32nd European Modeling & Simulation Symposium
17th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-44-7 © 2020 The Authors.
DOI: 10.46354/i3m.2020.emss.044

Towards a Stream based Machine Learning Testbed

Jan Zenisek[1,2*], Josef Wolfartsberger[1], Norbert Wild[1] and Michael Affenzeller[1,2]

[1]Center of Excellence for Smart Production, University of Applied Sciences Upper Austria, Softwarepark 11, Hagenberg, 4232, Austria
[2]Institute for Formal Models and Verification, Johannes Kepler University, Altenberger Straße 69, Linz, 4040, Austria

[*]Corresponding author. Email address: jan.zenisek@fh-hagenberg.at

Abstract

With the rise of data analytics in industrial applications a heterogeneous tool landscape developed over the past few years. To cope with highly dynamic and domain-specific requirements of such applications, scripting programming languages and frameworks, which offer ecosystems comprising numerous publicly available plugin libraries, are gaining more and more attraction as a starting point, since they enable rapid prototyping and thus, quick results. At the other end, cloud service providers are continuously extending their data analysis product palette in order to support large enterprise solutions for real-world deployments. As scripted prototypes and cloud based solutions have their strengths in different phases of an analytics project, we identified several pitfalls in recent case studies when moving a prototypic approach to release. In this work, we present a software design for a data stream analysis testbed, with the aim to address some of these challenges. Therefore, an interaction pattern for common analysis steps (data acquisition, visualization, preprocessing and machine learning model evaluation) is detailed, results gained from a sample case study are summarized and future leads are discussed.

Keywords: Machine Learning, Data Stream Analysis, Software Framework, Predictive Maintenance

1. Introduction

The motivation for this work originates from the recently intensively investigated idea of predictive maintenance (Schmidt and Wang, 2018). Therefore, machine learning methods are used to generate models, which detect and furthermore, predict machinery breakdowns. The model training algorithms use recorded sensor data streams of monitored machinery, which are categorized as being representative for certain system states by domain experts. To develop accurate and well generalizing models, however, usually not the complete raw data stream can be used. For most applications, faulty system behavior is very seldom, and hence, underrepresented in the recorded data. Therefore, significant effort has to be put on data preprocessing to extract representative partitions from the variety of states a system runs through. With a balanced system state display, machine learning algorithms are able to model all shades of a system's behavior comprehensively, which eventually enables precise state predictions (Orriols and Bernadó-Mansilla, 2005). To evaluate a model's accuracy, built on such training data sets, an equally engineered test partition, which is restrained from the previous modeling process, is commonly used. However, there are several differences between evaluating a model on a test partition and on the raw real-world data stream. We have experienced additional challenges

which revealed at the deployment level in recent real-world data analytics use cases with reference to predictive maintenance and Industry 4.0, we have worked on:

- **Data preprocessing:** There are several data-linked challenges regarding online preprocessing: consolidation of data streams from different origins; temporal alignment of data streams; standardization of data from differently situated systems etc.
- **Runtime performance:** The runtime performance of data preprocessing and model evaluation on a continuously updated data stream is crucial, since it has to be performed and finished in time according to the stream update frequency. Also accompanying visualizations and performance indicators (i. e. descriptive statistics), used for decision support, must be computable in time.
- **Model sensitivity vs. specificity:** In the context of a predictive maintenance application, it can be quite difficult to find a reasonable tradeoff between the concurring goals "high anomaly detection speed" and "low false positive rate", and tune the models accordingly.

Tackling such challenges which emerge from an online evaluation situation requires to mimic the real-world situation as close and as fast as possible. Before setting up a large-scale data analytics environment (e. g. using commercial cloud service platforms), usually prototypes in form of software scripts are developed, which make use of existing frameworks and packages. However, with the increasing functionality, necessary to simulate the real-world situation, such scripts quickly tend to turn into a confusing, slow and error-prone *big ball of mud*, which is eventually discarded, at latest after a use case is closed. In order to prevent starting from scratch for each similarly situated use case over and over again, we aim to persist our gained experiences with the creation of a data stream based machine learning testbed. This work depicts the general idea and architecture of the developed testbed and details the role of the participating components. Besides being a booster for upcoming applications, such a testbed may serve as a demonstration platform, used for early-stage discussions regarding possible data analytics employments in production industry.

In the subsequent section 2, related approaches are listed and open issues highlighted. The design of the developed testbed and hence, the main part of this contribution, is presented in section 3. Results from a synthetic application scenario are provided in section 4. The final section 5 briefly summarizes the main contribution of this work and provides an outlook.

2. Related work and unmet Needs

In the context of Industry 4.0 and more specifically, predictive maintenance, numerous software frameworks have been proposed in recent scientific work. Li et al. (2017) as well as Schmidt and Wang (2018) present cloud based frameworks for predictive maintenance applications. The authors profoundly elaborate on the challenges from real-world applications and how their frameworks are designed to deal with them on a conceptual level. However, technical implementation details, such as the communication protocol, are not further described, instead the machine learning methodology for predictive maintenance is detailed. Several other contributions present systems, tailored for a specific cooperate partner (e. g. Kanawaday and Sane (2017), Baptista et al. (2018)). These systems accomplish real-world problem solving, but often lack of general applicability. Most importantly however, as they are developed for production use cases, they are not easy to deploy and hence, suboptimal to serve as a testbed.

Cloud based machine learning platforms, such as Microsoft Azure Stream Analytics (https://azure.microsoft.com/), IBM Watson (https://www.ibm.com/watson) or RapidMiner (https://rapidminer.com/), ship with proper solutions for prototyping data stream analytics, as they provide ready-to-use infrastructure, eased setup and a broad range of stream mining features. However, since they are not exclusively dedicated to Industry 4.0 applications, exactly this richness of features may slow down the prototyping process. Another aspect to consider is that these systems are commercial products, such that ordering them solely for being a testbed might not justify the costs. Moreover, if used as a prototyping tool, one might risk an expensive vendor lock-in, right at the start of a project. Their main purpose is to provide enterprises a solid platform for eventual real-world deployments (i. e. production release).

Alternatively, the open source WEKA MOA framework (Bifet et al., 2010) does fulfill several needs, with zero costs. It ships with a large tool box for data stream mining, it is open for modification, it interoperates with its parental framework WEKA and it has a growing community. Most importantly, the framework covers necessary testbed functionality including the generation of synthetic data streams, data preprocessing and model training with various machine learning algorithms. Nevertheless, from the author's perspective, the configuration process via graphical user interface or command line argument is quite cumbersome, although this may be adapted quite easily. The major unmet need for the envisaged testbed however, is that MOA is not designed to simulate an industrial condition monitoring setup. The framework is certainly a great software suite for experimenting with machine learning modeling and evaluation algorithms, however, does not provide the pursued simulation characteristic.

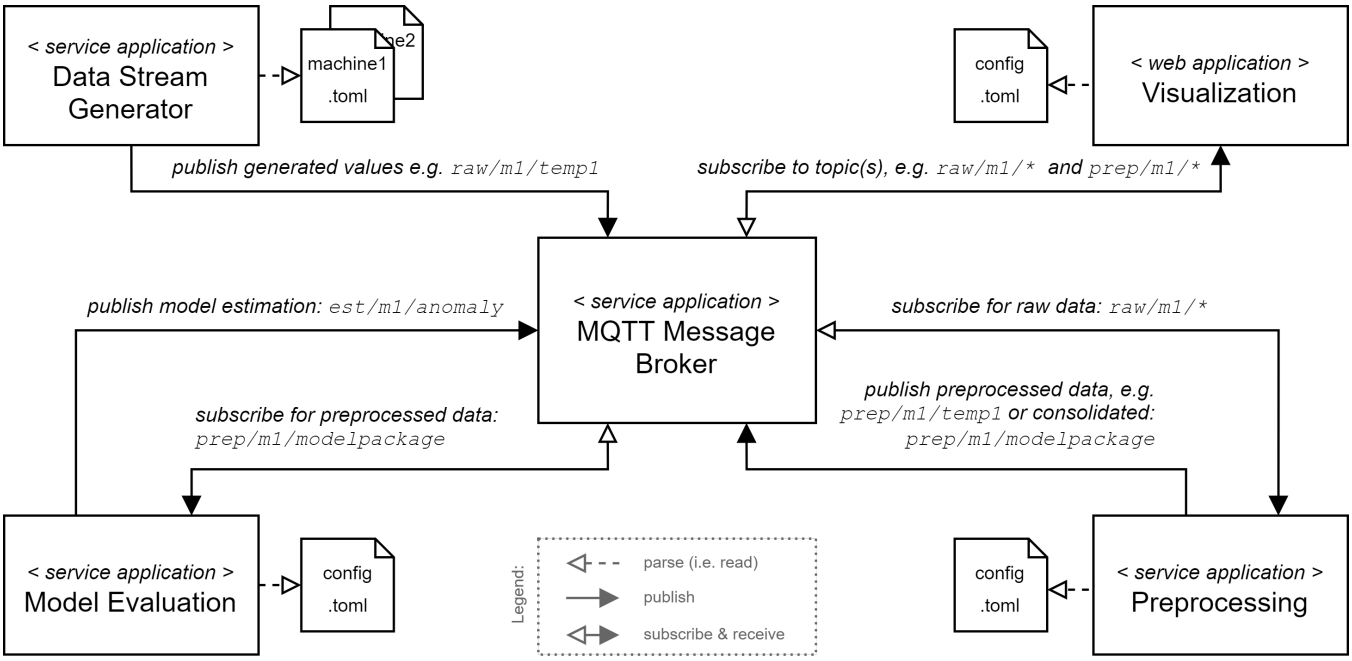

Figure 1. General design of the developed MQTT based data stream analytics testbed.

To the best of the authors' knowledge there is no dedicated machine learning model evaluation environment, which aims to mimic a real-world situation, while still maintaining an easy-to-use testbed character.

3. Design and Functionality

In order to cover the unmet needs, described in the latter section, we propose a software design with focus on the communication process between the participating components. The following paragraphs elaborate on the core ideas behind the implemented software environment and how they contribute to the aimed data stream evaluation testbed.

3.1. General Architecture

The central role in the design of the developed testbed is taken by an MQ telemetry transport (cf. MQTT) protocol based message broker (Katsikeas et al., 2017), for which we used the implementation of Pivotal's RabbitMQ software (https://www.rabbitmq.com/). MQTT is a standardized ligthweight network protocol, which is used for many IoT applications and widely adopted by leading cloud service providers (cf. *Microsoft Azure*, *Amazon Web Services* etc.). The MQTT broker routes messages following the publisher-subscribe pattern: Communication participants produce and direct messages to a listening MQTT broker, using a URL-like structured topic-string. Other participants may subscribe for certain topics or subtopics they are interested in and want to get updated on. The broker queues the received messages, manages the subscriptions and forwards messages from the message publishing party to the message consuming party. Figure 1 illustrates the basic design and functionality of the developed data stream evaluation testbed from a data transfer perspective. The software can be best described as configuration file driven discrete event simulation and consists of the central MQTT message broker and four communicating simulation participants (cf. data stream generator, visualization, preprocessing, model evaluation), which are individual service applications. Although the data transfer within such a stream analytics testbed may be accomplished leaner without separate services and a central message broker – e. g. by using peer to peer data transfer between applications, or by implementing the complete functionality within a single application – we opted for this MQTT based approach, since the protocol already deals with many real-world data transfer problems (e. g. expandability, scalability, reliability, security etc.) and hence, became one of the most important protocols for Industry 4.0 applications.

As a second core design idea, we decided to use configuration files to initialize the simulation. Therefore, all necessary parameters (e. g. topic subscriptions) of a simulation participant are parsed at startup from a *toml* file (cf. tom's obvious minimal language; https://github.com/toml-lang/toml). The very concise toml syntax basically consists of <key>=<value> pairs, but also allows more complex data structures. For the sake of simplicity, uniformity and minimal transmission overhead, we used this format also for the message payload.

3.2. Simulation Participants

A sample message flow between the four simulation participating components, is annotated in Figure 1 at the connecting arrows and reads as follows (clockwise): The data stream generator parses configuration files, in which several time series are mathematically defined and meta-information, such as topic and frequency for publication are determined. The generator starts to calculate the time series and continuously publishes each new generated data point independently (cf. first-level-topic *raw*). Depending on the generators configuration, data streams from a single machinery or a complete production plant may be simulated. Furthermore, the generator can be configured to produce data with drifting concept, in order to simulate gradually increasing wear and tear of machinery and hence, simulate sample predictive maintenance use case data. Besides streaming data points, the generator may also be used to produce complete data sets for the purpose of model training during an "offline" phase.

The subscribed preprocessing service application reads the data and performs several modification steps (filter outliers, feature aggregation, consolidate streams etc.), which are defined in the respective configuration file. The preprocessed data is subsequently published under a new first-level-topic (cf. *prep*).

The visualization component subscribes to both, raw and preprocessed data streams, aligns them and depicts the time series in an interactive chart for further analysis (Figure 2). Same as the data stream generator, parts of the web application for stream visualization originate from previous work (Zenisek et al., 2018) and have been enhanced for this testbed.

Finally, the model evaluation application reads from the preprocessed data stream and continuously feeds previously trained machine learning models with data updates. The classification result is published to another MQTT topic (cf. *est*) for further interpretation. Eventually, this information may be used by a decision support system at the stream-producing machinery or production plant, where it triggers maintenance actions. For model training and evaluation we used the open source framework HeuristicLab (Wagner et al., 2014), since it provides a broad range of machine learning algorithms and it is easy to extend and include, due to its plugin infrastructure.

4. Case Study

For a first test implementation of the presented framework, we re-purposed the methodological approach and the synthetic problem from Zenisek et al. (2019): This work presents a machine learning based algorithm, which utilizes continuously updated variable interaction networks (VIN) to detect changing behavior in a technical system, which might give indication for necessary maintenance actions. Therefore, time series from a synthetic system of communicating vessels are generated. To create a realistic system drift, i. e. a potential maintenance scenario, a gradually clogging communication channel is simulated and reflected by the data. Although the evaluation of the generated time series is performed based on a sliding window, each of the necessary data stream processing steps – i. e. data acquisition (read next data records from file), preprocessing, machine learning model evaluation (= creation of new VIN), VIN based concept drift detection – are performed time-serially in the original work, using a single process.

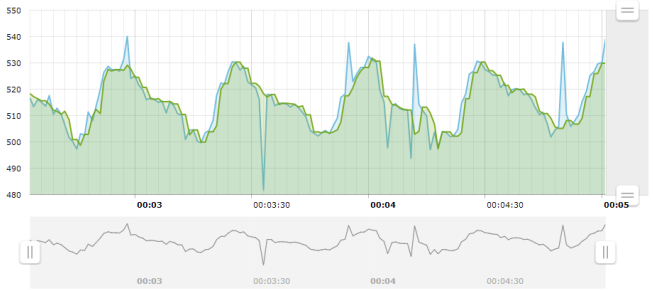

Figure 2. Visualization of the raw (blue) and the overlaid preprocessed (green) sensor time series. Note the smoother, filtered characteristics of the preprocessed signal.

4.1. Architecture Adoption

In order to adopt the software design, presented in section 3, for the described test problem, the aforementioned steps have been split up as follows: While in the original implementation, data processing steps were triggered by simply reading one record after another from a previously compiled test data set, the data acquisition part is now covered by the Data Stream Generator component. It produces time series data points according to calculation rules in the provided toml-files and hence, simulates the sensor signals online. For each of the defined signals an individual MQTT topic is used to publish data updates with a fixed interval of one second. In the basic version four time series are simulated to describe the system of communicating vessels and its controlled run from a *healthy* to an *erroneous* condition state. The generation process includes normally distributed signal noise and shock terms, making the signal simulation non-deterministic and thus, more realistic. Furthermore, we enhanced the existing web application with an additional, interactive graph visualization, which is depicted in Figure 3. Besides application-specific data preparation (here: basic outlier filtering, signal smoothing using a moving average), the main task of the preprocessor component is to consolidate the data streams (i. e. signals) to be able to forward one data package to the evaluator. This subsequent component reads the preprocessed data package and performs the evaluation of previously (i. e.

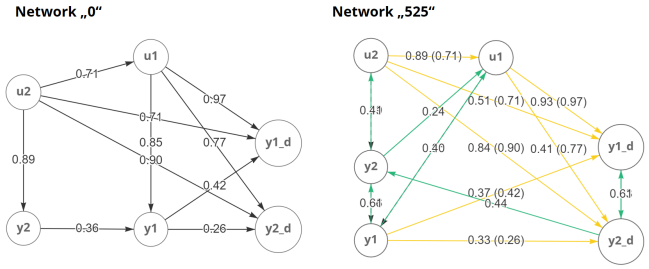

Figure 3. Visualization of variable interactions as acyclic directed graph from the web application simulation participant. Nodes represent variables, edges represent the variables' impacts on others. The left network represents the initial system state, the right network shows the system during an ongoing concept drift (green = new edge, yellow = impact change, values in parenthesis = initial impact value).

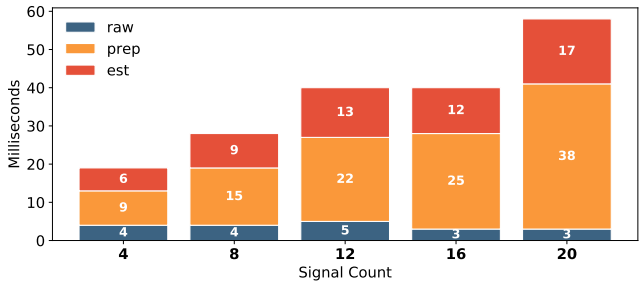

Figure 4. Data transmission and processing time of simulation components, gained from experiments with increasing signal count.

offline) trained machine learning regression models to calculate the alternating variable impacts and thus, the creation of a VIN. Subsequently, the evaluator computes a (mathematical) comparison of the initial and updated graph structures (cf. Figure 3). Eventually, the resulting comparison value is evaluated against a pre-defined threshold, which enables the targeted concept drift detection. For more information regarding methodological and test problem details, the reader is referred to Zenisek et al. (2019).

4.2. Performance Results and Discussion

The results regarding detection accuracy equalled those gathered in the original study, i. e. the implementation was successful. However, in order to dig deeper and show the added value and potential of using the presented architecture, we conducted a series of experiments with the test problem and evaluated the runtime performance as summarized Figure 4. All test runs have been performed on a standard desktop workstation with an Intel Core i7-6600U CPU 2.60 GHz (2 physical cores with hyperthreading enabled), 16 GB 2133 MHz RAM. In order to challenge the computational resources we increased the problem size gradually by adding more signals to transfer and process. Each of the simulation participants described above is started as an individual process on the same workstation. Parallelization options (e.g. preprocessing of data streams in parallel) have been disabled. The processing times are calculated as follows:

- *raw* = average transmission time of one signal update from data stream generator to preprocessor component; the time to synthesize a new data point is not included, since in a real-world situation, this represents the physical measuring process and is without the scope and influence of data analytics.
- *prep* = time to consolidate an update of all signals, preprocess and transmit them to the evaluator component
- *est* = time evaluate the machine learning models, interpret the computed VIN-comparison value and transmit the result to a subsequent component.

While the transmission time of *raw*-packages remains quite stable regardless of the amount, the *prep*-package time grows almost linear with increasing signal count. Hence, the simulated package traffic can be easily handled by the MQTT broker, without significant deviations. Since the preprocessor performs only, two minor time series modifications (cf. basic outlier filtering and signal smoothing), with almost no measurable runtime effort, we identified the signal consolidation task as main cause for the time increase. Also the *est*-package time increases along with the signal count, however much slower, since the evaluated machine learning models are of minor complexity and do not employ all available variables. In this synthetic problem, data acquisition is a bottleneck: Since the variables of the simulated system are closely interdependent, new data points are sampled in a particular, fixed order and thus, the generation process can not be parallelized, without changing the system behavior. However, also in real-world applications, sensor measurements from a single system are usually performed and published serially, due to hardware limitations of the connected microcontroller and minor software complexity. Regardless of the data source and a serial or parallel publication, the consolidation of data updates the actual bottleneck of this architecture, since a central data collection for the subsequent evaluation component is necessary. Facing the measured times, however, one can clearly observe that all of the exemplary package roundtrips (max. ca. 60ms) are easily finished within the defined update interval of 1000 milliseconds. As real-world applications may comprise far more signals to handle, alternative strategies may be necessary to pool data:

- use more powerful hardware for components with potential bottlenecks
- perform interval based consolidation and fill missing updates with past values
- pre-aggregate signal values at the machinery microcontroller
- reduce measuring frequency

The goal of this first setup, was to demonstrate basic behavior and show potential performance bottlenecks of the presented architecture. However, further tests may include larger setups, similar to possible real-world deployments. Therefore, service scaling (e. g. multiple MQTT broker nodes or multiple evaluation components) and service distribution (e. g. one compute resource for each service) are easy to realize with our design, since only the service addresses must be modified. Another advantage of our approach, relies in its plugin simulation character: For instance, visualization – as non-essential component – may be switched on/off without interfering the main data stream analysis process. Furthermore, additional simulation participants, such as a decision support system, can be easily appended. This perfectly facilitates the aimed iterative-incremental development process (cf. strength of scripting approaches) and mimics a feasible real-world deployment (cf. strength of cloud solutions).

5. Conclusion and Outlook

In order to deal with challenges regarding data stream analytics, motivated by, but not restricted to predictive maintenance applications, we presented a software architecture for a simulation environment on the base of the MQTT protocol and toml configuration files. The goal and main contribution of this work was not to provide a full-fledged stream mining tool box, rather than presenting a sound, yet simple software design for a data stream evaluation testbed for the purpose of prototyping, considering real-world deployment challenges. Future work might enhance the presented approach by dealing with the following points:

- Adopt additional machine learning frameworks in order to support their models (e. g. WEKA *.model-files) for evaluation.
- Perform comprehensive applicability tests with real-world case studies
- Extend the testbed towards online and continuous model training

6. Acknowledgements

The work described in this paper was done within the project “Smart Factory Lab” which is funded by the European Fund for Regional Development (EFRE) and the state of Upper Austria as part of the program “Investing in Growth and Jobs 2014-2020”.

References

Baptista, M., Sankararaman, S., de Medeiros, I. P., Nascimento Jr, C., Prendinger, H., and Henriques, E. M. (2018). Forecasting fault events for predictive maintenance using data-driven techniques and arma modeling. *Computers & Industrial Engineering*, 115:41–53.

Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., and Seidl, T. (2010). Moa: Massive online analysis, a framework for stream classification and clustering. In *Proceedings of the First Workshop on Applications of Pattern Analysis*, pages 44–50.

Kanawaday, A. and Sane, A. (2017). Machine learning for predictive maintenance of industrial machines using iot sensor data. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 87–90. IEEE.

Katsikeas, S., Fysarakis, K., Miaoudakis, A., Van Bemten, A., Askoxylakis, I., Papaefstathiou, I., and Plemenos, A. (2017). Lightweight & secure industrial iot communications via the mq telemetry transport protocol. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 1193–1200. IEEE.

Li, Z., Wang, Y., and Wang, K.-S. (2017). Intelligent predictive maintenance for fault diagnosis and prognosis in machine centers: Industry 4.0 scenario. *Advances in Manufacturing*, 5(4):377–387.

Orriols, A. and Bernadó-Mansilla, E. (2005). The class imbalance problem in learning classifier systems: a preliminary study. In *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*, pages 74–78.

Schmidt, B. and Wang, L. (2018). Cloud-enhanced predictive maintenance. *The International Journal of Advanced Manufacturing Technology*, 99(1-4):5–13.

Wagner, S., Kronberger, G., Beham, A., Kommenda, M., Scheibenpflug, A., Pitzer, E., Vonolfen, S., Kofler, M., Winkler, S., Dorfer, V., et al. (2014). Architecture and design of the heuristiclab optimization environment. In *Advanced methods and applications in computational intelligence*, pages 197–261. Springer.

Zenisek, J., Kronberger, G., Wolfartsberger, J., Wild, N., and Affenzeller, M. (2019). Concept drift detection with variable interaction networks. In *International Conference on Computer Aided Systems Theory*, pages 296–303. Springer.

Zenisek, J., Wolfartsberger, J., Sievi, C., and Affenzeller, M. (2018). Streaming synthetic time series for simulated condition monitoring. *IFAC-PapersOnLine*, 51(11):643–648.