High Level Design [HLD]
Predictive Maintenance RUL prediction

Documented version control

Date issued	Version	Description	Authors
30-04-22	1	Initial HLD	Shikhar Amar, Hitesh Pathak

Contents

Abstract

In the present scenario, Air Transport has become quintessential not only for human transportation but also for international trade, logistics and innumerable other departments. Accordingly, proper maintenance of the underlying components of this system is of utmost importance. Prognostics and Fault detection is an important topic in industry for predicting state of assets to avoid downtime and failures, failing which we put the lives of general public as well as the health of infrastructure at risk.

In this work, we aim at presenting a solution that targets this need of the hour, by predicting the remaining useful life of jet engines with varying operating conditions and failure modes. Prediction models are based on the simulated Run-to-failure data set for turbofan jet engines. Finally, the models are validated by checking against test data-set.

1 Introduction

1.1 Purpose of document

The purpose of this High-Level-Document (HLD) is to define the high level outline and description of this project, which provides a suitable programming model. This also helps in preemptive detection of contradictions and potential pitfalls / out-sights prior to coding and provides. To the reader, the document can serve as a reference manual to understand how various components of the project works at a high level and to understand the end goal of it.

HLD will :

- Design aspects of the system.
- Describe the API or UI that is exposed to the users.
- Describe all the internal software / hardware interfaces.
- Describe technical requirements.
- Describe enhancements that can be made.
- List the non functional attributes of system like :
  - Reliability
  - Maintainability
  - Resource Utilization
  - Response Time

1.2 Scope

The HLD discusses the structure of the system, including database architecture, application flow (navigation), and technology architecture. We use mildly technical language throughout the document which should be easily understandable to system administrators.

1.3 Definitions

Terms	Meaning
RUL	Remaining useful life
AWS	Amazon Web Services
IDE	Integrated Developement Environment
API	Application Programming Interface

2 General Description

2.1 Product perspective

Remaining useful life (RUL) prediction system works by modeling asset degradation through specific machine-learning algorithms that are selected based upon the type of data provided. This prognostics system can help eliminate the chance of failure or downtime leading to a more consistent and fault-tolerant infrastructure which in turn reduces the risk over civilians due to ill-maintained assets.

2.2 Experimental scenario

The training data is not empirical but simulated public data set for asset degradation modeling from NASA that contains Run-to-failure data for turbofan engines, simulated using C-MAPSS. There are several data sets that account for distinct operating conditions and fault modes. Each data set consists multiple multivariate time series, where a unit of time is a cycle of operation for the engine. Each time series in the set corresponds to a different engine, hence the data can be considered to be from a group of similar engines.

Each engine starts with a different initial condition including manufacturing flaws, variations; these however are not considered as a fault condition. The data also includes three *operational settings* and data from 21 *sensors*, all columns in the data set indicate different variables as shown :

1. Unit number of engine.
2. Time in cycles of operation.
3. Operational setting 1

4. Operational setting 2
5. Operational setting 3
6. Sensor 1
7. Sensor 2

   ...

26. Sensor 26

Every engine is operating normally at the start of its time series, and develops a fault at some time during the series. The fault grows in magnitude, until system failure.

2.3 Problem statement

The test data sets should be similar in structure to the training data, the only difference being that the time series stops somewhere before failure. Given a test data set, the end goal of our model is to predict the RUL for each engine. RUL is equivalent to the number of flights or cycles that the engine can undergo after the last cycle in the time series for engine and before its breakdown.

2.4 Proposed Solution

The solution proposed here is a RUL detection system, which selects a suitable machine-learning algorithm based on the characteristics of input data set to model degradation of the engine. The predicted RUL can then be used to estimate the heath of the engine and to decide if maintenance or replacement is needed. This solution can be of immense use in Prognostics industry, to help prevent failure of the infrastructure which can potentially endanger humans who rely on it.

2.5 Further improvements

This detection system can be further enhanced by adding more sensors, fault modes to the data sets. Also critical assets other than engine can be incorporated, by providing degradation data for those and training the algorithm.

Other prediction systems can also be used together with RUL prediction; for instance, together with a weather forecasting algorithm we can predict if a flight should be delayed or not.

2.6 Technical requirements

2.7 Data requirements

2.8 Tools used

- Pycharm is used as the IDE.
- For visualization and EDA; Matplotlib, Seaborn, Plotly libraries are used.
- Pandas is used for data processing and validation.
- Git is used for version control.
- GitHub is used to manage git repositories.

2.9 Constraints

- The RUL detection system should be user friendly, and should not require the user to understand the internal workings.
- The solution should be portable and should work irrespective of the operating system of the user.

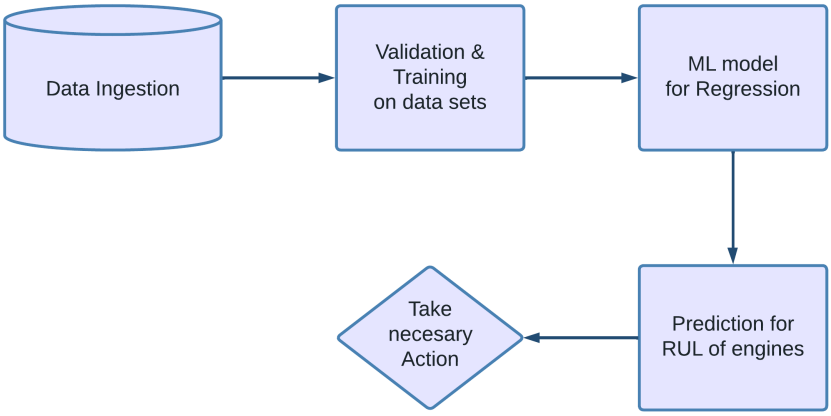
2.10 Assumptions

It is assumed that the test data for prediction, has a similar structure as the training data i.e., it contains readings from same number of sensors and the number of operational settings is also same as training data sets. Further we have assumed that for each engine, degradation starts after first use, this may or may not be the case in a practical scenario as a fault can develop at any time during the life-cycle of the engine.

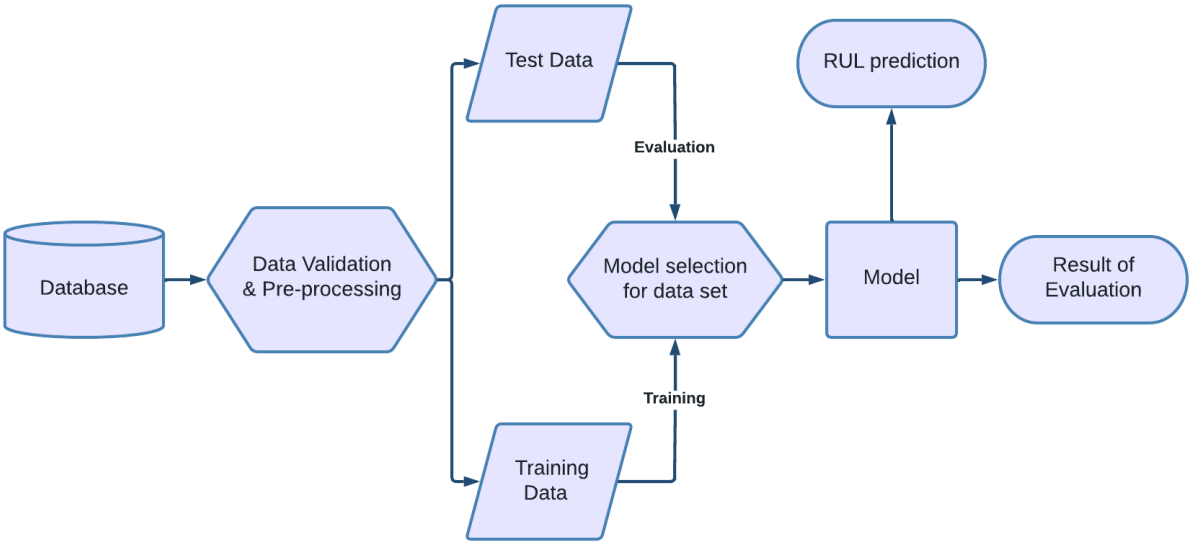
3 Design

3.1 Process flow

We use a Regression based model for RUL prediction. The process flow diagram is shown below.



3.1.1 Model training and evaluation



3.1.2 Deployment Process

3.2 Logging

Logs are kept for every event, so that a user or an administrator can easily troubleshoot or debug the program. Logging is done using the python logging library, the steps are described below :

1. For each event, it's logging level (Info, Warning, Debug etc.) is identified.
2. The logs are committed by the appropriate handler.
3. Along with writing logs to a file, appropriate messages are displayed to the user.
4. The logs can be either kept at a local system in flat file or database, as the developer required.
5. Logging should keep resource usage to a minimum, so it does not affect the program flow.

3.3 Error handling

An Error is an event that should not occur under normal and intended usage of the program. When an error occurs, the system logs it in the error log and an error message is displayed to the user which can be used for troubleshooting.

4 Performance

The RUL detection system can be used in Prognostics or health management industries, where it can alert the concerned department about RUL of engines, and if any of them are in a critical need of maintenance or replacement. As the solution can be easily deployed to cloud, it will be easily accessible to the users. Performance depends largely on the cloud engine and hence is irrespective of the user system.

The accuracy of the system relies on the training data, therefore as a prerequisite the user should make sure that the test data and training data correspond to similar class of engines. Otherwise a poorly trained model could raise false alarms, or fail to warn when required, which could have catastrophic repercussions.

4.1 Portability and re-usability

The model program can be entirely deployed on the cloud, after it has been trained by the developers. Therefore the solution is by design, portable.

The program code is written in a modular fashion, which aids in re-usability and maintenance of code over long time. The individual modules can be easily modified and reused.

4.2 Application compatibility

All the core components of this system are interconnected with Python as an interface. Each component performs its own individual task and Python interface provides communication between the modules.

4.3 Resource Utilization

The program utilizes maximum resources of the system while performing a task.

4.4 Deployment

5 Conclusion