ECE 225A Project Report

Onat Gungor[1]

[1]Department of Electrical and Computer Engineering, University of California San Diego
ogungor@ucsd.edu, PID: A53328013

I. Introduction

Predictive maintenance (PDM) estimates time-to-failure of a machine using a variety of mathematical approaches. PDM aims to calculate an optimum schedule for maintenance before any failure occurs. Remaining useful life (RUL) is defined as the time remaining for a component to perform its functional capabilities before failure. Its accurate prediction ensures a perfectly working PDM system. RUL prediction is essentially a regression problem where we predict a single number. Generally, the input is a time-series data which is collected from different sensors placed on a machine. Hence, there is a need to convert time series problem into a machine learning problem. Before that, understanding the data is crucial to obtain a great RUL prediction, that is why we start with data exploration and discover interesting information about our data set. We visualize data from different perspectives and we perform distinctive analyses such as correlation analysis, and trend analysis. Afterwards, we pre-process the data where we do normalization and perform automated feature selection using random forest. Ultimately, we utilize various traditional machine learning and deep learning models to predict RUL.

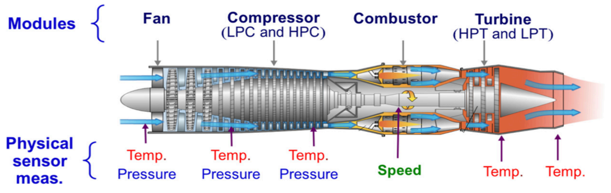

Fig. 1: Engine diagram simulated in C-MAPSS [1]

II. Data-set Description

NASA C-MAPSS (Commercial Modular Aero-Propulsion System Simulation)[1] is a simulation based aircraft engine data set. The engine diagram in Figure 1 depicts the main elements of the engine model. The main components include: fan, compressor, combustor, and turbine. Variety of sensors (e.g. temperature, pressure) placed on these components construct the input data. There exist four sets of data: FD001∼FD004. Each data set has separate training and testing sets. While the training data contains the entire lifetime of an engine, test data is terminated at some point before engine failure. Each row represents data during a single operating time cycle with 26 columns: the engine ID, cycle index, three operational settings, and 21 sensor measurements. At the beginning of each time series, the engine is operating normally and develops a fault at some point in the future. The ground-truth RUL values are provided for the test data, and the goal is to predict the RUL before failure on the test data. **For demonstration purposes, we select FD001 as our data-set**.

[1]https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/

III. Data Exploration

First, we separate test data to be used solely for inference. For our training data, we have 24,720 rows (cycles) and 26 columns where none of those values are empty or missing. In total, we have 100 different aircraft engine simulation data.

Aircraft engine life time analysis: We first analyze the life time (i.e. time when an aircraft engine fails) of engines. We demonstrate life time distribution in Figure 2 using a violin plot. In this figure, on the x-axis, we have cycle number of an engine and y-axis demonstrates the probability density of the data at different values. Inside the given distribution, we have also a box plot which shows some summary statistics such as mean, and median. Table I presents those statistics explicitly. This analysis gives us an idea about the possible range and different values of an engine life time.

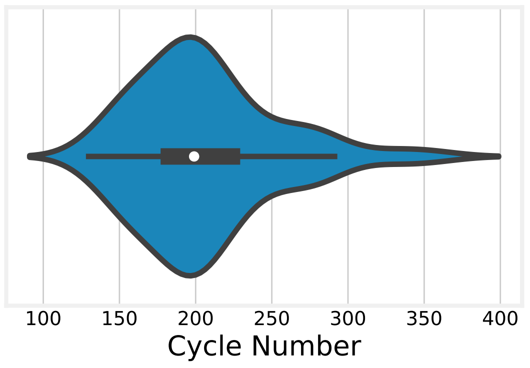

Fig. 2: Life time distribution of aircraft engines

TABLE I: Summary statistics for aircraft engine life time

Statistic	Mean	Median	Std	Min	Max
Value	206.3	199	46.3	128	362

Operational settings analysis: For this analysis, we visualize operational settings data columns. For the sake of clarity and simplicity, we perform illustration using the first aircraft engine. Figure 3 shows three different operational settings measurements. In this figure, each subplot represents different operational setting data where vertical axis gives measurements with respect to cycle number. Here, we observe that last operational settings column stays constant throughout the engine life time. This observation is also valid for the rest of the aircraft engines in our training data.

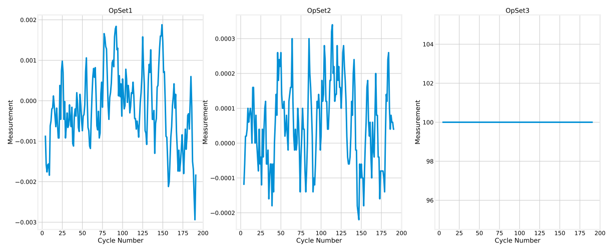

Fig. 3: Operational settings data visualization for the first aircraft engine

Sensor measurements analysis: Similar to previous analysis, we visualize different sensor measurements across time. Figure 4 illustrates all available sensor measurements for the first aircraft engine. Here, each subplot represents different sensor measurements with respect to cycle number. Similarly, we can observe that some sensor measurement values (sensors 1, 5, 6, 10, 16, 18, 19) did not change at all. This is again valid for other aircraft engines.

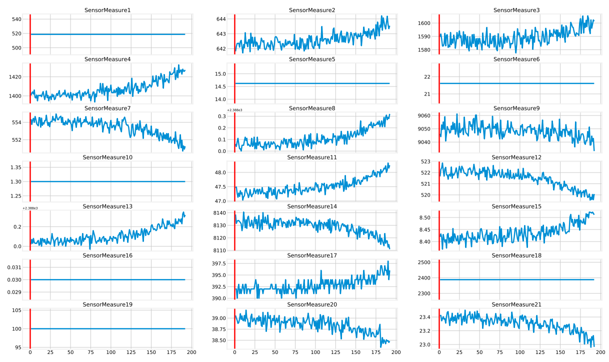

Fig. 4: Sensor data visualization for the first aircraft engine

Standard Deviation Analysis: We use all columns and measure their standard deviation to detect constant columns. We discover that operational setting 3, sensors 1, 5, 10, 16, 18, and 19 have zero standard deviation across all aircraft engines. We also observe that sensor 6 measurements either stay constant or fluctuates between two values. Hence, we also include sensor 6 into zero standard deviation group. We eliminate those features since they do not add any value to our further analysis.

Correlation analysis: In this analysis, we use Pearson's correlation coefficient between pairwise sensor measurements. Pearson's correlation coefficient measures linear correlation between two variables X and Y and it has a value between +1 and -1. A value of +1 is total (strong) positive linear correlation, 0 is no linear correlation, and -1 is total (strong) negative linear correlation. Person's correlation coefficient ($\rho_{X,Y}$) can be formulated as follows:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \qquad (1)$$

where X and Y corresponds to specific sensor measurements (e.g. $\rho_{4,8}$ gives the correlation coefficient between sensors 4 and 8), cov is the covariance between sensor measurements, σ is the standard deviation. We create a correlation coefficient matrix among sensor measurements illustrated in Figure 5. In this figure, darker red and blue corresponds to strong linear positive and negative correlation respectively (i.e. the lighter the color, the smaller the correlation). Note that in this matrix, we also have RUL cell to detect correlation of sensor measurements with RUL. Notably, there is a really high correlation ($\rho_{9,14} = 0.96$) between sensors 9 and 14. We also present the scatter plot between those two measurements in Figrue 6 which shows this high correlation. Accordingly, we remove sensor 14 since it is less correlated with RUL ($\rho_{RUL,14} = -0.31$ vs $\rho_{RUL,9} = -0.39$).

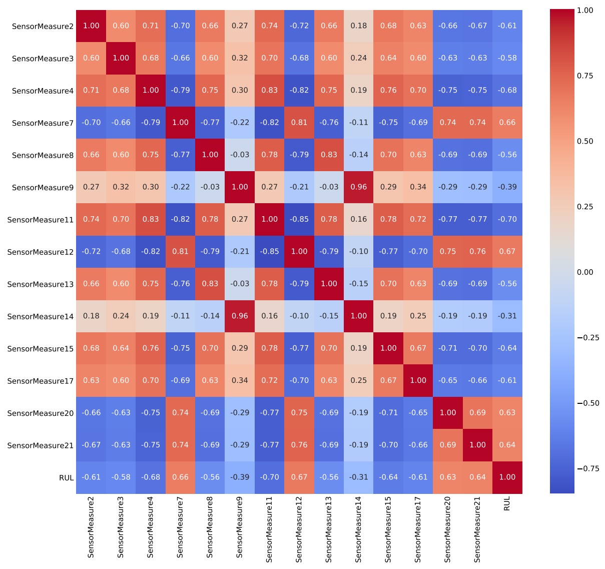

Fig. 5: Correlation matrix among sensor measurements+RUL

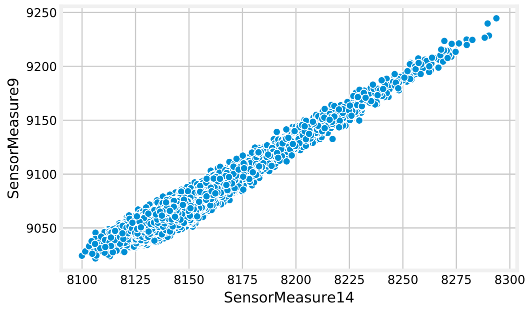

Fig. 6: Scatter Plot Using Sensors 9 and 14

Trend analysis: We perform a trend analysis (i.e. apparent upward/downward change in measurements) to observe whether degradation can be discovered specific to each feature. Similar to what we illustrated in Figure 4, we discover trend common in aircraft engines. Specifically, we look at the trend at the last 50 cycles of aircraft engines. Figure 7 provides an example trend analysis where we provide sensor 11 measurements over the last 50 cycles of the first 5 aircraft engines. Here, x-axis shows the cycle number and the y-axis provides the sensor measurement values. Each color represents a different aircraft engine. We can observe trendy behavior across time where sensor measurement values increase with time. Sensors showing trendy behavior across time can help more in RUL prediction, that is why we select those sensor measurements. We draw the same conclusion as in our sensor measurement analysis where we remove sensors 1, 5, 6, 10, 16, and 19. They are same due to the fact that changing sensor measurements also show trendy behavior across time.

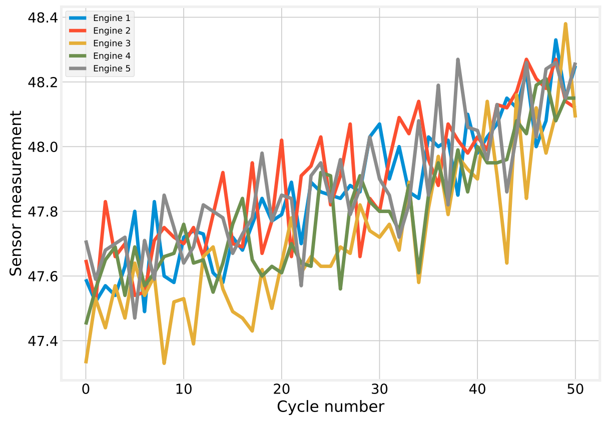

Fig. 7: Trend analysis for sensor 11

IV. Data pre-processing

Data pre-processing is an indispensable step in any machine learning task. Previously, we have presented an insight into our data-set. While doing that, we emphasize that some features may be redundant and we eliminate those ones. The goal of data pre-processing is to perform feature selection in an automated manner. We select the most useful features to (1) increase prediction accuracy, (2) decrease computational complexity, and (3) avoid overfitting. After we normalize our data, we measure variable importance using Random Forest (RF).

RUL Target Function: RUL is generally modeled linearly where its value decreases with time. However, the degradation of the machine performance is not apparent in the beginning of its lifetime and increases when a machine approaches its end of life. Hence, we model RUL using a piece-wise linear function. The maximum RUL limit constant (the break point) is set to 125-time cycles meaning that RUL values greater than 125 cycles set to 125, smaller ones staying the same.

Data Normalization: We normalize the data using min-max normalization according to Equation 2 where x_i is an input data to be normalized and $\ddot{x}_i$ indicates the normalized data in [0,1]. Data normalization is a fundamental step since sensor measurements have different ranges. By changing the measurement values to a common scale, we eliminate possible data related problems that can significantly affect the prediction performance.

$$\ddot{x}_i = \frac{x_i - \min x_i}{\max x_i - \min x_i} \tag{2}$$

Random Forest (RF) for Automated Variable Selection: We use RF to discover variable importance which is calculated based on the reduction in residual sum of squares. We analyze variable important values for different features. Figure 8 illustrates non-zero feature importance values for our training data. Y-axis has feature importance values in [0,1] and x-axis shows different features. We observe that the most informative feature is sensor 11 with a value of 0.54. The second most informative feature is normalized cycle number with a value of 0.22. We observe that the following features have zero variable importance: Operational setting 3, sensors 1, 5, 10, 16, 18, and 19. Sensor 6 has also an importance value of 0.0001 which is significantly small. Overall, we reach the same conclusion as in data exploration step. We eliminate those features since they do not provide useful information in RUL prediction. Additionally, in correlation analysis, we also observe that sensors 9 and 14 are highly correlated where we drop sensor 14 column since it is less correlated with RUL. By including this fact, we eliminate 9 features (37% of all features) overall from our data sets.

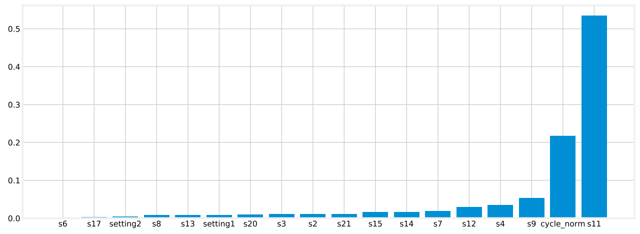

Fig. 8: Random Forest Variable Importance

V. Inference

Our goal is to predict the remaining useful life (RUL) of aircraft engines in the test data. We implement ML-based models under two categories: traditional machine learning and deep learning. Selected models are trained using the training data, and the performance is evaluated on the test data.

Performance Evaluation Metric: Prediction error (ϵ) is the difference between the estimated RUL (RUL_{est}) and the true RUL (RUL_{true}) (i.e. $\epsilon = RUL_{est} - RUL_{true}$). We use Root Mean Square Error (RMSE) for evaluation, formulated as:

$$RMSE = \sqrt{\frac{1}{\mathcal{N}} \sum_{i=1}^{\mathcal{N}} \epsilon_i^2} \tag{3}$$

A. *Traditional Machine Learning Models*

For traditional ML models we directly provide selected time series sensor data and ground truth RUL values. For the implementation of these models, we use Scikit-learn library.

Linear Regression (LR): LR models the relationship between two variables (explanatory and dependent) by fitting a linear equation to observed data [2]. A linear regression line has an equation of the form $Y = a + bX$, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept. The most common method for fitting a regression line is the method of least-squares. This method calculates the best-fitting line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line.

Support Vector Regression (SVR): Support Vector Machine (SVM) aims to find the decision boundary to separate different classes. While finding this decision boundary (i.e. hyperplane), it maximizes the margin which is defined as the perpendicular distance between the hyperplane and the closest elements from classes [3]. For our problem, we use SVM as a regression method (i.e. SVR). SVR uses the same principles as the SVM for classification, with only a few minor differences: predicting a single number as well as determining a margin of tolerance (ϵ). Training the SVR is equivalent to solving the following optimization problem [4]:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|w\|^2 \\ \text{subject to} \quad & |y_i - (w \cdot x_i + b)| \leq \epsilon \end{aligned}$$

where x_i is a training sample with target value y_i. The inner product plus intercept $w \cdot x_i + b$ is the prediction for that sample, and ϵ is a free parameter that serves as a threshold: all predictions have to be within an ϵ range of the true predictions. We utilize grid search to find the optimal hyper-parameters of SVR. Here are the found optimal hyper-parameter values: C (regularization parameter) = 10, ϵ (margin of tolerance) = 10, gamma (kernel coefficient) = 5, kernel = rbf.

B. Deep Learning Models

In traditional ML models, we do not consider the relationship within the time series data (i.e. time dimension). To find out that information and obtain better feature extraction, we adopt a sliding time window approach. The time window represents the number of past observations to be considered and we slide this window from the first observation to the last. To illustrate, provided that the window size is 30, our first window includes the observations from 1 to 30, second window includes observations from 2 to 31 and so on. Accordingly, 2-D input is provided to DL models where the first dimension represents the selected features and the other one has the time sequence of each feature. For the implementation of these models, we use Keras library. We run all models with the same hyper-parameter configuration: *Adam* optimizer with learning rate 0.001, *elu* activation function, *He* initialization, batch size of 512, and a max number of epochs of 250 where callback is activated (patience is set to 10 for validation data).

Deep Long Short-Term Memory (DLSTM): LSTM is proposed to prevent vanishing and exploding gradient problem. They are specific recurrent neural networks (RNN) with special memory cells to store information over longer periods of time. Updates in this cell can occur by the activation of three distinctive gates: 1) forget gate (the memory cell is cleared completely), 2) input gate (memory cell stores the received input), and 3) output gate (next neurons obtain the stored knowledge from the memory cell) [5]. Since LSTM can recall information for long periods of time, it is a good fit for RUL prediction tasks. Zheng et al. [6] propose a DLSTM network for enhanced RUL prediction, with 2 consecutive LSTM layers (each with 32 nodes) followed by 2 fully connected feed forward neural networks (each with 8 nodes). Final 1-dimensional output layer provides RUL prediction.

Deep Convolutional Neural Network (DCNN): DCNNs use multiple feature extraction stages that can automatically learn hidden representations. The convolutional layers convolve multiple filters with raw input data and generate features, and the following pooling layers extract the most significant local features. 1-D CNN is common for time series applications, making it a suitable model for RUL prediction. Li et al. [7] propose a DCNN model where their network structure consists of five consecutive 1-D CNN layers, flatten layer, one fully-connected layer (with 100 nodes) and an output layer with 1 node.

C. Performance Comparison

We measure RMSE values for the selected 4 models and present those values in Figure 9. X-axis represents the specific model, and y-axis provides the corresponding RMSE value. To be more exact, RMSE values for selected models are provided in Table II. We observe that DL models are more accurate than traditional ML based approaches. Overall, DCNN performs best, DLSTM being second. Since LR is the simplest model, we obtain the worst performance using that predictor.

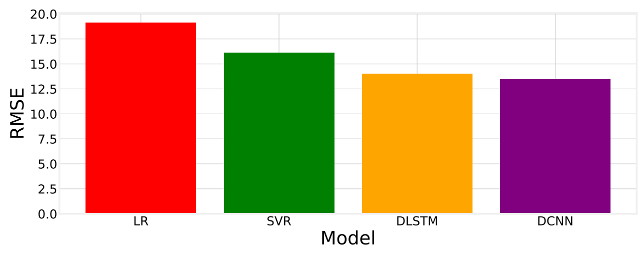

Fig. 9: RMSE Value Comparison

TABLE II: RMSE values for the selected models

Model	**LR**	**SVR**	**DLSTM**	**DCNN**
RMSE	19.14	16.11	14.04	13.45

References

[1] A. Saxena and K. Goebel, "Turbofan engine degradation simulation data set," *NASA Ames Prognostics Data Repository*, 2008.

[2] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis*, vol. 821. John Wiley & Sons, 2012.

[3] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[4] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[5] A. Gensler, J. Henze, B. Sick, and N. Raabe, "Deep learning for solar power forecasting—an approach using autoencoder and lstm neural networks," in *2016 IEEE international conference on systems, man, and cybernetics (SMC)*, pp. 002858–002865, IEEE, 2016.

[6] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *IEEE ICPHM*, pp. 88–95, IEEE, 2017.

[7] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety*, vol. 172, pp. 1–11, 2018.