Predictive Maintenance for Manufacturing

Sandaru Bandara

Table of Contents

1. Introduction

Objective

The primary objective of this project is to predict the Remaining Useful Life (RUL) of machinery using sensor data. Accurate prediction of RUL can help in implementing effective predictive maintenance strategies, thereby reducing unplanned downtimes and maintenance costs.

Background

Predictive maintenance leverages data analysis techniques to predict when equipment failure might occur so that maintenance can be performed just in time. This approach is crucial in industrial settings, where unexpected equipment failure can lead to significant operational and financial losses. By analyzing sensor data from machinery, we can develop models that forecast the RUL, enabling timely interventions and optimizing maintenance schedules.

2. Data Preparation

Dataset

The dataset used in this project is the NASA CMAPS dataset from Kaggle, which contains sensor measurements from various machinery. This data provides a rich source of information for developing predictive maintenance models.

Importing dependencies

```
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import mean_squared_error, mean_absolute_error
    import seaborn as sns
    import os
```

Loading invidual text files into a single data frame

```
[2] directory = '/content/drive/MyDrive/My_Projects/Nasa Turbo Fan  Project/archive/CMaps'

    #Reading the tarining data
    train_files = ['train_FD001.txt','train_FD002.txt','train_FD003.txt','train_FD004.txt']
```

Data Loading

The dataset comprises multiple text files, each containing sensor data. These files were read and combined into a single Data Frame for ease of analysis.

Loading invidual text files into a single data frame

```
[2] directory = '/content/drive/MyDrive/My_Projects/Nasa Turbo Fan  Project/archive/CMaps'

    #Reading the tarining data
    train_files = ['train_FD001.txt','train_FD002.txt','train_FD003.txt','train_FD004.txt']

    #Defining column names
    columns = [
        'unit_number', 'time_in_cycles', 'op_setting_1', 'op_setting_2', 'op_setting_3',
        'sensor_1', 'sensor_2', 'sensor_3', 'sensor_4', 'sensor_5', 'sensor_6', 'sensor_7',
        'sensor_8', 'sensor_9', 'sensor_10', 'sensor_11', 'sensor_12', 'sensor_13', 'sensor_14',
        'sensor_15', 'sensor_16', 'sensor_17', 'sensor_18', 'sensor_19', 'sensor_20',
        'sensor_21'
    ]

    #Loading and processing data
    def load_data(file_name):
      df = pd.read_csv(os.path.join(directory,file_name),sep='\s+', header=None)
      df.columns = columns
      return df

    #Loading the training data into a single data frame
    train_df_list = [load_data(file) for file in train_files]
    train_df = pd.concat(train_df_list, ignore_index=True)

    train_df.head()
```

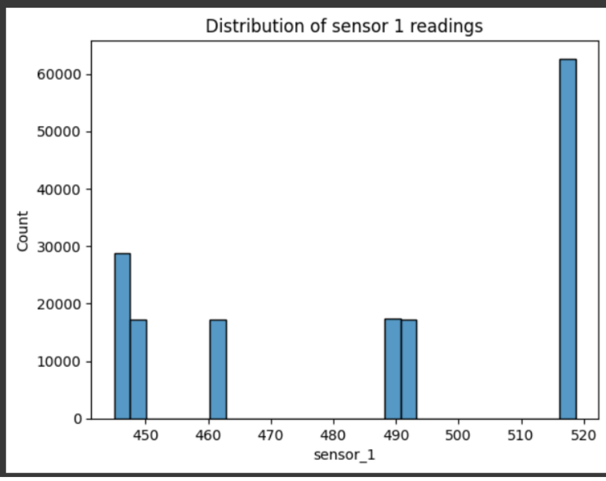
3.Initial Exploration

Initial exploratory data analysis (EDA) included:

- **Histograms and scatter plots**: Used to understand the data distribution and identify patterns.
- **Basic statistics**: Calculated to gain insights into the central tendency and spread of the data.

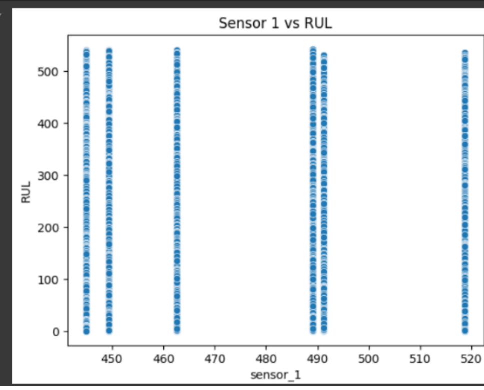
  This initial exploration helped in identifying potential issues, trends, and correlations in the data, setting the stage for further preprocessing and modeling steps.

```
[6] sns.histplot(train_df['sensor_1'])
    plt.title('Distribution of sensor 1 readings')
    plt.show()
```




```
[7] rul = train_df.groupby('unit_number')['time_in_cycles'].max().reset_index()
    rul.columns = ['unit_number', 'max_time_in_cycles']
    train_df = train_df.merge(rul, on=['unit_number'], how='left')
    train_df['RUL'] = train_df['max_time_in_cycles'] - train_df['time_in_cycles']
    train_df.drop('max_time_in_cycles', axis=1, inplace=True)

    # Scatter plot of sensor 1 vs RUL
    sns.scatterplot(x='sensor_1', y='RUL', data=train_df)
    plt.title('Sensor 1 vs RUL')
    plt.show()
```



Basic Statistics:

```
[5] train_df.describe()
```

	unit_number	time_in_cycles	op_setting_1	op_setting_2	op_setting_3	sensor_1	sensor_2	sensor_3	sensor_4
count	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000	160359.000000
mean	105.553758	123.331338	17.211973	0.410004	95.724344	485.840890	597.361022	1467.035653	1260.956434
std	72.867325	83.538146	16.527988	0.367938	12.359044	30.420388	42.478516	118.175261	136.300073
min	1.000000	1.000000	-0.008700	-0.000600	60.000000	445.000000	535.480000	1242.670000	1023.770000
25%	44.000000	57.000000	0.001300	0.000200	100.000000	449.440000	549.960000	1357.360000	1126.830000
50%	89.000000	114.000000	19.998100	0.620000	100.000000	489.050000	605.930000	1492.810000	1271.740000
75%	164.000000	173.000000	35.001500	0.840000	100.000000	518.670000	642.340000	1586.590000	1402.200000
max	260.000000	543.000000	42.008000	0.842000	100.000000	518.670000	645.110000	1616.910000	1441.490000

8 rows × 26 columns

4.Data Cleaning and Preprocessing

To ensure the data is ready for modeling, several cleaning and preprocessing steps were undertaken:

Handling Missing Values

- **Missing Value Check**: The dataset was checked for missing values.
- **Imputation**: Missing values were filled with the mean of the respective columns to maintain data integrity.

Check Data Types and Missing Values:

```
[4] train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 160359 entries, 0 to 160358
Data columns (total 26 columns):
 #   Column          Non-Null Count   Dtype  
---  ------          --------------   -----  
 0   unit_number     160359 non-null  int64  
 1   time_in_cycles  160359 non-null  int64  
 2   op_setting_1    160359 non-null  float64
 3   op_setting_2    160359 non-null  float64
 4   op_setting_3    160359 non-null  float64
 5   sensor_1        160359 non-null  float64
 6   sensor_2        160359 non-null  float64
 7   sensor_3        160359 non-null  float64
 8   sensor_4        160359 non-null  float64
 9   sensor_5        160359 non-null  float64
 10  sensor_6        160359 non-null  float64
 11  sensor_7        160359 non-null  float64
 12  sensor_8        160359 non-null  float64
 13  sensor_9        160359 non-null  float64
 14  sensor_10       160359 non-null  float64
 15  sensor_11       160359 non-null  float64
 16  sensor_12       160359 non-null  float64
 17  sensor_13       160359 non-null  float64
 18  sensor_14       160359 non-null  float64
```

Normalization

- **Standardization**: Sensor data was standardized using StandardScaler to ensure uniform scaling across all features.

Normalize/Standardize Sensor Data

Normalizing or standardize the sensor data to ensure all features contribute equally to the model

```
[9] from sklearn.preprocessing import StandardScaler

    #selecting sensor columns
    sensor_columns = [f'sensor_{i}'for i in range(1,22) ]
    scalar = StandardScaler()
    train_df[sensor_columns] = scalar.fit_transform(train_df[sensor_columns])
```

Feature Engineering

- **Rolling Statistics**: Created new features based on rolling mean and standard deviation to capture trends over time.

Feature Engineering

Creating additional features that might help the model based on rolling statistics

```
[10] for sensor in sensor_columns:
         train_df[f'{sensor}_rolling_mean'] = train_df.groupby('unit_number')[sensor].rolling(window=5).mean().reset_index(level=0, drop=True).fil
```

5. Modeling

Initial Model Training

Multiple regression models were trained to predict the Remaining Useful Life (RUL) of machinery. The models evaluated included:

- **Linear Regression**
- **Random Forest Regressor**
- **Gradient Boosting Regressor**

These models were trained on the preprocessed data, and their performance was evaluated using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

Training the models

```
[36] from sklearn.linear_model import LinearRegression
     from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

     # Initialize the models
     linear_model = LinearRegression()
     rf_model = RandomForestRegressor(random_state=42)
     gb_model = GradientBoostingRegressor(random_state=42)

     # Train the models
     linear_model.fit(X_train, y_train)
     rf_model.fit(X_train, y_train)
     gb_model.fit(X_train, y_train)
```

```
GradientBoostingRegressor
GradientBoostingRegressor(random_state=42)
```

Evaluate the Models

[37]
```python
# Defining a function to evaluate models
def evaluate_model(model, X_val, y_val):
    predictions = model.predict(X_val)
    mae = mean_absolute_error(y_val, predictions)
    rmse = np.sqrt(mean_squared_error(y_val, predictions))
    return mae, rmse

# Evaluate the models
linear_mae, linear_rmse = evaluate_model(linear_model, X_val, y_val)
rf_mae, rf_rmse = evaluate_model(rf_model, X_val, y_val)
gb_mae, gb_rmse = evaluate_model(gb_model, X_val, y_val)

print(f"Linear Regression - MAE: {linear_mae}, RMSE: {linear_rmse}")
print(f"Random Forest - MAE: {rf_mae}, RMSE: {rf_rmse}")
print(f"Gradient Boosting - MAE: {gb_mae}, RMSE: {gb_rmse}")
```

```
Linear Regression - MAE: 59.07693987884737, RMSE: 75.1324502597411
Random Forest - MAE: 54.30610875530058, RMSE: 70.16138296934336
Gradient Boosting - MAE: 57.177243470646246, RMSE: 73.65807904132173
```

Hyperparameter Tunning

The Random Forest model, which showed the best initial performance, was further optimized using GridSearchCV to find the best hyperparameters.

Model Selection

```python
from sklearn.model_selection import train_test_split

# Use 10% of the training data for hyperparameter tuning
X_train_small, _, y_train_small, _ = train_test_split(X_train, y_train, train_size=0.1, random_state=42)

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint
import joblib

# Define a reduced parameter distribution
param_dist = {
    'n_estimators': randint(50, 150),
    'max_depth': randint(10, 20),
    'min_samples_split': randint(2, 5),
    'min_samples_leaf': randint(1, 2),
    'bootstrap': [True, False]
}

# Initialize RandomizedSearchCV with fewer iterations
random_search_rf = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_dist,
    n_iter=10,  # Reduced number of iterations
    cv=3,  # Reduced number of cross-validation folds
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    random_state=42
)
```

```python
# Initialize RandomizedSearchCV with fewer iterations
random_search_rf = RandomizedSearchCV(
    estimator=rf_model,
    param_distributions=param_dist,
    n_iter=10,  # Reduced number of iterations
    cv=3,  # Reduced number of cross-validation folds
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    random_state=42
)

# Fit RandomizedSearchCV to the smaller training data
random_search_rf.fit(X_train_small, y_train_small)

# Get the best parameters and best score
best_params_rf = random_search_rf.best_params_
best_score_rf = -random_search_rf.best_score_

print(f"Best parameters: {best_params_rf}")
print(f"Best MAE score: {best_score_rf}")

# Save the best model
joblib.dump(random_search_rf.best_estimator_, 'tuned_random_forest_model.pkl')
```

```
Best parameters: {'bootstrap': True, 'max_depth': 17, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 149}
Best MAE score: 57.506449139213835
['tuned_random_forest_model.pkl']
```

5. Results

Model Performance

After hyperparameter tuning, the best Random Forest model was evaluated again on the validation set, showing improved performance.

```python
best_rf_model = joblib.load('tuned_random_forest_model.pkl')

best_rf_mae, best_rf_rmse = evaluate_model(best_rf_model, X_val, y_val)
print(f"Tuned Random Forest - MAE: {best_rf_mae}, RMSE: {best_rf_rmse}")
```

```
Tuned Random Forest - MAE: 56.735508391221714, RMSE: 73.00691326620256
```

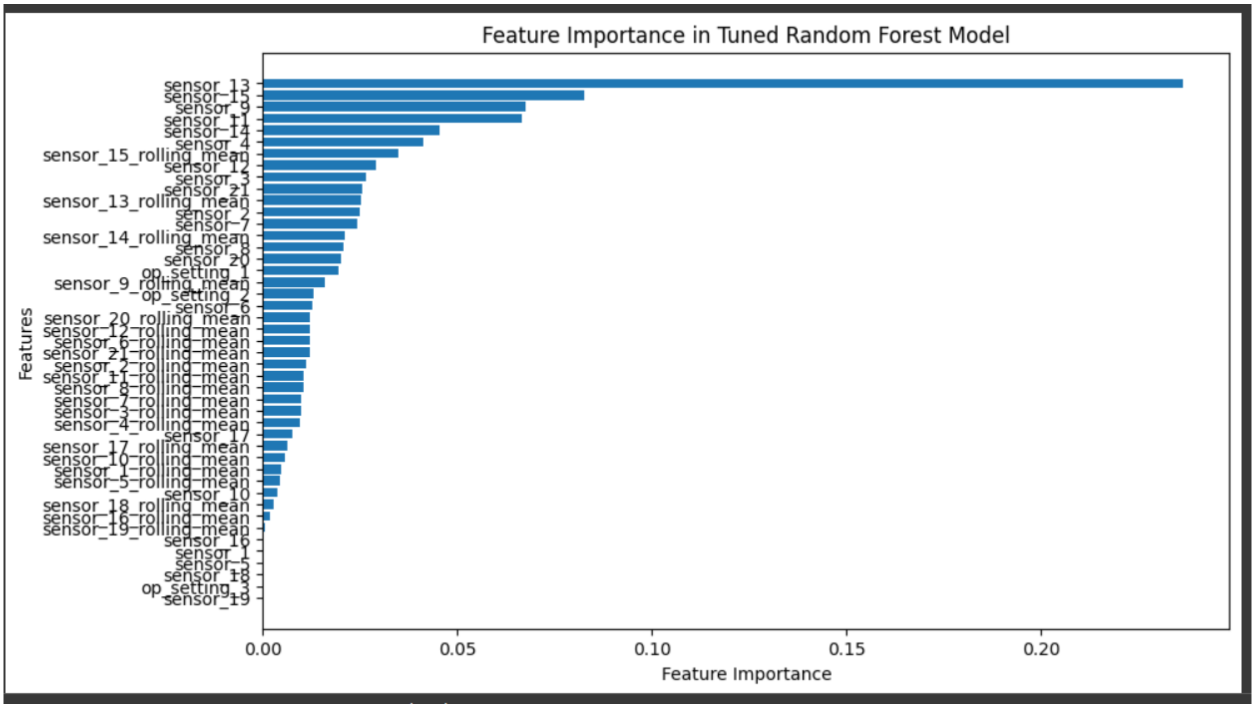
```python
import matplotlib.pyplot as plt

importances = best_rf_model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'])
plt.xlabel('Feature Importance')
plt.ylabel('Features')
plt.title('Feature Importance in Tuned Random Forest Model')
plt.gca().invert_yaxis()
plt.show()
```

Feature Importance

The feature importances of the tuned Random Forest model were analyzed to understand which features contributed most to the predictions.



6. Conclusion

- **Summary**:
  - Successfully built a predictive maintenance model to estimate the RUL of machinery.
  - Optimized the Random Forest model to achieve the best performance.
- **Future Work**:
  - Explore additional feature engineering techniques.
  - Implement the model in a real-time system for continuous monitoring.