CDA Lab 2: Intrusion detection in SCADA systems

Cas Bilstra 4381084
Roy Graafmans 4299442

https://github.com/Roy-Graafmans/CDA_CS4035

June 3, 2019

Please read the readme contained in the deliverable for instructions on how to run the code via Jupyter Notebook. Please zoom in if a figure is difficult to read, the are embedded with high resolution. Please do no run the complete notebook, it will take you VERY long because of an extensive ARMA prediction. Everything but ARMA can be run very fast, so please try that! The code for all tasks except the bonus is in main.ipynb. When training set is written, BATADAL dataset 3 is meant. When training set 2 is written, BATADAL dataset 4 is meant. When test set is written, the BATADAL test set is meant.

1 Familiarization

In the data, a column ATT_FLAG might be present indicating normal state (0), that the system is under attack (1) or that the state is unknown (-999). Furthermore, there are six types of signals in the dataset (with x being the component identifier):

- the water level in a tanks (L_Tx, float)
- flow through a pump (F_PUx, float)
- the status of a pump (S_PUx, 0(off)/1(on))
- the flow through a valve (F_Vx, float)
- the status of a valve (S_Vx, 0(closed)/1(open))
- the pressure in a junction (P_Jx, float)

Some of the signals are clearly correlated. The highest correlated sensors are showed in table 1. The pair S_PU4, P_J256 is visualised in figure 1, which shows the cyclic behaviour of the data as well. Only the first 500 data points are used to show this cyclic behaviour, as otherwise the plot would not be readable due to too many data points.

x	y	correlation
P_J306	S_PU8	0.989426
P_J269	S_PU2	0.962877
S_PU7	P_J415	0.960707
S_PU4	P_J256	0.958924
P_J317	S_PU10	0.861843
S_PU8	P_J307	-0.820805
S_PU8	P_J302	-0.823764
S_V2	P_J14	-0.919642
F_PU1	S_PU2	-0.961986
P_J280	S_PU2	-0.986258

Table 1: Correlation between sensors

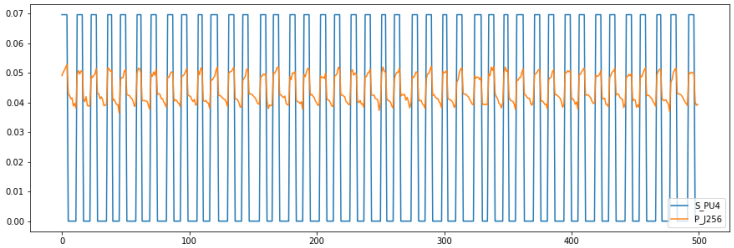


Figure 1: Cyclic behaviour for and correlation between S_PU4 and P_J256

For predicting the next value in a series, the Auto Regressive (AR) model was used. In figure 2, the prediction for S_PU2 can be seen. The Mean Squared Error for this prediction is 0.036. 80 % of dataset 3 was used for training the model, 20% for testing it. For this sensor, prediction is thus not very hard, with a relative small error. But this is not the case for F_PU7 for example. There is no space left to show this, but the MSE is 288.167 for that sensor, with the real values ranging from 0 to 50. In the python notebook, try changing signal = 'S_PU2' to signal = 'F_PU7' to visualise this. Useful for detecting anomalies is that F_PU3 is always 0 in the training data. But maybe not in the test data provided?

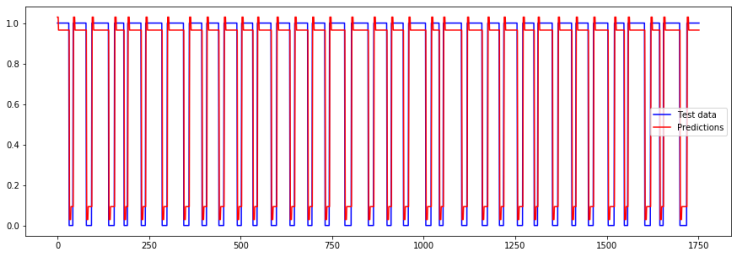


Figure 2: Predicting S_PU2 with the Auto Regressive model

2 ARMA

Akaike's Information Criterion was used to determine the parameters of the ARMA model. For every feature, an extensive search for p in range [1, 9> and j in range[0, 6> was performed on the training data to determine the best parameters (for the training dataset without attacks, and training set 2). Finding the parameters within this given range took already a few hours. There is a chance that even better parameters can be found if this range is extended further, but this requires days to run the code. The results can be seen in the Jupyter Notebook under 'Parameter estimation'. The threshold was determined as 1.5 * the average of the highest 5% of the residuals. This makes sure a value is only identified if it is very abnormal that it occurs. Under the heading 'Arma classification', the results from performing arma can be seen. ARMA was trained on all sensors, but looking at the results from testing on training set 2 (test_data = data2 under the Arma Classification header in the notebook), mainly the P_Jx sensors can be modeled effectively. The L_Tx sensors also detect some attacks, but a lot less than the P_Jx sensors. For the rest of the sensors, almost no attacks are detected. An example of the residual error, predictions, etc for P_J300 can be found in figure 3. An attack is signalled when the residual error is too big (when the actual data does not follow the prediction well). To see the raw ARMA results in the notebook, and not just the prediction, uncomment the appropriate lines (around line 61 under the ARMA Classification header, the comments guide which ones to uncomment). ARMA predicts future values based on past values. The anomalies ARMA can detect are those that deviate too much from what would be expected based on previous values.

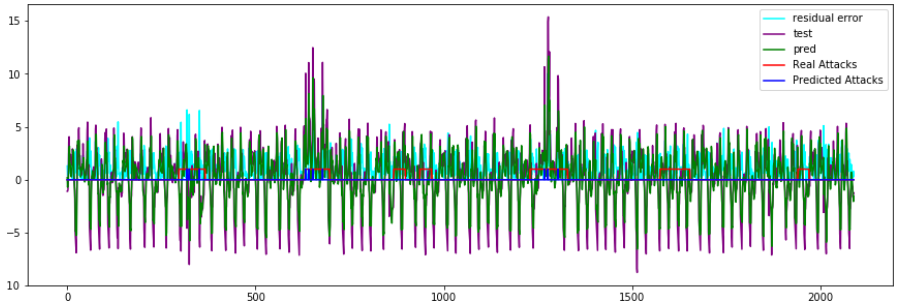


Figure 3: ARMA for P_J300

3 Discrete models

Symbolic Aggregate approXimation (SAX) was chosen as the discretization method. This method keeps the most important information from sensors (how high/low waterlevels are, if the sensor is off or on, if the pressure in a junction is high or low, etc.) while discarding the less meaningful information of the exact values of the sensors. By doing this, this approach greatly simplifies training n-grams or sequence alignment, as there are way less possible values the signal can take. It is important though to not only have the knowledge if a sensor is high or low, but we want to know as well if the value of a sensor is very high or very low. That is why an alphabet size of 5 was chosen for SAX, which can be interpreted as: very low, low, neither low nor high, high, very high. Before performing SAX, the data was scaled using a standard scaler (which can be described as z = (x - u) / s, where z is the scaled element, x is an element to be scaled, u is the mean of the training samples and s is the standard deviation of the training samples). Only the training data was used to calculate u and s, as otherwise knowledge would be used (from the test set) which we do not posses. The test data was scaled using the parameters derived from the training data. The discretization for test and training set can be seen in figure 4. For both the training and test data, the SAX representation was calculated per 10 hours. In the figure, only the first 500 hours are shown, as the figure would otherwise be difficult to read.

N-grams were used to find anomalies. A sliding window of size 5 was used, meaning the last 5 hours are taken into account in the sliding window. Smoothing was applied to make sure an unknown sequence is not immediately recognised as an attack. Only if the probability that the sequence ends with a certain element is sufficiently small (because the sequence was observed in the training data often with other last elements), the last element is marked as anomalous. This threshold was put at 0.01 by manual inspection of the detected attacks and False Positives from evaluation of dataset 4, making sure a good ratio between them. This threshold means that only if the probability that some sequence occurs is less than one percent, the last element of that sequence is set to anomalous. The anomalies which can now be detected are those sequences that end with an element which is different from the last elements for that sequence in the training set. In order to evaluate the discretization, it was evaluated how many attacks were detected (in the case of the test set maximal 7), at how many false positives (an attack signalled when there is none happening). To see which sensors can be modeled effectively, please have a look at the output in the jupyter notebook. Modeling the L_Tx sensors is quite effective, though L_T4 and L_T5 produce a lot of false positives. The F_PUx sensors in general do not produce a lot of false positives for the amount of anomalies they detect. F_PU3 can be modeled very effectively, while F_PU10 performs worse. The P_Jx sensors produce comparatively many false positives. For the final classifier, post processing was used, only labeling an attack if at least n=3 sensors confirm it, to remove some false positives. This classifier is visualised, together with some information about the amount of attacks detected and the False Positives, in the jupyter notebook under the heading 'Classification' when show_single_features is set to False. This classifier detects 4/7 attacks with only 4 false positives.

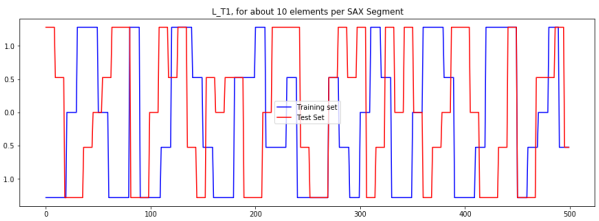
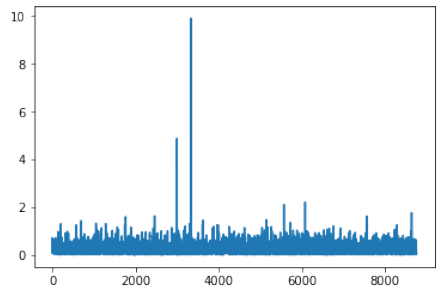


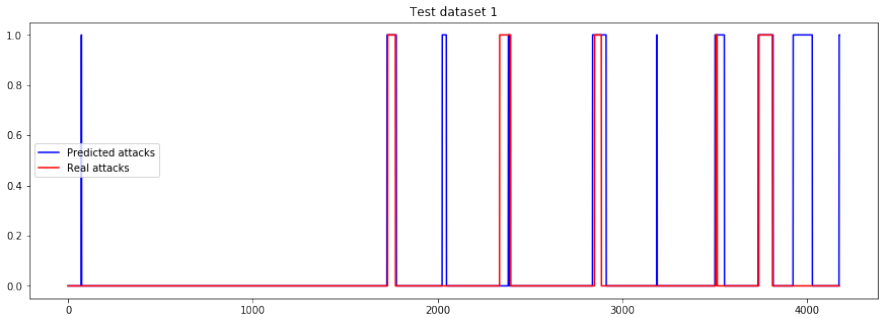
Figure 4: Discretization of L_T1

4 PCA

The residuals after scaling the training data and performing PCA with the amount of components that can explain at least 99% of the data can be seen in figure 5 (a), which makes sure that most data can be described by the components but not all to reduce the chance for overfitting. It can be seen that some of the data cannot be described well by the amount of components chosen. These may occur because of for example measurement errors or a very low amount of non-zero data points for a feature. The approach was chosen to remove abnormalities in the training data by looking at the scaled data, and removing the rows that do not fall within the interval of 4 standard-deviations from the mean of the data (the data that does not belong to for about the 99.99% closest datapoints to the mean when the data in normally distributed). After this, the ammount of components necessary to explain more than 99% of the data is determined, and used for the PCA. After performing the PCA, all data points above the threshold were classified as anomalous. The threshold was first set to $\mu + 4\sigma$, which means that 99.99% of the data would be normal (when a normal distribution is used), but the threshold 3 was found to work a lot better on training set 2 and thus used. After that, a grouping function was used to group the attacks that appear close to each other as one attack. The result is quite good, as can be seen in section 5 and figure 5 (b), which shows the predicted attacks for training set 2. It even detects the attacks which are going on, but which are not labeled (the big detection on the right and the third detection). It can detect all attacks with only 3 False Positives. The residuals can be plotted as wel, changing the PLOT_RESIDUALS boolean to True in the code. We can detect all anomalies which can not be described well by a representation which fits most of the training data (without its outliers). In this case, we can detect all attacks.



(a) Residuals without removing abnormal training data



(b) Predicted attacks for training set 2

Figure 5: PCA Plots

5 Comparison

The comparison method we created is taking into account the amount of False Positives detected, combined with the fraction of an attack that is detected. The fraction of an attack that is detected is defined as follows: When we define the 'detected length of an attack' as the amount of hours between the first detected attack signal and the end of the attack, the 'fraction of an attack that is detected' is defined as the 'detected length of an attack' divided by the total amount of hours an attack lasted. This takes thus into account that there may not be too many False Positives, as that is both annoying and expensive for the people operating the SCADA system. Also it takes into account if an attack is detected and how fast an attack is detected. This because of the obvious reason that an undetected attack, or an attack that is detected really late, is not worth much to the SCADA operators. For both ARMA and discretization methods, the signals for all sensors were combined, as an attack can in real life happen on each sensor (or part/function/machine of the SCADA system). PCA already works on all sensors combined. It should be noted that the total amount of attacks which can be detected is 7. The results are as follows: PCA is

Method	**False Positives**	**Detected Attacks**
ARMA	9	5.02
Discretization	4	4.00
PCA	2	6.62
Combination	12	6.68

clearly the best performing method, with a detection rate of 6.62/7, whilst having only two false positives. The next best is a combination of PCA, ARMA and Discretization, detecting a tiny bit more than only PCA, but at the cost of 12 false positives. ARMA clearly outperforms discretization on the amount of detected attacks, but at the cost of more False Positives. In general, the PCA method is recommended, as it find almost all attacks, with only two False Positives, making sure the people at the SCADA system should only respond when necessary. The detected attacks per method are visualised in figure 6. The prediction of chosen method (PCA) is visualised in figure 7.

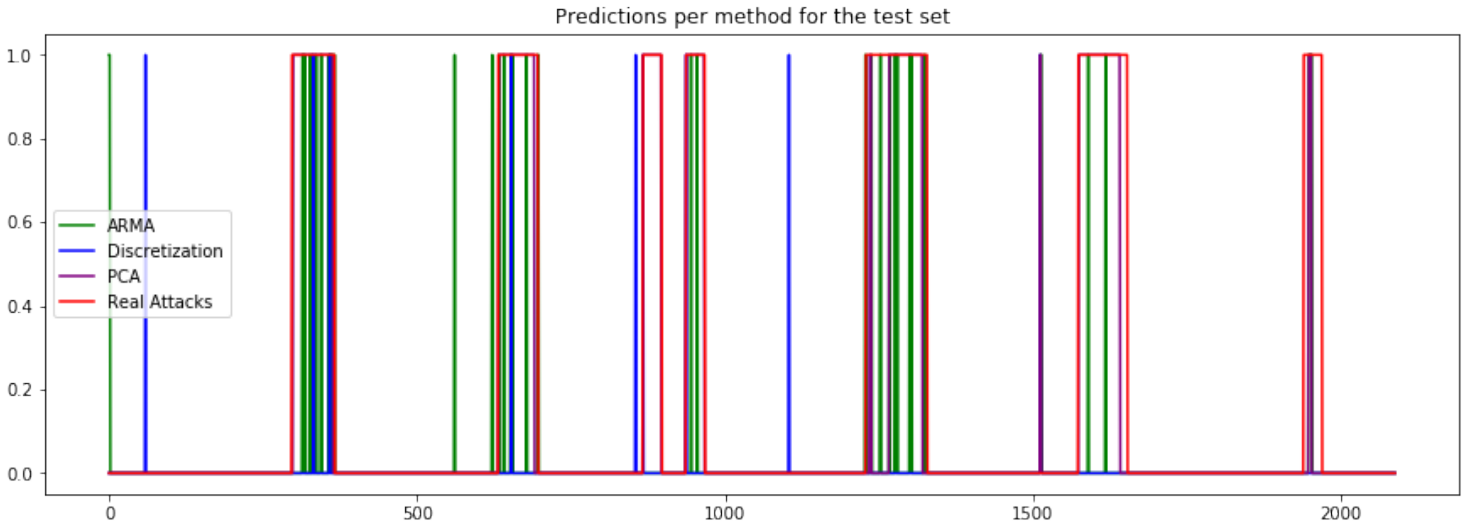


Figure 6: Predicted attacks per method on the test set

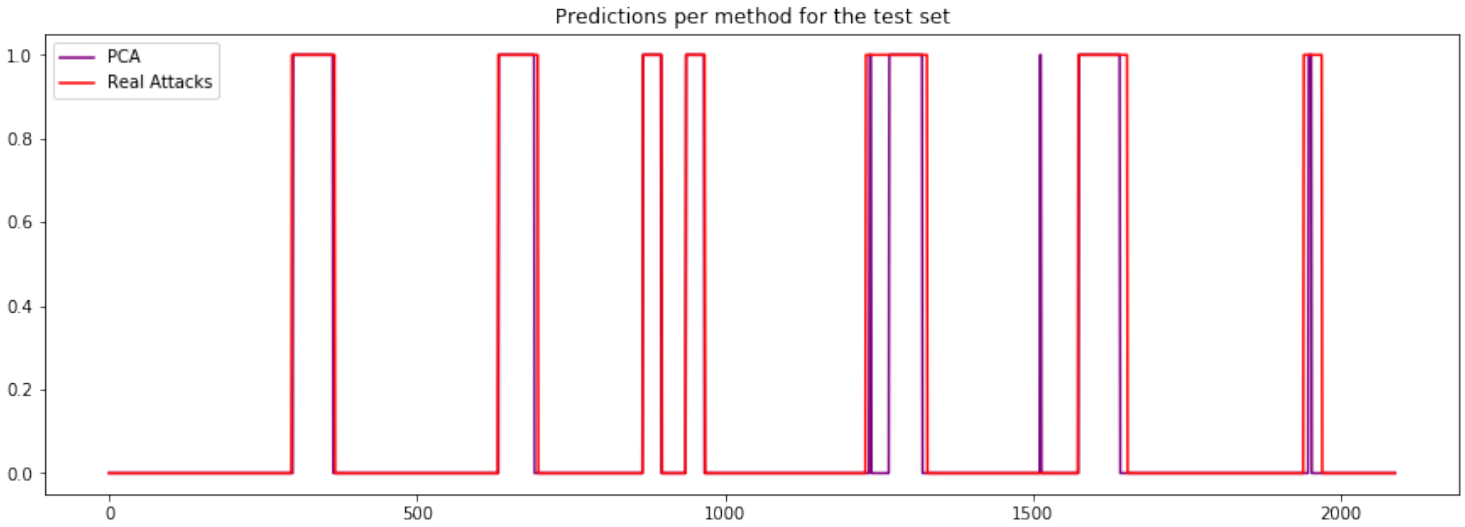


Figure 7: The results of the PCA method

6 Deep Neural Networks

The sample script provided in class, using Neural Networks to train Autoencoders, was adjusted and tested with different learning rates and thresholds on the validation set used in the script. The results can be seen in table 2. After inspection of this table, 0.005 was chosen as the learning rate and 2 as the threshold, because this combination leads to not too many FP or FN cases, while still detecting a fair amount of the TP cases. Using these parameters, results for the test set were predicted, the result of which can be seen in figure 8. When performing the same metric as in the comparison chapter, between 4 and 4.94 attacks are detected (depending on randomness) with 1 False Positive. This is better than ARMA (because similar amount of attacks detected with less False Positives) and Discretization (less FP and more attacks detected), but worse than PCA (PCA detects more attacks with only one more FP).

Learning Rate	Threshold	FN	FP	TP
0.01	1	22	191	20
0.01	1.5	29	8	13
0.01	2	35	3	7
0.01	2.5	35	3	7
0.01	3	36	2	6
0.005	1	22	191	20
0.005	1.5	24	17	18
0.005	2	25	8	17
0.005	2.5	25	8	17
0.005	3	25	8	17
0.001	1	24	93	18
0.001	1.5	33	3	9
0.001	2	36	3	6
0.001	2.5	36	2	6
0.001	3	36	2	6
0.0001	1	24	71	18
0.0001	1.5	35	3	7
0.0001	2	36	2	6
0.0001	2.5	36	2	6
0.0001	3	36	2	6

Table 2: Results on the validation set for different types of parameters

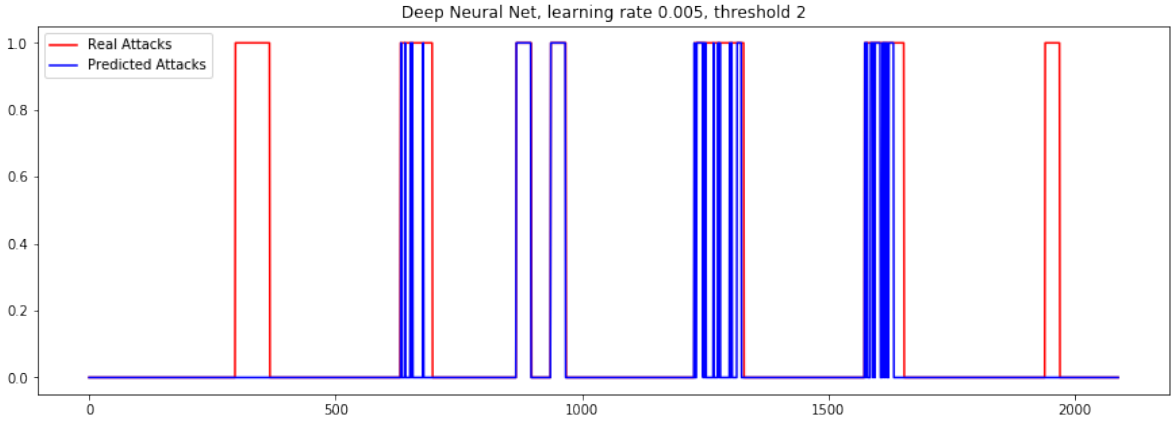


Figure 8: Prediction by the Neural Network