A Toolbox for Realtime Timeseries Anomaly Detection

Markus Böbel
TU Munich and DXC Technology
Munich, Germany
markus.boebel@tum.de

Ilias Gerostathopoulos
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
i.g.gerostathopoulos@vu.nl

Tomas Bures
Charles University in Prague
Prague, Czech Republic
bures@d3s.mff.cuni.cz

***Abstract*—Software architecture practice relies more and more on data-driven decision-making. Data-driven decisions are taken either by humans or by software agents via analyzing streams of timeseries data coming from different running systems. Since the quality of sensed data influences the analysis and subsequent decision-making, detecting data anomalies is an important and necessary part of any data analysis and data intelligence pipeline (such as those typically found in smart and self-adaptive systems). Although a number of data science libraries exist for timeseries anomaly detection, it is both time consuming and hard to plug realtime anomaly detection functionality in existing pipelines. The problem lies with the boilerplate code that needs to be provided for common tasks such as data ingestion, data transformation and preprocessing, invoking of model re-training when needed, and persisting of identified anomalies so that they can be acted upon or further analysed. In response, we created a toolbox for realtime anomaly detection that automates the above common tasks and modularizes the anomaly detection process in a number of clearly defined components. This serves as a plug-in solution for architecting and development of smart systems that have to adapt their behavior at runtime. In this paper, we describe the microservice architecture used by our toolbox and explain how to deploy it for obtaining an out-of-the-box solution for realtime anomaly detection out of ready-to-use components. We also provide an initial assessment of its performance.**

***Index Terms*—data-driven decisions, anomaly detection, toolbox, timeseries**

I. Introduction

Software architecture practice relies more and more on data for decision making: product managers, system administrators, and the systems themselves make decisions based on data they collect and process. Consider, for example, the continuous monitoring of CPU and memory of the servers used by an application to identify over- or under-utilizations and reacting by starting new servers or stopping running ones. A problem with data-driven decisions is that they can be influenced by the quality of measured data and in particular by data anomalies. Indeed, real-life data are often noisy, incomplete, and contain wrong or invalid values. Neglecting to detect and filter out such data anomalies can severely bias the outcome of data-driven decisions. This has been documented in decision-making via online experimentation, where data quality assurance is a top priority [1].

There is therefore a clear need to consider anomaly detection as an integral part of any analysis pipeline for timeseries data coming from running systems. Such an anomaly detection component should accurately and timely identify both simple anomalies, such as invalid or out-of-bounds values, and contextual ones, e.g. unusual or unexpected peaks or drops in timeseries. To do so, one could employ one of the existing libraries for timeseries anomaly detection available in R (e.g. tsoutliers [2] or Twitter's library [3]), Java (e.g. Yahoo's EGADS [4]), Python, or Matlab—see [5] for a collection of more than 10 actively maintained anomaly detection software packages. Still, one would need to provide the boilerplate code for injecting the data into the anomaly detection system, transforming the data in the format expected by each library, and filling in missing values (since some anomaly detection methods that rely on timeseries prediction do not work with gaps in the timeseries). Finally, once an anomaly is detected, it would need to be saved to a persistent store for follow-up analysis and the timeseries would need to be forwarded to next steps of the analysis pipeline, potentially after an on-the-fly replacement or filtering of the anomaly.

To automate some of the above steps and allow for smoother integration of anomaly detection in data monitoring and analysis, anomaly detection platforms have emerged. Skyline is a prominent example built on top of Graphite, an enterprise-ready monitoring tool. Other examples are Red Hat's Prometheus Anomaly Detection (PAD) built on top of Prometheus, another enterprise-ready monitoring tool, and Elastic's Anomaly Detection module built on top of Elasticsearch and Kibana. Although they are mostly based on open-source software components, such platforms are either not free (Elastic) or depend on specific technologies (Skyline and PAD) for monitoring and storage. An additional challenge is that such platforms can be difficult to both set up and extend with new anomaly detection methods.

In response, we created a toolbox for timeseries anomaly detection that can act as a platform that automates data injection, transformation, and other preprocessing steps without depending on a particular monitoring tool and modularizes the anomaly detection process in a number of clearly defined components. Its modular design makes the platform easy to configure for analyzing different kinds of timeseries and to extend with new anomaly detection methods. Our toolbox is tailored for realtime detection in the order of hundreds of milliseconds and allows for both online training and re-training of machine learning models.

In this paper, we describe the microservice architecture of our toolbox and explain how to deploy it for obtaining an out-of-the-box solution for realtime anomaly detection out of ready-to-use components. For detecting anomalies, our toolbox currently relies on three different methods: a method based on Exponential Weighted Moving Average, a method based on Convolutional Neural Networks and a method based on seasonality detection. We also briefly describe our experience with using the toolbox, and provide an initial assessment of its performance.

II. Anomaly Detection on Timeseries

Anomaly detection in timeseries data is not a new topic. Over the years, a number of approaches for timeseries anomaly detection have been proposed, ranging from simple threshold-based approaches to involved ones that learn a model of the normal behavior based on past data, predict the next value, and compare the prediction to the actual value to flag the value as anomalous if it is too far from the prediction. Such *prediction-based* approaches rely on different techniques for timeseries analysis and forecasting, such as simple exponential smoothing, Holt-Winters, Auto Regressive Integrated Moving Average (ARIMA), Long-Short Term Memory (LSTM), and Convolutional Neural Networks (CNN) [6]. Our toolbox can accommodate different anomaly detection approaches. We describe here the three approaches (the first two are prediction-based) that are shipped together with the toolbox and can be readily used. We chose to implement the specific approaches since they are quite different to each other and showcase that our toolbox can accommodate a range of approaches.

a) Predictions via Exponential Weighted Moving Average (EWMN): Having observed the values of d consecutive points in a timeseries at time t, a simple way to predict the value of the next point is to calculate the Exponentially Weighted Moving Average ($EWMA_t$), also known as Exponential Moving Average, of the d points and assume that the value of the point at time $t+1$ is equal to $EWMA_t$. This way, the noise around a value that captures the normal behavior can be removed. While the simple moving average calculates the mean of all values within a period, EWMA weights recently seen values higher than old ones. In particular, the weight factors decrease exponentially. This is achieved with the formula $EWMA_{\mathrm{t}} = V_t * a + EWMA_{t-1} * (1-a)$, where V_t is the value of the point at time t, and a is a smoothing factor with values in [0,1] that controls the impact older events have in the calculation (with higher values discarding older events faster). By default, a in our toolbox is set to $\frac{2}{1+d}$, where d is the number of past points considered, and EWMA is initialized by setting $EWMA_1 = V_1$. Applying EWMA is technically equivalent to learning an ARIMA (0,1,1)-without-constant model or performing simple exponential smoothing. A limitation of using EWMA for timeseries forecasting is that it cannot deal well with timeseries containing trends and seasonality. In such cases, higher-order exponential smoothing approaches (double exponential smoothing, Holt-Winters) can be used. Advantages of EWMA are its simplicity and its minimal memory requirements: its calculation only needs the previous value ($EWMA_{t-1}$) to be kept.

b) Predictions via Convolutional Neural Networks (CNN): Instead of performing timeseries forecasting using techniques from the exponential smoothing family, which are specific to timeseries data, a number of learning approaches based on neural networks have been proposed [7]. Both recurrent neural networks (RNNs), and their specialization in LSTMs, and CNNs can be used for capturing the evolution patterns in timeseries, with the latter showing better performance [8]. CNN is a type of neural network, originally proposed for image analysis and object recognition, that consists of several layers that include convolution layers, pooling layers, and fully connected layers. While convolutional layers extract features from the data by moving convolution filters in a predefined window, pooling layers take the results of the one or more convolutional layers as input and extract the most important features. The idea of applying CNNs for timeseries prediction is to learn filters capturing repeating patterns in the timeseries (treated as one-dimensional image) and use them to predict future values. An advantage of CNNs over recurrent-type networks is that they are more efficient to train [8]. Still, their training requires a large amount of timeseries data. Compared to EWMA, CNN-based prediction has two distinct phases: *training*, where data is collected and a CNN is built, and *operation*, where the CNN model is used to make a prediction given a sequence of n points. To keep the model performant, it may need periodic re-training.

c) Tunable Anomaly Detection Framework (TADF): A recently proposed framework for timeseries anomaly detection relies on the calculation of anomaly scores and thresholds with the intent to make the whole process easier to tune and interpret [9]. The framework focuses on the automated identification of the most effective seasonal pattern for anomaly detection and groups points according to that pattern. For example, in a timeseries with hourly measurements that has a daily seasonality, points are grouped in 24 groups, one for each hour of the day. Such groups are built for both the original timeseries and its differenced version. The framework assigns each point two scores according to the point's distance from the median in each group and selects a threshold for reporting anomalies via these scores in an interactive manner that involves expert users. Alternatively, as is also the case in our implementation, the framework identifies anomalies in an automated way via calculating the modified z score [10] of each point in each of the two groups it belongs to (corresponding to the original and the differenced timeseries). If either of the modified z scores of a point is more than a prescribed threshold (default is 3.5), then the point is considered an anomaly [10]. The time-consuming part in this approach is the identification of the prominent seasonality, the creation of corresponding groups, and the calculation of the median and MAD. These steps correspond to the "training" phase of the approach and have to be periodically performed. On the contrary, the calculation of modified z scores can be done online for each new point.

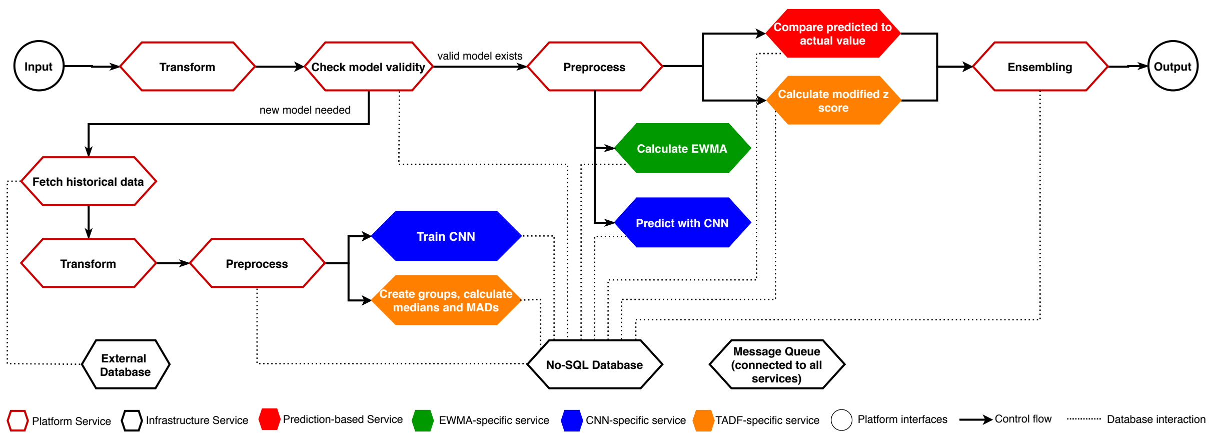

Fig. 1. Architecture of the toolbox.

III. Toolbox

The requirements of our anomaly detection toolbox are to (1) automate common tasks such as data ingestion and transformation, (2) allow for different anomaly detection techniques such as the ones described in Section 2 to be plugged in, and (3) work as a platform for realtime detection (in the order of hundreds of milliseconds) of anomalies in data streams.

To meet these requirements, our toolbox has a modular architecture that consists of a number of Python services packaged as docker containers (Figure 1). A number of these services are reused by all anomaly detection techniques ("Platform Services"). Others can be reused by only a group of such techniques. For instance, "Prediction-based Services" are only to be reused by techniques that rely on predictions, i.e. EWMA and CNN. There are also services that are specific to an anomaly detection technique, e.g. "Train CNN" is specific to CNN while "Calculate modified z score" is specific to TADF. Finally, there are infrastructure services that support the persistence of data in a NoSQL database (we use MongoDB) and allow the communication between services via a message queue (we use RabbitMQ).

Instantiating the toolbox services creates a platform for anomaly detection. Input to the platform are data streams obtained either via reading from an external database (we currently support MongoDB and ElasticSearch) or pulling data via HTTP calls. Similarly, the output of the platform are data streams with identified anomalies. When a new data point enters the platform, it first gets transformed from its JSON or XML format to a Pandas Dataframe to facilitate future operations ("Transform" service). Next, the "Check model validity" service checks whether there is a valid prediction model for the data stream the point belongs to. This can be done e.g. by considering the time or events passed since the last training activity took place. Since a data stream in our toolbox can be associated with many models, the validity of all associated models is checked here. If a model is not valid and needs re-training (because it is outdated or because not enough observations are yet gathered), the lower part of the flow in Figure 1 gets activated, i.e. the training phase.

In the training phase, first, historical data are fetched via an external database. In the current implementation, we rely on the existence of such a database since we do not save the raw data in our platform. Once the data are fetched, they are transformed to the expected form using the same service template as before ("Transform") and forwarded to the "Preprocess" service. This service performs a number of important tasks common to many anomaly detection methods such as removing invalid values, padding missing values, scaling, extracting features from data, and filtering based on time or content windows. Preprocessing can be configured per data stream and per model in our toolbox. The next step is specific to the anomaly detection method at hand. In the CNN-based approach, a dedicated service invokes the training of a CNN that includes hyperparameter tuning, while in TADF a dedicated service is responsible for identifying the most effective seasonality, creating the corresponding groups, and calculating the median and MAD per group. The resulting models are saved in the internal database.

If a valid model is found, the flow continues to the top right part of Figure 1. Here, the preprocessing service is invoked and prepares the new data point for the later stages. Since the same service template is invoked here as in the training phase, we ensure that new data are preprocessed in the same way as older data used in model creation. At this point, the flow splits into two parallel flows. First, if needed, a prediction service is executed, as in the case of EWMA and CNN. The "Calculate EWMA" and "Predict with CNN" services retrieve the corresponding model (which in the case of EWMA is simply the previous EWMA value) and use it together with the latest point to predict the value of the next point. The predictions are saved in the database to be used once the next point arrives. Second, the services that perform the anomaly check are invoked. In case of prediction-based methods, our toolbox provides a reusable service which retrieves the latest prediction from the database and compares it to the actual value of the latest point. If the prediction is far from the actual value (based on a configurable threshold), the point is marked as an anomaly by the corresponding method. In TADF, which

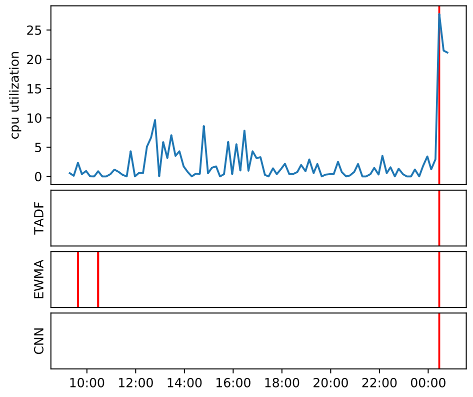

Fig. 2. Graphical output showing anomalies identified by CNN, TADF, EWMA, and at the ensembling phase (top box).

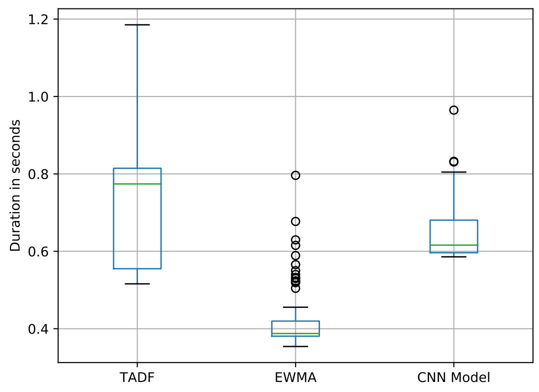

Fig. 3. End-to-end latency measurements for the three implemented methods. Each boxplot contains 100 samples.

does not rely on predictions, a dedicated service is provided that assigns the new point to a group, retrieves the metadata of its group, calculates its modified z score, and marks the point as an anomaly according to the result. The results from all the anomaly checking methods are saved in the internal database.

As a last step, a platform service ("Ensembling") retrieves the results from the different methods and combines them to make a final decision about whether a point should be marked as an anomaly. Although such ensembling logic can be very complex, in our first attempt we use a simple majority voting which outputs a point as an anomaly if half or more of the methods associated with its data stream indicate that it is.

IV. Experiments and Lessons Learned

We have deployed and used the platform at DXC Technology in identifying anomalies in data streams coming from performance measurements of application servers, in particular, CPU and memory utilization, and other OS-level metrics measured at a 10-minute frequency. At DXC Technology, the platform is used as part of a smart and adaptive monitoring system that allows customers to gather, visualize, and analyze key metrics of their infrastructure.

We have used all three implemented methods to identify anomalies after configuring them in the following way. The CNN model was trained with 3 months of data (~13000 points); the same amount was used in preparing the groups for TADF. For EWMA, a window size of 1 day (144 points) and a smoothing factor of 0.5 was used. In predicting with CNN, we gave a sequence of the last 12 points to the trained regressor. In our preliminary experiments, we first inspected that the ensembling service is indeed providing meaningful results, i.e. points that can be considered actual anomalies. As a sanity check, we also plotted the individual anomaly checks side by side (Figure 2). We then measured the end-to-end latency of the platform: We logged the time elapsed between the beginning of the first service ("Transform") until the end of the service that performs the anomaly check for each method ("Compare predicted to actual value" for CNN and EWMA and "Calculate modified z score" for TADF). The tests were executed on a Windows 10 with Intel Core i7 2.70 GHz and 32 GB RAM. Our results indicate that all three methods are able to indicate anomalies in subsecond latencies (Figure 3).

V. Conclusion

In this paper, we described a ready-to-use toolbox to efficiently detect anomalies by consolidating different methods. While at the current state of the tool three methods (EWMA, CNN and TADF) are provided, more methods can be flexibly added due to the modular architecture of the platform and the possibility of efficiently reusing its platform services across different methods. Our toolbox serves as a plug-in solution for architecting and development of smart systems that have to adapt their behavior at runtime.

Acknowledgement

This work was partially supported by the Czech Science Foundation project 20-24814J.

References

[1] Z. Zhao, M. Chen, D. Matheson, and M. Stone, "Online experimentation diagnosis and troubleshooting beyond aa validation," in *DSAA 2016*. IEEE, Oct 2016, pp. 498–507.
[2] J. López-de Lacalle, "tsoutliers: Detection of Outliers in Time Series," https://CRAN.R-project.org/package=tsoutliers, 2019.
[3] Twitter, Inc and other contributors, "AnomalyDetection R package," https://github.com/twitter/AnomalyDetection, 2015.
[4] N. Laptev, S. Amizadeh, and I. Flint, "Generic and Scalable Framework for Automated Time-series Anomaly Detection," in *KDD'15*. ACM, 2015, pp. 1939–1947.
[5] Roberto, "rob-med/awesome-TS-anomaly-detection," Jan. 2020, accessed: 2020-01-08. [Online]. Available: https://github.com/rob-med/awesome-TS-anomaly-detection
[6] A. v. d. Oord, Dieleman *et al.*, "WaveNet: A Generative Model for Raw Audio," *arXiv:1609.03499 [cs]*, Sep. 2016.
[7] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," in *ICACCI 2017*, Sep. 2017, pp. 1643–1647.
[8] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional Time Series Forecasting with Convolutional Neural Networks," *arXiv:1703.04691 [stat]*, Sep. 2018.
[9] M. R. Alam, I. Gerostathopoulos, C. Prehofer, A. Attanasi, and T. Bures, "A framework for tunable anomaly detection," in *ICSA 2019*, March 2019, pp. 201–210.
[10] B. Iglewicz and D. C. Hoaglin, *How to detect and handle outliers*, ser. ASQC basic references in quality control. Milwaukee, Wis: ASQC Quality Press, 1993, no. v. 16.