Predictive Maintenance Report

Hong Phuc Nguyen Nigel Yong Sao Young

Abstract

Our goal is to predict when different components of CNC machines will fail. We use time-series telemetry data obtained from sensors which include failure history, maintenance history, machine conditions and features such as the machine's age and model. This is a supervised learning problem in which we will use examples of failures and periods of healthy operations to discern the difference between the two states. The dataset was published on GitHub in 2018 with 5 CSV files and is around 77 Mb. We will perform EDA and feature engineering to eventually implement 3 algorithms: Isolation Forest, Random Forests and Extreme Gradient Boosting. We will then evaluate the results based on different metrics such as accuracy, recall and F1 scores.

1. Introduction

A common problem faced by industries such as manufacturing is the costs that are associated with the delays in production due to mechanical failures. The solution for these businesses is to predict which machine might fail soon, so that they can have proper maintenance before failing, reducing the overhead of time of repairing after failure. Predictive maintenance could reduce maintenance costs 10-40%, reduce downtime by 50%, and lower equipment and capital investment by 3-5% by extending machine life. [2]

In this project, we will solve this particular business problem, which can be formatted as a multi-class classification problem solved by training a predictive model, which learns from past data collected from the machines themselves. Our main objective is to achieve a minimum of 90% accuracy, recall and F1 score. To create the model, we will first do Exploratory Data Analysis to try and find patterns in the data, which might help us create a better model and know about it's structure. Then, we will perform feature engineering and label construction from which we will train 3 different selected models - Isolation Forest, Random Forest, XGB trees - and evaluate them with different metrics. We aim to tune the model which performs the best and reach the goal aforementioned. In the following sections, we will walk you through the each steps.

Related work has been done by Jaya Mathew, Senior Data Scientist at Microsoft who wrote a paper on "Building a Scalable Telemetry Based Multiclass Predictive Maintenance Model in R" [3]. While they used the same dataset, they had different objectives and used different technologies. The evaluation of their models was not explicitly discussed.

2. Materials and Methods

In this section, we will elaborate on the steps we took to reach our objectives, which helped us maximize the prediction performance.

2.1. Dataset

To start working with the data, it's essential that we first understand the data. The dataset was a total of 77MB, with the full version available at 770MB. It had logs of 100 unique machines for 1 year. It was obtained from GitHub. [4] Here is an overview of the files:

- **Telemetry**
  Machine specific data such as vibration, voltage, rotation and pressure.
- **Maintenance**
  The logs of when different components of the machines were replaced, either due to regular inspection or failure.
- **Machine**
  Machine specific information such as age and model.
- **Errors**
  Logs of when errors happened to a specific component.
- **Failures**
  Logs of when a component of a machine failed.

2.2. Technologies Used

The primary programming language we used was Python with Jupyter Notebooks, which made it ideal to visualize graphs as well as training the model. We also used PySpark for training and scaling easily and Scikit-learn for hyperparameter tuning and evaluation. For plotting and data manipulation, we used Pandas, Numpy, Seaborn and Matplotlib.

2.3. Feature Engineering

Feature Engineering is conducted to remove redundant data and combine different data sources to create features that best describe a machine's health condition at a given point in time.

- **Lag Features**
  Divide the duration of data collection into time units where each record belongs to a single point in time for each asset. A common method is to pick a window size for the lag features to be created and compute rolling aggregate measures. Once the frequency of observations is set, we want to look for trends within

measures, over time/rolling windows to predict performance degradation.

- **Days since last replacement**
  Possible features from this data set can be the number of replacements of each component over time or to calculate how long it has been since a component has been replaced. Replacement time is expected to correlate better with component failures since the longer a component is used, the more degradation would be expected.
- **Label construction**
  Label all observation cycles within the failure warning window as failed. The prediction problem then becomes estimating the probability of failure within this window. As this scenario is a multi-class classification, each error will be established independently and joined after, each as a unique column.

2.4. Algorithms used

After the feature engineering steps, we decided to train a model on 3 algorithms:

- **Isolation Forest (IF)**
  It is an unsupervised learning algorithm for anomaly detection that works on the principle of isolating anomalies. Even though being a binary classification model, it was interesting to see how it performs. We implemented IF with an open source library in spark.[1]
- **Random Forest (RF)**
  It is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. We implemented RF with the spark library. [5]
- **Extreme Gradient Boosting (XGB)**
  Similar to Random Forest, it uses an ensemble of decision trees. However, the decision trees are weak learners instead of fully grown trees. Trees are added one at a time to the ensemble and fit to correct the prediction errors made by prior models. It usually performs better than Random Forests.

3. Results

In this section, we will elaborate on the result of the study, that is, observations we found from dataset analysis and comparisons of the algorithms performance.

3.1. Data Analysis

After understanding the general structure of the data, we then analyse the distributions of the data.

1. **Telemetry**

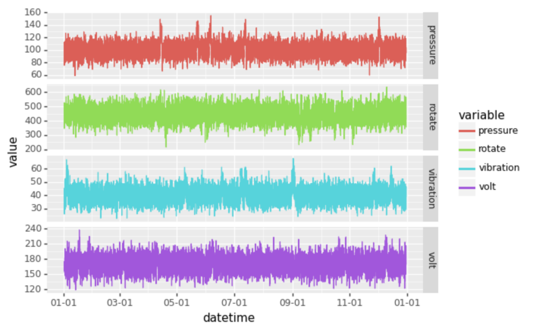


Figure 1: Telemetry distribution

Here is how pressure, rotation, vibration and voltage varies over the whole year. Telemetry has no specific pattern.

2. **Maintenance**

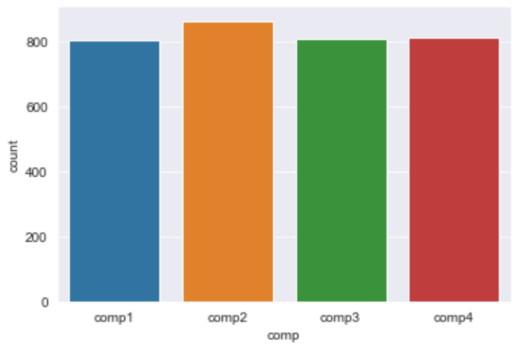


Figure 2: Maintenance distribution

Component 2 of the machines are slightly replaced more frequently.

3. **Machines**

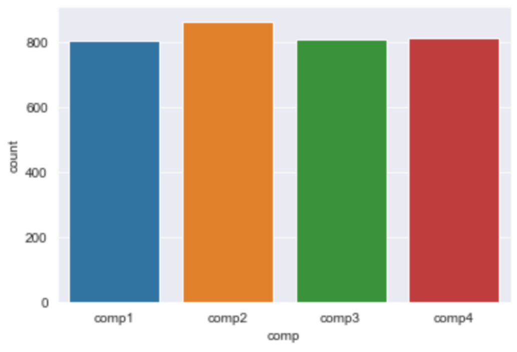


Figure 3: Maintenance distribution

The machines give some information on the type of model a machine is and its age. We observe that most machines are old, where the most common age is 14 years old, followed by 20 years old and 11 years old.

4. **Errors**

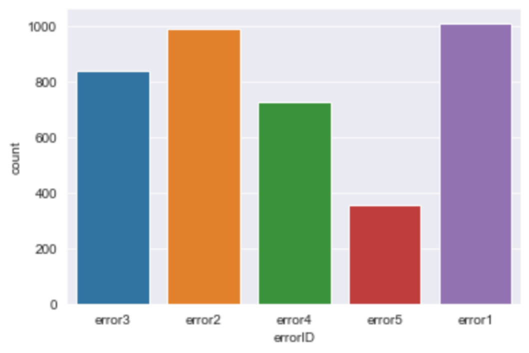


Figure 4: Errors distribution

We can observe that error1 and error2 are the most common.

5. **Failures**

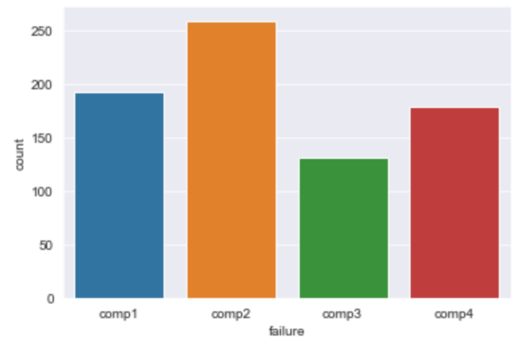


Figure 5: Failures distribution

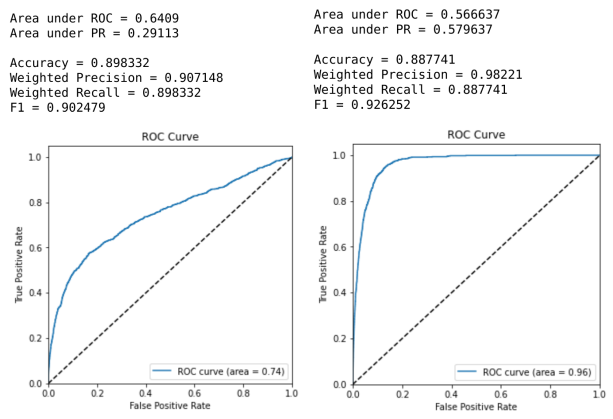
Component 2 is the one that fails the most.

3.2. Evaluation among models

It was essential to compare the models across different metrics other than accuracy because of the nature of the problem. Here we explain the results of the evaluation, as well as what made a difference in our performance. See Appendix for more. 10

- **Time Window**
  There was one variable with completely gave the models different results: time window range. We initially tried with 12 hours and the data reduced by 400%. However, the best recall we got was around 70%. We then tried smaller intervals and 3 hours gave us astounding results. It is a trade-off between space and performance.

  Here are the comparison for IF and RF:



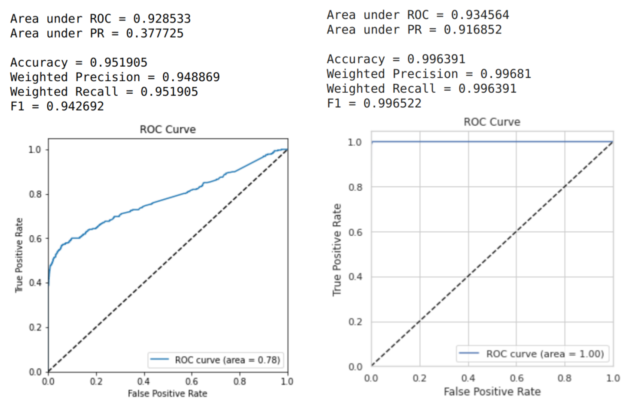
```
Area under ROC = 0.6409
Area under PR = 0.29113

Accuracy = 0.898332
Weighted Precision = 0.907148
Weighted Recall = 0.898332
F1 = 0.902479
```

```
Area under ROC = 0.566637
Area under PR = 0.579637

Accuracy = 0.887741
Weighted Precision = 0.98221
Weighted Recall = 0.887741
F1 = 0.926252
```

Figure 6: IF (3h) & IF (12h)



```
Area under ROC = 0.928533
Area under PR = 0.377725

Accuracy = 0.951905
Weighted Precision = 0.948869
Weighted Recall = 0.951905
F1 = 0.942692
```

```
Area under ROC = 0.934564
Area under PR = 0.916852

Accuracy = 0.996391
Weighted Precision = 0.99681
Weighted Recall = 0.996391
F1 = 0.996522
```

Figure 7: RF (3h) & RF (12h)

Our final results for all 3 algorithms are the following, indicating the metrics of failure prediction for the minority classes:

Models	Accuracy	Precision	Recall	F1
IF	88.7	≈ 13	≈ 92	≈ 23
RF	99.4	≈ 85	≈ 80	≈ 80
XGB	99.9	≈ 90	≈ 95	≈ 95

Table 1: Model Accuracy Comparison table. (In percentages)

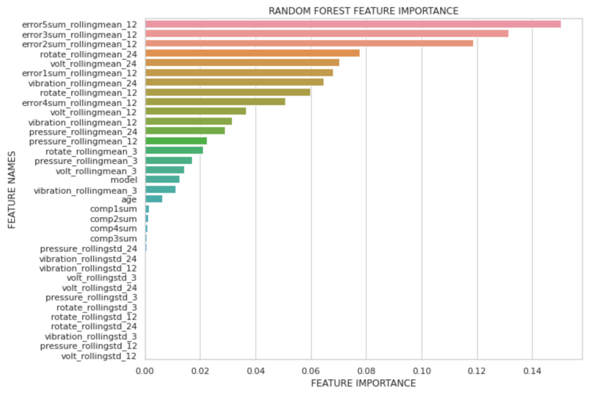
At this point, we concluded that the best model was XGB followed by RF.

- **Isolation Forest**

  Isolation Forest only works for the case of binary classification and doesn't take account the label of data-point. IF is not able to detect the type of failure component. The performance is above average and has the lowest precision and accuracy percentage among all three models.

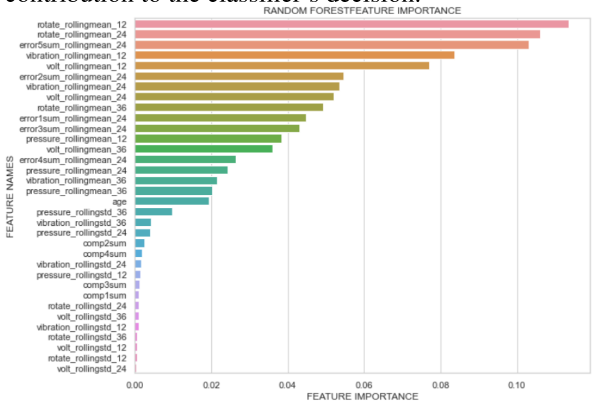
- **Random Forest**

  Random Forest provides good performance generally. The accuracy is significantly high however the recall score is just about 90%. From the trained model, error frequencies of error5, error3 and error2 are the most important features. The age is less important than we initially thought.



- **Extreme Gradient Boosting**

  XGB has an extremely high accuracy and recall score. However, the usage is complicated as the model requires intensive hyperparameter tuning. The model provides various tools for analysis and has a faster training speed. Errors frequency, model type and rotation speed are the most important features. Noticeably, the error of component 5 has a significant contribution to the classifier's decision.



4. Discussion

In this section, we will reflect on the relevance of the solutions, limitations we found as well as possible future work.

4.1. Relevance of solutions

Time window interval: the interval is essential to maintain sufficient information. The interval of 12 hours reduces the data storage by 4 times, however, it causes underfitting and the recall is just about 60%. A time window of 3 hours provides the necessary information to maximize minority class prediction. The recall, precision and f1 metrics are boosted to around 95%.

Pipeline: Spark ML provides an API to build a complete pipeline from raw data pre-processing to training and deployment. It takes us 40-45 minutes, if the entire data fit in the memory, to run whole pipeline process.

Spark streaming: Apache Spark Streaming enables scalable, high-throughput, fault-tolerant stream processing of live data streams. By adding streaming, we can provide an online prediction for a new batch of data, creating a self-updated model.

Scaling: the process is simple with a spark pipeline. Experimenting with full dataset of 770MB, we observed an improvement in the recall by 1% (with 12h time window) but it takes 5 times more time to train.

4.2. Limitations

Data centralization: the training for a subset of the full data take place in main memory. Especially in the scaling process, techniques such as caching, heap extension and swapping have to be applied to provide in memory-computation. This is a temporary solution and we will need to extend the training process for realistic data.

4.3. Possible future work

Distributed training: for a larger amount of data, the solution is to parallelize computation on large clusters.

Automatic streaming: the current system only supports TCP-IP basic communication. The system can be enhanced with a more advanced communication protocol and synchronization between remote nodes/clusters.

5. Conclusions

Prediction maintenance is very important for any aspect of manufacturing industries. All in all, we are satisfied in the project's outcome as we exceeded our objectives and our XGB model got over 90% in all the metrics. All the code can be found here. The main takeaways form the project are:

- Component 2 has the most failure ratio. The previous errors have the potential to lead to future failures.

- XGB is the best tool for failure detection. It provides very high metrics score but requires intensive tuning.
- The data for failure prediction is significantly imbalanced. Despite breaking data distribution, downsampling the majority class improves the failure recall from 3-5%.
- Scaling up and fast deploying is made easy with pipeline and spark streaming.
- The time window is crucial in feature engineering. We need a perfect balance of enough information, but not too much.

```
              precision    recall  f1-score   support

           0      1.000     0.999     1.000     73131
           1      0.870     0.968     0.916       277
           2      0.996     0.993     0.994       541
           3      0.959     0.929     0.944       225
           4      0.951     0.988     0.969       335

    accuracy                          0.999     74509
   macro avg      0.955     0.975     0.965     74509
weighted avg      0.999     0.999     0.999     74509
```

The diagram above is a more detailed metrics evaluation of the XBG model.

References

[1] Isolation forest. 2

[2] McKinsey Company. The internet of things: Mapping the value beyond the hype. 1

[3] Jaya Mathew. Building a scalable telemetry based multiclass predictive maintenance model in r. 1

[4] Schwan. Dataset source, Jan 2018. 1

[5] Spark. Random forest spark. 2

6. Appendix

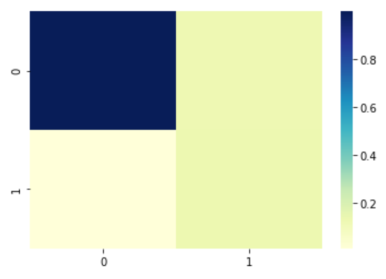


Figure 8: IF Confusion Matrix

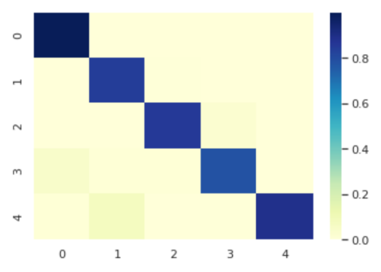


Figure 9: RF Confusion Matrix

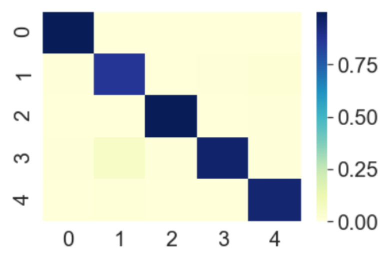


Figure 10: XGB Confusion Matrix