2022 PRACTICUM

CHANGEPOINT DETECTION OF TIME SERIES DATA

FOR COMNET MOBILE DATA PING TIME

BY

ALEX WALTON

(GT-ID: 903398136)

OMSA PRACTICUM

Abstract

The idea behind this project is to detect significant changepoints in ping response time from cell towers around the US. The data is unlabelled, and presented to us in raw form. The data is also extremely noisy, and of the 363 cell towers, there are various magnitudes of signal that the final method must be able to deal with. The approach taken needs to be able to deal with significantly noisy data, and be able to distinguish between noise and real underlying changes in the time series data. A range of methods are explored, from unsupervised detection to a neural net based approach, to see if a model can be chosen that has good accuracy in detection changepoints in the data.

The data is from 363 cell towers, with 4 variables of the multivariate data, being EFlarge, EFsmall, BElarge and BEsmall which represent different data packets. 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 all other data at all times. The algorithm not only has to be accurate in detecting changepoints within the time series data for each variable on each ping site, but also has a time constraint component. The data is pulled every 20 minutes from 2 servers, therefore this algorithm needs to be able to scan all the data within this time frame, as well as be able to show results, to allow engineers to highlight issues in reasonable time.

In this project, I will discuss the methods I have used, the difficulties, and also the successes that allowed a final approach to be recommended, one that should improve the detection of changepoints as well as improve the accuracy of false positives.

Table of Contents

1. Introduction

For this project, data from a plethora of mobile sites around the US, each with a time series 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)

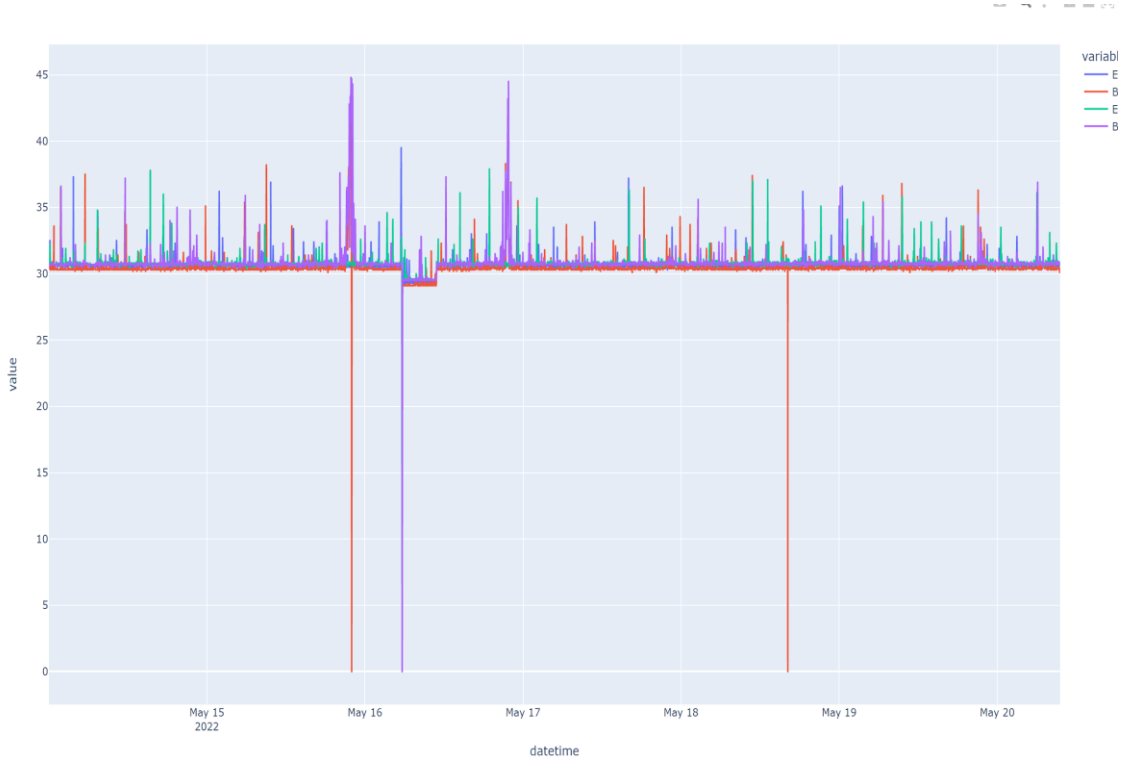

Figure 1.1

The aim of the investigation is to correctly identify when a statistically significant change in the mean of the time series for each variable occurs. Changepoint Detection (CPD) 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 difference between offline and online changepoints is 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 relevant areas in-between data being redownloaded, this would be a successful outcome from this project. Alongside this, using offline detection allows us to visually see what the data looked like before a potential changepoint. This allows us to visually see the accuracy of any algorithm used.

After making the choice of using an offline detection algorithm, discussed in the previous paragraph, 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 the aim is to reduce the number of false positives that are flagged to the user. CUSUM detection detects any change in mean, which can be highly inaccurate for the statistics of the dataset provided. In fig 1.1 we can clearly see a changepoint. Compared to fig 1.2, where the data is extremely noisy, and there is not a clear change in the mean of any of the pings observed.

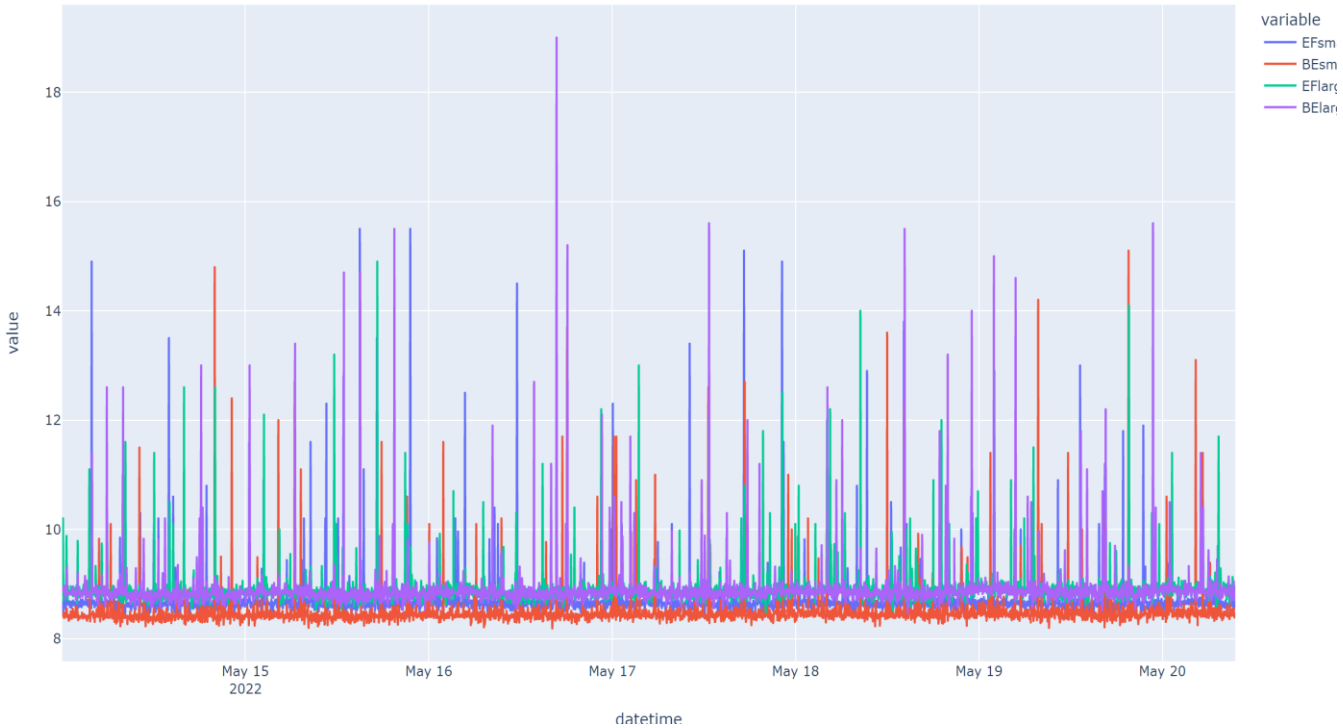

Figure 1.2

We can see from the two plots shown above that we have very different types of data. Alongside this, the data supplied is unlabelled, that is, there is no given timestamp of a known significant change in the mean. This provided a challenge, as to how to measure the accuracy of the model without manually analysing and labelling all plots to see if the algorithm tested was detecting the correct changepoints.

Therefore, for this project, to attempt changepoint detection in the given data, a number of more advanced detection algorithms were used, and evaluated. Initial attempts included simple changepoint detections, which were investigated before settling on a much more accurate ensemble of changepoint detection algorithms. I also created a python framework, which allows the end user to rerun the different cost functions, and on different data. This allows customisable CPD in the future.

2. Approach and methodology

There are many different approaches to changepoint detection, and therefore many different 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, or offer a unique approach altogether. Having unlabelled data meant that either the data could be hand labelled, or an approach that didn't require data to be labelled could be used. For this project, and given the fact that the data was only a small window of the time series data generated, I opted for an unlabelled approach, that would then be assessed graphically. This saved time initially in that labelling the data would have been a manual process which is time consuming and prone to human error, however, it meant that algorithms suitable for unsupervised learning had to be used. For example, libraries initially looked at were sktime (Alan Turing Institute) as well as a claSP approach (Patrick Schäfer, 2021) Both methods were initially attempted for changepoint detection but required labelled data and were not generalisable enough for this wide range of noisy data in the dataset provided. Fig2.1 shows the output of the claSP attempt on a particular site, where a known changepoint occurs, and it's clear that it is detecting erroneous changepoints.

Therefore, the three unsupervised changepoint detection algorithms tested are outlined below, as well as a Neural Net approach which is discussed later in the project.

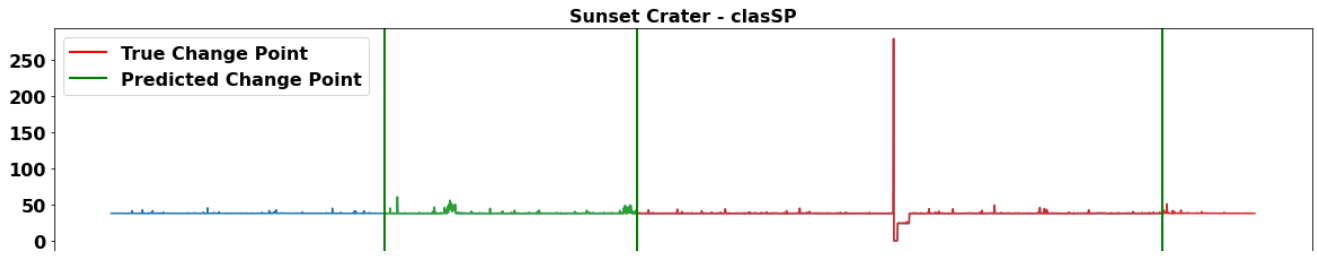

Figure 2.1

3. Methods

3.1. PELT

The Auto-detect number of changepoints (PELT) option uses the Pruned Exact Linear Time (Killick, 2012) algorithm to estimate the number and location of changepoints. PELT works by adding a penalty term for an inclusion of any additional changepoint. This means that the pay off between adding the new changepoint into a sequence of observations, the cost of the sequence is weighed against the penalty term for each point, if the cost is higher that the reduction in the segment cost, we will not include the segment as a changepoint Window (Killick, 2012 -Theorem 3.1 p.8).

3.2. 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 of the 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 log(n)).

3.3. Window

The sliding window algorithm is another quick method. 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.

3.4. Autoencoder

An autoencoder based approach to changepoint detection, using Time invariant representation (TIRE), was applied to the dataset. This approach detects changepoints including abrupt changes in the slope, mean, variance, autocorrelation function and frequency spectrum (De Ryck, 2021). The TIRE approach returns a probabilistic value to each timestamp in the time series data, suggesting if a changepoint is at, or near, this point in time.

Of the methods highlighted above, the first three of these are well known CPD methods, with a plethora of literature online. For this project, the main python library used to aid in the investigation and results was the ruptures library (Ruptures python library, 2022). This was chosen as an unsupervised machine learning library used to find the unknown number of changepoints within each multivariate time series. This library was chosen after trials of other similar libraries, as it was more customisable for the outcome of this project. For the changepoint methods above, these are made of three components: cost function, search method and a constraint or penalty value. The cost function measures homogeneity: this choice influences the type of changes that can be detected. For example, a cost function "l1" will measure changes in the median of the signal, whereas "l2" will detect mean shift. The search methods are those mentioned in 3.1 to 3.3, these

methods offer a pay off between accuracy and computational complexity. Finally, the constraint is used in the application of these methods, as we have an unknown number of changepoints in each time series. A penalty value too high penalises inclusions of potential changepoints too much, resulting in only the most extreme changes or even no changepoints at all. Conversely, if the penalty value is too low, the algorithm will detect too many changepoints, and become too noisy. Approaching this project from a systematic and analytics perspective, I wanted to test the three unsupervised methods above initially to understand whether any of these models were a clear outlier in their detection ability for the dataset. I set trials of each of the three CPD approaches.

4. Results and Discussion

Initial investigations surrounded observing if any of the previously mentioned CPD methods showed a better outcome than the other methods discussed above, on the data. To achieve this, I sampled some of the detection algorithms with different parameters. Below is one of the first unsupervised methods using bottom-up algorithm, method 3.1, with a penalty value calculated as the below. Here sensitivity is a value between 0 and 1 that can be changed. (https://pro.arcgis.com, 2022)

$$pv = (0.25 * n)^{(1-sensitivity)} * 2ln(n)$$

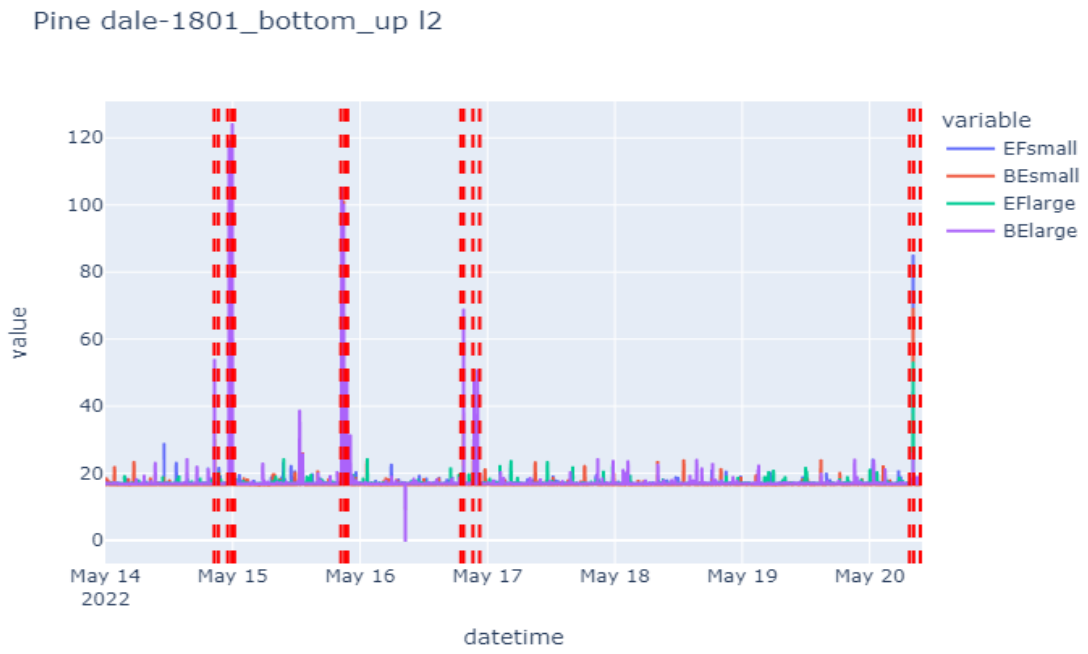

Figure 3.1

You can see from fig 3.2 that the bottom-up algorithm, with the predetermined hyperparameters, detects changes in the time series (albeit more than possibly required). However, this particular CPD approach did not seem to deal well with highly noisy, and non-substantial changes in mean. As fig 3.3 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.

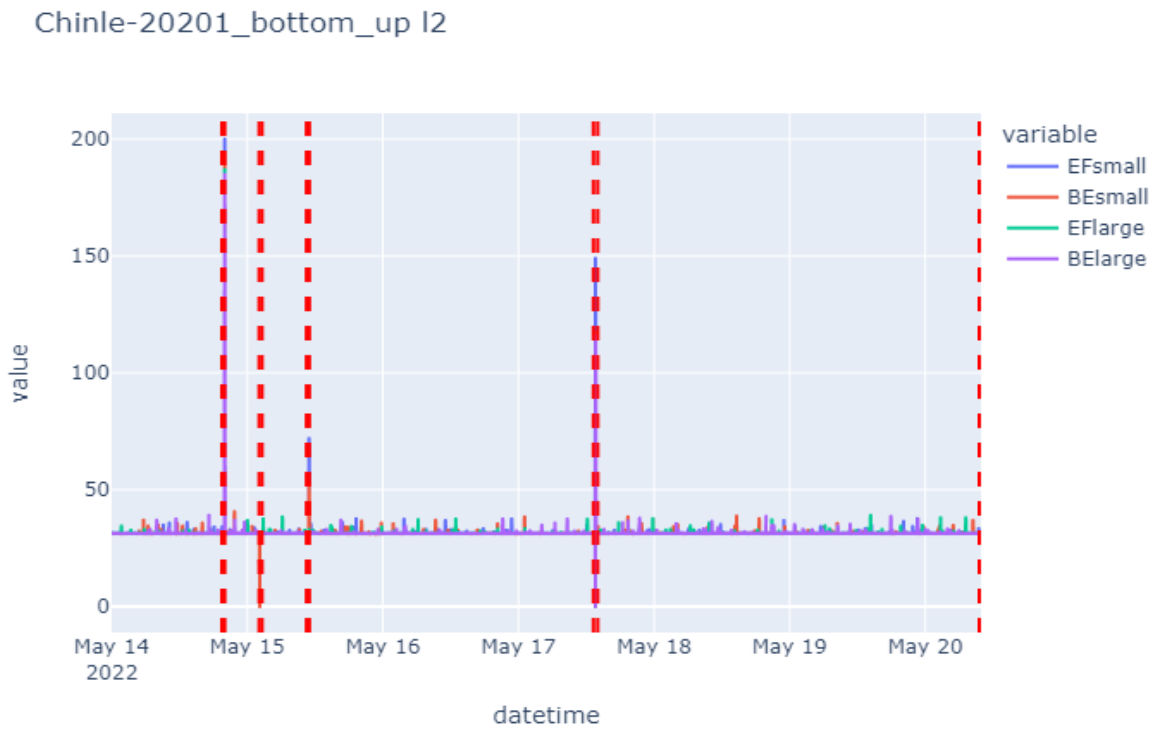

Figure 3.2

Looking at the same CPD algorithm, but this time with a different cost function (rbf as opposed to l2), we can see using a PELT CPD algorithm (method 3.1) with a kernel cost function, as shown in fig 3.3 can prevent too much noise impacting the change detection algorithm.

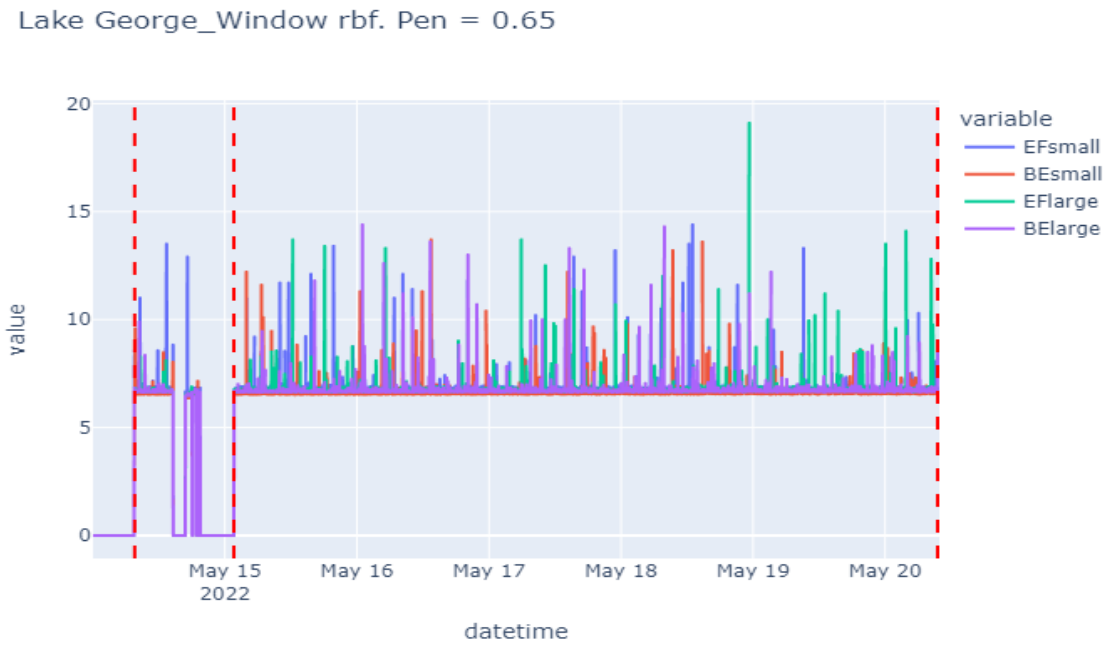

Figure 3.3

However, comparing this to figure 3.4 where we used a bottom up with l2 cost function (method 3.2), we can see how many more changepoints we have detected.

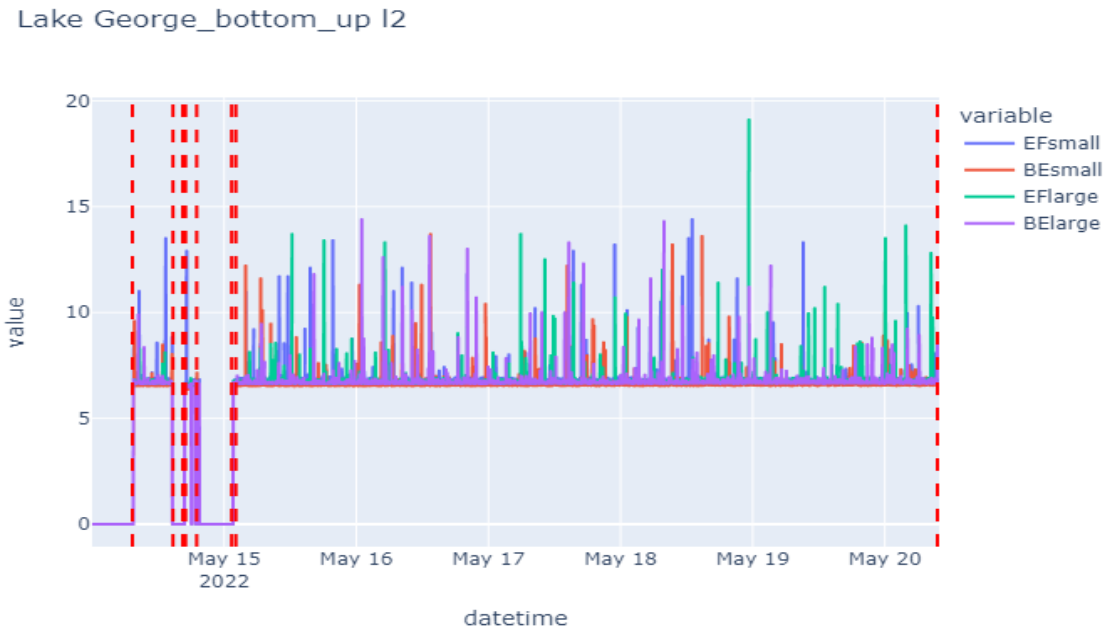

Figure 3.4

Although the rbf cost function with the PELT model looks good, if this is applied to the entire dataset, there are only 2 graphs out of 363, where this algorithm detects a changepoint. Analysis of the data we have, done graphically, shows that out of the 363 sites we have the data for, 43 appear to have a significate changepoint in the data, which is 4.76%. Detecting just 2 of the 43 potential changepoints gives us an accuracy of 4.65%. This is clearly not detecting the required changepoints in the data. From the above initial tests and examples, the number of possible hyperparameters combined with cost functions is very large. Clearly, a thorough approach is required to determine the best combination of these methods this stage, rather than manually labelling the changepoints in each graph, then running a supervised ML algorithm, I wanted to run with the unsupervised approaches given that this dataset is small and therefore back testing the algorithms is limited. It can be seen from the example plots above, that a multitude of factors can influence both the accuracy, and noise robustness of the CPD models. The running time of each of the aforementioned methods also needs to be taken into account. PELT has a much longer run time than the other CPD methods in chapter 3.

This then brings us to the favoured approach in this project for the unsupervised models mentioned. The above highlights the issue in CPD method detection, as well as the cost functions which can be applied to each of the CPD algorithms. From an analytics perspective, starting with an initial reasonable guess is a good first step. However, with the data provided there doesn't seem to be a good, generalisable method, cost function and penalty value to continue with. Learning hyperparameters would normally be approached using randomised grid search to determine the best hyperparameters to use. I attempted to replicate this approach, but with an accuracy determined from the graphical representation. Initially, I looked at combining each of these models with a weighting of CPD algorithms to test if some combination was able to improve the accuracy of the CPD whilst minimising false positives. We can see that for the initial plots, different algorithms are better under certain situations. However, this approach was superseded from further investigation. 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 a reasonable time to justify this approach. The chosen method was therefore to combine the different cost functions available (l1, l2, rbf) for each algorithm. Henceforth described as Changepoint Detection Ensemble (CPDE) (Katser, 2021), this approach takes the benefits of an ensemble method in choosing a cost function. This removes the user's choice of cost function selection. The cost function is always trying to be minimised; therefore, this approach then uses a scaling and

aggregation decides how the “votes” of the cost functions are chosen. The scaling method choices can be vast, and therefore iterating through these was required and tested.

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

The investigation focused on 2 hyperparameters to tune for testing, one being the CPDE algorithm, the other being the aggregation function, as this time, there is an aggregated cost function. The reason aggregating cost functions are used, is that each cost function looks at changes based on different statistics. So, ideally, the cost function would be specific to each time series within the dataset. This presents challenges, and therefore aggregating all cost functions available is a way to overcome this problem. The algorithm was then run with the test set (unobserved) to see if the algorithm picked up the correct changepoint location. The focus after testing all the above changepoint detection algorithms was then to find the quickest and most accurate one. Given the time constraints mentioned above, the chosen algorithm needed 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). It also allowed testing the CPDE scaling aggregation within a reasonable timeframe. The biggest issue with all the above 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 changepoints in the data (which we are not given in this instance) to methodically and statistically evaluate our results. I dealt with this by running an “evaluation” model, which set about plotting a plot of for each of the models and aggregation functions, for only a 24-hour lookback window.

The results from testing all aggregation functions within the CPDE framework, shows some differences when looking at the CPDE method, with different aggregation functions., but for each aggregation function, there are varying degrees of success. Fig 4.1 shows the for Max_Rank aggregation (Katser, 2021), using the window method, two potential changepoints are highlighted along with two none changepoints. Whereas, fig 4.2, with a Min_Znorm aggregation function two changepoints are highlighted, showing success. Min_Znorm is clearly the more accurate aggregation functions, when looking at fig 4.3, again it seems to highlight potential changepoints accurately. Although this aggregation function is better in general, is it still not perfect. Looking at fig 4.3, it is clear noisy data still poses a problem.

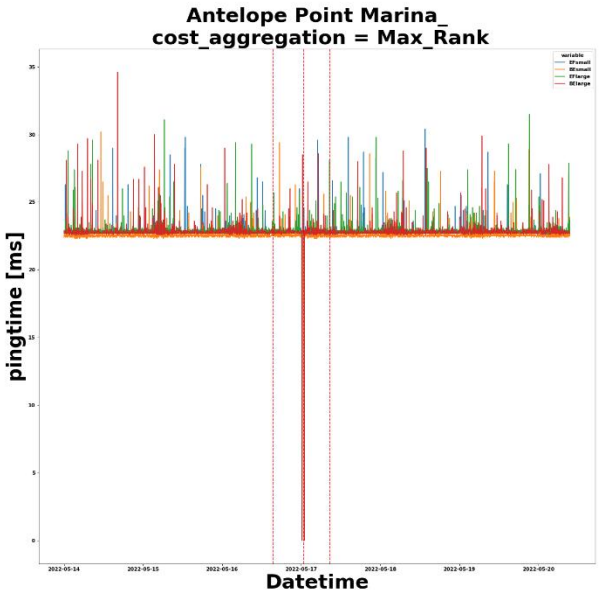

Figure 4.1

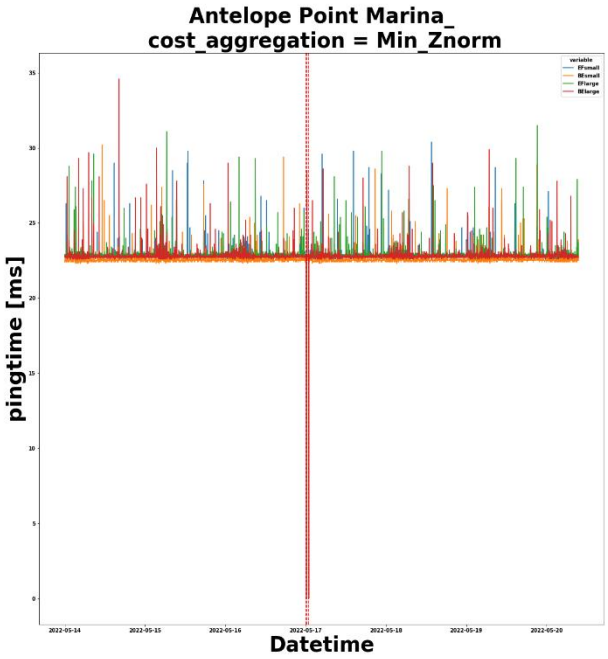

Figure 4.2

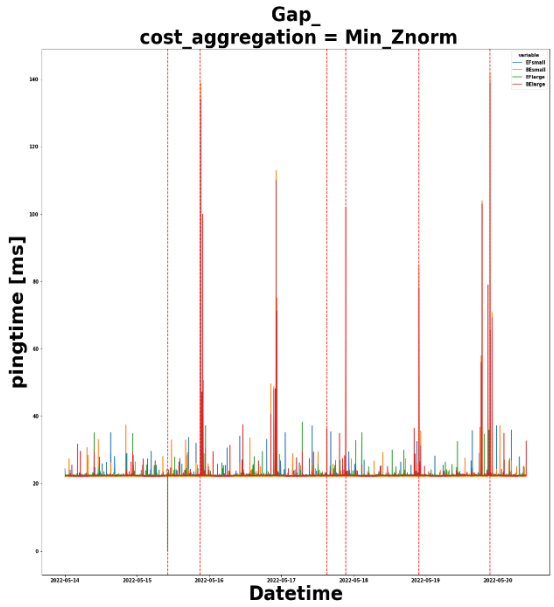

Figure 4.3

The final approach attempted to see if further improvements could be achieved was to attempt a Neural Network autoencoder. Autoencoders attempt to deconstruct the data, (here, we are referring to time series 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 noisy and individual idiosyncrasies of time series data currently presented for each phone data mast. Autoencoders are unsupervised, which again fits the specificity of the problem. This approach aims to locate significant changes in time series data. The auto-encoder based mythology attempted was the Time Invariant Representation (TIRE) model (De Ryck, 2021). The use of the TIRE model allowed me to run the changepoint detection with improvements: essentially from all the data, to observe a change in the time series, but also a significant and pronounced one. It can be difficult to remove false positives when the data is noisy. The output from the TIRE model, with statistics applied in how the model decided that peaks were dissimilar enough, meant improving the differentiation score between the potential changepoints observed from the model's output.

The autoencoder model produces two outputs, the first is a “similarity plot” which highlights the peaks the model detects, and how dissimilar they are to the other peaks/data within each variable for each site. The model then measures the width of each peak, but at a height ratio of 0.65. This means that we should be able to detect not just a datapoint that is very dissimilar, but also, is dissimilar for a reasonable amount of time. A high peak, but very narrow, is most likely noisy data that we don’t want to highlight as a changepoint. Measuring a peak slightly lower down should only capture data where the peak is reasonably wide, thus equating to a time significant change in the mean.

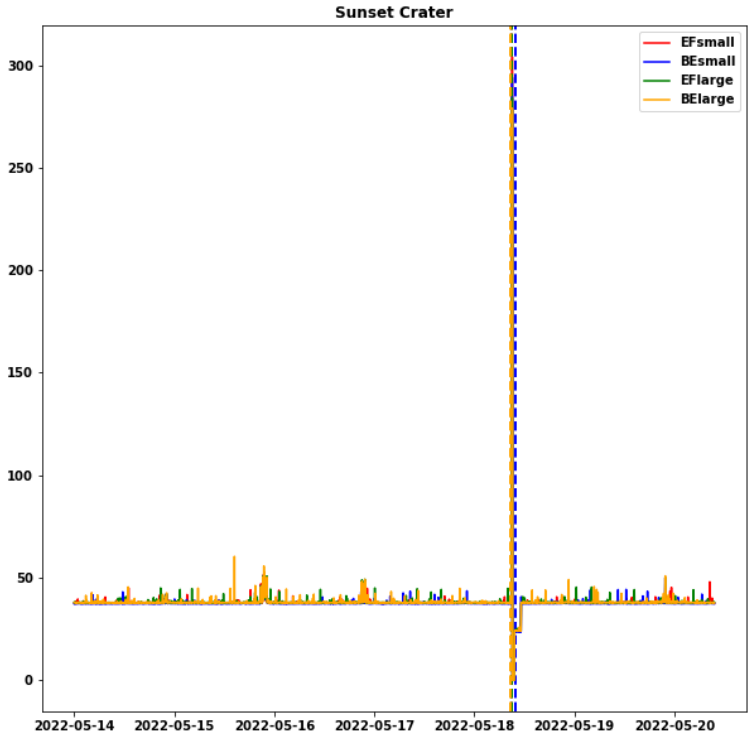

Figure 4.4

Figure 4.4 shows a reasonably noisy time series data, with one very clear changepoint that results in a significant and prolonged change in the mean. We can clearly see that for all variables there is a jump in the data, followed by a sharp drop. Looking at the model’s dissimilarity plots:

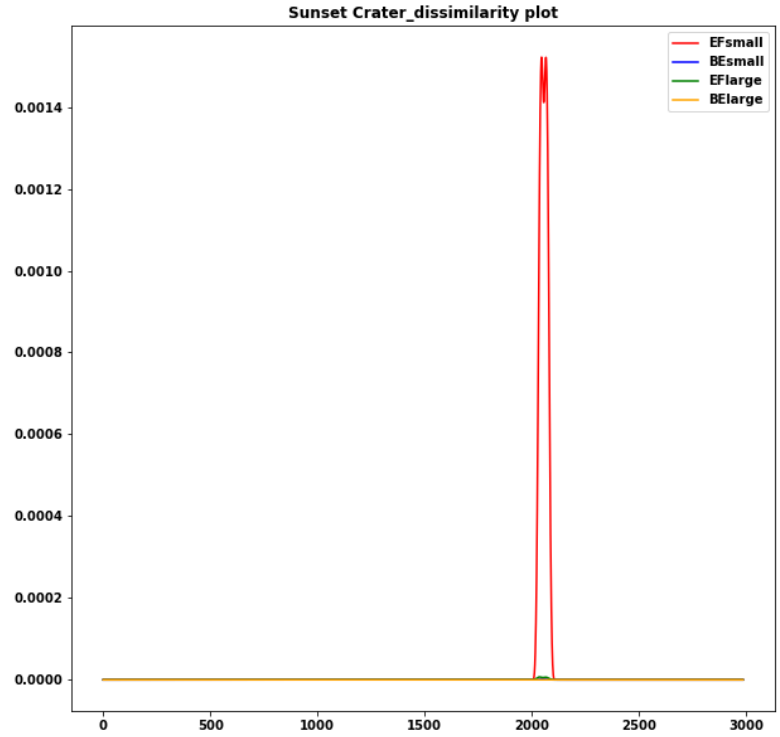

Figure 4.5

Figure 4.5 shows a clear pick up form the model's dissimilarity plots. For BElarge, the dissimilarity plot shows a large peak which is wide. This therefore means the changepoint detection can be approached in a slightly different way. Instead of trying to detect changepoints from extremely noisy data, by applying the change detection to the dissimilarity plots, there is a much smoother function to apply statistically changes to. One route was to re-run the changepoints we looked at earlier, however without making the algorithm too time consuming, we can leverage SciPy's signal module (https://docs.scipy.org/, 2022). This module allows us to detect peaks, and after applying some outlier analysis, we can obtain the index location of potential changepoints within the original data.

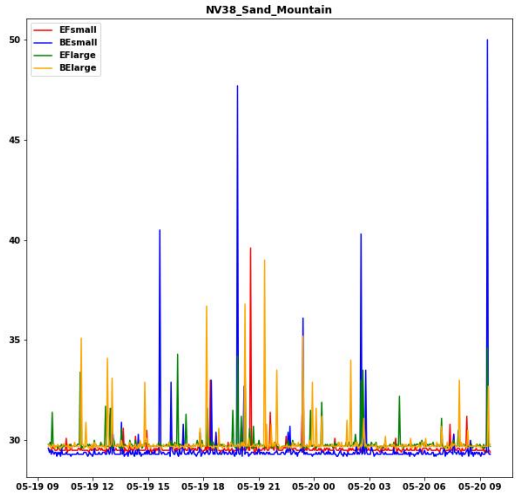

Figure 4.6

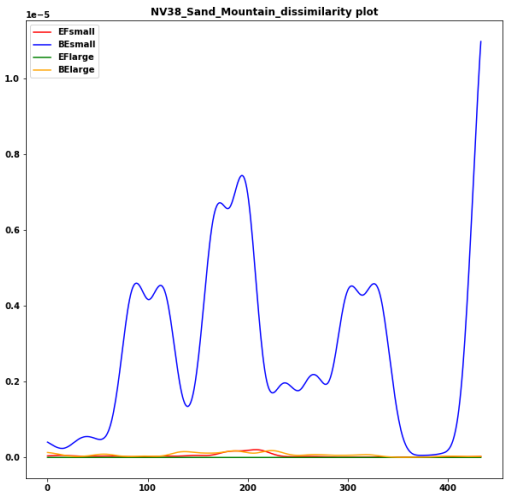

Figure 4.7

By looking at the dissimilarity plots, as opposed to the raw time series, we can see that identification of noisy data becomes much easier for the algorithm to observe. Fig 4.6 shows the extremely noisy ping data of one mast over 24 hours. The corresponding dissimilarity plot in fig 4.7 shows that for the spikes in ping time, although they identify dissimilarities, there are numerous, all similar. This implies that although there are spikes, these are not unique, hence this data is most likely noisy. The TIRE model, combined with SciPy's signal model and outlier removal seems to produce the best evidence yet of a method that is able to both detect changepoints in a plethora of timeseries data, along with dealing with the extreme noisiness of the data provided. By taking advantage of deconstructing the data to a smaller vector, the model is able to detect and ignore more noisy data, whilst still detecting changepoints that are significant.

The implementation of the autoencoder model will allow the project sponsor to apply this model to their current data, hopefully improving their changepoint detection methodology, thus allowing engineers to quickly and accurately highlighting potential issues with the data, whilst filtering out noise. The second advantage of the way this module has been constructed, is that the sponsor can alter and change the autoencoder inputs. This means they can increase the epochs, change the setting for the outlier detection, hence making the model more customisable. The value-add in quickly highlighting potential issues and not wasting time, cannot be underestimated where the alternative is sending an engineer out to a remote location to potentially fix the issue, which can be a waste of time and resource.

5. Conclusion and further work

The aim of this project was to apply and utilise the skills and analytic insight I have gathered over my time taking the MS analytics at Georgia Tech. Applying it in a real-world situation, both in attempting to build on that knowledge, as well as aid the sponsor in improving their detection of errors within their ping time series. Initially, the issue presented, to detect significant changepoints in ping response time from cell towers around the US, was approached with a simple application of detecting changes in time series with models that were relatively simple. However, that progressed into enhancing and improving libraries that were already available, within the python development framework. This led to a deep investigation of different, unsupervised, changepoint detection algorithms. Each of these were investigated, and a final model chosen. The code passed to the sponsor should allow them to review this data, but also make changes to any of the implantation of the models themselves, to back test or run-on different subsets of data. The application of the models, as well as the outcome, will allow the sponsor to detect changes in the times series more accurately, and within a reasonable time. This can be absorbed into their current detection framework seamlessly.

The big advantage there is to the company this project is for, is that they now have a customisable neural net, wrapped with statistical analysis of the dissimilarity data, that allows this to be improved and enhanced. For example, one could take the model and use the parameters generated for each time series and increase the number of epochs. Also, by saving down the weights of the Neural network, it may become clear over time that certain mobile sites have similar characteristics.

The hardest part of the project was assessing the accuracy of the algorithms. This has been talked about throughout this practicum, and it's never going to be perfect. By leveraging the knowledge gained from neural networks, the final algorithm related to application of autoencoders on time series data, then looking at detection of changes in the dissimilarity graph, is a somewhat unique approach and one that can be built on. By having the time to focus on the different changepoint detection algorithms, it has been possible to improve the current detection process for ping data and allow better and more accurate detection. Applying statistical and analytical knowhow to extremely noisy data, I was able to remove many of the false positives experienced in previous detection attempts

As highlighted in the results, I found an improved method to the detection of time series data from each cell tower, over and above a simple change in mean. This was done by harnessing the use of all the available data, as well as looking at the plethora of different ways we can detect changepoints within time series. Through observing the results from the different CPD methods, along with hyperparameter testing, we saw an improvement in detection. However, this was still not generalisable very well to the data we had. The last approach was to try the autoencoder TIRE model. This approach generated the best detection out of all those attempted in this project. It deals well with extremely noisy time series data and reduces the number of false positive changepoints.

Further work can now be built on from the project. Improving the training of the neural net has already been mentioned, along with placing the algorithm on a scalable and interactive dashboard. The real insight here is the ability to detect different change points in various types of time series data. Some being extremely noisy, whilst others are not. More work can be done on enhancing the neural net approach, as this seems to be the most promising: from increasing the number of epochs, tailoring the loss function and identifying different ways to detect peaks within the dissimilarity data, one could potentially improve the detection further. Further work on training the TIRE model to each of the masts themselves could also be tried. The underlying ping time for each mast may have idiosyncrasies that the model, once trained, would pick up on. By giving the model much more data than that provided for this project, one could have a model trained on each of the individual masts themselves. The model would then potentially learn the underlying data of each mast, producing even more accurate results.

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].

De Ryck, T. a. D. V. M. a. B. A., 2021. *https://github.com/deryckt/TIRE.* [Online]
Available at: https://github.com/deryckt/TIRE
[Accessed 1 06 2022].

De Ryck, T. D. V. M. a. B. A., 2021. Change point detection in time series data using autoencoders with a time-invariant representation.. *EEE Transactions on Signal Processing,* Volume 69, pp. 3515-3524.

https://docs.scipy.org/, 2022. *https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html.* [Online]
Available at: https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html
[Accessed 5 7 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].

Patrick Schäfer, A. E. U. L., 2021. ClaSP - Time Series Segmentation. *CIKM '21: Proceedings of the 30th ACM International Conference on Information & Knowledge Management,* pp. 1578-1587.

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

Sales Force, n.d. *https://opensource.salesforce.com/Merlion/v1.1.0/merlion.models.anomaly.change_point.html.* [Online]
Available at: https://opensource.salesforce.com/Merlion/v1.1.0/merlion.models.anomaly.change_point.html
[Accessed 15 6 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].