Assignment 2 CDA

Tim van Rossum, 4246306
Michiel Doesburg, 4343875

June 1, 2018

1   Familiarization with the data

The dataset contains a total of 44 signals, with some being static values, other being sinusoid signals, and others being partially discrete (as in, they can rise or drop at certain points). L-signals tend to be more sinusoidal, F-signals tend to be "partially discrete", and S-signals are all binary signals.

All the signals together seem to have little correlation. Some combinations of signals are reasonably correlated: e.g. L_T3 and P_J302 have a 0.42 correlation coefficient. Moreover, the signal P_J302 in figure 3 looks cyclic. A cyclic signal like this is much easier to predict than a 'random' signal, since there is a rough base model that it adheres to. This signal's values are within a clear band at about 2-6 on the Y-axis. In such a case it is very easy to come up with a basic anomaly detection technique: if the signal moves outside this band this would be a clear sign of an anomaly. Plotting the mean error we saw for the ARMA prediction on signal L_T1, we saw that there were only 2 cases were the error was greater than three times the standard deviation.
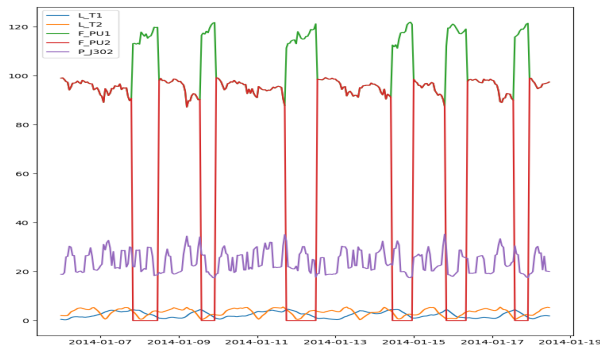

Figure 1: Visualization of some signals. The red and green signals are partially discrete.
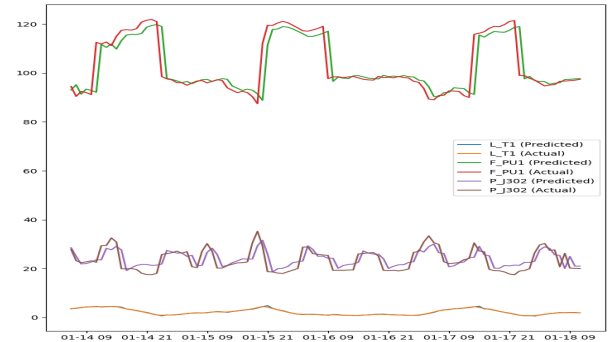

Figure 2: ARMA predictions on some signals. The predictions work better on signals with fewer rapid changes.
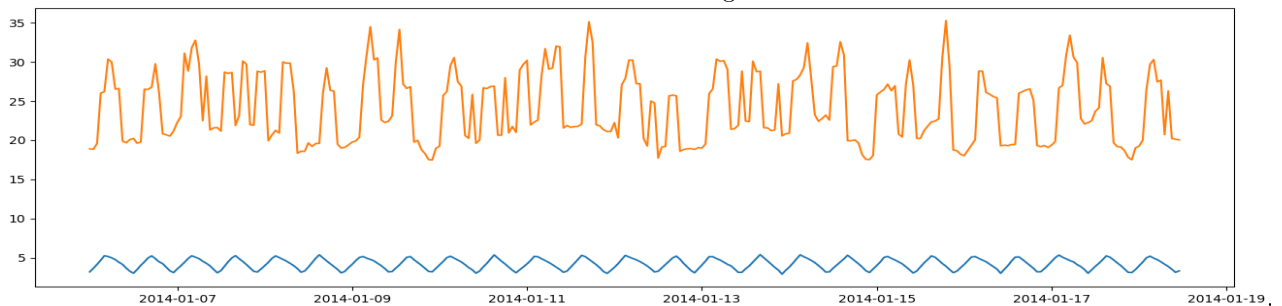

Figure 3: Signals L_T3 and P_J302. The peaks are roughly aligned. These signals have around 40% correlation.

2 ARMA

The script `ARMA.py` learns a ARMA model for any particular sensor. The parameters for the model are determined by testing out different p's and q's and then looking at the AIC value of the fitted model. A lower AIC means a better fit. Fitting the models takes quite some time, and as such the search space for parameters is limited. For p, only values in the range of $[0, 4]$ are tested, and for q, only values in the range of $[0, 2]$ are tested. Both these ranges are inclusive. We thought these ranges would still allow for a reasonable search space while limiting the time taken to search for the ideal parameters.

After the predictions are done, the predicted values are compared to the actual values. The difference between the two is used, and then the mean of the absolute values is computed, along with the standard deviation of the absolute values. An attack is detected if a predicted value differs from the actual value by more than the threshold, which is given by the absolute mean plus a number of times the standard deviation. We used mean absolute error instead of root mean squared error because the root mean squared error gives a larger weight to larger errors. The sensors that have cyclic time series can be modelled effectively by ARMA, and the anomalies that can be detected are contextual anomalies. Sensors that have rapid changes are less effectively modeled by ARMA, what can usually be seen is that the prediction lags behind the actual values, like in figure 2.
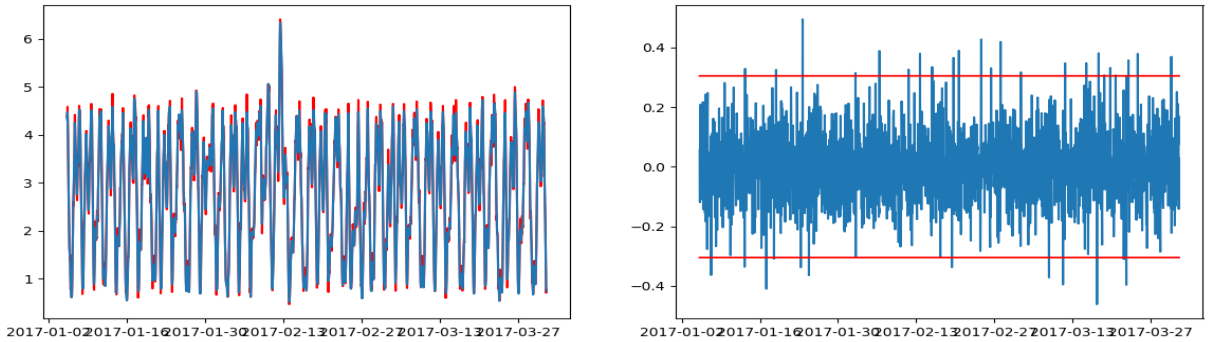

Figure 4: The prediction for signal L_TU1. On the left is the signal in blue versus the predicted signal in red. On the right are the residual errors and the threshold.

When setting the threshold to 3 times the standard deviation we can see that there are a lot of detected anomalies. Looking at the signal, most of these are probably benign. The threshold might need to be set a little higher for this signal to weed out a few more (likely) false positives. The detected anomalies are contextual anomalies.

3 Discrete models

For this task, we used the SAX method as shown in class. The script `SAX.py` is a script that can discretize a time series using SAX. The script was provided by Qin Lin, and we altered a few things to make it work with Python 3. Discretization of the signal in this way creates a representative string of characters of the signal, which allows us to perform sequential data mining methods on it. This is in contrast to PAA, which only discretizes the signal, which does not allow us to perform these data mining methods. Discretization by use of SAX makes sense as the signal values are now grouped by how much they differ from the mean of the signal. The window size determines how many mean values the signal will be converted to, and the amount of original data points used per mean. Using a very large window size will obviously result in a very rough approximation and much data loss, using a very small window size will simply approach the original data and be pointless. After some trial-and-error, we found a window size of 3 to work well. The alphabet size determines the range of possible discrete values the means will be mapped to.

The script `applying_SAX.py` does not only apply SAX on the signals, but also applies the N-grams data mining method to find anomalies. An N-gram is anomalous (and possibly an indicator of an attack) if the score is lower than either the mean score minus three times the standard deviation, or 0.3, whichever is higher. The 0.3 was chosen to prevent the score boundary from becoming negative. It is rather rare to have that happen on this data though, as having large word sizes allows us to have a mean score of around 0.98 for

most signals with a standard deviation of about 0.2. The sensors that do not have cyclic time series can be modelled effectively by SAX, and the anomalies that can be detected are collective anomalies, as can be seen in the visualization.
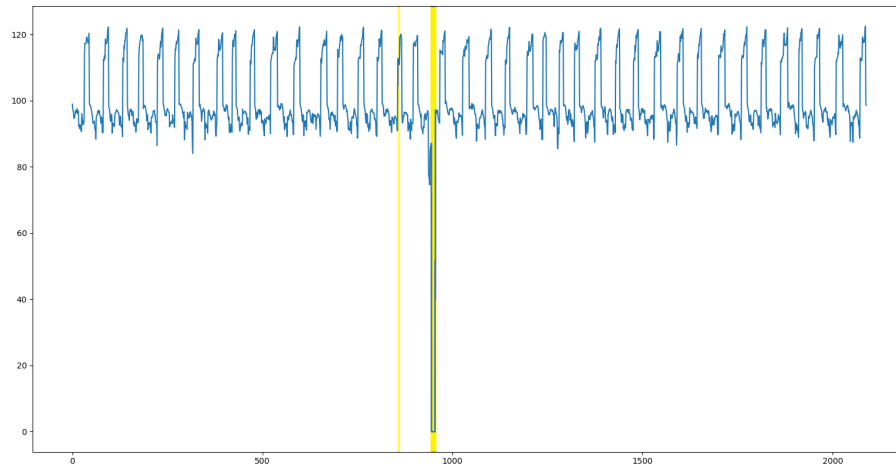

Figure 5: SAX and Ngrams applied to signal F_PU1. The anomalous regions are highlighted in yellow.

4 PCA

For this task, we first transformed the dataset using PCA. We do not normalize the signals as some signals have a standard deviation of 0, and working around this by dividing the time series only when the standard deviation is non-zero, or by discarding the signals from the dataframe that have a standard deviation of 0, causes the isolation forest to not detect anything at all any more. This does have the side effect that the first principal component will dominate the calculation of the residuals. The procedure can be found in the script PCA.py. We used the PCA implementation in Scikit-learn because we use Python as our platform. After the dataset was transformed, we could perform outlier detection. Scikit-learn also has several methods for that: one-class SVM, isolation forest, elliptic envelope, and local outlier factor. We decided to use the methods shown in class as we were familiar with them, which were the one-class SVM and the isolation forest. The isolation forest model finds significantly less anomalies than the one-class SVM model, possibly making the isolation forest model better in this case.

The workflow is as follows: the models are first fitted on the transformed dataset that contains no attacks whatsoever (which is the first of the BATADAL training datasets). Then, predictions are made for the transformed test dataset. If an index is predicted to be an outlier, then the timestamp is stored in a list. Figure 5 shows the residuals after transforming the data using eight principal components. Using PCA, we can detect point anomalies.
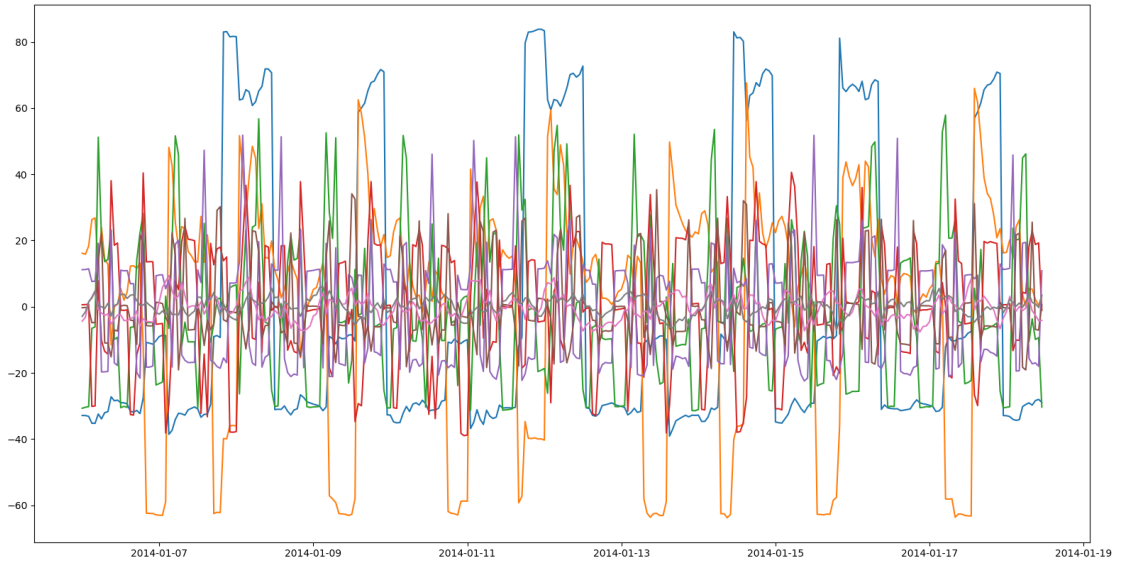

Figure 6: The first 300 data points of the PCA residuals. We chose for eight principal components as that meant that we detected quite a few anomalies (between 70 and 80) with the isolation forest model, instead of more than 100 or none at all.

5   Comparison