2022 PRACTICUM

CHANGE POINT DETECTION OF TIME SERIES DATA

FOR COMNET MOBILE DATA PING TIME

BY

ALEX WALTON

(GT-ID: 903398136)

OMSA PRACTICUM

# Abstract

The idea behind this thesis is to detect significant change points in ping response time from cell towers around the US. The approach taken needs to be able to deal with significantly noisy data, and be able to distinguish between noise and real underling changes in the time-series data. A range of methods are explored.

The data is from 300+ cell towers, with 4 variables. In this project, identifying that there is a change in mean, and that when there is a delay in ping time, we prioritise voice data over data at all times. The algorithm not only has to be accurate in detecting change points within the time series data, but also has a time constraint component. The data is pulled every 20 minutes from 2 servers, therefore we need this algorithm to be able to scan all the data within the time frame, as well as be able to show results, to allow engineers to highlight issues in reasonable time.

# Table of Contents

[Abstract 1](#_Toc108952888)

[Table of Contents 2](#_Toc108952889)

[1. Introduction 2](#_Toc108952890)

[2. Problem Statement 5](#_Toc108952891)

[1. Methods 5](#_Toc108952892)

[1.1. PELT 5](#_Toc108952893)

[1.2. Bottom up 5](#_Toc108952894)

[1.3. Window 5](#_Toc108952895)

[References 10](#_Toc108952896)

# Introduction

For this project, data from a plethora of mobile sites around the US, each with a timeseries lookback of ping times (in Ms) for 4 variables (EFlarge, EFsmall, BElarge, BEsmall). Each of these variables represents a type of data packet, and the response time of each of these packets of data, for every sample interval, is represented on a time series graph as shown below (Fig1.1)

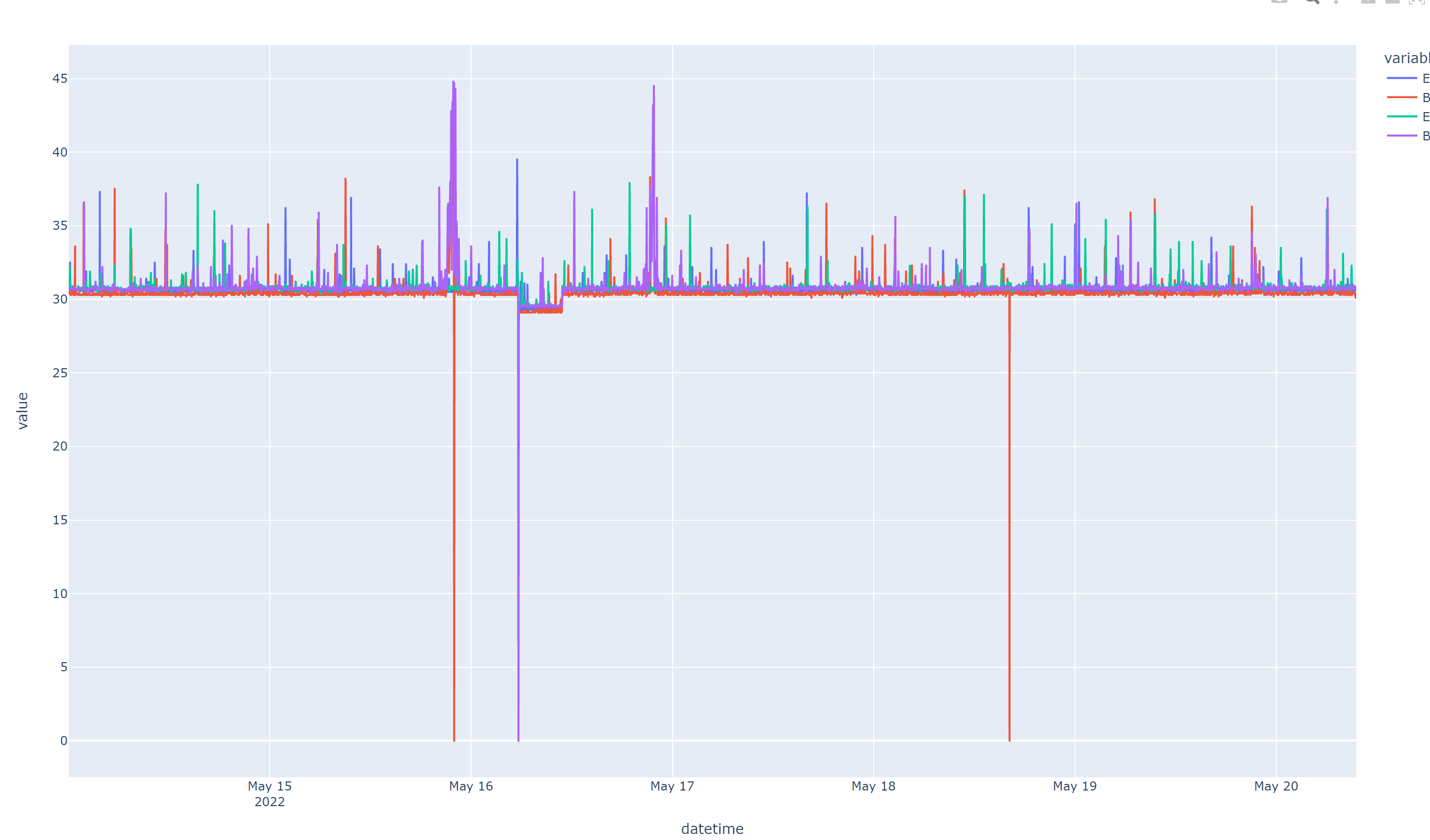


Figure .

The aim of the project is to correctly identify when a statistically significant change in the mean of the time series for each variable occurs. Alongside this, we want to ensure that when there is lag in the time series data, always we need to make sure that voice data is prioritised over data packets. The reasoning is simple: When on a mobile call, we can’t afford data to drop as the voice quality will sounds “computerised” and may not even get through at all. On the other hand, most mobile/internet devices have some form of local buffer, and therefore there is some tolerance if data is buffering/ delayed.

Changepoint Detection is a well-researched area of time series analysis, for example, Truong et al investigate different offline change detection methods for multivariate time series (Charles Truong, 2020). In our particular use case, we can investigate offline changepoint detection methods as we don’t need real time monitoring. The different between offline and online change points is essentially as follows. Offline changepoint detects changes when all samples are received, whereas online changepoint attempts to detect changepoints in real time settings. If we can spot a real and significant change in the time series, as well as highlight to relevant areas in-between data being redownloaded, this is perfectly acceptable.

Now we know that we can use offline change point detection, the next question is how do we approach detecting significant changes in a time-series? There are many approaches to address this question, simple CUSUM can be used to detect changes in data, however in this instance we attempt to reduce the number of false positives that are flagged to the user.

Of note, the data supplied is unlabelled. This provided a challenge in itself, as to how to measure the accuracy of the model without manually analysing all plots to see if algorithm tested was detecting the correct changepoints.

For this project, attempting change point detection in the given data, a number of more advanced detection algorithms were used, and evaluated. Initial attempts included simple change point detections, which were investigated before settling on a much more accurate ensemble of changepoint detection algorithms. This was then set out in an ease of use dash board, displaying the highest ranked changes with the most weight, allowing engineers and members of staff to view potential change points in their time series data. A python framework is also created, allowing the end user to rerun the different cost functions, and on different data. This allows customisable CPDE in the future.

# Problem Statement

There are many difference change point detection algorithms available, via open-source python libraries. Some of these libraries offer a range of different detection methods for changepoint detection in time series data, while others offer similar algorithms wrapped in different libraries. One issue initially having to face was the fact that the data was unlabelled. This meant that either the data could be hand labelled, or a method that didn’t require data to be labelled should be used. For this project, and given the fact that the data was only a small window of the timeseries data generated, I opted for an unlabelled approach, that would then be assessed graphically. This saved time initially in that labelling data was a manual process and was too time consuming, however, it meant that algorithms suitable for unsupervised learning had to be used. For example, one initially looked at was a library called sktime (Alan Turing Institure) Many were tried and tested to assess their usefulness in detecting changepoints in the data highlighted above, until one was settled on and tested further. These will be outline below

# Methods

## PELT

The Auto-detect number of change points (PELT) option uses the Pruned Exact Linear Time (Killick, 2012) algorithm to estimate the number and location of change points. PELT works by adding a penalty term for an inclusion of any additional changepoint. This is the means that the pay off between adding the new change point is weighed against the reduction in segment cost. For each point, if the cost is higher that the reduction in the segment cost, we will not include the segment as a change point Window

## Bottom up

This is described as a “fast signal segmentation” within the common libraries. The algorithm begins by creating the finest possible approximation of the time series, so that n/2 segments are used to approximate the length time series. Next, the cost of merging each pair of adjacent segments is calculated, and the algorithm begins to iteratively merge the lowest cost pair until a stopping criterion is met. (Keogh, 2001) Essentially the algorithm starts with many changepoints, and then iteratively deletes less significant ones. The complexity of order for the bottom-up algorithm is (n lon n)

## Window

The sliding window algorithm is also quick. It uses two windows that “slide” along the array of data. Then, the statistical properties of each window are calculated, and the discrepancies noted. If the discrepancies are significant, then we assume a boundary has been detected. The cost function here can be, for example, l2, which detected change in median and mean respectively

The methods above show the different change point detection algorithms attempted within this project.

# CPDE Approach

The ruptures library was the chosen unsupervised machine learning library used to find the unknown number of changepoints within each multivariate time series. (Ruptures python library, 2022) This was chosen after trials of other similar libraries, as it was much more customisable for the outcome of this project.

Initially, trials of each of the 3 previous highlighted models showed mixed results. Below is one of the first unsupervised methods using bottom-up algorithm (highlighted in 1.4) with a penalty value calculated as

) (https://pro.arcgis.com, 2022)

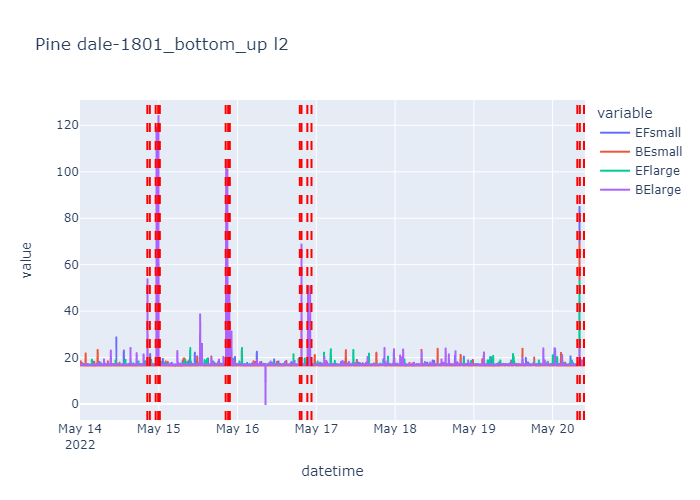


Figure .

You can see from fig 2.1 that we are detecting changes in the timer series (albeit many more than possibly required). However, this particular CPD doesn’t seem to deal well with highly noisy, and non-substantial changes in mean. As fig 2.2 shows below, this is a noisy dataset and there is no real change in mean for any parameter. This would clearly show too many false positives.

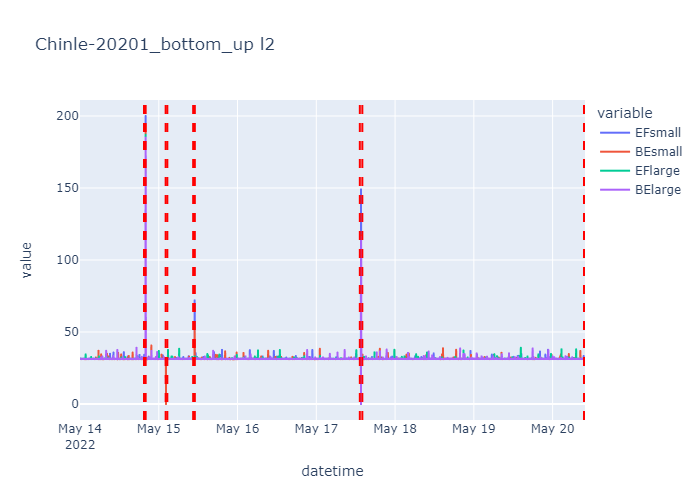


Figure .

Looking at another changepoint algorithm, we can see if we can use a different method (2.3) with a different cost function to see if we can prevent too much noise impacting the change detection algorithm.

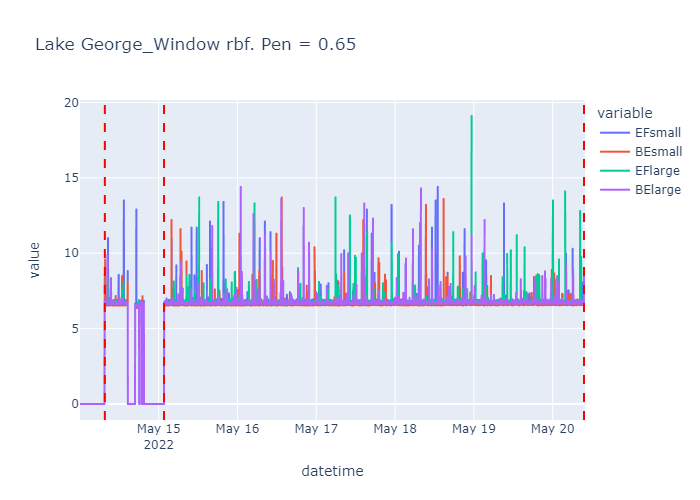


Figure .

However, comparing this to figure 2.4 where we used a bottom up with l2 cost function, we can see how many more changepoints we have detected.

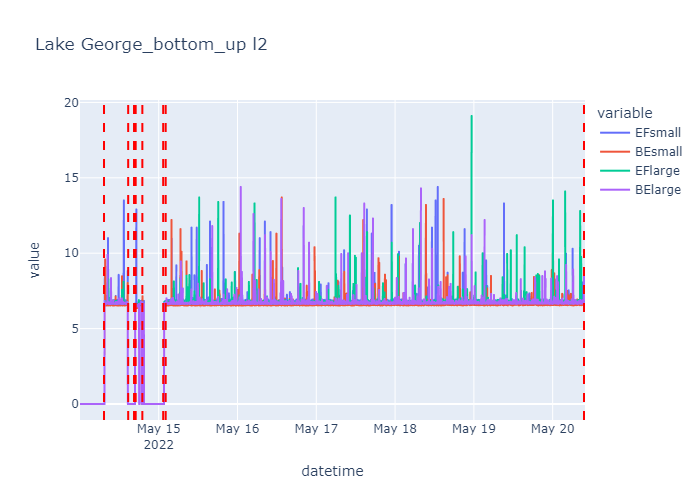


Figure .

Although the Rbf model looks good, if we run this on the entire dataset, we come back with a total of only 2 graphs in all 363, where this algorithm detects a changepoint detections. This is clearly wrong. At this stage, we could go through each graph and manually label the changepoints, then run a supervised ML algorithm, however, I wanted to run with the unsupervised given that this data set is limited and we don’t have that much data. Also, time is a constraint. We can see that we have a multitude of factors that can influence both the accuracy, time and noise robustness of the CPD models.

This then brings us to the approach favoured here in this report. We can see from the above the difficulty in hyper parameter testing and tuning for unsupervised CPD algorithms. Initially, I looked at combining each of these models and looking at a weighting of CPD algorithms to test to see if some form of combination was applicable to improve the accuracy of the CPD whilst minimising false positives.

We can see that for some plots, different algorithms are better under certain situations. After this, I attempted to combine each of the potential algorithms, initially through running each different algorithm using Pelt and Window methods mentioned in chapter 3. The issue was, as mentioned previously, time. It was too computationally expensive to run the multiple algorithms in quick enough time to make this worthwhile. The chosen method was to compare different cost functions (l1, l2, rbf) for each algorithm but also, henceforth described as Change point detection Ensemble (CPDE) (Katser, 2021)

Although time taken to generate the initial results was longer than would be acceptable for the problem highlighted, once an accurate algorithm, aggregated cost function was obtained, running this on a dataset of 24 hours for each of the 363 masts was reasonably quick, and definitely within the timeframe needed between pulling datasets.

The investigation focused on 2 hyper parameters to tune for testing, one being the CPDE algorithm, one being the aggregation function, as this time, we have an aggregated cost function.

The code ran multiprocessing in both determining the algorithm and cost to use, as well as running the algorithm throughout the day to keep up to date with new data being pulled from the cell towers. We tested all algorithms using the main cost functions that we came across. This was then checked and we can see an improvement in results. The

Once settled, we then ran through this algorithm with the test set (unobserved) to see if the algorithm picked up the correct Changepoint. The focus after testing all the above changepoint detection algorithms was then to find the quickest and most accurate one. Given the time constraints mentions above, we wanted the algorithm chosen to be reasonably quick, allowing sufficient time in between data pulldowns, to get a good picture of any significant changes within the time series data. Part of the methodology for achieving this was using both multiprocessing and multithreading within python. This allows us to maximise the speed for iterations through the data of cell sites (stack-overflow, 2022) This meant testing each CPDE scaling aggregation was doable within a reasonable timeframe. The biggest issue with all the above code is evaluating the results. We clearly have achieved an initial target of improving the change detection from a more basic level (for example, CUSUM) and we have back tested many of the hyperparameters explained above. We have to then look at accuracy. It is hard, without knowing the exact change points in the data (which we are not given in this instance) to methodically and statistically evaluate our results. One last approach attempted to see if further improvements could be achieved was to attempt a Neural network approach, specifically an Autoencoder. Autoencoders attempt to deconstruct the data, (here, we are referring to timeseries data) to a smaller feature vector, and then, using a decoder, reconstruct the data using a second Neural Network. An advantage of this approach is that it can be incredibly effective against noisy data. Making it a good approach to the highly noise data of the individual timeseries we currently have

# Neural Net Approach

# References

Alan Turing Institure, n.d. *sktime.* [Online]   
Available at: https://github.com/alan-turing-institute/sktime  
[Accessed 24 06 2022].

Charles Truong, L. O. N. V., 2020. *Selective review of offline change point detection methods.* [Online]   
Available at: https://arxiv.org/abs/1801.00718  
[Accessed 05 06 2022].

https://pro.arcgis.com, 2022. [Online]   
Available at: https://pro.arcgis.com/en/pro-app/latest/tool-reference/space-time-pattern-mining/how-change-point-detection-works.htm  
[Accessed 01 07 2022].

Katser, I. K. V. L. V. &. M., 2021. *CPDE.* [Online]   
Available at: https://github.com/theovincent/CPDE  
[Accessed 1 7 2022].

Keogh, E. C. S. H. D. a. P. M. 2., 2001. , November. An online algorithm for segmenting time series.. *IEEE international conference on data mining,* pp. (pp. 289-296)..

Killick, R. F. P. a. E. I., 2012. Optimal detection of changepoints with a linear computational cost.. *Journal of the American Statistical Association,* pp. 1590-1598.

paperswithcode, n.d. [Online]   
Available at: https://paperswithcode.com/task/change-point-detection/codeless  
[Accessed 10 06 2022].

Ruptures python library, 2022. *ruptures python library.* [Online]   
Available at: https://centre-borelli.github.io/ruptures-docs/  
[Accessed 10 06 2022].

S. F., n.d. [Online]   
Available at: https://github.com/salesforce/Merlion  
[Accessed 01 05 2022].

stack-overflow, 2022. [Online]   
Available at: https://stackoverflow.com/questions/27455155/python-multiprocessing-combined-with-multithreading  
[Accessed 1 07 2022].