A Method for the Joint Analysis of Numerical and Textual IT-System Data to Predict Critical System States

Patrick Kubiak[1](✉), Stefan Rass[2], Martin Pinzger[2], and Stephan Schneider[3]

[1] Volkswagen Financial Services AG, Brunswick, Germany
patrick.kubiak@vwfs.com
[2] Alpen-Adria-University, Klagenfurt, Austria
{stefan.rass,martin.pinzger}@aau.at
[3] University of Applied Sciences Kiel, Kiel, Germany
stephan.schneider@fh-kiel.de

Abstract. We present a method for the joint analysis of textual and numerical IT-system data usable to predict possibly critical system states. Towards a comparative discussion culminating in a justified model and method choice, we apply logistic regression, random forest and neural networks to the prediction of critical system states. Our models consume a set of different monitoring performance metrics and log file events. To ease the analysis of IT-systems, our models judge the future system state using one binary outcome variable for the system state's criticality as "alarm" or "no alarm". Moreover, we use feature importance measures to give IT-operators guidance on which system parameters, i.e., features, to consider primarily when responding to an alarm. We evaluate our models using different configurations, including (among others) the demanded lead time window for incident response, and a set of common performance measures. This paper is an extension to previous work that adds details on how to jointly process textual and numerical data.

Keywords: Machine learning · IT-operations · AIOps

1 Introduction

One of the major challenges for IT-operations departments is to manage a complex and heterogeneous IT-infrastructure landscape. This situation results in a heterogeneous toolbox of monitoring systems as well as a large number of different IT-system parameters that IT-operators have to monitor. Commonly, each monitoring system has its own set of rules to notify IT-operators in any case a system state turns from regular operation into a critical state. In some instances, such rules come as isolated thresholds for each IT-system parameter of interest, whose excess or undercut generates notifications, i.e., alarms. Furthermore, IT-systems store necessary information to judge the system state in different data sources having a non-compatible kind of format, i.e., numerical monitoring data and textual log file data. In this paper, we present a novel method to combine

M. van Sinderen et al. (Eds.): ICSOFT 2020, CCIS 1447, pp. 242–261, 2021.
https://doi.org/10.1007/978-3-030-83007-6_12

data from these two major data sources of IT-operations departments usable for *machine learning* (ML) models. We aim to use such a combined data set to i) judge the system state using one binary outcome instead of a set of different isolated alarms; ii) predict incoming critical system states using an experimental setup for data acquisition and iii) analyze the influence of monitoring metrics and log file events on the system state using a single ML model. We give a detailed procedure to transform textual log file data into a suitable format to be usable with numeric monitoring data for a joint analysis and predictive inference. Furthermore, we apply a set of classification methods, i.e., *logistic regression* (LogReg), *random forest* (RF) and *neural networks* (NN), on this data set to achieve i) and ii), and for a comparative study of the three methods. We evaluate the prediction quality of our models using a set of common diagnostic metrics. For iii), we use RF importance measures, i.e., *mean decrease in accuracy* (MDA) and *mean decrease in Gini* (MDG), to explain the reasons why each model did or did not raise an alarm. Hence, we use feature importance measures as a tool to analyze the influence of each monitoring metric and log file event as triggers for (critical) changes of the system state.

This paper will answer following research questions (RQ):

1. **RQ1:** How can we join numeric and textual IT-system data to be usable in a single ML model?
2. **RQ2:** How accurate are different models to predict the system state in the form of a binary classification problem?
3. **RQ3:** Which features do have the biggest influence on the system state and can be considered as most promising triggers, i.e., root causes, for changes?

We start with a related work section to demarcate this paper from previous work. In Sect. 3, we present a short description of our experimental setup that is similar to the configuration of our previous paper [1]. The focus of this paper is a detailed description of the data preparation process, which was necessary to "unify" data of heterogeneous types (textual and numeric) and out of multiple sources, as presented in Sect. 4. We extended our experiments using other ML methods as presented in Sect. 5. Furthermore, we used a diverse set of evaluation metrics and changed the approach to evaluate the relevance of features on the binary outcome that classifies the overall system state as normal or critical. Previous work [1] suggested a relatively simple importance measure by taking a relative count of how often a feature appeared as significant in a model (a decision tree), relative to all cases (technically, a conditional probability for the feature to appear or not appear in the prediction model). This work adopts the more popular concept of importance measures of the relative count of appearance (conditional probability of feature significance) is agnostic of the level of appearance in the tree. This extended work allows us a more detailed evaluation of our models and more informative results as presented in Sect. 6. We close our work with new findings related to threats of validity in Sect. 7 and a conclusion in Sect. 8.

2 Related Work

The IT-operations domain is a predestined field for ML researchers since it generates a huge amount of data, which often exceeds human analysis capabilities. Most ML models in the literature focus either on numerical or textual data and leave possible advantages of a joint analysis unexplored. However, the literature offers a vast lot different papers related to this domain, which enable practitioners to maximize the availability of IT-systems due to automation, issue prevention, easier problem determination and faster troubleshooting [2]. We can achieve such advancements using processual recommendations [3, 4], e.g., ITIL, or ML methods that accelerate the understanding of data and improve its potential as valuable resource for organizations. We consider *symptoms monitoring* and *detected error reporting* as main monitoring mechanisms for our work. Symptoms refer to as side-effects in case of abnormal behavior of IT-systems while errors occur when things go wrong and the system state differs to the expected system state [5]. Errors are undetected until monitoring systems or system users observe any differences in the system state. Symptom monitoring is the standard mechanism to permanently check if any threshold violations of runtime metrics, e.g., CPU utilization, occurred [5]. Detected errors are typically protocolled in log files that are an extensive collection of all system events. On monitoring data, different researchers applied, among others, regression methods [6, 7] or classification methods [8–10]. Often, we prefer to transform log file data into sequences of log file events beforehand. Thus, corresponding ML methods consume event-driven input data and allow us to apply event pattern mining [11–14] or event summarization methods [15, 16]. Among others, one approach discovers whether there is correlation between the system load intensity, recorded by monitoring agents, and the occurrence of computational intensive log file events [17]. Nevertheless, the analysis of both data sources in a complementary manner has not be exhaustively explored yet. We recommend two surveys [5, 18] to interested readers to get a more comprehensive selection of available ML methods in the IT-operations domain. Furthermore, we proposed a method selection guide for practitioners in dependency of the type of data and the application in an earlier work [2].

3 Experimental Setup for Data Acquisition

For data acquisition, we used an experimental concept for an automated load and performance test scenario to simulate real-life user interactivities on a small-scale digital twin of a real-life IT-system environment [1]. This IT-system resembles a productive system without being one and allows us to fully control and manipulate the systems behavior as requested. Since the IT-system is a training environment and mainly used for occasional (and non-periodically happening) user trainings, there was no continuous system load on it. Therefore, we developed test scripts to emulate regular system load, such as client transactions sent to the system, and load peaks. Here, we used *VuGen* (link) and scheduled the test scripts using *LoadRunner Enterprise* (link), which both are software products of *Micro Focus*. This step was necessary to generate the required lot of anomalies in short time, which we would otherwise have to collect over long periods (possibly months to years) on a productive system. The major advantage of this setup is to produce any sort of behavior to generate data for model training that satisfies our requirements best, e.g., a balanced data set. In particular, such an experimental setup allowed us to trigger rare events and anomalies of diverse kinds to the amount and extent required [1]. Thus, we consider artificially generated system load intensity based on real transactions of an industrial IT-system to obtain the data to evaluate our method and models.

3.1 Application Architecture and Implementation

The IT-system of our choice is a *Java* web application. The *contract management system* (CMS) is an on premise cloud application hosted at our data center running on an *OpenStack* environment. Figure 1 presents the architecture of the CMS.

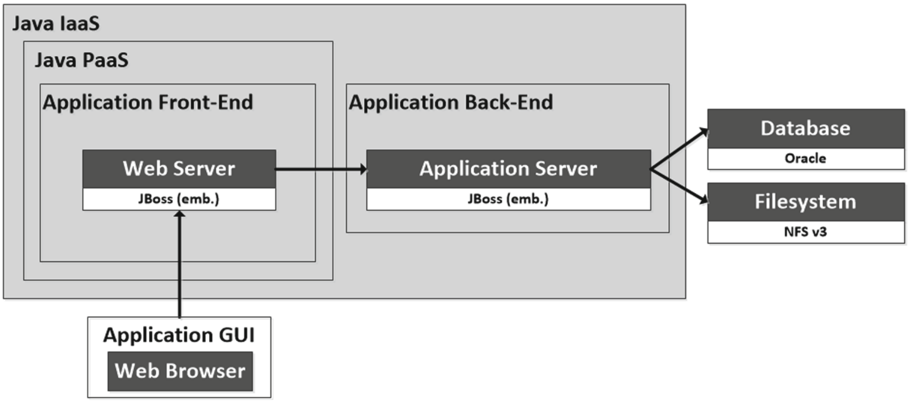

Fig. 1. Architecture of the CMS [1]

We use a *platform as a service* (PaaS) as frontend component of the CMS and an *infrastructure as a* service (IaaS) as backend component of the CMS. Both components run on *Linux RHEL 7.x* operating system but use different application runtime frameworks. The PaaS uses *WildFly* (link) while the IaaS uses *JBoss EAP* (link). However, both components generate own log file data with a varying structure. For the collection of the monitoring metrics, we used *DX Application Performance Management* (link) on both components. Table 1 presents the sizing of the PaaS and IaaS components.

Table 1. Sizing of the PaaS and IaaS

	PaaS	IaaS
CPU	4 x Intel Xeon CPU E5-2680 v4 @ 2.40 GHz	8 x Intel Xeon CPU E5-2680 v4 @ 2.40 GHz
Memory	8 GB	8 GB
Disk space	4 GB	20 GB

Section 3.3 presents a description of obtained log file messages and monitoring metrics, which then refer to as features of our predictive models.

3.2 Load and Performance Test Design

To generate necessary system load on our testbed, we triggered a varying number of *virtual users* (vUsers) on the system that act in the same way as human system users do. Hence, the vUsers call a set of system transactions, e.g., search for existing contracts in the database or create new contracts, and the CMS does not recognize any deviation to human users. The only difference depends on the scripted induction of the system load since the vUsers follow a predefined schedule and call system transactions without any breaks what human users naturally do. However, we designed a concept for a 10-day long experiment for data acquisition and let a varying number of simultaneous working vUsers be the trigger for the system load intensity. From data quality perspective, we aim to evaluate the suitability of our models with data whose underlying generative processes are entirely known to us. Thus, patterns can be explained and "noise" under normal conditions is distinguishable from load-induced anomalies. We produced system load for 8 h on each test day. To avoid patterns, we scheduled stepwise load peaks with a varying intensity to the CMS, which refer to as anomalies that we aim to predict. After each load peak, the system returned to a similar baseline that refers to as normal system state. Figure 2 presents an example of the induced system load of one test day.

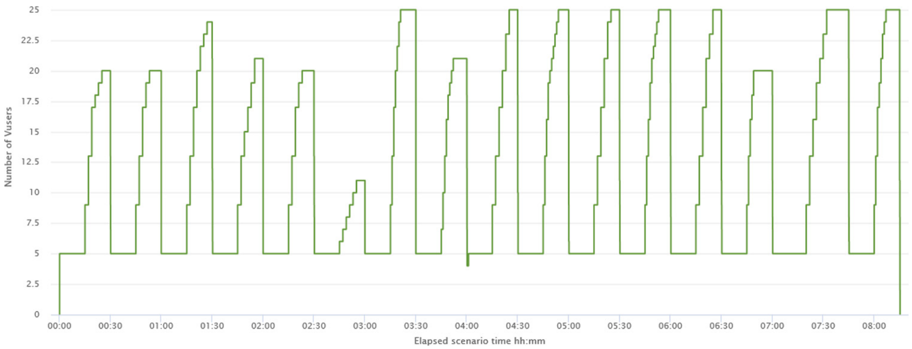

Fig. 2. System load of one test day

We triggered the changes of the system state in a 15-min interval to guarantee a balanced data set containing as much records for each system state as possible. We recognized a significantly negative influence on the accuracy of our models resulting from too imbalanced data in an earlier experiment. We used a rule-based approach for the data labeling related to the number of vUsers working on the system as presented in Sect. 4.4. Due to internal regulations of the enterprise, we had to limit our experimental setup to a maximum load intensity generated by ≤25 vUsers. This is one threat to validity and further discussed in Sect. 7.

3.3 Description of Monitoring Metrics and Log File Messages

The monitoring agents collected a set of groups of performance metrics, which consist of at least one but mostly of more metrics. For example, the CPU is a single measure while the agents collect measures for response times of over 50 different *JavaBeans*. Table 2 presents an overview of the collected groups of monitoring metrics.

Table 2. Collected groups of monitoring metrics [1]

Group of monitoring metrics	Description
Average response time (AR)	The average response time in ms of a JavaBean from the method call to the response
Memory pools (MP)	The dedicated part of the heap memory in bytes, which allocates memory for all instances and arrays at runtime
Concurrent invocations (CI)	The number of simultaneous calls of a JavaBean
CPU	The CPU utilization in %
% time spent in garbage collection (GC)	The percentage time within an interval, in which obsolete in-memory code is removed
Sockets (SO)	The number of available communication end points of the IT-system

From performance perspective, we can assume a critical system state if the IT-system meets at least one of the following conditions:

1. The values of the AR group of metrics increase significantly
2. The values of the AR and CI group of metrics increase at the same time
3. The value of the GC metric exceeds ≈25%
4. The values of the SO group of metrics range in the area of 0 over a longer period

For confidentiality reasons, we are unable to provide an overview of the exact log file messages and their meaning since inferences to the CMS are prohibited. Thus, Table 3 only presents an abstract overview of the log file messages grouped by their semantic meaning.

The total amount of different log file messages is 42 and all of them are error messages. We assume that errors are the most promising indicators to observe misbehavior on log file level. Therefore, we exclusively filtered out errors from the raw log file data and used them in our predictive models as described in Sect. 4.1.

4 Data Preparation

The core of our method is to unify textual and numerical IT-system data in one predictive model. In the following section, we describe necessary steps to transform the textual data in a numerical form joinable with the monitoring data. Afterwards, we applied a set of standard practices to analyze corresponding features in case of their predictive power and reduced the model complexity by removing features that did not satisfy the requirement of increasing the predictive ability.

Table 3. Overview of the log file messages

Description	Number of different Messages in the harvested logs	Component	
		PaaS	IaaS
Session data expired	12	X	X
Exception handling	8	X	X
Remote procedure call failed	5	X	X
Session timeout	2	X	X
Loading of language ID failed	1	X	
Generation of a new contract failed	1	X	
Database connection failed	1	X	
Error for some input string	1	X	
Top level exception	1	X	
Unexpected value	1		X
Some internal error	1		X
Failed to call a JavaBean	1		X
Some connection error	4		X
Invalid search request	1		X
Database error	1		X
Some missing parameter value	1		X

4.1 Extracting Error Event Sequences from Log Files

It is a common practice to transform raw log file data into event sequences since the analysis of structured log file events is much easier than exploring log file messages in the overall textual corpus, e.g., using appropriate forms of visualization. Furthermore, such log file event sequences allow us to apply different ML methods as described in Sect. 2. However, we focus on a timestamp-based structure of log file events in the form of an *event-occurrence-matrix* (EOM) as prerequisite for our method. Commonly, we can apply three different methods to obtain such a target structure from raw textual log file data: log parsers, classification or clustering methods. As described in Sect. 4.2, we applied clustering algorithms for this task since such methods identify clusters, i.e., log file events, in the data autonomously at the costs of accuracy (in comparison to log parser and classification methods). The major advantage is that clustering methods in this case offer a high degree of flexibility and may require less parameter tuning. We do not necessarily need deep knowledge about the log file structure, which is a requirement to develop specific (and possibly rigid) extractors, i.e., log parsers. Moreover, we avoid preparing any labeled training data as it is necessary to apply classification methods. However, log files contain a lot of information, which are not directly related to misbehavior and we want to train our models exclusively on conditions that lead to potential issues. Therefore, we filtered out error messages beforehand using *regular expressions*. We extracted 786,522 error messages out of approximately 95 million log file messages in total. Afterwards, we applied a set of *information retrieval* techniques to shrink the remaining error messages to text parts, which are necessary to generate the EOM. For example, we removed record specific data, e.g., unique identifiers (IDs), which may confuse clustering algorithms in the way that they may group error messages with same semantics into different clusters.

4.2 Numerical Representation and Clustering of Log File Messages

After we cleaned the texts in the log file data to the minimum extent required, we aim to transform the structure in the form of a timestamp-based EOM. A *document-term-matrix* (DTM) allows to convert text into numbers by counting the number of times each word, i.e., term, appears in the given document corpus. A DTM is a matrix, in which the element in row i and column j gives the number of appearance of term j (associated to the column) in the document i (associated to the row) and allows us to apply clustering methods on this – hereby numerical – representation of text [19]. For the clustering task, we applied the *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) algorithm [20]. In comparison to other clustering algorithms, e.g., *K-means*, the major advantage of DBSCAN is that it works without an a priori guess for the number of clusters. Given that we may hardly expect how many and what different kinds of messages will be in the logs to come, this is a crucial advantage. For the data in this work, we let a CMS generate logs for a period of 10 days. We explicitly refrain from manually digging into perhaps millions of rows in the raw log file to point out the applicability of the method with only least or no domain knowledge. In case that domain knowledge is available, refined results may be obtainable upon replacing DBSCAN by a more "pre-informed" clustering at this step. Nevertheless, in absence of specific domain knowledge, DBSCAN is a simple method to apply [1]. For our experiments, we took DBSCAN configured with $minPts = 4$ and $\varepsilon = 0.4$ after testing different configurations without considerable differences for the result. ε refers to as the radius and *minPts* refers to as the minimum number of points falling into the proximity of the cluster-center to reasonably call such an accumulation of points a "cluster". We obtained 28 clusters and 6 noise points for the PaaS logs and 14 clusters and 4 noise points for the IaaS logs. Unlike K-means or hierarchical clustering algorithms, DBSCAN does not force all observations into clusters. Hence, it has the ability to remove noise into separate noise clusters and prevents distorted clusters [21]. Now, we combine obtained cluster and timestamp information to generate the EOM having the structure of Table 4.

Each cluster then directly defines another indicator, i.e., feature x_i in the models; at each time t. Then, we can assign the error messages at this time to some cluster, corresponding to setting the variable $x_i = 1$ if error messages related to this cluster were found, i.e., they occurred, in the log, or $x_i = 0$ otherwise. Finally, the set of 0-1-valued variables $x_1, x_2, \ldots, x_{28}$ for the 28 clusters in the PaaS logs and $x_{29}, x_{30}, \ldots, x_{42}$ for the 14 clusters in the IaaS logs are the first part of the data set. Afterwards, we joined the EOM with the numeric monitoring data using the timestamp as primary key and obtained the final data set with 196 features.

Table 4. Event-occurrence-matrix

Timestamp	PaaS Event 1	…	PaaS Event 28	IaaS Event 1	…	IaaS Event 14
xx:xx:xx	1	…	0	0	…	1
xx:xx:xx	0	…	1	1	…	1

4.3 Feature Analysis and Dimension Reduction

A data set with 196 features is not easy to handle and may unnecessarily increase the complexity of our models. Therefore, we applied a set of standard practices to exclude features that do not improve the predictive power of the models. Removing such features results in a smaller data set that can significantly improve the efficiency and results of ML models [22]. Without any effort, we can directly remove 60 features. This features were constants having zero values due to the fact that the export of the monitoring data consists of each a column for all available JavaBeans regardless the CMS called corresponding JavaBeans within the 10 days or not. In other words, there were no measures for 60 monitoring metrics. To further decrease the dimensionality of the data set, we applied following statistical practices:

1. Remove features with low variance since they often suffer of little predictive information and have no positive influence to the skill of ML models [23]
2. Remove highly correlated features using a threshold of ≥ 0.9 [24] to judge the correlation as very strong
3. Remove outliers using the interquartile method [25]
4. Remove missing values using listwise deletion [26]

Step 2) may be optional in dependency of the applied ML methods. Strong correlation among features leads to the phenomenon of *multi-collinearity*. This can be troubling for some ML methods [23] and is a key problem for binary logistic regression [27], which is one of our choices for the prediction task. Finally, these steps reduced the number of features to 47 out of initial 196 features.

4.4 Data Labeling

For the labeling, we chose a deterministic approach based on the number of vUsers working simultaneously on the CMS since we assume the number of vUsers on the system as main trigger for the system load intensity. We added a column called "Alarm" that refers to as label of each record where $Alarm = 1$ denotes a critical system state and $Alarm = 0$ denotes a normal system state. We defined the following rules to label the data [1]:

1. Normal system state:≥ 5 and ≤ 17 vUsers working on the CMS
2. Critical system state:≥ 18 and ≤ 25 vUsers working on the CMS

Remark: it should be kept in mind that a deterministic labeling will generally produce data on which regression models may fail due to *separation.* Hence, a manual labeling or at least a manual check of a sample of the machine-generated labels is advisable, if regression models are to be evaluated. On the contrary, a divergence problem when fitting a regression model can in turn be an indicator of determinism and a pointer towards a trial with a deterministic (e.g., decision tree) model.

Internal regulations of the enterprise limited our experiments. We assume that a maximum of 25 vUser never really exhausted available resources of the CMS, i.e., the system was never overloaded or reached a serious critical system state. However, we deemed this experimental setup as suitable to get a first feeling in case of evaluating our method. Nevertheless, a critical discussion related to the limitations is part of Sect. 7 to ensure transparency to the readers.

5 Modeling and Evaluation

For the classification task, we applied three different ML methods: LogReg, RF and NN and evaluated their suitability using 16 different configuration cases and a set of common performance measures as described in the following sections. Our general prediction scheme is illustrated in Fig. 3. It visualizes the basic idea of our novel method for a joint analysis of time series data collected by monitoring agents and discrete event data.

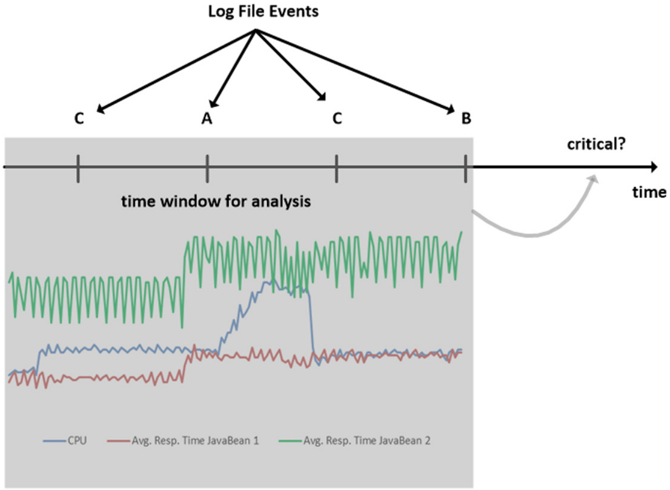

Fig. 3. General prediction scheme [1]

5.1 Choice of Classification Methods

Our initial choice for the classification task is LogReg since it is the de facto standard method for binary classification problems [28]. Moreover, it tells us - during the fitting

– if the dependent variable, i.e., alarm, has a deterministic dependence in question [1]. That is, either:

1. There is a stochastic element governing whether or not an alarm is raised, then the logistic model is a reasonable choice and the fitting of coefficients will converge
2. There is a deterministic process behind the alerts to occur, in which case the model fitting (a maximum likelihood optimization algorithm) will fail to converge, which is then the information that the logistic regression model is not a good choice.

In the case of 2), the LogReg tells us that we should rather fit a more "deterministic" model. In our previous work [1], we applied decision trees as our second choice since the LogReg in some cases failed to converge. We decided to rethink this decision since tree based models suffer of high variance and obtained results may be quite different in dependency of the randomly sampled data [29] in our evaluation design. Thus, we decided to use RF as alternative method, which reduces the variance of bagging as well as reduces the correlation between trees without increasing the variance too much [30]. Moreover, RF offers out-of-the-box analysis of the feature importance using the MDA and MDG measures, which enables us to identify features that are most likely relevant to judge the system state of the CMS. Additionally, we applied NN to complete the selection of candidates for the classification task since they are known to be powerful prediction methods and extend the evaluation of this work.

5.2 Configuration Cases

We want our models not only to classify the system state at current time t, we aim to predict incoming critical system states with a lead time window of t_{+1}, t_{+5}, t_{+10} and t_{+15} using historic records at t_{-1}, t_{-5}, t_{-10} and t_{-15}. Thus, we construct a set of 16 different cases, i.e., training data sets, in the following way [1]:

We denote the lead time window as Δt and the historic observations over a fixed time window as Δh. Now, we proceed as follows: At time t, collect all records within period $H = [t - \Delta h, t]$ and concatenate these records into a larger training data set that contains all data within this time window. In this way, we obtain a data set, in which each x_i occurs with multiple copies in the record. For example, if there fall three records into the past history, each carrying the features $x_1, \ldots, x_k$, we obtain a record with a feature set of $x_1^{(0)}, \ldots, x_k^{(0)}, x_1^{(1)}, \ldots, x_k^{(1)}$ and $x_1^{(2)}, \ldots, x_k^{(2)}$ where $x_i^{(j)}$ refers to as the i th feature at j timestamps prior to t. Hence, with the feature set constructed as above, $Alarm = 1$ if and only if there was an alarm in the records between t and $t + \Delta t$. Thus, we set $Alarm = 1$ if there was an alarm falling into $[t, t + \Delta t]$ and we instantiate the current record with historic observations collected from all records falling into the period $[t - \Delta h, t]$. Otherwise, we set $Alarm = 0$ since there has be no race condition occurred after t within Δt, which we aim to predict on the current system state and history. We consider following configurations for the evaluation of our models as presented in Table 5.

Each configuration case differs in its setting in case of Δt and Δh, which we both measure in minutes. We aim to identify whether a set of varying Δt and Δh influences the results in case of the prediction accuracy or the importance of the features and their

Table 5. Configuration cases [1]

Lead time (Δt)	Number of historic observations (Δh)			
	1	5	10	15
1	C1	C5	C9	C13
5	C2	C6	C10	C14
10	C3	C7	C11	C15
15	C4	C8	C12	C16

multiple copies to possibly identify somewhat like a prediction limit if the accuracy significantly decreases. Naturally, we would expect a larger retrospective window to increase the prediction accuracy, and likewise, the accuracy would be expected to deteriorate, the larger the forecasting window is made (i.e., the farer we attempt to look into the future). The second expectation turns out to be not the case.

5.3 Evaluation Design

The settings of the configuration cases allow us to test our models in case of a varying prediction horizon and history to identify whether there is an impact on the model results or not. Each of the cases (C1-16) represents a new data set, which we considered independently for training and test of our models. Thus, we fitted at least 16 models for each of the three classification methods. Furthermore, we fitted each model using a loop with 100 runs where we generate randomly sampled data for training and test in each of the runs. Unfortunately, we are unable to provide results for the NN for 100 runs but still present that their prediction quality seems to be similar to LogReg and RF after training and applying them once on all configuration cases. The reason is related to high computational time for evaluation of each case, repeating the evaluation 100 times for every method and running this experiment was considered as impractical. We evaluate the performance of our models using a set of common performance metrics, which we determine for every single run. These metrics are: *accuracy*, *precision*, *recall*, *F1-Score* and the *Matthews correlation coefficient* (MCC). Here, we follow a standard practice to evaluate our models on a broad range of performance metrics for a fair and honest evaluation. This practice is preferred over using a single metric that is being optimized (Zhang/Zhou 2014) as we only focused on the accuracy measure in our prior work. For all measures, we can consider the model quality as higher if the measures are higher with a maximum value of 1, which refers to as perfect classification. We remark that in the IT-operations domain we should give more attention to the recall than to the precision measure. This is because a miss, i.e., false negative, of predictive models in this area may cause expensive (tangible or intangible) damage, i.e., a service break, for organizations. Recall penalizes misses with high costs and is more reliable in this case. However, in each run, we additionally calculate the MDA and MDG measures to evaluate the importance of the features for the fitted RF. For evaluation of experimental studies, the popularity of both measures increases since both confirmed practical utility

[31] although there is a lack of clearance regarding their inner workings [31, 32]. We aim to use this information to identify the most promising indicators for the CMS turning into a critical system state. Moreover, we give practitioners, i.e., IT-operators, guidance at hand on which system parameters to focus on primarily if our models raise alarms to answer the "why" the system is turning into a critical state. For IT-operators, this is invaluable and can significantly ease the determination of root causes.

6 Results

Let us now present our results in case of the predictive quality along the set of performance measures and then present the analysis of the feature importance, which we exclusively obtained for the RF.

6.1 Performance Metrics

We start our evaluation with the results of the single run of the NN for all configuration cases as presented in Table 6.

Table 6. Results of the NN for a single run

Case	Accuracy	Precision	Recall	F1-score	MCC
1	0.97	0.96	0.98	0.97	0.94
2	0.93	0.93	0.93	0.93	0.86
3	0.94	0.88	0.95	0.92	0.87
4	0.95	0.94	0.90	0.92	0.88
5	0.99	1.00	0.99	0.99	0.98
6	0.99	0.99	0.99	0.99	0.97
7	0.99	0.99	0.98	0.98	0.97
8	0.98	0.96	0.98	0.97	0.96
9	1.00	1.00	1.00	1.00	1.00
10	0.97	0.98	0.97	0.96	0.93
11	0.97	0.96	0.95	0.95	0.93
12	0.99	0.98	0.98	0.98	0.97
13	0.99	0.96	1.00	0.98	0.96
14	0.98	0.97	0.98	0.97	0.95
15	0.98	0.97	0.98	0.98	0.95
16	0.99	0.99	0.98	0.98	0.97

For the single run, we obtained extremely high values for each performance measure for all configuration cases. During the model fitting and testing a set of different parametrizations for the NN, we obtained some interesting findings related to our experimental setup for data acquisition and the resulting data set(s). For each configuration case, we divided the data into training, test and validation data as it is common. We obtained high performance measures on the training as well as on the test and validation data. During model training, we recognized that the training loss steadily decreases while the validation loss steadily increases. Mostly, this indicates that the model suffers of overfitting. However, since our NN performed very good on training as well as on unseen test data, we assume that the model has a good generalization capability, which rules out overfitting as possible cause for the high performance metrics. The divergence of training and validation loss may indicate that the prediction results are high but not very confident. The reasons could be related to our deterministic approach for the labeling and the fact, that a maximum of 25 vUsers never really overloaded the CMS. Assuming that the feature values are too similar in both cases for records labeled with 1 or 0 may be a reason for the high prediction quality. In other words, the values do not vary enough for both labels and a clear allocation is missing. Further tests with different parametrization of the NN showed that NN with a low number of hidden layers, e.g., 1, and a low number of neurons, e.g., 2, counteracts the drift of the training and validation loss without a considerable decrease of the prediction quality. The losses differ more in case of deep NN having more hidden layers and a high number of neurons, e.g., 32, 64 or 128. In case of deep NN, we could counteract the increasing difference between the losses using a sigmoid hidden layer after a *rectified linear unit* (ReLu). Generally, the deep NN seem to perform better using sigmoid activation functions, e.g., *hyperbolic tangent function*, rather than using ReLu's. Thus, we assume that a set of <5 features is highly correlated with the binary target and the classification strongly depends on very few features. After analysis, we obtained that there is a considerable correlation between the target and the CPU and GC features (at least about 0.75). In the following, we present a selection of the obtained results for the LogReg and RF. We do this case wise and illustrate the results using boxplots. Figure 4 shows the performance for case 1 of LogReg and RF of 100 runs with in each randomly sampled training and test data.

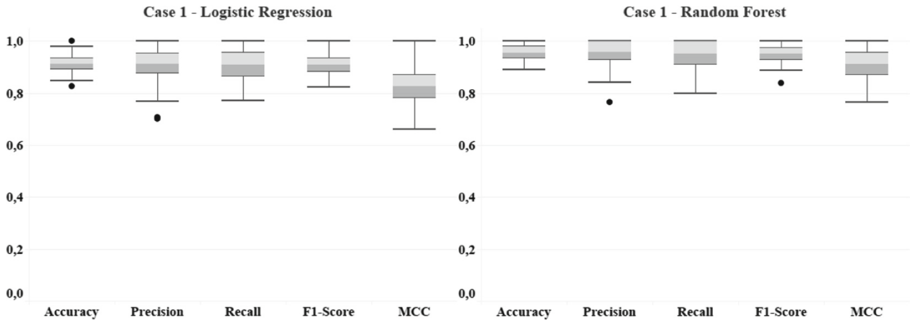

Fig. 4. Results of LogReg and RF for case 1 over 100 runs

We see that both classifier achieved good prediction quality within 100 runs but RF outperforms the LogReg in case of all performance metrics having median values of

0.96 (accuracy), 0.96 (precision), 0.95 (recall), 0.95 (F1-Score) and 0.91 (MCC) while the median values for LogReg are 0.91 (accuracy), 0.91 (precision), 0.91 (recall), 0.91 (F1-Score) and 0.83 (MCC). These results confirm the very good performance of the NN for the classification task on the data of the CMS. We remark that it is difficult to compare the results of a single run for NN and 100 runs for LogReg and RF but as a first impression, all of the three classification methods show a very strong predictive ability. The results of all remaining cases are similarly high without considerable differences. Thus, we summarize that a lead time window of 15 min has no apparent influence on the prediction accuracy. Due to our experimental setup, 15 min are the maximum horizon to forecast the system state of the CMS since we triggered changes of the system state every 15 min to the system as described in Sect. 3.2. This circumstance limits the forecasting horizon within our experiment for further analysis. However, we moreover investigated that an increasing number of past observations considered for analysis of cases with an identic lead time window significantly decreases the range of upper and lower quartiles and the whiskers. Thus, we obtained more stable results with less variation along the 100 runs. Figure 5 illustrates the results of case 13 of LogReg and RF.

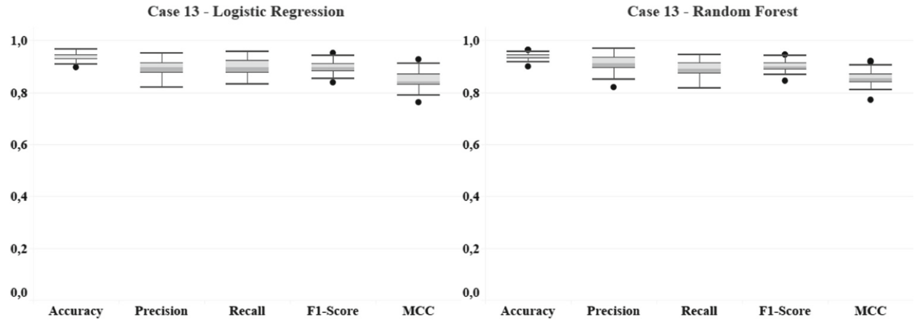

Fig. 5. Results of LogReg and RF for case 13 over 100 runs

For example, the quartiles of the recall measures of LogReg for case 1 are 0.96/0.86 and the whiskers are 1.00/0.77 while we obtained 0.92/0.88 for the quartiles and 0.96/0.83 for the whiskers of case 13. For RF, the recall measures of case 1 of the quartiles are 1.00/0.90 and the whiskers are 1.00/0.80 while for case 13, the quartiles are 0.91/0.87 and the whiskers are 0.95/0.82. This effect is consistently presents in all cases if the number of past observations increases and the lead time window remains unchanged. Summarized, we obtained (very) good prediction quality for LogReg, RF and NN and more stable, i.e., a less degree of variation, results if we consider more past observations to predict the system state with the same lead time window.

6.2 Feature Importance

To give IT-operators guidance about which features are most likely to be important for the judgement of the system state, we use MDA and MDG. The first one quantifies the feature importance by measuring the change of the prediction quality if the measures of the feature are randomly permuted, compared to the original observation. On the other hand, MDG is the sum of all decreases in Gini impurity to a given feature that the RF uses to form a split, normalized by the number of trees [33]. Similar to the performance measures, we calculated measures for MDA and MDG in each run and illustrate a selection of the results using boxplots. Figure 6 presents the results of MDG for case 1 that have a value of ≥5.

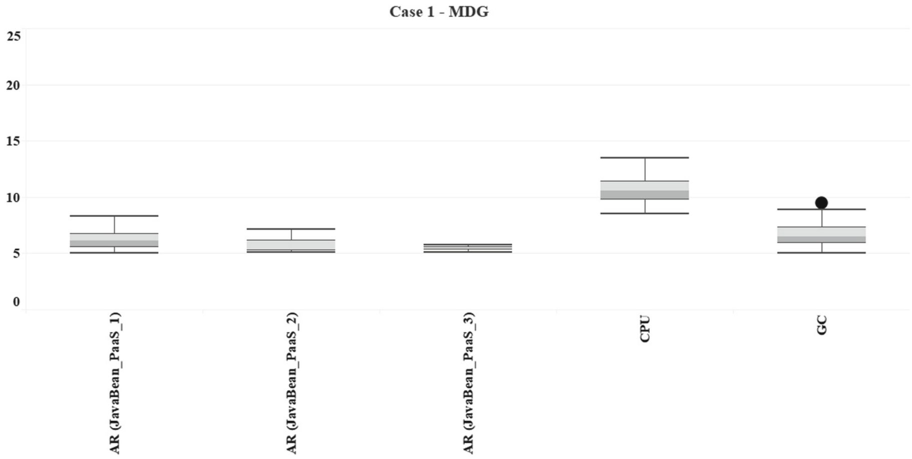

Fig. 6. MDG for case 1 over 100 runs

We see that MDG judges 5 features as important if the threshold is set to ≥5. These features are CPU, GC and three different features of the AR group of metrics, i.e., calls of JavaBeans. Using the same threshold for MDA, it judges 17 features in total as important. For the sake of space, Fig. 7 illustrates the results of the top-10 ranked features only.

We clearly see the overlap: all of the five features that MDG judged as important, MDA judges as important as well. Both measures show that the CPU utilization is the predominant feature. This impression confirms in case of the analysis of other configuration cases, which we but do not illustrate. For example, the increasing lead time window of case 4 increases the importance of the CPU up to a MDG median value of 66.83. MDA confirms this increase for case 4 having a median value of 31.92 for the CPU feature. Moreover, MDA judges the occurrence of an IaaS log file event as important for case 4, after all in 10 of the runs having a median value of 5.40. The importance of log file events is confirmed in several configuration cases, e.g., case 8 and 12, by MDA as well as MDG. Unfortunately, we are unable to derive any generic assumption for this effect since the importance of log file events occurs more likely sporadic. Nevertheless, it confirms the interplay of both types of IT-system data with the overall system state. Summarized, the CPU utilization is after analysis the most promising indicator for the system turning into critical in the most configuration cases having consistently MDA and MDG values of at least ≥5. Furthermore, the CPU utilization is the only feature that consistently shows to be important including its past observations, i.e., t_{-5} etc., for configurations considering multiple copies of the past observations in their data sets. This is consistent

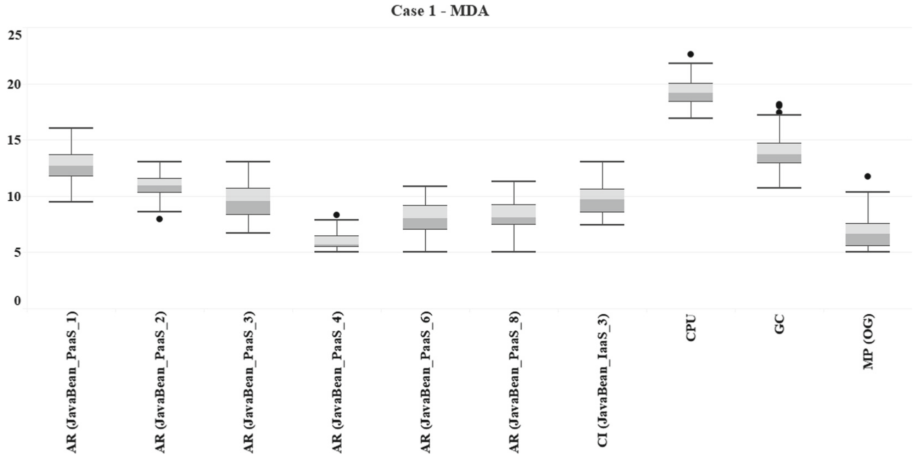

Fig. 7. MDA for case 1 over 100 runs (top-10 ranked features)

with our labeling approach based on the number of vUsers since each user very likely increases the CPU load. However, our results show that we moreover identified different AR, CI, MP or log file related features to be important in different configuration cases. We assume that these features would be difficult to investigate using domain knowledge only. IT-systems contain a high number of different system parameters and their impact to the system state may be hard to recognize without a statistical analysis. Thus, our method and models deliver advanced knowledge about the underlying IT-system and its inner workings related to the overall system state.

7 Threats to Validity

We acknowledge our experimental setup for data acquisition as main threat to construct validity and assume that the low load intensity on the CMS biased the evaluation of the predictive models. At the data-level, we identified that some of the measures of the features do not significantly differ independently whether the label is either 1 or 0. This is the result of:

1. A system state that probably never seriously endangered due to the maximum of 25 vUsers working on the CMS at the same time
2. An experimental and deterministic labeling on data that probably does not contain measures representing a real critical system state

We assume that our experimental setup for data acquisition is the main trigger for the conspicuous high performance metrics. Nevertheless, we addressed the threat of internal validity using an evaluation design with different configuration cases and 100 runs with randomly sampled data for training and test. Thus, at the algorithm-level we considered various parametrizations for our models, trained them on different data and tested them on unseen data to avoid phenomenon like overfitting to be the cause for the high prediction accuracy. Furthermore, we applied a set of different performance measures to ensure a fair and honest evaluation. In case of external validity, we propose a generalizable methodical approach for the joint analysis of textual and numerical IT-system data to predict the system state. However, the nature of ML methods is that they exclusively depend on the data used for the model fitting. Thus, our results are specific to the industrial IT-system used for data acquisition.

Despite all these threats and countermeasures, we emphasize that the main contribution of this work is the process and outline of steps that starts from heterogeneous data of incompatible type (numeric and textual), going through a data type unification for admissibility for statistical analysis, whose interpretation is presented with a discussion of potential pitfalls and possible conclusions. Thus, the threats to validity do not extend to the described method itself.

8 Conclusion and Future Work

We present a method to predict the overall state of IT-systems using a combination of heterogeneous data sources. Our method breaks down limitations of analyzing data with incompatible formats by compiling textual log file information and numeric data into a single prediction model. This method is designed towards explainability to identify root causes with help of statistical methods, which may ease the initiation of countermeasures to avoid system downtimes. We achieved following results:

1. **On RQ1:** We used a set of different data preparation processes to unify textual and numerical IT-system data in a single ML model. Our method requires a minimum degree of domain knowledge and is applicable to any IT-system (although data preparation processes always depend on the specific application and data) but is conceptually generalizable to incorporate domain knowledge if available.
2. **On RQ2:** We see that all models achieved high prediction quality even if the results of the NN seem to outperform LogReg and RF results. We admit that this impression may be biased since 100 runs on randomly sampled data to evaluate the NN were impractical due to the required computational time for training and testing.
3. **On RQ3:** The analysis of the feature importance points out the CPU utilization as most promising indicator to judge the system state. Thus, is should be considered as preferred root cause in case of alarms. It is also admitted that this result is to be taken specific for the experimental setup and may come out of different in other practical instances of systems. Nonetheless, the general reasoning behind this finding does apply to other settings than we described.

Our method and evaluation design allow us to analyze and to predict the overall system state using various available system parameters, covering a wide and diverse range of sources and formats. By analyzing the feature importance, we clearly see that monitoring metrics as well as log file events affect the system state and may be considered as root causes in different configuration cases. This analysis allows us to give IT-operators substantiated guidance on which system parameters to focus on in case of alarms. We believe that such a statistical analysis along all available system parameters accelerates the decision making of IT-operators. Moreover, we think that a detailed analysis would be hard to beat if we only consider domain knowledge and experience of the IT-operators although both are not negligible. Future work will complement the evaluation of our method by experts, i.e., monitoring architects and IT-operators. The expert evaluation will focus and the feasibility, utility and usability of our method in case of its practical applicability. Moreover, we will publish results of an empirical study that investigates the applicability of ML methods specific to the IT-operations area in general. Among others, this study will analyze the tradeoff between high accuracy vs. high explainability of ML models for prediction of IT incidents.

References

1. Kubiak, P., Rass, S., Pinzger, M.: IT-Application Behaviour Analysis: Predicting Critical System States on OpenStack using Monitoring Performance Data and Log Files, pp. 589–596. SCITEPRESS - Science and Technology Publications, Lieusaint - Paris (2020)
2. Kubiak, P., Rass, S.: An overview of data-driven techniques for IT-service-management. IEEE Access **6**, 63664–63688 (2018)
3. Hochstein, A., Tamm, G., Brenner, W.: Service-oriented IT management: benefit, cost and success factors. In: Proceedings of the 13th European Conference on Information Systems, Information Systems in a Rapidly Changing Economy, Regensburg, Germany (2005)
4. Potgieter, B.C., Botha, J.H., Lew, C.: Evidence that use of the ITIL framework is effective. In: Proceedings of the 8th Annual Conference of the National Advisory Committee on Computing Qualifications, Tauranga, New Zealand, pp. 160–167 (2005)
5. Salfner, F., Lenk, M., Malek, M.: A survey of online failure prediction methods. ACM Comput. Surv. (CSUR), **42**, 1–42 (2010)
6. Andrzejak, A., Silva, L.: Deterministic models of software aging and optimal rejuvenation schedules. In: 2007 10th IFIP/IEEE International Symposium on Integrated Network Management, pp 159–168. IEEE, Munich (2007)
7. Cheng, F.-T., Wu, S.-L., Tsai, P.-Y., et al.: Application cluster service scheme for near-zero-downtime services. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 4062–4067. IEEE, Barcelona (2005)
8. Murray, J., Hughes, G., Kreutz-Delgado, K.: Hard drive failure prediction using non-parametric statistical methods. In: Proceedings of the ICANN/ICONIP (2003)
9. Kiciman, E., Fox, A.: Detecting application - level failures in component-based inernet services. IEEE Trans. Neural Netw. **16**, 1027–1041 (2005)
10. Shen, J., Wan, J., Lim, S.-J., Yu, L.: Random-forest-based failure prediction for hard disk drives. Int. J. Distrib. Sens. Netw. **14**, 155014771880648 (2018). https://doi.org/10.1177/1550147718806480
11. Zeng, C., Tang, L., Li, T., et al.: Mining temporal lag from fluctuating events for correlation and root cause analysis. In: Proceedings of the 10th International Conference on Network and Service Management (CNSM), Rio de Janeiro, Brazil (2014)
12. Kiran, R.U., Shang, H., Toyoda, M., Kitsuregawa, M.: Discovering recurring patterns in time series. In: Proceedings of the International Conference on Extending Database Technology, Brussels, Belgium (2015)
13. Kiyota, N., Shimamura, S., Hirata, K.: Extracting mutually dependent multisets. In: Yamamoto, A., Kida, T., Uno, T., Kuboyama, T. (eds.) DS 2017. LNCS (LNAI), vol. 10558, pp. 267–280. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67786-6_19

14. Zöller, M.-A., Baum, M., Huber, M.F.: Framework for mining event correlations and time lags in large event sequences. In: Proceedings of the IEEE 15th International Conference on Industrial Informatics (INDIN), Emden, Germany (2017)
15. Kiernan, J., Terzi, E.: Constructing comprehensive summaries of large event sequences. ACM Trans. Knowl. Discov. Data (TKDD) **3**, 1–31 (2009)
16. Jiang, Y., Perng, C.S., Li, T.: Natural event summarization. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management (2011)
17. Luo, C., Fu, Q., Lou, J.-G., et al.: Correlating events with time series for incident diagnosis. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA (2014)
18. Li, T., et al.: Data-driven techniques in computing system management. ACM Comput. Surv. **50**(3), 1–43 (2017). https://doi.org/10.1145/3092697
19. Imai, K.: Quantitative Social Science: An Introduction. Princeton University Press, Woodstock (2017)
20. Ester, M., Kriegel, H.-P., Sander, J., Xiaowei, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96), pp. 226–231 (1996)
21. Raschka, S., Mirjalili, V.: Machine Learning mit Python und Scikit-Learn und TensorFlow: das umfassende Praxis-Handbuch für Data Science, Deep Learning und Predictive Analytics, 2nd Edn. mitp, Frechen (2018)
22. Zhang, S., Zhang, C., Yang, Q.: Data preparation for data mining. Appl. Artif. Intell. **17**, 375–381 (2003). https://doi.org/10.1080/713827180
23. Kuhn, M., Johnson, K.: Feature Engineering and Selection: A Practical Approach for Predictive Models. CRC Press, Taylor & Francis Group, Boca Raton (2020)
24. Schober, P., Boer, C., Schwarte, L.A.: Correlation coefficients: appropriate use and interpretation. Anesth. Analg. **126**, 1763–1768 (2018)
25. Salgado, C.M., Azevedo, C., Proença, H., Vieira, S.: Noise versus outliers. In: Secondary Analysis of Electronic Health Records, pp. 163–183. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43742-2_14
26. Sauer, S.: Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren und modellieren. Springer, Wiesbaden (2018). https://doi.org/10.1007/978-3-658-21587-3
27. Senaviratna, N.A.M.R, Cooray, T.M.J.A.: Diagnosing multicollinearity of logistic regression model. In: AJPAS, pp. 1–9 (2019). https://doi.org/10.9734/ajpas/2019/v5i230132
28. Hosmer, D.W., Lemeshow, S.: Applied Logistic Regression, 2nd edn. Wiley, New York (2000)
29. James, G., Witten, D., Hastie, T., Tibshirani, R.: An introduction to statistical learning: with applications in R. Springer, New York (2013). https://doi.org/10.1007/978-1-4614-7138-7
30. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd edn. Springer, New York (2009). https://doi.org/10.1007/978-0-387-84858-7
31. Louppe, G., Wehenkel, L., Sutera, A., Geurts, P.: Understanding variable importances in forests of randomized trees. In: Proceedings of the 26th International Conference on Neural Information Processing Systems (2013)
32. Genuer, R., Poggi, J.-M., Tuleau-Malot, C.: Variable selection using random forests. Pattern Recogn. Lett. **31**, 2225–2236 (2010)
33. Calle, M.L., Urrea, V.: Letter to the editor: stability of random forest importance measures. Brief. Bioinform. **12**, 86–89 (2011). https://doi.org/10.1093/bib/bbq011