Microsoft Azure Time Series Anomaly Detection Report

Azure Machine Learning Studio

The Time Series Anomaly Detection developed by Microsoft [1] is a module within the Azure Machine Learning Studio. In the Machine Learning Studio there are many different modules you can use to conduct machine learning experiments with virtually no coding needed. These modules are drag-and-dropped into the workspace and parameters can be selected. However, if needed there are modules that allow you to write scripts in Python, R or apply SQL transformations.

Figure 1 shows an example machine learning experiment that can be carried out in the Machine Learning Studio for automobile price prediction. Modules were used to preprocess the data, split it into a training and test set, train a model and, finally, evaluate it. No coding at all was needed for this, just selection of parameters and what columns of data to use.

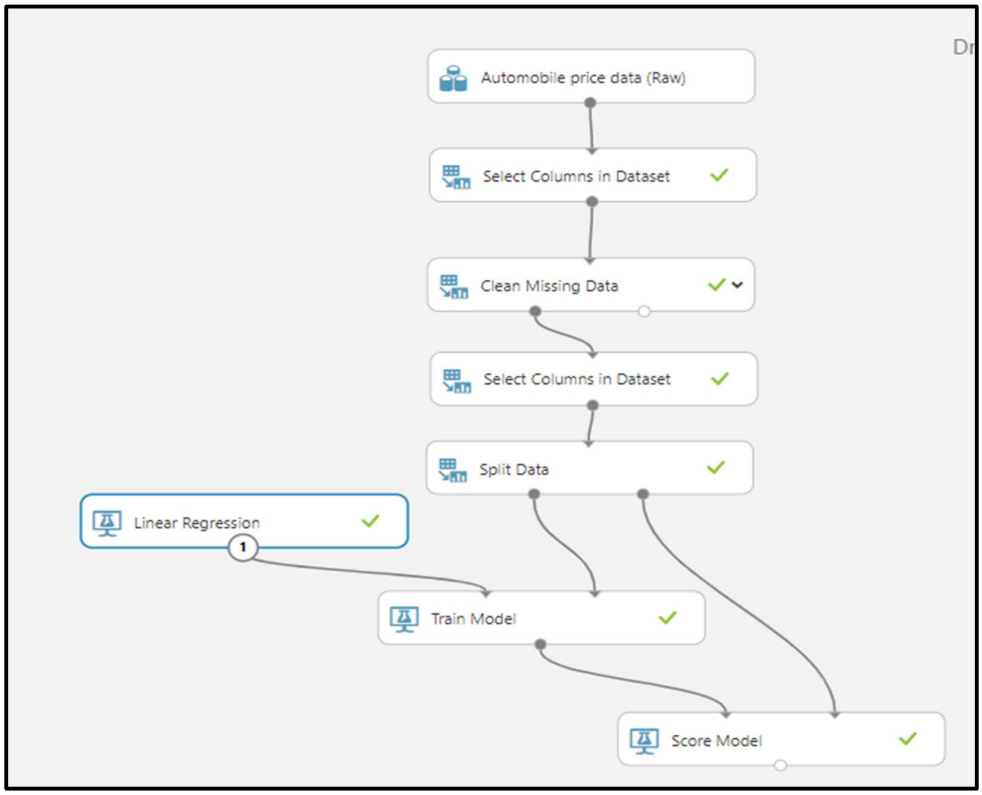

Figure 1: *Example workflow of a typical machine learning experiment in Azure Machine Learning Studio*

Time Series Anomaly Detection Module

This module, as it sounds, aims to detect anomalies in time series data. It does this via learning the normal characteristics of the data and then using those characteristics to detect deviations from the normal [1]. The module has 6 parameters that need to be chosen and two choices to make for inputs, as shown in Figure 2.

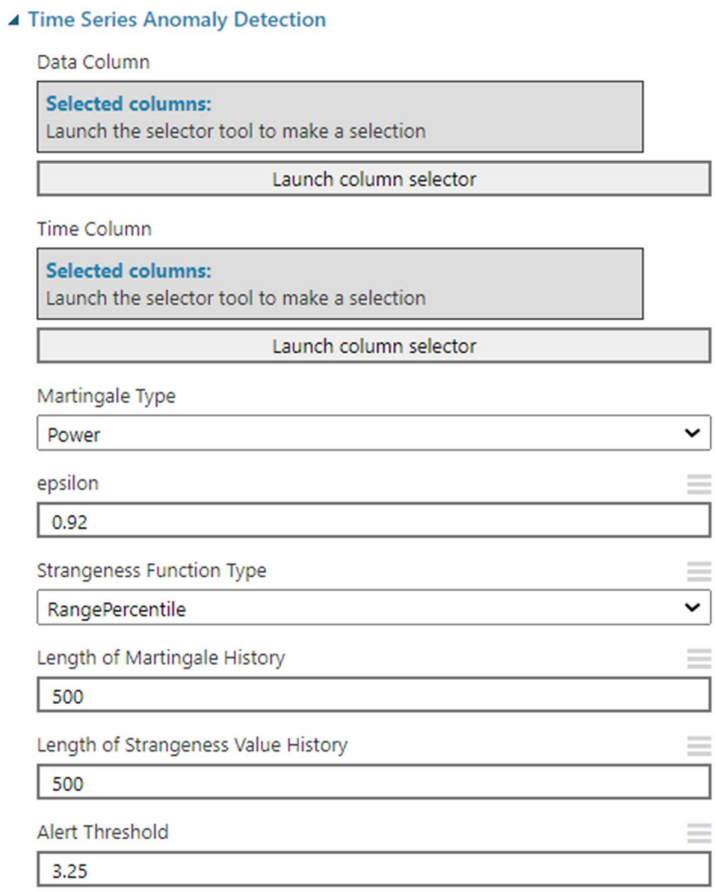

Figure 2: *Options for the parameters and data inputs of the Time Series Anomaly Detection module*

The first choice for the input is what to use as the data. For this module they only allow for a single data column to be chosen as input, i.e. the data can only have one feature. If you were only using a dataset such as population of a country over time then this would be suitable, but for more complex data involving multiple variables, the module would not work and you would have to choose what you believe to be the most important single feature. This limits the usefulness of the module.

The other input is the time column. This is very straightforward and is the column with the time the data sample was taken. The data in this column must be a DateTime object.

Strangeness Function

The "strangeness function" is a function that will assign a non-conformity score to a new sample based on the previous samples to give an indication of how well the new sample conforms to the rest of the data. The non-conformity scores for all the data points can then be used to calculate the p-value for the new sample.

$$p := \frac{|\{i = 1, \ldots, n+1 : \alpha_i \geq \alpha_{n+1}\}|}{n+1}$$

Where α_i is the non-conformity score of a sample and n is the number of previous samples used in the non-conformity score calculation.

In my experiments I have used "strangeness functions" such as k-NN, SVM and neural networks. For example, with k-NN the function for a non-conformity score could be distance to the nearest sample of the same class. If the distance is low, the sample conforms and will have a low non-conformity score, and vice versa. The k-NN and SVM methods have also been used in papers such as Fedorova et al., 2012 [2] and Vovk et al., 2003 [3].

However, from what I can tell, this module does not use any of those methods. It provides three "strangeness functions": Range Percentile, SlowPosTrend and SlowNegTrend. The documentation [1] does not provide any further information on how these functions work, other than that no parameters are required. Nor do they provide any references that could explain this. My internet research also didn't turn up any clear results on what they involve. From the name, I would guess that RangePercentile involves comparing the new samples to the quartiles of the rest of the sample to calculate a non-conformity score, as samples that lie outside of the interquartile range could be considered to be outliers. The documentation recommends this function if you are trying to detect level change anomalies. For example, in the data in Figure 3, taken from the documentation.

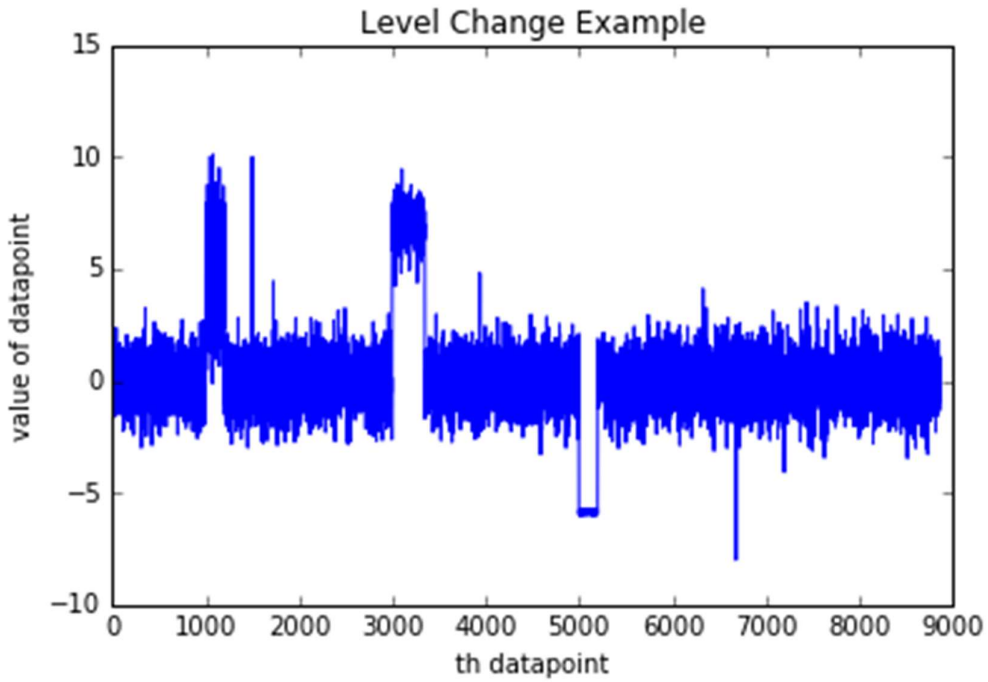

Figure 3: *Plot of level change data which could be used for anomaly detection. Graph is taken from the documentation for the module* [1]

The other two functions are for detecting anomalies in data where there are slow positive or negative trends. For the strangeness function there is also the parameter of "Length of Strangeness Value History", which controls the number of previous values to use in calculating the strangeness value.

Martingale Type

There are two options for the Martingale type, Power and PowerAvg. These are both described in Vovk et al., 2003 where the power martingale is given by:

$$M_n^{(\epsilon)} = \prod_{i=1}^{n} \epsilon p_i^{\epsilon-1}$$

Where p_i is the strangeness value for the sample and epsilon is a value between 0 and 1. The PowerAvg martingale is equivalent to the simple mixture of the power martingale and eliminates the dependence on the parameter epsilon. As with the strangeness function, there is the parameter of window size for calculating the martingale.

The final parameter is the "Alert Threshold". The anomaly detection algorithm outputs an anomaly score for each sample in the time series. The

documentation does not make it clear what this score corresponds to, but I believe it is the log of the martingale value for that sample. The alert threshold specifies when to classify a sample as an anomaly. If the anomaly score is greater than the alert threshold, the sample will be classified as an anomaly.

Comparison with my algorithm

To be able to compare the Azure module with my algorithm I had to use time series data which only contained a single feature. Therefore, I chose to use energy consumption data recorded by smart meters [4]. My algorithm uses online conformal prediction with the Laplace multipliers from a one-class SVM to calculate the p-value for each sample. Power martingales are then calculated from these p-values using an epsilon value of 0.92, these same parameters were used for the Azure algorithm. For calculating the p-values and the martingales I used a sliding window of 1000 samples as to match up with the Azure algorithm as best as possible. This was the maximum number of samples Azure allowed the sliding window to be.

For the first experiment I took ~42 days of energy consumption data from one household (Figure 4) and ran it through both algorithms to see how the martingale output compares. This is assuming that the values output by the Azure algorithm are the martingale values.

Figure 5 shows the log of the martingale values over time for my algorithm and the “Anomaly Score” over time for the Azure algorithm. From the graphs, it is clear that our output values differ significantly. Initially, my martingale values stay relatively stable, whereas the anomaly score for the Azure algorithm rapidly increases before suddenly dropping back down to zero. There are sections which mirror each other quite well, albeit with different values. For example, between samples 250 and 1000 we see a steady drop in the values then a period of small fluctuation and finally the values rising back up again. However, overall, the outputs of the two algorithms do not match up. I attribute this to be mostly due to the difference in functions used to calculate the p-values. Different conformity measures will produce different p-values and, therefore, different martingales. An example of this is shown in Figure 6, where power martingales with epsilon=0.92 have been calculated for the USPS dataset in the online mode using conformal prediction. 6a shows the martingales when using SVM as a conformity measure and 6b shows them when using one nearest neighbor. A possible explanation for the range percentile function causing a jump in the anomaly score at the start may be

much more sensitive to small changes in the energy consumption when it has little other data to work with. In Figure 4 we can see that for the first ~10 samples the energy consumption is around 0.2 kWh and then jumps up to about 0.6 kWh. Each sample represents a reading taken every 30 minutes. So, this increasing in consumption seems logical in relation to the day-night cycle, i.e. a household will use less energy in the day than in the night. However, when the range percentile function doesn't have full data on this cycle, the change in consumption appears to be an anomaly.

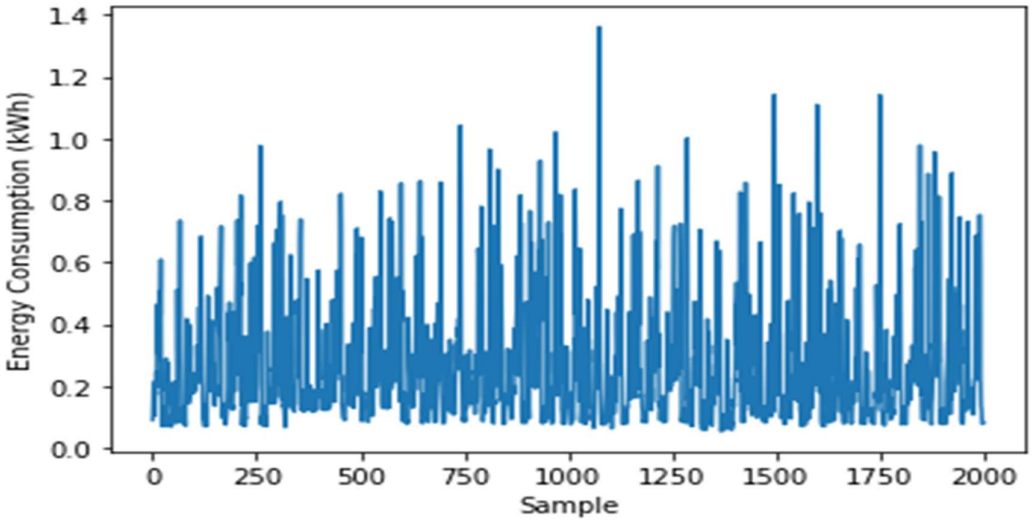

Figure 4: *Plot of the energy consumption time series data for a single household over ~42 days*

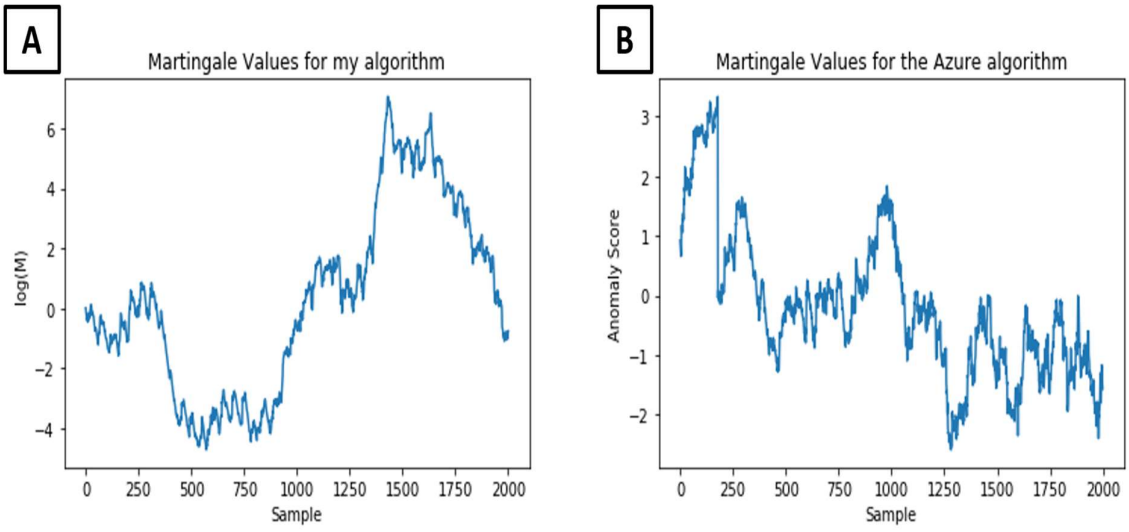

Figure 5: *Plot of the martingale output over time for a) my algorithm b) the Azure algorithm*

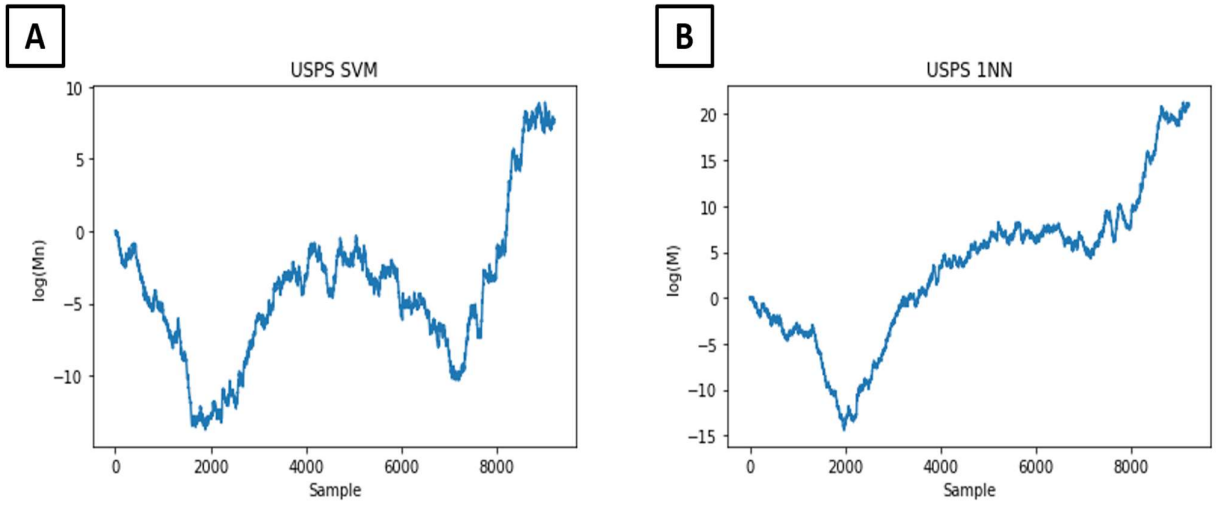

Figure 6: *Power martingales for the full USPS dataset calculated with the* ***a)*** *SVM non-conformity measure and* ***b)*** *1NN non-conformity measure using online conformal prediction.*

Since we do not know if the ~42-day sample has anomalous energy consumption periods within it, it may not be the most suitable data for comparing the two anomaly detection algorithms. For the next experiment I have added three clear anomalous periods of energy consumption data, as shown in Figure 7.

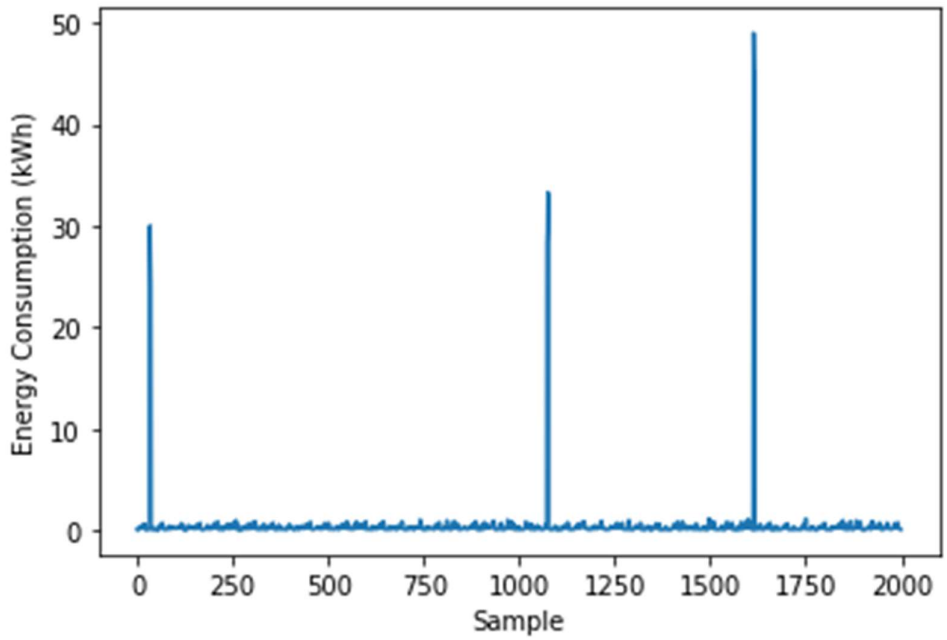

Figure 7: *Energy consumption data with synthetic anomalies added*

Although this data may not be a realistic version of the anomalies you would find in actual energy consumption data, it is still useful for testing the algorithms. If the algorithms cannot detect these anomalous periods, then they may not be fit for purpose and are unlikely to be able to detect more subtle anomalies.

Figure 8 shows the results of both algorithms with this data, with the anomalous periods marked with red vertical lines. As with the previous experiment, the martingale values between the two algorithms are quite different. However, both algorithms show large jumps in the value of the martingales at the positions of the added anomalies. Additionally, they both fail to detect the first anomaly. Although it may look as if the Azure algorithm detected the first anomaly, due to the larger anomaly score values, this is due to the samples before it and there is no large jump in the value as seen with the other anomalies. This initial increase in the anomaly score can also be seen in Figure 5b. It is likely that the two algorithms are unable to detect this anomaly due to the lack of other data at that point, since the first anomalous period spans the samples 35-37. With such a small amount of "normal" data it will be hard to detect a change from the normal.

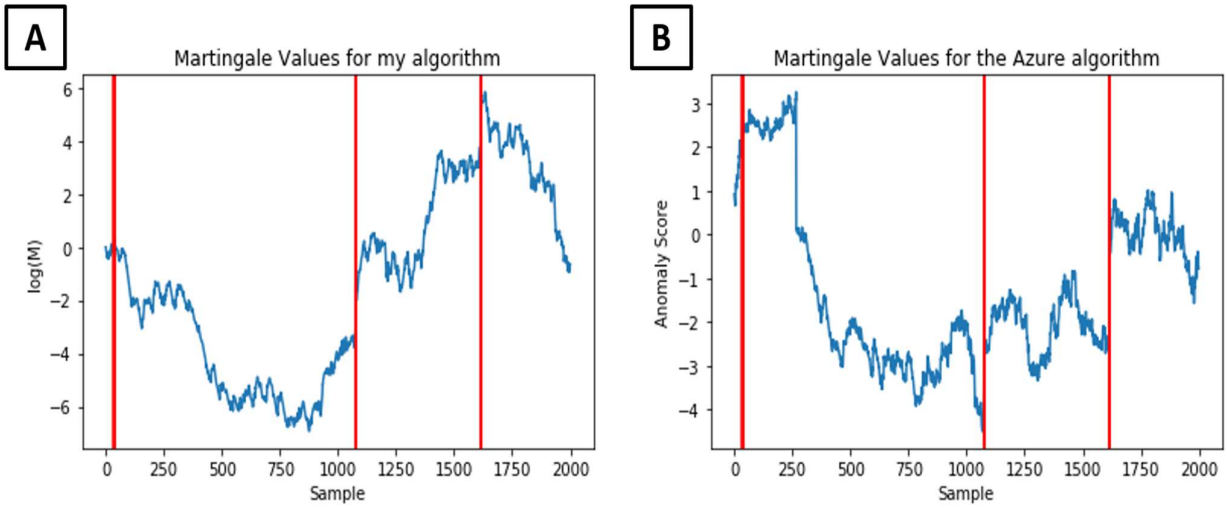

Figure 8: *Plot of the martingale output over time for the dataset with added anomalies for a) my algorithm b) the Azure algorithm. The red vertical lines mark the samples where anomalous values were added.*

Finally, I tested both algorithms on the full data set for a single household, shown in Figure 9. This dataset spans the timeframe of roughly a year. The energy consumption data for this household looks relatively normal. It has high

frequency variations caused by day-night cycles and potentially day-to-day weather variations. These high frequency variations are smoothed out in Figure 9b using a 14-day moving average. This begins to show longer term trends and highlights periods of higher energy consumptions. Since the dataset takes place over the period of a year, the higher energy consumption periods are most likely due to seasonality as more energy will be used in colder and darker months. Within these longer-term trends, however, there are still potential anomalies. The clearest of these is at around Sample 12500, where there is a significant drop in the energy consumption over a period. As this is during the long-term trend of lower energy consumption this is during the summer period and we can infer that this lack of energy consumption could be due to the homeowners going on a summer holiday. This drop is an anomaly that we would hope the anomaly detection algorithms would pick up.

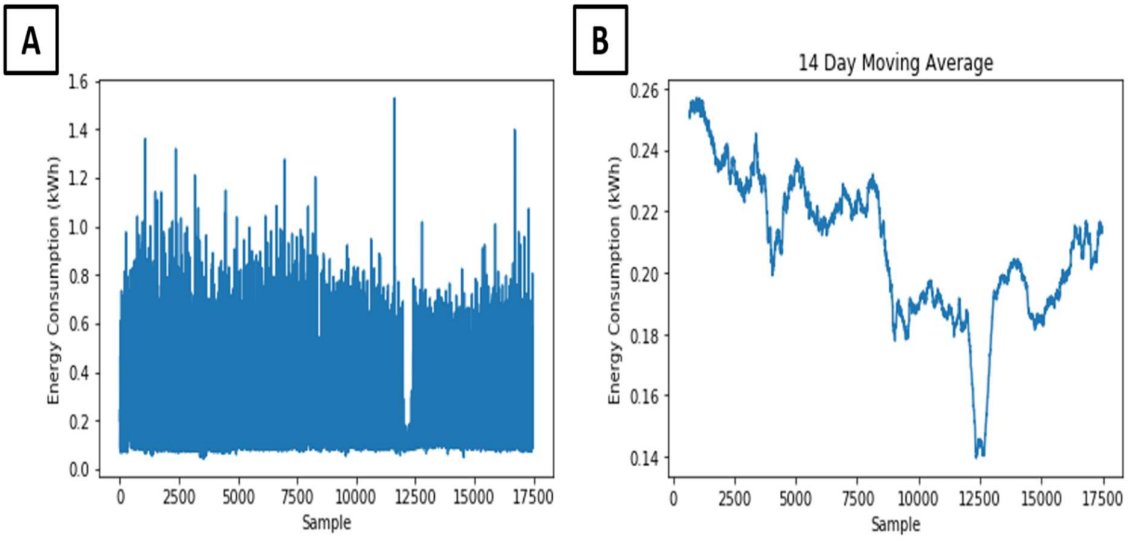

Figure 9: *Energy consumption data for a single household over a period of roughly a year,* ***a)*** *the raw data* ***b)*** *a 14-day moving average.*

Figure 10 shows the results of the two anomaly detection algorithms with this dataset. The same parameters as in previous experiments were used. Both algorithms show a similar rising and falling in martingale value over the time period. However, my algorithm has a large change in martingale value (log), from ~-15 to ~10, starting at around sample 12500. This corresponds to the anomalous period of low energy consumption and so my algorithm could be seen as successfully detecting the anomaly.

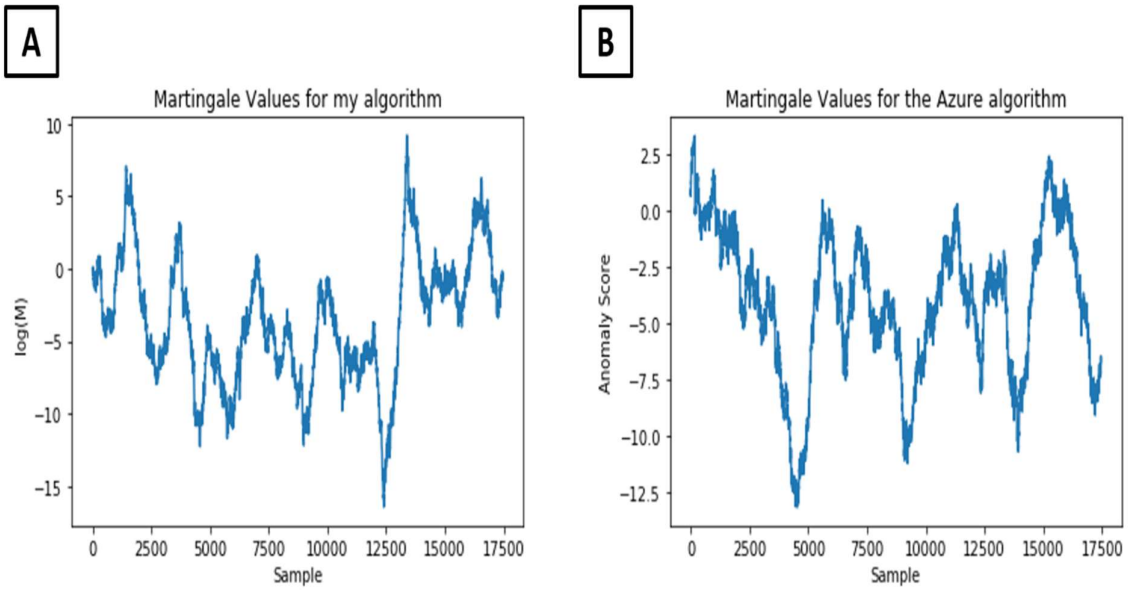

Figure 10: *Plot of the martingale output over time for the full dataset for* ***a)*** *my algorithm* ***b)*** *the Azure algorithm*

On the other hand, the Azure algorithm does not show a similar level of change for the anomaly. At the same time step there is an increase from ~-7.5 to ~-2.5 in the anomaly score, but this is significantly smaller than most of the other increases in the anomaly score over time. Additionally, since the final value of this increase is below zero, the algorithm can't detect the anomaly as the Alert Threshold parameter must be positive. This seems to be quite a significant problem with the Azure algorithm that I have encountered in my experiments. The "Anomaly Score" output tends to be below 0 and, therefore, it is impossible to detect any anomaly even if there is a large jump in the "Anomaly Score". This is evident in Figure 8b, where, even though there are two large increases in score from the synthetic anomalies, the score still remains below zero or very close to zero, making it unlikely or impossible to be detected.

In Figure 11, I have run the dataset through the Azure algorithm with a low alert threshold of 0.5 to test what samples it will flag as anomalous. This labelled 9 samples as anomalies and, as can be seen in Figure 11, the samples that it has marked as anomalous do not actually seem to be anomalous. Increasing the threshold further will just decrease the number of anomalies labelled, but they will still be the same samples labelled as anomalies before. Therefore, it can be concluded that the Microsoft Azure algorithm performs poorly on this dataset.

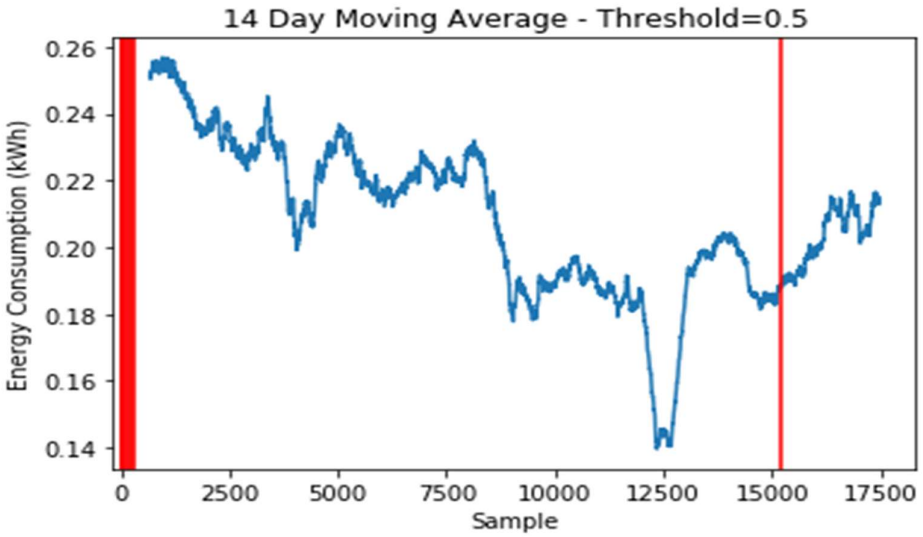

Figure 11: *Plot of the 14-day moving average of the energy consumption dataset. The samples marked as anomalies by the Azure algorithm with an Alert Threshold of 0.5 are shown as vertical red lines*

Changes to my algorithm

Before I run the same test on my algorithm, I have decided to make some changes. As can be seen in Figure 10a, the martingale values constantly oscillate between high and low values. This is due to using a sliding window. However, a sliding window is required otherwise the martingale values will get too small and not be able to recover when an anomalous sample appears. This is shown in Figure 12, where no sliding window was used in the martingale calculation.

The oscillation caused by the sliding window is not ideal as it can cause a delay in the time it takes for the algorithm to detect the anomaly. For example, if the log of the martingale starts at a low value of -10 and then rises to eventually to a value of 5, there will be a period of time where we can see the anomaly is occurring but it won't have reached the threshold for an alert. To solve this, Volkhonskiy et al., 2017[5] proposed a modification of the conformal martingale calculation which is as follows:

$$C_n = \max\{0, C_{n-1} + \log(g_n(p_n))\}, n = 1, 2, \ldots,$$

Where C_0=0, p_n is the p-value and g_n is the betting function.

This stops the martingale from getting too small and, therefore, reduces the delay for anomaly detection. The result of this change is shown in Figure 13, where no sliding window was used in the calculation. The plot shows that when a sample is anomalous and the martingale value begins to increase, it should take much less time to reach the threshold value. However, there is still an issue with these value as, after the large increase in value due to the anomaly at around sample 12500, C_n stays high even when those samples may not be anomalous. If, for example, a threshold of 4 was set for labelling a sample an anomaly, then most of the last 5000 samples would be labelled as anomalies. From looking at the data itself, it is unlikely this would be true. This issue can be solved by using a sliding window when calculating C_n and the results of this can be seen in Figure 14.

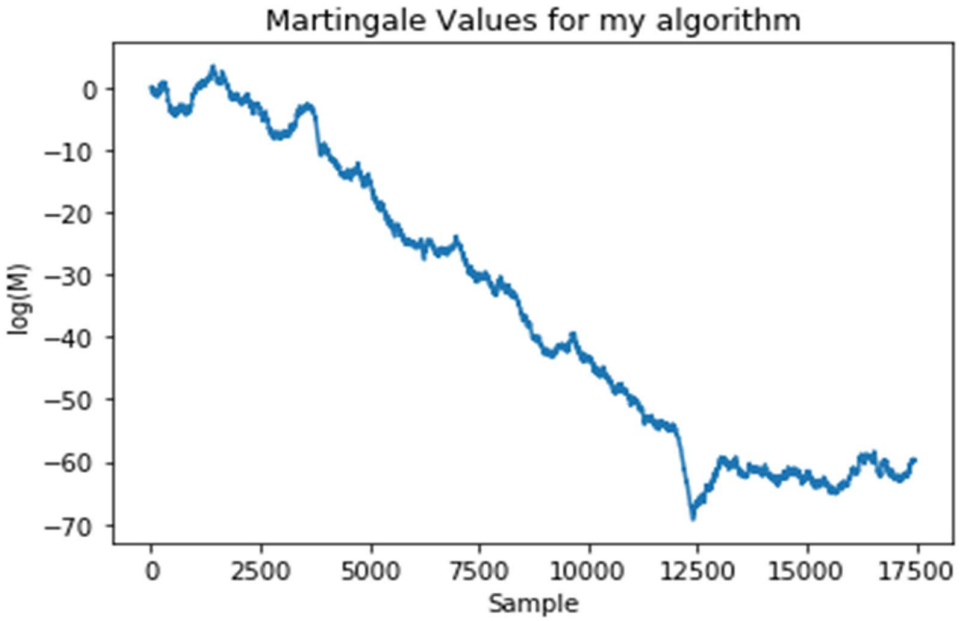

Figure 12: *Martingale values for the full dataset when no sliding window is used in the calculation.*

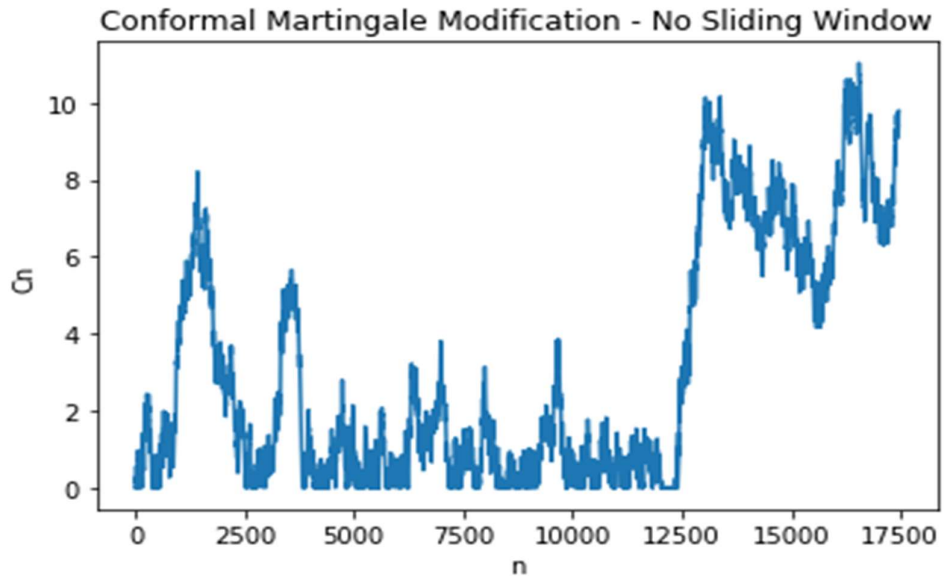

Figure 13: *C_n values calculated for the full dataset with no sliding window.*

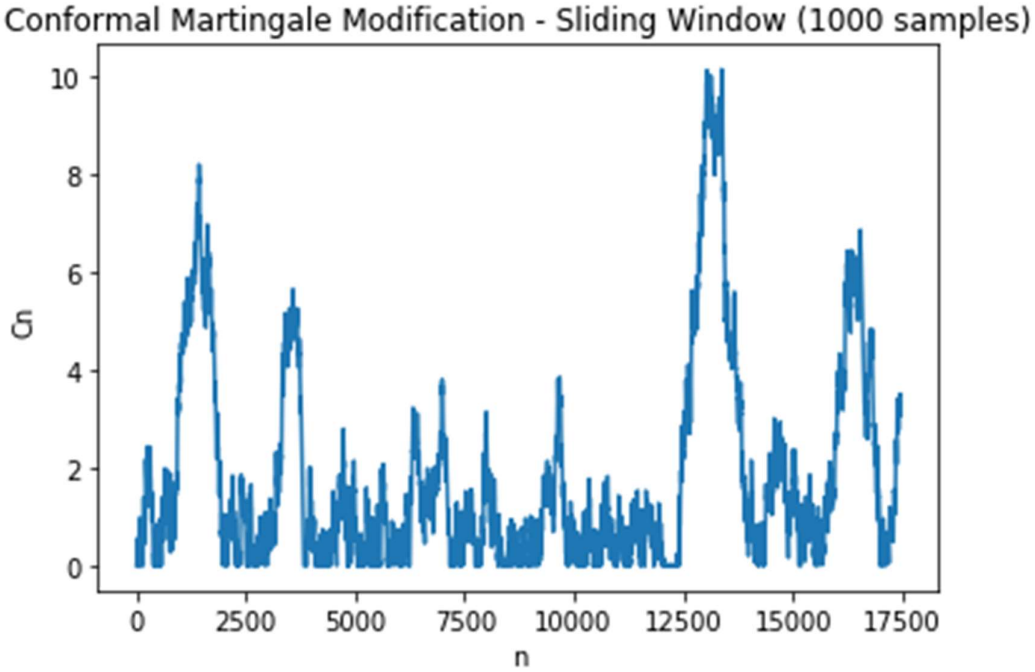

Figure 14: *C_n values calculated for the full dataset with a sliding window of 1000 samples*

As can be seen in Figure 14, after the anomaly at around sample 12500, the martingale value decreases back towards zero much faster, reducing the chance of false alarms. In the plot it is clear there are a number of samples

thought to be anomalies, shown by the spike in the martingale value. The largest spike corresponds to the period of low energy consumption in the data, as would be expected and already shows an improvement over the Azure algorithm. As with the Azure algorithm I will now test what samples my algorithm labels as anomalous for different thresholds.

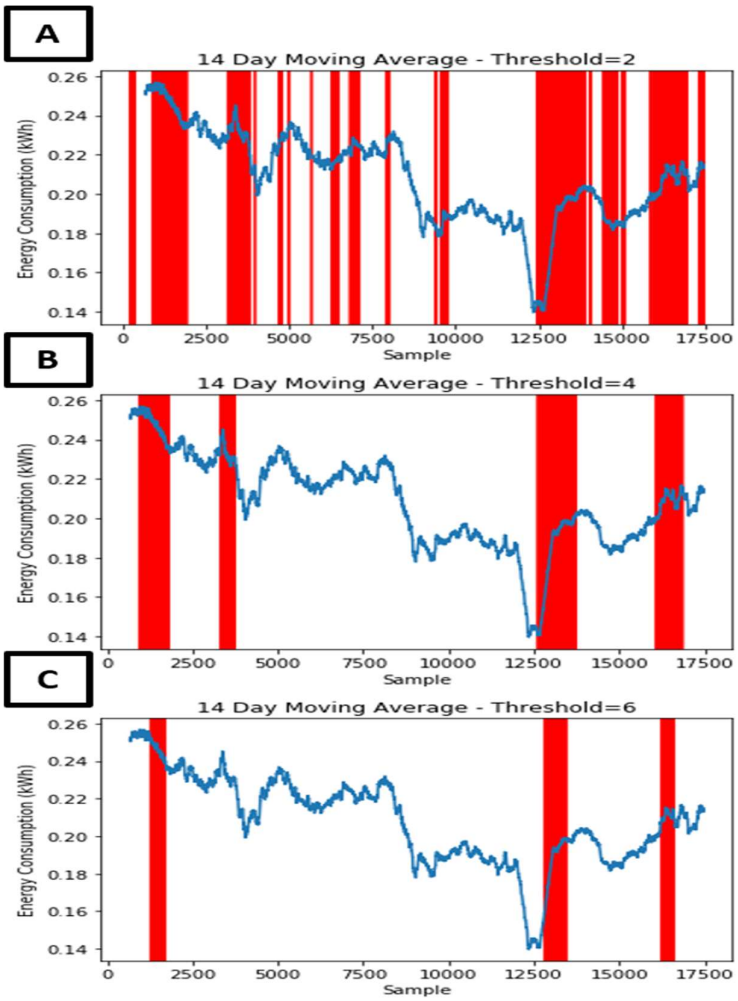

Figure 15: *Plot of the 14-moving average of the energy consumption dataset with the samples detected as anomalous marked in red. Three different threshold values were used for the anomaly detection,* ***a)*** *2,* ***b)*** *4 and* ***c)*** *6*

Lower thresholds resulted in a greater proportion of anomalies detected but also a higher chance of false alarms. In Figure 15a, where the threshold is 2, 28% of the samples are labelled of anomalous. This proportion is most likely too high and contains many false alarms. By increasing the threshold, as is

shown in Figures 15b and 15c, the percentage of anomalies detected significantly decreases to 16% and 6% respectively. This is likely to be more representative of the actual anomalies in the data. However, increasing the threshold comes with a price. You can be more sure that the anomalies detected are actually anomalies, but in the time series data there is a greater delay in detecting the anomaly, since the martingale value takes time to rise over samples instead of jumping straight up to a large value. This is evident in Figure 15, when the threshold is lower, at a value of 4, the anomalous period of low energy consumption is first detected at sample 12607, with a value of 6 it is first detected at sample 12843. The difference in samples corresponds to a time difference of 118 hours, which is just under 5 days. Ultimately, deciding the threshold value will be unique to each problem and will depend on what is more important in that specific problem, having a low false alarm rate or having a low delay time to the detection of anomalies.

Conclusions

Overall, from these experiments I've ran, I can conclude that my algorithm is more successful at detecting anomalous in these datasets. The Microsoft Azure algorithm failed in the full energy consumption dataset to detect the anomalous period of low consumption and failed in detecting the synthetic anomalies I added to the data in the other experiment. Typical "Anomaly Score" values output by this algorithm are below zero, making it impossible to label a sample as anomalous, since the threshold cannot be below zero. An example of where this became a problem was in Figure 8 where there were increases in the "Anomaly Score" for the synthetic anomalies, but the score never reached a value that was high enough to be labelled as an anomaly. Even if the algorithm had been able to detect the anomalies in the data, there are a number of factors that make it a very inflexible algorithm and, therefore, limits its viability in my opinion. The first of these is that only a single feature can be used in the anomaly detection, meaning complex time series data made up of many variables cannot be used in this algorithm. Additionally, there are only three "Strangeness Functions" that can be used, and it is not clear how those methods work. Finally, the length of the sliding window is limited to 1000 samples which I assume is for computational reasons, but it may stop the algorithm from being able to capture longer term trends when calculating strangeness values. Additionally, 1000 samples can have two very different meanings depending on the time series data that is input. If samples were

taken every 12 hours, this would correspond to a period of 500 days, but if samples were taken every 10 minutes, it would correspond to a period of around 7 days.

Further experimentation would be needed to see if it is just this dataset that the Microsoft Azure algorithm struggles with or if it is generally poor at detecting anomalies. I believe it is its choice in "Strangeness Functions" that limits its effectiveness the most as theoretically, without being able to see the implementation behind the Azure algorithm, that is the main difference between the two algorithms.

References

1) https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/time-series-anomaly-detection
2) Fedorova, V., Gammerman, A., Nouretdinov, I., Vovk, V.: Plug-in martingales for testing exchangeability on-line. In: Proceedings of the 29th International Conference on Machine Learning (2012)
3) Vovk, V., Nouretdinov, I., and Gammerman, A. Testing exchangeability on-line. In Proceedings of the 20th International Conference on Machine Learning (ICML 2003), pp. 768–775, 2003.
4) https://data.london.gov.uk/dataset/smartmeter-energy-use-data-in-london-households - UKPN-LCL-smartmeter-sample.csv
5) Volkhonskiy, D., Burnaev, E., Nouretdinov, I., Gammerman, A. and Vovk, V., 2017, May. Inductive conformal martingales for change-point detection. In *Conformal and Probabilistic Prediction and Applications* (pp. 132-153).