A Framework for Preprocessing Multivariate, Topology-Aware Time Series and Event Data in a Multi-System Environment

Andreas Schörgenhumer*, Mario Kahlhofer*, Peter Chalupar*, Paul Grünbacher†, Hanspeter Mössenböck‡
* Christian Doppler Laboratory MEVSS, Johannes Kepler University Linz, Austria
andreas.schoergenhumer@jku.at
† Institute for Software Systems Engineering, Johannes Kepler University Linz, Austria
‡ Institute for System Software, Johannes Kepler University Linz, Austria

***Abstract*—Monitoring and predicting quality properties of complex systems relies on collecting and analyzing huge amounts of data at run time. Machine learning is frequently adopted to analyze time series and event data, often coming from multiple systems. In such a context, extracting and preprocessing data is an essential but also highly tedious task. In this paper, we thus present an offline preprocessing framework that can handle multivariate time series and event data in a multi-system environment that also takes the system's topology into account. After a discussion of the key requirements, we present the architecture and implementation of our highly configurable and easy-to-use framework. We demonstrate how the framework allows to extract data and to yield output files for machine learning via configuration settings. In a two-step evaluation, we investigate the framework's usefulness and scalability. We demonstrate the usefulness in an event prediction case study of real-world multi-system time series data. Our results show the significant impact of different data preprocessing settings on machine learning. Our experiments further demonstrate that processing performance scales linearly with respect to the number of systems and time series.**

I. Introduction

In high assurance systems engineering, engineers need to collect and analyze massive amounts of data during operation to determine if a system still meets its requirements or to prevent and predict system failures. In particular, data processing is an integral and challenging task when developing machine learning applications for this application context [1]–[3]. Tools and frameworks assisting in data handling are therefore highly needed to lower the amount of code engineers have to write, to reduce errors, and to increase performance through automation.

For example, in the domains of performance monitoring and predictive maintenance [4]–[9] time series data from various interconnected sources is analyzed to predict critical events, therefore assuring a system's proper functioning. Many researchers have proposed methods, frameworks and workflows for analyzing univariate and multivariate time series data in preparation for machine learning. Existing approaches support data preprocessing, visualization or even provide full tool pipelines combing different techniques [8]–[15]. While these approaches are already complex on their own, additional research challenges arise when dealing with data from multiple systems [16]–[18]. Specifically, by *system* we mean a set of *entities*, each providing zero or more time series. For example, a system could comprise various sets of hardware and software components in a data center or different sets of machines and associated sensors in an industrial environment. The system's *topology* shows how these entities are connected and indicates, e.g., data flows or other communication links.

Despite some work also considering a system's topological view [9], there are currently no general frameworks for dealing with different topologies and time series data in a multi-system environment. To the best of our knowledge, existing research has not yet investigated frameworks or workflows capable of preprocessing multivariate time series data in a multi-system setting that also consider the dynamically changing system topologies. We expect that a highly configurable framework will improve the rate at which developers can analyze data and train machine learning models by reducing the overhead of the preprocessing step, thus enabling a rapid and iterative workflow. Such an assisted workflow is essential, since decisions made in the preprocessing step can have a high impact on the achievable results of the trained models [1].

In this paper, we first identify and discuss the requirements for a preprocessing framework that is capable of dealing with this kind of data. We propose an approach for implementing the framework and evaluate its feasibility on an industrial multi-system monitoring dataset to demonstrate its usefulness and the high impact of the preprocessing step. Furthermore, we evaluate our framework on a large data set to show its scalability and feasibility for big data problems. Specifically, our paper claims the following contributions:

(i) We propose a highly customizable, offline framework for preprocessing multivariate time series from multiple systems with different and evolving topologies.

(ii) We evaluate the practical usefulness, feasibility and scalability of our framework on an industrial dataset. We show that our framework provides useful results that can then be easily utilized as input for machine learning.

The rest of this paper is organized as follows: Section II describes our assumptions on the structure of the analyzed data. Section III describes the properties we expect from the preprocessing framework. Section IV explains our preprocess-

ing framework. In Section V, we evaluate our framework on an industrial dataset. In Section VI, we discuss related work. Section VII concludes this paper and outlines future work.

II. Data

In this section, we describe the key data elements we assume for multi-system analysis. Specifically, a system provides three different kinds of data:

(i) *Topology.* The topology defines the structure of a system's entities, i.e., its physical or abstract components (e.g., host, machine, sensor). Each entity must be mapped to exactly one entity type defining which other types an entity may be linked with. As systems evolve, the topology may change over time. Entities may have different lifetimes, starting when they become active and ending when they become inactive.

(ii) *Time series.* Each entity provides between 0 and n different time series, where n is determined by the entity type.

(iii) *Events.* Events are incidents relevant for run-time analysis that may occur at any entity (e.g., the slowdown of a service). An event only needs to hold information about its type, at what time it occurred, and which entity was affected.

For better understanding, we illustrate these definitions with an example. Figure 1 shows a tiny system with six entities: a_1 of type A, b_1 and b_2 of type B, c_1 and c_2 of type C and d_1 of type D. At some point between timestamps t_x and t_y, an event occurred at entity a_1. As described before, the topology may change, which is shown at timestamp t_y where b_1 became inactive. Each of the four entity types is associated with 0 to n time series, as listed in Table I. The table also includes concrete examples to provide a better understanding of what entities and time series might look like (cf. Section V). Each individual entity provides time series data specified by its type. Type A does not provide any time series data in contrast to the other three types. For instance, entities b_1 and b_2 of type B each provide two time series: B-1 and B-2. For b_1, however, the time series data are only available up to timestamp t_y.

Note that the entities of a system do not need to form a connected graph, i.e., individual entities or groups of entities may not be interlinked. The former is the case when topology data is missing. The latter often occurs in case of subsystems, i.e., groups of connected entities. It is also not required that each individual entity provides all time series specified by its type. For instance, in the above example, b_1 might only provide B-2. Finally, data does not have to originate from a single system only, but can also come from multiple independent systems, if they share the same entity types.

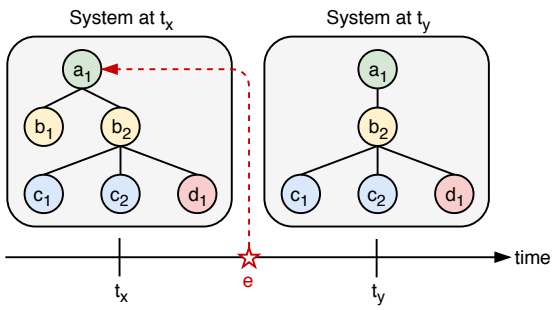

Fig. 1. Example of an event occurring at entity a_1 and a topology change where entity b_1 is no longer active at timestamp t_y.

TABLE I
Example for time series mapped to entity types.

Abstract		Concrete	
Type	**Time Series**	**Type**	**Time Series**
A	∅	Service	∅
B	B-1, B-2	Host	CPU, Memory
C	C-1	Disk	Space
D	D-1, D-2, D-3	Network	Sent, Received, Failed

III. Requirements

Bearing these assumptions about data and related work in mind, we identified five requirements for our data preprocessing framework:

R1. *Multi-system analysis* [16]–[18]. In real-world settings, it is common that data from multiple systems needs to be processed. The framework has to be capable of handling such a setup, especially considering that the involved systems may vary in size, which affects overall data balancing.

R2. *Topology support* [9]. The framework needs to be capable of resolving connections and mapping relevant events to interlinked entities, thus taking the structure of the system into account. This includes dealing with different lifetimes of entities.

R3. *Techniques for data selection and sampling* [10], [19]. Entities may have multiple time series. The framework needs features for extracting user-defined datasets, which also consider the properties of the time series. This includes choosing from different sampling and balancing modes and defining arbitrary observation windows for the time series.

R4. *Handling missing data* [13], [14]. Data recorded from real-world systems is often imperfect. A framework should be capable of handling missing entities as well as missing or incomplete time series.

R5. *Performance for big data* [20]. Time series data can be huge. A framework should scale reasonably with increasing amounts of data.

There are also several requirements which we consider out of scope for our framework. In particular, these involve the inclusion of machine learning algorithms or the tailoring of data to specific needs required by certain machine learning algorithms (e.g., normalization or standardization).

IV. Approach

Our main goal is to provide assistance for users who want to apply machine learning (ML) in conjunction with entities, events and time series, most notably for event prediction with supervised techniques. Our framework is specifically designed to aid users in tedious and repetitive tasks that are unavoidable when dealing with such data. This allows to reduce the time needed for manual data processing.

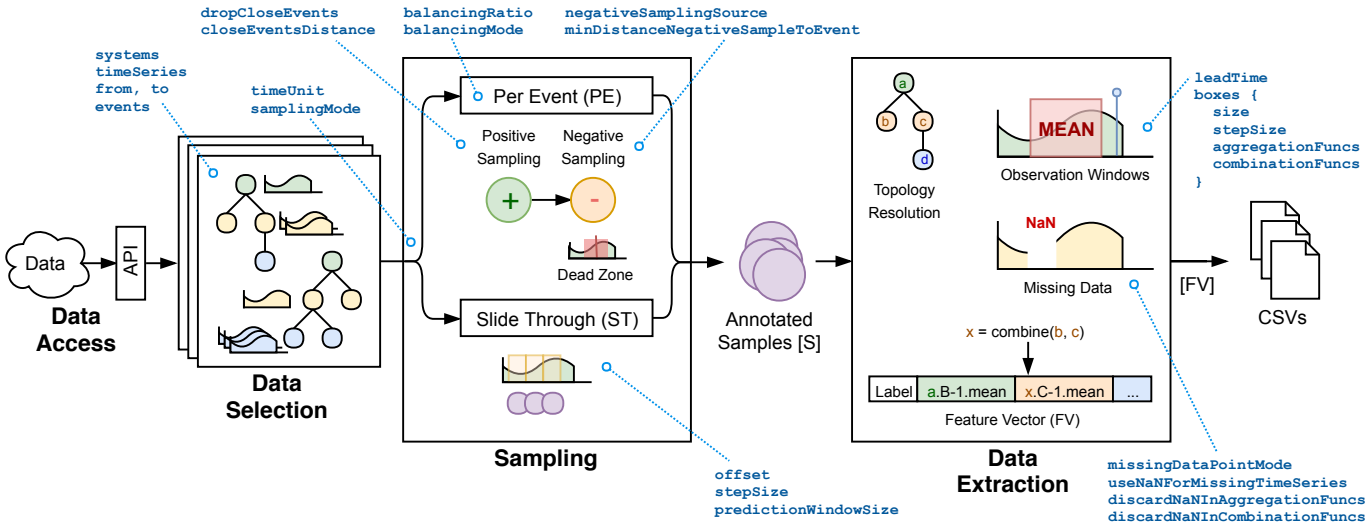

Fig. 2. Overview of our 4-step preprocessing framework. The annotations indicate configuration settings at the different steps.

The framework expects data as described in Section II and creates feature vectors to be directly used by subsequent ML scripts. We accomplish this with the *configuration*-based data preprocessing framework shown in Figure 2. We chose Java as the implementation language for platform independence. A configuration, or *config* for short, specifies all the settings needed to process and transform the raw data into feature vectors stored in comma-separated values (CSV) files. Users only have to provide and specify access to the data sources, create configs, plug those configs into the framework and finally use the generated output files for ML purposes. Configs are the main drivers for our framework, and the users may come from various domains. The YAML [21] format was selected for defining the configuration files due to its readability by humans and language independence.

Our framework can be divided into four main parts: *data access*, *data selection*, *sampling* and *data extraction*. In our pipeline, each of the four steps operates with specified input parameters and yields output that is used in the subsequent step. The last three steps can be customized by various configuration settings. First, the framework needs access to the raw data, i.e., the time series and the topologies of the systems. Once data access is established, the next step is to sample selected data, i.e., users can specify which subsets of the data they want to process (e.g., which systems, which time series, which time frames). Sampling is a highly configurable process ultimately yielding samples for which feature vectors (FV) need to be created. These samples are the input for the final data extraction step. For each of the samples, the framework builds a customizable FV based on the settings, defining how the time series data should be processed. The final output are CSV files containing FVs, which can be readily used for ML.

A. Data Access

The first step enables our framework to access the raw data of the systems, i.e., we create a structured view on the data, which allows us to access all time series, events and topologies. The data access step da can be formalized with the following function definition:

$$da(\text{rawData}) \rightarrow \text{structuredData} \tag{1}$$

Regarding time series data, our framework is not limited to a specific data storage and access technology. It simply provides an interface which must return the required data, specified by the system, the entity, and the time series ID. This way, arbitrary data sources can be connected. The same holds for the event data. The topology of each system, on the other hand, is a YAML file defining the interlinked entities and their lifetimes. The topology file is based on the *main entity type*, which provides the entry point to access all data. For instance, the main entity type in the example of Figure 1 and Table I could be A. The topology file then would contain all links based on this type, e.g., a_1 is connected to b_1 (up to timestamp t_y), b_2, and indirectly to c_1, c_2 and d_1.

B. Data Selection

The first choice users have to make is to specify which data they want to process. As shown in Figure 2, our framework provides various settings for the data selection, which can be controlled via the configs. Users can select the `systems` that should be processed, the `timeSeries` to be used, the `events` for which FVs should be created (e.g., service slowdowns or machine failures), and the time frames for which data should be inspected (`from`, `to`). These settings directly address requirements **R1** and **R3**. The data selection step ds can be formalized with the following function definition, using the output of Equation (1):

$$ds(\text{structuredData}, c_{ds}) \rightarrow \text{structuredData}_{ds} \tag{2}$$

where c_{ds} are the user-defined data selection config settings, and the ds subscript denotes the resulting subset of the data.

C. Sampling

The next step is to sample the selected data. Sampling is the selection of a subset of the original data and is essential in reducing the amount of raw data. Based on the selected data and different config settings, our framework yields a list of annotated samples for which FVs are then created in the data extraction step. Annotated samples contain the timestamp and the main entity (including its associated system) where the sample should be taken. Furthermore, a sample is considered as a positive sample if an event is associated with its timestamp and as a negative sample if no event is associated with its timestamp. The sampling step sp can be formalized with the following function definition, using the output of Equation (2):

$$sp(\text{structuredData}_{ds}, c_{sp}) \rightarrow [S] \tag{3}$$

where $[S]$ is the list of annotated samples and c_{sp} are the user-defined sampling config settings, which are shown in Figure 2 and which we explain in detail below.

Our framework supports two main options controlled via the `samplingMode`: *per event* (PE) and *slide through* (ST). PE means that positive samples are taken based on the events that occurred, which requires further settings for negative samples. ST, on the other hand, means that all samples (positive and negative) are created by sliding through the entire time frame and taking positive and negative samples at regular intervals. For both options, users can define the `timeUnit` for all subsequent time-sensitive settings.

For PE, sampling is split into two steps: positive and negative sampling. Positive samples are created by traversing through all occurred events and taking a sample at the entity at the event timestamp. However, users may choose to activate additional settings that influence the positive samples: `dropCloseEvents` and `closeEventsDistance` can be used to discard events on the same entity that are close to each other. This can be useful to avoid taking a correlated set of positive samples, for example, if a chain of events was triggered. After the positive samples, negative samples must be created. Since there are no events, random timestamps in the available time frame are chosen to take negative samples. There are three different `negativeSamplingSources`, i.e., the entry points (=main entities) from which samples are taken: In case of *non-event entities*, only main entities are used where no event occurred, *event entities* means that only main entities are used where events *did* occur, while *all entities* is a mixture of both. For the latter two, users may activate the additional setting `minDistanceNegativeSampleToEvent` to avoid negative samples that are too close to events (cf. the *Dead Zone* in Figure 2). The number of negative samples is calculated by multiplying the `balancingRatio` with the number of positive samples, either of all systems or of the individual systems (see below). The `balancingMode` defines how these samples are distributed (cf. requirement **R1**). It allows two options: *overall* and *per system*. The former takes the sizes of the systems into account and distributes negative samples accordingly. For instance, if there are two systems with one being ten times the size of the other and we want to create 100 negative samples (resulting from applying the balancing ratio on *all* systems), then this option would create roughly 90 negative samples of the bigger system and 10 negative samples of the smaller system. The *per system* option creates negative samples based on the positive samples of the *current* system.

ST is less complicated because it simply slides through the entire data without the need for balancing. The option `offset` specifies the initial time offset to the data time frame, and samples are taken at each main entity every `stepSize` steps. Users can specify a `predictionWindowSize` that is used to classify a slide-through sample as positive if the timestamp of this sample is close to an event. This is useful as it is often the case that events do not exactly align with the starting point and steps of the ST algorithm, which would result in zero positive samples in the worst case.

D. Data Extraction

In the final data extraction step, our framework creates a list of FVs (stored in CSV files) for each sample from the previous step. The data extraction step de can be formalized with the following function definition, using the output of Equation (3):

$$de([S], c_{de}) \rightarrow [FV] \tag{4}$$

where c_{de} are the user-defined data extraction config settings.

The data extraction step defines how the processed data should appear in the final output. As shown in Figure 2, multiple configuration options are provided. First, users have to specify a `leadTime`, i.e., the distance between the samples and the prediction points (event and non-event timestamps for PE, slide-through timestamps for ST). The lead time can thus be used for specifying how far the predictions should go into the future. Afterwards, users must specify how data is extracted from the time series specified in the data selection step. The topology is automatically resolved to access the correct data (cf. requirement **R2**). Our framework provides observation windows that consist of arbitrary, highly customizable `boxes` that precisely define how data should be extracted for each time series. Each box has four properties:

`size`: the total length of the observation window box.

`stepSize`: the step size in which single data points are visited inside the box.

`aggregationFuncs` (AF): If needed, users can specify functions aggregating the raw data points of the box, i.e., a feature vector sample then contains aggregated values instead of the raw time series data. Common statistical functions, such as the mean/average, standard deviation, median, minimum, maximum, quartiles, skewness or kurtosis, are available and can be arbitrarily combined.

`combinationFuncs` (CF): The same set of statistical functions can also be used for dealing with data from multiple similar entities (cf. requirement **R2**). It might often be the case that users want data from a specific time series for which multiple entities exist in the system. An example is C-1 with entities c_1 and c_2 in Figure 1, where we have two separate

datasets, one for c_1 and one for c_2. However, we cannot simply append them in the feature vector, as feature vectors must be identical in length and not all entities of type B might be linked to exactly two entities of type C. Therefore, we introduce combination functions (CF) that combine multiple datasets. If users selected AFs, they can specify a CF for each such AF. If users selected raw time series data without AFs, a single CF must be specified.

Figure 3 illustrates the data extraction using the example of Figure 1 and Table I. Given an event that occurred at entity a_1, the framework automatically selects all connected entities for which time series have been specified. For b_2 and time series B-1, the user specified two boxes: the first box uses the last three raw values, while the second box uses an aggregation based on the average. These values can directly be forwarded to the feature vector (`FV:B-1`). For c_1, c_2 and time series C-1, the user specified an aggregated box using the average, minimum and maximum of the enclosed raw data. Since there are two similar entities, they have to be combined, in this case using the same functions as in the aggregation step. The result is added to the feature vector (`FV:C-1`), which is assigned the label *Event*, as this sample was taken at an event occurrence.

Finally, there are various settings for dealing with missing data (cf. requirement **R4**). The option `missingDataPointMode` specifies how to resolve gaps in the time series data. Users can either decide to drop the data, or they can replace the missing values with not-a-number values (NaN), the previous value, the nearest available value or an interpolated value. Users can also specify how to handle missing time series via `useNaNForMissingTimeSeriesData`, which either creates NaN placeholders or discards the entire sample. To give users more control over NaN values and how they influence the AFs and CFs, the settings `discardNaNInAggregationFuncs` and `discardNaNInCombinationFuncs` are provided. If enabled, the values in the corresponding settings define a threshold below which NaN values are discarded before applying the AFs and CFs, respectively. This is useful to avoid getting only NaN values as an aggregation or combination result, when there are only few NaN values that could simply

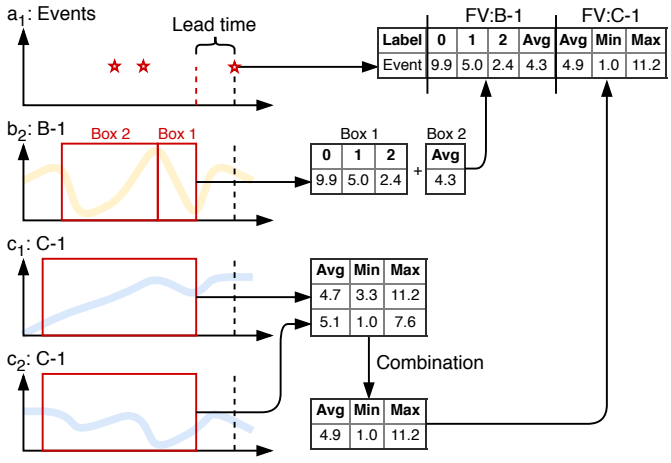

Fig. 3. Example of extracting data into a feature vector.

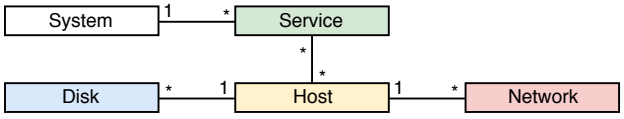

Fig. 4. ERD of a monitored system.

be ignored. If entire entities are missing, possibly because they became inactive (cf. requirement **R2**), the framework automatically either skips the sample altogether or uses NaN values as a replacement.

V. Evaluation

To show the usefulness and the feasibility of our approach (cf. requirement **R5**), we performed an evaluation based on a multi-system monitoring environment provided by an industry partner. Using monitoring time series data, the main task in this real-world scenario is to predict performance slowdown events, which occur at certain entities of a system. The goal is a binary classification, where we want to determine whether data attached to certain points in time indicate an event or not.

In this context, a system is an independent composition of hardware and software components, operated by some service provider, e.g., a web shop running on a certain hardware setup. Figure 4 shows an entity-relationship diagram (ERD) of such a system. Services are abstract entities and represent the business logic that is carried out on one or more hosts, which in turn can host multiple services. The performance slowdown events occur at these services if the average response time of a service exceeds a threshold compared to its baseline. A host is a physical or virtual computing unit to which zero or more disks and networks can be connected. In total, 34 different time series are collected at host, disk and network level, all with a resolution of one minute.

For accessing the monitoring data (cf. Section IV-A), we developed an interface for InfluxDB, a database specifically designed for working with time series data [22]. For creating the topology files, we built a topology model based on the ERD of Figure 4, where we decided to treat services as the main entities, as the events occur at these entities. Given this setup, we had access to 20 days of continuous data of 250 different systems, containing more than 200 000 entities and over seven billion individual time series entries. At the time of the evaluation, we only had limited hardware resources at our disposal. All of the experiments below were conducted on a Dell Precision notebook running on Windows 10 and powered by an i7-7820HQ CPU, 32GB RAM and a 1TB SSD.

A. Case Study

For evaluating the usefulness of our framework, we conducted a case study, in which we took the role of an engineer who was assigned to inspect the accuracy, false positive rate (FPR), precision and recall[1] of a machine learning classifier for event prediction when different datasets are applied. The data were the same as described above, i.e., 20 days of 250

[1]The corresponding formulae and further details can be found at https://en.wikipedia.org/wiki/Confusion_matrix

monitored software systems, where the first 14 days were used for training and the last 6 days were used for testing (test set). The following tasks had to be completed:

T1. Exploring different subsets of the entire data.

T2. Experimenting with various combinations of time series.

T3. Investigating time series with different observation windows.

T4. Experimenting with different sampling approaches.

T5. Using different techniques for handling missing data.

Building a working solution for these tasks from scratch would require significant effort. However, our framework already provides configurable capabilities for experimenting with the data sets. For **T1**, the settings `from`, `to` and `systems` are available to process data subsets based on time frames and the number of systems. For **T2**, `timeSeries` is the main setting to select the raw data sources, and the framework provides several settings for each time series, as required by task **T3**, e.g., `size` and `aggregationFuncs` for the defined observation window `boxes`. There are also plenty of settings for **T4**: PE `samplingMode` with its different options (`balancingMode`, `balancingRatio`, etc.) or ST sampling and its options, e.g., `predictionWindowSize`, `stepSize`, etc. Finally, there are also various settings for **T5**, which comprise how to handle missing values (`missingDataPointMode`) or entire time series (`useNaNForMissingTimeSeriesData`), including options for how to treat missing values in the aggregation and combination functions of the observation window boxes.

To assess the usefulness of the framework and the impact of the parameters on ML results, we created a high number of configs, plugged them into the framework and used the generated output CSV files as input for a machine learning algorithm. Specifically, we selected the default random forest classifier of scikit-learn [23]. We then created about 9 000 configs by building the Cartesian product over different settings, e.g., `size` = {5, 10, 15, 30, 60}, `aggregationFuncs` = {mean, (minimum, maximum)}, etc. to thoroughly investigate the influence of data preprocessing on event prediction performance based on monitoring time series data.

Figure 5 shows the distribution of the four evaluation metrics on the test set, where we trained and tested the random forest classifier with data from about 9 000 different configs. The x-axis represents the value of the corresponding metrics for the configs in percent, while the y-axis represents the number of configs in the corresponding buckets. Evidently, different configs yield drastically different metric values, supporting the claim that appropriate data retrieval is an essential and success-critical step in time-series-based machine learning.

B. Performance Scalability

Since the absolute preprocessing time of our framework highly depends on the chosen data access, we decided to give a relative performance overview. To this end, we selected a single system with 25 services, each of which is linked to one or two hosts, which again are connected to one or two disks and networks. For a fair comparison, the framework processed

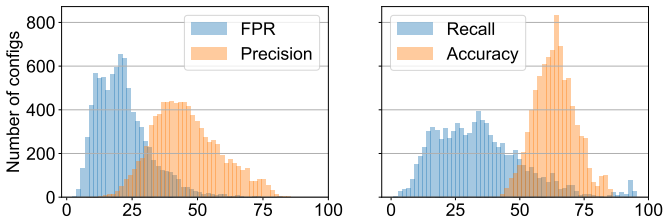

Fig. 5. Metric distribution on the test set of about 9 000 different configs. The x-axis represents the metric values in percent [%].

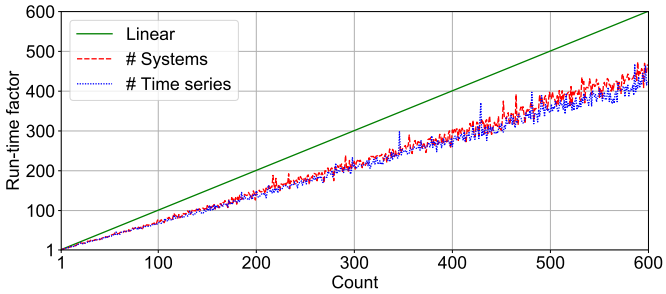

Fig. 6. Relative performance based on normalized run times: number of processed systems (red: *# Systems*) and number of processed time series (blue: *# Time series*) in comparison to a strictly linear normalized run time (green: *Linear*). Strictly linear means that the processing time of n systems would take exactly n-times longer than that of a single system.

the data sequentially. Naturally, a concurrent processing model (e.g., MapReduce [24]) would make sense, as independent systems and time series can be accessed in parallel. We will investigate such an approach in future work.

In the first experiment, we created copies of the selected system to investigate the scalability of our framework with respect to the number of processed systems. We used a simple config with the following main settings: ST sampling with a step size of 30 minutes, and a 60 minute observation window box with a single aggregation function (mean) applied to a single disk time series. The red line in Figure 6 displays the relative performance of up to 600 systems, which shows that our framework scales linearly with the number of systems. With respect to absolute processing time, it took about 560 seconds to process 100 systems on our hardware setup.

Similarly, we conducted another experiment on how well our framework performs when the number of processed time series increases. In this case, we used the same system but created copies of a disk time series to guarantee performance comparability. The config used was the same as above, with the difference that this time we defined multiple observation window boxes – one for each time series. The blue line in Figure 6 displays the relative performance of up to 600 time series, which shows that our framework also scales linearly with the number of time series. With respect to absolute processing time, it took about 520 seconds to process 100 time series on our hardware setup.

In both cases, the run time actually is slightly below strictly linear. This is because some parts of the framework are static, i.e., independent of the number of systems or time series, and secondly, because the Java virtual machine is influenced by factors such as warm-up, optimizations, etc.

VI. Related Work

Data processing to create and select features has been an essential part of machine learning practice for a long time [25]. Especially time series data has caught the interest of many researchers [26], [27], recently also with respect to big data [20], [28]. Efficiently handling such data is of high relevance, as the application of appropriate tools and frameworks can have a great impact in different research areas including failure and reliability prediction [4], finance and economics [29] or environmental challenges [30]. While there are approaches that do not rely on heavy time series preprocessing and feature engineering, such as [31], the majority of the approaches proposed in the literature suggest some form of processing.

A. Time Series Processing

Wilson [19] argues that different forms of data are required for time series mining, i.e., directly working on the raw data is mostly not applicable. He presents three main preprocessing techniques for dimensionality reduction: piecewise approximation, identification of important points and symbolic representation. To show the difference between the classification accuracy using raw time series values compared to feature-based representations, Nanopoulos et al. [10] tested a neural network on synthetically generated time series. For the feature-based representation, they used common statistical functions, such as the mean, standard deviation, skewness and kurtosis, which resulted in higher accuracies. Jansen et al. [8] trained neural networks on multivariate time series data of metal processing machines to predict failures. They included a comprehensive data preprocessing pipeline (cleaning, sampling, grouping, etc.) and tested various configurations (features, history length, network parameters) to find the best solution.

We incorporated many of the above preprocessing steps into our approach, carefully choosing which settings are required and fit best with respect to multivariate time series processing.

B. Frameworks Focusing on Time Series

Many researchers have acknowledged the value of frameworks to help the users in their daily practice. Many of these frameworks have different focus points, such as data preprocessing, time series data handling, visualization or an entire workflow pipeline. Bernard et al. [12] presented an iterative segmentation workflow for multivariate time series with visual analysis so users can see the impact of various processing parameters (data cleaning, sampling, segmentation, etc.) and can then adjust those parameters accordingly. In earlier work [11], Bernard et al. also created a tool for the interactive design and control of a time series preprocessing pipeline. Users can combine multiple preprocessing steps (data cleaning, sampling, normalization, etc.) and can visualize the outcome. TimeCleanser [13] is a visual-analytics-guided framework for cleaning time series data. The authors derived various time-induced data quality problems from related work and incorporated those into their framework. Users can visualize data for inspection and also directly fix data issues such as invalid syntax, missing intervals, or implausible sequences of values. Shi et al. [14] developed a more specific preprocessing framework in the domain of power grid systems. They focused on inserting missing values, data cleaning and data reduction to improve prediction accuracy for datasets with missing data. The authors of [15] introduced an entire data analytics tool pipeline. Their modular and flexible web-interface-based framework supports data storage, preprocessing, analysis and visualization, with a focus on time series data. The only approach also taking system topologies or structures into account is Hora [9], an online failure prediction framework. In addition to time series data of individual components (internal data, e.g., CPU, memory), the authors also considered the architecture of the inspected system to infer where failures would propagate through.

Many of the mentioned approaches can be seen as complementary to ours, for example, visualization to get more insight into the available data beforehand or preprocessing techniques targeting the requirements of particular machine learning algorithms. However, none of the previous work focused on a multivariate time series data preprocessing framework as preparatory work for event prediction, especially utilizing the topology information of the underlying system as well as data from multiple systems.

C. More General Frameworks

Beyond the specifics of time series data, Kirchner et al. [32] proposed a generic framework for data preprocessing with a focus on subsequent clustering, which includes handling missing values, dimensionality reduction, scaling and others. DQF4CT [33] is a conceptual data quality framework for classification tasks which includes a guide for dealing with problems in data quality as well as an ontology that assists in choosing the appropriate data cleaning techniques. YALE [34] is a flexible framework for various data mining tasks with the main goal of fast prototyping. They introduced operators that can be arranged and combined in trees, which are then used to process the data. YALE contains many built-in operators, such as data source operators, preprocessing algorithms (discretization, filtering, normalization, etc.), visualization operators and machine learning operators. Van Merriënboer et al. [35] presented the Fuel framework responsible for data preprocessing (data format standardization, shuffling, resampling, etc.), and the Theano-based Blocks framework (relying on Fuel) for training neural networks. Google Research developed the TensorFlow-based framework TFX [36], which incorporates various steps from a general-purpose machine learning platform. It includes components for data analysis, transformation and validation, training, model evaluation and validation and serving. It provides a multi-purpose framework with unified configuration to avoid project-specific glue code, to reuse components, and to ensure correct data processing.

Similar to the time-series-oriented frameworks, these approaches could be used complementary to our own framework as well, most notably for further preprocessing our CSV output (normalization, filtering, shuffling, etc.) or for plugging this output into a pipeline, e.g., based on TFX.

VII. Conclusion

In this paper, we presented an approach for providing preprocessing assistance for engineers, who apply machine learning on event and time series data. We identified multiple requirements for a framework that has to handle events and multivariate time series from multiple systems, where each such system is represented by interconnected entities, which may also evolve over time. Based on these requirements, we developed a novel, offline preprocessing framework that can be customized with configuration files to extract and preprocess the data as the users desire. The output of our framework are CSV files, which can directly be used for further machine learning purposes. Our framework is especially helpful in supporting an iterative workflow, as changes to the preprocessing steps can be easily made via the configs.

We evaluated our framework using real-world data from a multi-system monitoring environment. First, we conducted a case study to demonstrate the usefulness of our framework for exploring various data configurations and the influence of different data settings on the accuracy of the trained machine learning models. Second, we investigated the performance and scalability based on two major data contributors: the number of systems and the number of available time series. Both experiments revealed a linear scaling of our framework.

Despite the successful evaluation, there is much room for improving our prototype framework. For example, parallel processing based on MapReduce [24] could prove valuable. Moreover, we plan to apply our framework to data from other domains as well. Future work also includes creating the necessary extensions for real-time (online) prediction tasks, where data is continuously streamed into the framework.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and our industry partner is gratefully acknowledged. We would also like to thank Markus Schedl for his advice and fruitful discussions.

References

[1] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised leaning," *International Journal of Computer Science*, vol. 1, no. 2, pp. 111–117, 2006.

[2] S. S. Skiena, *The Data Science Design Manual*. Springer, 2017.

[3] F. Cady, *The Data Science Handbook*. Hoboken, NJ, USA: John Wiley & Sons, Inc., Feb. 2017.

[4] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, no. 3, pp. 10:1–10:42, 2010.

[5] J. Borges *et al.*, "Predicting target events in industrial domains," in *Machine Learning and Data Mining in Pattern Recognition*. Cham, Switzerland: Springer International Publishing, 2017, pp. 17–31.

[6] B. Sharma *et al.*, "CloudPD: Problem determination and diagnosis in shared dynamic clouds," in *Proceedings of the 43rd International Conf. on Dependable Systems and Networks*, June 2013, pp. 1–12.

[7] L. Yu *et al.*, "Practical online failure prediction for blue gene/p: Period-based vs event-driven," in *Proceedings of the 41st Int'l. Conf. on Dependable Systems and Networks Workshops*, June 2011, pp. 259–264.

[8] F. Jansen *et al.*, "Predicting machine failures from industrial time series data," in *Proceedings of the 5th International Conference on Control, Decision and Information Technologies*, April 2018, pp. 1091–1096.

[9] T. Pitakrat *et al.*, "Hora: Architecture-aware online failure prediction," *Journal of Systems and Software*, vol. 137, pp. 669–685, 2018.

[10] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Feature-based classification of time-series data," *International Journal of Computer Research*, vol. 10, no. 3, pp. 49–61, 2001.

[11] J. Bernard *et al.*, "Visual-interactive preprocessing of time series data," in *Proceedings of the International Conf. on Interactive Visual Analysis of Data*. Linköping University Electronic Press, 2012, pp. 39–48.

[12] ——, "Combining the automated segmentation and visual analysis of multivariate time series," in *Proceedings of the EuroVis Workshop on Visual Analytics (EuroVA)*, 2018, pp. 49–53.

[13] T. Gschwandtner *et al.*, "TimeCleanser: A visual analytics approach for data cleansing of time-oriented data," in *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*. New York, NY, USA: ACM, 2014, pp. 18:1–18:8.

[14] W. Shi *et al.*, "An integrated data preprocessing framework based on apache spark for fault diagnosis of power grid equipment," *Journal of Signal Processing Systems*, vol. 86, no. 2, pp. 221–236, Mar 2017.

[15] J. Frenzel *et al.*, "A generalized service infrastructure for data analytics," in *Proceedings of the 4th International Conference on Big Data Computing Service and Applications*, March 2018, pp. 25–32.

[16] T. Zimmermann *et al.*, "Cross-project defect prediction: A large scale experiment on data vs. domain vs. process," in *Proc. of the 7th Joint Meeting of the European Software Engineering Conference and the Symposium on The Foundations of Software Eng.*, 2009, pp. 91–100.

[17] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–40, 2017.

[18] A. Schörgenhumer *et al.*, "Using crash frequency analysis to identify error-prone software technologies in multi-system monitoring," in *Proceedings of the 18th International Conference on Software Quality, Reliability and Security (QRS)*, July 2018, pp. 183–190.

[19] S. J. Wilson, "Data representation for time series data mining: time domain approaches," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 9, no. 1, pp. 1–6, 2017.

[20] S. García *et al.*, "Big data preprocessing: methods and prospects," *Big Data Analytics*, vol. 1, no. 1, pp. 9:1–9:22, Nov 2016.

[21] O. Ben-Kiki, C. Evans, and B. Ingerson, *YAML Ain't Markup Language*, 2009. [Online]. Available: http://yaml.org/spec/1.2/spec.pdf

[22] InfluxDB, "InfluxDB 1.6 documentation," 2018. [Online]. Available: https://docs.influxdata.com/influxdb/v1.6/

[23] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[24] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[25] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1, pp. 245–271, 1997.

[26] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, pp. 11–24, 2014.

[27] T. chung Fu, "A review on time series data mining," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[28] Y. Xu *et al.*, "Industrial big data for fault diagnosis: Taxonomy, review, and applications," *IEEE Access*, vol. 5, pp. 17 368–17 380, 2017.

[29] R. C. Cavalcante *et al.*, "Computational intelligence and financial markets: A survey and future directions," *Expert Systems with Applications*, vol. 55, pp. 194–211, 2016.

[30] C. Damle and A. Yalcin, "Flood prediction using time series data mining," *Journal of Hydrology*, vol. 333, no. 2, pp. 305–316, 2007.

[31] F. Karim *et al.*, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.

[32] K. Kirchner, J. Zec, and B. Delibašić, "Facilitating data preprocessing by a generic framework: a proposal for clustering," *Artificial Intelligence Review*, vol. 45, no. 3, pp. 271–297, Mar 2016.

[33] D. C. Corrales, A. Ledezma, and J. C. Corrales, "From theory to practice: A data quality framework for classification tasks," *Symmetry*, vol. 10, no. 7, pp. 248:1–248:29, 2018.

[34] I. Mierswa *et al.*, "YALE: Rapid prototyping for complex data mining tasks," in *Proceedings of the 12th Int'l. Conf. on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2006, pp. 935–940.

[35] B. Van Merriënboer *et al.*, "Blocks and Fuel: Frameworks for deep learning," *arXiv preprint arXiv:1506.00619*, 2015.

[36] D. Baylor *et al.*, "TFX: A TensorFlow-based production-scale machine learning platform," in *Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1387–1395.