

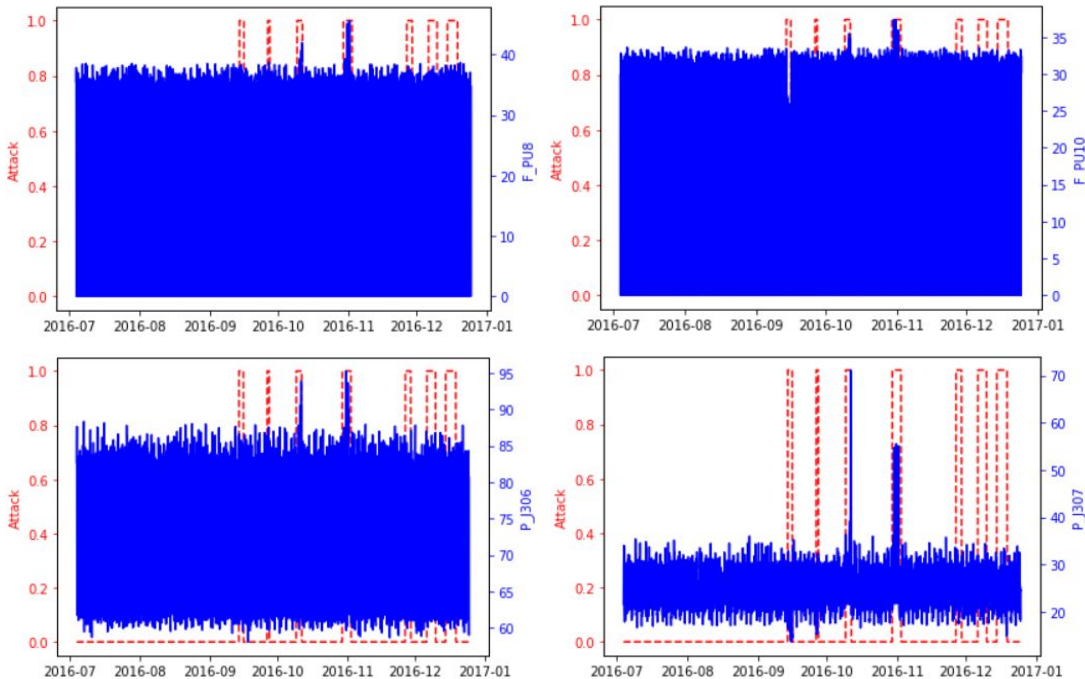
Cyber Data Analytics

Anomaly Detection

Group 2: Cristian Rotari (4985044) | Vahur Varris (2202867)

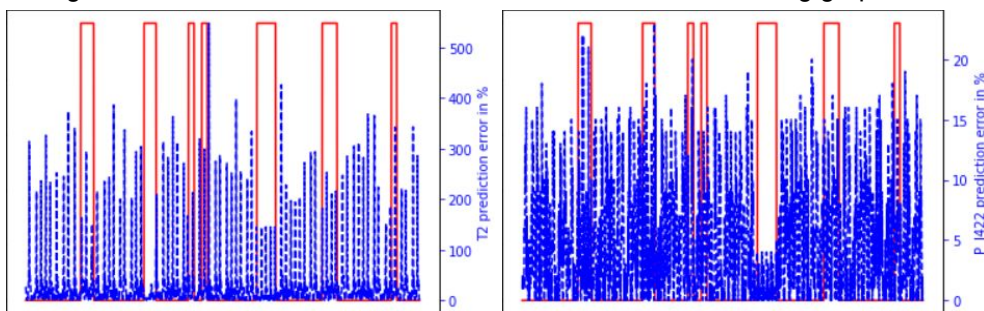
Familiarization

The dataset contains hourly sensor data such as water level in the tanks, the state and flow of the pumps and valves plus the pressure at the junctions. Some of the data is labeled whether it was recorded during an attack or not. Before visualizing the data, it has been pre-processed by dropping 4 columns as they did not contain any information and all the samples were labeled accordingly for verification purposes.



As we can see in the above figures, which have been created on the train data 2, some of the sensor data is highly correlated, e.g. F_PU8 with F_PU10 and P_J306 with P_J307. This can be explained by their location and correlation with other physical elements (e.g. tanks) which were being targeted during the attack. Also, the signals are cyclic, fluctuating around the same mean, making it easy to visually spot the attack.

We have used moving average together with rolling windows of size 5 to predict the next values of some sensors from the test data. To measure the performance of prediction, we plotted the prediction error calculated as percentages of the actual value and obtained results like in the following graphs.

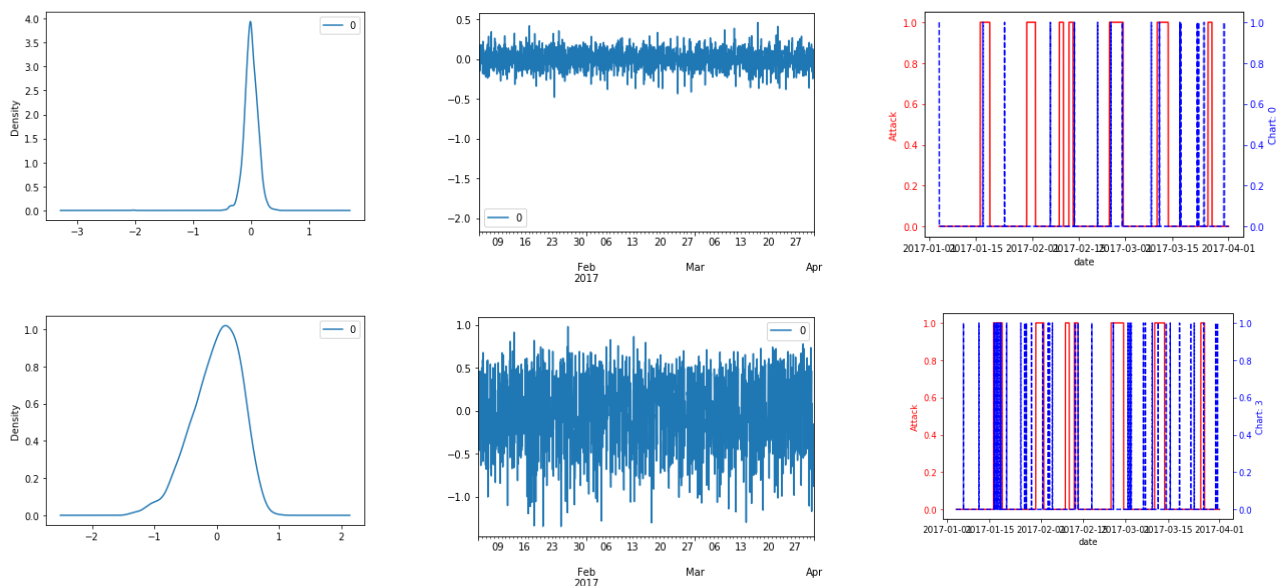


The average T2 prediction error is 33% while for P_J422 is 5%. Even though some of the results are statistically good, it is not suitable for anomaly detection as the high number of outliers makes it impossible to set a threshold which would raise the alarm only in attack cases (marked with red).

By completing this task we have determined that we deal with contextual type of anomalies and identified a set of sensors that can potentially contain useful data for the detection of attacks. We have also decided to work with each sensor individually because attacks affect a certain part of the system, therefore only several sensors at a time. Moreover, if we can detect anomalies for each individual sensor, doing so for the whole system is a trivial task.

ARMA

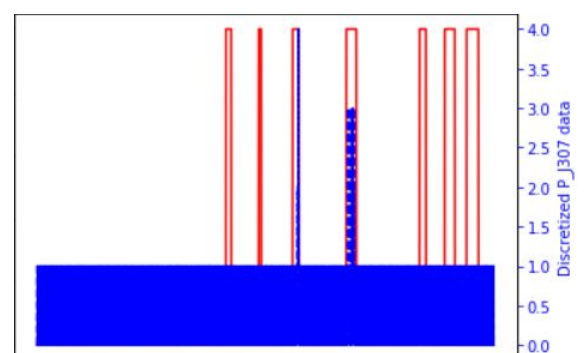
The goal of the task is to learn the ARMA model with best params, and then find the thresholds for the anomalous values. We used tank levels as our sensor data and took ACF and PACF plots (graphs available in the ARMA.py notebook) to determine the orders for AR(p) and MA(q). After that we picked the best values based on AIC and mean squared error. After that we found the residuals for the trained data, and computed the detection threshold. We plotted the probability distributions of the residuals and as they were close to normal distributions we found the best threshold to be 2.5 times the standard deviation from the mean.



The first row above is about tank 1 and the second is tank 4. We can see that both residuals follow a normal distribution, but the std is much larger for the latter one. Also, the last figure illustrates the flagged anomalies with actual attacks; both of them have managed to capture some of the attacks, but have also quite a lot of false positives. The anomalies that are found are contextual, since they depend on the context the model was trained and the thresholds set.

Discrete model

As the sensor data has various and chaotic fluctuations, we decided to use binning discretization, which turns numerical values into categorical counterparts. This has the advantage of immunity to noise and fluctuations, while still being able to successfully detect outliers. As most of the normal data does not exceed certain thresholds, the only challenge is to find the optimal number of bins that would categorize the benign and attack points.

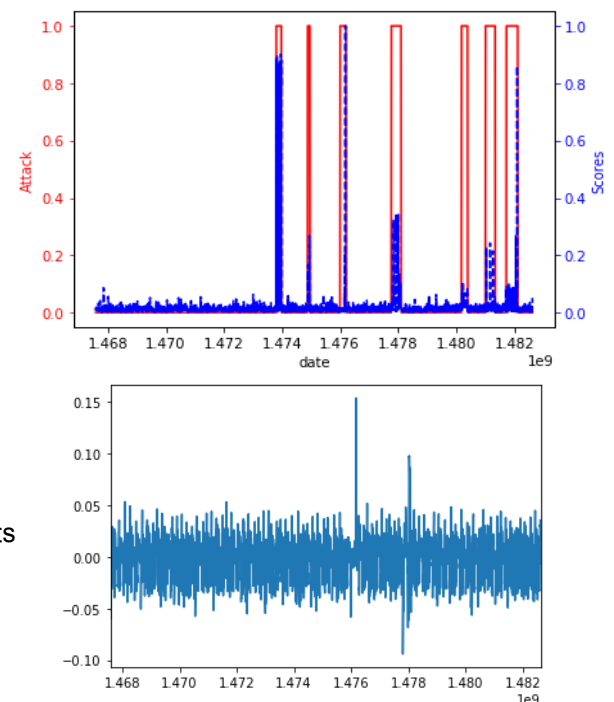


differently. We have discovered experimentally that usually 4 bins results in a good separation. For the P_J307 sensor shown above, the discrete data looks as follows:

We have applied N-grams to the sliding windows on the 2 merged datasets and tested it on the testing one. For each individual sensor, the number of bins, size of the sliding window and the probability threshold have were set for the best performance. These parameters have been tuned in order to obtain the least amount of false positives and most amount of attack regions detected. The results for the sensors that could be best modeled can be seen in Appendix. As can be seen, the model is able to detect rare combinations of states that either have not occurred before or are in minority, which makes them suitable for context anomalies detection. By combining the models for each sensor, it is possible to detect all but last attack regions.

PCA

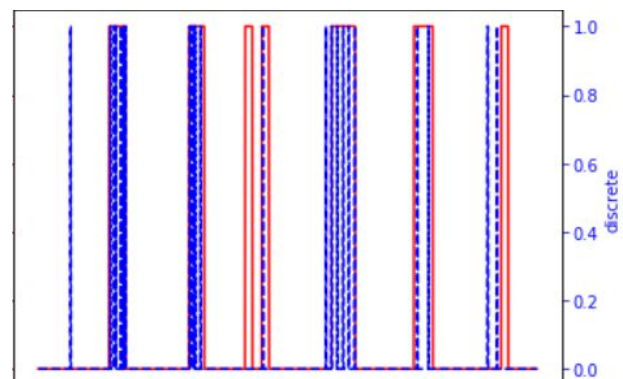
To perform the Principal Component Analysis we need to determine the n-components of the model. In order to do this we computed the eigenvalues and plotted their significance. From the variance we determined that best n_value would be 10. As a anomaly score we used the sum of the squared differences by the max-min range of the sum of the squared differences for the entire dataset¹. Then we needed to determine the anomaly threshold for the detection. In the figure on the upper right, there is an attack data plotted together with anomalies, we can set the threshold 0.1 and then detect all the attacks with very low amount of false positives. When we plot the data of residuals, nr 10 as an example, we see some abnormalities but they correlate with attacks, so we don't need to remove anything from the dataset. Detected anomalies can be classified as contextual, as it detects abnormal behaviour based on divergence from average.



Comparison

For anomaly detection case, the most appropriate metric in our opinion would be counting a true positive if at least one anomaly is found in an anomalous region. Moreover, it has to be combined with a low number of false positives as action has to be taken for each alarm risen. Even better is when the alarm is risen at the beginning of the anomalous region as it reduces the attack impact on the system and gives more time for taking measures.

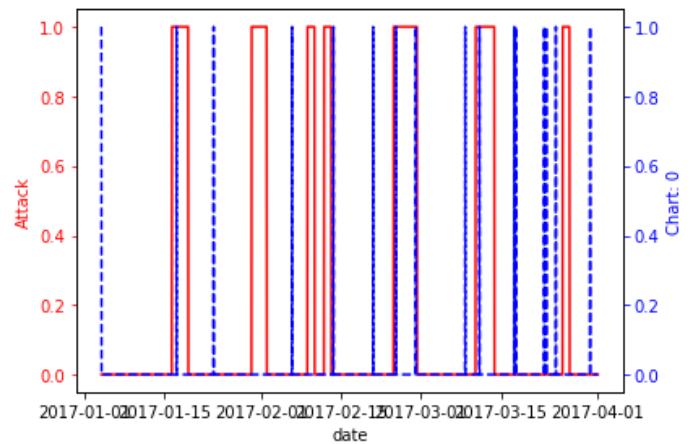
The performance of the discrete model has been achieved by combining the individual results of the modeled sensors. As can be seen from the figure to the right, out of 7 regions, it manages to successfully detect 5, with a total number of 3 false positives. The



¹ <https://www.oreilly.com/library/view/hands-on-unsupervised-learning/9781492035633/ch04.html>

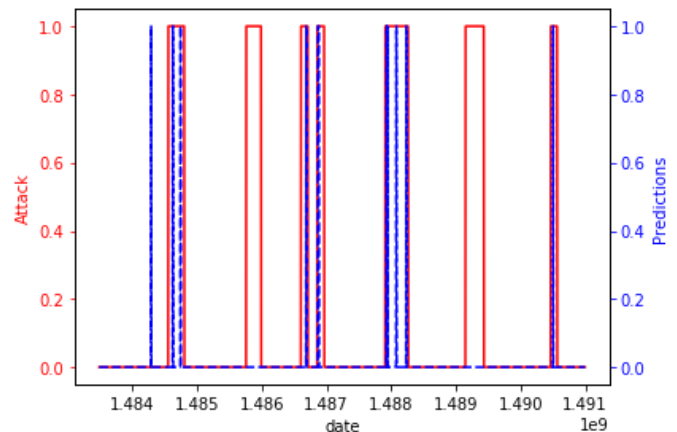
performance is quite impressive, taking into account how simple and computationally inexpensive is the model.

The ARMA model is mathematically quite simple and flexible, with few parameters, it is possible to configure many ways, to include seasonality for example. However, finding the right parameters can be tricky, since the grid search is quite time consuming. Also, one-step training (fit new model on every $n+1$ iteration) is not feasible in practice with even medium datasets, since it results with exponential time complexity. Also, the API of the statsmodels library is quite raw, and had many issues, so maybe some other tools such as R has better support for it. In terms of performance, the model on the right had params $(p,q)=(4,12)$. We can see that it detects 4 attack regions, but also has 9 false positives.



The PCA model detects 5 attack regions while having only one false positive. The PCA uses linear algebra and matrix multiplication to work so it is really fast, furthermore, as PCA is well-known technique in statistics, there is a very good support from various tools and lot's of learning material available.

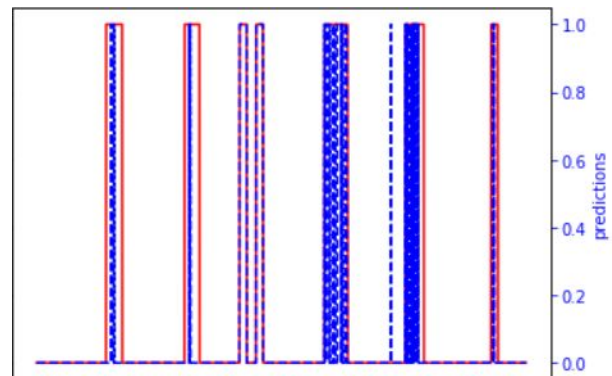
Both discrete model and PCA detected 5 regions, however the latter one didn't manage to detect a 2 longer attacks, while the first one didn't detect shorter ones. Also, the discrete model had more prediction points during the attacks. The both mentioned techniques are simple and computationally fast. The ARMA model does not perform very well, while it manages to detect only one attack less than the others, it has too many false positives, therefore the recommended model is the discrete one.



Deep Learning

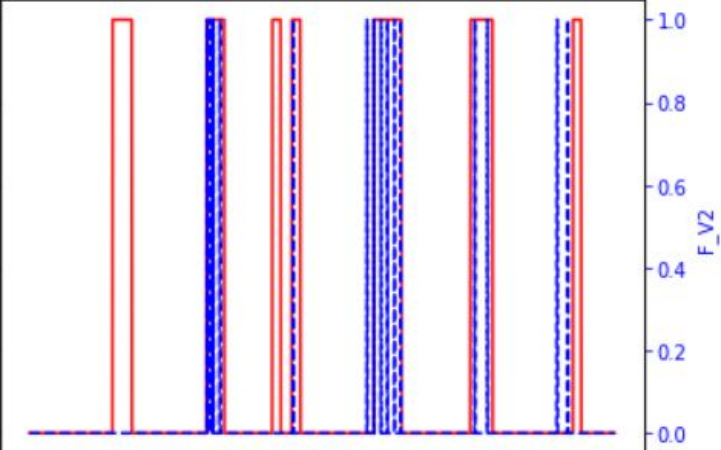
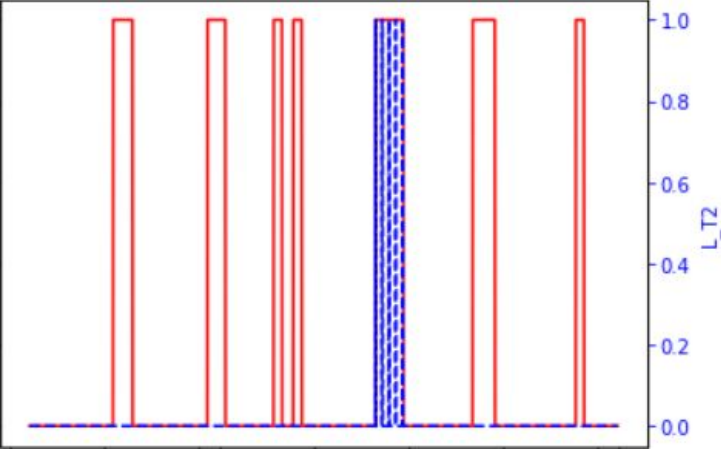
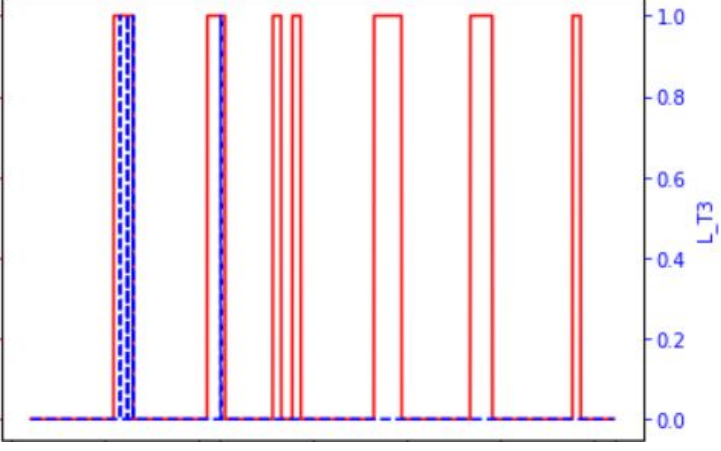
Following the given template in class, we have used deep neural networks to try to classify the points in time when attacks were occurring. As in all previous steps, we have used the fully labeled training data for learning purposes and testing labels for verification purposes and 20% of the training data has been used for validation before the testing phase. All the data has been scaled to $[0,1]$ range. We have increased the learning rate to 0.1 from 0.001 as we wanted the model to learn more from the given attack examples. This resulted in a large number of false positives, which in the scope on anomaly detection is a bad thing as

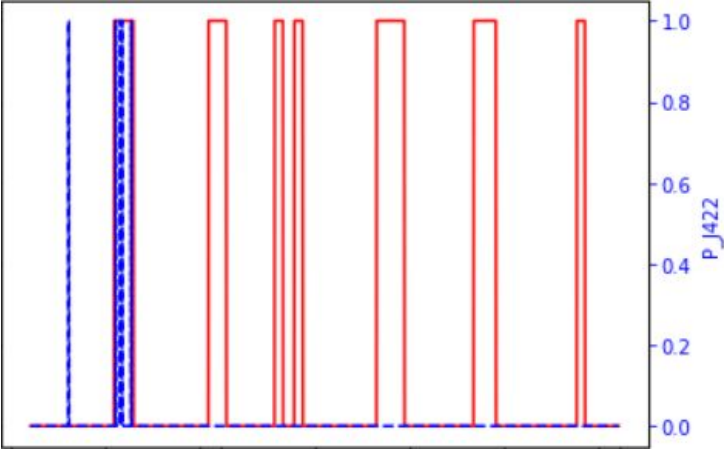
someone needs to take actions upon each alarm. By raising the detection threshold to 1.9 from 1.5, we were able to detect a good part of the attack points having a minimal number of false positives. As can be seen in the figure, each attack region has at least several true positives, specifically at the beginning of the attack, making it suitable in the given context. Point-wise, the model has 3 false positives, 150 true positives and 250 false negatives. The high number of false negatives does not represent a concern, especially when they are located immediately after a true positive. Even though the obtained performance is impressive, the disadvantage of this approach is that the training data has to be of good quality, labeled and accessible before the system is put in place. If the normal behaviour of the system does not support major changes in time, deep learning is an ideal candidate for anomaly detection, otherwise the model has to be retrained when new data is available.



Appendix

Discrete model task results:

Sensor Name	#bins	Window size	Threshold	Result
F_V2	4	4	0.00002	 <p>The plot for F_V2 shows two data series: a red solid line and a blue dashed line. Both series exhibit several sharp, narrow peaks reaching a value of 1.0. The peaks are distributed across the horizontal axis, with some overlapping between the two series. The y-axis is labeled 'F_V2' and ranges from 0.0 to 1.0.</p>
L_T2	4	3	0.0005	 <p>The plot for L_T2 shows two data series: a red solid line and a blue dashed line. The red solid line has several sharp peaks reaching 1.0, while the blue dashed line has a single, wider peak also reaching 1.0. The y-axis is labeled 'L_T2' and ranges from 0.0 to 1.0.</p>
L_T3	3	3	0.0025	 <p>The plot for L_T3 shows two data series: a red solid line and a blue dashed line. The red solid line has several sharp peaks reaching 1.0, while the blue dashed line has a single, wider peak also reaching 1.0. The y-axis is labeled 'L_T3' and ranges from 0.0 to 1.0.</p>

P_J422	3	3	0.0025	
--------	---	---	--------	--