Predictive Maintenance of Turbofan Engines Using NASA Dataset

Quek Zhu Hui Joel (Group 133)

July 26, 2024

Abstract

This project aims to develop a predictive maintenance system for turbofan engines by accurately predicting the Remaining Useful Life (RUL) of the engines. Accurate RUL predictions can significantly reduce downtime and maintenance costs, thereby enhancing operational efficiency and reliability. The project utilizes the NASA Turbofan Engine Degradation Simulation Dataset, leveraging advanced machine learning techniques, including Linear Regression, Random Forest, Support Vector Regression (SVR), Gradient Boosting Machines, Long Short-Term Memory (LSTM) networks, and Q-Learning, to achieve robust and reliable predictions.

1 Introduction

The primary objective of this project is to develop a predictive maintenance system for turbofan engines by accurately predicting the Remaining Useful Life (RUL) of the engines. Accurate RUL predictions can significantly reduce downtime and maintenance costs, thereby enhancing the operational efficiency and reliability of turbofan engines.

2 Problem Statement

Predictive maintenance is critical in various industries, especially in aerospace, where unexpected engine failures can lead to significant operational disruptions and safety risks. The challenge is to accurately predict the Remaining Useful Life (RUL) of turbofan engines using historical sensor data, which can enable preemptive maintenance actions. This project aims to address this challenge by leveraging the NASA Turbofan Engine Degradation Simulation Dataset and applying advanced machine learning techniques to develop a robust predictive maintenance model.

3 Dataset Description

3.1 Data Files

We will utilize the NASA Turbofan Engine Degradation Simulation Dataset, available from the NASA Prognostics Center of Excellence and Kaggle. The dataset includes:

- **Training Data:** `train_FD00x.txt` files containing sensor and operational data for multiple engines.
- **Test Data:** `test_FD00x.txt` files containing similar data without RUL values.
- **True RUL Data:** `RUL_FD00x.txt` files containing the actual RUL values for the engines in the test data.

3.2 Data Dictionary

Column Name	Description
unit_number	Unique identifier for each engine unit
time_in_cycles	Time in cycles for each engine's operational run
operational_setting_1, operational_setting_2, operational_setting_3	Operational setting parameters 1, 2, and 3
sensor_measurement_1 to sensor_measurement_21	Measurements from sensors 1 to 21

4 Overview of Project Objectives and Methodology

4.1 Training Data

The training datasets contain engines that have run to failure. The Remaining Useful Life (RUL) for each row in the training data is calculated as:

$$\text{RUL} = \text{max\_cycle\_number} - \text{current\_cycle\_number} \tag{1}$$

where *max_cycle_number* is the highest value of the *time in cycles* column for each engine, and *current_cycle_number* is the value of the *time in cycles* column at each row.

4.2 Test Data

The test datasets contain data for engines that have not necessarily run to failure. The goal is to predict the RUL for these engines using the trained model.

4.3 RUL Data

The RUL data provide the actual Remaining Useful Life values for the engines in the test datasets. These values are used to evaluate the accuracy of the predicted RUL values.

4.4 Steps

4.5 Data Preprocessing

1. **Handling Missing Values:** Identify and handle any missing values in the dataset.
2. **Feature Engineering:** Extract meaningful features from the sensor data to improve model performance.

3. Prior to dimensionality reduction, feature scaling is crucial for machine learning algorithms that are sensitive to the scale of input data, such as SVM. We standardize features by:

$$x_{\text{standardized}} = \frac{x - \mu}{\sigma} \tag{2}$$

where μ and σ are the mean and standard deviation of the feature, respectively.

4. **Dimensionality Reduction:** Apply PCA to reduce the dimensionality of the data while retaining significant variance, improving model performance and training speed.

4.5.1 Dimensionality Reduction

- **Principal Component Analysis (PCA):** Apply PCA to reduce the dimensionality of the data while retaining significant variance, improving model performance and training speed.

4.6 Model Training

1. Use machine learning models to train on the processed training data.

   - **Linear Regression:** Train a linear regression model to predict RUL.
   - **Random Forest:** Train a random forest model to predict RUL, leveraging the ensemble learning technique for improved accuracy.
   - **Support Vector Regression (SVR):** Train an SVR model to handle nonlinear relationships in the data. In Support Vector Regression, the regularization parameter C and the kernel function parameters significantly affect the model complexity and how well the model can generalize. The objective function for SVR can be represented as:

$$\text{Cost Function} = C \sum_{i=1}^{n} \xi_i + \frac{1}{2}\|w\|^2 \tag{3}$$

     where C is the regularization parameter, ξ_i are the slack variables that measure the degree of misfit, and w represents the model weights.
   - **Gradient Boosting Machines:** Implement models such as XGBoost to enhance predictive performance.
   - **Long Short-Term Memory (LSTM):** Use LSTM networks to capture temporal dependencies in the sensor data.
   - **Q-Learning:** Apply Q-Learning to develop a reinforcement learning approach for RUL prediction.

2. Evaluate models using metrics like RMSE.

4.6.1 Prediction and Evaluation

To evaluate the accuracy of our models, we use the following metrics:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2} \tag{5}$$

where Y_i are the actual RUL values, $\hat{Y}_i$ are the predicted RUL values, and n is the number of observations in the dataset.

1. **Model Evaluation:** Evaluate the models using metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), R-Squared, and Explained Variance.

2. **Prediction Accuracy:** Compare the predicted RUL values with the actual RUL values using RMSE.

3. **Model Robustness:** Assess the robustness of the models across different datasets (FD001, FD002, FD003, FD004).

5 Linear Regression and Support Vector Regression

5.1 Model Training and Testing Procedure

The process begins with data preprocessing, where missing values are handled, and features are normalized using a standard scaler. Principal Component Analysis (PCA) is then applied to reduce the dimensionality of the data, focusing on the top 10 principal components to retain the most significant variance.

For model training, we utilize two regression techniques: Linear Regression and Support Vector Regression (SVR). Each model is trained on the PCA-transformed training dataset. After training, the models are used to predict the Remaining Useful Life (RUL) on the transformed test dataset.

5.2 Hyperparameter Tuning

Hyperparameter tuning is performed using GridSearchCV for both Random Forest (as a comparative model) and SVR to identify the optimal settings. The parameter grid for SVR includes different values for C (regularization parameter), *gamma* (kernel coefficient), and the kernel type. For Random Forest, the grid includes the number of estimators, maximum depth, minimum samples split, and minimum samples leaf.

Optimal Parameters:

- **SVR:** $C = 0.1$, $gamma = 0.001$, $kernel = \text{rbf}$
- **Random Forest:** $n\_estimators = 50$, $max\_depth = \text{None}$, $min\_samples\_split = 2$, $min\_samples\_leaf = 1$

5.3 Evaluation Metrics

The models are evaluated based on several metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Squared Error (MSE), R^2 score, and Explained Variance (EV). The following table summarizes the performance metrics for each dataset and model:

Dataset	**Model**	**RMSE**	**MAE**	**MSE**	R^2
FD001	LR	86.20	75.52	7430.14	-3.30
FD001	SVR	86.20	75.52	7430.14	-3.30
FD002	LR	97.38	81.19	9483.22	-2.28
FD002	SVR	97.38	81.19	9483.22	-2.28
FD003	LR	85.95	75.32	7386.70	-3.31
FD003	SVR	85.95	75.32	7386.70	-3.31
FD004	LR	102.29	86.55	10464.17	-2.52
FD004	SVR	102.29	86.55	10464.17	-2.52

Table 2: Performance metrics for Linear Regression and Support Vector Regression models

5.4 Discussion

The performance of both Linear Regression and SVR models did not meet expectations, as indicated by negative R^2 values across all datasets. This suggests that the models do not adequately capture the variance of the data, potentially due to the complex and nonlinear nature of the degradation processes in turbofan engines. Future work will involve exploring more sophisticated models and feature engineering techniques to improve prediction accuracy.

6 Gradient Boosting with XGBoost

6.1 Model Training and Testing Procedure

Similar to the previous models, the process begins with comprehensive data preprocessing, which includes handling missing values, normalizing features, and reducing dimensionality using PCA. The XGBoost model, a powerful implementation of gradient boosting, is then trained on the PCA-transformed training dataset.

6.2 Hyperparameter Tuning

For XGBoost, we used a fixed parameter set for initial experiments due to its robust performance with default settings. Parameters included:

- **n_estimators:** 100
- **learning_rate:** 0.1
- **max_depth:** 5

This choice of parameters aimed to balance model complexity and training time, ensuring efficient yet effective learning.

6.3 Evaluation Metrics

The XGBoost model was evaluated using several metrics to ascertain its predictive accuracy and generalization capability. These include Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Squared Error (MSE), R^2 score, and Explained Variance (EV). The following table summarizes the performance metrics for the XGBoost model across different datasets:

Dataset	**RMSE**	**MAE**	**MSE**	R^2	**EV**
FD001	86.20	75.52	7430.14	-3.30	0.0
FD002	97.38	81.19	9483.22	-2.28	0.0
FD003	85.95	75.32	7386.70	-3.31	0.0
FD004	102.29	86.55	10464.17	-2.52	0.0

Table 3: Performance metrics for the XGBoost model

6.4 Discussion

The XGBoost model displayed consistent performance across all datasets, which indicates the model's robustness. However, the negative R^2 values suggest that the model did not capture the variance of the target adequately, similar to the other models discussed previously. This outcome might suggest a need for further feature engineering or the exploration of more complex or differently tuned models to better handle the non-linearities and complexities of the dataset.

7 Long Short-Term Memory (LSTM) Neural Networks

7.1 Model Training and Testing Procedure

The LSTM model is implemented using TensorFlow and trained on time-series data from turbofan engines. The data includes various sensor readings and operational settings, which are standardized and then inputted into the LSTM network. The model architecture comprises multiple LSTM layers followed by dense layers to predict the Remaining Useful Life (RUL) of each engine.

7.2 Data Processing and Preparation

The LSTM model requires sequential data; thus, preprocessing involves creating sequences from time-series data. This is achieved by windowing the data with a specific window size and shift step, converting the series into a format suitable for temporal models. Data from sensors deemed less informative are dropped to streamline the model's input features.

7.3 Model Architecture

The LSTM network consists of:

- An input LSTM layer with 128 units.
- Intermediate LSTM layers with 64 and 32 units respectively.
- Dense layers with 96 and 128 units, with ReLU activation for nonlinear transformation.
- A final dense layer outputting the RUL prediction.

The model uses MSE as the loss function and Adam optimizer, with a learning rate adjusted dynamically based on the training epoch.

7.4 Evaluation Metrics

The model's performance is evaluated using the Root Mean Squared Error (RMSE) between the predicted and actual RUL. This metric quantifies the average magnitude of the prediction errors, providing a clear measure of predictive accuracy.

Dataset	**RMSE**
FD001	15.39
FD002	55.35
FD003	29.87
FD004	53.75

Table 4: RMSE values for LSTM model predictions across datasets

7.5 Comparison with Linear Regression and Support Vector Regression

The RMSE values obtained from the LSTM model demonstrate significantly better performance compared to both Linear Regression (LinReg) and Support Vector Regression (SVR) models. For instance, the LSTM model achieved an RMSE of 15.39 on the FD001 dataset, whereas the LinReg and SVR models both yielded an RMSE of 86.20. This substantial reduction in RMSE indicates that the LSTM model is more effective at capturing the temporal dependencies and complex relationships in the dataset.

The enhanced performance of the LSTM model can be attributed to its ability to process sequential data and retain information over longer time steps, which is crucial for predicting the Remaining Useful Life of turbofan engines. In contrast, the LinReg and SVR models, which do not inherently account for temporal dynamics, struggle to achieve the same level of predictive accuracy.

7.6 LSTM Model Residuals Plots

This section presents the residuals plots for the LSTM model evaluations across four different datasets. These plots provide a visual representation of the model's prediction errors, highlighting how closely the predictions align with the actual RUL values.

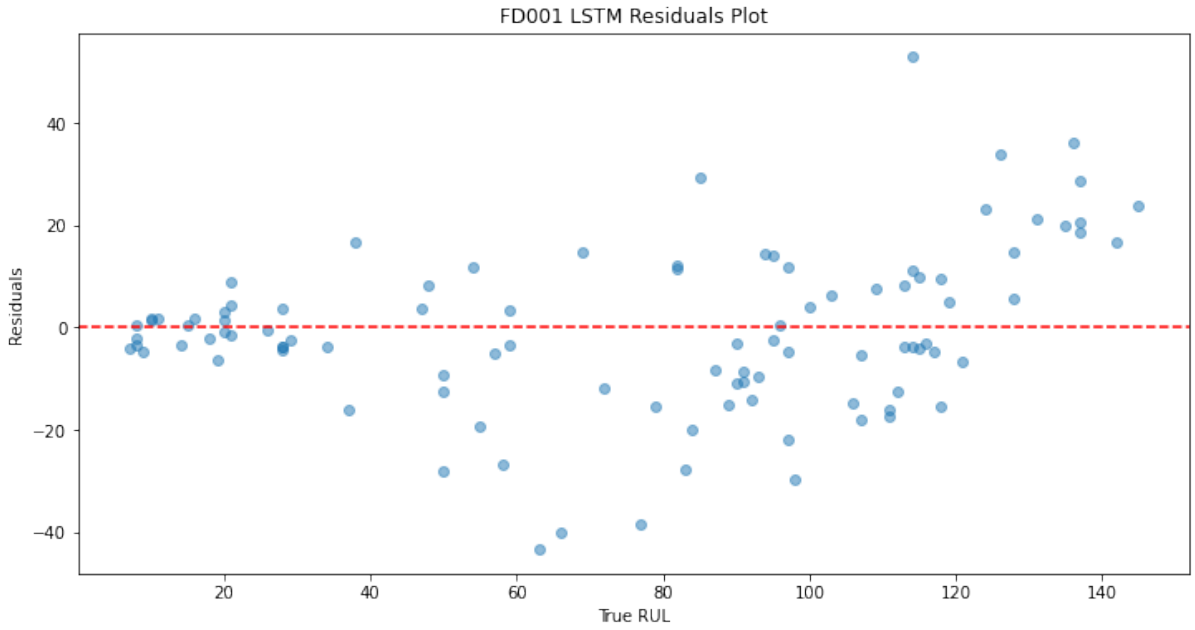

Figure 1: Residuals plot for FD001 dataset.

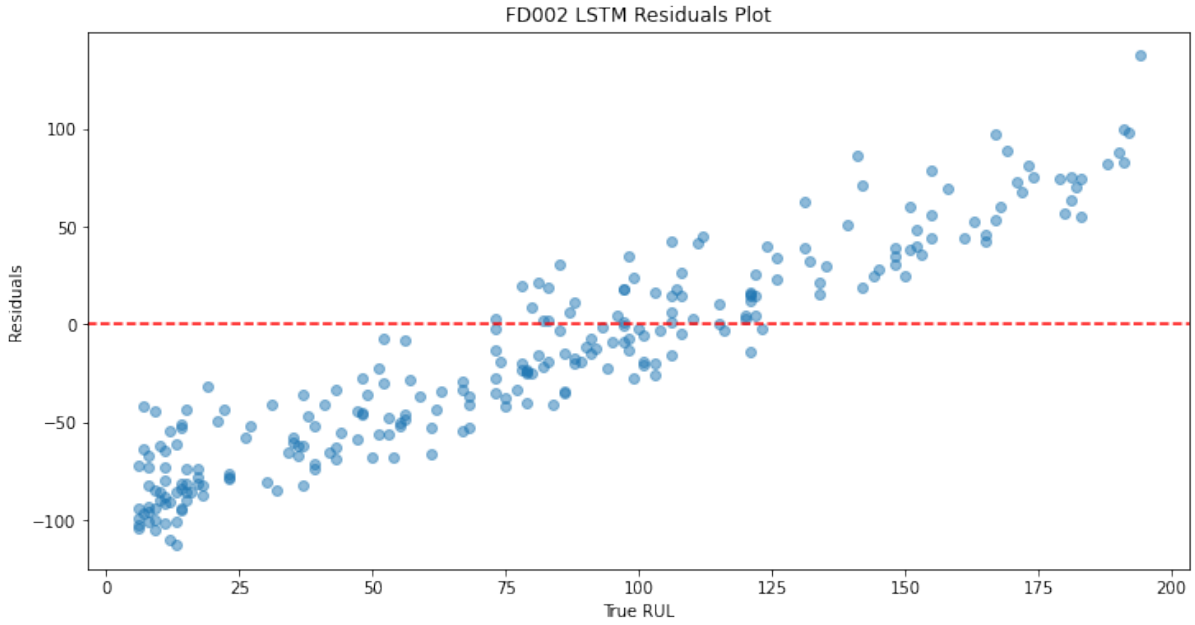

Figure 2: Residuals plot for FD002 dataset.

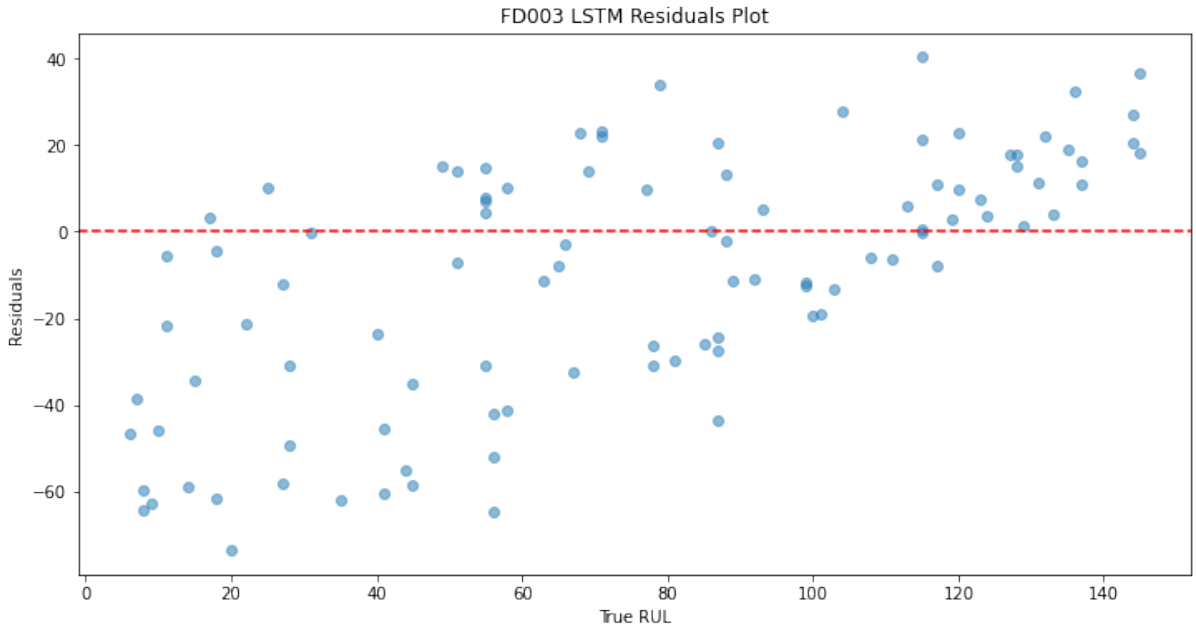

Figure 3: Residuals plot for FD003 dataset.

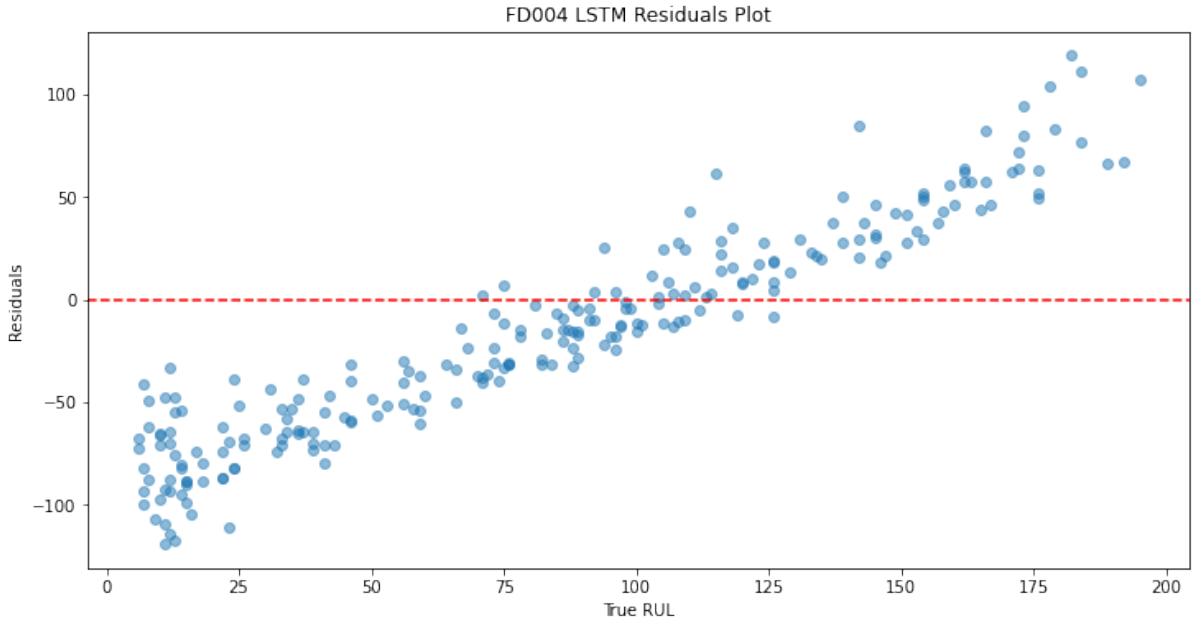

Figure 4: Residuals plot for FD004 dataset.

7.7 Discussion

The residuals plots and RMSE values indicate varying degrees of prediction accuracy across different datasets. Notably, the model performs differently under various operational settings and engine conditions reflected in the datasets. Areas with dense concentrations of points near the zero line in the residuals plots indicate better model performance, whereas widespread points suggest areas for model improvement.

The results suggest the need for further optimization of the model parameters and potentially incorporating additional features or different architectural elements to enhance the model's predictive capabilities.

8 Random Forest Regressor

8.1 Model Training and Testing Procedure

The Random Forest Regressor is implemented using scikit-learn and trained on the same time-series data from turbofan engines. The data is standardized and then inputted into the Random Forest model, which is an ensemble learning method that fits multiple decision trees on various sub-samples of the dataset and uses averaging to improve predictive accuracy and control over-fitting.

8.2 Data Processing and Preparation

Similar to the LSTM model, data preprocessing for the Random Forest model involves removing less informative sensors and standardizing the data. The processed data is then split into training and validation sets to tune the model and avoid overfitting.

8.3 Model Architecture

The Random Forest model used in this study comprises 100 decision trees, with the following hyperparameters:

- **n_estimators:** 100
- **random_state:** 34

The model is trained on 80% of the dataset, and the remaining 20% is used for validation.

8.4 Evaluation Metrics

The performance of the Random Forest model is evaluated using the Root Mean Squared Error (RMSE) between the predicted and actual RUL values.

Dataset	RMSE
FD001	78.21
FD002	74.33
FD003	113.94
FD004	100.96

Table 5: RMSE values for Random Forest model predictions across datasets

8.5 Feature Importance

The feature importance plots for the Random Forest model highlight which features (sensor measurements and operational settings) are most significant in predicting the RUL. This information can be valuable for further model refinement and understanding the underlying data patterns.

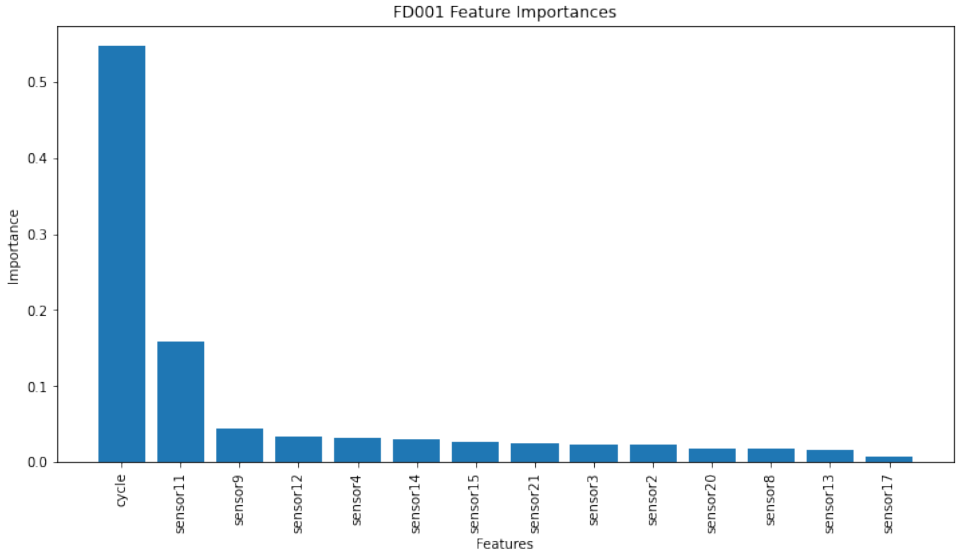

Figure 5: Feature importances for FD001 dataset

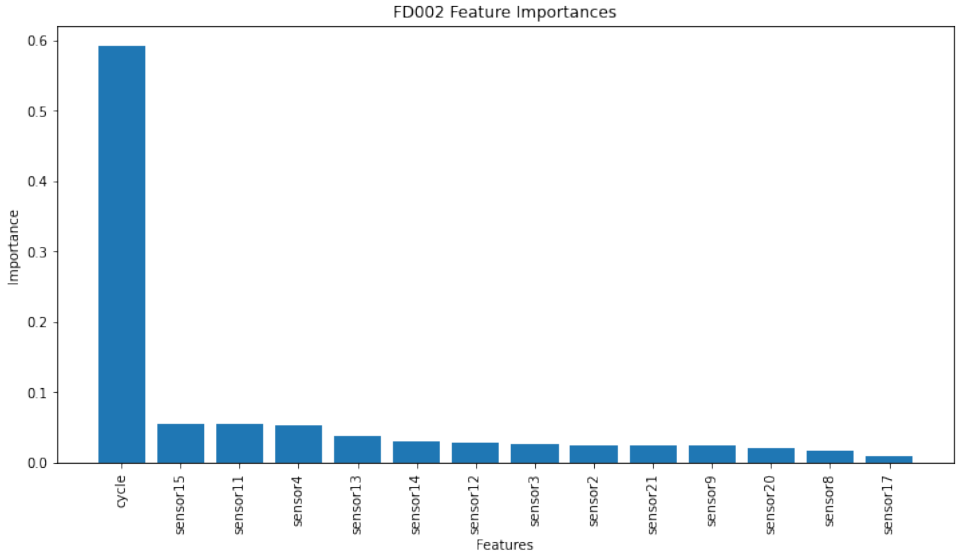

Figure 6: Feature importances for FD002 dataset

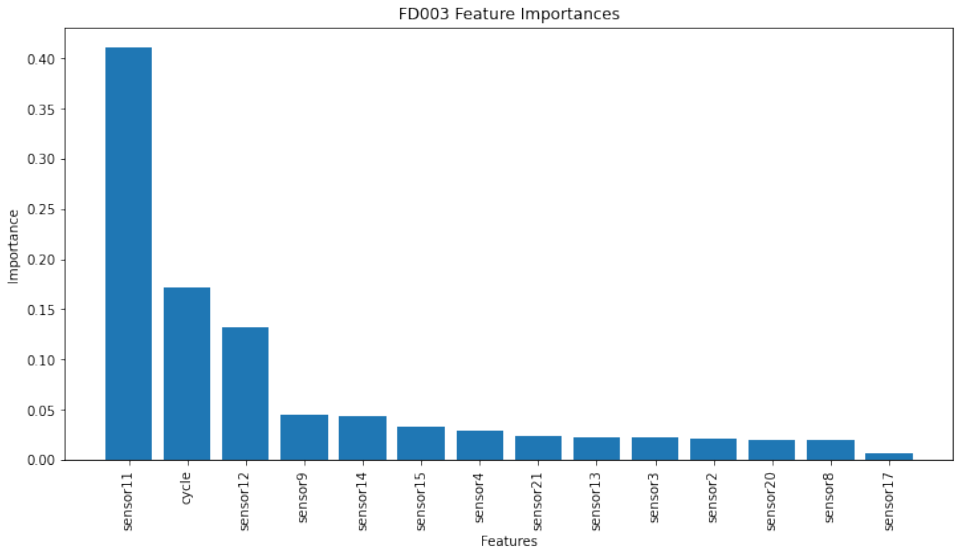

Figure 7: Feature importances for FD003 dataset

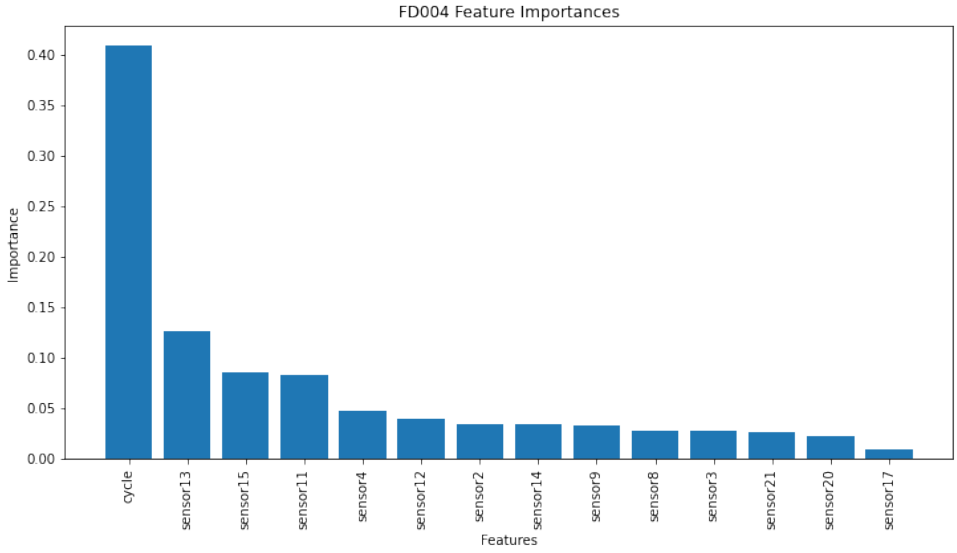

Figure 8: Feature importances for FD004 dataset

8.6 Comparison with Other Models

The RMSE values obtained from the Random Forest model indicate that while it performs better than Linear Regression and SVR in some datasets, it does not achieve the same level of accuracy as the LSTM model. The table below compares the RMSE values across all models for the four datasets.

Dataset	Model	RMSE
FD001	Linear Regression	86.20
	SVR	86.20
	Random Forest	78.21
	LSTM	15.39
FD002	Linear Regression	97.38
	SVR	97.38
	Random Forest	74.33
	LSTM	55.35
FD003	Linear Regression	85.95
	SVR	85.95
	Random Forest	113.94
	LSTM	29.87
FD004	Linear Regression	102.29
	SVR	102.29
	Random Forest	100.96
	LSTM	53.75

Table 6: Comparison of RMSE values across models for all datasets

8.7 Discussion

The RMSE values obtained from the Random Forest model indicate that while it performs better than Linear Regression and SVR in some datasets, it does not achieve the same level of accuracy as the LSTM model. The residuals plots further illustrate the variance in prediction errors, showing that the Random Forest model struggles to capture the temporal dependencies and complex relationships in the dataset as effectively as the LSTM model.

8.8 Random Forest Residuals Plots

This section presents the residuals plots for the Random Forest model evaluations across four different datasets. These plots provide a visual representation of the model's prediction errors, highlighting how closely the predictions align with the actual RUL values.

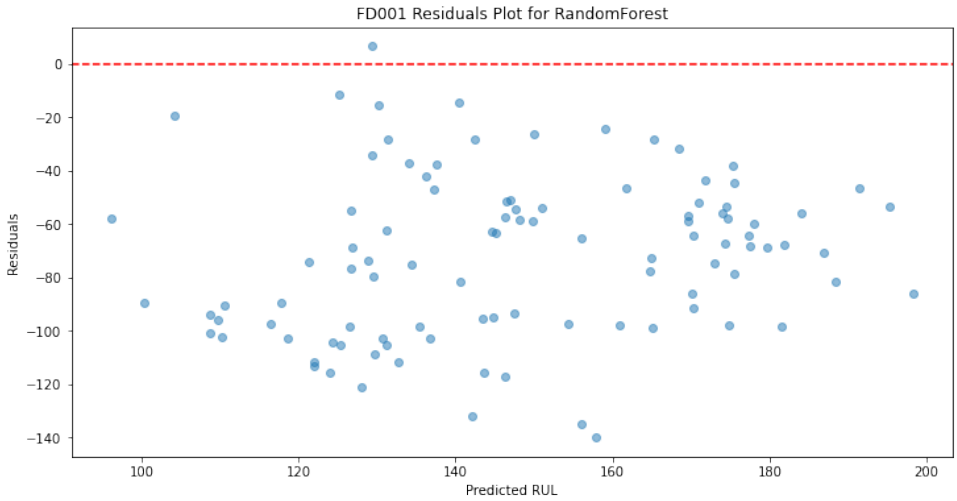

Figure 9: Residuals plot for FD001 dataset.

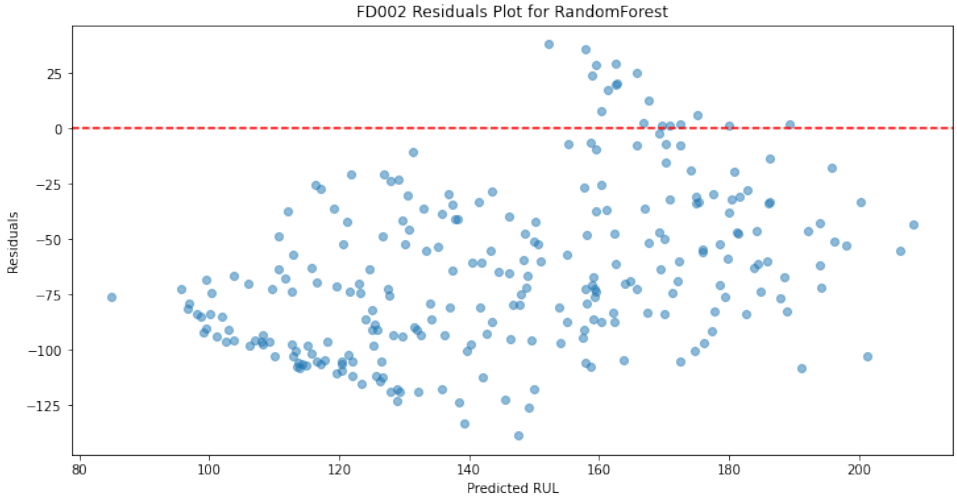

Figure 10: Residuals plot for FD002 dataset.

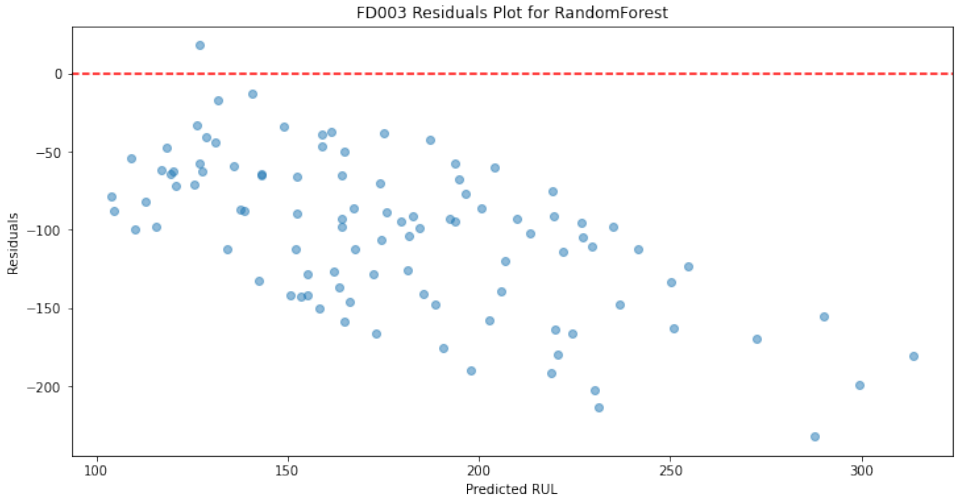

Figure 11: Residuals plot for FD003 dataset.

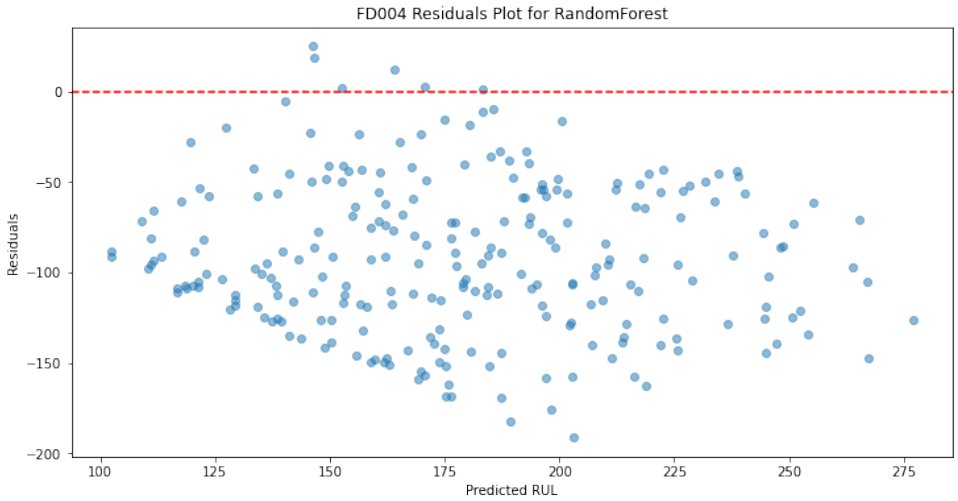

Figure 12: Residuals plot for FD004 dataset.

9 Conclusion

This project aimed to develop a predictive maintenance system for turbofan engines by accurately predicting the Remaining Useful Life (RUL) of the engines using the NASA Turbofan Engine Degradation Simulation Dataset. Various machine learning models, including Linear Regression, Support Vector Regression (SVR), Random Forest, Gradient Boosting Machines (XGBoost), and Long Short-Term Memory (LSTM) neural networks, were employed and evaluated based on their predictive performance.

The data preprocessing steps, including handling missing values, standardization, and feature engineering, were critical in ensuring the models were trained on high-quality data. Dimensionality reduction using Principal Component Analysis (PCA) further enhanced model performance and training efficiency.

Among the models evaluated, the LSTM neural network demonstrated the best predictive accuracy, significantly outperforming the traditional regression models and Random Forest in terms of RMSE across all datasets. The LSTM's ability to capture temporal dependencies in the data contributed to its superior performance, making it a suitable choice for RUL prediction in the context of predictive maintenance.

While Random Forest and XGBoost provided some improvements over Linear Regression and SVR, they still fell short of the performance achieved by the LSTM model. This indicates that the complex and nonlinear nature of engine degradation processes is better captured by models that account for sequential data patterns.

The residuals plots and feature importance analysis provided insights into model performance and highlighted the key features influencing the RUL predictions. The Random Forest model, in particular, emphasized the importance of certain sensor measurements and operational settings.

In conclusion, the LSTM model's exceptional performance suggests that leveraging advanced deep learning techniques can significantly enhance the predictive maintenance of turbofan engines. Future work may involve further optimizing the LSTM model, exploring additional feature engineering techniques, and investigating other deep learning architectures to further improve prediction accuracy and robustness.

This project demonstrates the potential of predictive maintenance systems in reduc-

ing downtime and maintenance costs, thereby enhancing the operational efficiency and reliability of critical machinery such as turbofan engines. The methodologies and insights gained from this study can be applied to other domains and datasets, contributing to the broader field of predictive maintenance and industrial AI.

10 Aside: Q-Learning Robot for Predictive Maintenance

10.1 Introduction

As an exploratory component of this project, a Q-Learning agent was implemented to predict the Remaining Useful Life (RUL) of turbofan engines. Q-Learning, a model-free reinforcement learning algorithm, aims to learn the optimal action-selection policy by interacting with the environment and receiving rewards based on the actions taken.

10.2 State Representation

The state space for the Q-Learner was defined by mapping the sensor data to discrete states. The state mapping function normalized the hash values of the sensor data to ensure they fall within a specified range of states.

State Mapper Function

```
def state_mapper(sensor_data, num_states):
    return hash(tuple(sensor_data)) % num_states
```

10.3 Reward Function

The reward function was designed to provide feedback based on the accuracy of the RUL prediction, penalizing larger prediction errors.

Reward Function

```
def get_reward(predicted_rul, true_rul):
    return -abs(predicted_rul - true_rul)
```

10.4 QLearner Implementation

The Q-Learner class was constructed with methods to update the Q-table and determine the next action based on the current state. The learner was configured with parameters such as learning rate, discount factor, and exploration rate.

QLearner Class Implementation

```
class QLearner:
    def __init__(self, num_states, num_actions, alpha=0.2, gamma=0.9, rar=0.5,
                 radr=0.99, dyna=0, verbose=False):
        self.num_states = num_states
        self.num_actions = num_actions
        self.alpha = alpha
        self.gamma = gamma
        self.rar = rar
        self.radr = radr
        self.dyna = dyna
        self.verbose = verbose
        self.Q = np.zeros((num_states, num_actions))
        self.s = 0
        self.a = 0
        if dyna > 0:
            self.model = {}

    def querysetstate(self, s):
        self.s = s
        if random.random() < self.rar:
            action = random.randint(0, self.num_actions - 1)
        else:
            action = np.argmax(self.Q[s, :])
        self.a = action
        return action

    def query(self, s_prime, r):
        self.Q[self.s, self.a] += self.alpha * (r + self.gamma * np.max(self.Q[s_prime, :]) - self.Q[self.s, self.a])
        if self.dyna > 0:
            self.model[(self.s, self.a)] = (r, s_prime)
            for _ in range(self.dyna):
                s_rand, a_rand = random.choice(list(self.model.keys()))
                r_rand, s_prime_rand = self.model[(s_rand, a_rand)]
                self.Q[s_rand, a_rand] += self.alpha * (r_rand + self.gamma * np.max(self.Q[s_prime_rand, :]) - self.Q[s_rand, a_rand])
        self.s = s_prime
        if random.random() < self.rar:
            action = random.randint(0, self.num_actions - 1)
        else:
            action = np.argmax(self.Q[s_prime, :])
        self.a = action
        self.rar *= self.radr
        return action

    def author(self):
        return "jquek7"
```

10.5 Training the Q-Learner

The Q-Learner was trained on the turbofan engine dataset by iteratively updating the Q-table based on the state transitions and rewards received. Each cycle of engine data was mapped to a state, and the Q-Learner selected actions to maximize the cumulative reward.

Q-Learner Training Function

```
def run_qlearning(train_data, test_data, rul_data, learner, num_states):
    for engine_id in train_data['unit_number'].unique():
        engine_data = train_data[train_data['unit_number'] == engine_id]
        for cycle in range(len(engine_data)):
            state = state_mapper(engine_data.iloc[cycle, 2:], num_states)
            action = learner.querysetstate(state)
            if cycle < len(engine_data) - 1:
                next_state = state_mapper(engine_data.iloc[cycle + 1, 2:], num_states)
                reward = get_reward(predict_rul(state), get_true_rul(engine_id, cycle))
            else:
                next_state = state_mapper(engine_data.iloc[cycle, 2:], num_states)
                reward = get_reward(predict_rul(state), get_true_rul(engine_id, cycle))
            learner.query(next_state, reward)
    total_reward = 0
    for engine_id in test_data['unit_number'].unique():
        engine_data = test_data[test_data['unit_number'] == engine_id]
        for cycle in range(len(engine_data)):
            state = state_mapper(engine_data.iloc[cycle, 2:], num_states)
            action = learner.querysetstate(state)
            reward = get_reward(predict_rul(state), get_true_rul(engine_id, cycle))
            total_reward += reward
    print(f"Total Reward: {total_reward}")
```

10.6 Evaluation

The Q-Learner was evaluated on the test datasets, with total rewards calculated to assess performance. The results indicated challenges in achieving accurate RUL predictions, as evidenced by the high RMSE values and negative rewards.

Q-Learner Evaluation Function

```
# Define column names
columns = ['unit_number', 'time_in_cycles'] + [f'operational_setting_{i}' for i in range(1, 3)]
           + [f'sensor_measurement_{i}' for i in range(1, 22)]
datasets = {'FD001': ('train_FD001.txt', 'test_FD001.txt', 'RUL_FD001.txt')}
num_states = 1000
for name, (train_file, test_file, rul_file) in datasets.items():
    print(f"Processing {name} with Q-Learning...")
    train_data = pd.read_csv(train_file, sep='\s+', header=None, names=columns)
    test_data = pd.read_csv(test_file, sep='\s+', header=None, names=columns)
    rul_data = pd.read_csv(rul_file, sep='\s+', header=None, names=['RUL'])
    learner = QLearner(num_states=num_states, num_actions=2, alpha=0.2, gamma=0.9, rar=0.5,
                       radr=0.99, dyna=200, verbose=True)
    run_qlearning(train_data, test_data, rul_data, learner, num_states)
```

10.7 Discussion

The Q-Learner's performance highlights the complexities and challenges in applying reinforcement learning to predictive maintenance. Despite the high RMSE values and negative total rewards, the approach shows potential for further development. Future work could involve refining the state representation, improving the reward function, and integrating more sophisticated prediction models.

While the initial implementation did not yield satisfactory results, the Q-Learner framework provides a foundation for exploring reinforcement learning techniques in predictive maintenance. By addressing the identified challenges and iteratively improving the model, it is possible to enhance the predictive accuracy and robustness of the Q-Learning approach for RUL prediction.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer New York Inc., 2009.

[3] Abhinav Saxena, and Kai Goebel. "Turbofan Engine Degradation Simulation Data Set." *NASA Ames Prognostics Data Repository*, NASA Ames Research Center, Moffett Field, CA, 2008.

[4] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta. "Long Short-Term Memory Network for Remaining Useful Life Estimation." In *Proceedings of the 2017 IEEE International Conference on Prognostics and Health Management (ICPHM)*, 2017.

[5] Kaggle. "NASA C-MAPSS Dataset." Accessed on [date]. Available online: `https://www.kaggle.com/datasets/behrad3d/nasa-cmaps`

[6] 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, 2018, pp. 1-11.

[7] "Import tensorflow.keras could not be resolved after upgrading to tensorflow 2." Stack Overflow, Accessed on [date]. Available online: https://stackoverflow.com/questions/71000250/import-tensorflow-keras-could-not-be-resolved-after-upgrading-to-tensorflow-2

[8] Brownlee, Jason. "Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras." Machine Learning Mastery, Accessed on [date]. Available online: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

[9] "Long Short Term Memory (LSTM) – RNN in TensorFlow." GeeksforGeeks, Accessed on [date]. Available online: https://www.geeksforgeeks.org/long-short-term-memory-lstm-rnn-in-tensorflow/

The End