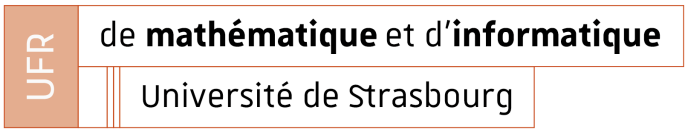


Detecting Patterns in a Time Series

Ecker Nick

21st August 2023

Supervised by: Cedric Schockaert

Contents

1 Introduction

In the realm of industry and manufacturing, steel stands as a fundamental material for constructing structures and machinery. This report delves into the power of Machine Learning within the steel sector.

Central to my internship experience was a focus on "Pattern Detection". This involves detecting shapes or behaviors that hold importance in the steel industry. The primary goal was to assess a platform that employs Machine Learning techniques, with a specific emphasis on its pattern detection capabilities. Additionally, I conducted extensive tests to compare different methods of pattern detection and identify the most effective approach. All of these evaluations were carried out using a specialized dataset provided by Paul Wurth, where I conducted my internship.

Steel is a cornerstone in building various structures and machines. Through the application of Machine Learning, we can enhance efficiency, reduce waste, and optimize resource utilization in the steel sector.

This report shares my internship experience and identifies the most suitable method for pattern detection.

2 Context

2.1 Paul Wurth

The base of my internship experience is connected to my time at Paul Wurth, a big and forward-thinking company in the steel industry. They were founded in 1870 and are really well-known for making smart solutions for steel. They've been around for a long time and have come up with new ways to do steel-making.

Paul Wurth is known for helping to make steel in a really good way. They have lots of different things they do, like making new technologies and helping with engineering. They're experts in areas like building furnaces, taking care of the environment, and making steel in a way that's good for the planet.

Overall, being part of Paul Wurth during my internship let me learn a lot about the steel industry and how they do things in a smart and modern way, like the combination of steel production and machine learning.

2.2 AIXPert

More precisely, I used and evaluated the program AIXPert, a software developed by the firm Datathings, that enables to perform a variety of supervised machine learning algorithms. Therefore, you only need to choose the kind of pipeline/model you want to use and provide the program with a dataset. From there on, the program basically runs itself. However, there are a lot of options and parameters that you can use and alternate to make the algorithm suit your specific desires. These are the different types of pipelines/ML models that you are able to create:

- Regression
- Classification
- Pattern Detection
- Auto-Encoder (yet to be implemented)

3 Basic Concepts of Machine Learning

The term "machine learning" was coined in the mid-20th century by a significant individual: Arthur Lee Samuel, an engineer at IBM. From an early stage, Samuel was a believer in the concept of machines or computers learning. This belief led him to successfully teach a computer how to play checkers. He achieved this using a technique called a "decision tree." Due to limited computational resources, Samuel employed a method known as "alpha-beta pruning." Instead of exhaustively exploring each possible move, he created a scoring function based on the board's position at a given moment. The program's move selection was driven by a minimax strategy. This means the program chose a move that optimized the value of the scoring function, considering that the opponent was also trying to maximize the value of the same function from their perspective.

Generally speaking, it can be said that machine learning is based on many mathematical and statistical processes that are used to be recorded and reproduced by computers. Several definitions also refer to the notion of independent or autonomous learning.

In the next section, we will introduce some applications of machine learning.

3.1 Applications of Machine Learning

First of, it is essential to note that science differentiates between two main categories in machine learning: supervised and unsupervised learning.

- Supervised Learning:

  Supervised learning is a machine learning paradigm where the algorithm learns from labeled training data. In this approach, the input data is paired with corresponding target outputs, allowing the algorithm to learn a mapping between inputs and outputs. The goal is to generalize from the training data to make accurate predictions or classifications on new, unseen data. Supervised learning encompasses tasks like regression, where the algorithm predicts a continuous value, and classification, where it assigns inputs to predefined categories based on patterns learned from the training data.

- Unsupervised Learning:

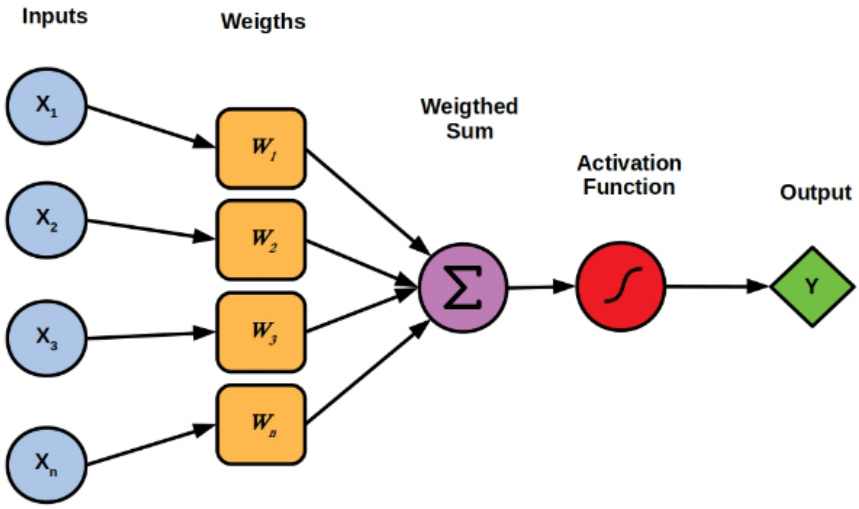
  Unsupervised learning is a machine learning approach where the algorithm learns patterns and structures in the data without explicit labels or target outputs. It involves exploring the inherent relationships and groupings within the data. Unlike supervised learning, there are no predefined correct answers to guide the learning process. Common tasks within unsupervised learning include clustering, where the algorithm groups similar data points together, and dimensionality reduction, where the algorithm simplifies the data while retaining its key characteristics. Unsupervised learning techniques are often used for data exploration, pattern discovery, and gaining insights into the underlying structure of the data.

4 Neural Networks

After having defined the term Machine Learning, we would now like to introduce one of the key technologies for methods based on Deep Learning, namely Neural Networks. Also often called Artificial Neural Networks, the word "neural" refers to their architecture, which can be compared to how neurons in the human brain signal each other. Before diving deeper into a use case of Neural Networks, we are first going to explain their architecture and how these networks function mathematically.

4.1 Perceptron

The term perceptron was first introduced in 1958 by Frank Rosenblatt and was used to perform binary classification tasks, meaning it generates an output of 0 or 1 for example. In Machine Learning, the perceptron is a machine-based algorithm used for supervised learning of various binary classification tasks.



A perceptron has 4 main parameters: input values, weights, a bias and an activation function. Let's consider a case where we have n input values $x_1, x_2, ..., x_n$ and our output value is y. The output of the perceptron is calculated by calculating the sum of the weighted input values, adding the bias to this sum and then applying the activation function to this value:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

This process is often referred to as "feed-forward".

The output of the neuron, shown in the figure with the integral symbol, has a binary output. For example this output can be 0 or 1, depending if it is greater or smaller than a certain threshold θ. This equation is called the activation or threshold function $f(x)$:

$$f(x) = \begin{cases} 1 & \text{if } x > \theta \\ 0 & \text{else} \end{cases}$$

The process by which a perceptron learns how to determine the right values for its parameters is the following:

1. Assign random weights w_i for each input
2. Provide the perceptron with an input vector X, for which there is a known result Y
3. Let the perceptron calculate the output y by the feed-forward process
4. Compute the error: $\varepsilon = \mathrm{Y} - y$
5. In case of an incorrect result, meaning $y \neq \mathrm{Y}$, adjust all the weights according to the error: $w_i = w_i + \varepsilon * x_i$
6. Repeat all the steps from 2. until you get a correct result

This is the most basic example for a perceptron. However, there are 2 additional parameters that can be used in a perceptron to increase its convergence speed and its accuracy.

The first one is the learning rate. The learning rate is a scalar that can influence the accuracy of the weights found at the end and the speed at which they are found. The learning rate would be applied in step 5. when recalculating the weights as

followed:

$$w_i = w_i + \varepsilon * x_i * l$$

with l being the learning rate.

A bigger learning rate will lead to the weights adjusting faster to the right values, however these results won't be as precise at the end as if the learning rate is a smaller value.

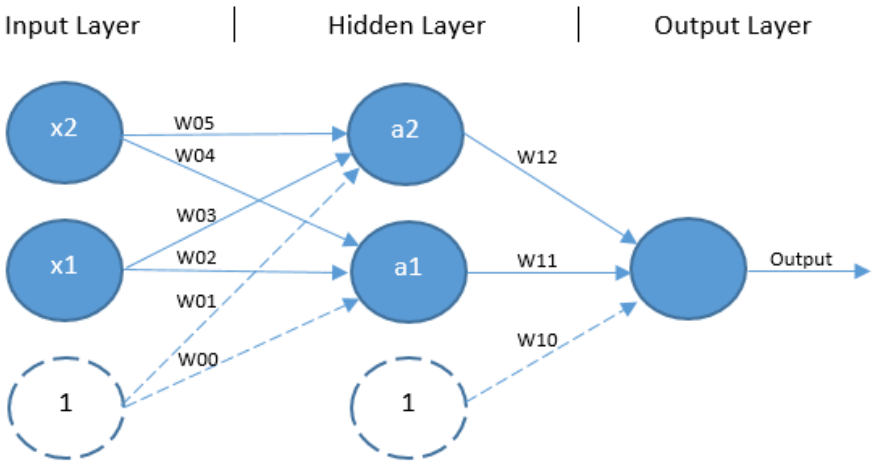
The second parameter is the bias. Let's consider a perceptron with two inputs. If there is an input $(0, 0)$ for example, the weighted sum will always be 0, no matter the values of the weights. Therefore it is useful in some cases to add a bias to the weighted sum. One example of how you could add the bias and correct it over time in a perceptron is to add one input value that is always 1. Over time, the weight for this input value will be adjusted and when you finally get a good result, the bias of the perceptron will be the weight for the input value that is always 1.

Although the perceptron is quite impressive in its simplicity and was groundbreaking when it was first introduced, it is also quite limited in the number of problems it can solve. This is because perceptrons only work with linearly separable cases. One of the simplest and most-known problems that a perceptron isn't able to solve is the XOR problem. However, by adding one or more layers to the architecture, the network is able to solve a much bigger number of problems.

4.2 Multilayer Perceptron and Feed-Forward Neural Networks

When adding one or more hidden layers to a basic perceptron, we are talking about a multilayer perceptron. In contrast to a simple perceptron, they are able to solve more complex problems that are not linearly seperable.

Feedforward is the process in a neural network in which the output is calculated from th input(s). To describe more in detail what happens in the feedforward process, let's consider a pretty simple multilayer perceptron consisting of 2 input nodes, 1 hidden layer with 2 nodes and a single output node.



In this figure, the third nodes in the input and hidden layer represent the bias added at each step.

In this example, the feedforward process consists of 2 major steps: calculate the values of the hidden layer and then calculate the output.

Let's start with describing in detail the process to calculate the hidden values:

1. Like for a simple perceptron, begin by assigning random weights to each connection between an input and hidden node.

2. We form a matrix using the weights:

$$\begin{bmatrix} w_{05} & w_{03} & w_{01} \\ w_{04} & w_{02} & w_{00} \end{bmatrix}$$

   The column i corresponds to the weights of the input node/parameter i.

   The row j corresponds to the weights of the hidden layer node j.

3. Define the input and output (here the hidden) vector: let's say that I is the vector $\begin{bmatrix} x2 \\ x1 \\ 1 \end{bmatrix}$ consisting of the input values and H is the vector $\begin{bmatrix} a2 \\ a1 \end{bmatrix}$ that we want to calculate in this step.

4. Calculate the hidden values: If W is the matrix created earlier and f the activation function we want to apply, the output values are calculated as followed:

$$H = f(W * I)$$

The process to calculate the output node is basically the same as calculating the hidden nodes. We just have to replace the input vector with the hidden vector and the matrix consisting of the weights has to be rebuild using the weights between the hidden and output layer. In this case, the output layer consists of 1 single node. This means that the matrix with the weights has only 1 row, so it will be a simple vector. If there was more than 1 output node, then again we would have a matrix instead of a vector for the weights.

4.3 Gradient Descent and Backpropagation

After the feedforward process, the next essential step is backpropagation. This is the key algorithm used to adjust the weights (and biases) of the connections of a neural network. The goal of backpropagation is to minimize the difference between the predicted outputs and the actual target outputs.

When performing backpropagation in a neural network, you update the weights layer by layer, starting with the last layer (the hidden layer connected to the outputs) and tracking back all the way to the first layer (the input nodes connected to the first hidden layer).

5 Pattern Detection

The term "Pattern Detection" refers to the process of examining a dataset and selecting the structures or trends in the dataset having a certain pattern. Pattern detection is used in a wide range of fields, including face recognition, speech recognition and many other branches. It is important to not confuse pattern detection with pattern recognition. While pattern detection concentrates on identifying one predefined pattern in a dataset, pattern recognition consists of placing a single pattern in one of several predefined classes.

In our case, where we concentrated specifically on detecting a pattern in a time series, I implemented a rolling window that compares the pattern we desire to detect with every possible window. There are several methods to compare two signals, and the

ones being used during the research will be presented in the next sections.

5.1 Euclidean Distance

The simplest approach to compare two signals is probably by the euclidean distance. For this method, the two signals have to be of the same length and the error between the two signals is calculated as followed:

$$\epsilon = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

for $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ being the two signals that are being compared.

5.2 Symbolic Aggregate Approximation

Symbolic Aggregate Approximation (SAX) is a technique mainly used for time series data compression and pattern recognition. It transforms a continuous time series into a symbolic representation, which reduces the dimensionality of the data while preserving important features. SAX is particularly useful for efficiently processing and analyzing large datasets of time series information.

When comparing two signals using SAX, the first step consists in converting the two signals into their respective symbolic representations. This conversion can be performed in 2 steps:

1. Normalization:

   The values within each segment are normalized to have a mean of zero and a standard deviation of one.

2. Mapping to Symbols:

   The normalized values are then mapped to a predefined set of symbols. These symbols are typically represented as letters from the alphabet. The mapping is based on thresholds that divide the range of the normalized values into equal intervals, with each interval assigned a corresponding symbol.

After having applied the SAX transformation to the two signals, we can compare them using a similarity measure of our choice. For example, the Hamming distance between two symbol sequences counts the number of positions where the symbols differ. A lower Hamming distance indicates a greater similarity between the signals.

Another possibility would be the euclidean distance. In this case, each symbol is treated as a point in a space, and the distance is calculated as the straight-line distance between points.

5.3 Dynamic Time Warping

The main focus on comparing two patterns with each other was lying on Dynamic Time Warping. DTW is another method to calculate the similarity or the distance between two time series: However, unlike the Euclidian distance and SAX, the main advantage of DTW is that it enables comparing two time series of different lenghts. Therefore we are able to detect a similar pattern to our base pattern, even if it is contracted or expanded, meaning that the signal has the same form, but over a longer or shorter time period than our base pattern.

In general, Dynamic Time Warping calculates the optimal match between two time series. To determine the distance between two time series using DTW, we need to construct a matrix named the "cost matrix". Let's consider the input sequences A and B of length n and m respectively. The algorithm to construct the cost matrix goes as followed:

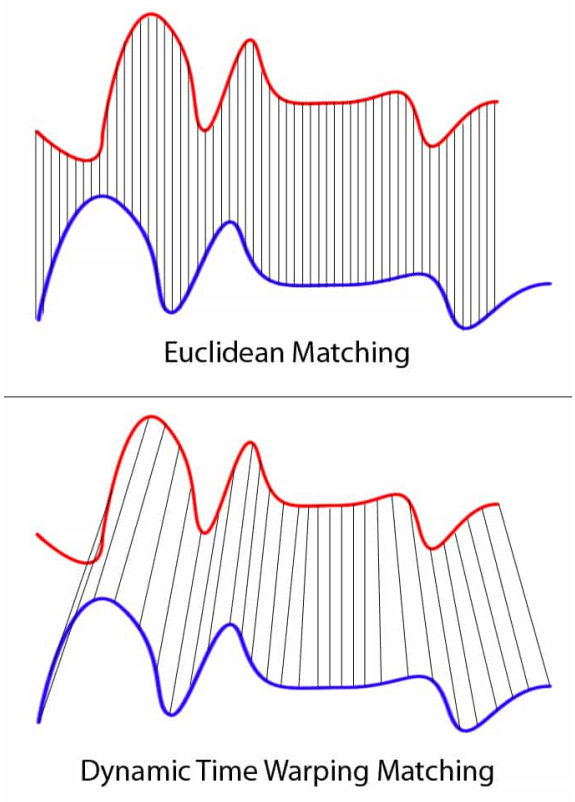
1. We begin by initializing the cost matrix: create a matrix M of dimensions $(n+1)$ x $(m+1)$ and set the values of the first row and first column to $+\infty$. The only value that is initialized differently is $M[1][1]$ which is set to 0.

2. After having initialized M, you can calculate the other values of the cost matrix either row by row or column by column, but always starting at the smallest index of a row/column and going up to the highest. The values are calculated as followed:

For $2 \leq i \leq n+1, 2 \leq j \leq m+1$:

$$M[i][j] = |A[i] - B[j]| + min(M[i-1][j-1], M[i-1][j], M[i][j-1])$$

3. After having assembled the cost matrix, the DTW distance between the sequences A and B is the value of $M[n][m]$.



Note that DTW is a much more complex method than the euclidean distance or SAX to compare two signals. While the complexity of comparing two signals with size n using the euclidean distance is $O(n)$, the complexity when using DTW is $O(n^2)$.

5.4 Local Pattern vs Global Pattern

When considering the similarity between two signals, the assessment can be approached either from a local or a global perspective. When comparing two signals that have a relatively big difference in amplitude globally, the distance between these two signals will necessarily also be pretty big, no matter if you use DTW or the euclidian distance. As a result, instances can occur where a signal with a com-

pletely different shape from our base pattern might get a greater similarity rating than another signal. This other signal could closely mirror the baseline pattern in shape, but display a larger variation in amplitude compared to the first signal.

To evade this problem, the patterns can be compared locally. Therefore I exploited 3 different types of transformations that allow to compare signals locally:

1. Standardization:

   $$z = \frac{x-\mu}{\sigma}$$

   where:

   - x is the value of the signal

   - μ is the mean (average) of the signal values

   - σ is the standard deviation of the signal values

   Standardization is also known as Z-score normalization

2. Normalization:

   $$z = \frac{x-\min(x)}{\max(x)-\min(x)}$$

   with x the value of the signal

3. Adjusting the mean to 0:

   $$z = x - \mu$$

   - x is the value of the signal

   - μ is the mean (average) of the signal values

   This is a relatively naive approach, where the scale of a signal will not change, it's mean will only be adjusted to 0.

6 Data Presentation

6.1 Context

Before going into detail on our dataset and looking at the type of patterns we want to detect, we should first introduce in what context our dataset was generated. Basically, our dataset is generated by monitoring a section mill. A section mill is a specialized facility in the steel industry, whose primary purpose is to transform semi-finished steel products into finished sections with precise dimensions and profiles.

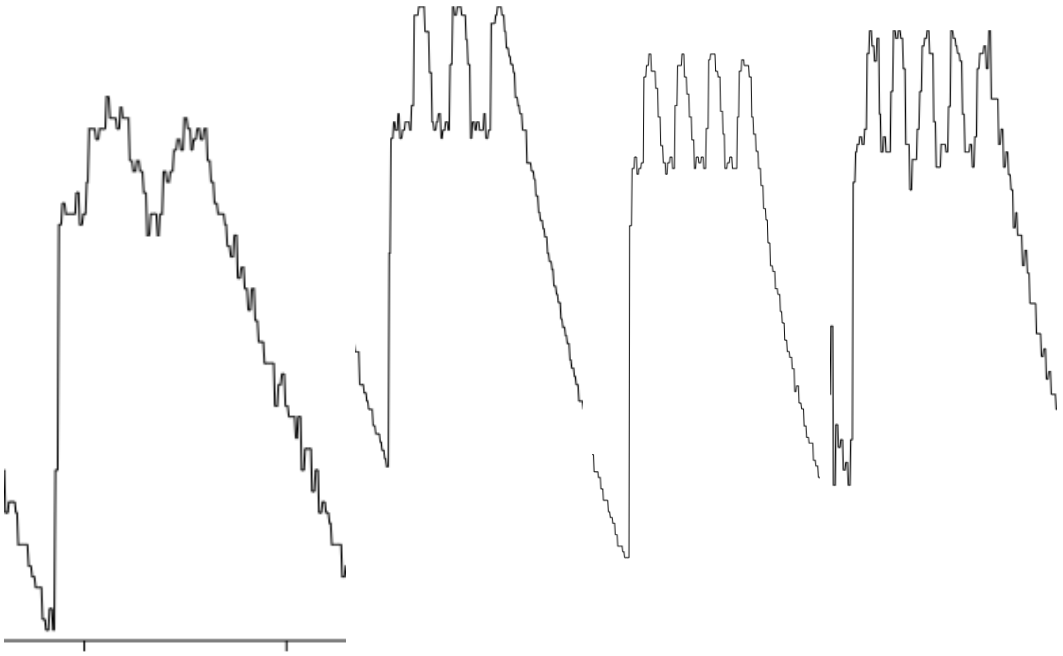
In our case, the data was generated by monitoring 4 idler rolls in a section mill. Idler rolls aren't directly involved in the process of shaping the steel in a section mill, their role is mainly to ensure the proper alignment, stability, and smooth movement of the steel sections during the process.

More precisely, the speed of each idle roll is captured at each time step.

6.2 Characteristics

Because our dataset was generated by monitoring the speed of 4 idler rolls over a specific period of time, we basically have 4 time series that we can work with. Each of these timeseries has over 600.000 points. However, these 4 time series have more or less the same shape, which is logical, because they are installed in the same section mill. Therefore, it is sufficient to restrict ourselves to only one time series to perform our tests on pattern detection.

The patterns we would like to detect in our dataset have peaks with 2 up to 5 picks. Here are the different patterns that can be detected in the time series:



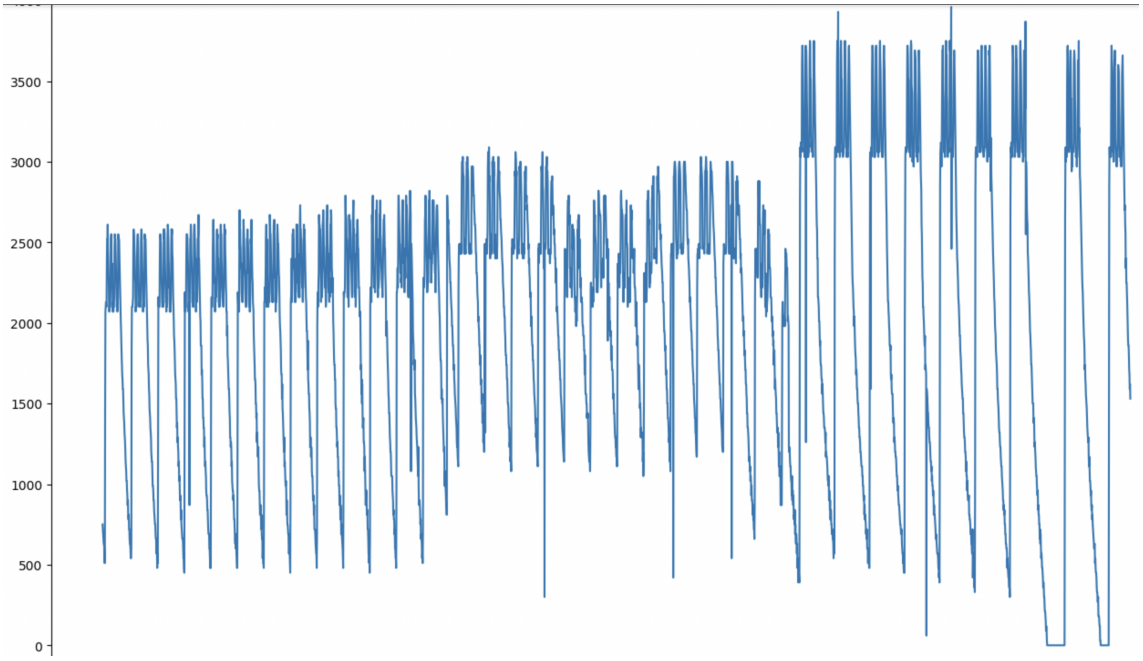
7 Application on the Dataset

In the pursuit of identifying the most effective method for pattern detection, a comprehensive assessment was undertaken to compare the performance of the Euclidean distance and Dynamic Time Warping techniques. This involved a systematic series of tests encompassing global signal comparisons, in addition to signal refinement techniques such as standardization, normalization, and mean adjustment. The experimentation was executed using the Python programming language, wherein dedicated classes were meticulously developed. These classes served the dual purpose of pattern definition and the implementation of a rolling window approach for optimal pattern detection within time series data. This work forms a key component of my internship report, reflecting the practical aspects of the investigative process. However, because of the enormous size of the dataset and the relatively high computation time for DTW, the tests to detect a pattern were always executed on smaller chunks of the time series, of around 4000 points.

7.1 Single Pattern Detection

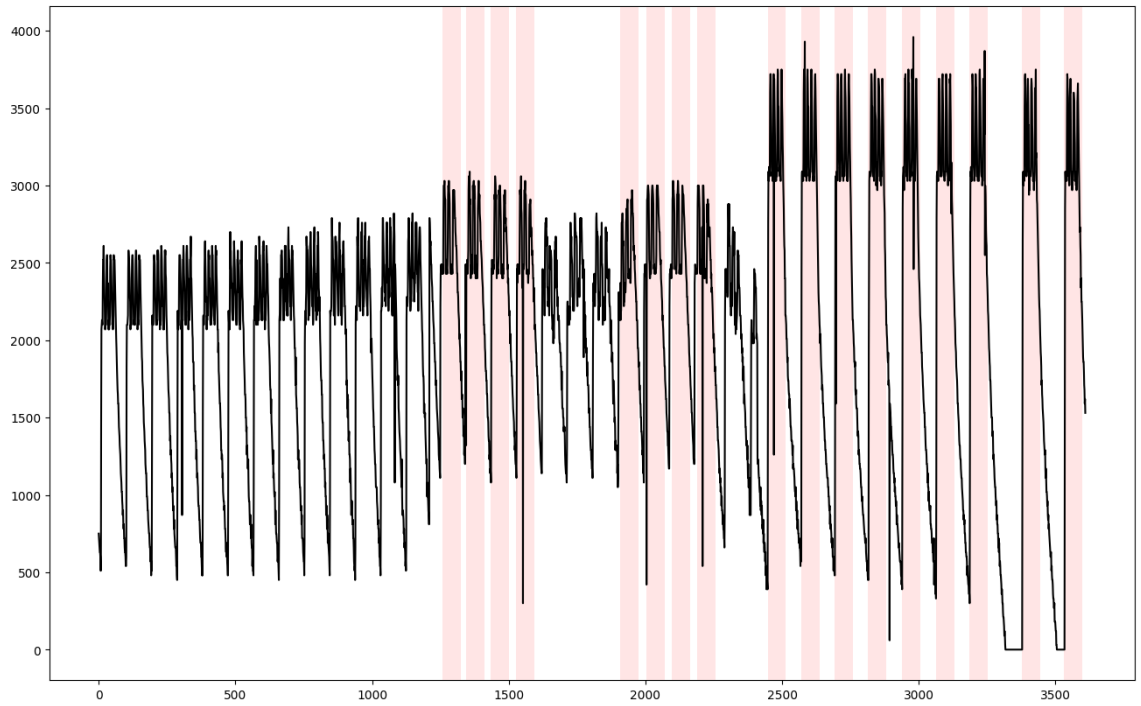
In a first instance, the goal was to evaluate the best method to detect the most occurrences of a single pattern in a time series.

These first tests were conducted on pretty simple timeseries, like this one consisting of only 3- and 4-picks.



The first and last thirds of this time series are 4-picks. The signals in the second third, from the first bigger elevation in amplitude up to the last signal that doesn't have an amplitude over 3500 are all 3-picks.

For this example, we tried to detect the 4-picks patterns in the time series, using one of the 4-picks patterns of the last third as our base pattern. Therfore we compared the signals globally using DTW. In the ideal case, all of the 4-picks patterns would be detected as the best detections and none of the 3-picks patterns. However, in reality, the result is this:

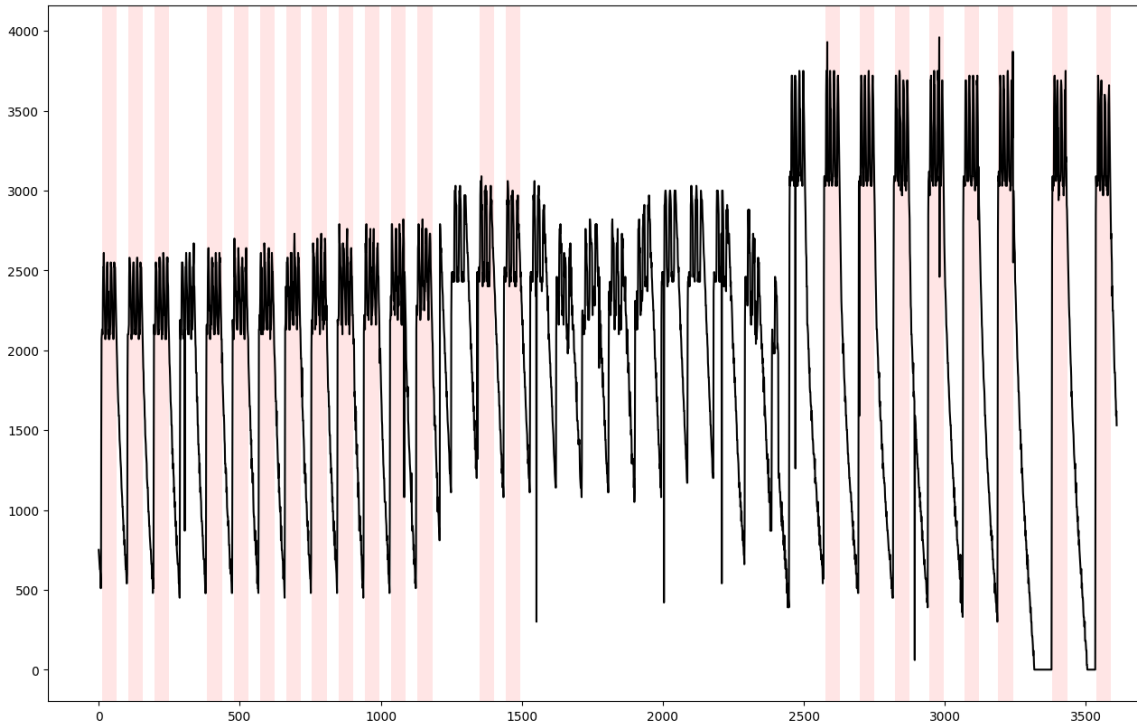


Although all of the 4-picks with a high amplitude have been correctly detected, none of the ones with a lower amplitude have been detected, and additionally there are 8 wrong detections of 3-picks signals.

Using the euclidean distance the result was similar, with a lot of the 3-picks signals ranked as better detections than most of the 4-picks signals with a lower amplitude.

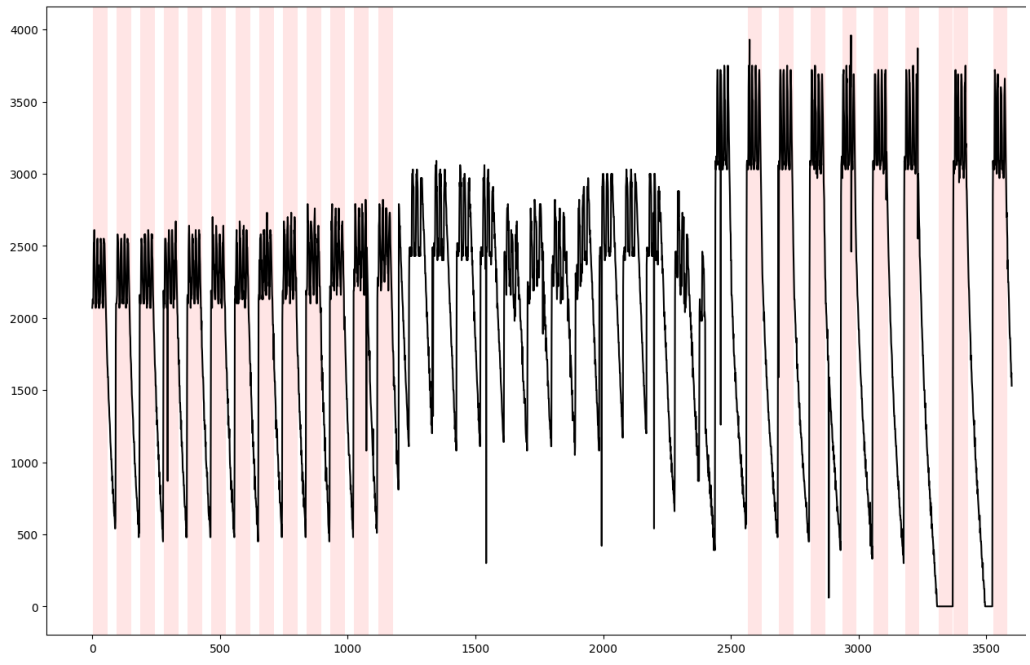
To attack this problem, the next step consisted in comparing the signals locally to our base pattern, using the 3 methods established earlier. Throughout nearly all the tests conducted with the local comparison methods, standardization was pretty much always as good or better than the other 2 methods. Therefore we will take a closer look at the results received when using standardization.

In our example, there are 22 4-picks signals in total, so let's look at the 22 best detections, first when using DTW combined with standardization:



Analyzing the results, it is obvious that the results received when comparing the signals locally are a lot more convenient when comparing them globally. In this case, there are only 2 wrong detections.

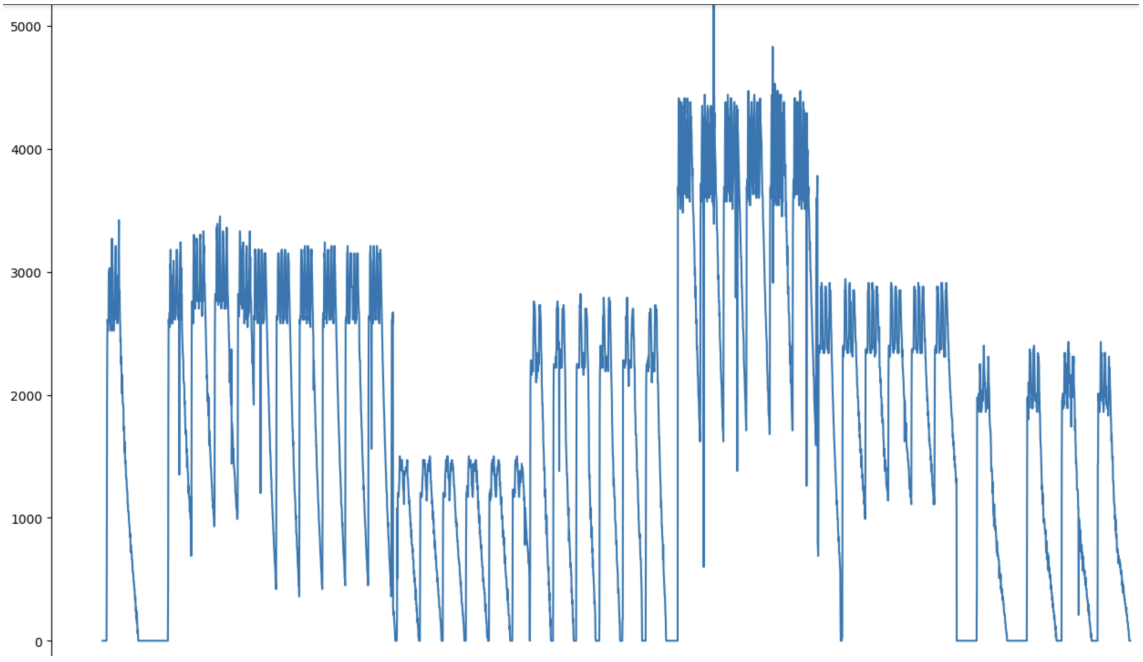
Now let's look at the result when using the euclidean distance with standardization:



With this method, the result is even slightly better, with only 1 wrong detection.

From now on, all of the results that are shown are have been produced when using standardization.

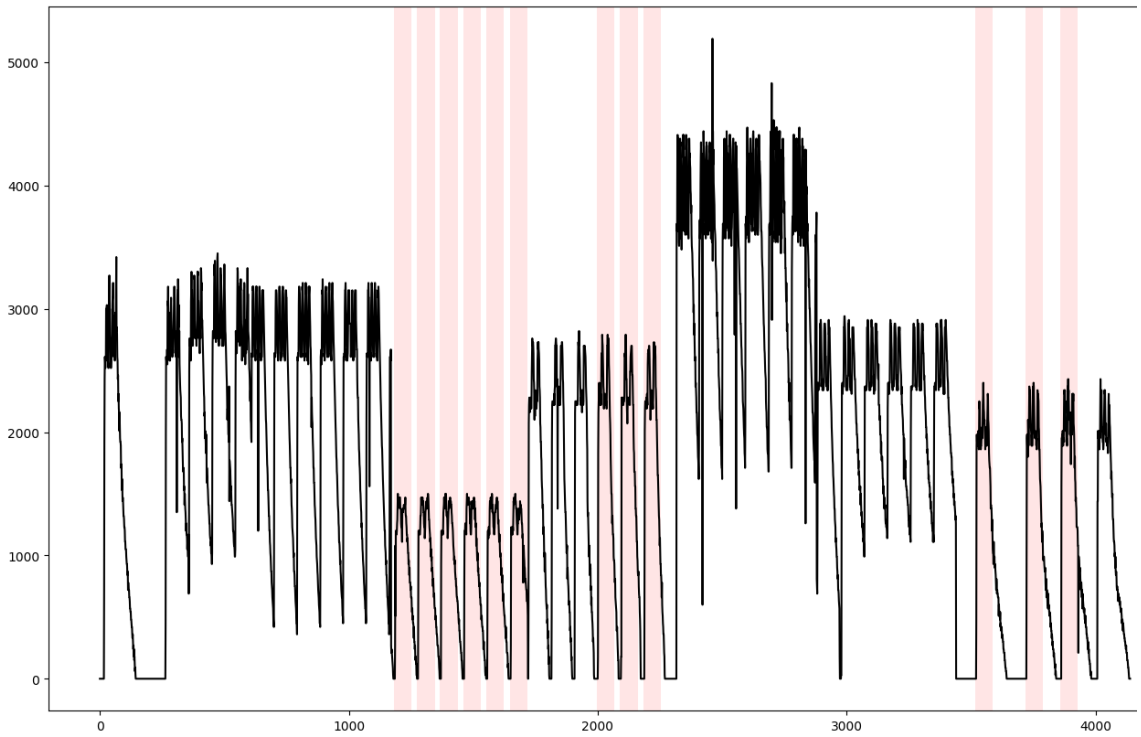
Now we are going to take a look at a more complex time series consisting of signals with 2, 3, 4 and 5 picks.



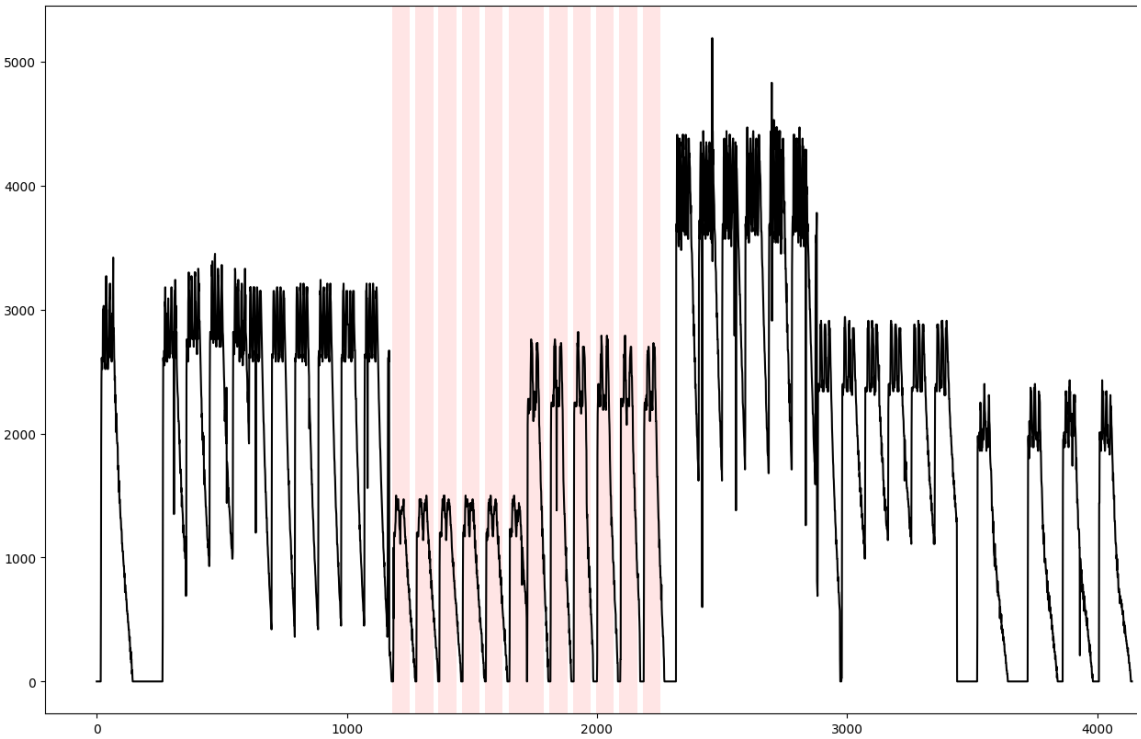
The first block of signals having a relatively even amplitude consists of 4-picks signals. After that the block with a very low amplitude and the one having a slightly higher one are 2-picks. The signals with the very big amplitude are 5-picks and all the signals after these are 3-picks.

This time series will also be used to visualize the results obtained in the next section, but before that let's compare the results for DTW and the euclidean distance when trying to detect the signals having 2 picks.

First when using DTW:



When using the euclidean distance:



As we can see, the euclidean distance produced a perfect result, while the DTW method produced 3 wrong detections.

In pretty much all the tests conducted when comparing DTW and the euclidean distance when using standardization, the results produced by the euclidean distance were as good if not better than the results recieved by DTW. There was a very little number of cases where DTW produced slightly better results, but in general the euclidean distance was superior in making right detections.

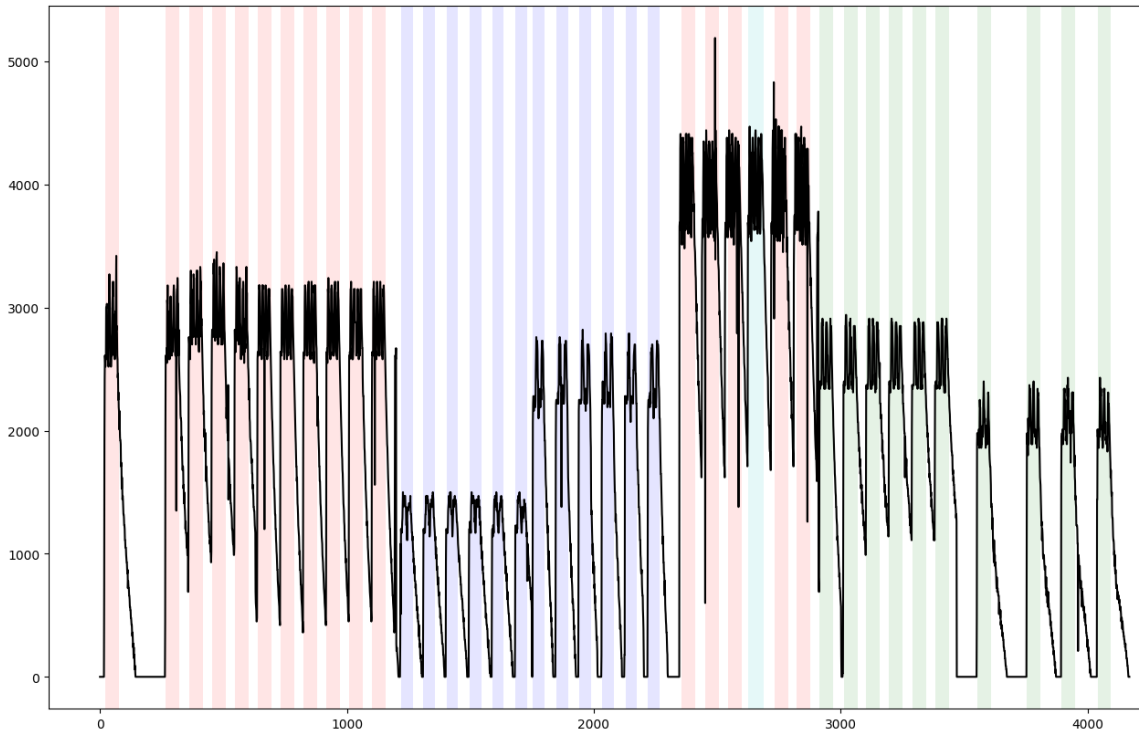
7.2 Multiple Pattern Detection

After having evaluated the methods when trying to detect one single type of pattern, the next step was to make the right classification for each signal when having multiple patterns that we want to detect.

The procedure for this step wasn't that different to the procedure when having a single pattern. Here we perform a rolling window for each base pattern. By doing so we get the distance of each possible signal compared to each pattern, and then the signal is associated to the pattern who has the smallest distance to the signal.

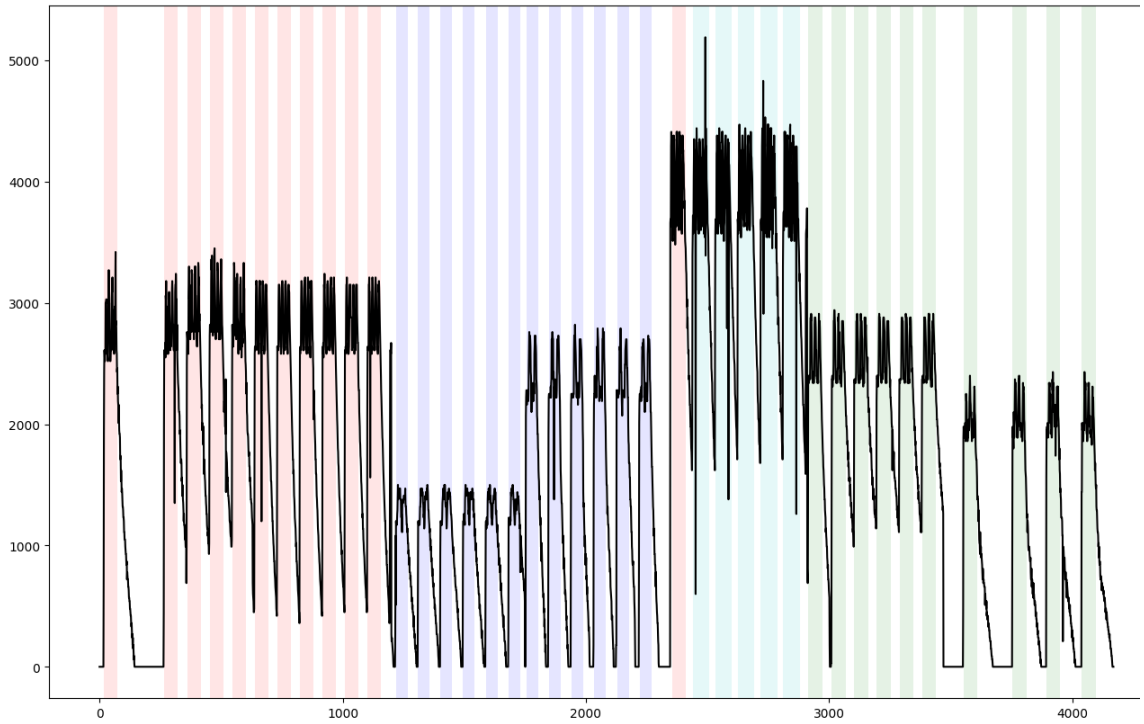
Let's take a look at our time series we already used earlier on, but this time we try to associate all the possible signals to either a 2-, 3-, 4-, or 5-picks signal.

Let's start when using DTW:



Nearly all the signals are associated to the right pattern, apart from the 5-picks, they are all classified as 4-picks (apart from one, which is the base pattern for the 5-picks).

When using the euclidean distance:



With the euclidean distance, there is only 1 error: one of the 5-picks signals has been associated to the 4-picks pattern. However, apart from this error, all the signals have been associated to the correct pattern.

As for the single pattern detection, the tests conducted for the multiple pattern detection lead to the same results. We get the best results when using standardization, and in most cases the euclidean distance is as good as DTW if not even better.

8 Conclusion

After conducting an extensive series of tests on various time series and diverse patterns, a clear and evident conclusion emerges. The integration of standardization into AIXPert is imperative, as it produces the most optimal results while effectively

resolving the current limitation faced by AIXPert. This limitation refers to its inability to detect patterns that exhibit significant amplitude differences, despite their resemblances in shape.

Regarding the incorporation of DTW, additional testing is essential to determine its applicability in specific scenarios. Based solely on the tests conducted during my internship, it appears that DTW might not be universally beneficial. This assessment is drawn from the consistent superior performance of the euclidean distance, which not only produces better results in most instances, but also has a notably shorter computation time. However, it is crucial to recognize that these evaluations were confined to a single dataset. There exists the potential that in alternative time series datasets, where patterns might undergo stretching or compression, DTW could potentially surpass the euclidean distance.