2 | Lab 2

Github the code is on github at: `https://github.com/mennobo/CS4035_CDA_3/tree/master/Assignment2`
Running the code To run the solution, open the file *Task1.ipynb* in jupyter notebook. Make sure you have installed the dependencies: imblearn, seaborn, pandas, sklearn, numpy, matplotlib, tslearn. Install them using pip or any other method of choosing.

2.1 Familiarization task (Task 1)

2.1.1 What kind of signals are there?

We use pandas DataFrame:describe() method to get insight into the data contained inside the dataframe. There are 44 column is total, 16 of which being integers and the rest floating point values. The integer values are all either 0 or 1, and nothing else.
Thus, we can say there are 2 main types of data, boolean values and numerical, these belong to the actuators and sensors respectively.
We have the columns L_T1, ..., L_T7. All of them are floating point values between 0 and 5.5, likely they describe the same kind of signal, preumably water levels in different tanks. Then we have the F_PU1, ..., F_PU11 and S_PU1,..., S_PU11. The F-columns are floating point values and the S-columns booleans. The naming suggests these are linked, and thus likely describe an actuator linked to a sensor in some way. Some F_PU columns contain all zeroes, its worth noting that if this is the case, the corresponding S-column contains all zeroes as well. We have columns F_V2 and S_V2 containing similar data as the colums described above, S_V2 being boolean and F_V2 float. Then come another 12 float columns named P_J followed by a (seemingly random) number they seem to contain similar data as the F_PU columns. Lastly there is ATT_FLAG, another boolean column indicating the presence of an attack or not.
The website[1] states: "the flow data unit is LPS, pressure and water level units are meters." This suggests that the F_* columns contain flow data, and the L_T* columns describe water levels.
If we take a few of the signals and plot them together, we see that many of them are cyclical, at least the F_PU* signals and L_T* move in regular intervals. The P_J* signals seem to be less regular, although they still move up and down with a fairly steady period and amplitude. Additionaly, at least some of them are definitely correlating signals, like F_PU1 and F_PU2.
If we make a heatmap we also see that some columns correlate heavily, also between the P_J and F_PU columns. For example, there is a strong negative correlation between F_PU1 and P_J269 and between F_PU2 and P_J280. There is a strong positive correlation between F_PU4 and P_J256 (among many others).
The attack dataset provided contains labels for some attacks. However, we know that there are some more attacks that are unlabelled, as wel as some that are labelled for shorter that they really are.[2] Because of this we add the missing labels for those attacks to get the best results.

2.1.2 Prediction

An Moving Average predictor was used to predict the series. A prediction is made by taking the average value of the last few points in the dataset as the prediction of the next value. For a window size of x the first x points are discarded, then point x + 1 is predicted by taking the mean of the x points. Then, point x+2 is predicted by taking the mean of x + 1, x and x - 1 (The actual value point x+1, not the predicted one from before).
Using this we predict the values of 3 randomly selected sensors: F_PU1, P_J14, L_T1. The resulting mean square error for windows sizes 1 upto 4 is given in the following table:

Window size:	MSE: F_PU1	MSE: P_J14	MSE: L_T1
1	22.529	7.821	0.055
2	28.055	9.263	0.117
3	34.441	10.744	0.195
4	41.027	12.223	0.286

From the results of the F_PU1 prediction using window size 3 we can create the following graph to visualize:
From this we can see that we can not predict the series well using the moving average predictor, the larger the window the worse the prediction becomes. The best results given by window size of 1, which amounts to just copying the previous seen point as prediction.

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

Figure 2.1: Caption

2.2 Task 2, ARMA Task

In this task we detect anomalies in the system using ARMA. First we have to pick the sensors we are going to use, to do this we looked at the highly correlating sensors and removed the highest. From every pair of sensors with a correlation of above 0.80, one was removed so that we only have sensors with a correlation below 0.80. When reducing this value the F and P columns tend to disappear.
The remaining columns are: L_T1, L_T2, L_T3, L_T4, L_T5, L_T6, L_T7, F_PU1, F_PU4, F_PU6, F_PU7, F_PU8, F_PU10, F_PU11, F_V2 and P_J300. Since this is still quite many columns, we select some 5 of these on which to run ARMA. We select L_T1, L_T4, L_T7, F_PU7 and P_J300. Since we know that these (except P_J300) are the sensors that were affected by some attacks [3]. P_J300 is included to see how well we can detect anomalies on this different sensor type.
Every time we run the ARMA function we first find out the optimal order of parameters for the ARMA model p and q. We do this using the AIC criterion. The approach we took is first fitting a model on the training data with increasing parameter p but keeping q at 0. We then look at the produced AIC values and select the highest p value for which the AIC still decreased dramatically. We then do the same for the parameter q while using the p selected above. Again, we look for the moment when the AIC stops decreasing steeply and use the corresponding q. This process is repeated for each sensor.
After the parameters are selected, the training and test models are created using these optimal p and q values. We then plot the residuals in a line plot and a qqplot.
In order to detect anomalies in the dataset we calculate the standard deviation of the test model residuals. As a threshold for anomaly we use the standard deviation multiplied by 2. This value was selected in order to produce a good accuracy value and create a balance between the amount of false positives and true positives being detected.
In order to show actual attacks vs detections, we generate a graph that plots the residuals together with the actual and detected attacks. For most sensors we zoom the graph to show at least the attack that directly affected the sensor. Looking at these graphs we can conclude that often we detect the attacks that directly affect the sensors on which ARMA is ran, but often also detect some false positives. The graphs for 2 sensors is shown below, see the notebook file for graphs for all tested sensors, as well as the true positives, false positives, accuracy and precision scores.

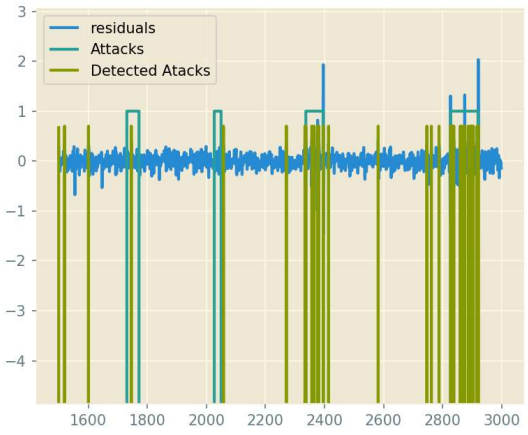

Figure 2.2: Detected attacks in L_T1

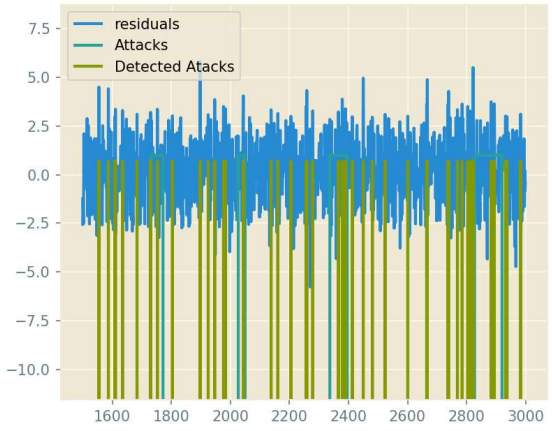

Figure 2.3: Detected attacks in P_J300

Important to note is that using ARMA we mainly detect sudden anomalies that last for a short period of time. This is because arma uses past datapoints to create a prediction. If an anomaly lasts for a longer period of time, the model will start to interpret it as a normal signal.
It is for example hard to detect Attack 5 and 6 using PU7, because the attack consists of lowering the speed of PU7. Lowering the speed does not create high residuals in the ARMA model, however, after the attack is done a peak occurs which is detected as possible attack.

[3] https://batadal.net/images/Attacks_TrainingDataset2.png

2.3 Task 3, Discretization task

To use discrete models we first need to discretize the data. This means that we convert the continuous data from the dataset into discrete data like for example high medium and low values. To do this we use Symbolic Aggregate Approximation (SAX). We picked this method as it is easy and fast to implement, it offers the ability to select the amount of segments you want to divide the dataset into as well as the amount of symbols to use to describe the final dataset. As you can see in figure 2.4, the dataset is converted into a discrete dataset with in this case 8 symbols. The image shows a significant reduction in dimensionality. Finally an anomaly is flagged if the n-gram of a datapoint is not present in the training set.

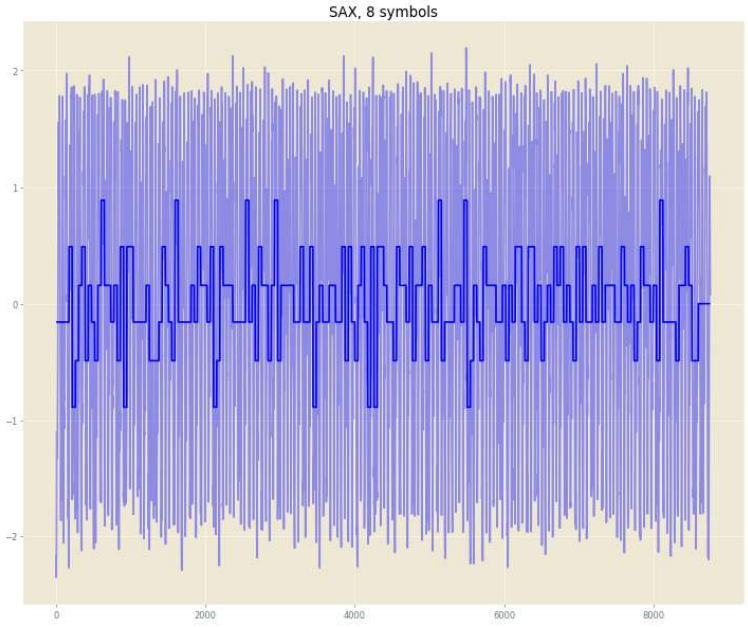

Figure 2.4: SAX visualization

In order to find the optimal anomaly detection. We decided on experimenting with 3 parameters: The amount of symbols used (between 1 and 20 in steps of 1), the amount of segments which is the window size (between 1 and 250 in steps of 20) and finally the n in n-grams (between 1 and 5 in steps of 1). First we did an exploration of all parameters selecting on combinations of parameters that provide the highest precision $\frac{TruePositive}{TruePositive+FalsePositive}$. A short overview of all combinations with a precision over 0.5 is displayed in figure 2.5. This overview does not display all combinations with a precision under 0.5 thus there are a lot of combinations (1235 in total) which have been tried but which do not provide a good combination. It is interesting to see that setting the window size to around 80 produces the best results, the amount of symbols is optimal around 5 to 7 and the n-gram size for the better scoring combinations doesn't matter a lot. More symbols require smaller n-grams which sounds logical as there are more symbols to differentiate abnormalities. The highest scores give 51 true positives with 0 false positives.

After getting a feel for optimal parameters we ran the algorithm to determine the best scores for the selection of sensors as described in Task 2. To determine the best parameters for each sensor we ran the whole experiment as described above and only saved the highest precision score. The resylts are displayed in figure 2.6. Noteworthy is that every sensor requires different parameters for an optimal score. The L_T1 sensor scores the best here, other sensors don't really score as high.

Experiment	Symbols [1 - 20]	Segments [1 - 250]	X_gram [1 - 5]	False negative	True negative	False positive	True positive	Precision
0	4	81	1	168	3912	46	51	0,525773
1	4	81	2	167	3909	48	52	0,52
2	4	81	3	166	3906	50	53	0,514563
3	4	81	4	165	3903	52	54	0,509434
4	4	81	5	164	3900	54	55	0,504587
5	5	61	1	176	3933	25	43	0,632353
6	5	61	2	175	3927	30	44	0,594595
7	5	61	3	174	3921	35	45	0,5625
8	5	61	4	173	3915	40	46	0,534884
9	5	61	5	172	3909	45	47	0,51087
10	5	81	1	168	3958	0	51	1
11	5	81	2	167	3955	2	52	0,962963
12	5	81	3	166	3952	4	53	0,929825
13	5	81	4	165	3949	6	54	0,9
14	5	81	5	164	3946	8	55	0,873016
15	7	81	1	168	3958	0	51	1
16	7	81	2	167	3954	3	52	0,945455
17	7	81	3	166	3950	6	53	0,898305
18	7	81	4	165	3946	9	54	0,857143
19	7	81	5	164	3942	12	55	0,820896
20	8	81	1	168	3912	46	51	0,525773
21	8	101	1	178	3922	36	41	0,532468
22	11	81	1	168	3958	0	51	1
23	11	81	2	167	3937	20	52	0,722222
24	11	81	3	166	3916	40	53	0,569892
25	12	101	1	178	3922	36	41	0,532468
26	14	81	1	168	3912	46	51	0,525773
27	16	81	1	168	3912	46	51	0,525773

Figure 2.5: SAX on the LT_1 sensor

Sensor	Symbols [1 - 20]	Segments [1 - 250]	X_gram [1 - 5]	False negative	True negative	False positive	True positive	Precision
L_T1	5	81	1	168	3958	0	51	1
L_T2	5	61	2	218	3955	2	1	0,33
L_T3	9	141	1	161	3842	116	58	0,33
L_T4	7	101	2	218	3957	0	1	1
L_T5	17	101	1	168	3845	113	51	0,31
L_T6	15	181	1	196	3889	69	23	0,25
F_PU4	8	221	5	213	3948	6	6	0,5
F_PU6	9	101	1	139	3797	161	80	0,33
F_PU7	7	41	1	139	3634	324	80	0,2
F_PU8	5	201	2	218	3957	0	1	1
F_PU10	7	241	2	218	3957	0	1	1
F_PU11	3	41	1	177	3899	59	42	0,42
F_V2	7	101	2	218	3956	1	1	0,5

Figure 2.6: SAX on selection of sensors

After looking at the anomaly detected by SAX (as displayed in figure 2.7 it is clear that only 1 anomaly is detected. It is however detected correctly.

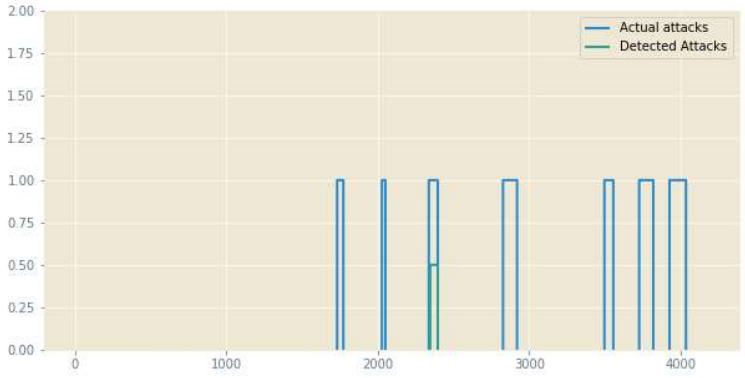

Figure 2.7: SAX prediction

A final note is that we used a rudimentary experimental design, a better method for finding a globally optimum set of parameters might be able to find other results. A different way of flagging anomalies could also produce better results.

2.4 Task 4, PCA

In order to do PCA analysis we first need to do some pre processing. In order for PCA to work correctly the data needs to be normalized so that it has a mean of zero and unit-variance, such that each feature will be weighted equally in our calculations. This is done using the "tslearn" package.
Secondly we need to filter out abnormalities in the training dataset. In order to find any abnormalities if they exist we plot the (squared) PCA residuals. This plot is shown in figure 2.8.

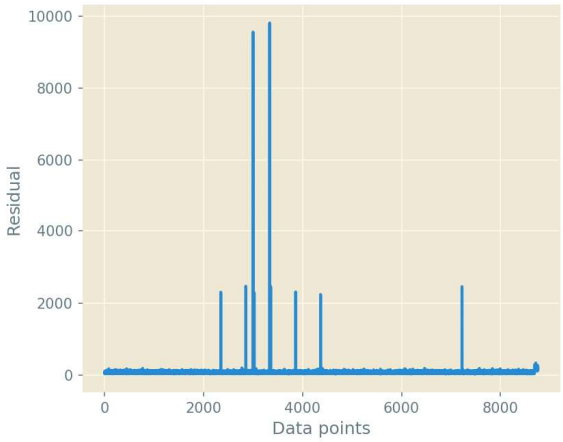

Figure 2.8: PCA residuals (squared)

This plot shows a few clear abnormalities. These need to be removed as they can affect the principal components. After removal the mean of the time series is no longer 0, and the data no longer has unit-variance. So, we re-normalize the cleaned data to achieve this again.
Next we need to tune the parameters, primarily the number of principal components we will use. We need a number of components that captures a big percentage of the variance. In order to decide this we plot the fraction of variance captured by each principle component, for this plot see figure 2.9. Then, we plot the cumulative variance captured by an increasing number of principal components in figure 2.10. Reading from this plot we can see that at 15 principal components, about 99% of the variance should be captured.

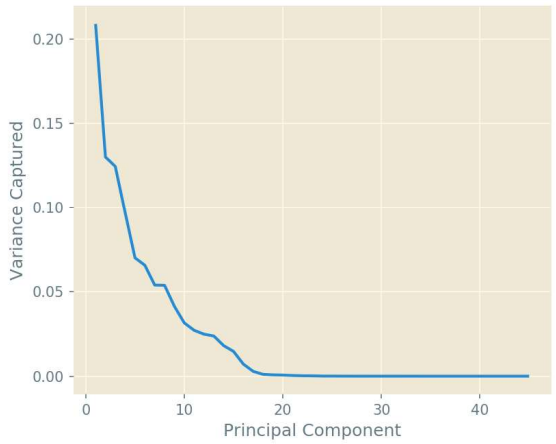

Figure 2.9: Variance captured by each principal component

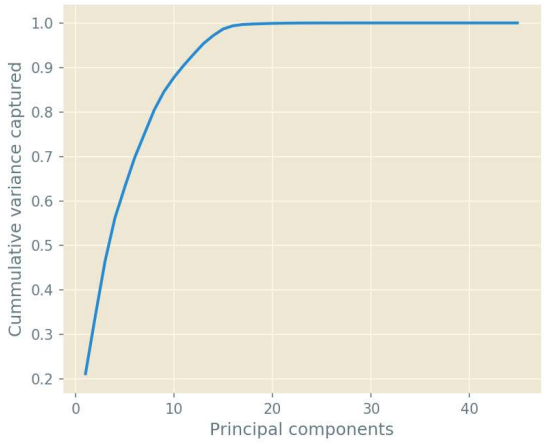

Figure 2.10: Cumulative variance captured by principal components

The thresholds are set to the maximum and minimum residual of the training data, (this will result in 0 false positives in the training data).
We now run the PCA analysis using 15 principal components on the test data. Any resulting residuals that are either above the maximum threshold or below the minimum are marked as possible attacks.
The PCA analysis results in 123 true positives being detected and 22 false positives, see the graph below for a visual represen-

tation. The PCA analysis seems to detect anomalies at each of the labelled attack moments, but it does detect a few loose false positives.

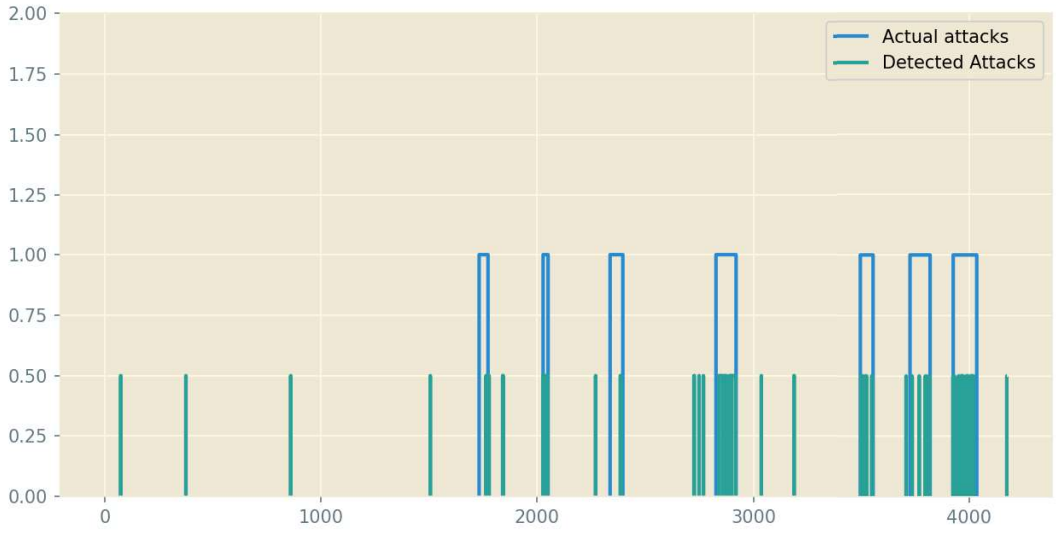

Figure 2.11: PCA analysis results

If we increase the maximum threshold by a factor of 2, and decrease the minimum by dividing it by 2, we can improve the model so that it generates 92 true positives and only 4 false positives. Using these exaggerated thresholds we still detect at least once during each attack in the dataset.

2.5 Task 5, Comparison

In order to compare the different methods we look at the ratio of true positives compared to false positives. Because in practise someone has to perform an inspection on every positive, we want to minimize time inspecting false positives. This means that a high amount of true positives in comparison to a low amount of false positives is a good score. We also want to select for a method that shows the start of attacks and not necessarily the whole duration of the attack, because in practise on an alarm someone will inspect and remedy the attack if it is an actual attack. The detection should at least detect all present attacks at least once too, since if theres no detection at all no one will inspect the system and the attack will go unnoticed.

ARMA
ARMA is useful to detect the start of attacks after which an inspection should take place and action should be taken. ARMA detects sudden unexpected changes in the system, which is often the case when there is an attack. However, when the attack consists of reducing the pump speed for example (attacks 5 and 6), ARMA has a hard time detecting this. This is the case for sensor F_PU7, we do see a clear pattern in the residuals when there is an attack, however nothing is detected until there is a spike in the residuals right after the attack. If all signals analysed using ARMA simultaneously, we would at least have some detections in each attack. However, the large amount of false positives that would also be detected would make using this in practice unfeasable.

SAX
SAX is really useful because it correctly detects attacks and doesn't sound false alarms. This is very useful because every time SAX raises an alarm something is actually wrong. The final best score was 51 True positives but only on a single attack.
PCA
The PCA analysis resulted in 123 true positives being detected and 22 false positives. PCA does a good job in detecting all the attacks present in the dataset, however there are still too many false positives that are also detected. Using the updated threshold the PCA algorithm becomes a very good anomaly detector for practical use.
In summary it is better to look at a combination of different methods but mainly focus on using PCA.