

ANOMALY DETECTION

FOR WATER QUALITY

Agenda

1. Topic Introduction
2. Baseline: a Moving Average method
3. Precision Improvement with Fourier Transform
4. Vector Autoregressions



01

Introduction

Topic Presentation

GECCO 2018 CHALLENGE

The Genetic and Evolutionary Computation Conference took place in Tokyo in July 2018 and organized the following contest: **Anomaly Detection for Drinking Water Quality**.

The real world dataset was provided by the TFW organization which is a water provider in Germany. Some key informations:

- 11 columns and 13.9K rows
- 4 months of data
- data was collected from sensors on a minutely basis
- Flow rate and the Temperature **can not** be used as features
- The “EVENT” column is our binary label:
 - False means no anomaly

DATASET OVERVIEW

	Time	Tp	Cl	pH	Redox	Leit	Trueb	Cl_2	Fm	Fm_2	EVENT
0	2016-08-03T09:49:00Z	6.5	0.17	8.36	749.0	211.0	0.011	0.118	1677.0	695.0	False
1	2016-08-03T09:50:00Z	6.5	0.17	8.36	749.0	211.0	0.011	0.118	1561.0	696.0	False
2	2016-08-03T09:51:00Z	6.5	0.17	8.35	749.0	211.0	0.011	0.117	1581.0	696.0	False
3	2016-08-03T09:52:00Z	6.5	0.17	8.35	749.0	211.0	0.011	0.118	1579.0	693.0	False
4	2016-08-03T09:53:00Z	6.5	0.17	8.35	749.0	211.0	0.011	0.118	1567.0	689.0	False

- Time: Time of measurement, given in following format: yyyy-mm-dd HH:MM:SS
- Tp: The temperature of the water, given in °C.
- Cl: Amount of chlorine dioxide in the water, given in mg/L (MS1)
- pH: PH value of the water
- Redox: Redox (oxidation) potential, given in mV
- Leit: Electric conductivity of the water, given in $\mu\text{S}/\text{cm}$
- Trueb: Turbidity (clarity) of the water, given in NTU
- Cl_2: Amount of chlorine dioxide in the water, given in mg/L (MS2)
- Fm: Flow rate at water line 1, given in m^3/h
- Fm_2: Flow rate at water line 2, given in m^3/h
- EVENT: Marker if this entry should be considered as a remarkable change resp. event, given in boolean.

Dataset with more detailed information is available [here](#).

EXPLORATORY DATA ANALYSIS

Main points to keep in mind regarding the dataset:

- Less than 1% of data is missing
- On top of that, missing data is never for just one cell but entire row
 - Therefore we dropped those rows
- Dataset is very unbalanced:
 - With only 1 726 occurrences, anomalies count for only 1.2% of the data

EVENT

0 137840

1 1726

	Time	Tp	Cl	pH	Redox	Leit	Trueb	Cl_2	Fm	Fm_2	EVENT
37152	2016-08-29T05:01:00Z	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	False
37153	2016-08-29T05:02:00Z	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	False
37154	2016-08-29T05:03:00Z	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	False
37155	2016-08-29T05:04:00Z	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	False
37156	2016-08-29T05:05:00Z	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	False

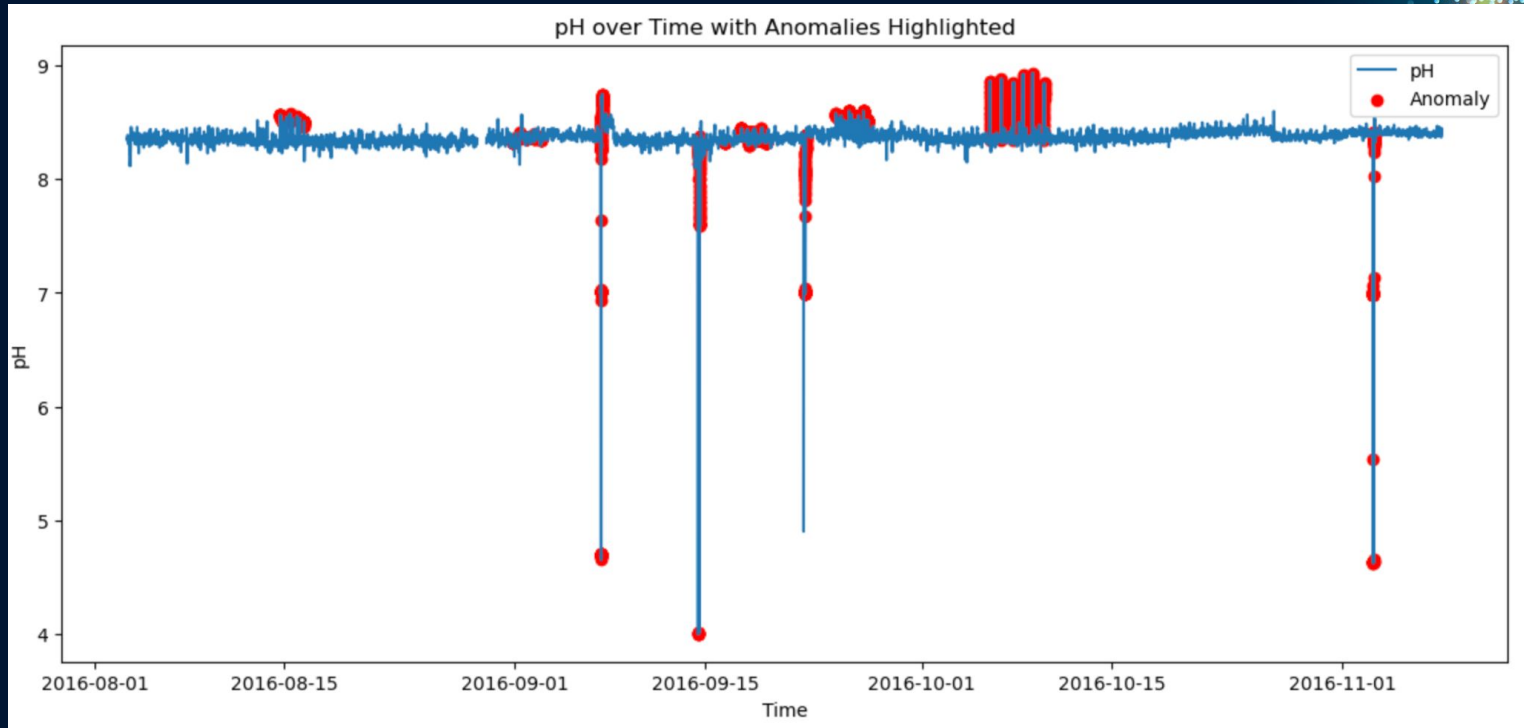
UNIVARIATE ANALYSIS (1/2)

In the context of this project we decided to go for an univariate analysis as recommended.

After making some research and plots (available in EDA notebook) we selected the PH value as our main feature:

- Water should have a PH between 6.5 and 8.5 as per the US Environmental Protection Agency.
- Pure tap water has a PH of 7.5.
- Water with a PH below 6.5 is more likely to be contaminated with pollutants and can be acidic and/or corrosive.
- Finally, some anomalies were observable with a human eye, which makes interpretability easier.

UNIVARIATE ANALYSIS (2/2)





02

Baseline

Can moving average solve our problem?

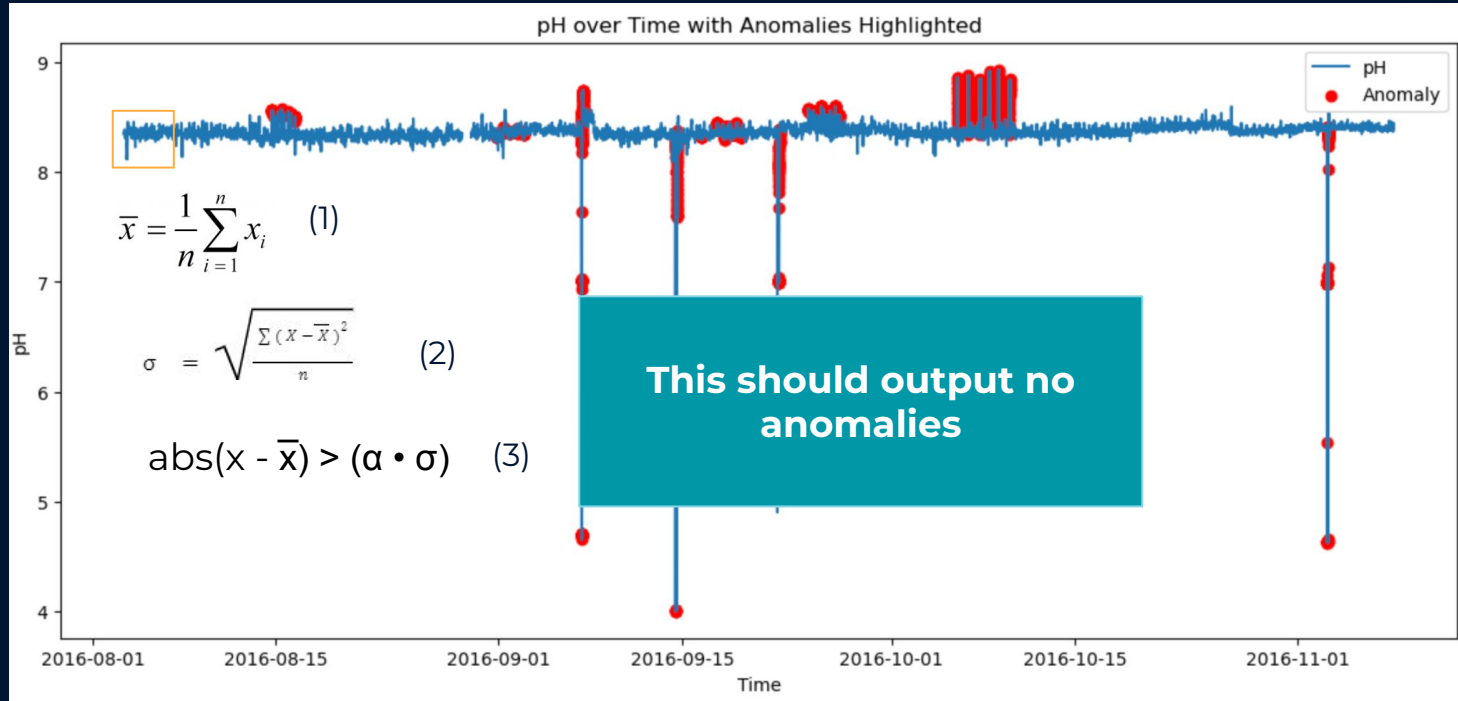
MOVING AVERAGE FUNCTION

As a baseline, we decided to use a simple method to see if that could solve our problem, and if not, why and how to improve.

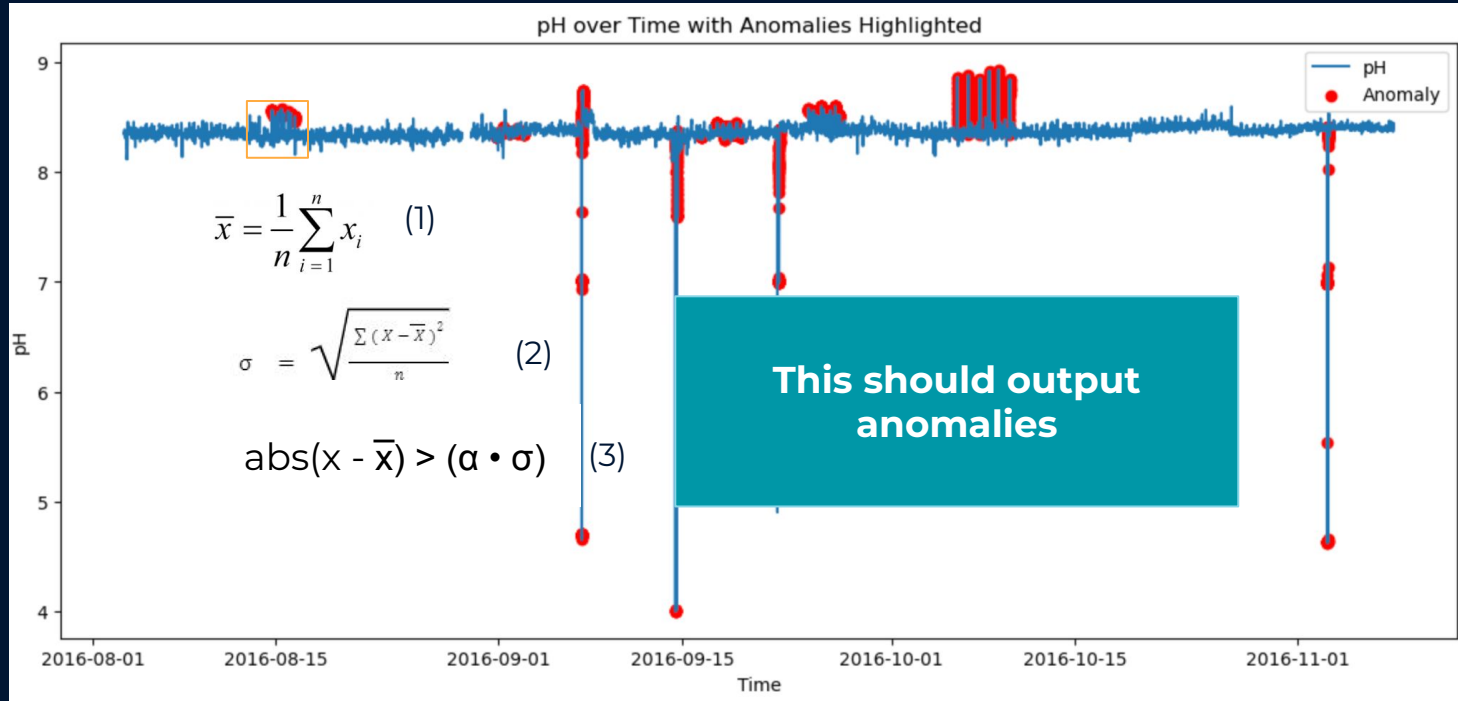
The idea is pretty simple: we create a sliding window selecting N data points. Then we compute in this window:

- the mean
- the standard deviation
- the deviation for each points
- If the **deviation** is **greater** than a define **threshold**, the value is labelled as an **anomaly**.

MOVING AVERAGE FUNCTION



MOVING AVERAGE FUNCTION



MOVING AVERAGE: RESULTS

	Window_Size	Precision	Recall	F1_Score
0	10	0.015658	0.069525	0.025559
1	20	0.017635	0.159328	0.031755
2	30	0.018770	0.220742	0.034599
3	40	0.022074	0.286790	0.040992
4	50	0.026145	0.340093	0.048557
5	60	0.029009	0.368482	0.053784

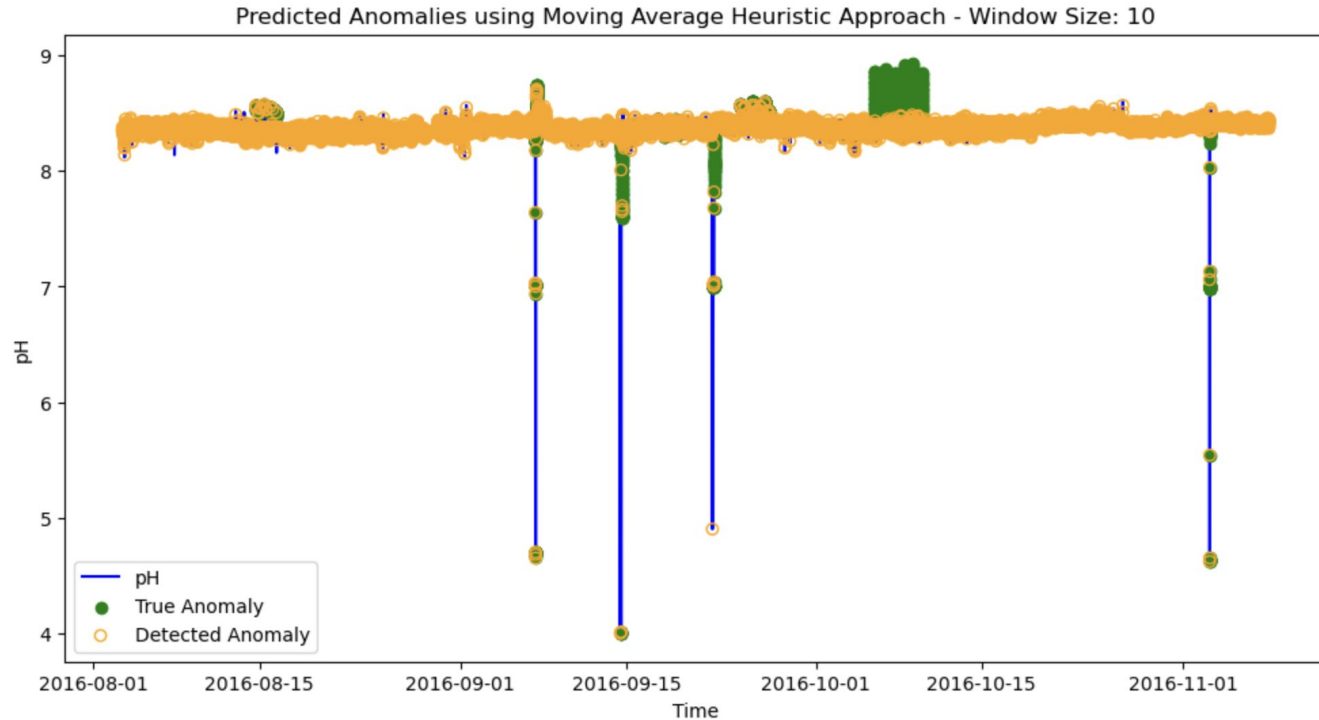
$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

MOVING AVERAGE: RESULTS

Window Size: 10, Precision: 0.015657620041753653





03

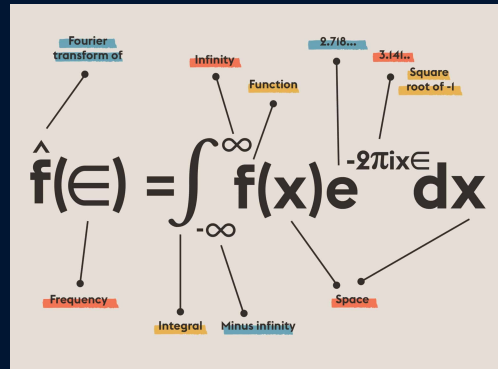
Fourier Transform

Can the father of
Thermodynamics save us?

A GENTLE INTRODUCTION TO DISCRETE FOURIER TRANSFORM

The Discrete Fourier Transform is a mathematical formula that “says that any pattern in space and time can be considered a superposition of sinusoidal patterns with different frequencies”.

This means the component frequencies can then be used to analyze patterns in a signal, extract them and remove noise.

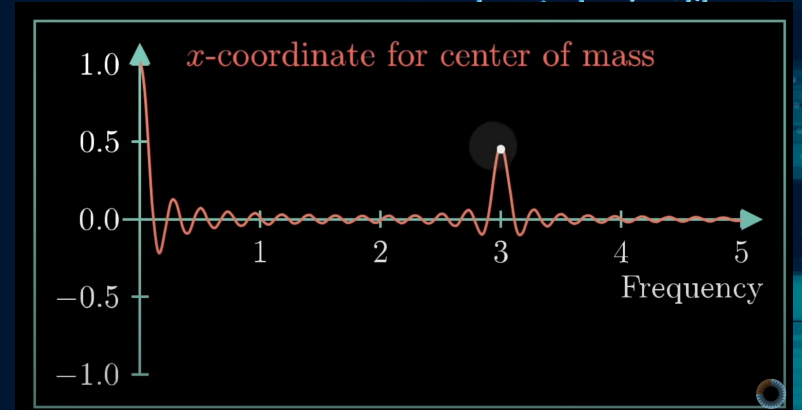
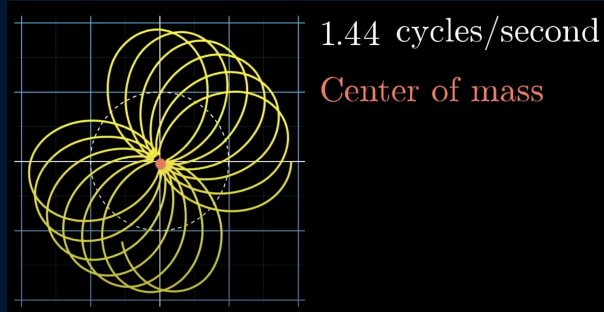
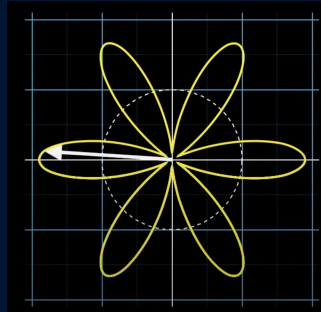
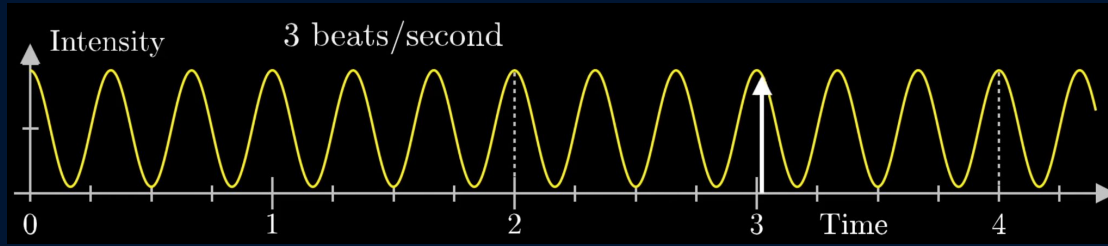


The diagram shows the Fourier Transform formula with various parts annotated:

- Fourier transform of** points to $\hat{f}(\epsilon)$
- Frequency** points to ϵ
- Integral** points to the integral symbol \int
- Minus infinity** points to the lower limit $-\infty$
- Infinity** points to the upper limit ∞
- Function** points to $f(x)$
- Space** points to x
- 2.718...** points to the base of the exponential function e
- 3.1415...** points to π
- Square root of -1** points to i

$$\hat{f}(\epsilon) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \epsilon} dx$$

A GENTLE INTRODUCTION TO DISCRETE FOURIER TRANSFORM



OUR PROPOSED ALGORITHM

Our algorithm is divided in two parts:

1. Fourier Transform:

We retrieve the original values of X , and computes the Discrete Fourier Transform with the `fft` function of `numpy`. Then, our **magnitudes** are just the `abs(fft(X))`.

After that, we order the Fourier values in ascending order and keep only the last **top k samples**, rest being cast as 0.

2. Fourier Prediction:

We compute the **restructured signal \hat{X}** using `ifft()` function from `numpy`. After that, we compute the **deviation** between X and \hat{X} and compare it to a threshold to detect anomalies

```
from numpy.fft import fft
from numpy.fft import ifft

def fourier_transform(df: pd.DataFrame, serie_name: str, k_sample=10):
    k_sample *= -1
    x_values = df[serie_name].values
    x_fft = fft(x_values)
    magnitudes = np.abs(x_fft) # computing the amplitude of each frequency
    significant_frequencies = np.argsort(magnitudes)[k_sample:]
    filtered_fft = np.copy(x_fft)
    filtered_fft[~np.isin(np.arange(len(filtered_fft)), significant_frequencies)] = 0

    return x_values, filtered_fft

def fourier_prediction(df: pd.DataFrame, x_values: pd.Series, filtered_fft: np.array, alpha=2):
    reconstructed_signal = ifft(filtered_fft) # inverse of fourier transform
    abs_deviation = np.abs(x_values - reconstructed_signal) # computing absolute deviation from original signal. If 0 no deviation.
    mean_deviation = np.mean(abs_deviation)
    std_deviation = np.std(abs_deviation)

    threshold = mean_deviation + alpha * std_deviation
    anomalies = np.abs(x_values - reconstructed_signal) > threshold # if our fourier values are above the defined threshold, they are anomalies
    df['FFT_Anomaly'] = anomalies

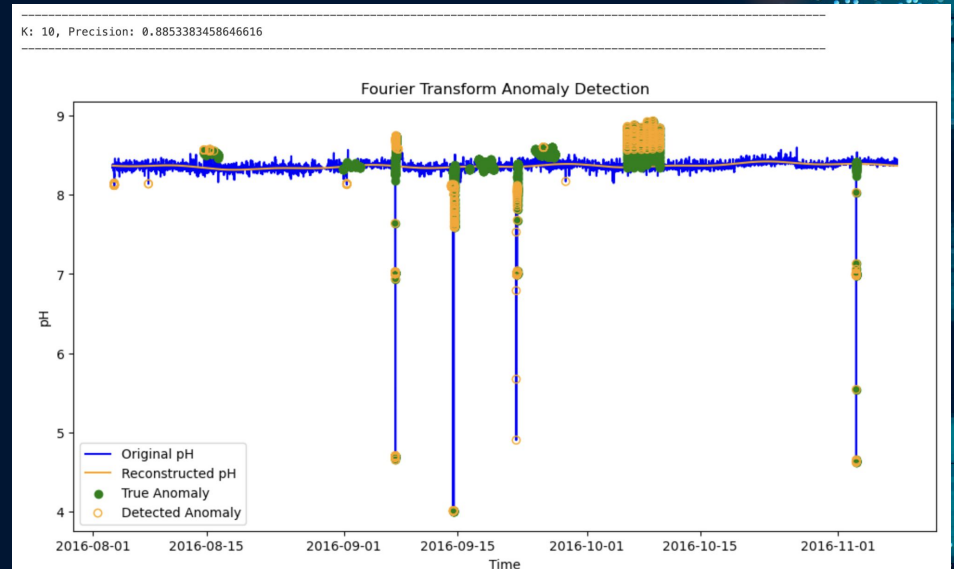
    return reconstructed_signal, anomalies, df
```


FOURIER TRANSFORM : RESULTS

A choice must be made between the number of samples k and the alpha value to balance the amount of False Negatives. To date we haven't find any optimal solution. Some ideas to explore would be: (i) change threshold function (median, quantiles, ...) or (ii) create a sliding window Fourier function.

	k_value	Precision	Recall	F1_Score
0	10	0.885338	0.272885	0.417183
1	20	0.932584	0.288528	0.440708
2	30	0.948052	0.296060	0.451214

	alpha_value	Precision	Recall	F1_Score
0	1.6	0.667665	0.387601	0.490469
1	1.8	0.789683	0.345886	0.481064
2	2.0	0.885338	0.272885	0.417183
3	2.2	0.942350	0.246234	0.390446
4	2.4	0.990453	0.240440	0.386946





04

Advanced ML Models

VAR + LSTM

Vector AutoRegression Brief Recap

Why VAR?

- 1 - Multivariate nature of data
- 2 - Interdependencies among variables

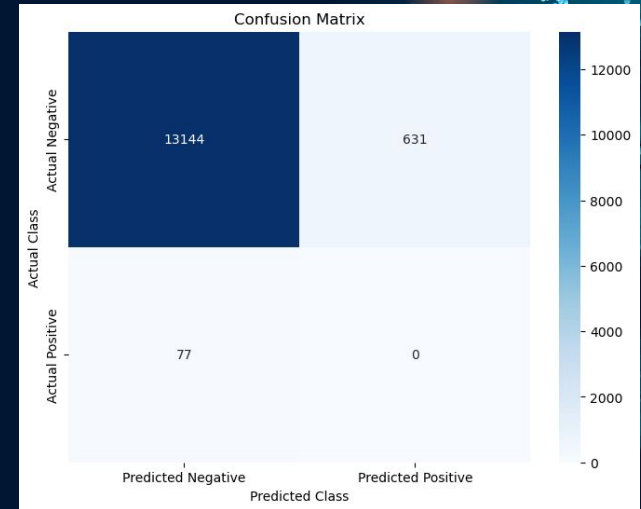
VAR : RESULTS (1) and Model Error Analysis

Accuracy (94.89%): High but misleading due to class imbalance, primarily reflecting the model's ability to recognize the majority class of non-anomalous instances.

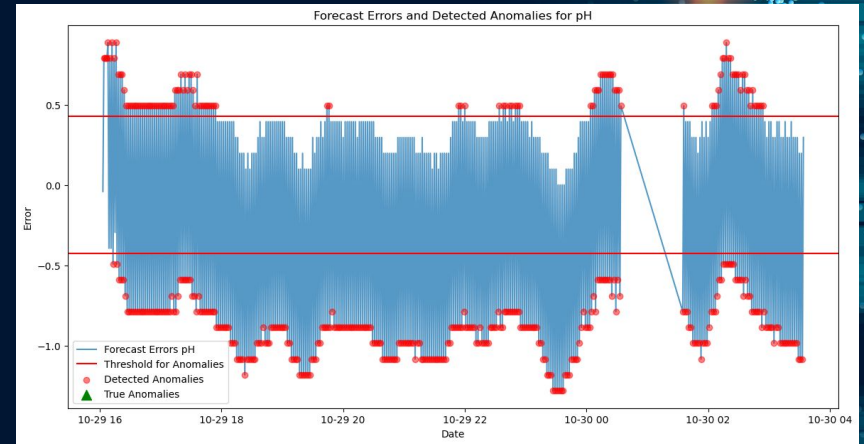
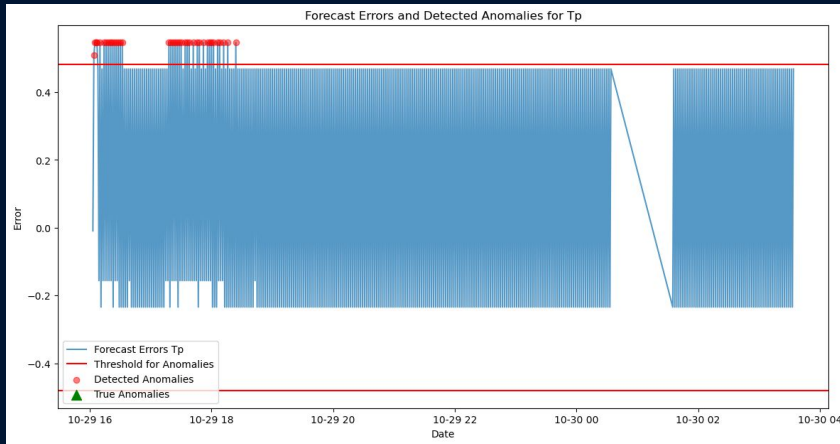
Precision (0.0%): Indicates a total lack of true positive anomaly detections, questioning the model's utility in predicting actual anomalies.

Recall (0.0%): Demonstrates the model's inability to identify any true anomalies, highlighting a critical failure in its detection capabilities.

F1 Score (0.0%): Reflects poor performance, confirming the model's ineffectiveness in balancing precision and recall, and its overall failure in anomaly detection.



VAR : RESULTS (2) and Model Error Analysis

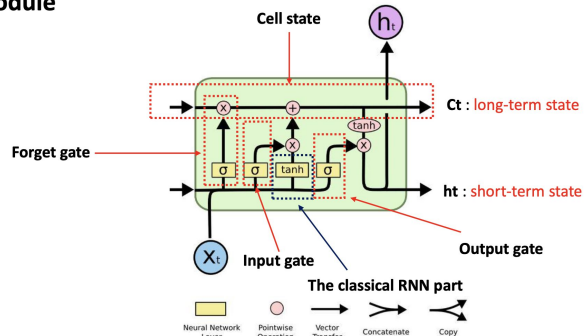


Long-Short Term Memory (LSTM)

Why LSTM?

- RNN are very good at understanding sequences
- LSTM solve the problem of long term memory of RNN
- So if we can build sequences from the time series, it should work

LSTM module



LSTM - METHOD

1

Split Data into sequences

```
def create_sequences(data, column_name, sequence_length):
    sequences = []
    labels = []
    for i in range(len(data) - sequence_length):
        sequences.append(data.iloc[i: i + sequence_length].drop(columns=['column_name']).values)
        labels.append(data.iloc[i + sequence_length]['column_name'])
    return np.array(sequences), np.array(labels)

sequence_length = 10
sequences, labels = create_sequences(data, "EVENT", sequence_length)
```

2

Build LSTM Model

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 10, 50)	11400
dropout_4 (Dropout)	(None, 10, 50)	0
lstm_5 (LSTM)	(None, 50)	20200
dropout_5 (Dropout)	(None, 50)	0
dense_2 (Dense)	(None, 1)	51
Total params: 31651 (123.64 KB)		

LSTM - RESULTS (1/2)

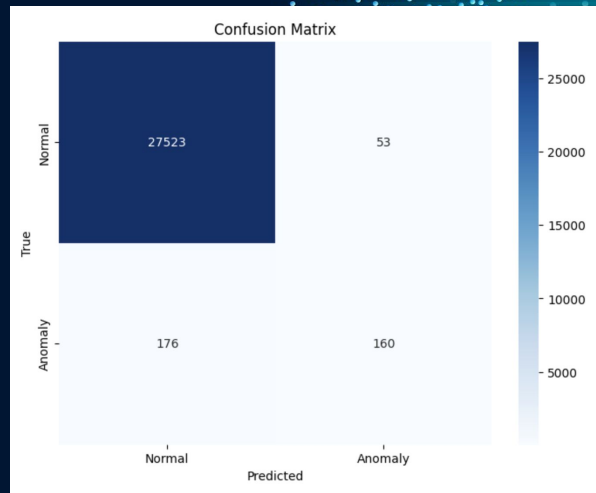
LSTM is our best performing model in terms of Recall and F1-score.

The model is essentially making mistakes at detecting anomalies, but almost never confuse anomalies with normal data.

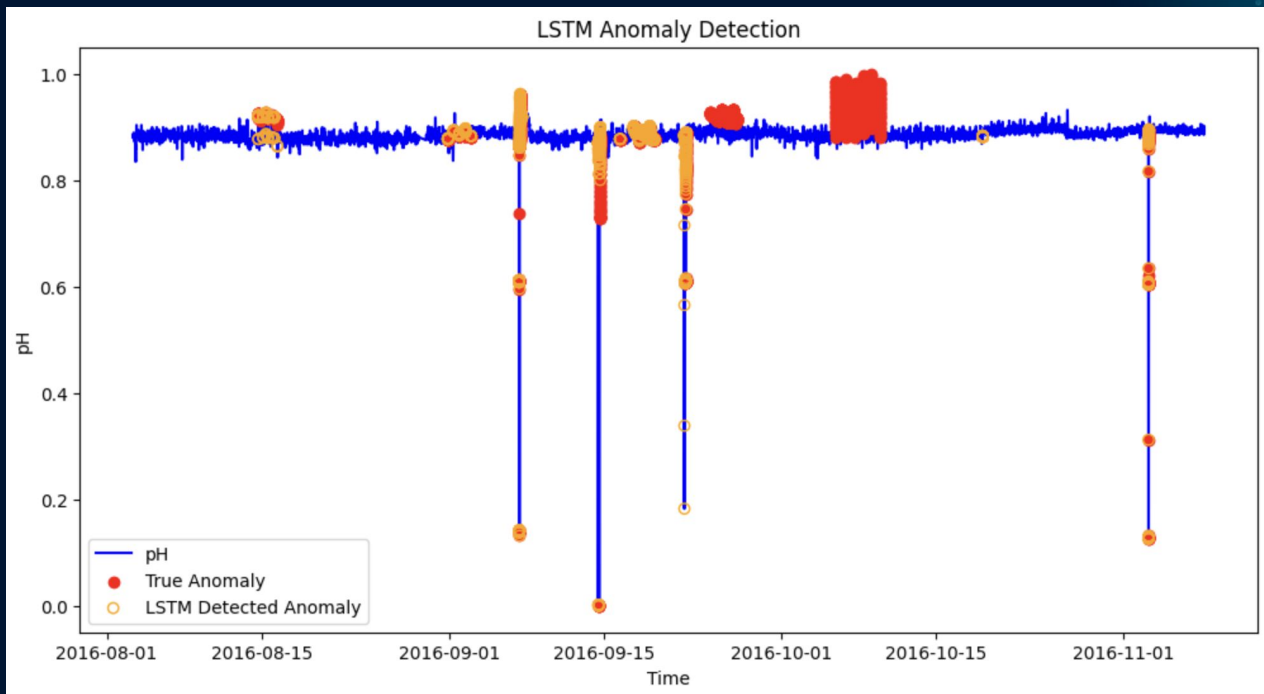
More tests would be needed to improve the model:

- Sequence Length
- Layers size and numbers
- Data Preprocessing: Scaling? Augmentation? Feature Space?
- Autoencoder?

Precision: 0.7512
Recall: 0.4762
F1-Score: 0.5829

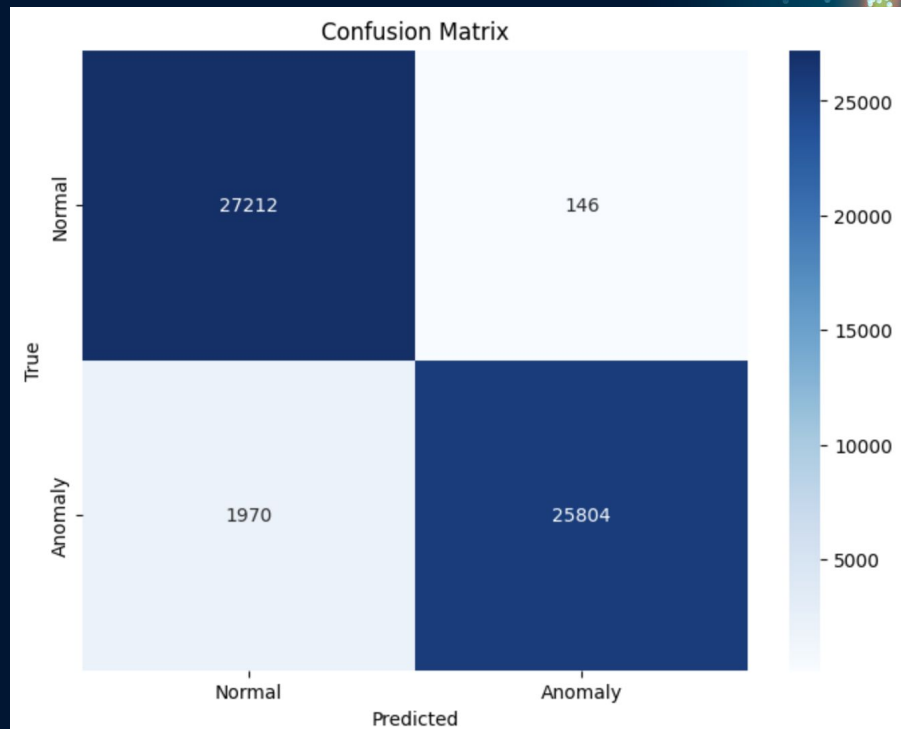


LSTM - RESULTS (2/2)



ACTUALLY...

Precision: 0.9944
Recall: 0.9291
F1-Score: 0.9606

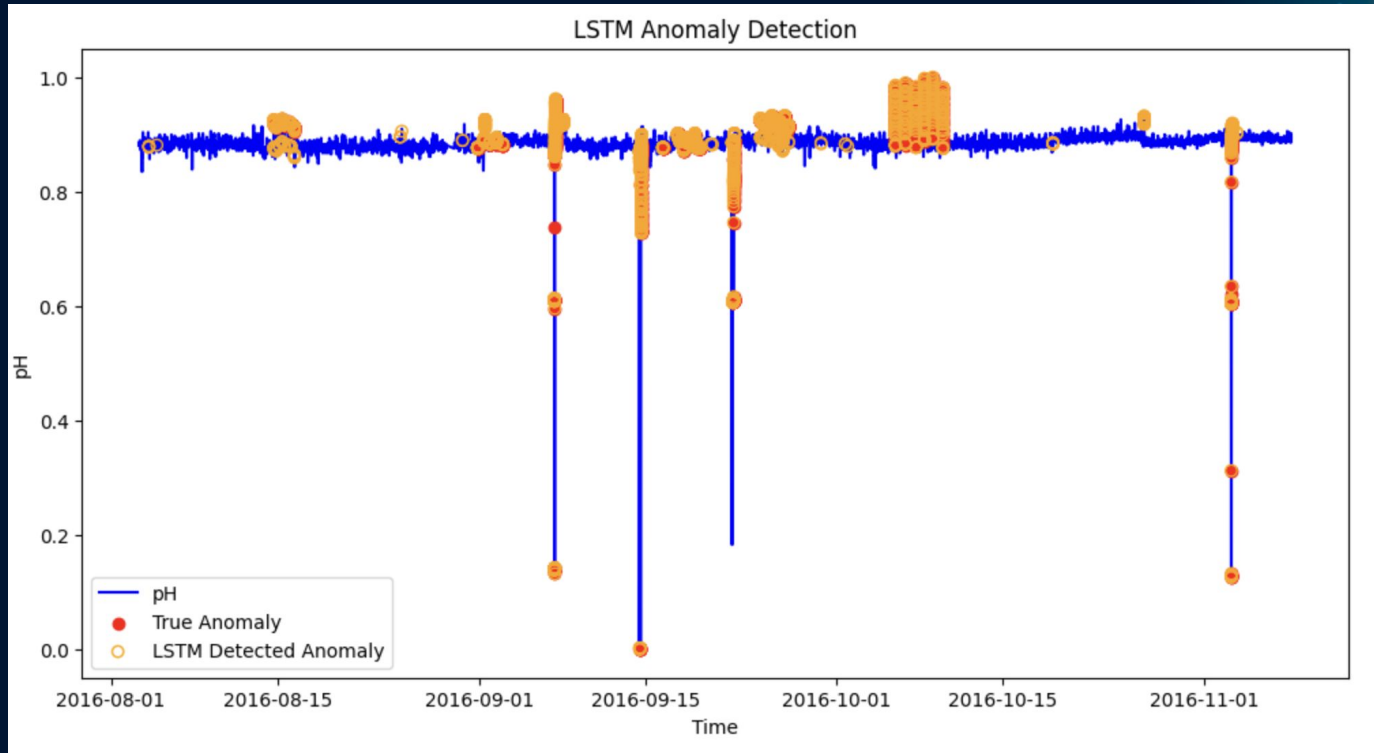


HOW?

We combined LSTM with SMOTE:

- SMOTE stands for Synthetic Minority Oversampling Technique
- Works by selecting examples that are close in the feature space
- Then it draws a line and “create” a new sample in this feature space
 - 1 - Randomly pick a sample from minority
 - 2 - Find k nearest neighbors (5 by default)
 - 3 - Create sample from these neighbors
 - 4 - Repeat
- Code from [imbalance-learn.org](https://imbalanced-learn.org) (sklearn) is available [here](#).

FINAL RESULTS!





Thank You

Any questions?

Bibliography

Medium, [Binary Classification Metric](#), Harikrishnan N B, 2019 ,

Medium, [Time Series Anomaly Detection With LSTM AutoEncoder](#), Max Melichov, 2022

Medium, [Anomaly Detection in Time Series Data using LSTM Autoencoders](#), Zhong Hong, 2024

Medium, [Unveiling Anomalies: Harnessing Fourier Transformation in Machine Learning for Effective Detection](#), Everton Gomedé, 2023

Machine Learning +, [Vector Autoregression \(VAR\) – Comprehensive Guide with Examples in Python](#), Selva Prabhakaran

Wikipedia, [Vector autoregression](#)

github.io, [Vector autoregression models](#), Kevin Kotzé

Machine Learning Mastery, [SMOTE for Imbalanced Classification with Python](#), Jason Brownlee, 2021