ARCHITECTURE

Prediction of failure in APS of Scania Trucks

Revision Number: 1.0

Last date of revision: 01/03/2023

Lovely Patra

# Document Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Date Issued | Version | Description | Author |
| 1st March 2023 | 1.1 | ARCHITECHTIRE | Lovely Patra |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[Document Version Control 2](#_gjdgxs)

[**Abstract** 4](#_30j0zll)

[1](#_1fob9te) Introduction 5

[1.1](#_3znysh7) Why this Low-Level Design Document? 5

[1.2](#_2et92p0) Scope 6

[2](#_4d34og8) Technical specifications 7

[3](#_26in1rg) Technology stack 8

[4](#_35nkun2) Model training/validation workflow 10

[5](#_1ksv4uv) User I/O workflow 12

[6](#_44sinio) Exceptional scenarios 13

**Abstract**

Air Pressure System is a vital part of heavy vehicles like Scania trucks. The braking system in these vehicles is dependent on-air pressure and hence there is a need for the proper functioning of the air pressure system. Predictive maintenance in automobile industry reduces the maintenance cost and improve the performance of the vehicle. This can be achieved either manually or automatically. Manual predictive maintenance needs interference of the human task and may induce some errors. Automatic predictive maintenance through artificial intelligence techniques explores the hidden cause for failure of air pressure system in the Scania trucks. ln the proposed system, machine learning approaches are investigated for predictive maintenance of the trucks hised on condition of air pressure system. The dataset used in this work consist of 35,188 negative instances and 1,000 positive instances. Hence there is a need to address the class imbalance issue therefore applying machine learning algorithms. Many resampling techniques like under sampling, over sampling and SMOTE are analysed for the efficiency of the classifier. After pre-processing the data, machine learning classification algorithms like Random Forest, Logistic Regression, Gradient Boosting, Decision Tree, K-Neighbors Classifier, XGBClassifier, CatBoosting Classifier and AdaBoost Classifier are implemented and accuracy of the classifiers are analysed. Experimental results show that XGBoost Classifier with 99.6% accuracy and cost of 2950.

# Introduction

## Why this Low-Level Design Document?

The purpose of this document is to present a detailed description of the Air pressure system failure in Scania Trucks. The air pressure system (APS) plays a critical role in heavy Scania trucks: APS generates pressurized air that is used in various critical functions such as braking and gear changing.

The operational cost of Scania truck fleet can be significantly reduced by accurate prediction of the failure status of APS based on the measurements of truck mechanical system attributes.

Machine Learning (ML) and Deep Learning (DL) models can be developed and deployed to make predictions so as to minimize costs due to either unnecessary checkups or breakdowns caused by APS failures that have been missed.

## Scope

This is a **Binary Classification** problem where the positive class tells us that the failure was due to a specific component of the APS, whereas, the negative class tells us that the failure has nothing to do with that component. Therefore, given a new data point (sensor information), we can build an ML model that would tell us if the failure was due to the truck’s APS or not*.*

# Technical specifications

# 2.1 Business Constraints

**Latency** must be fairly low, to detect a failure in the APS and avoid increase in maintenance cost.

**Cost of misclassification is very high** since an undetected failure of the APS component can lead to failure of the truck during operation and therefore an increase in maintenance cost.

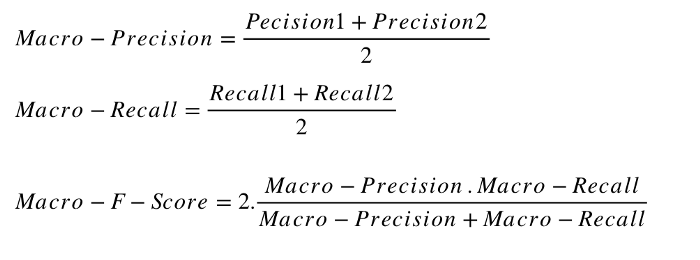
# 2.2 Dataset Overview

The training dataset consists of **36,188 data points** and **171 features**, of which one is the class label. The features are a combination of numerical data and histogram bins data. The feature names are kept anonymized for proprietary reasons. 35,188 data points belong to the negative class and the remaining 1,000 belong to the positive class. This tells us that we are dealing with a **highly imbalanced dataset** and is usually the type of data we can expect in a real world scenario.

Another problem observed is that a large part of the **data is missing**. In extreme cases, some instances have 80% of the values missing. The dataset is classified as **Missing Completely At Random (MCAR)**, as there is no relationship whether a data point is missing and any value in the dataset is missing or observed. Therefore, we have to find ways to resolve these issues by feature engineering methods.

# 2.3 Performance Metric

We will be using **Macro-F1 Score** as our performance metric for this project. Macro F1 score takes in to account the F1 scores of each class. It may be beneficial in showing us the performance of our model based on the number of correctly classified points for both classes. This is useful because the cost of misclassification is very high since an APS failure which is not detected can lead to failure of the truck during operation and increase in maintenance cost.



# 2.4 Literature Review

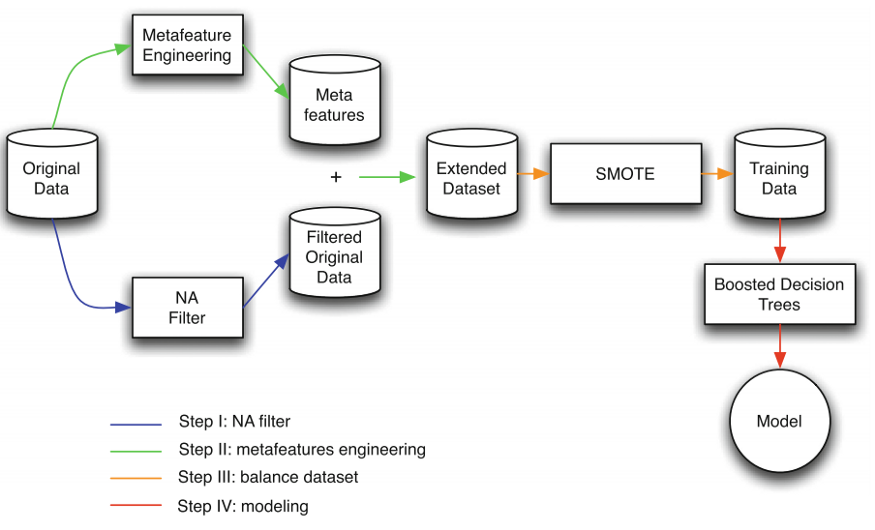
“**Combining Boosted Trees with Metafeature Engineering for Predictive Maintenance.**” International Symposium on Intelligent Data Analysis. Springer, Cham, 2016.

This paper mentions that the authors’ approach to this problem consists of 4 steps. (i) A filter that excludes a subset of features and data points based on the number of missing values; (ii) A metafeature engineering procedure used to create new features based on existing information; (iii) a biased sampling method to deal with class imbalance problem (SMOTE); and (iv) use of boosted trees for classification.

Features having a high percentage of missing values were removed. During their analysis, they found that some features had an extremity of 80% data missing, and 8 out of 170 features had more than 50% missing values. After removing the said features, it was seen that there were duplicate data points, indicating that the removed features have a little effect in getting a good score.

They mentioned that they are treating the problem as an Anomaly Detection problem since the positive class of the data are characterized by rare events in the domain. They used BoxPlot Analysis (for each feature, compare each value to the typical value found in that feature), Local Outlier Factor (compare data point to it’s local neighborhood through density estimation) and Hierarchical Agglomerative Clustering (each step merges two similar group, and the last observation that are merged might be an outlier) for their metafeature engineering.

[**SMOTE**](https://www.geeksforgeeks.org/ml-handling-imbalanced-data-with-smote-and-near-miss-algorithm-in-python/#:~:text=SMOTE%20(synthetic%20minority%20oversampling%20technique)%20is%20one%20of%20the%20most,instances%20between%20existing%20minority%20instances.) is a method of duplicating the data points of the minority class of the imbalanced dataset, to balance it out. The use of SMOTE + MetaFeature Engineering with XGBOOST library was seen to give the best result.



It is a large scale matrix completion algorithm that replaces missing values with current guesses and solves an optimization problem. The imbalance data was handled by setting a high threshold (cut-off) value, meaning the model will predict a negative class only if it is extremely sure.

The final result showed that XGBoost Classifier performed the best, giving a Total Cost of 2950 with a accuracy of 99.6%.

## 2.5 Logging

We should be able to log every activity done by the user.

* The System identifies at what step logging required
* The System should be able to log each and every system flow.
* Developers can choose logging methods. You can choose database logging/ File logging as well.
* System should not be hung even after using so many loggings. Logging just because we can easily debug issues so logging is mandatory to do.

## 2.6 Database

System needs to store every request into the database and we need to store it in such a way that it is easy to retrain the model as well.

1. The User gives required information.

2. The system stores each and every data given by the user or received on request to the database. Database you can choose your own choice whether MongoDB/ MySQL.

**2.7 Deployment**

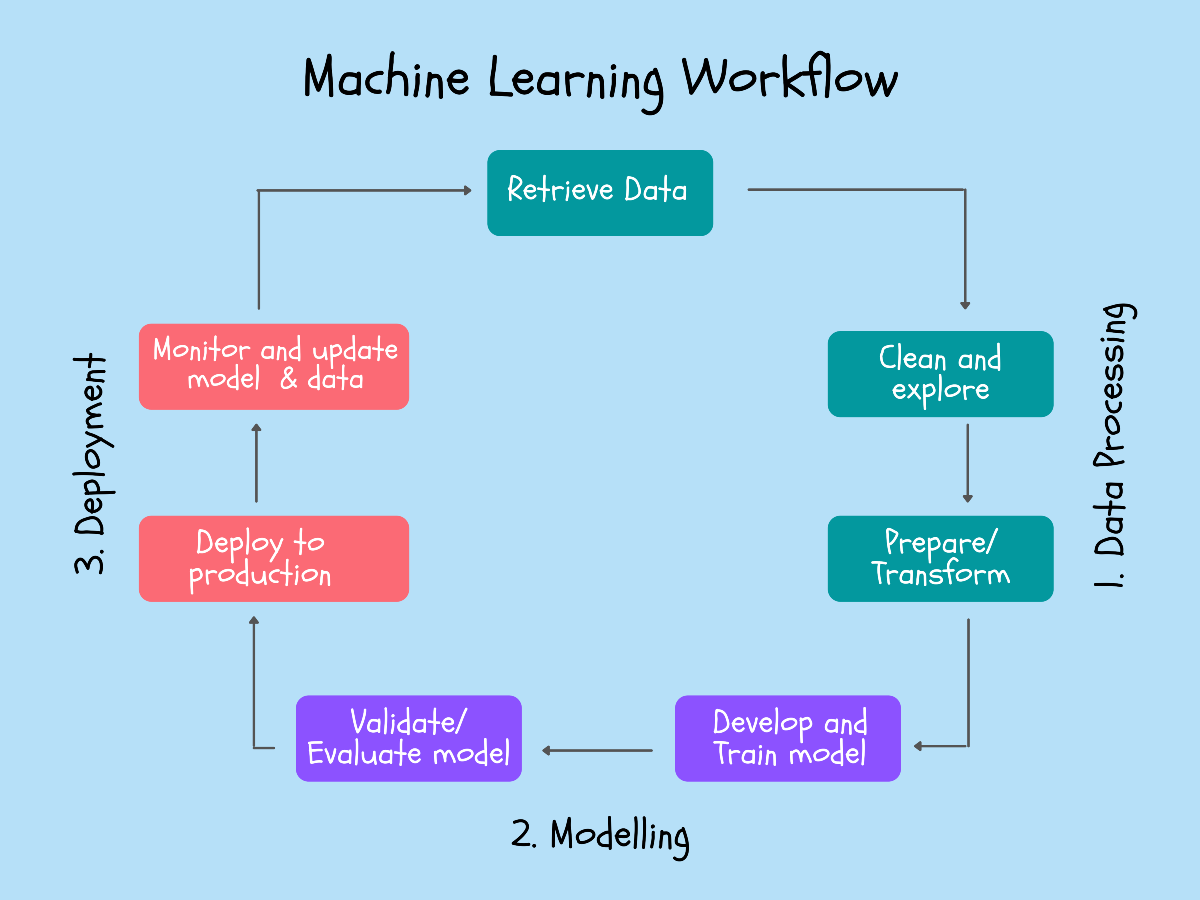
1. AWS



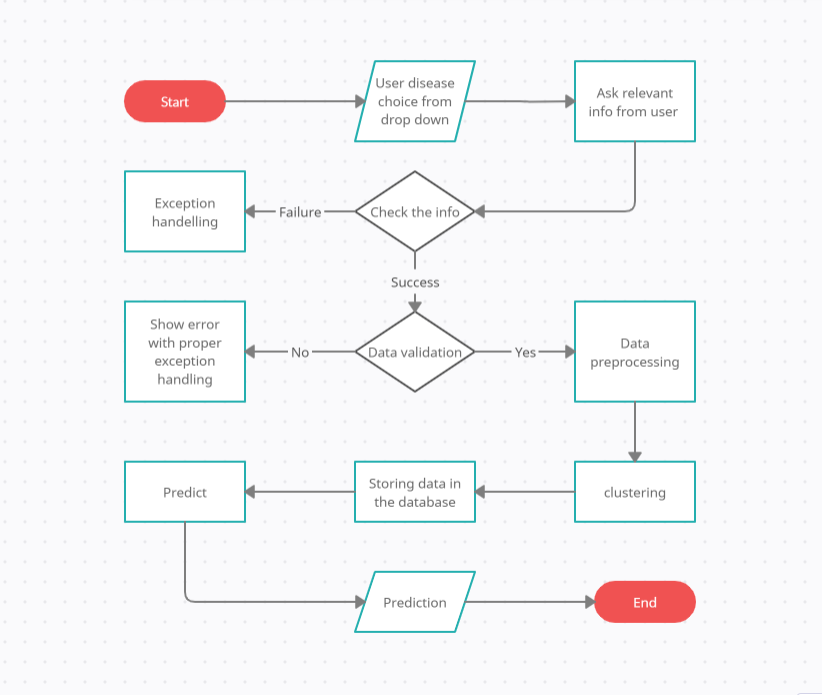
# Technology stack

|  |  |
| --- | --- |
| **Programming Language** | Python |
| **Project** | Vscode |
| **Database** | MongoDB |
| **Deployment** | AWS |

# Model training/validation workflow



# User I/O workflow

****

# Test cases

|  |  |  |  |
| --- | --- | --- | --- |
| Test case | Steps to perform test case | Module | Pass/Fail |
|  |  |  |  |