CHAPTER 5

A Novel Soft Computing Method for Engine RUL Prediction

5.1 Introduction

Prognostics is an engineering discipline focused on predicting the Remaining Useful Life (RUL) of a system or a component using raw multimedia(sensor) data. This Chapter presents a novel machine learning model for this task, which includes a smart ensemble of gradient boosted trees (GBT) and feed-forward neural networks.

5.1.1 Problem Overview

Prognostics aims at improving health management of systems through warnings about performance degradation and impending failures. It can be classified into two categories: model-based and data-driven. Physics-based prognostics relies on system-specific knowledge. Data-driven prognostics uses pattern recognition and machine learning techniques to detect changes in system states. In data-driven prognostics, the task is to learn a function from training data, mapping degradation patterns of time-series sensor

data to engines life and then use this model to predict the number of cycles a test system could run before failure, given its time-series from the initial state till an arbitrary point. From its introduction in PHM 2008 competition, many papers have used the C-MAPSS datasets. This is because they present global prognostics challenges such as high variability due to sensor noise, multiple operating conditions, and simultaneous fault modes [212]. Besides, there are very few public run-till-failure datasets available for prognostics experimentation, and they are also limited in size and corrupt with noise. This chapter presents a discussion on C-MAPSS data analysis, a few important works, and existing benchmarks. It concludes by presenting evaluations of multiple prediction models, including gradient boosted trees and a novel method of assembling feedforward neural networks. Gradient Boosted Trees are efficient in that they produce an encouraging scoring model with minimum effort and also return feature importance information. MLP gives relatively poor results which are explored to infer a need for an ensemble of models. A method to ensemble feed-forward neural networks are presented which improve the scores.

5.1.2 Data Overview

The data-set is composed of multiple multivariate time series signals generated by a simulation model built on the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) at NASA Ames Research Centre. No system specific information was made available, which makes these datasets suitable for data-driven approaches [231]. C-MAPSS datasets include four datasets which are further divided into training and test subsets. Each subset contains many multivariate time-series sequences corresponding to different engines.

There are three operating conditions and twenty-one sensor measurements. Each training dataset is a run-till-failure dataset, that is, every engine runs till failure, whereas each time-series in a test dataset ends at an arbitrary time before complete degradation. Thus, the objective is, the prediction of the number of remaining operational cycles for each time-series or say, each engine in a test dataset. A total of 26 signals were generated. 21 are the record of the sensory data; 3 others represent the setting of the operating conditions. The remaining represent the engines ID and number of cycles. Each time series represents a different engine from the same complex system. Each engine consists of different elements, such as pressure compressors, turbines, etc. Engines are operating normally at the start and develop a fault at some time before system failure. The dataset is characterized by one failure mode and one operating condition. It comes in three text files: TRAIN FD001 contains 100 training units. The measurement recordings start at a similar degradation level that is considered healthy and stop when failure is reached. TEST FD001 contains 100 test units. The data is incomplete, and the time series end up before failure. The objective is to predict the remaining useful life. RUL FD001 contains the actual RUL values.

5.1.3 Motivation of the Research Work

Data-driven prognostics pose a very practical and important challenge in supervised learning. Assessing the health of systems accurately can assist in finding RUL and learning degradation patterns from time-series data to model systems to increase their efficiency and reduce energy consumptions. Every machine can have a different initial state or number of faults. This makes it a challenging task to explore and develop general prognostics algorithms.

5.2 Problem Settings

In data-driven prognostics, RUL prediction is an important problem which uses time series data collected by multiple sensors as features. If d sensors are used to obtain a time-series data D for a system running for N time cycles, then it can be formally stated as:

$$D \in R^{N*d}$$

So, if a training dataset *X* has time-series sequences for *N* systems, where n_i is the total number of life cycles of a system *i*. Then,

$$X^i \in R^{n_i*d} \text{ for } i \in \{1,\ldots,N\}$$

So, the j^{th} sequence of X^i denoted as

$$X_j^i \in R^d \text{ for } j \in \{1,\ldots,n_i\}$$

is a vector consisting of the signals from the *d* sensors at the $\mathbf{j^{th}}$ time cycle of th**e** $\boldsymbol{i^{th}}$ system. Similarly, a test dataset *Y* has time-series sequences for *M* systems, where m_i is the number of cycles i^{th} system has completed. Then,

$$Y^i \in R^{m_i*d} \text{ for } i \in \{1,\ldots,M\}$$

Also, from training data, true labels that are, RULs can be directly calculated for all time sequence. RUL for the j^{th} sequence of X^i can be written as:

$$RUL_j^i = (n_i - j)$$

The task is to predict RUL of each test system, which is the same as RUL of the last sequence of that system:

$$RUL^i = RUL_{m_i}^i$$

Thus, this is a supervised machine learning problem to infer an approximate function f mapping multivariate time-series signals to continuous predictions of remaining useful life, such that the error between these predictions and given true test RUL values are minimized.

$$\text{Minimise error } (f(Y^i), RUL^i)$$

In these experiments, the Root Mean Square Error (RMSE) of estimated RUL has been used as the metric of a model's accuracy. Figure 5.1 is a diagram of a generic supervised learning

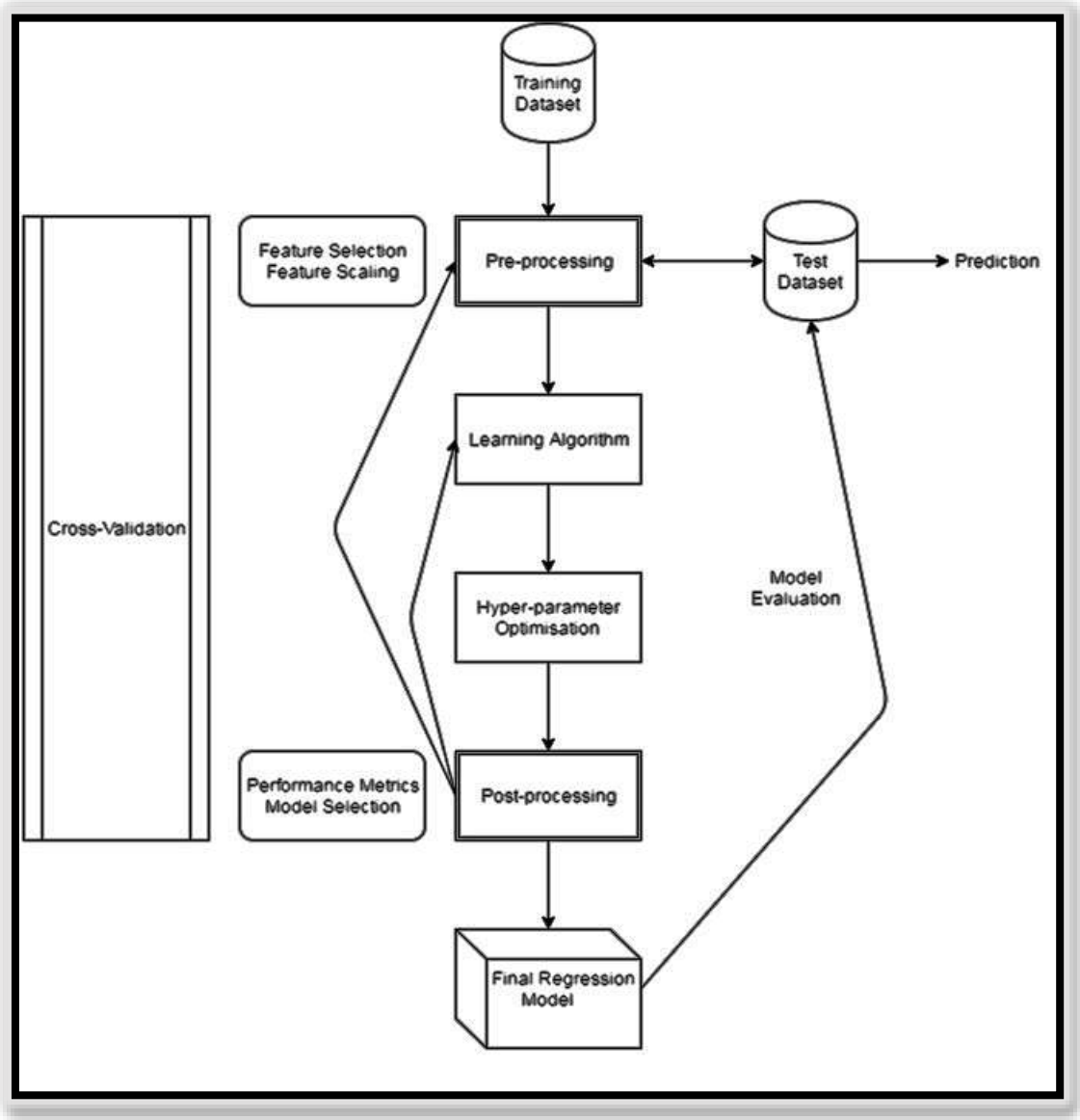

Figure 5.1; Supervised learning model

5.3 Data Exploration

C-MAPSS datasets include four train and test datasets. Each dataset consists of multiple multivariate time series, each time series from a different engine. Each engine starts with varying degrees of initial wear and manufacturing variation which is unknown to the user. There are three operational settings and twenty-one sensor measurements which are contaminated with noise [231]. Time series for a train engine is given till failure, while time-series for a test engine is given until any arbitrary point. The objective is to predict the number of operational cycles left after the last cycle for which the test engine will continue to operate before failure. Table 5.1 summarizes the four datasets.

Table 5.1: Description of C-MAPSS datasets

C-MAPSS Datasets	Fault Conditions	Train Systems	Test Systems
#1	1	100	100
#2	1	260	259
#3	2	100	100
#4	2	249	248

5.3.1 Data Visualization

In each training dataset, systems run for a variable number of cycles before failure. Table 5.2 is a summary of statistics about the distribution of run lengths for each dataset.

Table 5.2: Distribution of run lengths in training datasets

Train Dataset	Minimum	Maximum	Mean	Standard Deviation
#1	128	362	206.31	46.11
#2	128	378	206.77	46.69
#3	145	525	247.20	86.05
#4	128	543	245.98	72.96

Figure 5.32 shows that, for test dataset 4, approximately only 20 percent of the RUL values are greater than 140. Also, Figure 5.2 indicates that the median of these values is around 85. It is interesting to note that the linear regression approach is no better than the best guess of a constant value as RUL for all the test systems, except for test dataset-1.

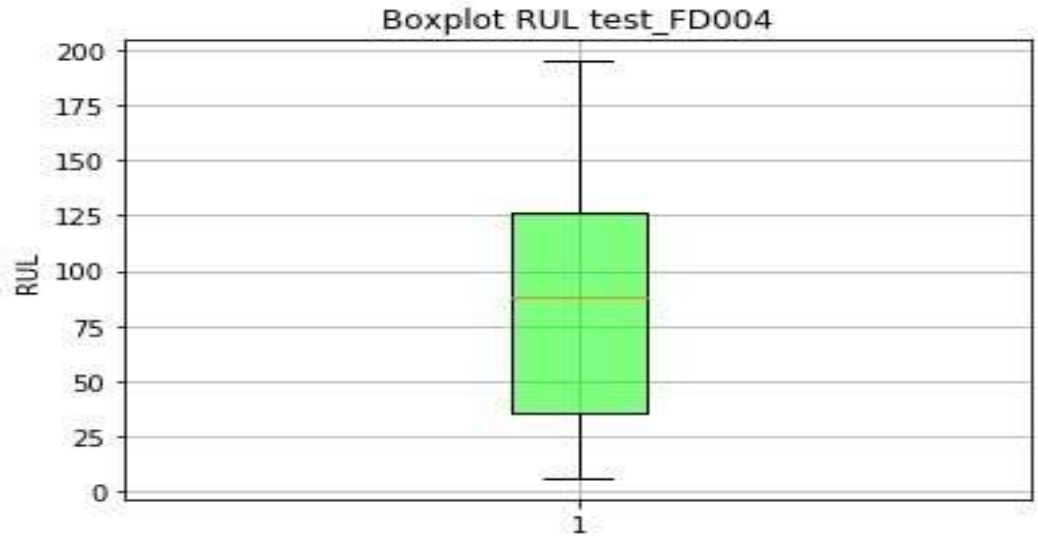

Figure 5.2: Boxplot of RULs for test dataset 4

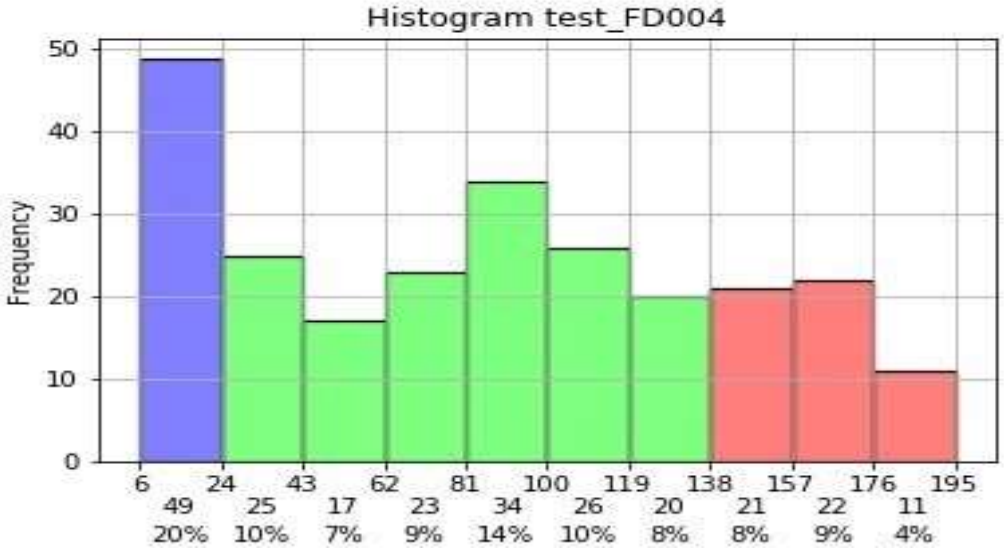

Figure 5.3: Histogram of RULs for test dataset 4

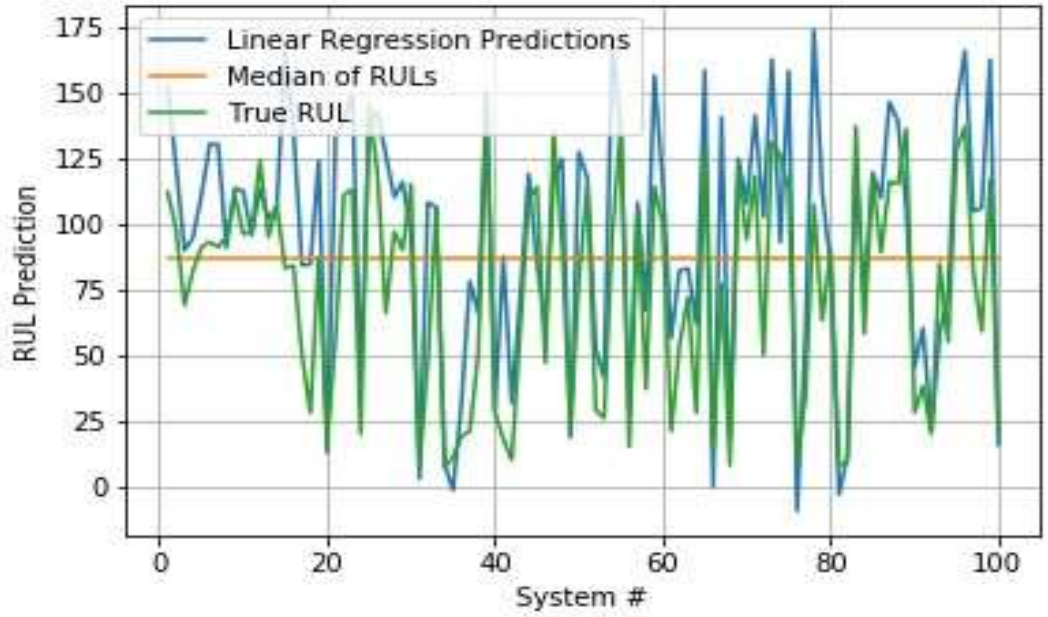

Figure 5.4: True RUL vs. median RUL vs. linear model prediction of dataset-1

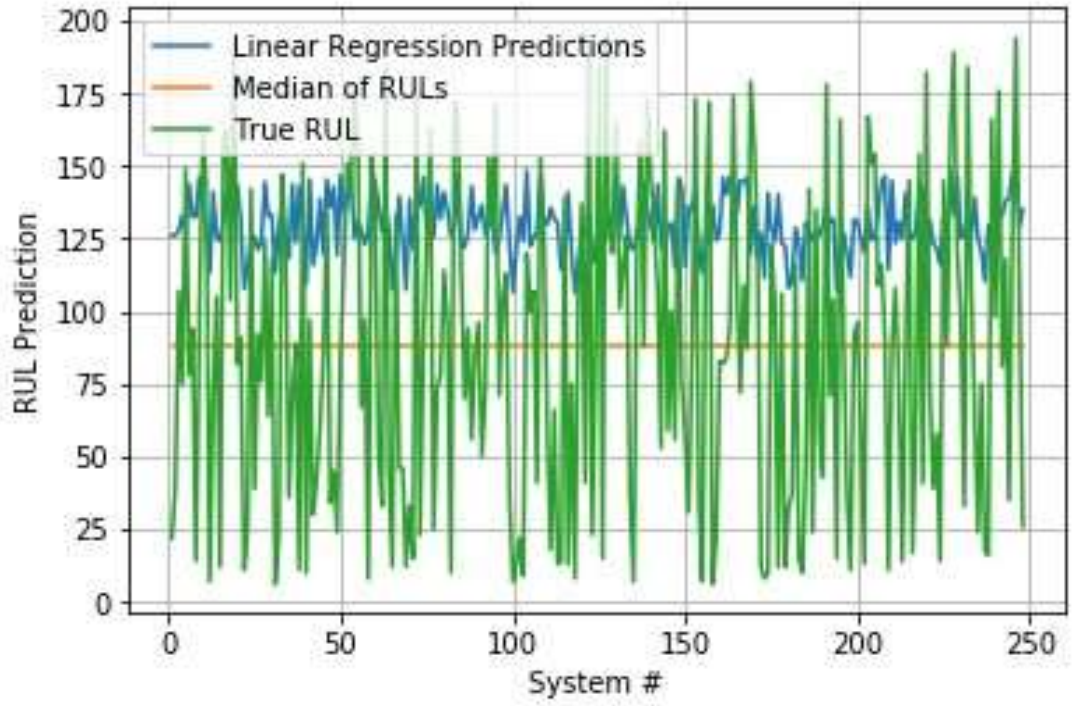

Figure 5.5: True RUL vs. median RUL vs. linear model prediction for dataset- 4

Table 5.3: Linear regression is no better than guessing

Test Dataset	Median of True RMSE	Best Guess RUL	Best Guess RMSE	Linear Reg RM
#1	87	76	41.55	31.21
#2	80	81	53.77	55.42
#3	78	75	41.39	56.91
#4	88	87	54.52	67.18

Table 5.3 shows this. Also, this best guess is almost the same as the median of true RULs for that test datasets, as shown in Figure 5.3. This does not hold for dataset 1, which corresponds to its outlier behaviour on the best guess method is better than the linear regression model. From Figure 5.5 and Figure 5.6, it can be conveniently inferred that the

linear regression model fails to learn any pattern. Median value divides RUL distribution most evenly. This is why the best constant value, as in Figure 5.5, is around the median value and gives better RMSE scores than the linear regression approach.

5.3.2 Operating Conditions

The guidelines of C-MAPSS data challenge mention that the operational settings significantly impact a unit's performance [231],[195]. It has been shown by several works [195],[95],[13],[285] that plotting the operating setting values clusters the data points into distinct clusters. Data points from the datasets-1 and datasets-3 in C-MAPSS data set are all clustered at a single point (Figure 5.6), while those in datasets- 2 and datasets-4 are grouped into six distinct clusters (Figure 5.7).

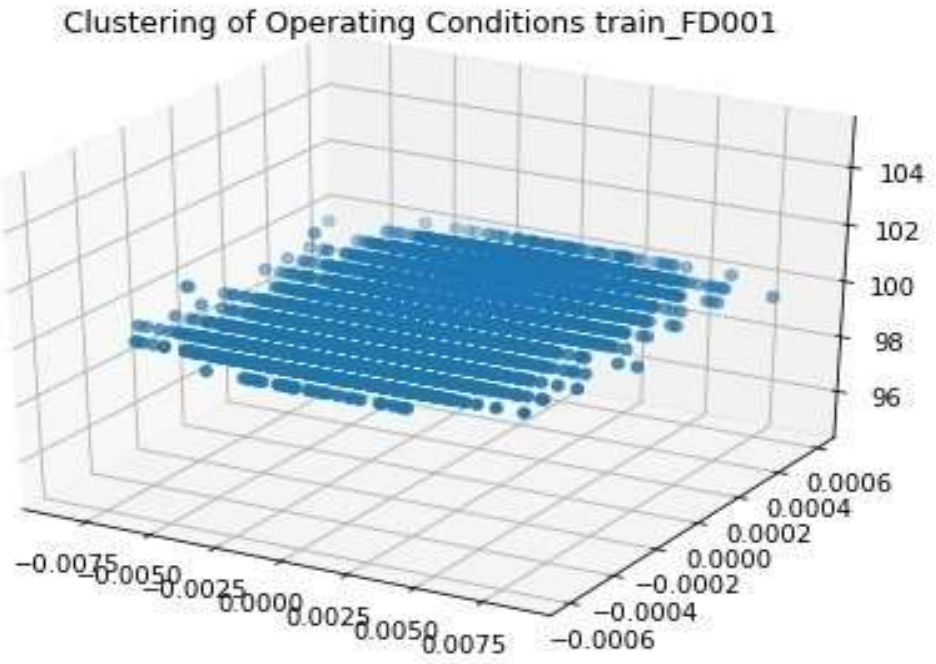

Figure 5.6: Dataset-1 Plot of operating setting values

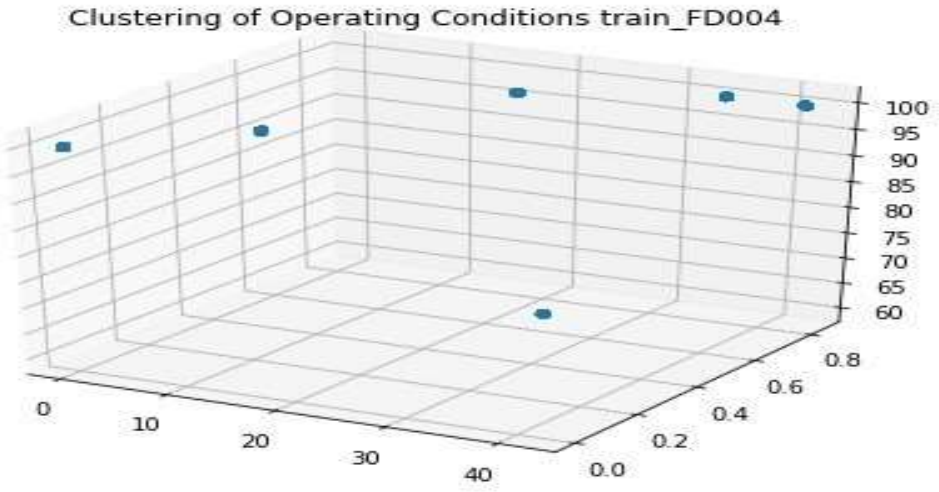

Figure 5.7: Dataset- 4 Plot of operating setting values

C-MAPSS is a time-series data, and so it becomes necessary to consider how to represent previous temporal observations for stateless machine learning models such as MLP. Experiments with MLPs do not suggest any substantial improvement with the sliding window representation of time-series data. Also, plotting sensor signals for any unit shows that there is no trend in the data, and signals jump between operating points during each run, as illustrated in Figure 5.8 [95].

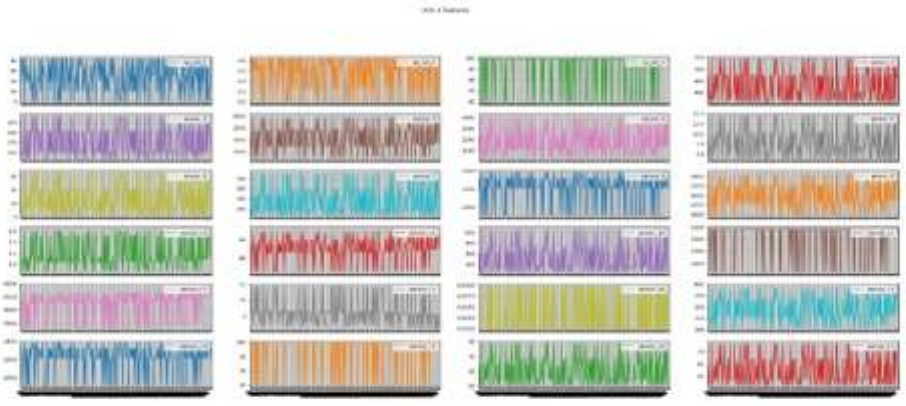

Figure 5.8: Plot of features for Dataset-1

Considering the importance of operating modes and that there is a requirement of time representation, an empirical decision was made to include operational mode history as features in a no memory machine learning model, except for RNNs. This is achieved by adding extra six features containing a total of the number of cycles spent in that mode since the beginning of the series [195]

5.3.3 RUL Target Function

The plots in Figure 5.9 show the MLP predicted RUL versus the original RUL for three different sequences. Each plot presents a consistent characteristic that the predicted curve starts flat, bends somewhere in the middle and decreases almost linearly towards zero. The plots in Figure 5.10 shows a consistent characteristic that the predicted curve starts flat, bends somewhere in the middle and decreases almost linearly towards zero. This establishes that degradation consists of four phases, namely, steady, knee, acceleration of degradation, and failure [95].

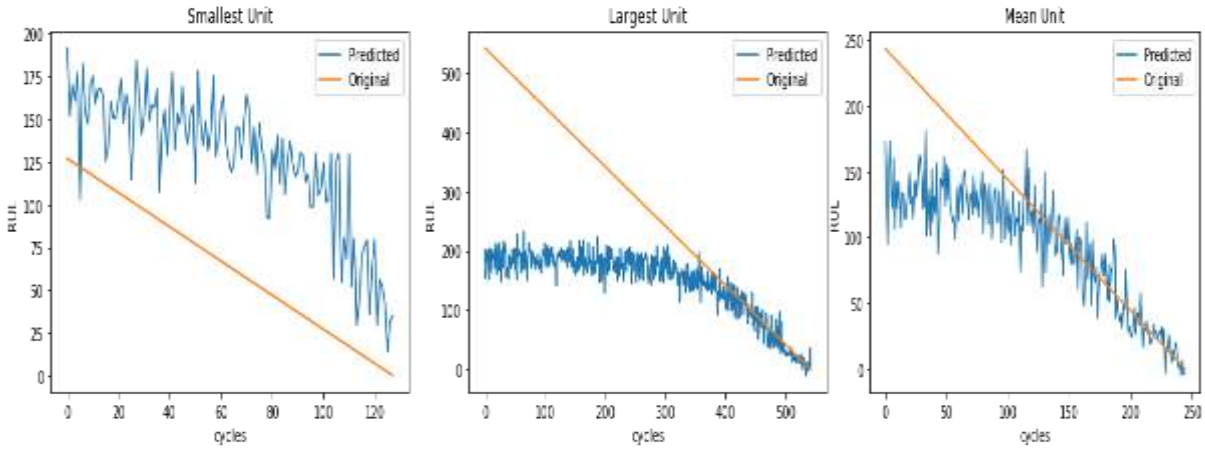

Figure 5.9: MLP neural network output versus linear target output

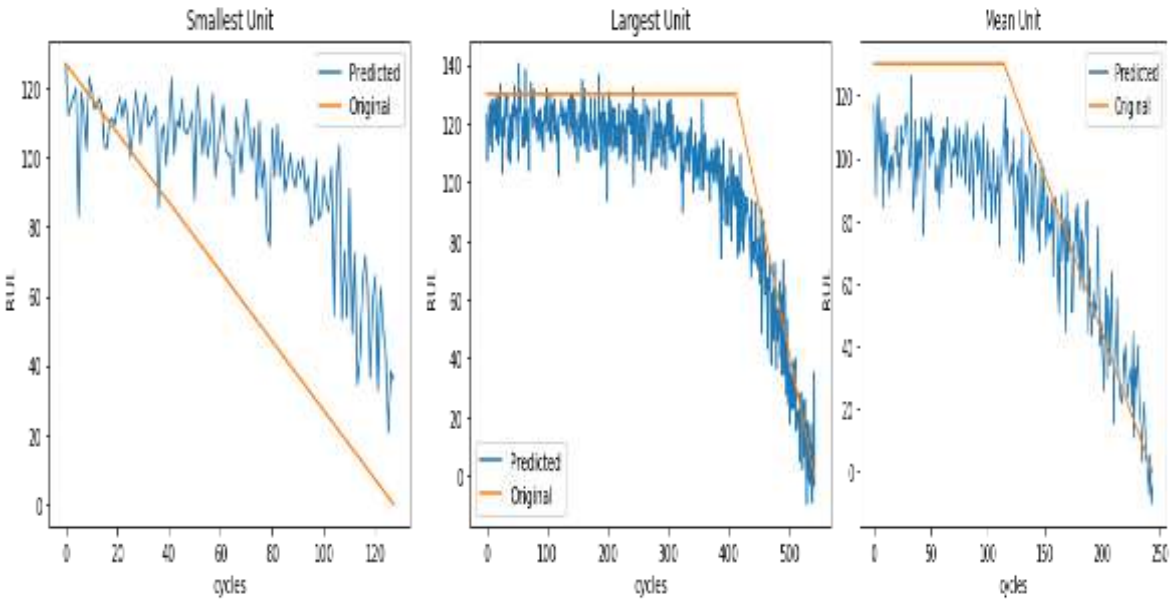

Figure 5.10: MLP neural network output versus piece-wise linear target output

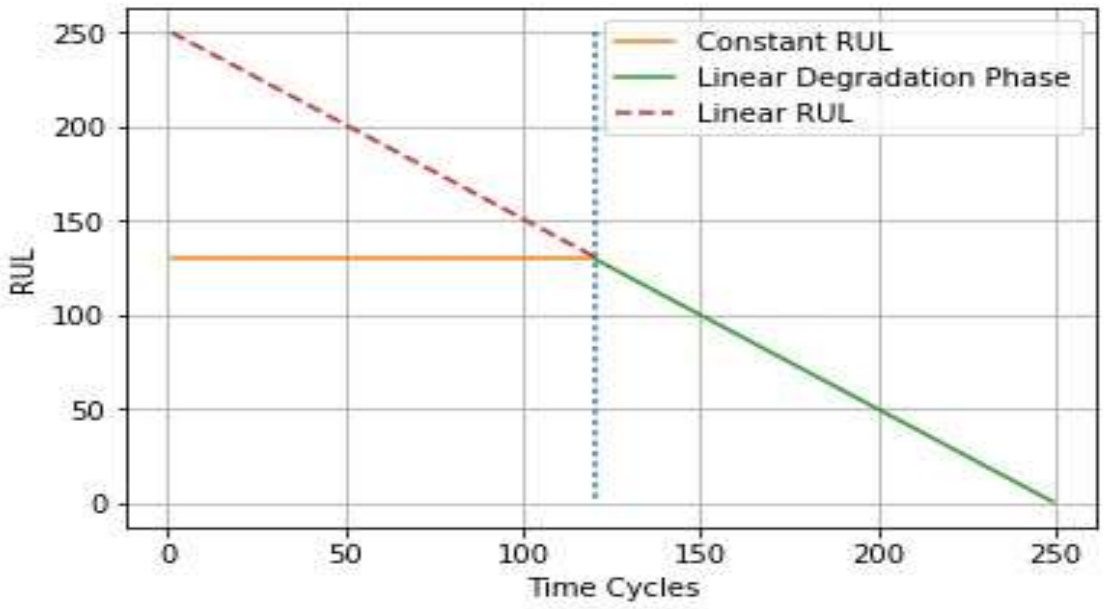

Figure 5.11: Piece-wise linear RUL target function

This pattern in trained model outputs is interpreted as implying that RUL estimation is not reasonable from the start, but only after the system has run for some time, and an initial failure has developed. This motivated changing the target output function from linear to piece-wise linear; that is, the maximum target output is limited to a constant value for all sequences. This is illustrated in Figure 5.11. This maximum value was chosen independently for each dataset based on observations and cross-validation.

5.3.4 Data Normalization

Many machine learning models are affected by the scale and variance of features. So, to provide a standard range across all the features, data is normalized in the pre-processing step of data processing. A usual method of scaling is:

$$Norm(x^f) = \frac{x^f - \mu^f}{\sigma^f}, \forall f$$

Where, for a feature f, x^f, μ^f, and σ^f are original data values, the mean and the standard deviation respectively for that feature.

5.4 Literature Review

Since its release, C-MAPSS datasets have attracted tremendous research work and more than seventy publications. Following is the summary review of work from four papers which claim most impressive results.

5.4.1 Earlier Performances on C-MAPSS Datasets

Multi-Layer Perceptron and Kalman Filter Based Approach: Peel *et al.* (2008) used advanced neuro scale mapping visualization technique to discover the six operating conditions from sensor data. They showed that inclusion of operational mode history as

features, helped the models. Also, a need for Kalman filtering was discussed to tackle sensor noise and sensitivity of RUL prediction to the arbitrary point of termination for a test system. It also allowed integrating past information in data instances. An ensemble of MLPs and Radial Basis Functions (RBFs) was implemented on Netlab to predict the RUL.

Recurrent Neural Network Approach: Heimes *et al.* (2008) started with data exploration and made some useful observations such as degradation consists of four phases, namely, steady, knee, acceleration of degradation and failure and introduced piece-wise linear target function for this task. They also showed that an MLP network was able to classify healthy and faulty states within an error of 1 percent.

Similarity-Based Approach: Wang *et al.* (2008) used several pre-processing techniques such as PCA and kernel smoothing. Datasets were divided into six bins corresponding to the six operating modes found using K-Means clustering applied on channels 3, 4, and five only. Sensors 7, 8, 9, 12, 16, 17, and 20 were manually selected as relevant features since they exhibit consistent trends.

CNN Based Approach: Zhao *et al.*(2016) introduced Convolutional Neural Network (CNN) on these datasets. They were motivated by application of Convolutional Neural Network architectures for classification tasks on multi-channel time series,. All the senor signals and six operation mode history features were used. Sliding window technique was used to transform each data instance into a 2D matrix form to include temporal information. In the CNN architecture proposed, two pairs of convolution layers and

pooling layers were used, such that the convolutional kernel moved only along the temporal dimension. Then, all end layer feature maps were concatenated into a vector to be used as the input to the final fully-connected neural network regressor layer for RUL estimation.

ELM vs. ANN: Zhe Yang *et al.* (2016) presented a comparative study of Backpropagation based Artificial Neural Network and Extreme Learning Machines (ELM) for RUL prediction task on C-MAPSS datasets. Data were normalized and filtered. Building on prior work by Coble *et al.* (2015), six prognostics signals were selected using the measures of Monotonicity, Prognosabilty, and Trendability. A Z-test method was used to find the time instant at which degradation started, and only data after this instant was used for the prediction model. For both the models, input was modelled as w consecutive measurements of the six signals, where w was the length of the time window. Experiments showed that an ELM with a large number of hidden neurons generalized well when the input dimension was large. The authors concluded that ANN outperformed ELM in accuracy, but ELM took considerably shorter training time than ANN, because ELM is computationally less intensive than ANN.

Bayesian Method: Mosallam *et al*. (2016) proposed a two-phase data-driven method wherein the online phase, training data was used to build health indicators from variables selected using an unsupervised method. For a new signal in the online phase, this method found the most similar signal from the saved health indicators to be used as a RUL predictor and finally estimated the new signals health status using a Bayesian approach. The steps involved in this method are summarized below: Offline Phase: Offline phase extracted representative features from the training data using trend construction method. It involves

two main steps: 1. Variable selection: Pairwise symmetrical uncertainty (SU) was calculated using an unsupervised variable selection algorithm based on information theory. Hierarchical clustering based on SU distance was then used to form and rank clusters according to the quality of the included signals in representing interesting relationships using normalized self-organizing map distortion measure. 2. Health indicator construction: Smooth monotonic representative features were extracted from the selected raw signals, which represented degradation as a function of time. This was done in the following steps: Variable compression: Principal Component Analysis (PCA) was used to reduce the dimensionality of selected features to one. Trend extraction: Empirical Mode Decomposition algorithm (EMD) was used to get a monotonic trend at each cycle. Feature extraction: Mean and variance of the input trend and slope and Y-intercept of the linear-fit of the input trend were used as the representative features. These extracted features were then used as health indicators and were saved in the database as reference models. For each cycle of a training instance, a health indicator was stored with Online Phase: Only the sensors selected in the offline phase were used to collect new test data. Representative features were calculated using the same variable compression, trend extraction, and feature extraction methods from the offline model. The generated feature vector was fed to a K-NN classifier to find the most similar offline signal. The end of life value of the offline signal with the highest posterior probability was then considered to be the RUL of the test signals. A recursive discrete Bayesian filter was finally applied to the online trends to estimate the actual value of the online health indicator at the predicted end of life value. The model was evaluated on two datasets: C-MAPSS turbofan engines and lithium-ion battery data, and claimed to get low Mean Absolute Percentage Error (MAPE) for both the applications

Time-window Neural Network Method: Lim Pin *et al.* (2016) proposed a multilayer neural network with K-Means based feature extraction method and a moving time window to mimic the dynamic temporal behaviour of RNNs. In the pre-processing step, features were selected after normalization using an MLP. Three feature extraction techniques were compared in a 10-fold cross-validation step, namely, PCA, Kernel PCA, and K-Means based feature extraction method. When the sliding window was used, K-Means based feature extraction method gave the best results on all datasets, otherwise, for dataset 4, no feature extraction method boasted an improvement. It was inferred from the experiments that the K-Means feature extraction method with more centres results in lower RMSE values. This paper, in essence, presented a technique to improve the performance of a basic MLP.

5.4.2 Methods on other datasets

RUL prediction task is a very broad area of research where researchers have used several intelligent algorithms on many datasets, other than C-MAPSS, including custom developed run-till-failure data. Following is a summary review of some of the most recent and diverse data-driven RUL prediction methods.

Deep Learning Approach: Jason Deutsch and David He (2016) proposed a Restricted Boltzmann Machine (RBM) to model vibration data to make predictions L steps ahead in the future. In any time, interval t, raw sensor data signals were represented by their Root Mean Square (RMS) value. RMS values for the last d time intervals became the d input features to the RBM. The RBM automatically mapped these lagged RMS input values onto the L-step ahead RMS values. The output of the last layer learned from the unsupervised stage of RBM learning was used as an input to a linear regression layer which predicted an

estimated point in time of bearings life. Experiments were performed on custom acquired run-till-failure data using a hybrid ceramic bearing. Data were normalized in the range [0, 1], and grid-search was used to tune the hyperparameters. The authors observed that prediction was better towards the end of the bearing life and that it also improved with increasing training data size. The paper concludes that the deep learning-based approach gave poor results than the particle-filter based approach, but that it did not require explicit model equations needed in particle filter-based approach and was also scalable for big data applications.

RNN-HI Method: Liang *et al.* (2017) proposed a recurrent neural network-based health indicator (RNN-HI) and RUL was predicted through an exponential model. The model consisted of three stages, including feature extraction, selection of sensitive features, and construction of the RNN-HI. This paper stated its two main contributions as 1. As an approach to extract statistical features within a specific range, a Related Similarity (RS) feature selection method was proposed. This ensured that all the features contribute equally to HI construction. RS calculated the similarity between the current inspection data and data at an initial operation point as a feature which ranges from 0 to 1. Therefore, the RS features could be directly used for RUL prediction without normalization. 2. An RNN is trained to automatically map selected features to an RNN-HI which represented the degradation percentage of a bearing. In the feature extraction step, six related-similarity features were combined with eight classical time-frequency features to form an original feature set. Then, a subset of most sensitive features was selected based on the average of monotonicity and correlation metrics. Finally, these selected features are used as input to an RNN to construct the RNN-HI. The proposed model was evaluated on two datasets: bearing data sets collected

from experiments and an industrial field dataset. The paper concluded that this RNN-HI method obtained relatively high monotonicity and correlation values, which are useful for RUL prediction and also achieved better performance than a self-organization map-based method.

5.5 The Proposed Model

5.5.1 Gradient Boosted Trees

Gradient Boosted Tress are also ensemble learners like Random Forest, which uses decision trees as base learners, but they differ in these following manners: 1. Random Forest is an ensemble of low bias and high variance deep decision trees whereas, boosted trees use high bias and low variance shallow trees as base learners. 2. In Random Forest, base decision trees grow in parallel, whereas in boosting it is sequential. Since boosting is a directed search where a new tree learns the gradients of the residuals at each iteration, between the target values and the currently predicted values. The algorithm then conducts gradient descent based on the learned gradients. Though, there are faster implementations of boosting where a decision tree itself is made to grow in parallel.

Introduction to Gradient Boosted Trees

Gradient Boost is a generalization of Adaboost in order to handle a variety of loss functions. In Gradient Boosted Trees, Classification And Regression Trees (CART) trees of a fixed size are used as base learners. A CART tree has a real score at each leaf unlike decision

values in decision trees. In an ensemble of trees, predictions of all the trees are added to get the final score. This can be formally written as:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F \qquad (5.1)$$

where, *K* is the number of trees, *f* is a function in the functional space F containing the structure of the tree and the leaf scores, and F is the set of all possible CART trees. We use the training data (with multiple features) x_i to predict a target variable y_i.

An, an additive training or boosting is used where the model starts with a constant prediction and adds a new function (i.e. CART tree) at each step.

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f1(xi) = \hat{y}_i^{(0)} + f1(xi)$$

$$\hat{y}_i^{(2)} = f1(xi) + f2(xi) = \hat{y}_i^{(1)} + f2(xi)$$

.....

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(xi) \qquad (5.2)$$

where, $\hat{y}_i^{(t)}$ is the prediction value at step *t*. So, the objective to optimize can be written as:

$$obj(\theta) = L(\theta) + \Omega(\theta) \qquad (5.3)$$

Where, L is the training loss function, and Ω is the regularization term

a commonly used training loss is mean squared error

$$L(\theta) = \sum_i (y_i - \hat{y}_i) \wedge 2 \tag{5.4}$$

So, the objective function is

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + \text{constant} \tag{5.5}$$

If we consider using MSE as our loss function, it becomes the following:

$$obj^{(t)} = \sum_{i=1}^{n} \left(y_i - \left(\hat{y}_i^{(t-1)} + f_t(x_i) \right) \right) \text{^} 2 + \sum_{i=1}^{t} \Omega(f_i)$$

$$= \sum_{i}^{n} [2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i) \text{^} 2] + \Omega(f_t) + constant \tag{5.6}$$

It should be noted that such a nice form with a first order term (usually called the residual) and a quadratic term is not necessary for all the loss functions, so we take the Taylor expansion of that loss function up to the second order.

$$obj^{(t)} = \sum_{i=1}^{n} \left[l\left(y_i, \hat{y}_i^{(t)} \right) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant \tag{5.7}$$

where the g_i and h_i are defined as

$$g_i = \partial_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)} \right) \tag{5.8}$$

$$h_i = \partial^2_{\hat{y}_i^{(t-1)}} l\left(y_i, \hat{y}_i^{(t-1)} \right) \tag{5.9}$$

After we remove all the constants, the specific objective at step t becomes

$$obj^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \tag{5.10}$$

This cost function is to be minimised for the parameters f_i. Gradient descent is performed by the algorithm based on the gradients of the residuals at each iteration. That new tree is generated which minimises this cost function.

Model Complexity: We need to define the complexity of the tree Ω(f). In order to do so, let us first refine the definition of the tree f(x) as

$$f_t(x) = w_{q(x)}, w \in R^T, q: R^d \longrightarrow \{1,2,\dots,T\} \tag{5.11}$$

Here w is the vector of scores on leaves, q is a function assigning each data point to the corresponding leaf, and is the T number of leaves. In XGBoost, we define the complexity as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{5.12}$$

There are more than one way to define the complexity, but this specific one works well in practice. The regularization is one part most tree packages treat less carefully, or simply ignore. This was because the traditional treatment of tree learning only emphasized improving impurity, while the complexity control was left to heuristics. By defining it formally, we can get a better idea of what we are learning, and yes it works well in practice.

Application on C-MAPSS Datasets: Grid search with K-Fold cross-validation is used to tune the hyperparameters of a Gradient Boosted Tree model, such as learning rate, maximum depth of base trees, a minimum number of instances in each node, subsample ratio of columns in creating each tree and regularization parameters. Early stopping criteria was used to stop generating new estimators if the validation score was not improving from the last 500 estimations. This is important to avoid overfitting resulting from training over the noise. Experiments presented some interesting observations:

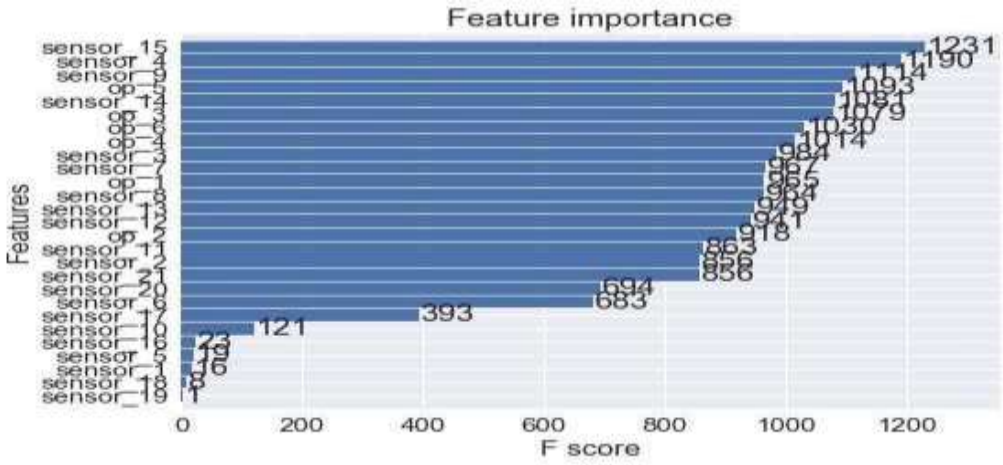

Figure 5.12: Feature importance with colsample=0.25

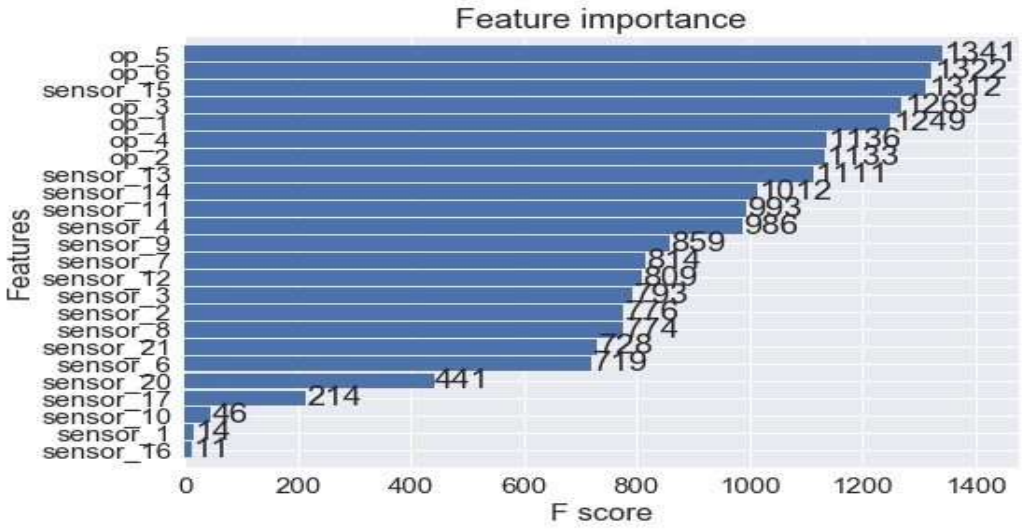

Figure 5.13: Feature importance with colsample=1

Error decreased with decreasing subsample ratio of columns used in creating each tree (say, colsample) Which offered an important observation that operating mode history shadows the patterns in sensor signals. In both cases, six operating mode histories were among the top features which re-establishes their importance in RUL prediction. But, reducing colsample value allowed the model to explore smaller subsets of features, and thus it was able to find patterns in sensor features otherwise missing. Thus, lower colsample value

made the model less sensitive to bad features. Feature importance arrays for two colsample values are plotted in Figure 5.12 and Figure 5.13. Sensors 3, 4, 7, 9, 14 and 15 features consistently among the most important features, apart from six operating mode history features.

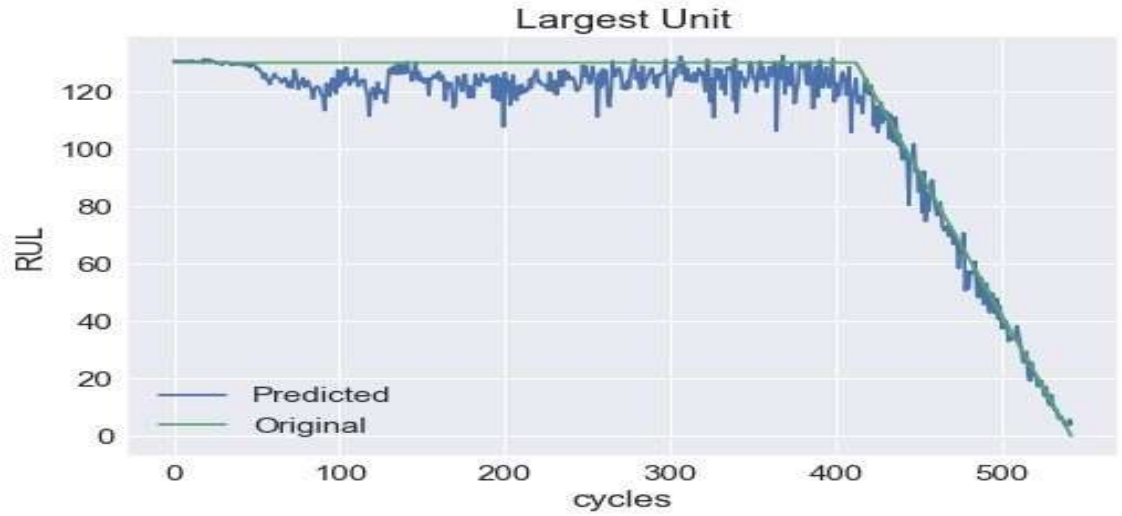

Figure 5.14: The Gradient Boosted Trees Prediction on training system for largest unit.

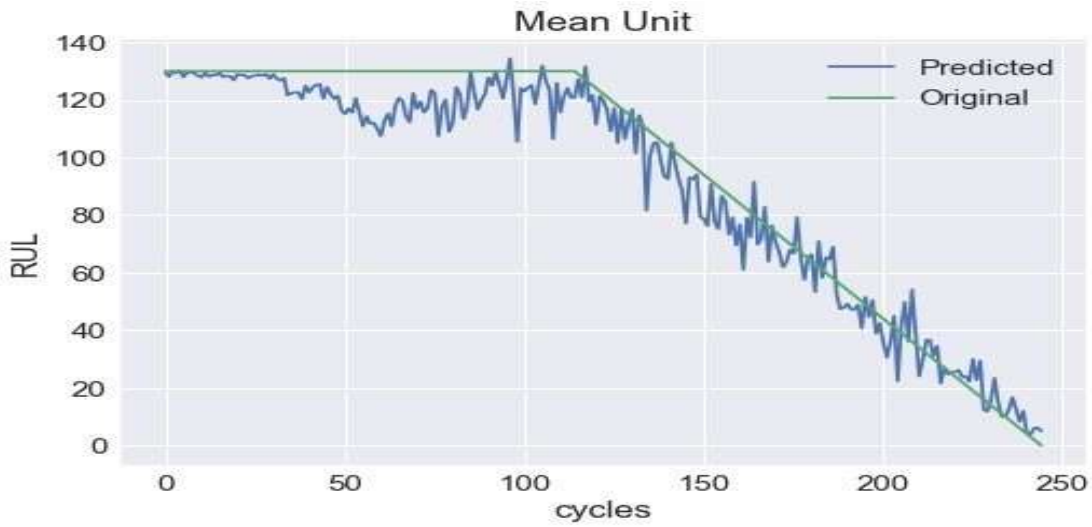

Figure 5.15: The Gradient Boosted Trees Prediction on training system for meant unit.

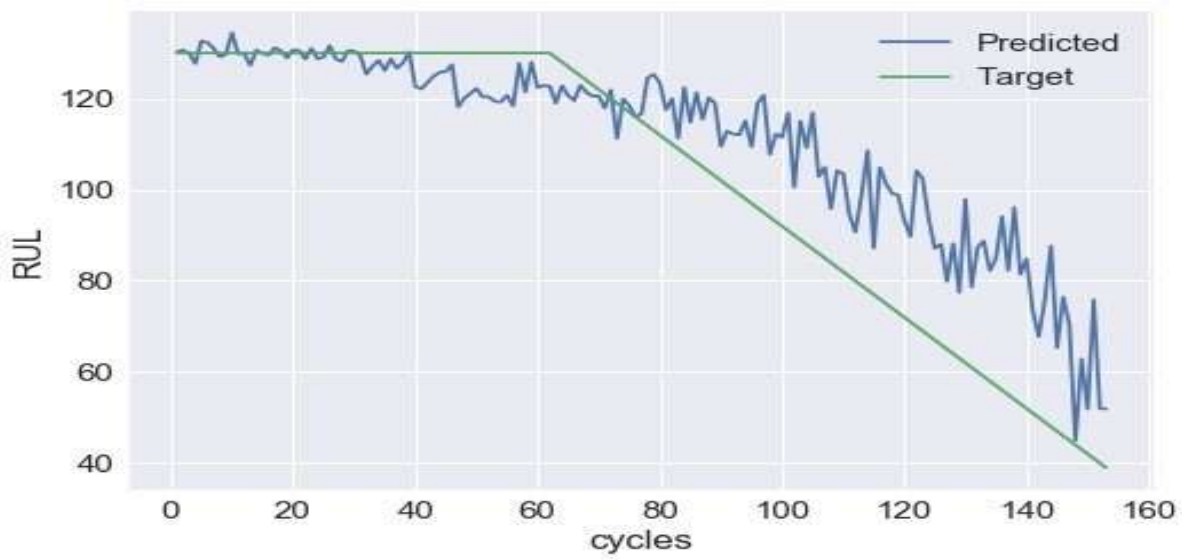

Figure 5.16: Gradient Boosted Trees predictions on test systems

Table 5.4: RMSE from gradient boosted trees model

Train Dataset	#1	#2	#3	#4
RMSE	17.53	26.89	19.76	27.98

RMSE scores achieved with XG Boosts implementation of gradient boosted trees are noted in Table 5.2. Fig 5.14, Fig 5.15 and Fig 5.16 show that this model over-predicts after the first phase of degradation.

5.5.2 MLP

Multi-Layer Perceptron is a feedforward artificial neural network architecture where each layer is fully connected to the next layer of nodes. It is a standard non-linear supervised learning model which uses backpropagation technique for training the network [217]. The grid search technique with K-fold cross-validation was used to tune various hyperparameters

of a model, such as the number of hidden layers, the number of hidden units in each layer, the number of training iterations, batch size, weight initialization, and activation function. Experiments presented some interesting observations: Noise: Plots of model predictions versus target outputs showed that the sensor data is quite noisy. Vanishing Gradient: It was observed that both wide and deep neural networks were learning slowly with the sigmoid activation function.

This is the problem of vanishing gradient. As the absolute value of z increases, the sigmoid gradients get close to zero and the net learns slowly. Rectifier function (ReLU) does not squeeze its output in range. So, it learns faster due to its constant gradient nature for z = 0.

ReLU(z) = max (0, z) Experiments also showed ReLU to be converging more quickly and giving better results. So, it was used as the activation function in all the layers [72], [73]. Dense than narrow: A wide single hidden layer network was found to be performing better than the narrow ones, which reflects the presence of complex non-linear relationships between features. Table 5.5 notes RMSEs for a different number of hidden units in a single hidden layer network, averaged over three runs:

Table 5.5: RMSE vs. size of single hidden layer

Train Dataset	5	10	20	30	50	75	100	150	200	250	Best
1	21.78	19.37	18.87	18.47	18.44	18.33	18.62	18.63	18.34	17.95	17.95
2	30.20	30.06	28.97	29.09	29.24	28.76	29.15	29.88	28.46	29.65	28.46
3	21.89	21.98	21.79	21.78	21.93	20.31	21.44	20.41	20.92	21.57	20.31
4	37.01	35.60	34.16	34.18	33.49	33.04	32.47	32.36	31.13	31.59	31.13

Deep or shallow: A deep network did not help in improving the scores, but it was observed that, with equal computational complexity regarding the number of parameters, it was converging faster in the number of epochs.

5.5.3 The Stacking Ensemble

Stacking is an ensemble method in which a second-level meta-learner is trained on predictions from many base learners [245]. The goal in stacking is to both minimize variance and increase predictive force. It can produce better results than a single model and averaging methods. The stacking method can be outlined in Figure 5.17, as follows:

1. Set up the ensemble

(a) L models with a particular set of parameters for each are selected as base learners.

(b) A second-level model is specified.

2. Training the ensemble

(a) All the base algorithms are trained on the training set via K-Fold cross-validation.

(b) From training each of the L algorithms, N cross-validated predicted values are saved, where N is the number of training instances.

(c) These predicted values are combined into a new N*L matrix. This matrix and true training values are called level-one input.

(d) The second-level algorithm that is the meta-learner is trained on the level one data.

(e) The ensemble model consists of these L base learning models and the meta-learning model. This ensemble model can be used to generate predictions on a test set.

3. Prediction on test data

(a) First, predictions are generated from the base learners for the test data.

(b) Finally, those predictions are fed into the meta-learner to produce the ensemble prediction.

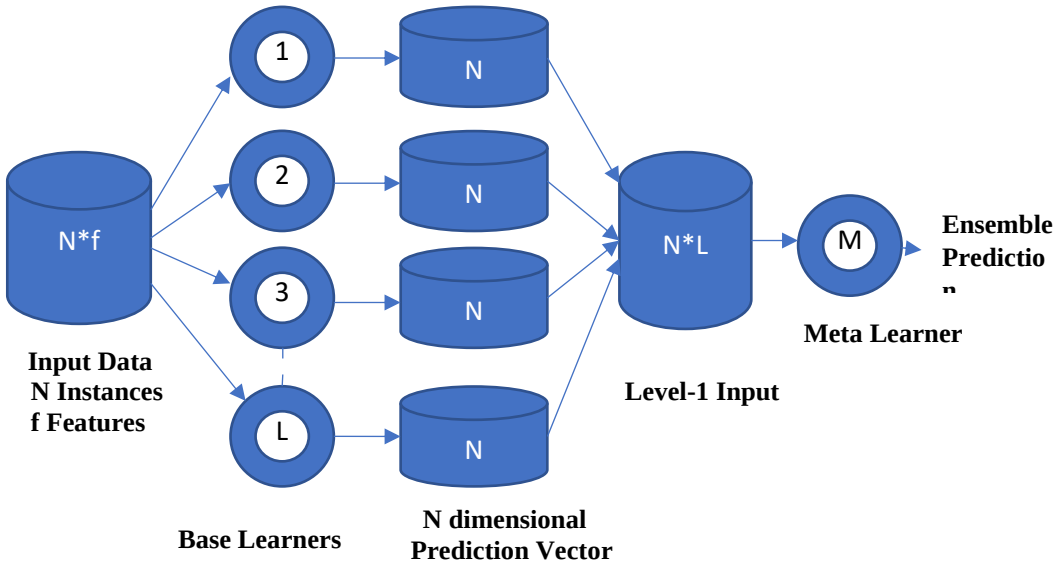

Figure 5.17: Stacking model diagram

Following observations were made during experiments with ensemble modelling: An ensemble of less correlated base learners is more efficient than that, when, the base functions are all learning different representations. In experiments, this was experienced that a model with all the base learners being neural network models was less accurate than selecting from

both neural networks and gradient boosted trees. Even, the average of predictions from all neural network base learners produced poorer results. As meta-learner, a single hidden layer neural network was found to be better than a gradient boosted tree. Gradient Boosted Trees cannot extrapolate predictions similar to Random Forest. Best results so calculated using this stacking method are tabulated in Table 5.6.

Table 5.6: RMSE for stacking method

Train Dataset	1	2	3	4
RMSE	16.67	25.57	18.44	26.76

5.6 Experimental Results

Three algorithms are tested on the four C-MAPSS datasets, namely MLP, Gradient Boosted Trees (GBT) and the stacking ensemble method. For comparison purposes, methods from other papers which report RMSE on C-MAPSS datasets are also included in the Table 5.7. These cited methods include SVR and CNN by Zhao *et al.* (2016). The benchmark score by Ramasso *et al.* (2014). For the time-window Neural Network method, RMSE value is available only for dataset-1 [148]. Table 5.7 illustrates their comparison results. It is observed that GBT is significantly better than MLP, SVR, and CNN on the datasets with multiple operating conditions, which are 2 and 4. For single operating condition datasets-1 and datasets--3, GBT is again better than these three, but the improvement in RMSE is small. This indicates that GBT is strongly dependent on operating mode history features in learning degradation patterns,

Table 5.7: RMSE values for various methods on C-MAPSS datasets

MLP	17.95	28.46	20.31	31.13
SVR	20.96	41.99	21.05	45.35
CNN	18.45	30.29	19.82	29.16
GBT	17.53	26.89	19.76	27.98
Proposed Stacking Ensemble	16.67	25.57	18.44	26.76

which was also obvious from feature importance plots (Figure 5.12). SVR achieved the highest RMSE values across all the datasets. It can be seen that a dense single hidden layer MLP is performing better than the CNN model presented by Zhao *et al.* (2016). Also, the stacking method consistently achieves the best results among all these methods, across all the data subsets, regardless of the operating conditions. This demonstrates its ability to increase predictive power and to handle noise better than any single model. Time-window neural network method reports better RMSE score than the proposed stacking ensemble method on dataset-1(RMSE = 15.16). A total comparison is not possible due to unavailability of RMSE values on the rest of the datasets, but this suggests that using feature selection algorithms should be a good idea for future work.

5.7 Conclusion

In this chapter, two supervised learning algorithms are introduced for RUL prediction task, namely Gradient Boosted Trees, and a stacking ensemble method. Experiments were performed on the C-MAPSS datasets, and the metric used for scoring models was RMSE. A comparison is performed with other methods for which RMSE scores are available for the C-MAPSS datasets. Results show the effectiveness of the proposed models. Gradient Boosted Trees model outperforms MLP, SVR, and CNN. It also gives feature importance information. Feature importance shows that operation mode history is the most significant pattern in degradation recognition. It also finds that the sensors 3, 4, 7, 9, 14, and 15 give the most relevant sensor signals. Plotting predictions versus target RULs show that sensor signals are noisy and that degradation process is a four-step process. A method of stacking ensemble of feed-forward neural networks and boosted trees with a 1-hidden layer fully connected neural network as the second-level learner is presented. This ensemble gives better results than any of the models alone or than a weighted average of their predictions. It is concluded from the results that this stacking method can increase predictive accuracy, as well as handle noise better than individual models by increasing variance. Future works include improving on the proposed architectures by applying various feature selection techniques in pre-processing steps and extending the stacking ensemble technique to use other types of base models, such as Support Vector Regression (SVR) and Recurrent Neural Networks (RNNs) to increase the diversity of the population in the ensemble.