Anomaly Detection in Predictive Maintenance

Time Series Prediction, Training & Prediction without a class

Rosaria Silipo
Rosaria.Silipo@knime.com

Iris Adä
Iris.Adae@knime.com

Phil Winters
Phil.Winters@knime.com

Summary

Anomaly Detection covers a large number of data analytics use cases. However, here with Anomaly Detection we refer to the detection of unexpected events, specifically of mechanical failures.

The “unexpected” character of the event means that no such examples are available in a dataset. So, how do we proceed in a case where no examples are available? It requires a little change in perspective. In this case, we can only train a machine learning model on non-failure data, i.e. on the data describing the system operating in “normal” conditions. The evaluation of whether the input data is an anomaly or just a regular operation must be performed during deployment after predictions have been made.

The idea is that a model trained on “normal” data can only predict the next “normal” sample. However, if the system is not working in “normal” conditions anymore, the model prediction will be far from reality. Thus, the distance between the reality sample and the predicted sample can tell something about the underlying system’s condition.

This whitepaper shows the implementation of an auto-regressive model for time series prediction, trained on a time window where the rotor was working properly. The distance between predicted and real signal, together with its statistics, is then calculated on this same training window.

During deployment, new samples are predicted from input samples, distance is computed, alarm signals are calculated, and statistics based thresholds are applied.The alarm signal wandering off the thresholds triggers a set of verification procedures.

Workflows and data used for this whitepaper can be found on the KNIME EXAMPLES server under *50_Applications/17_AnomalyDetection*

Table of Contents

Anomaly Detection as a Time Series Problem

We are all witnessing the current explosion of data: social media data, clinical data, system data, CRM data, web data, and lately tons of sensor data! With the advent of the Internet of Things (IoT), systems and monitoring applications are producing humongous amounts of data which undergo evaluation to optimize costs and benefits, predict future events, classify behaviors, implement quality control, and more. All these use cases have been relatively well established by now: a goal is defined, a target class is selected, a model is trained to recognize/predict the target, and the same model is applied to new never-seen-before real-life data.

The newest challenge lies in predicting the "unknown". The "unknown" is an event that is not part of the system past, an event that cannot be found in the system's historical data. In the case of network data the "unknown" event can be an intrusion, in medicine a sudden pathological condition, in sales or credit cards a fraudulent transaction, and, finally, in machinery the breakdown of a mechanical piece. A high value, in terms of money, life expectancy, and/or time, is usually associated with the early discovery, warning, prediction, and/or prevention of the "unknown" and, most likely, undesirable event.

Specifically, prediction of "unknown" disruptive events in the field of mechanical maintenance takes the name of "anomaly detection".

In this particular project, attributes and their evolutions are monitored over time before the catastrophic - previously unseen - event occurs. A deviation of attribute evolution from historical evolution patterns can be a warning sign for an anomaly to happen. This might trigger an anomaly alarm, requiring further mechanical checkups.

There are a lot of use cases suitable for an anomaly detection application: turbines, rotors, chemical reactions, medical signals, spectroscopy, and so on. In this whitepaper we deal with rotor data.

When a rotor is slowly deteriorating, one of the sensor measurements might change gradually over time until eventually the rotor breaks. On the one hand we want to keep the rotor running as long as possible – mechanical pieces are expensive! –; on the other hand, we want to avoid the rotor breaking down completely, producing even more damage.

The easiest approach to an anomaly detection problem is just to observe the signal wandering over time. Signal boundaries are defined in the anomaly-free time window. The boundaries are usually centered on the average signal value and bounded by twice the standard deviation in both directions. If the signal is wandering off this anomaly free area, an alarm should occur. This technique is named **Control Chart**,

A more sophisticated approach predicts the signal's future values with a more complex model than just its average. We used here an **Auto-Regressive (AR) model** to predict the future numerical values of each one of the time series in the data.

On an anomaly-free time window, AR models are trained, future values are predicted, distance between predicted and real values is calculated, boundaries are defined on distance statistics, and a few alarm signals are built based on the distance values. During deployment, if the alarm signals wanders off the defined boundaries, an alarm is fired off requiring further checkups.

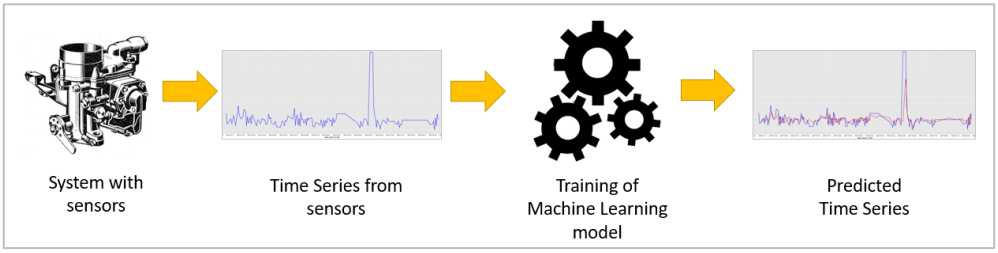

Figure 1.

Anomaly Detection problems often do not offer anomaly examples in the training set. In this case, we can only train a machine learning model on anomaly-free data and calculate a distance between the original and the predicted signal to trigger an alarm.

Data and Pre-processing

We used here a twenty-eight sensor matrix, focusing on eight parts (Fig. 2) of a mechanical rotor, on a time frame spanning January 1, 2007 through to April 20, 2009. In total, we have 28 time series from 28 sensors attached to 8 different parts of the mechanical rotor.

A1	input shaft vertical
A2	second shaft horizontal upper bearing
A3	third shaft horizontal lower bearing
A4	internal gear 275 degrees
A5	internal gear 190,5 degree
A6	input shaft bearing 150
A7	input shaft bearing 151
M1	torque KnM

Figure 2.

The 8 locations of the 28 sensors on the rotor

The signals reach us after the application of the Fast Fourier Transform (FFT), generating spectral frequency and amplitude values, in the form of:

```
[date, time, FFT frequency, FFT amplitude]
```

Amplitude values have been averaged across frequency bands, time, and sensor channel. This results in 313 time series, describing the system evolution in different locations and frequency bands. Each spectral amplitude originates from one of the original 28 sensors and refers to a 100Hz-wide frequency band falling between 0Hz and 1200Hz (file AlignedData.csv).

The whole data set shows only one breakdown episode on July 21, 2008. The breakdown is visible only from some sensors and especially in some frequency bands. After the breakdown, the rotor was replaced and much cleaner signals were recorded afterwards.

Figure 3.

Evolution over time of time series A1-SV31[0, 100] and A1-SV31[500, 600]. The rotor breakdown episode on July 21st 2008 is easily visible in the higher frequency bands [500, 600] Hz rather than in the lower frequency band [0, 100] Hz. There are 313 such time series in the data set referring to different frequency bands of the original 28 time series.

Anomaly Detection Technique: Control Chart

The most straightforward approach to anomaly detection calculates the average value of the past signal for each time series and monitors any deviation from it, going forward.

Average, Standard Deviation, and Level 1 Alarm

After reading the data from the file named AlignedData.csv and after fixing the missing values in the time series with the latest available value, we loop across all columns to calculate:

- The cumulative average (`avg`) with a Moving Aggregation node

- The cumulative standard deviation (`stddev`) with the same Moving Aggregation node
- The boundaries for “normal” time series behavior as
  - `UCL = avg + 2* stddev`
  - `LCL = avg - 2* stddev`
- The level 1 alarm signal for each column / time series as:

```
$messure$ < $LCL$ => 1
$messure$ > $UCL$ => 1
TRUE => 0
```

The Moving Aggregation node allows for a number of statistical / aggregation measures to be calculated on a moving window, such as sum, average, standard deviation, median, maximum, and more.

The moving window can be forward (the first window sample is substituted with the average value calculated on the whole window), central (the central window sample is substituted with the average value calculated on the whole window), and backward (the last window sample is substituted with the average value calculated on the whole window).

The Moving Aggregation node also implements cumulative calculations of the same statistical / aggregation measures, by enabling the flag “Cumulative computation”. “Cumulative” means that the measure is calculated on all samples of the time series prior to the current row. So, a cumulative sum is the sum of past values up to the current row, a cumulative average is the average calculated on all past samples up to the current row, and so on.

The level 1 alarm is set through a Rule Engine node. The node sets 1 when the signal wanders off of the “normality” boundaries and 0 otherwise. The first level alarm time series are then collected by a Loop End node (Fig. 5).

Level 2 Alarm and Taking Action

The level 1 alarm time series are just series of 0s and 1s. However, a 1 alone does not mean much. It could be due to anything temporary: an electricity spike, some turbulence, or something else quick and unexpected. What is much more worrisome is a sequence of positive level 1 alarms across all time series, i.e. across all frequency bands and across all rotor monitored variables.

In order to differentiate between a single level 1 alarm episode and a more generalized episode, a Column Aggregator node calculates the average value across all time series and for each day. A Rule Engine then produces a level 2 alarm value, with value 1 if the calculated average exceeds 0.25. Thus, a value of 1 in the level 2 alarm time series indicates a more consistent deviation of the signal and therefore triggers a checkup procedure.

Figure 4 shows the time series for level 1 alarms in blue and for level 2 alarms in red. The first level 2 alarms are visible just briefly at the beginning of 2007 and more substantially starting in March 2008, a few months before the actual breakout of the rotor.

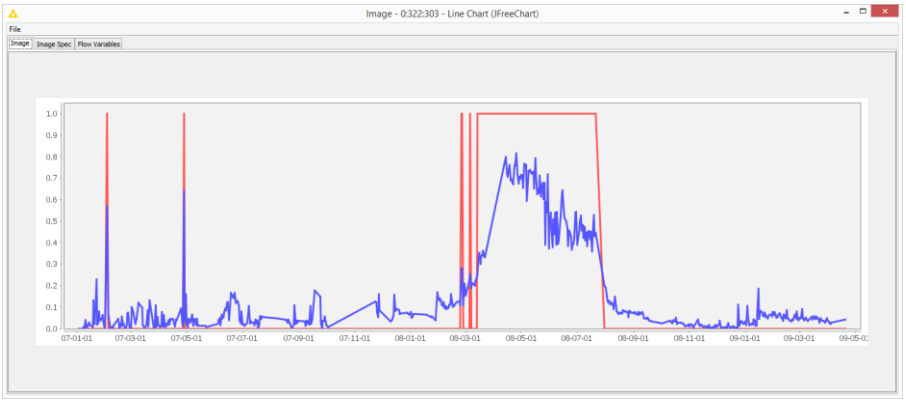

Figure 4.

Level 1 Alarm signal (blue) and Level 2 Alarm signal (red).

When a level 2 alarm is fired we have a number of possible actions we can take: howling sirens, switching off the system, or just sending an email to the person in charge of mechanical checkups. We chose to proceed with sending an email. The final workflow, named *04_Creating_a_ControlChart_of_a_Time_Series*, is reported in Figure 5.

This analysis is intuitive and easy to implement. However, it is a bit primitive in looking for signal anomalies. For example, the level 2 alarm fired already at the beginning of 2007. This might have been a bit premature, since the real breakdown only happened in July 2008.

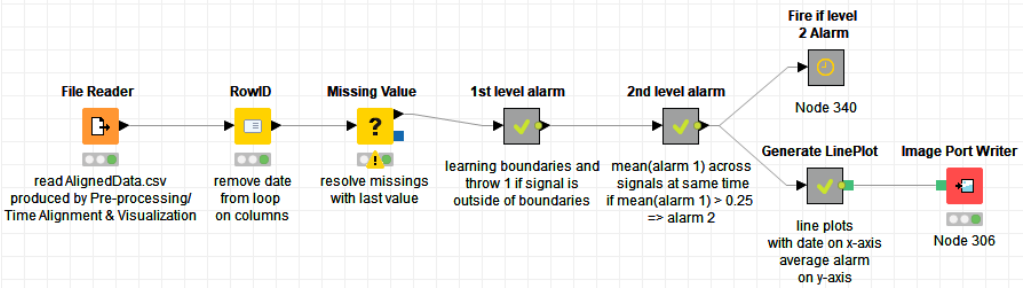

Figure 5.

Workflow implementing the Control Chart strategy for anomaly detection

Anomaly Detection Technique: Auto-Regressive Models

While the concept of defining “normality” boundaries seems to be a promising one, we need to use more sophisticated models to describe the normal functioning of the rotor. A simple average and standard deviation might not be accurate enough.

With this in mind, we move to the next approach involving auto-regressive models to describe the “normal” behavior. Here, instead of using the average “normal” value, we predict the next “normal” value from an auto-regressive model trained on a time window where the rotor was working fine. For that, we used the time window from January to August 2007 as the training window. We use the time window after that, from September 2007 to 21 July 2008, as the maintenance window to check whether the rotor measures are compliant with the model predictions.

Auto-Regressive Models and Distance Statistics

On the training window, an auto-regressive (AR) model is trained to predict the current value using its past for each time series. This generates 313 AR models, i.e. as many as the available time series. In practice, we loop on all time series and on each time series:

- We build a vector of 10 past samples together with the current value using a Lag Column node
- We impute missing values with the latest available value in time
- We train a Linear Regression model on the 10 past samples to predict the current one
- We save the linear regression model in PMML format to be used later in deployment
- We calculate the distances between predicted and real values
- We calculate the distance statistics (mean and standard deviation)

Note. The parameter N=10 of past samples could be optimized, by using an optimization loop that maximizes the model performance (see Numeric Scorer node) on the number of past samples.

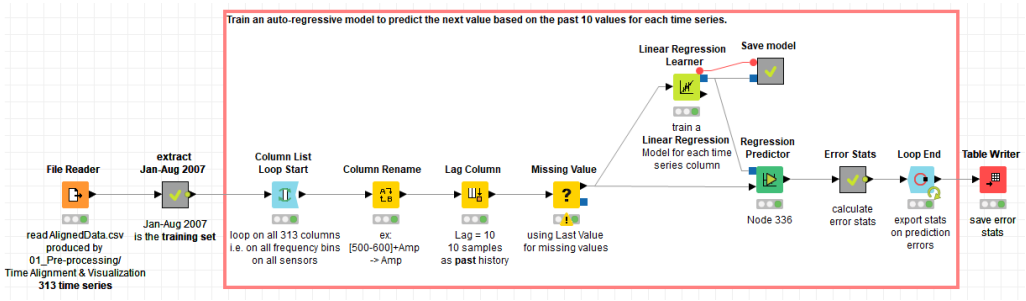

Figure 6.

Training workflow 02_Time_Series_AR_Training. It trains an auto-regressive model on 10 past values to predict the current value for each of the input time series. Notice that the training set describes a system correctly working. The model will be able only to predict sample values for a correctly working system.

Alarm Levels and Deployment

Once the models are in place, we can deploy them to define an alarm system. The idea is that the models, being trained over a time window with normal functioning, are able to predict the next sample value only for a correctly working rotor. The model will actually fail at predicting the next value, if the rotor has started to mal-function.

Here, during deployment, for each time series, we predict the next value based on the past 10 values, then measure the distance between the predicted value and the current real value, and finally compare such distance with the error statistics generated during training. If the error distance is above (or below) the error mean +(-) 2* standard deviation, an alarm spike is created as large as the distance value, otherwise the alarm signal is set to 0. This alarm signal is named “alarm level 1”.

The whole prediction, error distance calculation, comparison with mean and standard deviation of training error, and final level 1 alarm calculation is performed inside the column loop within the deployment workflow. 313 level 1 alarm time series are calculated, i.e. one for each time series.

The level 1 alarm is a series of more or less high spikes. A single spike per se does not mean much. It could be due to electricity fluctuation, temperature quick change, or whatever temporary cause. A series of spikes, on the opposite, might mean a more serious and permanent change in the underlying system. Thus, an level 2 alarm series is created as the moving average of the previous 21 samples of the level 1 alarm series, on all 313 columns. These are the level 2 alarm time series and are calculated in the metanode named “Alarm Level 2” (Fig. 7).

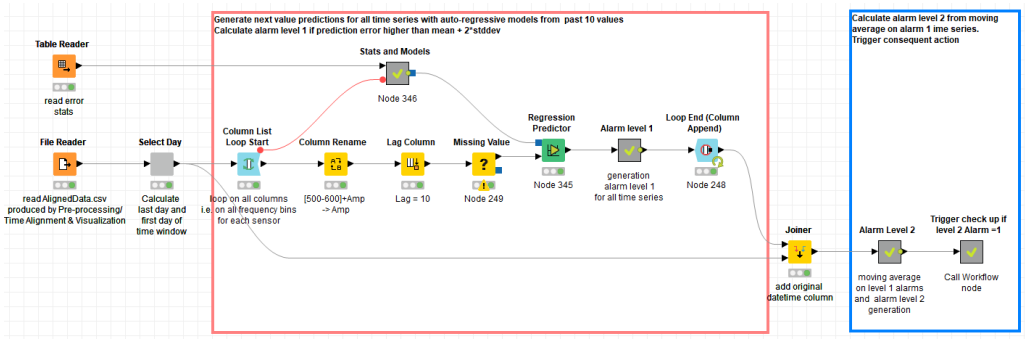

Figure 7.

Deployment workflow 03a_Time_Series_AR_Deployment. Here we read the models trained and saved in the training workflow, we apply them to the data in a new time window (at least 2 months long), we calculate the distance between predicted samples and original samples and we generate two level alarms. If alarm level 2 is active, a checkup procedure is triggered.

At this point, all level 2 alarm values are summed up across columns for the same date. If this aggregated value exceeds a given threshold (0.01) the alarm is taken seriously and a checkup procedure is triggered, in metanode named “Trigger Checkup if level 2 Alarm = 1”.

The trigger agent in that metanode starts an external workflow via the “Call Workflow (Table Based)” node. This node in its configuration window is set to start the external KNIME workflow “Send_Email_to start_checkup”. The workflow “Send_Email_to start_checkup” has just one central node: the Send Email node. The Send Email node - as the name says - sends an email using a specified account on an STMP host and its credentials.

Just barely modifying the deployment workflow, we get the chance to test this strategy on a number of data points and therefore observe the evolution over time of the level 2 alarm time series. In the modified version, we read all data after the training set portion, i.e. from Sep 2007 till July 2008. The level 2 alarm time series are visualized for each frequency band for each sensor in a stacked area chart. As you can see, level 2 alarm values are raising already at beginning of March 2008 across all frequency bands and all sensors. However, the change in the system becomes evident at the beginning of May 2008, especially in some frequency bands of some sensors (see [200-300] A7-SA1 time series).

Considering that the rotor broke off in July 22 2008, this would have been a fairly advanced warning time!

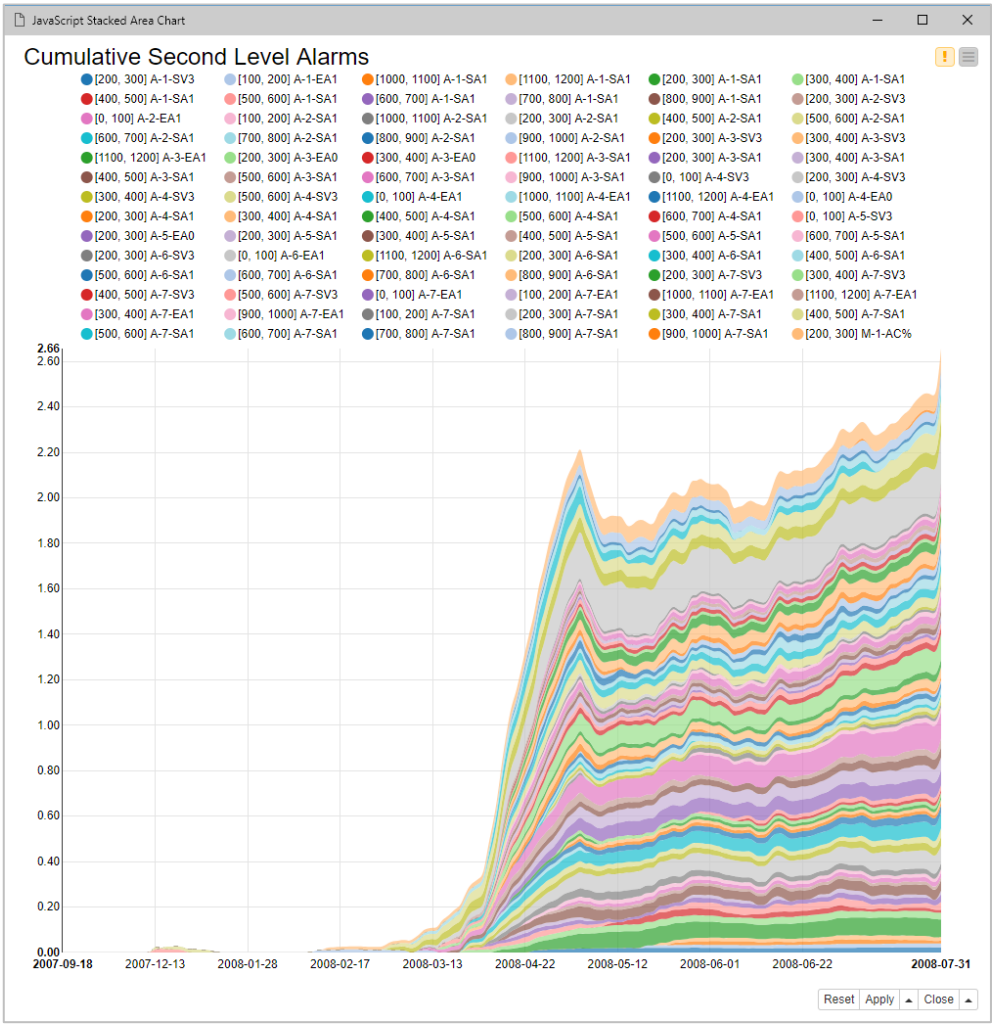

Figure 8.

The deployment workflow has been slightly modified to run a test on the remaining time window till the breakup episode (Workflow 03b_Time_Series_AR_Testing). The result is a stacked area chart piling up all level 2 alarms from Jan 2007 till July 2008. You can see the alarm signal rising in March 2008 and more in May 2008.

Conclusions

In this whitepaper we investigated two techniques used in anomaly detection applications.

The first technique (Control Chart) approximates FFT pre-processed sensor time series with their respective average values from an anomaly free time window.

The second technique (AR models) models the FFT sensor time-series with predictions from auto-regressive models trained on a time window where the rotor was working fine.

In both cases, boundaries are set and alarm signals are calculated, in order to discover early signs of anomalies.

The idea here was to train machine learning models on signals from the normally working rotor - because that is all we had available - and then to compare each model predictions with the real signal to find possible early signs of system changes. In this case an email is triggered, setting in motion a series of mechanical checkups.

Workflows and data used for this whitepaper can be found on the KNIME *EXAMPLES server under 50_Applications/17_AnomalyDetection*