PREDICTIVE MAINTENANCE AND PERFORMANCE OPTIMIZATION FOR JET ENGINES BASED ON ROLLS-ROYCE ENGINE MANUFACTURER AND SERVICES WITHIN THE AEROSPACE SECTOR

SITI SYAHIRAH BINTI MOHD YUNUS
MCS241012

CHAPTER 4

UNIVERSITI TEKNOLOGI MALAYSIA

Chapter 4.

4.1 Introduction

4.2 EDA

Agenda

Objective

1. Predict RUL
2. Categorize maintenance required engines < 42 cycles and check accuracy

Flow

1. Data extraction, EDA & Feature selection/engineering
2. Predictive model

Remark : Only work with FD001

1. Data Extraction, EDA & Feature selection/engineering

```
# Import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings

warnings.filterwarnings("ignore", category=FutureWarning, message=".*use_inf_as_na.*")
```

Load the data

```python
"""Load data from files"""
import glob as gl
import pandas as pd
import pathlib as pth


def load_file(path: str, load_all: bool = False, filter: str = None, sep: str = None, header: int = 0) -> pd.DataFrame:
    if pth.Path(path).is_file():
        if path.endswith(".csv"):
            dfs = pd.read_csv(path, header=header)
        elif path.endswith(".txt"):
            dfs = pd.read_csv(path, sep=sep, header=header)
        elif path.endswith(".xlsx"):
            dfs = pd.read_excel(path, header=header)
    elif pth.Path(path).is_dir() and load_all:
        df_list = []
        files = gl.glob(f"{path}/*")
        if filter:
            files = [i for i in files if filter in i]
        print("files to be loaded:", files)
        for file in files:
            if file.endswith(".csv"):
                df = pd.read_csv(file, low_memory=False, header=header)
                df_list.append(df)
            elif file.endswith(".txt"):
                df = pd.read_csv(file, sep=sep, header=header)
                df_list.append(df)
            elif file.endswith(".xlsx"):
                df = pd.read_excel(file, header=header)
                df_list.append(df)
        dfs = pd.concat(df_list)
        dfs.reset_index(drop=True, inplace=True)
    elif pth.Path(path).is_dir() and not load_all:
        raise ValueError("load_all should be True")
    else:
        raise ValueError("error")

    return dfs
```

```python
# Dict for column headers; refer to the paper
df_columns = {
    0: "engine",
    1: "cycle_time",
    2: "operational_set_1",
    3: "operational_set_2",
    4: "operational_set_3",
    5: "T2",
    6: "T24",
    7: "T30",
    8: "T50",
    9: "P2",
    10: "P15",
    11: "P30",
    12: "Nf",
    13: "Nc",
    14: "epr",
    15: "Ps30",
    16: "phi",
    17: "NRf",
    18: "NRc",
    19: "BPR",
    20: "farB",
    21: "htBleed",
    22: "Nf_dmd",
    23: "PCNfR_dmd",
    24: "W31",
    25: "W32",
}

RUL_columns = {
    0: "RUL"
}
```

```python
# Rename columns
train_df = train_df.rename(columns=df_columns)
test_df = test_df.rename(columns=df_columns)
RUL_df = RUL_df.rename(columns=RUL_columns)
RUL_df["engine"] = RUL_df.index + 1
```

+ Code + Markdown

```python
# Display first 5 rows of each df
print(train_df.head())
print(test_df.head())
print(RUL_df.head())
```

Remaining Unit Life (RUL) mapping into training data

```python
# calculate maximum cycle_time per engine
max_time_cycles = train_df.groupby("engine")["cycle_time"].max()
# To check result
# for idx, row in max_time_cycles.reset_index().iterrows():
#     print("Engine:", row["engine"], "Failure_time:", row["cycle_time"])
# merge into maximum cycle time into train df
merged = train_df.merge(max_time_cycles.to_frame(name="max_time_cycle"), left_on="engine", right_index=True)
# calculate RUL = maximum cycle time - each cycle time
merged["RUL"] = merged["max_time_cycle"] - merged["cycle_time"]
# drop maximum cycle time (not used)
merged = merged.drop("max_time_cycle", axis=1)
# update train df with RUL
train_df = merged
```

```python
train_df.head()
```

EDA

Target Value

```
# plot maximum cycle_time per engine
failure_time=max_time_cycles.reset_index()

plt.figure(figsize=(8,16))
sns.set_theme()
sns.barplot(y=failure_time["engine"], x=failure_time["cycle_time"], orient="h")
plt.title("Failure cycle on each engine")
plt.ylabel("Engine")
plt.xlabel("Cycle")
plt.tight_layout()
plt.show()

# distribution of maximum cycle_time
plt.figure()
sns.histplot(failure_time["cycle_time"], kde=True, bins=20)
plt.show()
```

```
# distribution of RUL (answer)
plt.figure()
sns.histplot(RUL_df["RUL"], kde=True, bins=20)
plt.show()

print(test_df.shape)
```

General EDA

```python
"""EDA modules"""
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt


class EDA:
    """Class for General EDA.

    variables
    ----------
    df: pd.DataFrame
    target_col: str
        Target column on dataframe (only one column)
        - This column would be defined if you want to explicitly compare to other features
    ----------
    """
    def __init__(self, df: pd.DataFrame, target_col: str = None) -> None:
        """Initialize variables and define columns."""
        self._df = df
        self._tgt_col = target_col
        self._num_cols = df.select_dtypes(include="number").columns
        self._cat_cols = df.select_dtypes(include=["object", "category"]).columns
        print("Numerical columns")
        print(self._num_cols)
        print("--------")
        print("Categorical columns")
        print(self._cat_cols)
        print("--------")
        print("Target column")
```

```python
def describe_df(self) -> None:
    """Describe dataframe; info, nunique, null ratio & describe."""
    print("Overall")
    print(self._df.info())
    print("--------")
    print("Unique value count")
    print(self._df.nunique())
    print("--------")
    print("Null value rate")
    print(self._df.isnull().sum()/len(self._df)*100)
    print("--------")
    print("Describe numerical columns")
    print(self._df.describe())
    print("--------")
    if len(self._cat_cols) > 0:
        print("Describe categorical columns")
        print(self._df.describe(include="object"))
        print("--------")

def dist_plot(self) -> None:
    """Distribution plots: histgram, boxplot & violin plot.

    If target column defined, then plots will be generated except the target column.
    """
    if self._tgt_col:
        for col in [x for x in self._num_cols if x != self._tgt_col]:
            f, ax = plt.subplots(1,3,figsize=(12,3))
            sns.histplot(self._df, x=col, ax=ax[0], kde=True, bins=20, stat="percent", hue=self._tgt_col)
            sns.boxplot(self._df, x=col, hue=self._tgt_col, ax=ax[1])
            sns.violinplot(self._df, x=col, hue=self._tgt_col, ax=ax[2], alpha=0.3)
            plt.legend()
            plt.show()

        else:
            for col in self._num_cols:
                f, ax = plt.subplots(1,3,figsize=(12,3))
                sns.histplot(self._df, x=col, ax=ax[0], kde=True, bins=20, stat="percent")
                sns.boxplot(self._df, x=col, ax=ax[1])
                sns.violinplot(self._df, x=col, ax=ax[2], alpha=0.3)
                plt.show()

    def counter_plot(self) -> None:
        """Counter plots.

        If target column defined, then plots will be generated except the target column.
        """
        for col in self._cat_cols:
            f, ax = plt.subplots(1,1,figsize=(12,4))
            sns.countplot(data=self._df, x=col, ax=ax)
            x_labels_1 = [label.get_text() for label in ax.get_xticklabels()]
            ax.set_xticklabels(x_labels_1, rotation=45)
            plt.show()


    def pair_plot(self) -> None:
        """Pair plot"""
        if self._tgt_col:
            _ = sns.pairplot(self._df, hue=self._tgt_col)
        else:
            _ = sns.pairplot(self._df)

    def pie_plot_target(self) -> None:
        """Pie plot for target column."""
        tgt_counts = self._df[self._tgt_col].value_counts()

        plt.figure()
        plt.pie(tgt_counts, labels=tgt_counts.index, autopct='%1.1f%%', colors=["#5B91B5", "#E47336"])
        plt.title(f"Distribution of {self._tgt_col}")
        plt.show()
```

```
    def dist_plot_target(self) -> None:
        """Distribution plot for target column."""
        plt.figure()
        sns.histplot(self._df[self._tgt_col], kde=True)
        plt.show()

    def corr_plot(self) -> None:
        """Calculate Pearson correlation among numerical columns and plot it."""
        df_corr = self._df.corr(numeric_only=True)
        mask = np.tril(np.ones(df_corr.shape),k = -1).astype(bool)
        df_corr_fil = df_corr.where(mask)

        plt.figure(figsize=(16,16))
        sns.heatmap(df_corr_fil, annot=True, cmap="crest", fmt=".2f", linewidths=0.01)
        plt.show()

        plt.figure(figsize = (16,16))
        mask = df_corr_fil.where(abs(df_corr_fil) > 0.9).isna()
        sns.heatmap(df_corr_fil, annot=True, cmap="crest", fmt=".2f", linewidths=0.01, mask=mask)
        plt.show()


    def plot_all(self) -> None:
        """Execute all functions."""
        self.describe_df()
        # If Binary value on the target column, then create a pie plot; otherwise histogram.
        if self._tgt_col:
            if len(self._df[self._tgt_col].unique()) > 2:
                self.dist_plot_target()
            else:
                self.pie_plot_target()
        self.dist_plot()
        self.counter_plot()
        self.corr_plot()
```

```
eda = EDA(train_df)
```

```
eda.plot_all()
```

Data Cleansing

```
# Fine 3 shorter RUL cases and 3 longer RUL cases as an example
mins = failure_time.sort_values(by="cycle_time")[:3]["engine"].to_list()
maxs = failure_time.sort_values(by="cycle_time", ascending=False)[:3]["engine"].to_list()
print("3 engines shorter cycle_time:", mins)
print("3 engines longer cycle_time:", maxs)
cols = mins + maxs
print("Engines to be plotted:", cols)
```

```
warnings.filterwarnings("ignore", category=FutureWarning, message=".*length-1 tuple.*")

# Plot trends of each feature
pallete = sns.color_palette("flare", 6)

for col in [i for i in train_df.columns if i not in ["engine", "cycle_time", "RUL"]]:
    plt.figure(figsize=(12,4))
    sns.set_theme()
    sns.lineplot(data=train_df[train_df["engine"].isin(cols)], x="cycle_time", y=col, hue="engine", palette=pallete)
    plt.title(f"{col} output per cycle")
    plt.tight_layout()
    plt.show()
```

Feature selection & engineering

```
"""Feature engineering."""

import pandas as pd
from typing import Union

def remove_col(df: pd.DataFrame, col: Union[str, list]) -> pd.DataFrame:
    df.drop(columns=col, inplace=True)
    return df

def normalize_by_first_n_values(df: pd.DataFrame, grp_col: Union[str, list], features: Union[str, list] = None, rolling: int = 1) -> pd.DataFrame:
    start_value = df.groupby(grp_col).transform(lambda x: x.iloc[:rolling].mean())

    if features:
        df[features] = df[features] / start_value[features]
    else:
        df = df /start_value

    return df

def cliping(df: pd.DataFrame, col: Union[str, list], value: Union[int, float]) -> pd.DataFrame:
    df[col] = df[col].clip(upper=value)

    return df
```

```
# drop columns of operational setting and which have an unique value
drop_cols = ["operational_set_1", "operational_set_2", "operational_set_3", "T2", "P2", "P15", "epr", "farB", "Nf_dmd", "PCNfR_dmd"]
train_df = remove_col(train_df, drop_cols)
test_df = remove_col(test_df, drop_cols)
```

```python
# check static of failure timing on train for further modification
print("train data, failure timing")
print("average =", int(failure_time["cycle_time"].mean()))
print("median =", int(failure_time["cycle_time"].median()))
print("min = ", failure_time["cycle_time"].min())
print("max = ", failure_time["cycle_time"].max())
print("0.25 quantile = ", failure_time["cycle_time"].quantile(0.25))
print("0.75 quantile = ", failure_time["cycle_time"].quantile(0.75))
```

```python
# Normilization of sensor signal by first 10 values with smoothing average
grp_col = "engine"
features = [i for i in train_df.columns if i not in ["engine", "cycle_time", "RUL"]]
train_df = normalize_by_first_n_values(train_df, grp_col, features, 10)
test_df = normalize_by_first_n_values(test_df, grp_col, features, 10)

print(train_df.head())
print(test_df.head())
```

```python
# Plot trends of each feature
pallete = sns.color_palette("flare", 6)

for col in [i for i in train_df.columns if i not in ["engine", "cycle_time", "RUL"]]:
    plt.figure(figsize=(12,4))
    sns.set_theme()
    sns.lineplot(data=train_df[train_df["engine"].isin(cols)], x="cycle_time", y=col, hue="engine", palette=pallete)
    plt.title(f"{col} output per cycle")
    plt.tight_layout()
    plt.show()
```

```python
# correlation heatmap with selected columns
eda2 = EDA(train_df)
eda2.corr_plot()
```

```python
# highly correlated columns (corr > 0.9)

add_drop_cols = ["NRc"]

train_df = remove_col(train_df, add_drop_cols)
test_df = remove_col(test_df, add_drop_cols)
```

```python
# we are not so interesting in RULs which are not requiring maintenance, thus we set up a upper limit from historical data * 0.95 (as safetynet)
# This allows to avoid over prediction

min_failure_time = int(failure_time["cycle_time"].min()*0.95)
train_df = cliping(train_df, "RUL", min_failure_time)
train_df.head()
```