Fault Prediction

Machine Learning Internship Summer 2021

Vishal Rishi MK

Mentored by Saya Date, Michael Wu, Meenal Pathak, and Santosh Ganti

Problem Statement and Objectives

Problem Statement :

Build an efficient learning algorithm that learns patterns from asset failures in the past and predicts the probability of an asset failure in a specified future time window

Objectives :

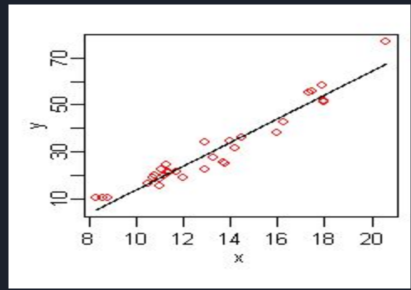
- The algorithm should have good predictive power on unseen examples
- It should learn the underlying pattern in the data and not just overfit the data
- It should be generalizable across all similar machines

linecraft.ai

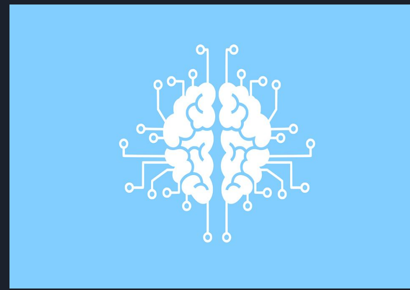
Process flowchart :

1. Data collection

2.Feature engineering



3.Baseline evaluation



4.Training a GRU

5.Cross validation

6.Interpretation - Model agnostic methods

7.Test across similar machines

Challenges :

- Class Imbalance (Percentage of faults in the entire time range was ~ 20 %)
- Particularly, the OP30 MC and OP20 MC had the worst imbalance - the percentage of faults was ~ 5 %
- Because of the imbalance, the train data would not be representative of the test data
- There is a need for sophisticated algorithms that can identify the anomalies

Data preprocessing :

- A time window is chosen (either 15, 30, or 120 minutes)
- An optimal time window is one that reduces the class imbalance in the dataset
- For a particular asset, the entire VPI duration is split into several time windows
- For each time window, we collect several features that could be used to predict fault occurrence in the future
- DataBase : PSS SEGMENT 1 AUGUST 2020

Features used :

1. Time from the previous fault
2. Number of faults in the time window
3. Average duration of faults in the time window
4. Number of cycles in the time window

linecraft.ai

Baseline 1 :

- For an asset, to predict whether a fault would occur in the next time window, we collect the “average duration of faults” for the previous 50 time windows and compute their average
- If the average is greater than the MTTR of the asset, the baseline outputs “1” (fault occurrence in the next window). Else, it outputs “0” (no fault occurrence in the next window)

Baseline 2 :

- For an asset, to predict whether a fault would occur in the next time window, we collect the “time from previous fault” feature for the current time window

$$baseline2(time\ from\ previous\ fault) = \begin{cases} 0 & if\ (MTBF - time\ from\ previous\ fault) \geq duration\ of\ time\ window \\ 1 & otherwise \end{cases}$$

Results on OP30 Machines :

Baseline 1 :

Window	120 minutes				30 minutes				15 minutes			
Machines	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
Precision	0.49	0.30	0.23	0.41	0.29	0.25	0.20	0.19	0.18	0.17	0.14	0.12
Recall	0.49	0.32	0.43	0.48	0.42	0.47	0.46	0.46	0.33	0.39	0.42	0.38
F1 score	0.49	0.31	0.30	0.44	0.34	0.33	0.28	0.27	0.24	0.23	0.21	0.18
Accuracy	0.61	0.69	0.59	0.70	---	---	---	---	---	---	---	---

For windows of 15 and 30 minutes, the datasets were highly imbalanced.
This makes the accuracy metric irrelevant

Results on OP30 Machines :

Baseline 2 :

Window	120 minutes				30 minutes				15 minutes			
Machines	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
Precision	0.38	0.17	0.17	0.22	0.11	0.05	0.04	0.05	0.05	0.02	0.03	0.03
Recall	1.0	0.50	0.57	0.68	0.18	0.04	0.08	0.09	0.08	0.02	0.05	0.05
F1 score	0.55	0.25	0.25	0.33	0.13	0.06	0.05	0.06	0.06	0.02	0.03	0.03
ROC AUC	0.50	0.42	0.42	0.46	0.45	0.4	0.45	0.5	0.45	0.45	0.4	0.5

For windows of 15 and 30 minutes, the datasets were highly imbalanced.
This makes the accuracy metric irrelevant and ROC AUC is chosen

Recurrent Neural Networks - Single machine

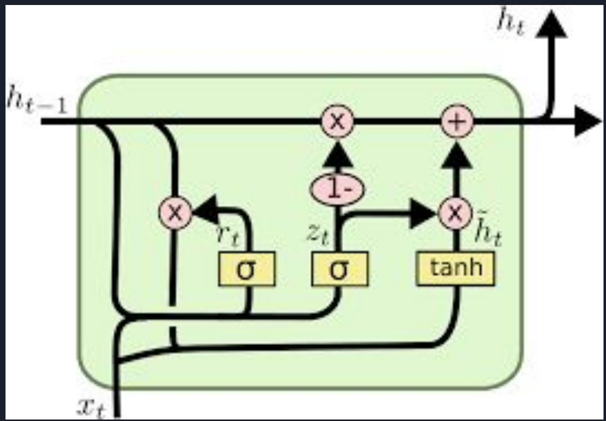
- We use two GRU hidden layers with 20 neurons in each layer
- The output layer has a single neuron with ‘sigmoid’ activation
- Binary cross entropy - loss function (classification problem)
- The dataset is reshaped as

  (*batch size, no. of windows, no. of features*)

  No. of windows = 50 No. of features = 4

- For each sequence of time windows, we have a target - 1 if next window has fault occurrence, 0 otherwise
- Shape of the targets : **(*batch size, 1*)**

Gated Recurrent Units - Tackling the short term memory problem



- Due to the transformations that the data goes through when traversing an RNN, some information is lost at each step
- Hence, processing large sequences are harder
- To tackle this problem, LSTM cells have been introduced.
- There are many variants of LSTM. One particularly popular variant is the GRU cell

Results on OP30 Machines :

linecraft.ai

Window	120 minutes			
Machines	**M1**	M2	M3	M4
Precision	**1.0**	0.66	0.57	0.62
Recall	**0.93**	0.50	0.57	0.68
F1 score	**0.96**	0.57	0.57	0.65
ROC AUC	**0.97**	0.62	0.59	0.67

- The model performs well on the test set of OP30 M1 but could not generalise across machines
- Time window of 120 minutes was chosen because it lead to a relatively balanced dataset (40:60)

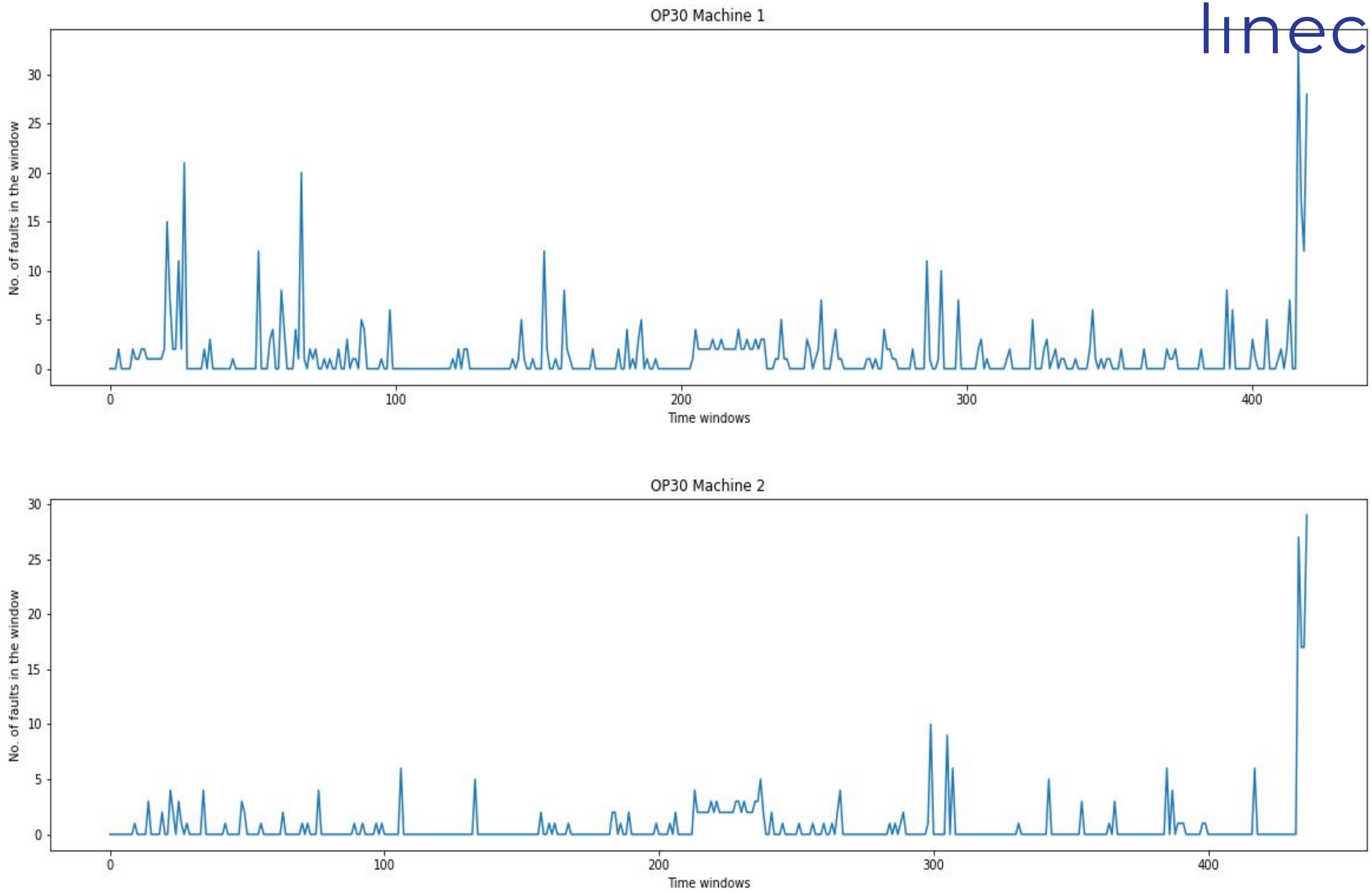
The model was trained on data from OP30 M1. Results shown here are obtained after testing the model on test set of M1. Also, the entire data for M2, M3, M4 are unseen examples

Gradients with respect to inputs :

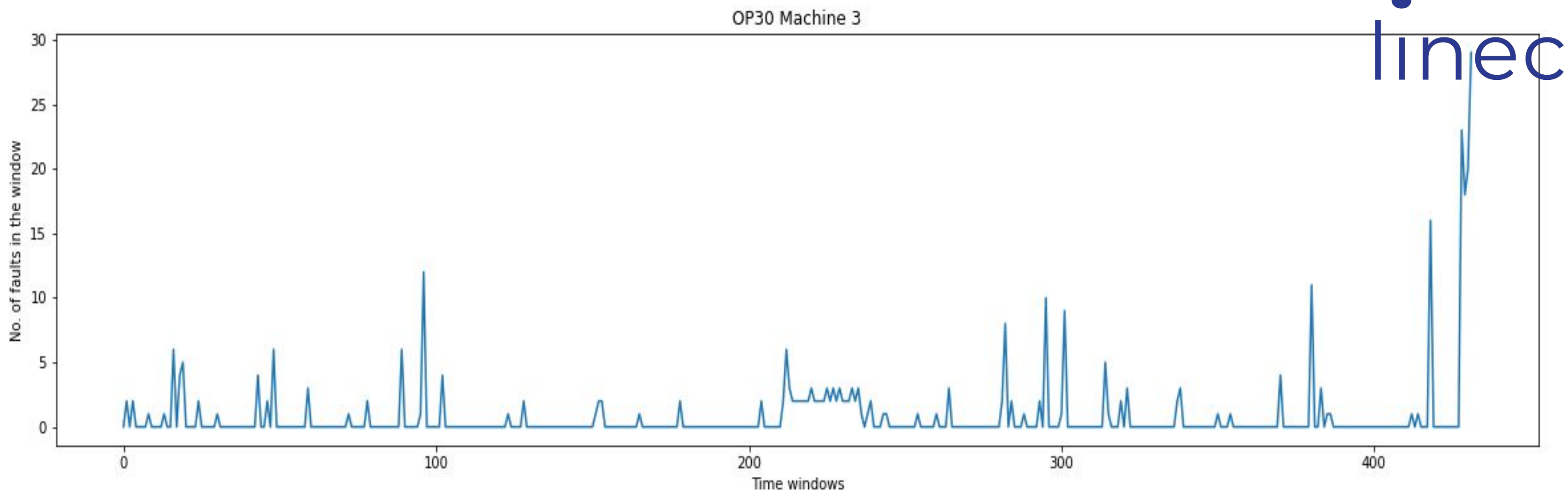
The average gradients of the output with respect to the four input features

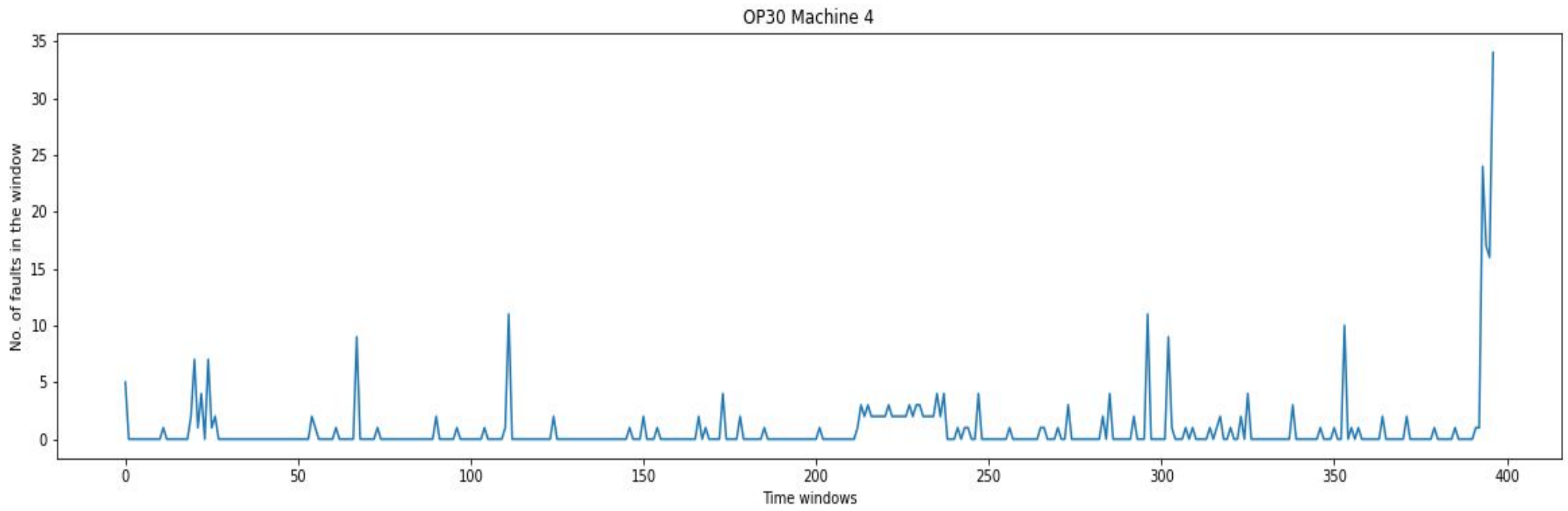
[0.15554674, 0.25962123, 0.45405376, -0.07183856]

- The model, trained on OP30 M1 data, is very sensitive to the features - “Number of faults”, “Average duration of faults”
- Because of this, the “Average duration of faults” feature was chosen for Baseline 1
- This particular feature was very different for M2, M3, and M4. This could be a reason for the lack of generalizability
- The “Number of cycles” feature seems to have negligible effect on the output


OP30 Machine 1
No. of faults in the window
Time windows
OP30 Machine 2
No. of faults in the window
Time windows

linecraft.ai





Differences between the machines :

	Average Time between spikes	Percentage of faults	Average duration of faults
M1	3.2 hours	38.2 %	14 minutes
M2	7.4 hours	21.2 %	13.5 minutes
M3	7.8 hours	20.3 %	13 minutes
M4	6.4 hours	23.4 %	17 minutes

These differences between machines does not help the model in generalizing. The frequent spikes in OP30 M1 has made the model predict a lot of false positives in M2, M3, M4. This is indicated by the low precision scores

linecraft.ai

Model per Machine : Results on OP30

Window	120 minutes			
Machines	M1	M2	M3	M4
Precision	1.0	0.95	1.0	1.0
Recall	0.93	1.0	0.97	0.83
F1 score	0.96	0.97	0.98	0.90
ROC AUC	0.97	0.98	0.96	0.92

- When we train a separate model for each machine, the model is able to learn patterns that are specific to the machine
- All the results are the average of the scores obtained from 5-fold cross validation

But a separate model for each machine may not be feasible in all cases !

Multi-label Classification Approach

Multi-label Classification Approach :

linecraft.ai

- We train a model that gets past features of all similar machines as inputs
- It outputs the probability of asset failure for each individual machine
- The dataset is reshaped as

  (***batch size, no. of windows, no. of features*no. of machines***)

  No. of windows = 50   No. of features = 4

- Shape of the targets : (***batch size, no. of machines***)

Results on OP30 Machines :

Multi-label Classification :

Window	120 minutes				30 minutes				15 minutes			
Machines	M1	M2	M3	M4	M1	M2	M3	**M4**	M1	M2	M3	M4
Precision	1.0	1.0	1.0	1.0	1.0	1.0	1.0	**0.8**	1.0	1.0	0.95	1.0
Recall	0.94	0.77	0.9	0.91	1.0	1.0	0.83	**0.37**	0.92	0.86	0.90	0.92
F1 score	0.96	0.87	0.94	0.95	1.0	1.0	0.91	**0.51**	0.95	0.92	0.92	0.95
Accuracy	0.97	0.94	0.97	0.97	---	---	---	---	---	---	---	---

The results for OP30 M4 (30 minutes window) are not good. We can use a separate model for that particular setting

Testing on other Machines :

Results on OP20 Machines :

Window	Baseline 1		Baseline 2		Multi-label classification	
Machines	M1	M2	M1	M2	**M1**	**M2**
Precision	0.43	0.33	0.27	0.17	**1.0**	**0.91**
Recall	0.43	0.38	1.0	0.47	**1.0**	**1.0**
F1 score	0.43	0.35	0.43	0.24	**1.0**	**0.95**
ROC AUC	0.59	0.58	0.5	0.42	**1.0**	**0.97**

A time window of 120 minutes was found to be the optimal option. Since the multi-label classification model performed well on both machines, there is no need for separate models

Testing on other Machines :

Results on OP10 Machines :

Models	Baseline 1			Baseline 2			Multi-label classification		
Machines	M1	M2	M3	M1	M2	M3	**M1**	**M2**	**M3**
Precision	0.61	0.49	0.36	0.60	0.56	0.31	**1.0**	**1.0**	**1.0**
Recall	0.45	0.42	0.36	1.0	1.0	0.57	**1.0**	**0.95**	**0.9**
F1 score	0.52	0.45	0.36	0.75	0.71	0.41	**1.0**	**0.97**	**0.94**
ROC AUC	0.49	0.44	0.52	0.5	0.5	0.46	**1.0**	**0.97**	**0.95**

A time window of 120 minutes was found to be the optimal option. Since the multi-label classification model performed well on both machines, there is no need for separate models

Interpretation : Model Agnostic methods

What is Permutation Importance ?

The permutation feature importance is defined to be the decrease in a model score when a single feature is randomly shuffled. This procedure breaks the relationship between the feature and the target, thus the drop in the model score is indicative of how much the model depends on the feature

- In our case, we permute a particular feature (out of the four) in the test set of a machine
- Once they are permuted, we compute the metrics on the permuted test set using the pre-trained model for the machine
- This procedure is repeated 150 times
- Steps 1 - 3 is done for all the features
- The reduction in the metric value indicate feature importance

Permutation importance - OP30 M2

	OP30 M2 - 120 minutes			
Feature	F1	**F2**	F3	**F4**
ΔPrecision	0.43	**0.72**	0.58	**0.76**
ΔRecall	0.71	**0.85**	0.72	**0.87**
ΔF1 score	0.64	**0.81**	0.66	**0.84**
ΔROC AUC	0.35	**0.35**	0.43	**0.45**

F1 - Time from previous fault

F2 - Number of faults in the window

F3 - Average fault duration

F4 - Number of cycles in the window

The values reported are the reduction in the metric value from the original score obtained when the test set was not permuted

Permutation importance : OP30 M3 & OP30 M4

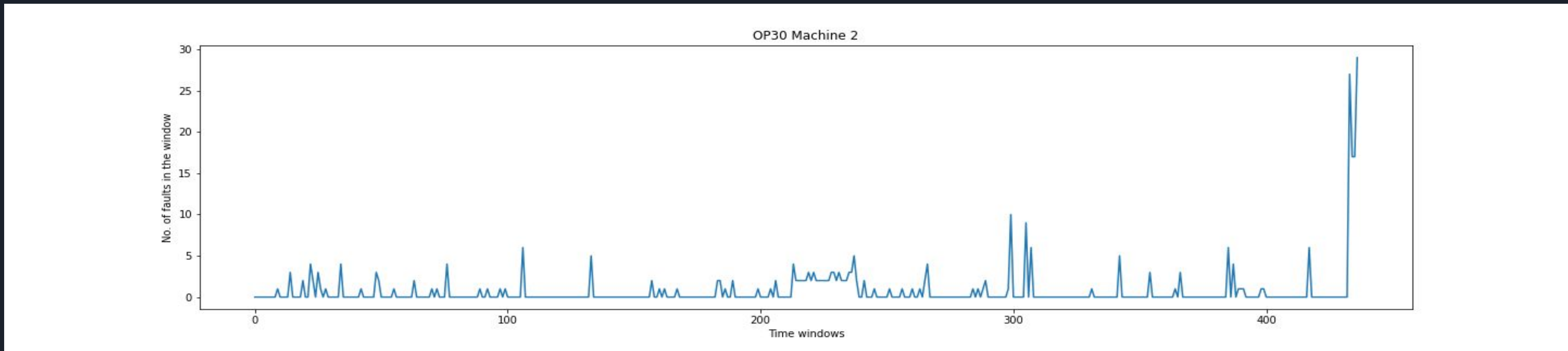
	M3 - 120 minutes			
Feature	F1	**F2**	**F3**	**F4**
∆Precision	0.44	**0.72**	**0.67**	**0.76**
∆Recall	0.75	**0.91**	**0.92**	**0.84**
∆F1 score	0.66	**0.85**	**0.87**	**0.82**
∆ROC AUC	0.41	**0.56**	**0.63**	**0.56**

	M4 - 120 minutes			
Feature	**F1**	F2	F3	**F4**
∆Precision	**0.87**	0.76	0.81	**0.89**
∆Recall	**0.79**	0.59	0.75	**0.83**
∆F1 score	**0.84**	0.72	0.78	**0.87**
∆ROC AUC	**0.58**	0.39	0.53	**0.67**

Columns in red, indicate strong relationship between the feature and the target

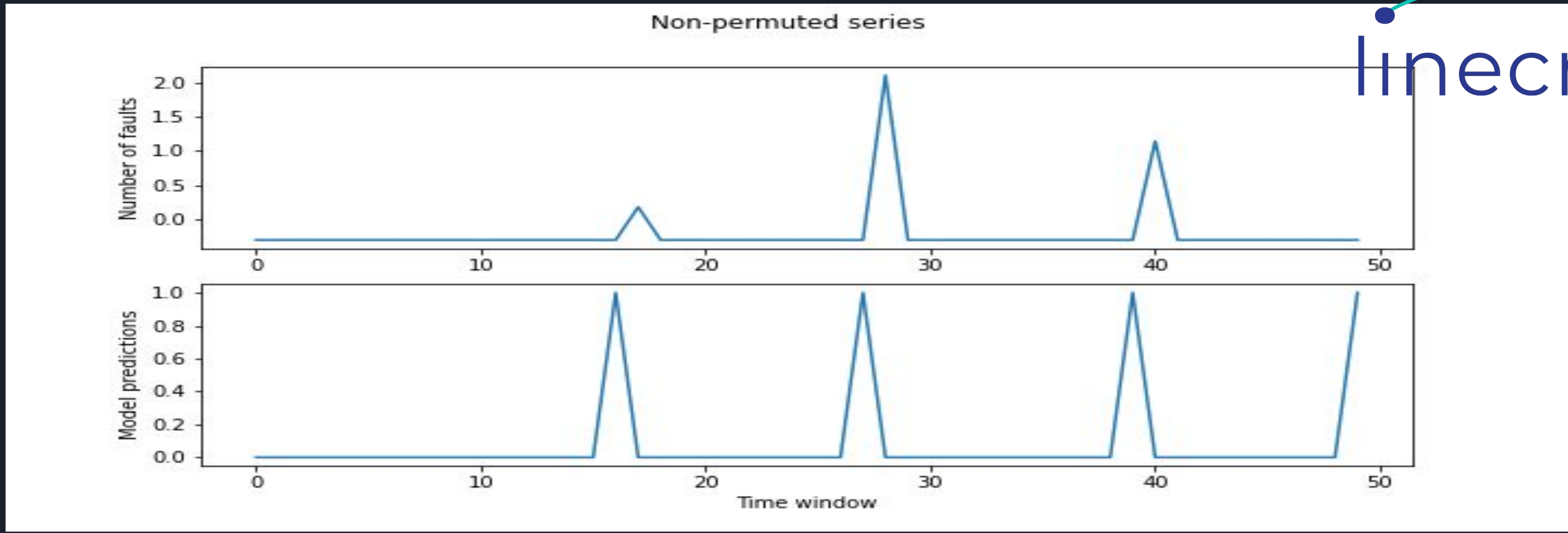
Observations :

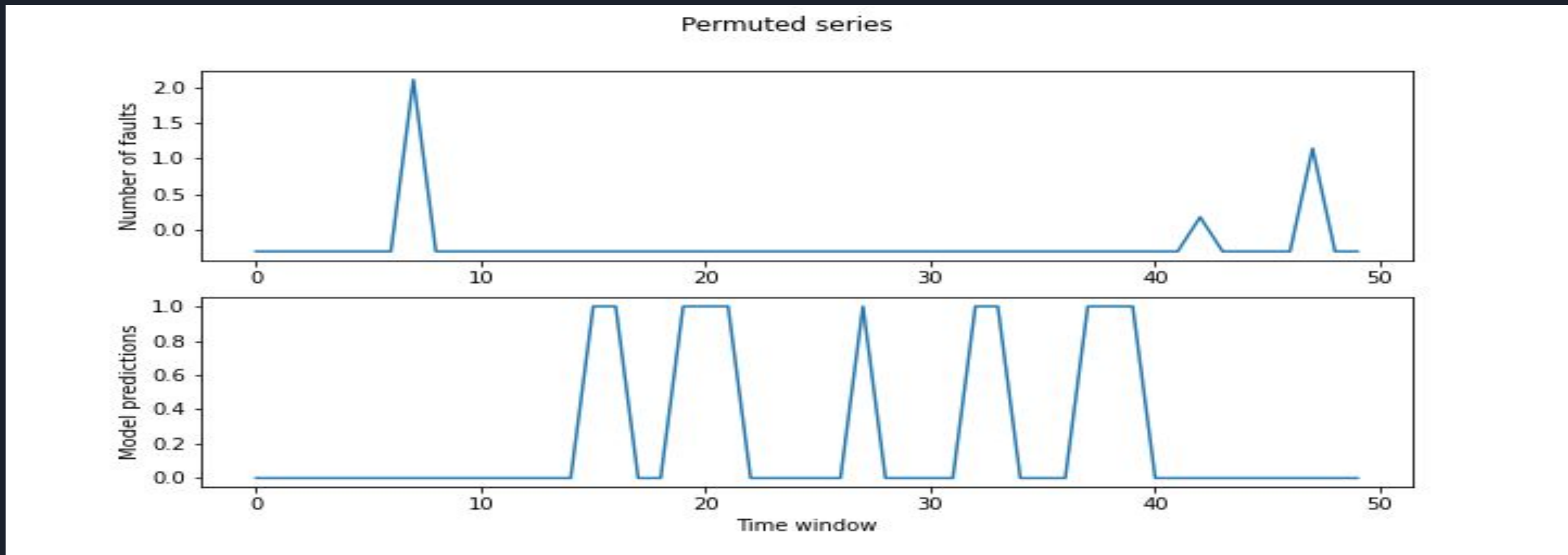
- Since rearranging the feature values has a drastic impact, we can intuitively say that the model has found patterns in past failure data and if the pattern is shuffled, it can confuse the model
- The “Number of faults” feature has a strong relationship with the targets (also indicated by the gradients). Hence, failure patterns are learnt from this feature

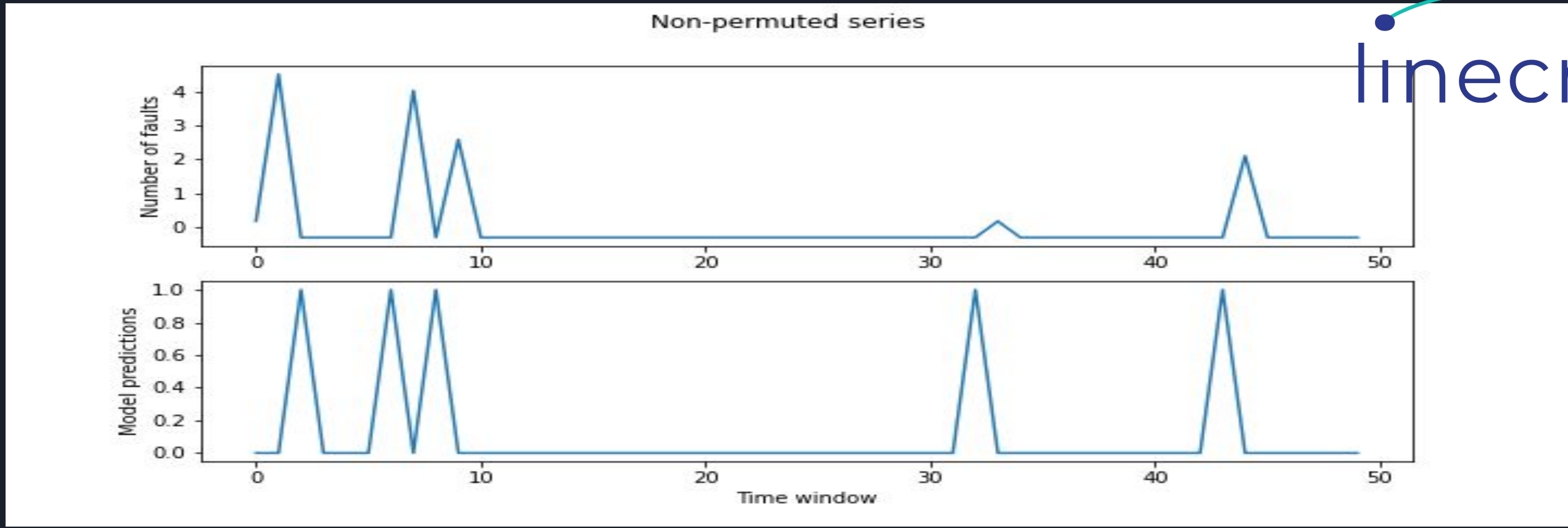


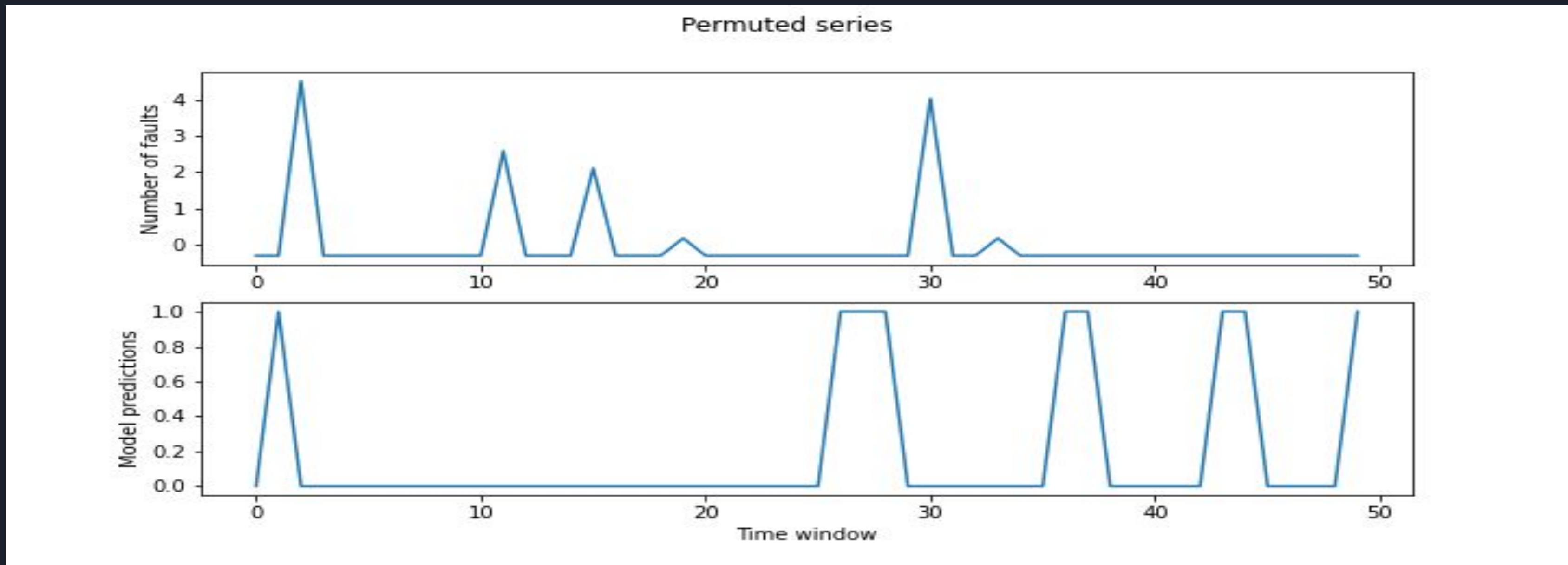
Observations :

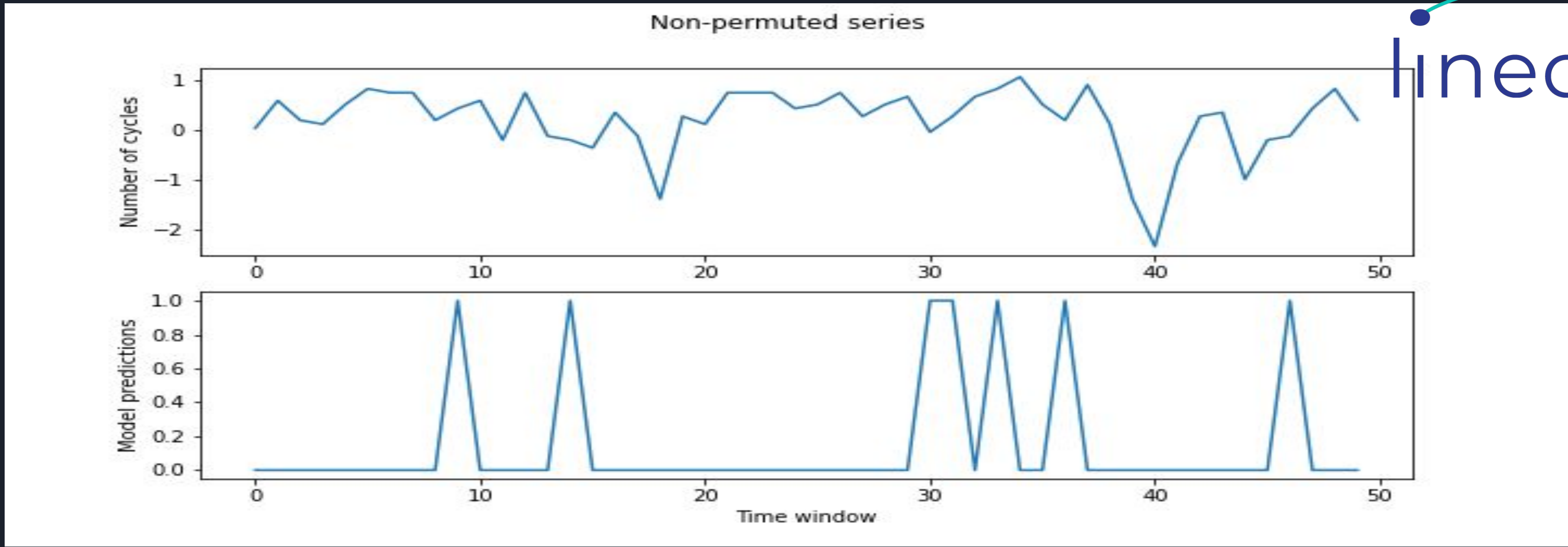
- The pattern learnt, in this case, is the “time between spikes”
- Spikes are indicative of asset failure
- The model is able to learn when the next spike (asset failure) is going to happen.
- Also, permuting the “Number of cycles” feature does have a very serious impact (which was not indicated by the gradients)

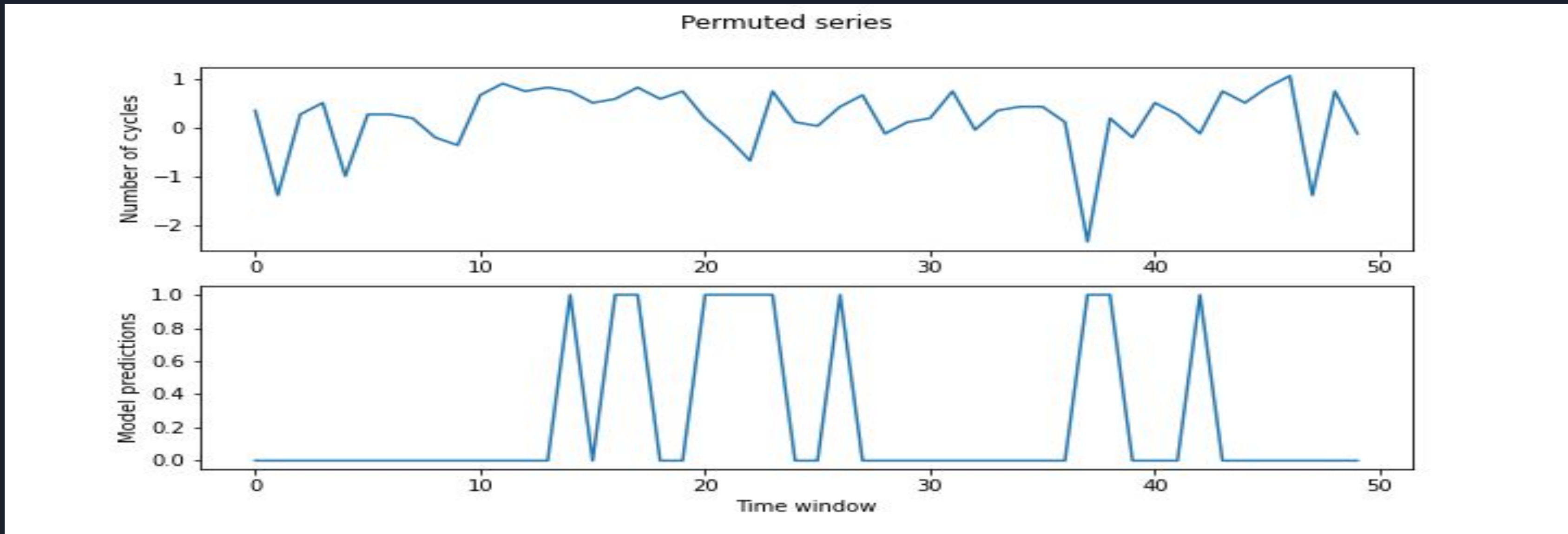

Non-permuted series
Number of faults
Model predictions
Time window

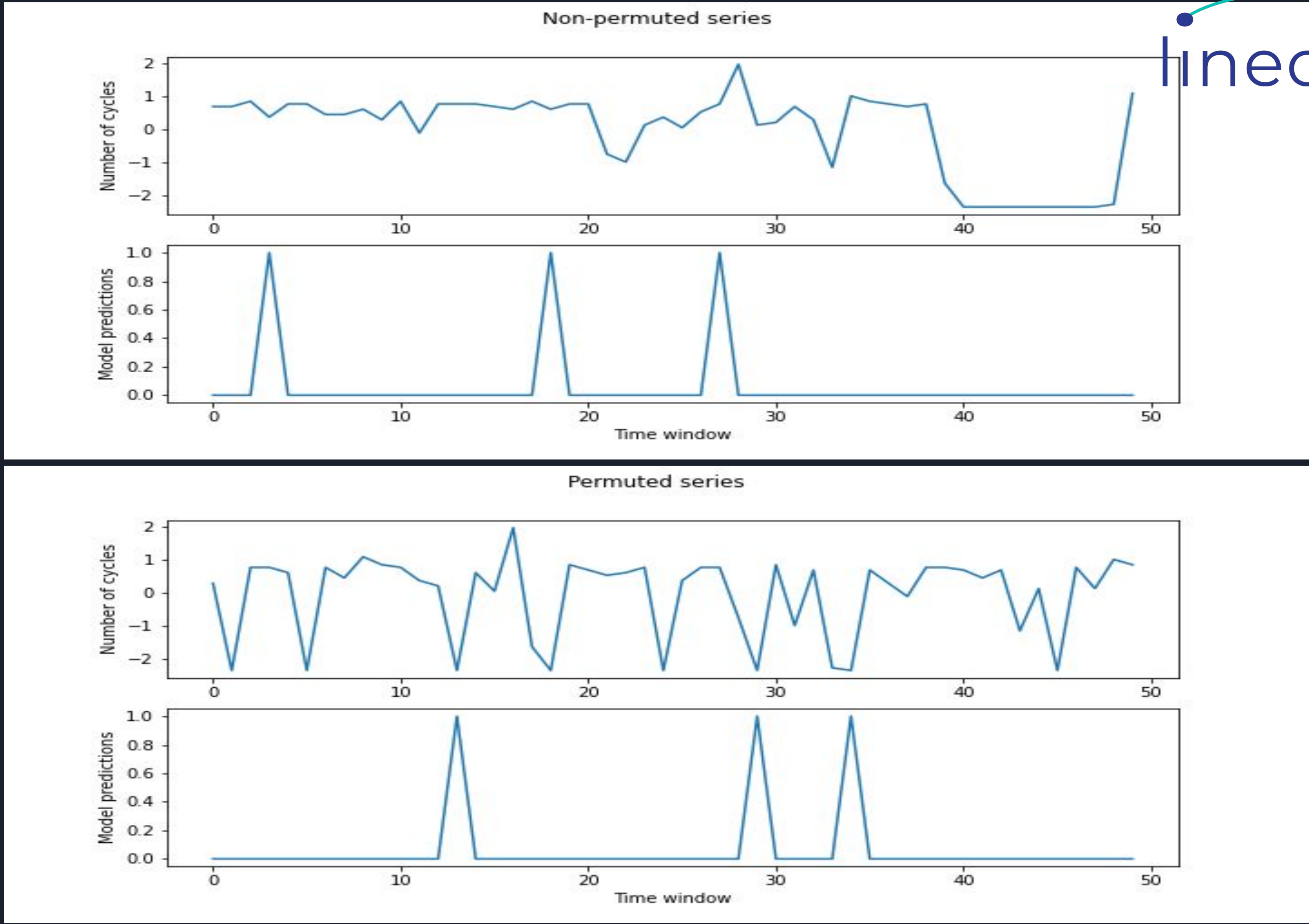

Permuted series
Number of faults
Model predictions
Time window


Non-permuted series
Number of faults
Model predictions
Time window


Permuted series
Number of faults
Model predictions
Time window


Non-permuted series
Number of cycles
Model predictions
Time window


Permuted series
Number of cycles
Model predictions
Time window



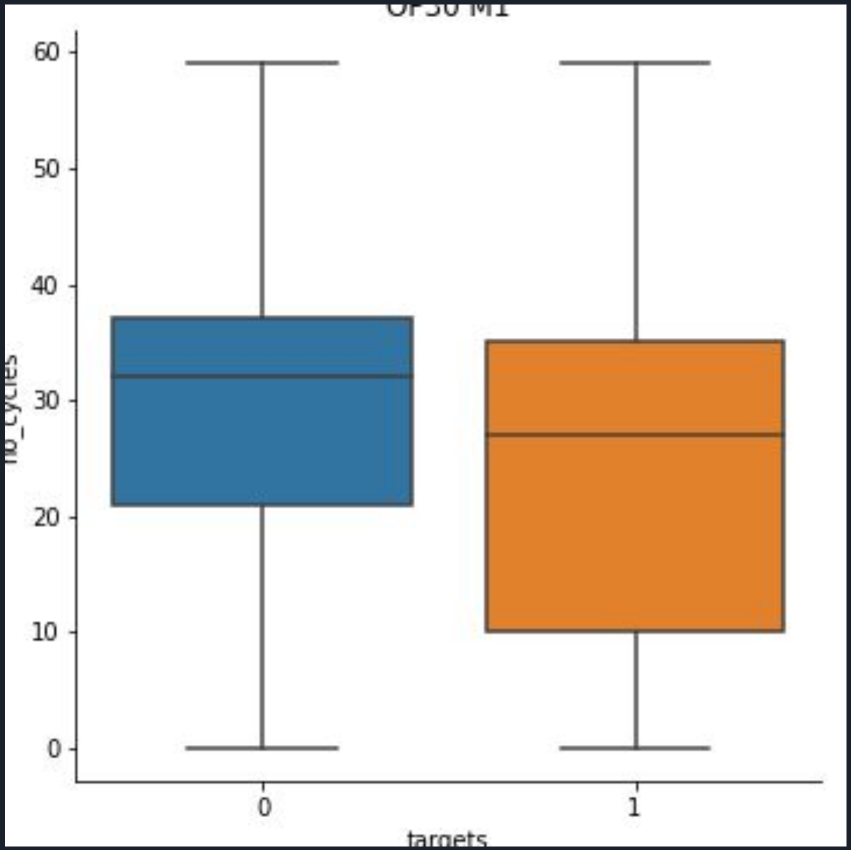
Non-permuted series
Number of cycles
Model predictions
Time window
Permuted series
Number of cycles
Model predictions
Time window

Frequency distribution of the “Number of cycles” feature for each class

linecraft.ai

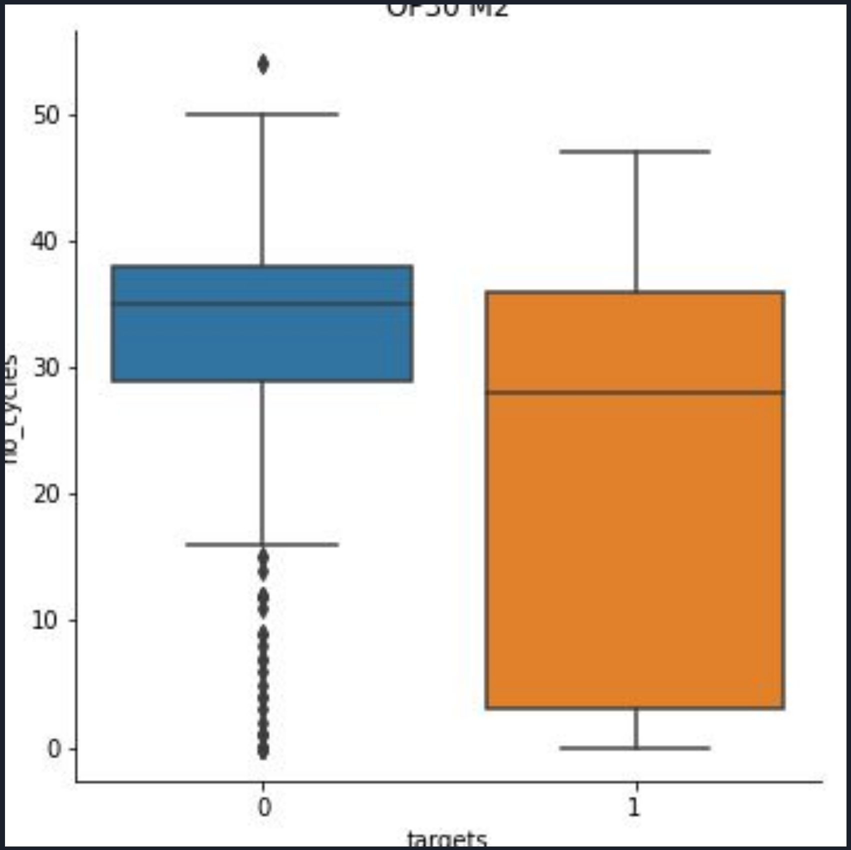
Number of cycles

OP30 M1



Targets

OP30 M2

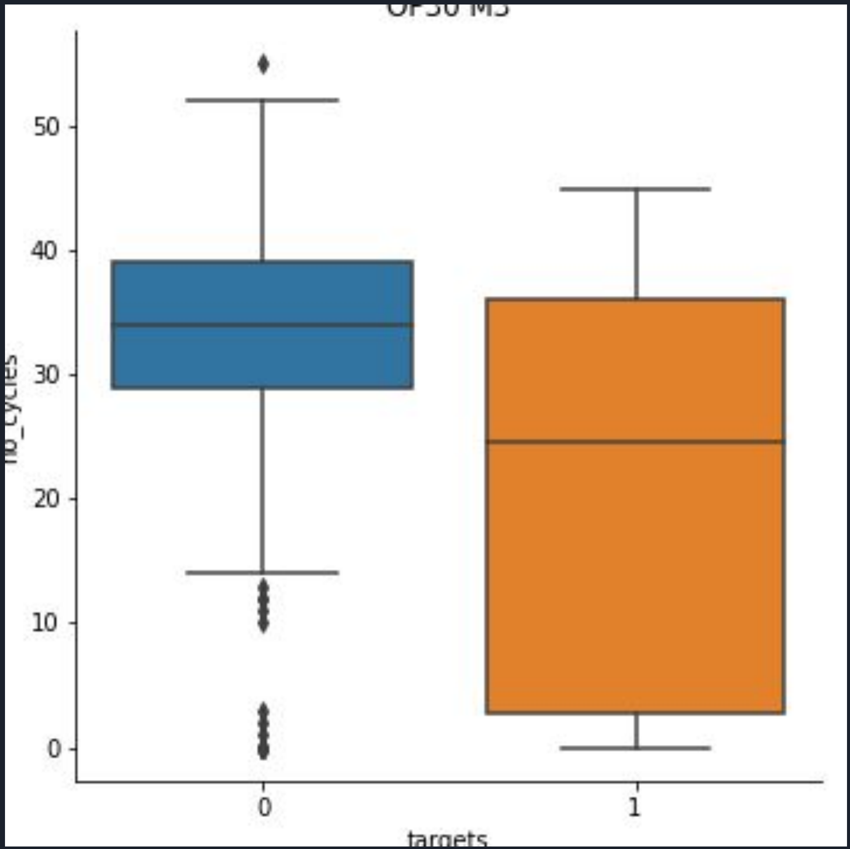


Targets

linecraft.ai

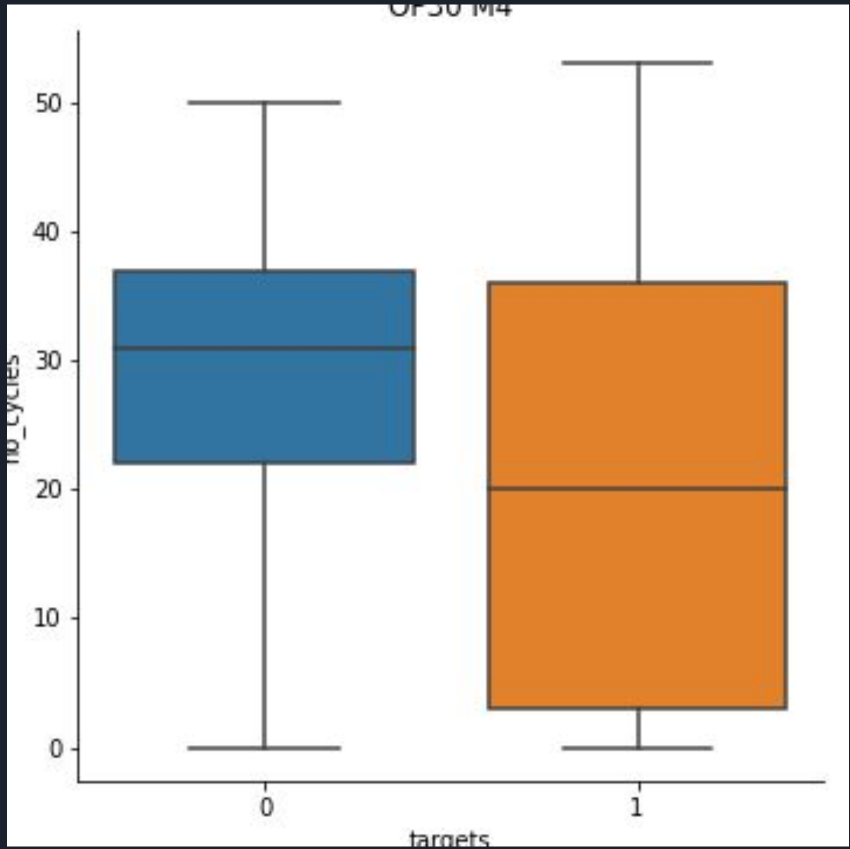
Number of cycles

OP30 M3



Targets

OP30 M4



Targets

More challenging Machines :

Results on OP30 MC :

Window	Baseline 1				Baseline 2				Multi-label classification			
Machines	M1	M2	M3	M4	M1	M2	M3	M4	M1	M2	M3	M4
Precision	0.24	0.05	0.02	0.02	0.07	---	0	0	1.0	1.0	0.95	0.99
Recall	0.56	0.21	0.09	0.11	0.01	0	0	0	**0.96**	**0.71**	**0.75**	**0.76**
F1 score	0.34	0.09	0.03	0.03	0.01	---	---	---	0.98	0.83	0.83	0.99
ROC AUC	0.62	0.47	0.40	0.42	0.5	0.5	0.5	0.5	0.9	0.7	0.8	0.7

The results for OP30 MC (30 minutes window) are not stable and exhibit high variability. This is due to the high degree of imbalance (only ~5 % faults). In that case, training separate models would be the solution

More challenging Machines :

Results on OP20 MC1 : 120 minutes

Window	Baseline 1	Baseline 2	Binary classification
Machines	OP20 MC1	OP20 MC1	**OP20 MC1**
Precision	0.43	0.28	**1.0**
Recall	0.43	1.0	**0.80**
F1 score	0.43	0.44	**0.88**
ROC AUC	0.59	0.50	**0.90**

A binary classification model was trained on OP20 MC1 data. The results did not show variability unlike OP30 MC. (Percentage of faults - ~ 25 %)

linecraft.ai

More challenging Machines :

Results on OP20 MC2 : 15 minutes

Window	Baseline 1	Baseline 2	Binary classification
Machines	OP20 MC2	OP20 MC2	**OP20 MC2**
Precision	0.08	0	**1.0**
Recall	0.16	0	**0.65**
F1 score	0.11	---	**0.78**
ROC AUC	0.52	0.49	**0.71**

A binary classification model was trained on OP20 MC2 data. Though a separate model was trained, the results showed variability. (Percentage of faults - ~ 5 %)

Limitations :

- Since the models were trained only on two months of data, the model could have learnt short term patterns. That would not be helpful in predicting faults in the long run
- This is still a blackbox model. We still could not accurately interpret the model, from a domain viewpoint
- Very few features are used. They might be helpful in uncovering short term patterns. But we need more reliable features for prediction in the long run
- For OP30 MC, and OP20 MC, the results show variability. The results are unstable when the imbalance in the data is very high (~ 5 %)

Future work :

- Include Process IOs, and self cycle time as features
- Train the model on more data so that it can learn long term patterns
- Increase the interpretability of the model
- Try One Class SVM as an anomaly detection algorithm, instead of viewing this as a sequence learning problem
- Train models that also outputs the type of faults
- Using special metrics instead of the traditional classification metrics
- Train models that outputs the time at which the next fault would occur - a hard version of this problem
- Look for methods that can tackle class imbalance

Thank you !!