Sri Sri University

Predictive Maintenance for Aircraft Engines & Aviation Safety

LOW-LEVEL DESIGN
Predictive Maintenance

Baccalaureus Technologiae
Faculty Of Engineering & Technology

SUBMITTED BY
Subham Nayak | FET-BDS-2022-26-019
Indigibilli Harshit | FET-BDS-2022-26-015
Dhruti Deepam Mohapatra | FET-BDS-2022-26-005
Khushi Singhania | FET-BDS-2022-26-013
Lokesh Patra | FET-BDS-2022-26-20

Under the Guidance of
Dr. Pradipta Kumar Mishra
Asst. Professor
Faculty Of Engineering & Technology
Sri University

1.Introduction

1.1 Scope of the document:

This document outlines the low-level design (LLD) of a predictive maintenance system for aircraft engines. It covers the design aspects of machine learning models such as Linear Regression, Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN), Gradient Boosting (GBU), and Random Forest (RF). Additionally, it includes data preprocessing, model training, evaluation, and visualization techniques required for developing a proactive maintenance solution.

1.2. Intended Audience:

The intended audience for this document includes data scientists, machine learning engineers, aviation maintenance experts, and project stakeholders involved in the development and implementation of the predictive maintenance system.

1.3. System Overview:

The predictive maintenance system aims to leverage machine learning techniques to analyse sensor data, flight patterns, and historical maintenance records to predict the likelihood of engine failure in aircraft. By identifying potential issues before they occur, the system enables proactive maintenance scheduling, reducing unplanned downtime and enhancing aviation safety.

2. Low-Level System Design

2.1. Model Architecture:

This section describes the architectural design of the machine learning models employed in the predictive maintenance system, including Linear Regression, LSTM, RNN, GBU, and RF. Architectural diagrams highlighting the layers, connections, and flow of information will be provided. The architecture should clearly illustrate how data flows through each model and how predictions are made.

2.2. Model Training:

Here, we detail the training process for each machine learning model, including optimization algorithms, hyperparameter tuning, and cross-validation techniques. Code snippets for training each model will be provided, along with explanations of key parameters and techniques used.

2.3. Model Evaluation:

This section presents the evaluation metrics and methodologies used to assess the performance of each predictive maintenance model, including accuracy, precision, recall, and F1 score. Visualizations comparing model performance will be included.

3. Data Design

3.1. Data Loading and Concatenation:

This section covers the loading and concatenation of datasets containing information about aircraft engine performance. Data preprocessing steps, such as feature engineering and labelling, will be described.

Loading Datasets

- **df_train, df_test, df_truth:** Reading datasets from Excel files that contain information about aircraft engine performance.

```
df_train = pd.read_excel("E:\MinoR IBM\PM_train.xlsx")
df_test = pd.read_excel("E:\MinoR IBM\PM_test.xlsx")
df_truth = pd.read_excel("E:\MinoR IBM\PM_truth.xlsx")
```

Data Concatenation

- **Concatenation:** Combining `df_train` and `df_test` into a single dataframe (`df`) to work with the entire dataset.

```
df = pd.concat([df_train, df_test], ignore_index=True)
df.head()
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392.0	2388	100	39.06	23.4190
1	1	2	0.0019	-0.0003	100	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392.0	2388	100	39.00	23.4236
2	1	3	-0.0043	0.0003	100	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390.0	2388	100	38.95	23.3442
3	1	4	0.0007	0.0000	100	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392.0	2388	100	38.88	23.3739
4	1	5	-0.0019	-0.0002	100	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393.0	2388	100	38.90	23.4044

5 rows × 26 columns

3.2. Feature Engineering

Feature engineering techniques, such as calculating Remaining Useful Life (RUL), will be detailed. Feature columns for model training will be defined.

Feature Engineering

RUL

Remaining Useful Life (RUL) is a crucial concept in predictive maintenance, representing the estimated operational lifespan or cycles remaining for an asset before it is expected to fail. In the context of aircraft engines, RUL helps anticipate the point at which an engine may no longer meet performance requirements. By calculating RUL, organizations can proactively plan maintenance activities, optimize resource allocation, and minimize downtime. This predictive approach enables timely interventions, reducing the risk of unexpected failures and enhancing the overall efficiency and reliability of the asset. RUL serves as a key metric for making informed decisions in maintenance and operational strategies.

- **Remaining Useful Life (RUL):** Calculated by subtracting the current cycle from the maximum cycle for each engine.
- **Feature Columns:** Defined a list of feature columns for model training.

```
df['RUL'] = df.groupby('id')['cycle'].transform(max) - df['cycle']
feature_columns = ['setting1', 'setting2', 'setting3', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12',
```

4. Model Implementation and Evaluation

4.1. Linear Regression Model

Code for implementing and training the Linear Regression model will be provided, along with model evaluation metrics such as accuracy and mean squared error (MSE).

Prediction and Evaluation

- **Making Predictions:** Applying the trained model to predict RUL on the test set.
- **Binary Conversion:** Converting RUL values to binary labels (0 or 1) for evaluation as a classification problem.
- **Evaluation Metrics:** Calculating accuracy and classification report metrics for assessing the model's performance.

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# y_test contains the true labels and y_pred contains the predicted labels
y_test_binary = (y_test <= threshold).astype(int)
y_pred_binary = (y_pred <= threshold).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test_binary, y_pred_binary)
print(f'Accuracy: {accuracy}')
print(classification_report(y_test_binary, y_pred_binary))
```

```
Accuracy: 0.9455019957015659
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      5883
           1       0.91      0.48      0.63       631

    accuracy                           0.95      6514
   macro avg       0.93      0.74      0.80      6514
weighted avg       0.94      0.95      0.94      6514
```

4.2. LSTM Model

Code for implementing and training the LSTM model will be provided, along with model evaluation metrics.

```
def generate_sequences(df, sequence_length, feature_cols):
    sequences = []
    for engine_id in df['id'].unique():
        engine_data = df[df['id'] == engine_id][feature_cols].values
        num_samples = len(engine_data)
        for start_idx in range(0, num_samples - sequence_length + 1):
            sequence = engine_data[start_idx:start_idx + sequence_length]
            sequences.append(sequence)
    return np.array(sequences)
```

```
feature_cols = ['s2']  # Choose a single feature for LSTM input
train_sequences = generate_sequences(train_df, sequence_length, feature_cols)
test_sequences = generate_sequences(test_df, sequence_length, feature_cols)
```

```
# LSTM Model
model_lstm = Sequential()
model_lstm.add(LSTM(units=50, input_shape=(sequence_length, len(feature_cols))))
model_lstm.add(Dense(units=1, activation='sigmoid'))
model_lstm.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# Train LSTM model
history_lstm = model_lstm.fit(train_sequences, train_df['failure_within_w1'],
                              epochs=50, batch_size=32, validation_split=0.2)
```

```
Epoch 3/50
394/394 [==============================] - 5s 13ms/step - loss: 0.4300 - accuracy: 0.8448 - val_loss: 0.4098 - val_accuracy:
0.8595
Epoch 4/50
394/394 [==============================] - 5s 13ms/step - loss: 0.4299 - accuracy: 0.8448 - val_loss: 0.4072 - val_accuracy:
0.8595
Epoch 5/50
394/394 [==============================] - 5s 13ms/step - loss: 0.4297 - accuracy: 0.8448 - val_loss: 0.4088 - val_accuracy:
0.8595
Epoch 6/50
394/394 [==============================] - 5s 13ms/step - loss: 0.4296 - accuracy: 0.8448 - val_loss: 0.4064 - val_accuracy:
0.8595
Epoch 7/50
394/394 [==============================] - 5s 13ms/step - loss: 0.4293 - accuracy: 0.8448 - val_loss: 0.4071 - val_accuracy:
0.8595
```

4.3. RNN Model:

Code for implementing and training the RNN model will be provided, along with model evaluation metrics.

```
out_dim = label_set.shape[1] # 1 label/output for one sequence.
features_dim = seq_set.shape[2] # Number of features (1)


print("Features dimension: ", features_dim)
print("Output dimension: ", out_dim)

RNN_fwd = Sequential()

# Add the RNN unit.
# Understand the parameters.
RNN_fwd.add(SimpleRNN(
         input_shape = (sequence_length, features_dim),
         units = 1,
         return_sequences=False))
RNN_fwd.add(Dropout(0.2))

RNN_fwd.add(Dense(units=out_dim, activation='sigmoid'))

# Compile the model.
RNN_fwd.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])

print(RNN_fwd.summary())

# Define the path to save the model.
RNN_fwd_path = '/kaggle/working/RNN_fwd.h5'
```

```
Features dimension:  1
Output dimension:  1
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn (SimpleRNN)      (None, 1)                 3

 dropout (Dropout)           (None, 1)                 0

 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 5 (20.00 Byte)
Trainable params: 5 (20.00 Byte)
```

4.5. Random Forest Model (RF):

Code for implementing and training the RF model will be provided, along with model evaluation metrics.

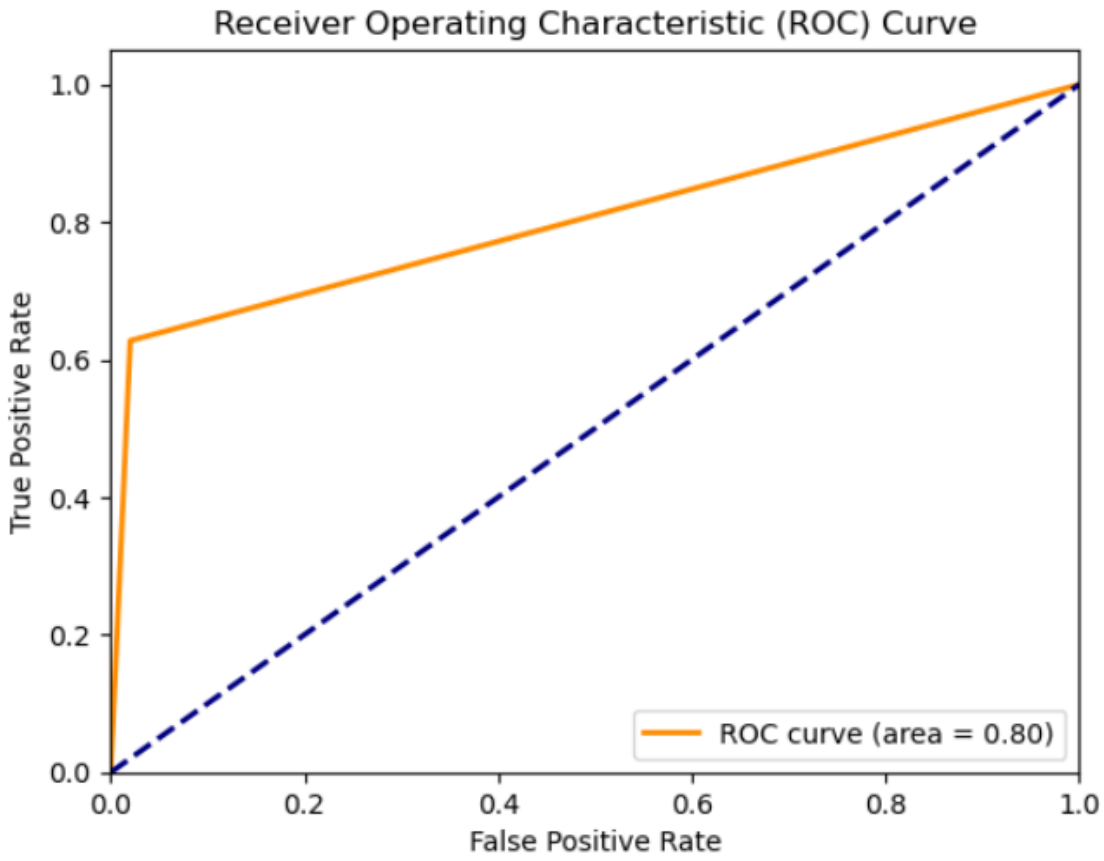
```
# Convert predictions and true labels to binary
y_test_binary = (y_test <= threshold).astype(int)
y_pred_binary = (y_pred <= threshold).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test_binary, y_pred_binary)
print(f'Accuracy: {accuracy* 100}')
print(classification_report(y_test_binary, y_pred_binary))
```

```
Accuracy: 91.83297513048818
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      5376
           1       0.87      0.63      0.73      1138

    accuracy                           0.92      6514
   macro avg       0.90      0.80      0.84      6514
weighted avg       0.92      0.92      0.91      6514
```

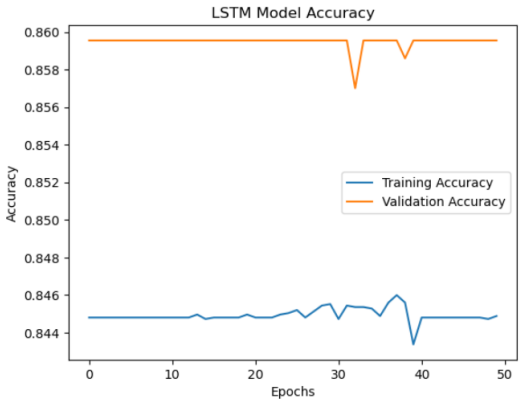


5. Visualization and Comparison

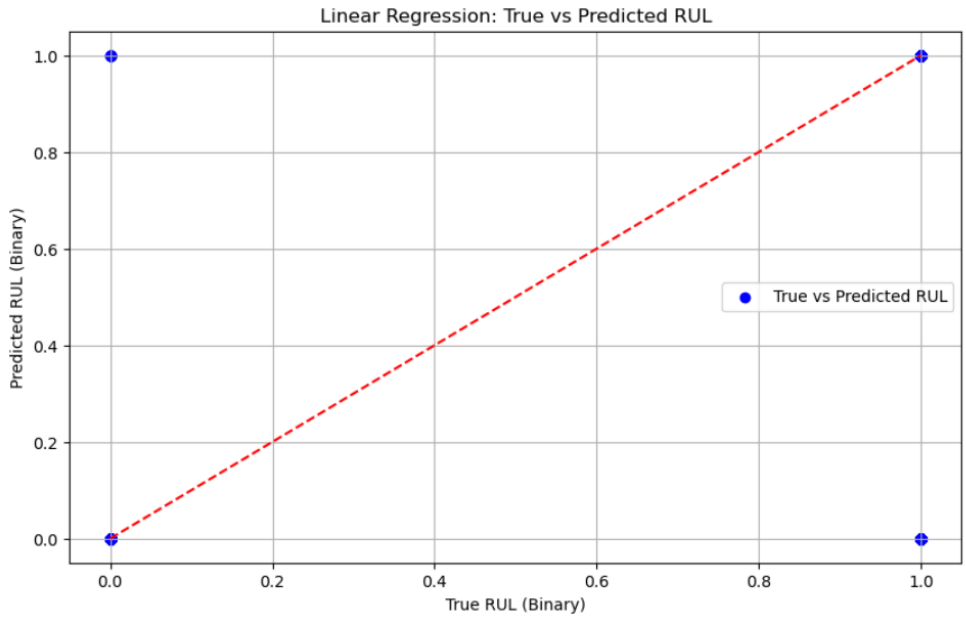
5.1. Model Performance Visualization:

Visualizations comparing the performance of different predictive maintenance models will be provided, including accuracy, precision, recall, and F1 score.

LSTM Model



Linear Regression



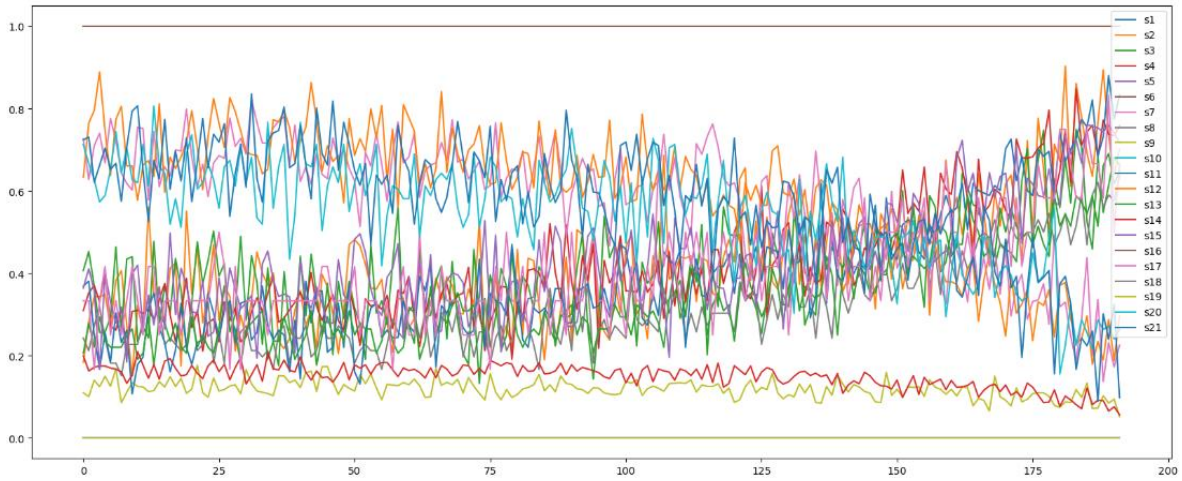
EDA (Exploratory Data Analysis)

```
''' Exploratory Data Analysis [EDA] '''
```

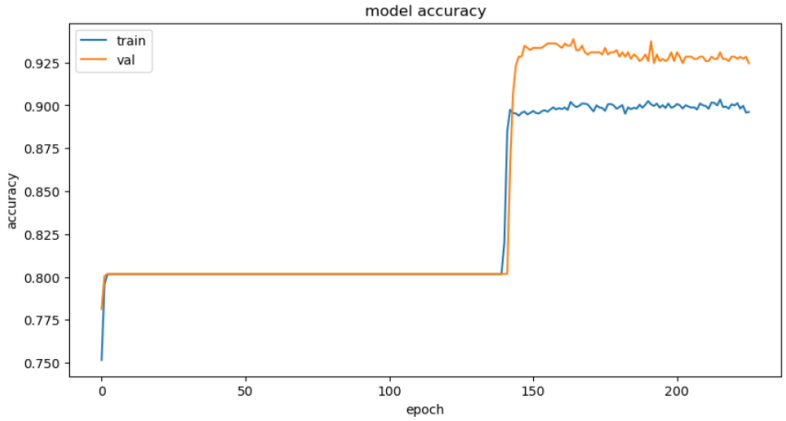
```
' Exploratory Data Analysis [EDA] '
```

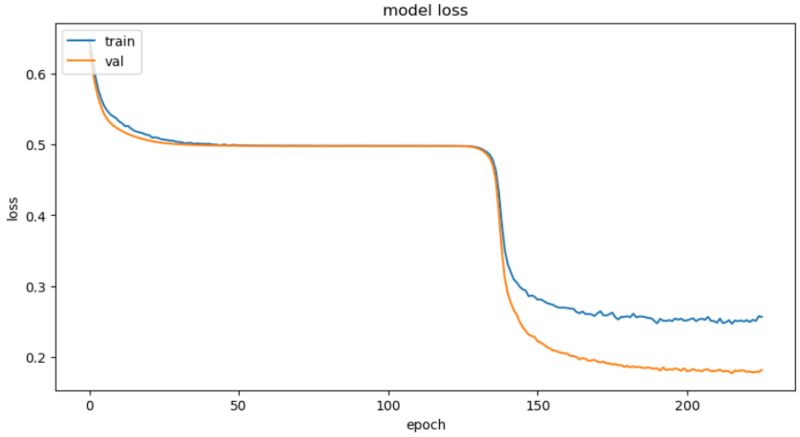
```
sensor_cols = cols_names[5:]
train_df[train_df.id==1][sensor_cols].plot(figsize=(20, 8))
```

```
<AxesSubplot:>
```



RNN





6. Conclusion

6.1. Summary of Findings

In this document, we have outlined the low-level design (LLD) of a predictive maintenance system tailored for aircraft engines. By leveraging various machine learning models including Linear Regression, Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN), Gradient Boosting (GBU), and Random Forest (RF), our system aims to analyse sensor data, flight patterns, and historical maintenance records to predict the likelihood of engine failure. Throughout the document, we have delved into the architectural design, training processes, evaluation methodologies, and data preprocessing techniques essential for the development and implementation of such a proactive maintenance solution.

During the model implementation and evaluation phase, we provided detailed insights into each model's training process, hyperparameter tuning, and evaluation metrics. By evaluating the performance of these models using metrics such as accuracy, precision, recall, and F1 score, we gained valuable insights into their effectiveness in predicting maintenance needs for aircraft engines.

6.2. Future Work

While the models presented in this document showcase promising results, there is still ample room for improvement and future exploration. Suggestions for future work include:

1. **Exploring Additional Models:** Investigating the potential of additional machine learning models beyond those discussed in this document could provide further insights and potentially enhance predictive capabilities.
2. **Refining Existing Models:** Continuously refining existing models by fine-tuning hyperparameters, optimizing feature engineering techniques, and exploring advanced algorithms can help improve the accuracy and robustness of the predictive maintenance system.
3. **Integration with Real-Time Data:** Integrating real-time data streams into the predictive maintenance system can enhance its responsiveness and adaptability, enabling more timely and accurate predictions of engine failures.
4. **Enhancing Visualization Techniques:** Further enhancing visualization techniques to provide intuitive and comprehensive insights into model performance and data trends can facilitate better decision-making for aviation maintenance experts and stakeholders.

By pursuing these avenues of future work, we aim to continuously advance the predictive maintenance system, ultimately contributing to improved aviation safety, reduced downtime, and enhanced operational efficiency.