Cyber Data Analytics: Anomaly Detection

K. Wendel - 4331362
M. Post - 4286146

June 4, 2019

1 Familiarization

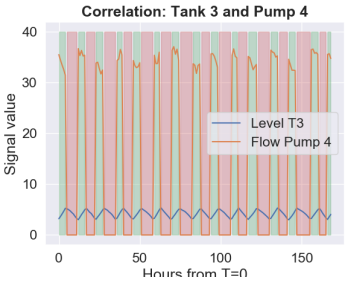
The BATADAL sensor dataset provides three different datasets: a training set without anomalies, a training set with partial labeled anomalies, and a testing set that has anonalies but no labels. This section will analyse the different kind of sensor signals and attacks that exist in these sets.

1.1 Signals

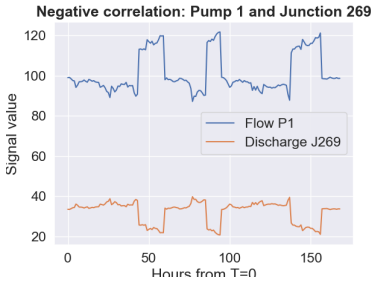
The BATADAL datasets consist of hourly sensor data of the main distribution water system of C-Town. This distribution system is monitored and controlled by a SCADA system. The sensors of the water system are placed at water tanks, pumps, valves and junctions in C-Town network. Firstly, the sensor data of the water tanks indicate the water level in meters in each of the tanks t_i. These measurements of these sensors are indicated with l_t_i with $i = 1, .., 7$. Secondly each pump and valve in the C-Town network has two different sensorsm, namely a on-off status (indicated with s_id) and a flow measurement sensor (indicated with f_id). The identifiers can be either pu_i with $i = 1, .., 11$ for pumps or v_i with $i = 1, .., 2$ for valves. Finally, each junction j_i contains a sensor that measures the pressure at that junction. For the placement of each sensor in the network we refer to the official BATADAL site [1]. Cyclic behaviour is extremely common in SCADA systems as all visualisations in Figure 1 show.

1.2 Correlation

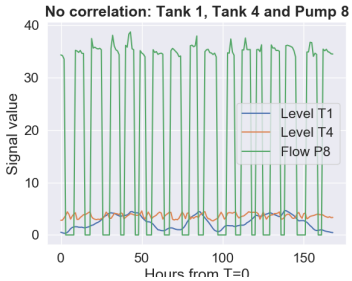
Next we analyse some of the signals for a timeperiod of a week and visualise the different kinds of correlation that exist in the dataset. Correlation is measured with the Pearson's correlation coefficient ρ. The first visualisation in Figure 1a shows the level of t_3 and the flow and switch sensor of pu_4. As expected, the switch signal is almost perfectly correlated with the flow signal with $\rho = 0.99$, while the level of the tank is $\rho = -0.17$ with the flow signal. In Figure 1b show that negative correlation also occurs with $\rho = -1$, and this can be explained by the fact that both these sensors are close to each other. Finally in Figure 1c, we see three sensors that are not close to each other and have a insignificant correlation.



(a) Sensor data tank 3 and pump 4, the red and green background indicate if the switch sensor is either 0 or 1 respectively.



(b) Sensor data of pump 1 and junction 269 that are negatively correlated.



(c) Sensor data of tank 1, tank 4 and pump 6 that have an insignificant correlation.

Figure 1: Visualisation of different sensors for the first week until $T = 168$

1.3 Predicting

A sliding window approach is used to predict the next value in a series. Two different methods were used for predicting the next value: persistence prediction and linear regression.

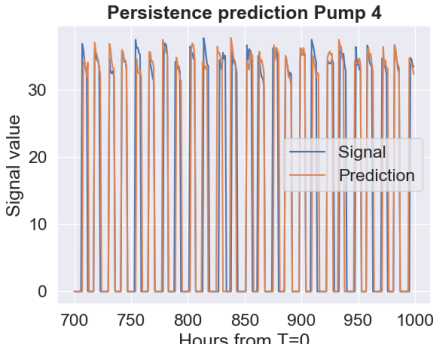
With persistence prediction the model returns the value that was seen p timesteps before. The value of p was determined by plotting the root mean squared error (RMSE) of the predictions against different values of p, where p with the lowest RMSE gives the best model. The persistence model was used for the flow sensor in pump 4 with $p = 24$ and the predictions are shown in Figure 2a. This shows that good prediction can be achieved by returning the sensor value of the previous day.

Next a linear regression model was trained to predict the flow of pump 4, based on the switch sensor of pump 4 and the level in tank 3. The model uses all values in the sliding window to train the regressor

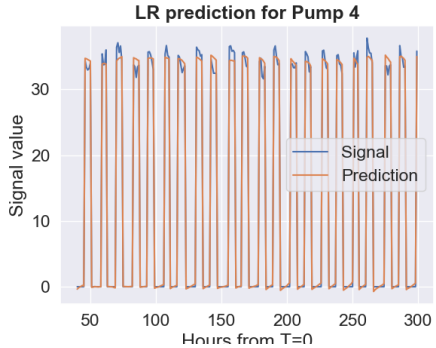
[1] `https://batadal.net/challenge.html`

and then predicts the flow value in the next time step. The optimal sliding window size w was again determined by computing the RMSE of the predictions for different values of w, where the w with the lowest RMSE is optimal. The predictions of the LR model are shown in Figure 2b and show that prediction the next value in the series can be achieved with a low RMSE.

For visualisation for the RMSE plots, we refer to the provided Juypter notebook `Predictions.ipynb`.



(a) Persistence prediction with $p = 24$. The test data ranges from T=700 to T=1000. The RMSE of the model is 11.699.



(b) Linear regression with a sliding window of $w = 40$. The test data ranges to T=300. The RMSE of the model is 0.833.

Figure 2: Predicting the next value in a series with a persistence model (left) and linear regression on a sliding window (right).

1.4 Attacks

Finally, the known attacks in the training dataset with anomalies were also visualised but are not included in this report due to the page limitations. We refer to the notebook `Attacks.ipynb` for the interested reader.

2 ARMA

To check which sensors could possibly be modeled effectively using an autoregressive moving average (ARMA) model we have taken a look at the plots of the sensors. We looked for sensors that showed no sudden relatively big changes in their values since these are hard to predict using an ARMA model. We chose to apply ARMA models on the signals that are mentioned [2] in the attacks in the Batadal training dataset 2, since these should be showing impact of the attacks. We analysed the autocorrelation (AC) and partial autocorrelation (PAC) plots of the values of these sensors (see Figure 6). To determine the parameters of the ARMA models for these signals we have used the Box-Jenkins method [1]. To check if the parameters should be tweaked a bit more we check if there is correlation in the residual errors of the model using the Durbin-Watson statistic (DW) and compare the Akaike's Information Criterion (AIC). The DW value is always between 0 and 4 and suggests that there is no correlation if the value is around 2, which is what we want for the residual errors.

For the *l_t1* sensor we can see an initially declining signal in the AC plot and in the PAC plot we can see that the first 3 values are significant, so we start with an *ARMA(3,0)* model. Since the DW value of 1.92899 did not indicate any autocorrelation in the residual errors and the distribution of the errors is normal this is probably a good model. Increasing the parameter values also did not result in a significantly better AIC value.

The parameters for *l_t4*, *l_t7*, *f_pu1*, *f_pu7*, *f_pu10* are determined in a similar way which resulted in *ARMA(3,1)*, *ARMA(2,1)*, *ARMA(2,2)*, *ARMA(3,1)* and *ARMA(2,1)* respectively.

[2] `https://batadal.net/images/Attacks_TrainingDataset2.png`

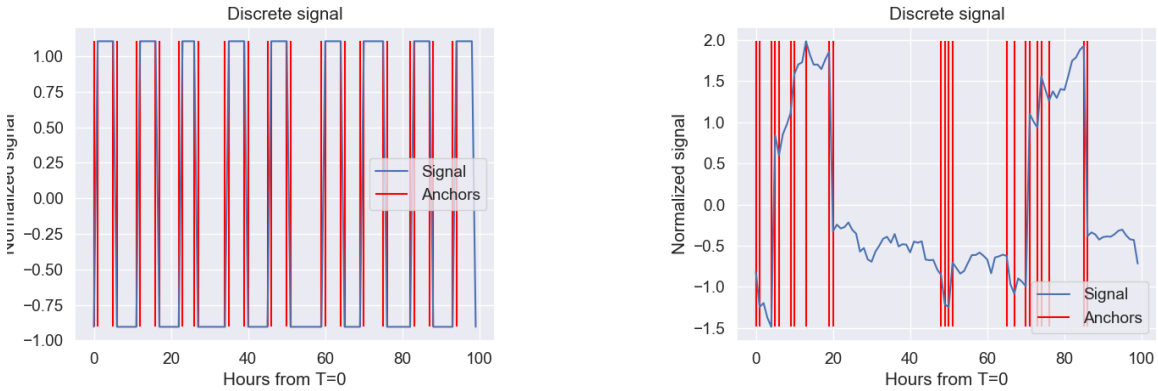
2.1 Detection

The ARMA model worked pretty good on the *l_t1* sensor. We got 8 false positives on the *BATADAL_training2.csv* set by training on the last 1000 hours for every predicted hour, starting with predicting at 2016-09-01 00h. It detects the start and end of the 3th (overflow in T1) and 4th (overflow in T1, but concealed with replay attack on *l_t1*) attack (respectively 2016-10-09 09h till 2016-10-11 20h and 2016-10-29 19h till 2016-11-02 16h) perfectly. Although, it is not perfect on detecting an ongoing attack, i.e., it has a lot of false negatives in between the start and end of the attacks.

Actually the only sensor on which the ARMA model worked well is on the *l_t1* sensor. But by combining the results of the different sensor models the results were OK.

3 Discrete models

The first step in training a discrete model for anomaly detection is making the sensor data discrete. For this task, the SWIDE algorithm is used from [3] that maps the data into different linear segments. These segments are then discretized, based on the gradient of the segment, into one of the five classes: constant (SC), slow up (SU), slow down (SD), quick up (QU) or quick down (QD). The result of SWIDE is visualised in Figure 3a and Figure 3b. These visualisations show that the SWIDE algorithm is best for signal that have certain states and switch between those, such as the switch sensor that can be either in state on (1) or off (0). The SWIDE algorithm is less efficient on more difficult sensor such as the flow sensor.



(a) Discrete signal of the switch sensor of pump 4.

(b) Discrete signal of the flow sensor of pump 1.

Figure 3: Visualisations of the discrete signal data for two sensors. Each segment between the red anchor lines is mapped to discrete class based on the gradient of that segment.

Next, this discrete signal is used count the occurrences of each occurring N-gram, where N is a parameter that influences the size of these grams. Laplace smoothing is then applied to these counts. Now to detect anomalies, the given dataset needs to be made discrete and then for each N-gram it needs to be checked if the N-gram count on the training dataset is above the threshold. A discrete model was trained on the l_t_3 sensor, with $n = 3$ and a threshold of 20. On the second dataset it was able to partial detect attack 3, 4 and 5 on that sensor, for a total of 123 true positives. A downside of this model is that is also marked 82 other data points as anomalous. Another discrete model was trained on the s_pu10, with $n = 3$ and a threshold of 10. It was able to detect 7 timesteps of attack 1, while having zero false positives. Not all anomalies were detected with the discrete model and this was partial caused by the SWIDE algorithm. As the trained discrete model on s_pu10 shows, it works best on signals that have predetermined states. It therefore does not work properly on flow and level sensor data.

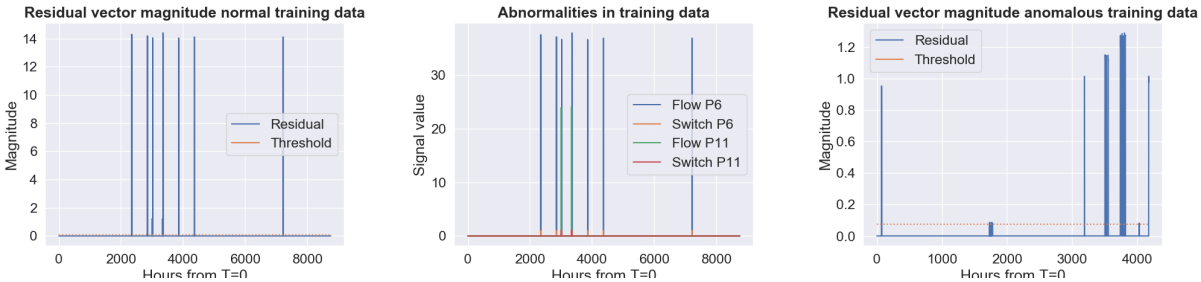
4 PCA

Another method to detect anomalies is to do a principal component analysis (PCA) of the signal data. This analysis will find the principal components of the data that are orthogonal to each other, and where the first component points in the direction of the highest variance of the data. The second component will

point in the direction of the second highest variance and this repeats until all variance is explained with a maximum amount of components as the size of the original dimensions of the data. As anomalies differ from normal signals, it is expected that most anomalous data will not have a high variance in normal signal directions but in other directions. Therefore, anomalies will most likely have the highest variance in the last principal component [2]. The PCA projection to the residual subspace will be according to [2] as $\tilde{y} = (1 - PP^T C)y$ where P is the matrix with the $k - 1$ principal components as columns, where k is equal to the number of dimensions in the data. This basically means that the residual is computed with only the last principal component.

The PCA transformation is computed on the first training dataset as these are normal signals only. This transformation was then applied to this dataset and yielded a plot of the magnitude of the residual error in Figure 4a. This plot shows that PCA detected anomalies in the training data, which is not something that is usefull as we are training to learn the variance of the normal signals. The abnormalities in the training data are shown in Figure 4b and it can be seen that these signals are almost always zero except for a few hours. These abnormalities were removed to let PCA only model the normal behaviour. Next, the learned PCA transformation was applied to the second training dataset with anomalies and the magnitude of the residual error is shown in Figure 4c. The threshold was set to 0.075 which gave 7 false positives, 112 true positives, and 380 false negatives on training dataset were all anomalies were labeled correctly.

An important aspect of PCA anomaly detection is that it can only anomalous signals if there is a significant difference with the normal signal. When we look at attack 3 and 4 of the training set, a SCADA concealment was done in the form of a replay attack or a polyline offset. As these replay attack have the same variance as the normal signals, the magnitude of the residual error will not increase significantly and thus the anomaly can not be detected.



(a) Residual error magnitude of the training set.

(b) The abnormalities that caused the high residual error of the training set.

(c) Residual error of the second training set

Figure 4: PCA anomaly detection for both training sets. The middle plot visualizes the abnormalities that occur in the normal training set that were removed for PCA anomaly detection.

5 Comparison

To compare the results of the different techniques we have used test point wise precision and recall and after how many hours we were able to detect an attack. Since action should be taken as soon as possible if a SCADA system is under attack we think this is a good metric.

5.1 Precision and recall

For the ARMA task we calculated the precision and recall for each sensor model in Table 1.

The overall results of all models are in Table 2.

As can be seen the PCA model did best on precision and pretty good on recall compared to the rest. The discrete model for the first sensor did pretty well. And ARMA performed poor.

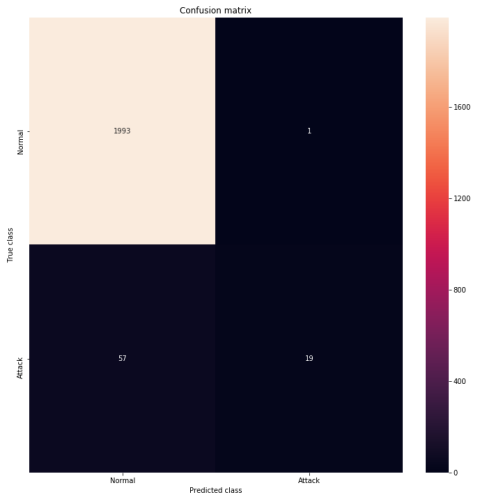
5.2 Attack detection duration

For the ARMA task the amount of hours it took to detect an attack is shown in Table 3. As can be seen, the models did not great overall, although by combining the results we were able to at least detect

all attacks and four of them even at the moment they started. PCA and Discrete1 did pretty good, but Discrete2 performed poorly.

6 Bonus: Deep learning

Finally, another way to detect anomalies in data can be done by using a deeplearning network. The provided script was used to train an autoencoder on all available training data that was relabeled such that all labels are correct. After 100 epochs the model achieved a training loss of 0.5889 and a validation loss of 0.5876. The anomaly prediction is then made with a reconstruction error threshold of 2. The resulting confusion matrix is show in Figure 5. An issue with using deep learning in anomaly detection is the interpretation of the model. Deep learning models are notorious for being hard to understand how the model achieved his predictions. In real world anomaly detection it might be important to understand why the anomaly prediction is made.



(a)

Figure 5: The resulting confusion matrix of the auto encoder.

Sensor	Precision	Recall
l_t1	0.724	0.043
l_t4	0.184	0.014
l_t7	0.219	0.014
f_pu1	1.000	0.006
f_pu7	0.000	0.000
f_pu10	1.000	0.006

Table 1: Precision and recall for all ARMA models for different sensors

Method	Precision	Recall
ARMA	0.358	0.079
PCA	0.957	0.227
Discrete1	0.600	0.250
Discrete2	0.100	0.014

Table 2: Precision and recall for different methods

A ARMA

B Comparison

References

[1] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. San Francisco, CA, USA: Holden-Day, Inc., 1990. ISBN: 0816211043.

[2] Anukool Lakhina, Mark Crovella, and Christophe Diot. "Diagnosing network-wide traffic anomalies". In: *ACM SIGCOMM computer communication review*. Vol. 34. 4. ACM. 2004, pp. 219–230.

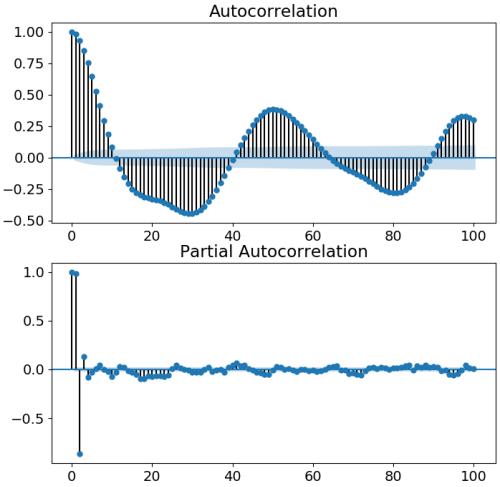
[3] Qin Lin et al. "TABOR: A Graphical Model-based Approach for Anomaly Detection in Industrial Control Systems". In: *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. ACM. 2018, pp. 525–536.

	Attack						
Sensor	1	2	3	4	5	6	7
l_t1	-	-	0	0	-	-	-
l_t4	-	6	33	43	-	-	0
l_t7	49	0	27	50	-	54	-
f_pu1	-	-	52	-	-	-	-
f_pu7	-	-	-	-	-	-	-
f_pu10	6	-	-	-	-	-	-
min	6	0	0	0	14	12	0

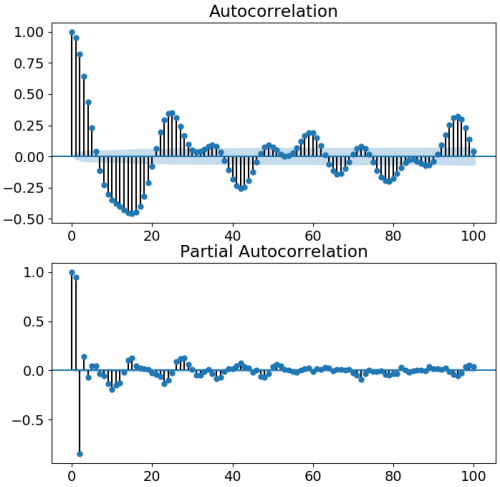
Table 3: Hours before an attack gets detected

	Attack						
Method	1	2	3	4	5	6	7
ARMA	6	0	0	0	14	12	0
PCA	0	-	-	-	6	11	101
Discrete1	-	-	0	0	26	52	43
Discrete2	7	-	-	-	-	-	-

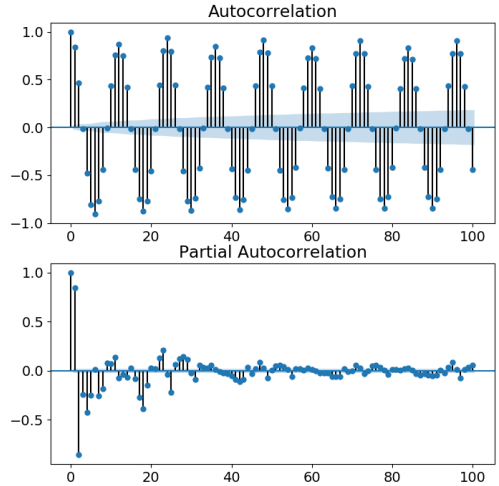
Table 4: Hours before an attack gets detected



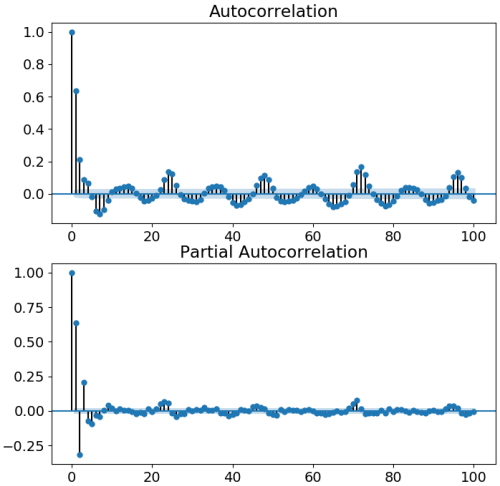
(a) l_t1



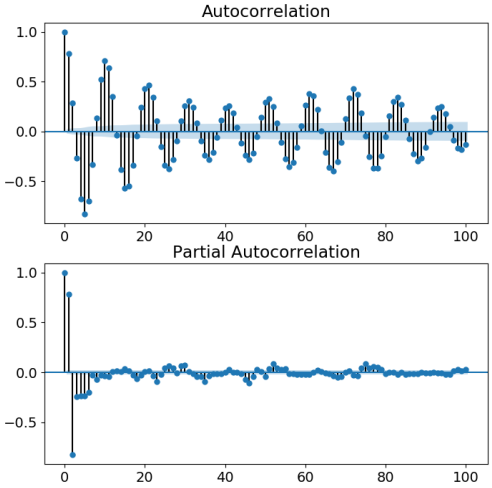
(b) l_t2



(c) l_t3



(d) l_t4



(e) l_t5

Figure 6: Autocorrelation plots