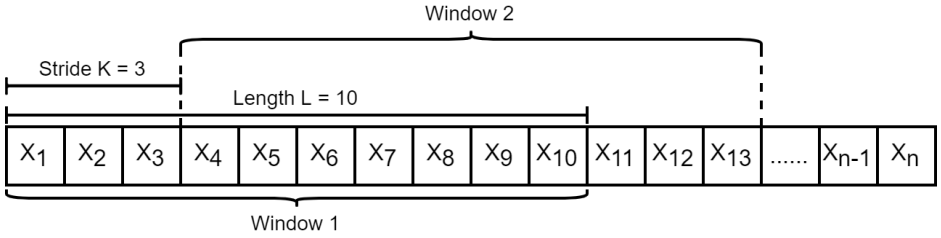

Figure 3.6: Illustration of the slicing mechanism

This mechanism is illustrated in Figure 3.6. In this report, the choice of length L is set as 20 and stride K is also set as 20. There will therefore not be any overlap between consecutive windows. Given any single window slice $w_i = x_0, x_1, ..., x_L$ in the time series data of an engine, its mean is then calculated as the average value of the smoothed normalised reading:

$$mean_{w_i} = \frac{1}{L} \sum_{i=1}^{L} x_i \tag{3.5}$$

Trend is then defined as the regression coefficient derived from the time series linear regression of w_i where the target variable is each of $x_0, x_1, ..., x_L$ with the independent variable being time defined in cycles:

$$x_t = \beta_i t + \alpha_i \tag{3.6}$$

where β_i is the regression coefficient for w_i. Finally, the RUL for each window is simply set equal to the RUL for the last time series data in that window.

3.4 Federated Data Sets

Federated learning is applicable when there are multiple independent workers with isolated pools of private data. These pools of federated data can take the form of either a horizontal or vertical data set.

A federated data set is said to be horizontal if the individually distributed data sets have homogeneous feature space as shown in Figure 2.2. This means that uniquely identifiable data points across each pool share the same features. FD001 fits the definition of a horizontal data set, since all engine trajectories share the same number of sensor output.

A precursor to simulating and testing a horizontal federated learning model is therefore access to multiple sets of data with the same feature space across unique samples. In the context of this report, we are interested in splitting both train and test sets of FD001 into three and five sub data sets, each of which contains engines that have a similar degradation process.

The technique chosen to perform this split is time series hierarchical clustering using Dynamic Time Warping (DTW) as the distance metric [59]. DTW performs a pairwise comparison of data points in two time series to derive the extent of similarity (or difference) between each sequence.

It can be applied to time series of varying length but aligned in terms of time, as is the case in FD001. For our purpose, clustering is performed on the extracted trend feature from Chapter 3.3.4. The result of the clustering is shown in Figure 3.7 where each of the 100 engines are allocated to a cluster.

Three general clusters can be seen from this exercise which forms the basis of the split of FD001 into three imbalanced federated train data sets of size 85, 13 and 2 respectively.

Figure 3.8 shows the difference in trend between the 3 clusters. In addition, a balanced train and test data set was also prepared by randomly assigning 34, 33 and 33 engines to workers A, B and C respectively.

The same steps were then reapplied to arrive at five splits of data for five workers on both an imbalanced and balanced basis. This resulted in a balanced data set of 20 engines and an imbalanced train data set of 1, 1, 13, 5 and 80 engines respectively. Appendix A shows the dendrogram and trend clusters under the five workers split, similar to Figure 3.7 and 3.8.

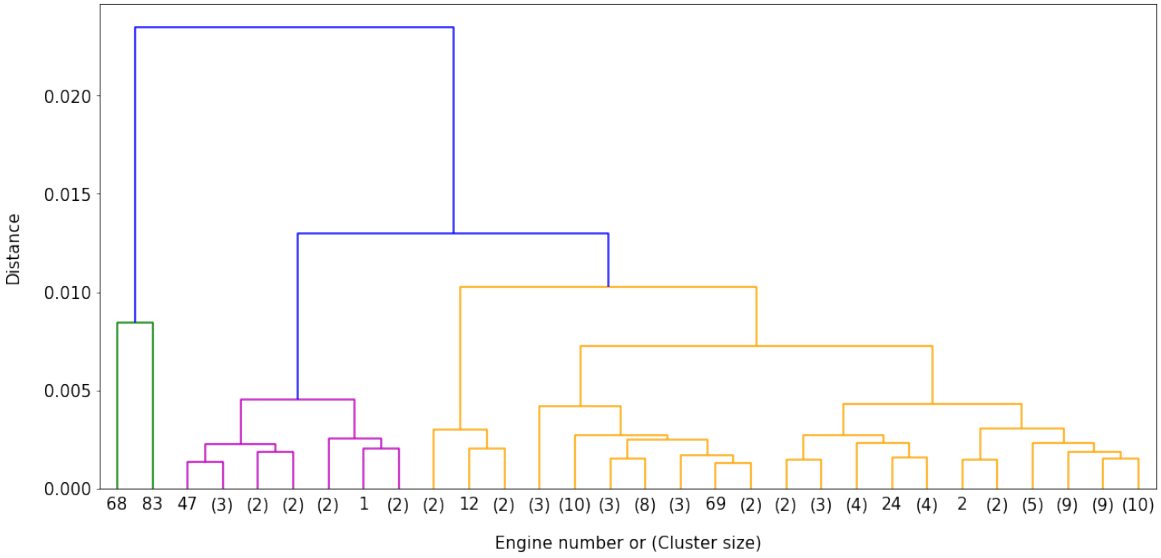

Figure 3.7: Dendrogram of 3 allocated clusters for 100 engines in FD001 (root colour (green, purple, orange) represents each of the 3 clusters)

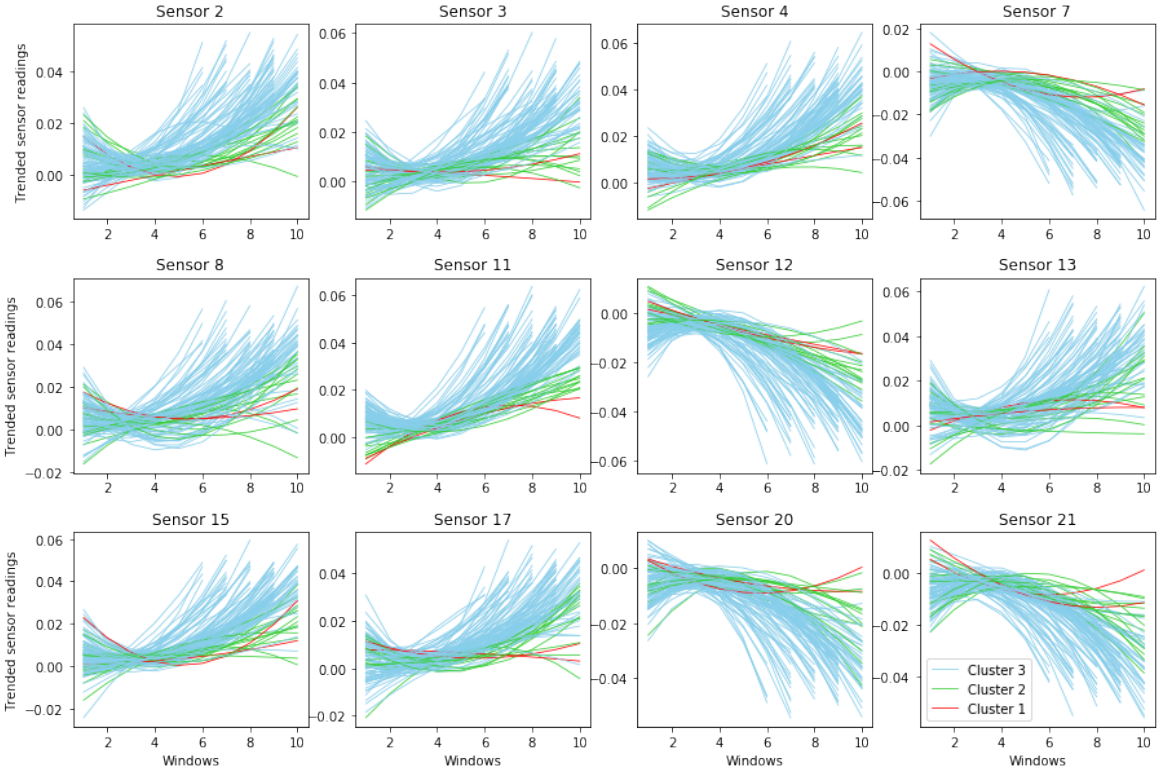

Figure 3.8: 3 allocated trend clusters for each of the 100 engines in FD001

Chapter 4

Experimental Setup

4.1 Introduction

Introduced by Google in 2016 [13], federated optimisation or federated learning as a technique was utilised to shift from a centrally trained model to a shared model that is locally trained on mobile devices. The trained model parameters are then aggregated on a central server which never accesses the raw training data. This technique enhances privacy, alleviates collaborators' concerns over privacy preservation and offers the benefits of a scaled up machine learning model.

Since then, the application of federated learning techniques to failure prediction in manufacturing has made several advances in the last two years. Silveria et al. [15] applied horizontal federated learning techniques to predict springback for sheet steel materials whilst Ge et al. [16] applied both horizontal and vertical federated learning techniques to predict failure in a Bosch production line.

4.1.1 Previous Studies

Federated learning has also been applied in the context of the turbofan engine degradation simulation data set used in this report. Rosero et al. [22] deployed a traditional three layers feed-forward neural network under both a centralised and federated basis to compare the performance of two key federated learning algorithm - Federated Averaging (FedAvg) and Federated Proximal Term (FedProx) in estimating RUL. The use of FedProx is beyond the scope of this report.

Although Rosero et al. applied federated learning to the turbofan engine data set, it was performed in a theoretical simulated mode without the use of publicly available federated learning frameworks to investigate the difference between FedAvg and FedProx algorithm. There was also no comparisons to the model performance of other established centralised models with respect to this data set that would have provided an understanding of the performance drawback of a decentralised model.

Lau [60] on the other hand demonstrated a proof of concept (POC) using PySyft [50] and Pygrid [61] for deploying and training machine learning model on edge devices in a federated environment. Edge devices in this case are represented by individual turbine engines with the aim of estimating RUL of each engine as they operate.

This POC showed that deploying a decentralised model is plausible but went no further to discuss implementation challenges and comparison of model performance. Table 4.1 shows a summary of previous studies on this data set as well as the gaps that are covered in this report.

Papers	Model	Summary
Chen et al. [53]	Random Forest	• Applied a series of feature engineering steps to improve model performance which is adopted in this report • Showed that a computationally efficient centralised RF model can perform on par with other more advanced models
Wu et al. [55]	Deep LSTM	• Introduces a novel Deep-LSTM model to predict RUL on a centralised basis and showed that it can perform on par with other more advanced models • Model is optimised via grid search to arrive at an optimal layer and neuron number
Chuang et al. [56]	Hybrid SVR/LSTM	• Introduced a hybrid model with a specific loss function to perform risk-averse RUL predictions, which penalises predictions that occur after the true failure cycle more • Performed on a centralised basis
Chu et al. [62]	Hybrid CNN/LSTM	• Proposed a hybrid centralised model that outperforms most other advanced models
Rosero et al. [22]	Fed MLP	• Showed that the performance of federated models using FedAvg and FedProx aggregating algorithm is on par with its centralised counterpart • Showed that FedAvg converged faster than FedProx but with higher uncertainty • Evaluated the impact on number of workers on the performance of federated models using only a balanced data set • Does not discuss how the federated learning process was implemented and can be deployed
Lau [60]	Fed LSTM	• Simple proof of concept using PySyft • Only applied FedAvg algorithm • No formal results analysis

Table 4.1: Summary of previous studies (Red highlights gaps covered in this report)

4.1.2 Aims of Experimental Setup

The aim of our experimental setup is to enable us to address some of the gaps in previous studies and provide another perspective to practically deploying a federated learning environment. More specifically, the setup should:

1. Be comparable with previous benchmark studies on the same data set using the same scoring criteria
2. Demonstrate the practical deployment of another publicly available federated learning package, in addition to the POC performed in [60]
3. Highlight the key steps and considerations for practitioners when deploying a federated learning framework, particularly in a production environment

4.2 Benchmark Models

In order to understand the effectiveness of a federated model, we first need to establish benchmark models and metric which will allow us to compare our model performance. These benchmark models are trained, optimised and tested on the same FD001 engine degradation data set across a wide range of model complexity.

A summary of these models and the associated metric is shown in Table 4.2. Of these morels, there is only one previous study [22] on the performance of a decentralised model on data set FD001.

Source of comparison	RMSE
Random Forest [53]	12.01
Deep LSTM [55]	18.43
Hybrid SVR/LSTM [56]	19.11
Hybrid CNN/LSTM [62]	13.73 - 28.64
Federated MLP (FedAvg and FedProx) [22]	20.90 - 23.92

Table 4.2: Benchmark performance on FD001 test data using RMSE

The selected metric is Root Mean Square Error (RMSE), which is a consistent metric used across previous studies on the same data set:

$$RMSE = \sqrt{\frac{1}{N}(RUL_t - \hat{RUL}_t)^2} \tag{4.1}$$

In addition to the models in Table 4.2, we have also prepared baseline models using survival analysis and simple supervised machine learning techniques discussed in Chapter 2. These includes RUL estimation based on the Kaplan-Meier estimator,

Cox PH model, random forest regression and a simple three layer neural network architecture.

The rationale for these additional models is to compare traditional survival analysis techniques and more importantly to enable a fair comparison to its decentralised counterparts. Deploying a federated learning environment requires the support of publicly available packages such as FATE, PySyft and dc_federated.

However, some of these packages currently support only a number of standard aggregation algorithm such as FedAvg on which a limited number of out-of-the-box federated learning algorithm is built. This greatly restricts the level of model complexity that we can practically implement on a federated basis. Replicating benchmark models such as hybrid SVR/LSTM and CNN/LSTM is therefore not possible at this stage, without developing our own algorithm tied to a specific deployment architecture. This is discussed in greater detail in Chapter 4.3.

4.3 Federated Learning Environment

At the point of writing this report, there are a number of open-source Python packages that support a federated learning environment, with different levels of complexity and documentation. These include but are not limited to:

1. Federated AI Technology Enabler (FATE) [23] framework developed by Webank's AI department
2. PySyft [50] developed by Openmined
3. Federated Learning and Differential Privacy Framework [63] developed by Sherpa.ai
4. dc_federated [24] developed by Digital Catapult
5. TensorFlow Federated (TFF) [64] developed by TensorFlow
6. PaddleFL [65] developed by Baidu

In this chapter, we will focus primarily on FATE and PySyft, which are the two most developed and supported open-source federated learning library. They also support industrial grade deployment with graphical user-interface to manage individual workers which is not seen in other open-source frameworks. A more comprehensive and detailed review of open-source federated learning environment can be found in [48] and [66].

4.3.1 Key Components

Although the concept of federated learning is simple, that is to train a global model using all available data while preserving its privacy, the practical implementation of a secure and privacy preserving federated environment from scratch is not. The idea therefore is to rely on well defined open-source federated learning packages that performs a combination of communication, federated modelling and coordination tasks between each worker. This architecture is illustrated in Figure 4.1.

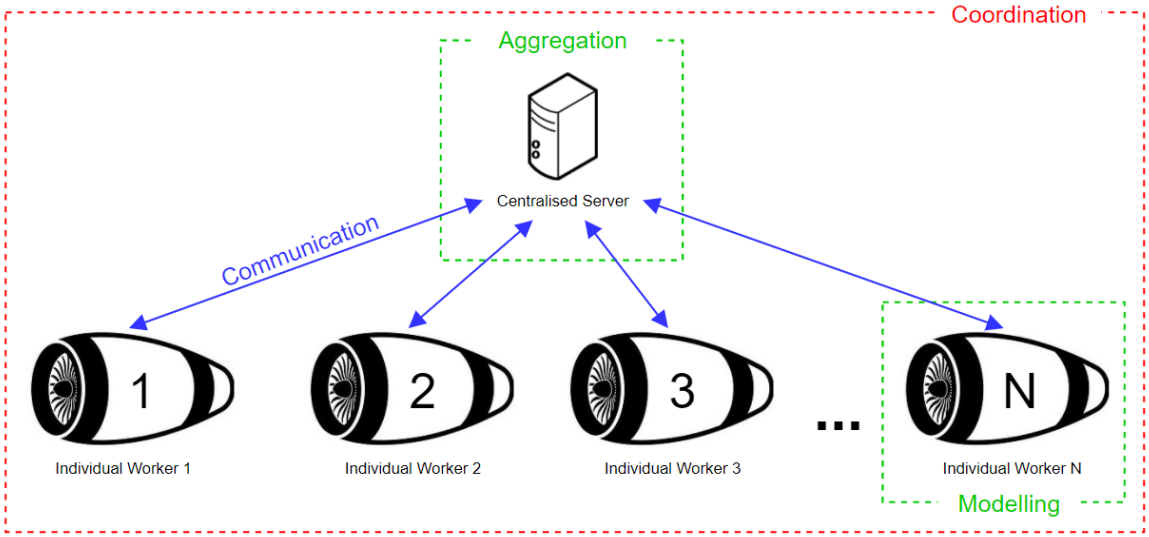

Figure 4.1: Architecture of a federated learning framework

Communication architecture is critical to ensure that each individual worker is able to send locally trained model parameters and thereafter receive globally aggregated model parameters. Communication between each node and the server needs to be secure and is usually performed via HTTPS as is the case in dc_federated [24] package. This responsibility is handled by PyGrid and FATE Flow for PySyft and FATE respectively.

According to [48], there are two main communication setups among federated learning systems - centralized and decentralized schemes. The former requires a central server through which the federated learning process among the workers is coordinated while workers in the latter are able to communicate directly to aggregate a common global model. It should be noted that both FATE and PySyft operates using the centralized scheme.

Federated modelling and aggregation of model parameters is the centrepiece of any federated learning system. This is further broken down into two sub parts - the federated machine learning model and its associated aggregation algorithm as well as the encryption protocol.

The first determines the type of machine learning models that is supported within the package. For example, PySyft primarily supports FedAvg aggregation algorithm

which enables neural network modelling supported by PyTorch [67] and TensorFlow [68]. On the other hand, FATE supports a number of different aggregation algorithm which in turn enables a number of out-of-the-box federated solutions of common machine learning models such as logistic regression, decision trees, transfer learning and K-means clustering. These are usually accompanied by the common FedAvg aggregation algorithm or Google's secure aggregation protocol [69].

The second part involves the encryption of modelling parameters during the process of aggregation. Encryption is a crucial and necessary step in federated learning to prevent adversarial attacks from both individual workers and model owner. Such attacks include membership attacks, unintended memorisation and model inversion attacks. The main aim of these attacks is usually to learn specific data features of a contributing worker during the aggregation process despite the security provided by encryption.

Details of encryption algorithm employed by each of the federated learning packages is out of the scope of this report. However, we note that both PySyft and FATE employs secure Multi-Party Computation (MPC) Homomorphic Encryption (HE) within its suite of encryption techniques.

Coordination relates primarily to the practical deployment and orchestration of federated learning activities among the various workers and the server. The ability to do so at an industrial scale with more than a thousand edge devices is particularly important here.

Existing open-source federated learning packages are most commonly deployed on Linux based operating systems, with PySyft also supporting Windows, iOS and Android mobile devices. Both FATE and PySyft enable pre-deployment simulation mode and also actual subsequent deployment at scale, while TFF does not yet support federated deployment [66]. Graphical user-interface is also provided by FATE through FATE Board and PyGrid through PyGrid Admin to manage the federated learning process.

Features	FATE	PySyft
# of commits	7,611 (July 2021)	9,457 (July 2021)
# of contributors	50 (July 2021)	362 (July 2021)
Operating System	Mac Linux	Mac Linux Windows iOS Android
Data Partitioning	Horizontal Vertical	Horizontal Vertical
Mode	Simulated Federated	Simulated Federated
ML Algorithm	NN (classification only) Various regression models GBDT Transfer Learning	NN within PyTorch and TensorFlow frameworks
Encryption Protocol	MPC HE RSA Diffne Hellman Key Exchange Feldman Verifiable secret sharing Oblivious Transfer	Differential privacy MPC HE
Deployment platform	Docker Coordinated using FATE Flow	Docker Coordinated using PyGrid
Graphical UI	FATE Board	PyGrid Admin

Table 4.3: Summary of comparison between FATE and PySyft [48]

4.3.2 Selected Federated Learning Framework

Both FATE and PySyft are well established open-source libraries for deploying a federated learning environment with comparable features as summarised in Table 4.3. Both are also supported by a core team of developers at WeBank and OpenMined who have made continuous progress to improve, update and address issues within each framework.

However, a key aim of our experimental setup is to be comparable with centralised benchmark models which FATE is able to fulfil with its range of out-of-the-box solutions, including neural networks and GBDT. While PySyft and PyGrid can offer

a similar setup for neural network, deployment of a non-neural network based federated learning algorithm is more complicated since only FedAvg is implemented in the library [66].

PySyft and PyGrid provide a limited set of documentation (in the form of a README file [50]) at the time of this report compared to FATE which makes modification of source code to implement customised federated algorithm difficult. However, this is expected to change given the level of support and number of developers working on PySyft and PyGrid.

According to [66], FATE is the only open-source federated learning framework that supports decision trees. Hence, for the propose of this report, FATE has been selected as the default federated learning framework to implement the decentralised machine learning model to estimate RUL.

However, as FATE's federated neural network does not support regression task, we have also implemented Digital Catapult's dc_federated federated framework as an alternative. This limitation of FATE is a major drawback to deploying FATE in a commercial setting, despite its support for a multitude of other models.

4.3.3 Review of FATE architecture

The selected FATE [23] framework used in this report is an open-source project developed by WeBank's AI department. Although it has a fairly extensive documentation [70] compared with other federated learning systems at the point of writing this report, the setup process remains convoluted.

Many of the key steps and API description are either incomplete or are mistranslated from Chinese to English. This chapter aims to document and review the key steps in setting up FATE under a standalone deployment setup while addressing key challenges encountered.

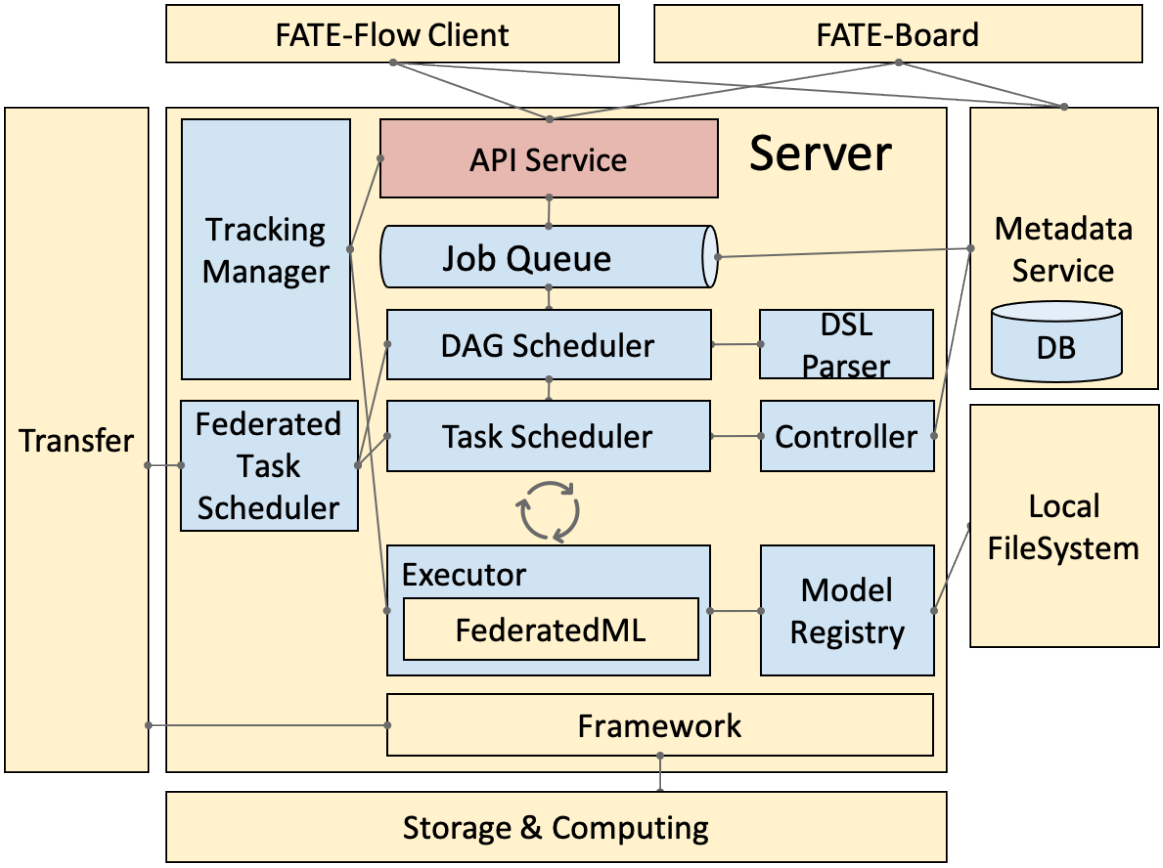

Figure 4.2: Architecture of FATE federated learning system [23]

4.3.3.1 Standalone Deployment

FATE v1.6.0 can be deployed in two ways. The first is standalone deployment, which is used for prototyping and simulation of a federated learning environment before actual deployment. The second is cluster deployment which supports large scale deployment to remote machines for actual multi-party federated learning.

Standalone deployment is the focus of this report to serve as a proof of concept of the usability of FATE. The standalone mode also acts as a precursor for future actual deployment since the general federated learning pipeline remains the same when transitioning to cluster deployment, with the exception of initial server and networking setup.

Standalone deployment using Docker containers is the recommended choice. However, there is an immediate OS compatibility issue as FATE is designed to run on Unix based machines. Remote machines with Windows operating system would not be able to run FATE. To set up FATE for our purpose, we had to run an Ubuntu 20.04 distro via Windows Subsystem for Linux 2 (WSL2).

While this enabled us to use FATE, the use of WSL2 consumed a non-trivial amount of memory which could be an issue for remote machines that have limited processing

capability. The hardware setup used in this report is shown in Table 4.4.

Setup	Recommended	Actual
Operating System	CentOS Linux release 7	Ubuntu 20.04 on WSL2
Processor	8 cores	4 cores
RAM	16GB	16GB
Storage	500GB	50GB

Table 4.4: Recommended versus actual hardware setup to deploy FATE v1.6.0

In order to execute federated learning models other than the test examples provided, it was necessary to mount a directory on the host machine to the Docker container. This is strictly necessary on a standalone basis so that changes to the federated models can be made on the host machine and be subsequently accessed by the container. Bind mount is performed by amending the "install_standalone_docker.sh" file to run the Docker container using the following command instead:

```
$ docker run -d --name fate -p 8080:8080 --mount type=bind,source="$(
    ↪ pwd)"<directory-on-host>,target=<directory-on-container>,
    ↪ readonly fate:latest /bin/bash
```

The provided tests in FATE-test can be executed to verify that the container is correctly set up. FATE Board, which is used to manage each federated process or jobs, should also be available at http://<host-ip>:8080 when the container is executed.

4.3.3.2 Modelling Pipeline

According to FATE's documentation [70], the FATE pipeline is "a high-level python API that allows user to design, start, and query FATE jobs in a sequential manner". Individual pipeline components shown in Figure 4.3 can be used modularly, allowing customisation of a single federated learning pipeline. This pipeline can then be executed with a single command.

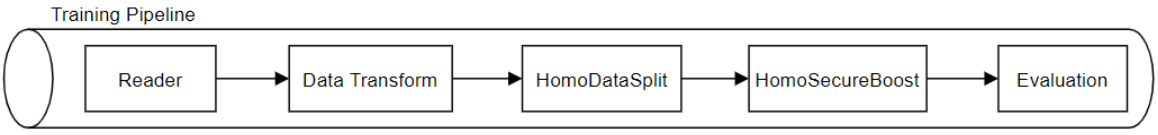

Figure 4.3: FATE training pipeline used in this report

This feature sets FATE apart from other federated learning frameworks. The modular pipeline components allow easy customisation of federated learning processes such as applying train/test split and one-hot encoding to federated data sets. Similar pre-packaged functional modules are not yet available or seen in other federated

learning framework. For the purpose of this report, the $HomoDataSplit$ component is used to split the training data into train and validate data sets. A list of sample components is shown in Table 4.5.

Component	Description
Reader	This component loads and transforms data from storage engine so that data is compatible with FATE computing engine
Federated Sampling	Federated Sampling data so that its distribution becomes balance in each worker. This module supports standalone and federated versions.
Feature Scale	Module for feature scaling and standardization
OneHot Encoder	Transfer a column into one-hot format
Homo-NN	Build homogeneous neural network model through multiple workers
Homo Secure Boosting	Build homogeneous secure boosting model (GBDT) through multiple workers
Evaluation	Output the model evaluation metrics for user

Table 4.5: List of sample components used in FATE pipeline [70]

4.3.3.3 FATE Board

The executed FATE pipeline can then be tracked and managed via FATE Board, which is a GUI that allows users to track and manage historical and currently running jobs. Although much of its capability can also be performed via FATE's command line interface (CLI), FATE Board is particularly useful in standalone deployment to prototype federated models and understand the interaction and communication between each worker.

For example, run logs and run time for each worker in the federated process can be accessed and viewed via FATE Board. It also provides an interactive interface for users to examine individual components in the pipeline as shown in Figure 4.4.

In a cluster deployment scenario however, the FATE Board would potentially have limited impact since most remote machines would not typically come equipped with a screen through which users can interact with the GUI. The ability to manage each worker and the federated process would then fall back to FATE's CLI - FATE Flow, which has a fairly well documented API [70].

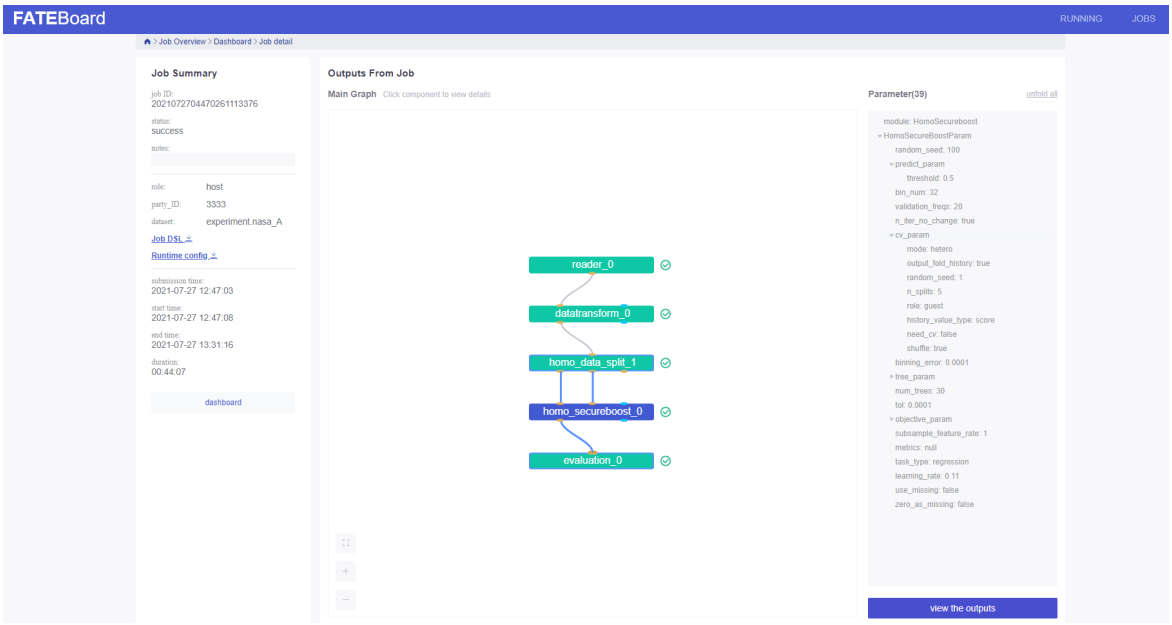

Figure 4.4: Component set up for a job ID in FATE Board

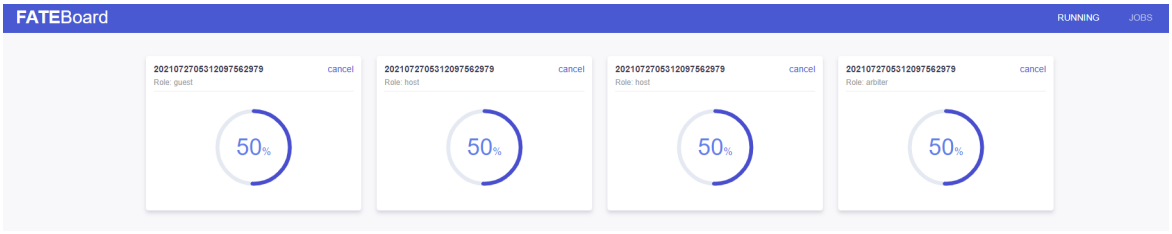

Figure 4.5: Progress bar for jobs running in FATE

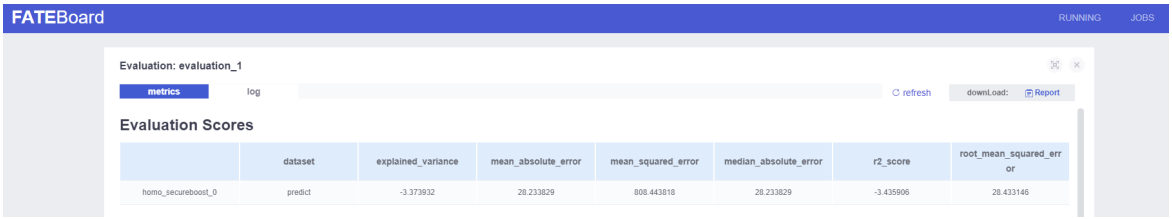

Figure 4.6: Display of model evaluation results in FATE Board

4.3.3.4 Evaluating Results

The FATE Board also provides an early indication of the model's performance using the *Evaluation* pipeline component which calculates metrics such as Mean Absolute Error (MAE), R2 Score and Root Mean Squared Error (RMSE) as shown in Figure 4.6. However, it is insufficient to rely solely on that. To enable additional analysis and comparison with centralised models, the train and test results have to be extracted via FATE Flow CLI using the following command:

```
$ flow component output-data -j <job_id> -r <role> -p <worker_id> -
    ↪ cpn <component_name> --output-path <directory-on-container>
```

This outputs the predicted results for each worker to the indicated directory. This works only on a standalone basis for all workers since each individual worker is hosted in a common simulated container. In a cluster deployed environment, access to predicted results would not be possible without access to the remote machine's CLI. Results of the federated model is discussed in Chapter 6.

Chapter 5

Model Architecture

5.1 Problem Statement

Before delving into the detailed model architecture selected, we first have to understand the problem that we need to solve with these models. As discussed in Chapter 3, we will be attempting to model the Remaining Useful Life (RUL) of individual turbofan engines at each cycle in their operational life. Each of the 100 individual engine in the training set contributes a time series of sensor readings. A unit of time is then defined as a single operating cycle.

For example, engine 39 survived the shortest before failure with 128 cycles before failure while engine 69 survived the longest with 362 cycles before failure. The RUL for engines 39 and 69 at their respective failure cycle is therefore 0. Similarly, the RUL for engines 39 and 69 at cycle 100 and 300 is 28 and 62 respectively.

Hence, the objective of each model discussed in this chapter is to predict the RUL for each engine at every cycle in time. This problem is best framed as a single dimensional regression problem with a common RMSE scoring metric.

5.2 Baseline Centralised Models

Baseline centralised models for estimating RUL covers both statistical and machine learning models as shown in Figure 2.1. The theoretical background underlying these models have also been introduced in Chapter 2. Here, we will discuss the actual architecture and parameters employed to perform RUL predictions using these models.

5.2.1 Kaplan-Meier

The Kaplan-Meier model involves estimating a single survival probability curve from the entire training data set. The estimated survival probability curve is then assumed to be representative of the population true survival curve, which includes the test data set.

In order to estimate RUL, the Restricted Mean Survival Time (RMST) is calculated from the survival curve, which gives the average survival time for engines in the data set. Since the survival curve is assumed to be representative of the population, all engines in the test data set will have the same RMST. The RUL at an engine's last cycle can then be calculated as the difference between the RMST and the cycle, floored at zero. This is an over-simplified model but it serves as a good baseline before more in-depth survival models are explored.

5.2.2 Cox Proportional Hazard

The Cox PH model first involves estimating the exponent portion of Formula 2.5 which is known as the log-partial hazard. It is only necessary to estimate the log-partial hazard under the Cox model to understand the relative time to failure since the baseline hazard function is assumed to be the same across all engines.

It is important to note at this stage that the log-partial hazard itself does not give an indication of the RUL at each cycle. Although it is sufficient in predictive maintenance to set a threshold for the log-partial hazard before performing maintenance, additional steps need to be taken to estimate RUL.

This is done by fitting an exponential model [58] to the relationship between log-partial hazard and RUL. For every estimated log-partial hazard from the Cox model, we can then re-estimate the corresponding RUL.

Similar to the Kaplan-Meier model, this set up does not lend itself well to solving our problem statement as we will discuss in Chapter 6.

5.2.3 Neural Network

A neural network is one of the more common machine learning candidate when it comes to solving regression problems. For our problem statement, we applied an architecture with 16, 32 and 64 nodes in each of the three hidden layers and 1 node in the output layer. The input layer accepts 24 features (12 selected sensors x 2 extracted feature per sensor). According to Heaton [71], two or more hidden layers allow the neural network to learn more intricate representations of the data. A neural network with three hidden layers was therefore selected which is sufficient to learn the data yet not overly complex for the purpose of federated learning.

We note that this is relatively simpler than the Deep LSTM model proposed by Wu et

al. [55] which has 5 layers with 100 neurons in each layer. The trade off of using a simpler model is weaker predictive performance but a more efficient implementation in a federated environment, particularly when deployed to weaker edge devices.

Relu activation function is used for each of the hidden layers and mean squared error is selected for the loss function. This set of hyperparmeters was selected using randomised grid search over 500 iterations.

This is a relatively simple neural network model compared to the selected benchmark models in Table 4.2 which includes hybrid neural network models. Remote machines running federated models are often low powered machines with limited processing power and memory. Hence, deployment of complex multi-layered and highly optimised hybrid models would usually be technically constrained. Existing federated learning frameworks such as FATE and PySyft also provide limited to no support for aggregating model parameters of complex models.

The selected neural network architecture is then used to estimate the RUL based on the extracted features - mean and trend - for each window of operating cycles. The predicting ability of this simple neural network is therefore dependent on the extensive feature engineering steps performed as discussed in Chapter 3.3.

5.2.4 Random Forest

The random forest set up to this regression problem is equally simple. This algorithm is based on 20 decision trees with a mean squared error loss function. The square root function is selected as the maximum number of features to consider when searching for the best split. Again, despite the simplicity of this model, it has one of the better performing results as discussed in Chapter 6.

Hyperparameters	Centralised Random Forest	Federated GBDT
Task type	Regression	Regression
# of trees	20	20
Max depth	None	10
Loss function	MSE	LSE
Max features	# of features	SQRT(# of features)

Table 5.1: Centralised Random Forest vs Federated GBDT Hyperparameters

5.3 Federated Learning Architecture

5.3.1 Hyperparameters in the Federated Process

Hyperparameters in a federated environment control the overall federated learning process. This is a set of parameters that operates at a level above the parameters of the federated model and are more often than not dictated by the federated environment. This can include the following:

1. Number of workers involved in each round of aggregation
2. Number and homogeneity of data points in each worker's data set
3. Number of aggregation rounds between the local and global models

The number of workers involved in our experimental setup is initially fixed at three workers. This is based on practical consideration of our machine's ability to handle significantly more workers, especially when simulating using FATE. Without actually deploying federated learning in a production environment, it is difficult to simulate a large number of federated workers due to hardware constraints.

We have therefore tested the impact of variation in the number and homogeneity of data points in each worker's data set on the federated learning process. This was possible by splitting FD001 train and test data set into three sets via time series clustering and randomised equal split as discussed in Chapter 3.4.

Based on the initial set of results, we then replicated the entire federated learning pipeline but for five workers to validate our findings.

5.3.2 Federated Models

The selection of federated learning models is primarily restricted by the support provided by federated learning frameworks discussed in Chapter 4.3. Whilst PySyft provides support for neural network based architecture with a FedAvg aggregation algorithm, FATE provides a wider range of federated models, including neural networks, various regression models and the selected gradient boosted decision tree (GBDT).

Gradient Boosted Decision Tree

FATE's GBDT or Homogeneous Secureboost supports both classification and regression tasks and is well suited for our problem statement. The theoretical background of the secureboost aggregating algorithm is discussed in Chapter 2.2.2.

For the purpose of training the model, the training data set is split into train and validation sets based on a 60/40 proportion, using FATE's $HomoDataSplit$ pipeline

component. A set of hyperparmeters was then selected using randomised grid search over 100 iterations.

The final GBDT model is based on 30 decision trees each with a maximum depth of 5 splits and optimised using a learning rate of 0.11. The maximum depth in this case determines the number of aggregation rounds with the server.

Each individual worker to the federated environment will have access to the same model setup and subsequently train and share a local model before receiving globally aggregated model parameters. Prediction of local test data set will then be performed using these aggregated global parameters.

Hyperparameters	Centralised NN	Federated NN
Task type	Regression	Regression
Hidden layers	32/64/128	64/128/256
Activation Function	relu	sigmoid
Learning rate	0.2	0.03
Loss function	MSE	MSE
Optimiser	Adam	Adam

Table 5.2: Centralised NN vs Federated Neural Network Hyperparameters

Neural Network

As FATE's federated neural network supports only classification task, it is not possible to utilise the FATE framework to predict RUL via regression. Hence, Digital Catapult's dc_federated framework was selected for this purpose. However, it is not possible to take advantage of FATE's modularity and GUI as dc_federated is an entirely separate framework from FATE.

Similar to GBDT, a randomised grid search was performed to select the optimal hyperparameters. This resulted in a simple 3 layer neural network with 64, 128 and 256 nodes per layer with Sigmoid activation function in each hidden layers. A learning rate of 0.03 with 12 rounds of iteration per aggregation.

A round of aggregation is defined as receiving the global model update, retraining the local model parameters and communicating the updated local model parameters to the server.

Chapter 6

Results and Evaluation

6.1 Summary of Aggregate Test Results

Models	Test RMSE	Test MAE
External Benchmark Models		
Random Forest [53]	**12.01**	not available
Deep LSTM [55]	18.43	not available
Hybrid SVR/LSTM [56]	19.11	not available
Hybrid CNN/LSTM [62]	13.85	**9.44**
Fed. MLP (FedAvg/FedProx) [22]	20.90 - 23.92[1]	16.41 - 18.70[1]
Selected Centralised Models		
Kaplan-Meier	31.86	19.74
Cox PH	36.95	30.82
Random Forest	**16.63**	**12.24**
Neural Network	20.80	15.42
Selected Federated Model - 3 workers		
Fed. GBDT (imbalanced data split)	23.36	18.07
Fed. GBDT (balanced data split)	19.15	13.81
Fed. NN (imbalanced data split)	**17.37**	**13.21**
Fed. NN (balanced data split)	20.74	14.82
Selected Federated Model - 5 workers		
Fed. GBDT (imbalanced data split)	25.32	19.89
Fed. GBDT (balanced data split)	**17.58**	**13.03**
Fed. NN (imbalanced data split)	19.28	14.88
Fed. NN (balanced data split)	20.47	14.61

Table 6.1: Summary of model results on FD001 test data (except for Fed. MLP)

[1]Results from this model are based on FD004 data set which contains engines ran under different operating conditions. The range of result is due to different number of workers used.

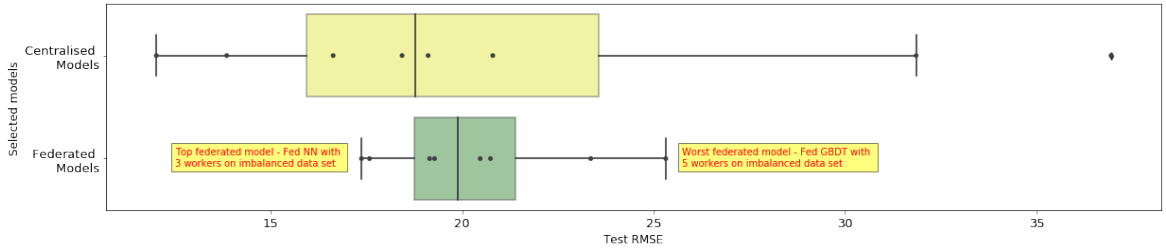

Figure 6.1: Comparison of FD001 test RMSE

Finding 1 Table 6.1 shows the RMSE performance of external benchmark models and selected centralised/federated models that were run as part of this report. This measures the difference between the predicted RUL and true RUL for each engine.

All results in Table 6.1 were computed based on FD001 test data. The only exception to this is Fed. MLP which was performed on FD004 but included in the table as it is the only federated learning literature on this data set.

The simpler selected centralised machine learning models performed within range of the external benchmark models with an RMSE of between 16 and 23. However, the statistical based survival models performed poorer with a RMSE exceeding 30.

In general, federated models performed in line with centralised models as shown in Figure 6.1. From Table 6.1, we can also see that Federated GBDT performed better than federated NN on the balanced data set while the contrary was true for the imbalanced data set in both the three and five workers scenarios.

Overall, the results suggest that the federated GBDT model is more suited when the data set is balanced and more spread out as it is the best performing model in the five workers scenario while the federated NN model is more suited when the data set is imbalanced and concentrated. Additional discussion can be found in Chapter 6.2.2.1 (Finding 4) and Chapter 6.2.2.2 (Finding 5).

6.2 Analysis of Results

Here, we would review the results from both the centralised and federated models, making comparisons to external benchmark models where applicable.

6.2.1 Centralised Models

6.2.1.1 Statistical Models

As noted in Chapters 5.2.1 and 5.2.2, both the Kaplan-Meier and Cox PH models required additional assumptions in order to predict RUL, which contributed to a poorer test RMSE performance compared to the machine learning models.

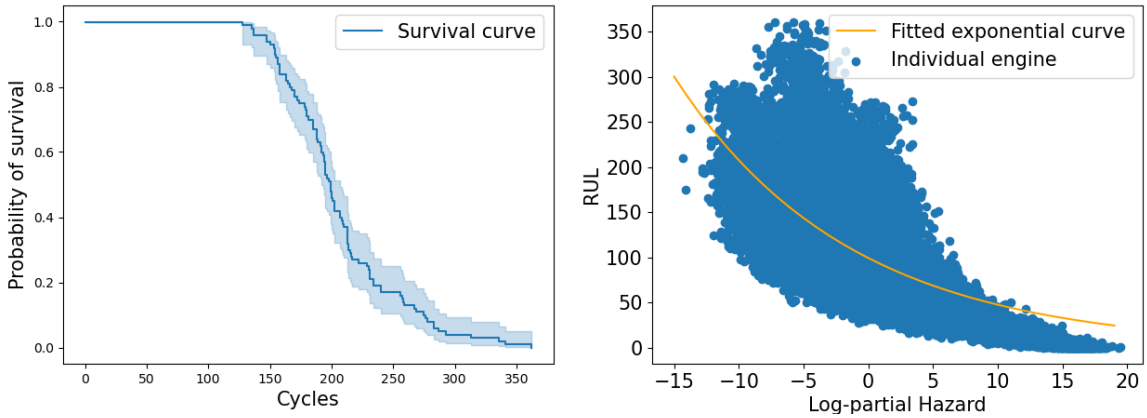

Figure 6.2: Survival curve from Kaplan-Meier model (left) and log-partial hazard plot for Cox PH model (right)

For the **Kaplan-Meier** model, we were able to obtain a survival probability curve shown in Figure 6.2 that is representative of the entire training data set. Although this provides a high level view of the probability of survival for an engine at each cycle, it does not generalise well to predict RUL.

Different engines in the data set have different variations of this survival curve. The RMST estimated from this curve is at the 206th cycle, suggesting that engines in this population would fail at this cycle on average. Using a single average failure cycle to estimate RUL would clearly lead to equally under and over prediction of actual RUL.

Despite the limitations however, the test RMSE of the Kaplan-Meier model is 31.86. This would serve as a good average benchmark for comparison going forward.

Similarly for the **Cox PH** model, we were able to obtain a prediction of the log-partial hazard for each engine, which can be used as a relative indication of impending failure [58]. Engines with a higher actual RUL have a relatively higher log-partial hazard as shown in Figure 6.2.

However, it was necessary to model the relationship between log-partial hazard and RUL to enable prediction of RUL. This was performed by fitting an exponential curve to the relationship in Figure 6.2 with a low R-squared value of -0.04. Based on predictions from the fitted curve, we obtained a test RMSE of 36.95 which was worse than the Kaplan-Meier model.

Finding 2 Both these results showed that while it is technically possible to estimate RUL using statistical models, the strength of the model is very much dependent on the additional layer of assumption to arrive at useful RUL prediction.

Generally, these two models are applied to subjects belonging to the same study group and hence exhibit similar survival deterioration with time, usually in a medical setting. Statistical differences in survival time are often compared between groups using the log-rank test while TTE for prediction via RMST is usually reserved as an intuitive metric to communicate

remaining survival time [72].

Hence, statistical methods - including Random Survival Forest - remains limited in the context of predicting RUL for turbofan engines. This is also the primary reason why federated survival models were not explored further in this report.

6.2.1.2 Machine Learning Models

Both the centralised **Random Forest** and **Neural Network** models obtained average performance when compared with external benchmark models. The performance of these models is largely attributed to the feature engineering performed. These centralised models obtained a test RMSE of 27.80 and 24.77 respectively for RF and NN without feature engineering performed. Hence, the simplicity and acceptable test RMSE performance of these models suggests a potential application on a federated basis.

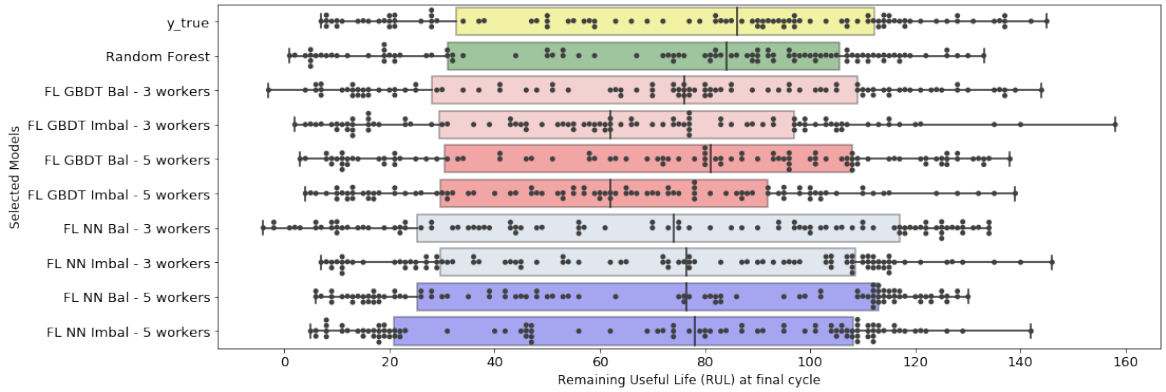

Figure 6.3: Comparison of RUL prediction on FD001 test set. "Bal" refers to the use of a balanced data set while "Imbal" refers to the used of an imbalanced data set. Each model is trained with based on a combination of balanced/imbalanced and 3/5 workers.

6.2.2 Federated Learning Models

6.2.2.1 Federated GBDT Model

The federated GBDT model test RMSE performance of 23.36 on the imbalanced data set is in line with the top range of results from the set of highly optimised benchmark models of 23.92. The construction of the balanced and imbalanced dataset is explained in Chapter 3.4.

Unsurprisingly, this result is primarily driven by the worker with the largest data set which has a test RMSE of 26.58. The corresponding combined train RMSE on the imbalanced data set is 10.83. The distribution of RUL prediction is largely in line with actual RUL as shown in Figure 6.3.

When trained on the balanced data set, the federated GBDT model improved its performance with a test RMSE of 19.15 from 23.36. This result was achieved on the same model

hyperparameters obtained from the imbalanced data set. This suggests that a balanced data set offers better prediction performance than an imbalanced data set for the GBDT model.

Feature importance

Finding 3 In terms of feature importance, the trend feature for sensor 4 and 17 ranks among the top three under both the gain and split measure as shown in Figure 6.4. As shown in Table 3.1, sensor 4 measures total temperature at the LPT outlet while sensor 17 measures the bleed enthalpy.

The split measure counts the number of instances a specific node utilises a feature as a criteria while the gain measure is the cumulative decrease in a node's impurity using information gain criteria when splitting using a specific feature.

Performance on balanced data set

Finding 4 We then repeated our experiment with 5 individual workers in the federated environment. Again, the GBDT model obtained the better test RMSE performance of 17.58 on the balanced dataset and also outperforms the model in the 3 workers setup. This suggests that the GBDT model is not only suited on balanced dataset, but also when there is a higher number of workers contributing to the tree building process.

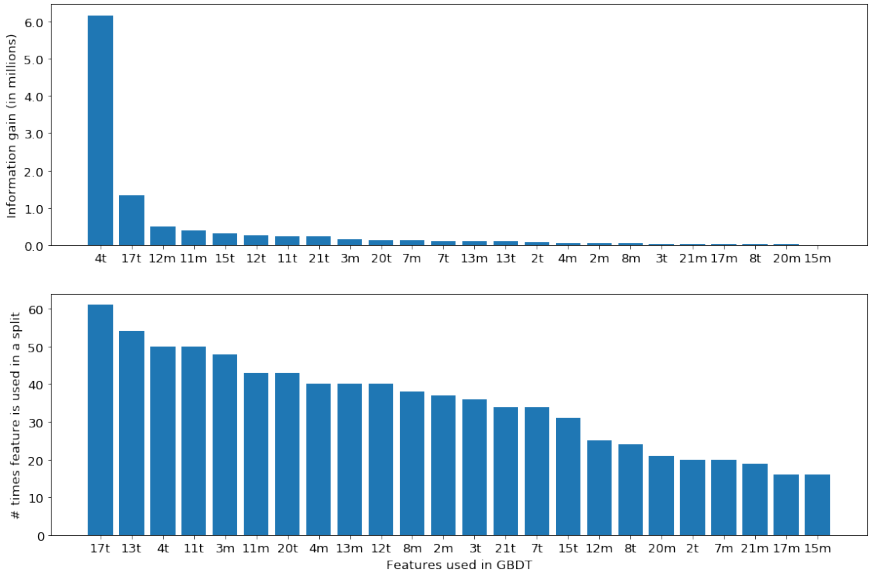

Figure 6.4: Feature importance by information gain (top) and split (bottom). 4t refers to sensor 4's trended reading while 2m refers to sensor 2's mean sensor reading.

6.2.2.2 Federated Neural Network Model

The federated NN model trained on the imbalanced data set obtained a test RMSE of 17.37 which is at the higher end of results from the set of highly optimised benchmark

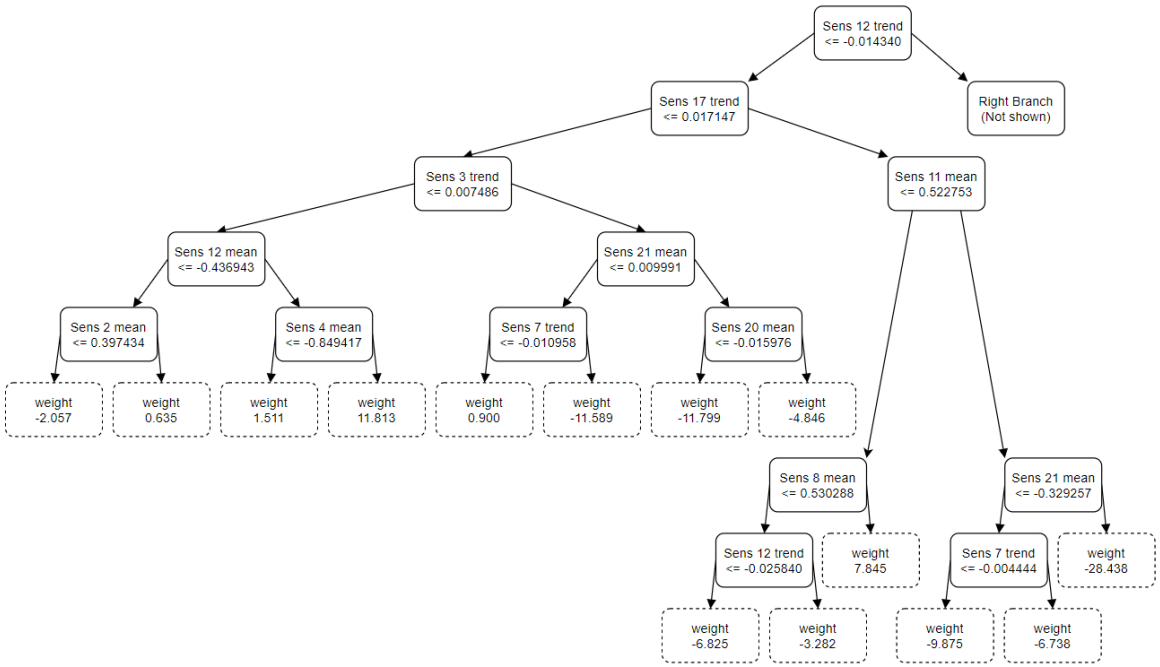

Figure 6.5: 21st decision tree (left branch only) trained on imbalanced data set

models. Similar to GBDT, this result is primarily driven by the worker with the largest data set which has a test RMSE of 20.03.

Test RMSE for the federated NN model trained on a balanced data split is 20.74. This is generally the average test RMSE across all three workers. This performance puts it on par with the results obtained by Rosero et al. [22], also using FedAvg aggregation algorithm, albeit on FD004.

Poorer performance of imbalanced data set

In their analysis, Rosero et al. explored the impact of different number of workers (2/4/8) while using a balanced data set across all workers. However, it was inconclusive in determining how an imbalanced data set distribution would affect their results.

Finding 5 Here, we show that an imbalanced or a single concentrated data set actually improves the performance of the federated NN model using the FedAvg algorithm. This is also inline with the findings made by McMahan et al. [43] that models trained on imbalanced data sets learn faster. They hypothesizes that this is due certain dominant data sets increasing the value of each round of local training.

As discussed in Chapter 2.2.3, the FedAvg algorithm aggregates and derives a global model weight by taking the average of the local model weights. We postulate that some information is lost during the averaging process, leading to worse model performance for the balanced data set.

On the other hand, the imbalanced data set is able to train a better model since the

averaging process is weighted by the number of data points in each worker. Hence, the dominant local model's weight dictates the global model as well. This is evident in both Figure 6.6 and Figure 6.7 where the RMSE progression through each aggregation round for the imbalanced model aligns with Worker C which is the dominant worker.

However, with the number of workers increased from three to five, the FedAvg algorithm performed poorer with a test RMSE of 19.28 versus 17.37. This further confirms that FedAvg's model performance deteriorates as the concentration of data in a single worker gets diluted as was the case in the balanced data set.

Number of aggregation rounds required

The train RMSE for all workers trended downwards in the first aggregation round between the local and global models, as shown in Figure 6.6. For subsequent aggregation rounds, the train RMSE eventually converges near the minimum RMSE by the fourth aggregation rounds for all three workers.

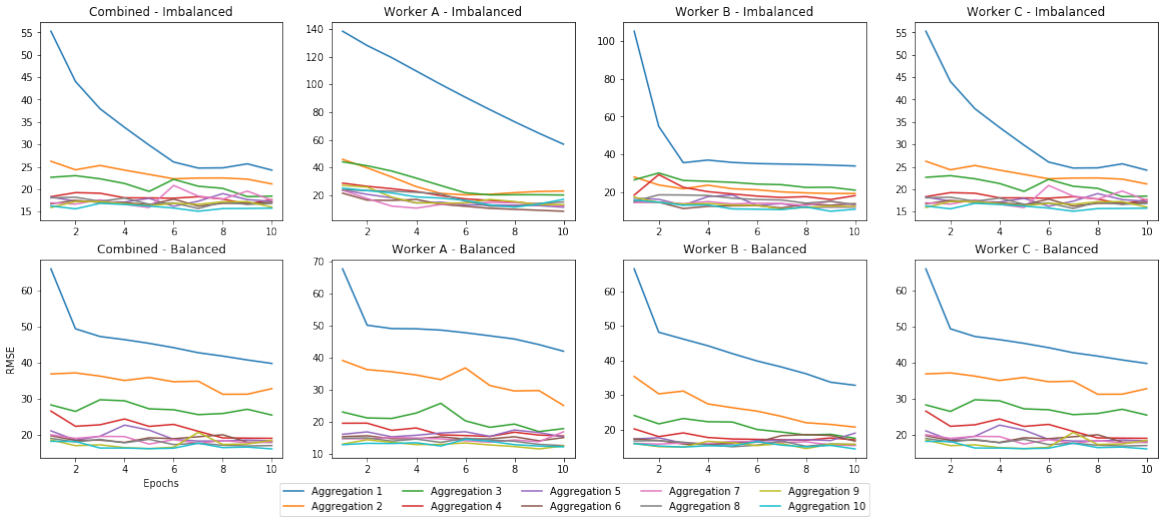

Figure 6.6: Progression of FD001 train RMSE for each aggregation round

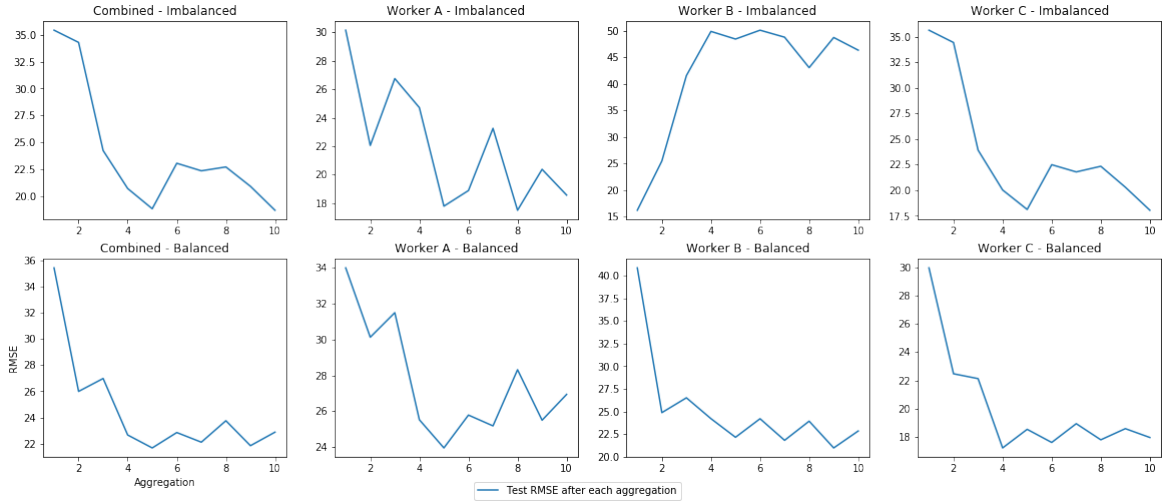

Figure 6.7: Progression of FD001 test RMSE at each aggregation round

Finding 6 This suggests that four aggregation rounds was sufficient to train the model. It is also interesting to note that the imbalanced data set sufficiently converged by the second aggregation round while the imbalanced data set required at least four rounds.

This was expected as 85% of engines is located in one worker in the imbalanced data set. This concurs with the observation in [43] where the time taken for their neural network model to converge using the FedAvg algorithm on the imbalanced data is approximately 7 times faster than for the balanced data.

The development of FD001 test RMSE in Figure 6.7 shows a similar downward trend as the local models are aggregated. The combined trend for the imbalanced data set closely resembles Worker C which has the largest number of engines. Again, we observe that the test RMSE converges to a minimum by the fourth aggregation round.

6.3 Individual Engine's Results

Figure 6.8 shows a comparison of each model's RUL prediction against the actual RUL for selected engines. Each row in this figure represents four selected engines from workers A, B and C respectively. These engines are selected as they have also been individually featured in [56], [62] and [55], which allows for a meaningful comparison.

The "jagged" nature of the predictions is due to the window approach taken during the feature engineering step as described in Chapter 3.3.4. Each spike in the prediction represents a window of size 20 cycles. Within each window, the RUL decreases by one cycle (one unit of time) from the predicted RUL for that window.

For test engines 31, 91 and 82, the predicted RUL of all models aligns closely to the true RUL. A commonality among these engines is their longevity which allowed sufficient cycles for a defined degradation trend to surface. This led to better RUL predictions as we see

from Figure 3.3 that degradation trends are more obvious nearing the end of life.

For test engines 2, 55, 78 and 25 which has an actual RUL of more than 100 cycles at the last data point, the models' prediction tend to diverge wildly. This is reasonable, given that no discernible degradation trend would have emerged at that point. This also validates our assumption to clip RUL at the threshold of 150 cycles, which explains the horizontal line in the Figure 6.8.

	Root Mean Squared Error				Mean Absolute Error			
Models	Engine 31	Engine 82	Engine 2	Engine 78	Engine 31	Engine 82	Engine 2	Engine 78
Selected Centralised Models								
Kaplan-Meier	**1.86**	30.03	31.36	16.14	1.57	26.91	28.00	**11.29**
Cox PH	27.60	36.31	17.35	75.59	23.98	32.79	**14.30**	71.25
Random Forest	5.95	9.93	30.23	33.43	4.33	7.90	28.76	32.58
Neural Network	7.47	14.44	28.15	27.71	9.44	13.25	28.04	27.31
Selected Federated Models - 3 workers								
Fed. GBDT (imbalanced data split)	8.08	**6.83**	33.87	23.48	6.12	**6.10**	26.86	18.75
Fed. GBDT (balanced data split)	9.59	14.00	**16.76**	15.37	7.94	9.01	15.29	13.56
Fed. NN (imbalanced data split)	3.44	15.37	40.43	**12.82**	**2.56**	12.07	33.18	11.64
Fed. NN (balanced data split)	10.69	25.04	41.93	16.90	9.79	18.69	40.39	15.36

Table 6.2: Summary of selected engine's test results

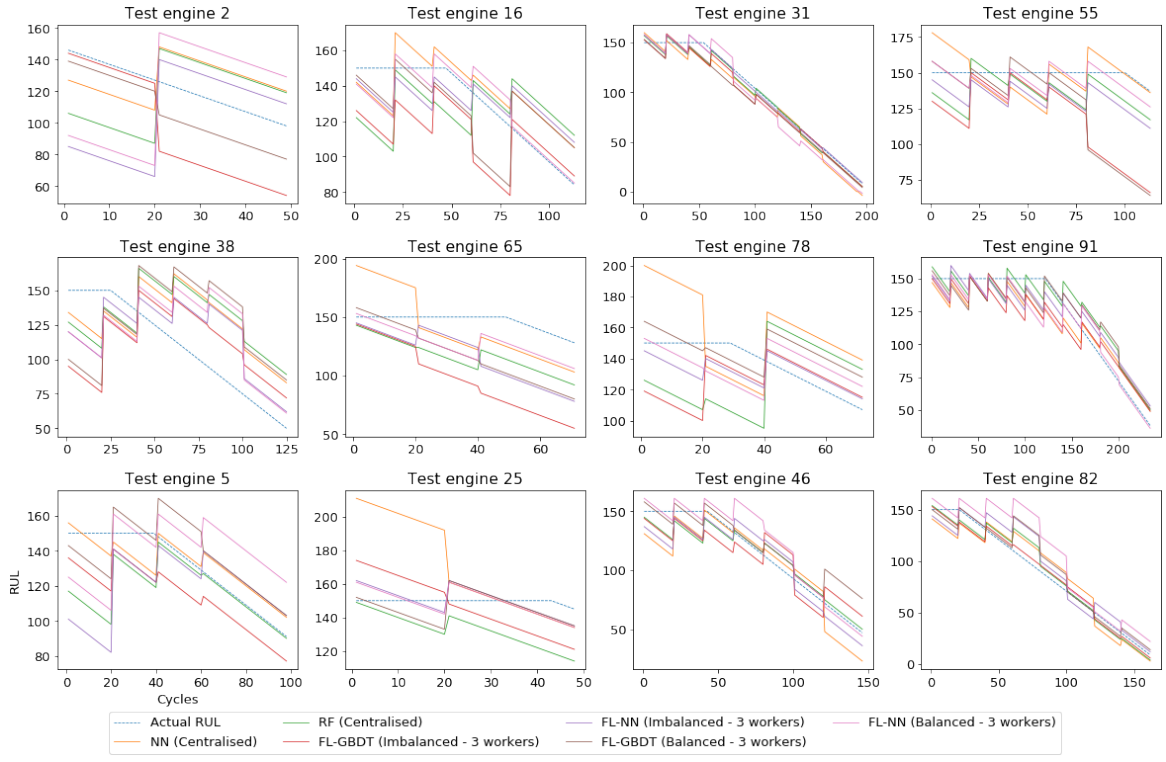

Figure 6.8: Comparison of RUL for selected engines in FD001 test

As shown in Figure 6.8, Engine 31 was fairly well predicted by both centralised and federated models with an average RMSE of 7.95 and 10.72 respectively. The low RMSE from the Kaplan-Meier model suggests that Engine 31 is representative of an average engine's

degradation in FD001 data set. However, Kaplan-Meier's performance for other engines and at an overall basis suggest that this average does not generalise well.

6.3.1 Federated Models

The development of RMSE through each federated aggregation round for individual engines in Figure 6.9 also closely follow Figure 6.6, where we can observe convergence by the fourth aggregation round. Similarly, the balanced data set required one to two more aggregation rounds before convergence.

Relative performance between balanced and imbalanced data sets is also determined by the second aggregation round. We can see that where the imbalanced model performed better than the balanced model in the second aggregation round, it remains the case throughout the remaining federated learning process. This suggests that increased aggregation frequency cannot compensate for poor model performance due to imbalanced/balanced data set.

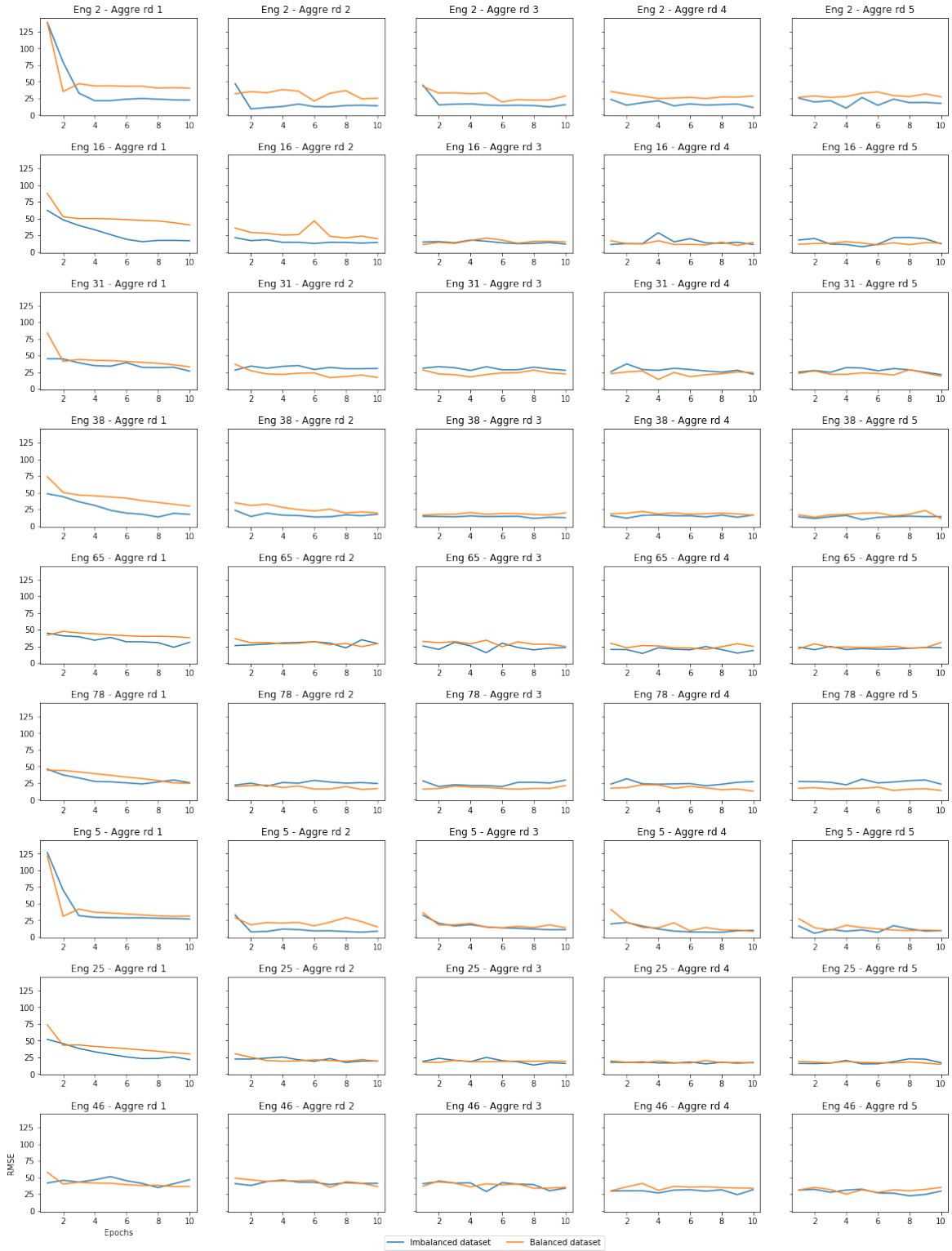

Figure 6.9: Convergence of RUL for selected engines in FD001 test set

Chapter 7

Conclusions and Next Steps

7.1 Conclusion

In this report, we have shown that a federated GBDT and NN model can perform as well as a centralised model in predicting RUL for turbofan engines. This is despite the privacy preserving nature of federated learning algorithm which meant that the global model would need to be trained on segregated data sets.

This result aligns with findings made by Rosero et al. [22] that acceptable RUL prediction performance can indeed be achieved in a federated environment, albeit on a slightly different data set. Furthermore, we came to this conclusion by applying the GBDT model in addition to using purely NN-based approaches and the FedAvg algorithm.

This was made possible by open-source federated learning packages such as FATE which we evaluated and demonstrated on the turbofan dataset. We also showcased the federated learning experimental setup using these packages in Python which would allow replication of our published results. **This was an area that was not readily apparent in other published research which focused purely on the theoretical setup leading to their findings.**

We also discussed key limitations of FATE such as the lack of support for regression task in neural network models and lack of documentation in certain areas. Despite these limitations, we showed that FATE is currently the only viable non-NN based federated learning approach that is accessible and deployable with satisfactory performance.

Finally, we showed that an imbalanced data set required additional aggregation rounds between each worker compared to a balanced data set before convergence when using federated NN. The imbalanced federated NN model trained on an imbalanced split of engine data performed better than the federated GBDT model trained on a balanced split. This suggests that the federated GBDT model is more suited for federated process where the data is well distributed between multiple individual workers.

7.2 Next Steps

Actual deployment

The results and findings discussed in Chapter 6 are based on the experimental setup showcased in Chapter 4 which describes a simulated federated environment without actual deployment of IIoTs onto individual worker devices. A key limitation of this report is therefore the inability to deploy federated learning models in a production environment.

Deployment in a real-life production environment was only technically possible by collaborating with an industrial partner who have access to hundreds if not thousands of individual workers attached to IIoT sensors. Hence, a logical next step would be to productionize the setup used in this report. Specifically, both FATE and dc_federated are able to support large scale development of federated learning environment although the exact implementation details to scale up are beyond the scope of this report.

Hyperparameters in the Federated Process

As discussed in Chapter 5.3.1, hyperparameters in the federated environment control the overall federated learning process. Although Rosero et al. [22] and this report have explored the impact on prediction performance of varying the number of workers and of using a balanced or imbalanced data sets, these were investigated while maintaining a balanced data set or the number of workers constant.

We therefore propose for future research to investigate the concurrent impact of varying both the number of workers and using a balanced/imbalanced data set to model performance and subsequently propose mitigating alterations to the federated learning model in a production environment.

An ideal outcome of this research is to propose a standard measure of the impact of a balanced and imbalanced data set on the performance of a federated model. Different federated models are impacted by this hyperparameter in different manners as we have shown in this report. Ideally, this measure should inform users of federated models on the ideal distribution of data between workers for specific federated models and quantify the impact on model performance due to any deviation from the parameterized distribution.

Appendices

Appendix A

Dendrogram and trend clusters of 5 workers imbalanced split

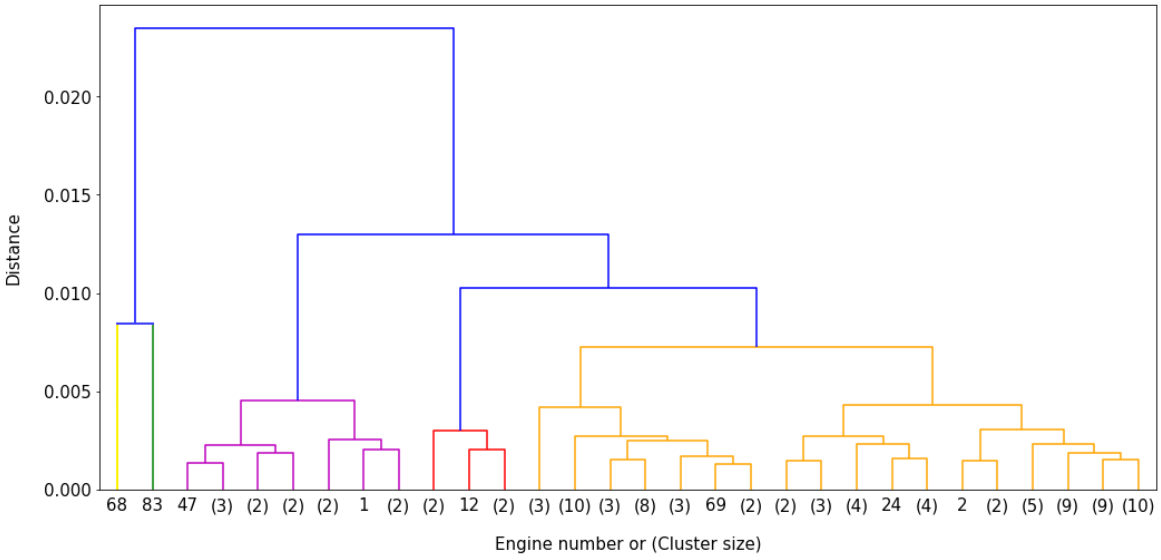

Figure A.1: Dendrogram of 5 allocated clusters for 100 engines in FD001 (root colour (yellow, green, purple, red, orange) represents each of the 5 clusters)

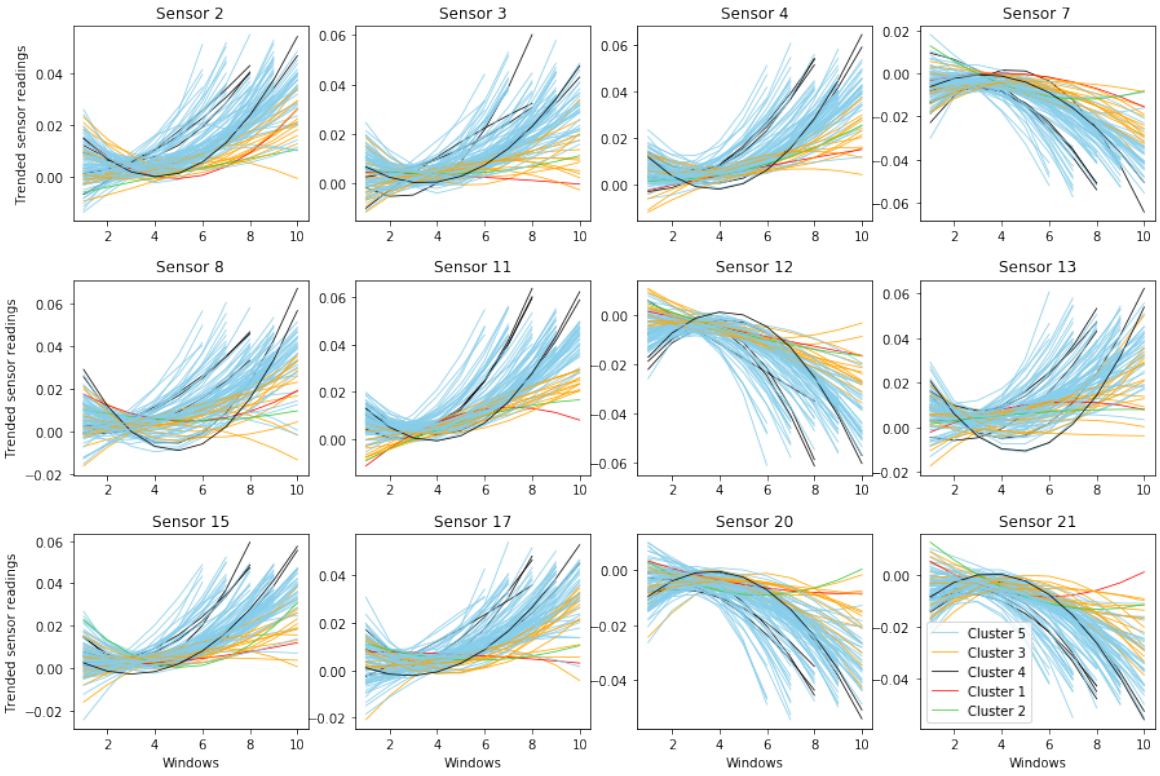

Figure A.2: 5 allocated trend clusters for each of the 100 engines in FD001

Bibliography

[1] Zonta T, da Costa CA, da Rosa Righi R, de Lima MJ, da Trindade ES, Li GP. Predictive maintenance in the Industry 4.0: A systematic literature review. Computers & Industrial Engineering. 2020;150:106889.

[2] Kammerer K, Hoppenstedt B, Pryss R, Stökler S, Allgaier J, Reichert M. Anomaly Detections for Manufacturing Systems Based on Sensor Data—Insights into Two Challenging Real-World Production Settings. MDPI. 2019. Available from: https://res.mdpi.com/sensors/sensors-19-05370/article_deploy/sensors-19-05370.pdf, visited on 2021-05-20.

[3] Emiliano S, Abusayeed S, Song H, Ulf J, Mikael G. Industrial Internet of Things: Challenges, Opportunities, and Directions. IEEE Transactions on Industrial Informatics. 2018;14 (11):4724–4734. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3932959/, visited on 2021-05-21.

[4] Niyonambaza I, Zennaro M, Uwitonze A. Predictive Maintenance (PdM) Structure Using Internet of Things (IoT) for Mechanical Equipment Used into Hospitals in Rwanda. MDPI. 2020. Available from: https://doi.org/10.3390/fi12120224.

[5] Zeki C, Abubakar N, Qasim Z, Orhan K, Mohammed A, Babak S. Machine Learning in Predictive Maintenance towards Sustainable Smart Manufacturing in Industry 4.0. Sustainability. 2020 10;12:8211. Available from: https://doi.org/10.3390/su12198211.

[6] Sailendu B, R SG. Design and development of a wind turbine test rig for condition monitoring studies. 2015 International Conference on Industrial Instrumentation and Control (ICIC). 2015:891–896. Available from: https://doi.org/10.1109/IIC.2015.7150869.

[7] Timo H, Alexander J. Predictive Maintenance of Photovoltaic Panels via Deep Learning. 2018 IEEE Data Science Workshop (DSW). 2018:66–70.

[8] Nguyen KTP, Medjaher K. A new dynamic predictive maintenance framework using deep learning for failure prognostics. Reliability Engineering & System Safety. 2019;188:251–262.

[9] Hard A, Rao K, Mathews R, Ramaswamy S, Beaufays F, Augenstein S, et al.. Federated Learning for Mobile Keyboard Prediction; 2019. Available from: https://arxiv.org/abs/1811.03604.

[10] of China CA. Cybersecurity Law of the People's Republic of China; 2016. Available from: http://www.cac.gov.cn/2016-11/07/c_1119867116.htm.

[11] Commission E. REGULATION (EU) 2016/679 General Data Protection Regulation; 2016. Available from: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

[12] of Electronics M, Technology I. Personal Data Protection Bill; 2018. Available from: https://www.meity.gov.in/writereaddata/files/Personal_Data_Protection_Bill,2018.pdf.

[13] Konečný J, McMahan HB, Ramage D, Richtárik P. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. CoRR. 2016;abs/1610.02527. Available from: https://arxiv.org/abs/1610.02527, visited on 2021-05-21.

[14] Jeromemetronome. Federated learning process central case; 2019. Online; accessed August 28, 2021. Available from: https://commons.wikimedia.org/wiki/File:Federated_learning_process_central_case.png.

[15] da Silveira Dib MA, Ribeiro B, Prates P. Federated Learning as a Privacy-Providing Machine Learning for Defect Predictions in Smart Manufacturing. ASTM International. 2021. Available from: https://doi.org/10.1520/SSMS20200029, visited on 2021-05-21.

[16] Ge N, Li G, Zhang L, Liu Y. Failure Prediction in Production Line Based on Federated Learning: An Empirical Study. Journal of Intelligent Manufacturing. 2021. Available from: https://doi.org/10.1007/s10845-021-01775-2.

[17] Lee J. Machine performance monitoring and proactive maintenance in computer-integrated manufacturing: review and perspective. International Journal of Computer Integrated Manufacturing. 1995. Available from: https://www.tandfonline.com/doi/abs/10.1080/09511929508944664, visited on 2021-05-21.

[18] Wang P, Li Y, Reddy CK. Machine Learning for Survival Analysis: A Survey. ACM Computing Surveys. 2019. Available from: https://doi.org/10.1145/3214306, visited on 2021-05-21.

[19] Gujre VS, Anandb R. Machine learning algorithms for failure prediction and yield improvement during electric resistance welded tube manufacturing. Journal of Experimental and Theoretical Artificial Intelligence. 2019. Available from: https://doi.org/10.1080/0952813X.2019.1653995, visited on 2021-05-21.

[20] Wu D, Jennings C, Terpenny J, Gao RX, Kumara S. A Comparative Study on Machine Learning Algorithms for Smart Manufacturing: Tool Wear Prediction Using Random Forests. Journal of Manufacturing Science and Engineering. 2017.

[21] Saxena A, Goebel K. Turbofan Engine Degradation Simulation Data Set;. Available from: https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/#turbofan.

[22] Rosero RL, Silva C, Ribeiro B. Remaining Useful Life Estimation in Aircraft Components with Federated Learning. Prognostics and Health Management Conference 2020. 2020;5(1). Available from: https://papers.phmsociety.org/index.php/phme/article/view/1228.

[23] WeBank AI Department. Federated AI Technology Enabler. GitHub; 2019. Available from: https://github.com/FederatedAI/FATE.

[24] Catapult D. dc-federated. GitHub; 2020. Available from: https://github.com/digicatapult/dc-federated.

[25] L KE, Meier P. Non-parametric estimation from incomplete observations. American Statistical Association. 1958. Available from: http://www.jstor.org/stable/2281868, visited on 2021-05-21.

[26] Jager KJ, van Dijk PC, Zoccali C, Dekker FW. The analysis of survival data: the Kaplan–Meier method. Elsevier. 2008. Available from: https://doi.org/10.1038/ki.2008.217, visited on 2021-05-21.

[27] Rich JT, Neely JG, Paniello RC, Voelker CCJ, Phil D, Nussenbaum B, et al. A practical guide to understanding Kaplan-Meier curves. PubMed Central. 2010. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3932959/, visited on 2021-05-21.

[28] Bencomo T. Kaplan Meier Mistakes; 2019. Available from: https://towardsdatascience.com/kaplan-meier-mistakes-48cd9e168b09.

[29] Cox DR. Regression Models and Life-Tables. London Royal Statistical Society. 1972;34 (2):187–220. Available from: https://www.jstor.org/stable/2985181.

[30] Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. The Annals of Applied Statistics. 2008;2(3). Available from: http://dx.doi.org/10.1214/08-AOAS169.

[31] Breiman L. Random Forests. Machine Learning. 2001;45:5–32. Available from: https://doi.org/10.1023/A:1010933404324.

[32] L B, Friedman RA J H Olshen, J SC. Classification and regression trees; 1984.

[33] Laura R, Kilian S. Theoretical Comparison between the Gini Index and Information Gain Criteria. Annals of Mathematics and Artificial Intelligence. 2004 05;41:77–93. Available from: https://www.doi.org/10.1023/B:AMAI.0000018580.96245.c6.

[34] Wei-Yin L. Classification and Regression Trees. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. 2011 01;1:14 – 23.

[35] Jehad A, Rehanullah K, Nasir A, Imran M. Random Forests and Decision Trees. International Journal of Computer Science Issues(IJCSI). 2012 09;9. Available from: https://www.researchgate.net/publication/259235118_Random_Forests_and_Decision_Trees.

[36] Floriana E, Donato M, Giovanni S, John K. A Comparative Analysis of Methods for Pruning Decision Trees. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 1997 06;19:476 – 491. Available from: https://www.doi.org/10.1109/34.589207.

[37] Breiman L. Bagging predictors. Machine Learning. 1996;24:123–140. Available from: https://doi.org/10.1007/BF00058655.

[38] LeBlanc M, Crowley J. Survival Trees by Goodness of Split. Journal of the American Statistical Association. 1993;88 (422):457–467. Available from: https://doi.org/10.2307/2290325.

[39] Jie H, Youngsoon K, Tejaswini M, Hun OJ, Mingon K. Interpretable deep neural network for cancer survival analysis by integrating genomic and clinical data. BMC medical genomics. 2019;12 (10):189–189. Available from: https://doi.org/10.1186/s12920-019-0624-2.

[40] Bora L, Hoon CS, Hyung HJ, Sook WI, Seoree K, Won JJ, et al. DeepBTS: Prediction of Recurrence-free Survival of Non-small Cell Lung Cancer Using a Time-binned Deep Neural Network. Scientific Reports. 2020;10 (11):1952–1952. Available from: https://doi.org/10.1038/s41598-020-58722-z.

[41] B B, T VG, M S, den Poel D V, J V. Neural network survival analysis for personal loan data. The Journal of the Operational Research Society. 2005;56 (9):1089–1098. Available from: https://doi.org/10.1057/palgrave.jors.2601990.

[42] Faraggi D, Simon R. A Neural Network Model for Survival Data. Statistics in Medicine. 1995;14:73–82. Available from: https://doi.org/10.1002/sim.4780140108.

[43] McMahan HB, Moore E, Ramage D, y Arcas BA. Federated Learning of Deep Networks using Model Averaging. CoRR. 2016;abs/1602.05629. Available from: http://arxiv.org/abs/1602.05629.

[44] Cheng K, Fan T, Jin Y, Liu Y, Chen T, Papadopoulos D, et al. SecureBoost: A Lossless Federated Learning Framework. arXiv. 2019. Available from: https://arxiv.org/abs/1901.08755.

[45] OpenMined. Understanding the types of federated learning; 2020. Available from: https://blog.openmined.org/federated-learning-types/.

[46] Liu Y, Liu Y, Liu Z, Zhang J, Meng C, Zheng Y. Federated Forest. IEEE Transactions on Big Data. 2019. Available from: https://doi.org/10.1109/TBDATA.2020.2992755.

[47] Jaideep V, Basit S, Fan W, Danish M, David L. A Random Decision Tree Framework for Privacy-Preserving Data Mining. IEEE transactions on dependable and secure computing. 2014;11 (5):399–411. Available from: https://arxiv.org/abs/1905.10053.

[48] Kholod I, Yanaki E, Fomichev D, Shalugin E, Novikova E, Filippov E, et al. Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis. Sensors. 2021;21 (1):167. Available from: https://www.mdpi.com/1424-8220/21/1/167.

[49] Fu F, Jiang J, Shao Y, Cui B. An Experimental Evaluation of Large Scale GBDT Systems. arXiv. 2019. Available from: https://arxiv.org/pdf/1907.01882.pdf.

[50] OpenMined. PySyft. GitHub; 2020. Available from: https://github.com/OpenMined/PySyft.

[51] Saxena A, Goebel K, Simon D, Eklund N. Damage Propogation Modeling for Aircraft Engine Run-to-Failure Simulation. 2008 International Conference on Prognostics and Health Management. 2018. Available from: https://ti.arc.nasa.gov/publications/154/download/.

[52] Frederick DK, DeCastro JA, Litt JS. User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS); 2007. Available from: https://core.ac.uk/download/pdf/10539072.pdf.

[53] Chen X, Jin G, Qiu S, Lu M, Yu D. Direct Remaining Useful Life Estimation Based on Random Forest Regression. Institute of Electrical and Electronics Engineers. 2020. Available from: https://ieeexplore.ieee.org/document/9281004.

[54] Mann HB. Nonparametric Tests Against Trend. The Econometric Society. 1945;13:245–259. Available from: https://doi.org/10.2307/1907187.

[55] Wu J, Hu K, Cheng Y, Zhu H, Shao X, Wang Y. Data-driven remaining useful life prediction via multiple sensor signals and deep long short-term memory neural network. ISA Transactions. 2020;97:241–250. Available from: https://www.sciencedirect.com/science/article/pii/S0019057819302939.

[56] Chuang C, Ningyun L, Bin J, Cunsong W. A Risk-Averse Remaining Useful Life Estimation for Predictive Maintenance. IEEE/CAA Journal of Automatica Sinica. 2021;8(2):412–422. Available from: https://ieeexplore.ieee.org/document/9317711.

[57] Hussain M, Mahmud I. pyMannKendall: a python package for non parametric Mann Kendall family of trend tests. Journal of Open Source Software. 2019 7;4(39):1556. Available from: http://dx.doi.org/10.21105/joss.01556.

[58] Peters K. Survival analysis for predictive maintenance of turbofan engines;. Available from: https://towardsdatascience.com/survival-analysis-for-predictive-maintenance-of-turbofan-engines-7e2e9b82dc0e.

[59] Amidon A. How to Apply Hierarchical Clustering to Time Series;. Available from: https://towardsdatascience.com/how-to-apply-hierarchical-clustering-to-time-series-a5fe2a7d8447.

[60] Lau M. Predictive Maintenance of Turbofan Engines using Federated Learning with PySyft and PyGrid;. Available from: https://blog.openmined.org/predictive-maintenance-of-turbofan-engines-using-federated-learning/.

[61] OpenMined. PyGrid. GitHub; 2020. Available from: https://github.com/OpenMined/PyGrid.

[62] Chu CH, Lee CJ, Yeh HY. Developing Deep Survival Model for Remaining Useful Life Estimation Based on Convolutional and Long Short-Term Memory Neural Networks. Hindawi. 2020. Available from: https://doi.org/10.1155/2020/8814658.

[63] ai S. Federated Learning and Differential Privacy Framework. GitHub; 2020. Available from: https://github.com/sherpaai/Sherpa.ai-Federated-Learning-Framework.

[64] TensorFlow. TensorFlow Federated. GitHub; 2020. Available from: https://github.com/tensorflow/federated.

[65] Baidu. Paddle Federated Learning. GitHub; 2020. Available from: https://github.com/PaddlePaddle/PaddleFL.

[66] Li Q, Wen Z, Wu Z, Hu S, Wang N, Li Y, et al. A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection. arXiv. Available from: https://arxiv.org/pdf/1907.09693.pdf.

[67] PyTorch. PyTorch. GitHub; 2020. Available from: https://github.com/pytorch/pytorch.

[68] TensorFlow. TensorFlow. GitHub; 2020. Available from: https://github.com/tensorflow/tensorflow.

[69] Bonawitz K, Ivanov V, Kreuter B, Marcedone A, McMahan HB, Patel S, et al. Practical Secure Aggregation for Privacy-Preserving Machine Learning. Association for Computing Machinery. 2017. Available from: https://doi.org/10.1145/3133956.3133982.

[70] FATE. FATE documentation; 2020. Available from: https://fate.readthedocs.io/en/latest/index.html.

[71] Heaton J. Introduction to Neural Networks for Java, 2nd Edition. 2nd ed. Heaton Research, Inc.; 2008.

[72] Perego C, Sbolli M, Specchia C, Fiuzat M, McCaw ZR, Metra M, et al. Utility of Restricted Mean Survival Time Analysis for Heart Failure Clinical Trial Evaluation and Interpretation. JACC: Heart Failure. 2020;8(12):973–983. Available from: https://doi.org/10.1016/j.jchf.2020.07.005.