Leveraging Clustering and Natural Language Processing to Overcome Variety Issues in Log Management

Tobias Eljasik-Swoboda[1][a] and Wilhelm Demuth[2]
[1]*ONTEC AG, Ernst-Melchior-Gasse 24/DG, 1100 Vienna, Austria*
[2]*Schoeller Network Control GmbH, Ernst-Melchior-Gasse 24/DG, 1100 Vienna, Austria*

Keywords: Industrial Applications of AI, Intelligence and Cybersecurity, Machine Learning, Natural Language Processing, Trainer/Athlete Pattern, Log Analysis, Log Management, Event Normalization, Security Information and Event Management, Big Data.

Abstract: When introducing log management or Security Information and Event Management (SIEM) practices, organizations are frequently challenged by Gartner's 3 Vs of Big Data: There is a large volume of data which is generated at a rapid velocity. These first two Vs can be effectively handled by current scale-out architectures. The third V is that of variety which affects log management efforts by the lack of a common mandatory format for log files. Essentially every component can log its events differently. The way it is logged can change with every software update. This paper describes the Log Analysis Machine Learner (LAMaLearner) system. It uses a blend of different Artificial Intelligence techniques to overcome variety issues and identify relevant events within log files. LAMaLearner is able to cluster events and generate human readable representations for all events within a cluster. A human being can annotate these clusters with specific labels. After these labels exist, LAMaLearner leverages machine learning based natural language processing techniques to label events even in changing log formats. Additionally, LAMaLearner is capable of identifying previously known named entities occurring anywhere within the logged event as well identifying frequently co-occuring variables in otherwise fixed log events. In order to stay up-to-date LAMaLearner includes a continuous feedback interface that facilitates active learning. In experiments with multiple differently formatted log files, LAMaLearner was capable of reducing the labeling effort by up to three orders of magnitude. Models trained on this labeled data achieved > 93% F1 in detecting relevant event classes. This way, LAMaLearner helps log management and SIEM operations in three ways: Firstly, it creates a quick overview about the content of previously unknown log files. Secondly, it can be used to massively reduce the required manual effort in log management and SIEM operations. Thirdly, it identifies commonly co-occurring values within logs which can be used to identify otherwise unknown aspects of large log files.

1 INTRODUCTION

Computers and network components like switches, routers or firewalls create log files that list events occuring on them. Originally, these log files were only used for troubleshooting problems after errors have occurred. Since then, technology has been developed that can react immediately to the occurrence of specific events within logs. More interesting use cases arise when one combines log files from multiple machines to get a bigger picture of events occuring within the IT environment of an organization. The process of generating, transmitting, storing, analyzing and disposing of log data is referred to as *log management* (Kent and Souppaya, 2006). Centrally collecting log files from different components provides a number of tangible advantages. Firstly, log files can be accessed even if the machine that originally generated them is no longer available. Secondly, central analysis and correlation is only possible if the relevant log files are centrally stored. The widespread deployment of modern computing and networking machinery generates challenges for organizations attempting to leverage central log management. These challenges are similar to those expressed by Gartner (2011) as the three Vs of Big Data:

[a] https://orcid.org/0000-0003-2464-8461

Eljasik-Swoboda, T. and Demuth, W.
Leveraging Clustering and Natural Language Processing to Overcome Variety Issues in Log Management.
DOI: 10.5220/0008856602810288
In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence (ICAART 2020) - Volume 2*, pages 281-288
ISBN: 978-989-758-395-7; ISSN: 2184-433X

Firstly there is the problem of *volume*: The generated amount of log files can easily reach Terabytes. Secondly there is the problem of *velocity* as depending on what is happening in the computing environment, log files can be generated at a rapid pace. Scale-out architectures, that distribute the workload across multiple machines are capable of handling these volume and velocity problems effectively (Singh and Reddy, 2014). The third problem of big data is *variety*. In the case of log management, it stems from the lack of a mandatory norm for log formats. Essentially, every log file is different. Manually interpreting enormous amounts of such log data can easily overwhelm a human being that attempts this task (Varanadi, 2003).

The process of transforming heterogenous log formats to a common output that is centrally stored is also referred to as *event normalization* (Teixeira, 2017). This is mainly achieved by using parsers that apply regular expressions to extract specific values from logs in previously known formats. These are subsequently stored in a normalized fashion within searchable databases. The data quality within these databases is strongly dependant on the parsing quality. Even though parsing rules for common log files are freely shared on the Internet, these usually only match highly specific fields such as time stamps and source ip adresses. The essential unstructured, natural language message gets frequently stored as a string.

Security Information and Event Management (SIEM) attempts to leverage centrally managed logs to increase the IT security of an organization (Williams and Nicolett, 2005). In practice, specific event types are oftentimes visualized by occurrence per time frame. For instance the amount of failed log in attempts per hour. One can also define alert rules that trigger automated stepts. An example for such an alert rule are to inform a human being if there are more than 10 firewall rejections per minute or that malware was detected on a computer within the network (Swift, 2010). Currently, the only way to identify specific events is to know the log format in advance and having a pattern that can match to the specific event during normalization. As pointed out, there is no norm for log events. Additionally, the format of the log file can change with updates of the utilized software. Therefore, the implementation and updating of Log Management and SIEM systems are labor intensive.

In this work, we introduce the Log Analysis Machine Learner (LAMaLearner). It uses a blend of different artificial intelligence techniques in order to minimize the effort of implementing SIEM and log management practices within organizations. This is especially relevant for Small or Medium Enterprises (SMEs) that oftentimes do not possess the necessary human resources to employ large teams focusing on log management and SIEM. To do so, this paper outlines the relevant state of the art and technology of this field in section 2. Section 3 describes our underlying model which is followed up by section 4 that provides some details about the implementation of this technology. Section 5 evaluates the effectiveness of LAMaLearner for a number of different log file formats and provides information about the time and effort that was saved by using LAMaLearner for Log Management projects. Last but not least, section 6 finishes with our conclusions about the usage of AI to overcome variety challenges in log management and SIEM.

2 STATE OF THE ART

There are many tools for log management. Some frequently mentioned commercial options are *Splunk*, *IBM QRadar*, *Loggly*, *Logentries*, and *sumo logic* (Splunk, 2019) (IBM, 2019) (Loggly, 2019) (Logentries, 2019) (Sumo Logic, 2019). Most of the aforementioned systems are cloud based and require a connection to the provider in order to perform the necessary log management. A popular open source solution is the *Elastic Stack* (formerly known as *ELK Stack*), which is maintained by the company *ElasticSearch* which also offers support for the solution (Elastic, 2019). This company's full-text search engine goes by the same name and is an integral part of the *Elastic Stack*. Another popular open source solution is *Graylog* which is maintained and supported by the company of the same name (Graylog, 2019). To the best of our knowledge, all these log management solutions either require manual pattern defintions to match relevant known events or provide taxonomies of relevant events for specific systems. None use artificial intelligence technology for this purpose. In the context of log management, AI technologies are frequently used for anomaly detection of aggregated event occurences per time frame instead of the identification and representation of relevant events (Splunk, 2019) (IBM, 2019) (Elastic, 2019).

A detailed examination of these log management technologies as well as all contemporary AI techniques goes well beyond the scope of this paper. Therefore the remainder of this section focuses on relevant approaches useful for overcoming the aforementioned log management variety problem.

Vaarandi (2003) proposes a data clustering algorithm for mining patterns from event logs. Different from other text clustering approaches, this algorithm has the key insight, that log messages are actually created by fixed patterns in which variables are substituted by their specific values. Vaarandi's algorithm uses this insight by the creation of *1-regions*, which are specific terms at specific positions within multiple events. Events that share multiple 1-regions are candidates for clusters. The 1-regions essentially provide the fixed parts of the message while the intermediate words form the variables that are used in the messages. Vaarandi's algorithm works in $O(|events|)$ time as it only needs to iterate a fixed amount of times over the events to identify 1-regions and group common events into clusters. It also outputs a representative pattern that can be used as regular expression to match to all events making up the cluster.

Besides the clustering of events and generation of representations for these clusters, interactive labeling and machine learning based text categorization are important corner stones of LAMaLearner. The Cloud Classifier Committee (C3) is a collection of microservices that ease the implementation of text categorization solutions (Swoboda et al., 2016). In their work, Eljasik-Swoboda et al. (2019) extended the core C3 idea and described two relevant concepts: Firstly, the trainer/athlete pattern which allows scale-out for machine learning based text categorization tasks. Here, a trainer node computes a model that is shared with athlete nodes. The actual inference work is performed by the athlete nodes.

Secondly, the TFIDF-SVM was proposed. This service implements the trainer/athlete pattern and uses the LibSVM library to implement Support Vector Machines (SVMs) for supervised machine learning (Chang and Lin, 2011). Besides the SVMs, TFIDF-SVM works with a feature extraction and selection method that is inspired by the TFIDF formula common for information retrieval (1). It essentially measures the importance of how representative certain terms (t_k) are for certain documents (d).

$$tfidf(t_k,d_i)=\#(t_k,d_i)*\log(|TS|/\#TS(t_k)) \quad (1)$$

The TFIDF-SVM trainer service selects the most relevant features based on their TFIDF values and combines them with a SVM model that is evaluated using n-fold cross-validation. TFIDF-SVM was evaluated in the challenging argument stance recognition task and achieved up to .96 F1 for previously unknown arguments about the same topic it was trained on. Encouragingly, it was also able to achieve up to .6 F1 when determining the stance of arguments for previously unseen topics. This suggests that TFIDF-SVM models can be transferred to new problems with completely unseen data without modification. The aforementioned is highly interesting for the log analysis use case as not knowing the precise format of new log files is the overall challenge this research aims to overcome.

Chawla et al. (2002) introduced the Synthetic Minority Over-sampling Technique (SMOTE). The idea is to overcome issues arising from imbalanced datasets by synthetically oversampling minority classes so that the acutal machine learning model is trained on a more balanced dataset. This is crucial for the log analysis problem because relevant error messages are oftentimes few and far between repeatedly occuring success messages.

Named-entity recognition is the act of identifying specific named entities, such as locations from text (Jurafsky, 2009). These could be names or locations in which multiple strings can point to the same entity or type of entity. For instance, *Malta* and *Austria* are both countries. The next section illustrates how these design patterns and techniques are used to create LAMaLearner.

3 MODEL

LAMaLearner starts its operation without any information about the log format and content. To start working with log messages, LAMaLearner implements a key-value store for evens: A numerical key is used to identify an object that contains a message string and a time string. As there can be many different time formats, LAMaLearner ignores the time value only offering this field for users to read. It can also be left empty.

As soon as LAMaLearner is provided with events, it can create clusters of events with corresponding representations by using a modified version of Vaarandi's algorithm described in section 2. Our modification is in the identification of nested clusters, so that LAMaLearner can also compute sub-clusters of found outer clusters. This operation however requires the comparison of all clusters with each other, so that this operation requires $O(|events|+|clusters|^2)$ steps. Vaarandi's algorithm has a threshold hyper-parameter that determines in how many events the same term has to occur at the same place to be considered a 1-region (see section 2). Increasing this parameter decreases the amount of clusters that are found. The result set also displays all stored events that do not fit into any of the generated clusters.

The individual cluster representations are a sequence of fixed 1-region terms $F=\{f_1,\dots,f_n\}$ intermixed with variable terms $V=\{v_1,\dots,v_m\}$. For all events within one cluster, F is identical, while the values for V contain different terms. We use this property for the created clusters for two additional analysis steps: Firstly, the detection of named entities. In the current version, we use lists of terms or regular expressions to represent named entities such as usernames, DNS names, IP addresses, and email addresses. Identifying named entities for F is performed by matching $f_1,\dots,f_n$ to the stored entities. As $v_1,\dots,v_m$ are lists of different values per event that was assigned to the cluster, these are actually each a list of different terms. Therefore, LAMaLearner attempts to identify, if all terms within $v \in V$ are matching to the same regular expression. This can be used in the cluster representation by creating representations such as *Router {hostname} interface {ip} down* in which *{hostname}* and *{ip}* are named entities represented by common regular expressions.

The identification of F and V per cluster also yields another opportunity for analysis. Namely the identification of commonly co-occurring variables in V. A matrix of how often variable values co-occur in the same event can be computed. Without additional knowledge about the analyzed log files, one can infer related values within clusters. For instance co-occurring hostnames and IP addresses or Microsoft Active Directory Security Identifier and human readable user names. These commonly co-occurring values are provided to the user after clustering events.

This way, users can quickly gain an overlook about the available messages within the analyzed log files. Depending on their required application, users can start to annotate messages accordingly. Examples for appropriate labels largely depend on the analyzed log files. Practical examples from an Apache Tomcat application log file are *success*, *exception*, *database connection terminated*, *SQL syntax error* and *client caused db error*. Example labels for security log files can be *successful logins*, *failed logins*, and *malware detected*.

In comparison to annotating thousands of individual events, this clustering step compresses the task to annotating tens of clusters, massively speeding up the process. Annotated clusters are stored in a portable way. This means, that every cluster object contains a list of annotated labels. LAMaLearner implements the trainer/athlete pattern to compute its models. The trainer node extends the TFIDF-SVM approach with the synthetic minority oversampling technique as follows:

Labelled clusters form the documents for which a TFIDF-matrix is computed. As cluster labels are known, the amount of clusters per label is also known. To present the subsequent process with a more balanced problem, LAMaLearner generates synthetic samples for all minority classes by randomly concatenating terms occurring in existing cluster representations of the minority labels. For the sake of repeatability, LAMaLearner performs this process using a fixed random seed. After this creation of synthetic samples, there is an equal amount of training samples for each class. Additionally, LAMaLearner computes the average amount of labels that are assigned to each existing sample cluster. LAMaLearner subsequently uses the same feature extraction scheme as described by Eljasik-Swoboda et al. (2019). To do so, it only takes real samples into account. This means, that it ignores the synthetically generated samples for its feature extraction and selection scheme.

After determining relevant terms for feature extraction and selection, LAMaLearner triggers an n-fold cross-validation process. It is important to note, that only real labelled clusters are used as evaluation samples. The generated synthetic samples are only used for training. As with TFIDF-SVM, LibSVM is used to compute hyperplanes capable of identifying appropriate labels. While TFIDF-SVM works with an assignment threshold to determine if documents should be assigned to a certain label, LAMaLearner extends this decision with the average amount of assignments learned from its un-augmented training set. It is noteworthy, that for this supervised learning process, the exact positioning of terms within events is intentionally ignored. This information is only used for the unsupervised clustering phase. The purpose for ignoring the exact positioning of relevant terms is to create robustness against changing formats. While the ordering of specific terms can change with every software update, semantic shift happens much slower. For example logged terms like *failure*, *exception*, *fatal*, or *malware* don't change in meaning depending on where they are in the log message.

After *n* models have been computed, the best is selected. The selection metric (precision, recall, F1, microaverage, macroaverage) can be selected before training. This best model is then stored by the trainer node, so that any athlete node can obtain the model by querying it. The model object also contains relevant metadata about the model. Besides information about the creator, log type and use case it has been trained for, detailed evaluation results are stored.

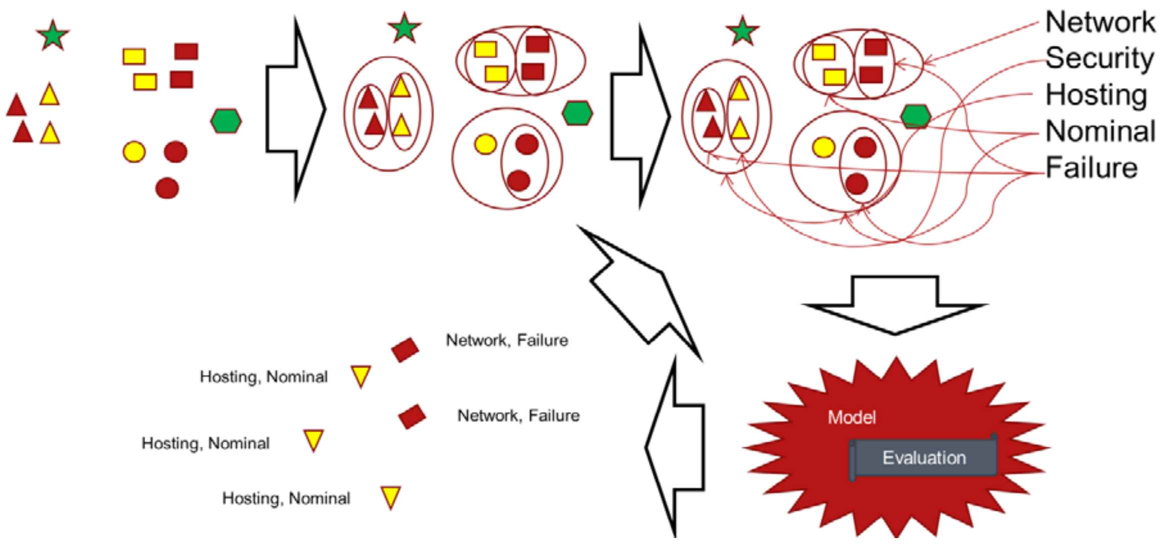

Figure 1: LAMaLearner overall learning process: Firstly, unknown event messages are collected in nested clusters. These can be labeled in < 1% of the time necessary to label all individual messages. Based on these labelled clusters, a model capable of labelling clusters and individual messages is computed. It contains its effectiveness evaluation. This model can successfully be used to label previously unknown log messages in the same and different formats. If mistakes occur, these can be corrected and the model can be improved.

This way, whenever an athlete node is using this model, users can display how effective the used model was during evaluation.

A LAMaLearner athlete node needs an active model so that it can automatically annotate any event or cluster with a label. A big advantage of SVMs is their speed and low resource consumption. Combined with the feature selection scheme, large amounts of events can rapidly be automatically labelled. This task can easily be scaled out across multiple machines. LAMaLearner also allows for the definition of fixed rules. These rules are made up of indicator terms which occurrence strongly suggest a specific label. Rule based results can be combined with the active model either using a logic AND or a logic OR operator. In addition to label individual events, the before mentioned approach to identify named entities is used on every event message. These allow for filtering of labels in combination with entities and values, for instance to display only failure events for specific usernames.

Besides manually annotating clusters, LAMaLearner can also work with manually annotated events to increase the size of its training and evaluation set. This allows for an interactive training loop in which false results can be corrected and a new training process can be triggered to further increase a model's effectiveness. Labels are assigned with a risk score which is a value indicating the urgency of events having this label. As multiple labels can be assigned to each event or cluster, LAMaLearner computers an overall risk score per cluster or label using formula 2.

$$r(e_i) = \frac{\prod_{j=0}^{|l_i|} p(e_i, l_j) * r_j}{|l_i|} \qquad (2)$$

The formula to compute the overall risk of event or cluster i $r(e_i)$ is the product of the individual probabilities for this event or cluster to have label j $p(e_i,l_j)$ computed by the model and fixed rules multiplied with the assigned risk of label j r_j. The utilized SVMs output a probability for an assigned label. Indicator terms are also configured with a probability to indicate certain labels. LAMaLearner automatically removes unlikely labels from the label set of event or cluster i l_i. Therefore it seldom multiplies all label risks per event only concentrating on relevant values. LAMaLearner can be configured with an overall risk threshold. Whenever an instance identifies an event or cluster of a higher risk score than this threshold, it can call a freely configurable external program via CLI. This can be used to raise alarms or initiate automated further actions depending on the detected labels. We chose this multiplication based method of computing overall risk values per cluster or event as it allows for negation. For instance one can model different aspects of a log managed environment with different base risk values. E.g. error messages in firewalls can be regarded as more important than those of storage components. One can also model success messages with a negative risk value. This means, that success messages of different components get negative risk scores while error messages obtain risk scores in relation to the base risk score of impacted component class.

4 IMPLEMENTATION

The core idea behind Hadoop's popular MapReduce programming model is to move the program to where the data resides instead of the other way around (Dean and Ghemawat, 2008). We took this idea to mind when designing LAMaLearner in a way that it can easily be transferred to where ever necessary and scaled out where possible. This way, potentially sensible information contained within log files do not have to leave a secured network environment. To meet this objective, we based LAMaLearner on Java and packaged it as fat jar file. It communicates via a REST/JSON interface. This way, it can operate on any platform that supports java and has a network interface, allowing for integration into many existing log management technologies. In order to ease direct interaction with LAMaLearner, it also renders a Web GUI which is based on JavaScript and communicates with the underlying REST/JSON interface. To do so, LAMaLearner is based on the Dropwizard framework (Dropwizard, 2019). All relevant data is stored in memory.

The creation of clusters or assignment of labels to uploaded events can be triggered by sending POST requests to the appropriate resources. Hyper-parameters for these processes are transferred as JSON objects to the LAMaLearner instance. LAMaLearner keeps track of whether a clustering or event labelling process is in progress. If that is the case, a new process cannot be triggered. While events are clustered, annotated with labels, or a new model is computed, LAMaLearner returns a progress list that indicates how many of the necessary steps have been performed. In order to keep the web server responsive and maintain the ability to query the LAMaLearner instance for existing results, all clustering, labelling and model creation processes are executed in independent threads.

This setup is very flexible and has no external dependencies except for Java. Intentionally, LAMaLearner does not implement a database to persistently store events, clusters, labelled events or created models. All data is kept in memory and can be exported as JSON object which in turn can be imported into another instance. As log files themselves are usually not stored as JSON objects, this creates the need to interact with an existing log management solution that is capable of packaging log entries into JSON objects and trigger LAMaLearner. It also has to be able to store results and process them further, for instance by triggering alerts if there are more than 10 firewall rejections within a specific time interval. For this purpose we use an in-house technology called *Modular Abstract Data processing Tool* (MAD2).

This piece of software can collect log files from lots of different source systems. Using a relational data format, Huffman encoding and a multitude of compression algorithms, MAD2 can reduce the storage requirements for log files. A single instance can also process up to 10.000 events per second, making this software the interface between the actual log files and the LAMaLearner AI.

5 EVALUATION

LAMaLearner is a useful tool for exploring new unknown log formats and processing them into a labelled form for further downstream analysis. To evaluate its capabilities, it was tested with different real life log files. The person inspecting these log files had no prior knowledge about the environments they have been created in.

The first tested log file was a Microsoft-Windows-Security-Auditing log file. The analyzed part of the log file contained 10,000 events. Using a threshold of 2, LAMaLearner condensed the messages to 79 nested clusters. Manual inspection showed that the events contained mainly three types of events: Successful logins, successful logoffs, and login failures. These clusters were then annotated with the labels *success* and *failure*. LAMaLearner was able to create a model with F1=1. In combination with named-entity based filtering, users can quickly identify which users or hosts are involved in failures. Additionally, the variable co-occurrence feature of the clustering process allowed matching Microsoft Security Identifier to the human readable user name only from automatically analyzing the log files.

Interestingly, the same model was then used on a Check Point Firewall log which had a vastly different format. In a manual evaluation of 50 events, LAMaLearner was capable to tell successful connections (label success) from rejected connections (label failure).

In another test, LAMaLearner was presented with a mixed collection of events coming from three different source systems: Microsoft-Windows-

Security-Auditing logs, Check Point firewall logs, and an Apache webserver access log. This mix contained 18,110 events. LAMaLearner created 47 nested clusters with an assignment threshold of 5 (56 with an assignment threshold of 2). A model capable of telling different source systems apart obtained an overall .93 F1 value. In both these cases, the clustering approach reduced the time necessary to label a large quantity of events by more than two orders of magnitude. This means that the time required to annotate log events for machine learning purposes was reduced to less than 1% of the original amount of required time. The learned models were highly effective in detecting the correct label for any event. On a Windows 10 machine with an Intel I7-7870 (4 cores, 2.9 GHz) and 16 GB RAM, LAMaLearner is capable of labeling >10,000 events per second. Besides this core use case of correctly classifying events without requiring predefined regular expressions, LAMaLearner also provides interesting insights into unknown log files. Specifically by clustering the afore mentioned Check Point firewall log revealed which type of network traffic was routed over this firewall.

An interesting observation was made when analyzing the log files of an Apache Tomcat application server log. 963 events logged in one day were grouped into 56 clusters. Upon first view, four reasonable labels were determined: *Success, static exception*, *database connection terminated*, and *exception*. The 56 clusters from that day were manually annotated with these labels and a model with F1=1 was computed. As evaluation, the model was tasked with labelling the events of the next day of this specific server. The second day's log file contained 1,713 messages that were quickly labeled with the available labels. Inspection revealed that known types of events were correctly labelled. There however were new types of events that were either interpreted as success or exception. The first one was an *SQL Syntax Error* that has been logged in the application server log file. The second one was a *client caused database error*, where a user attempted to delete an entry that didn't exist. By labeling these events, an updated model that can determine these different labels with F1=1 was easily created. Besides having the ability to track security related entries in a SIEM platform, this revealed potential errors in existing software that one might not have noticed in production.

Another interesting result was obtained by leaving the world of classical information technology components and analyzing the log files of an industrial control computer. 1601 Events were clustered into 54 clusters. Manual inspection has shown that there are three general classes of events within the log file: *General status information*, *malfunction*, and *exceeding thresholds*. Using LAMaLearner, these can subsequently be visualized in a dashboard and enable the operator of the environment with a quick overview about what has happened in previous time intervals. This provides operators with a quick update about the environment on shift changes. Named-entity based filtering and correlation between variable values in clusters also allowed to quickly filtering which component failed or exceeded its threshold. In case of exceeding thresholds, the values can then quickly be checked and actual malfunctions can get investigated. As this can be performed on multiple control systems at once, a much better overview is gained.

As last experiment, we generated an artificial log file in two different formats. It contained 1000 events which LAMaLearner clustered to 8 high-level clusters (two of which contained a large collection of sub-clusters). This artificial log contained events about simulated traffic over two different network routers. The two high level clusters each represented a different router. Their sub-clusters were connections from different peers to these routers. Overall 98 peers were simulated, each of which had their own cluster. The clusters found in the first log file were annotated with the labels *nominal* and *failure*. The latter was used for interface outages. Again, a high effectiveness model (F1=1) was generated by LAMaLearner. This model was then applied to label the events of the second log. The second log file contained the same information per event but had a completely different ordering of the individual variables and different accompanying words and characters between those. In this second log file, the model was also able to identify *nominal* and *failure* events with F1=1. Even though this last experiment was not conducted with a real-life log file, it illustrates LAMaLearner's robustness against changing log formats as soon as models to label said formats have been learned.

6 CONCLUSIONS AND FUTURE WORK

In this work, we have introduced a flexible method to overcome variety issues in log management by engineering multiple state of the art AI methods into a single powerful solution. Our contributions can reduce the amount of manual effort in log management projects dramatically. It can also shine a light on previously undiscovered log file entries as it allows their exploration in a reasonable time frame. Additionally frequently logged variables such as hostnames or users can be identified for further investigation. These capabilities are highly interesting for small or medium enterprises that intent or have to use log management but do not have the necessary personnel to successfully implement such a practice. It also is not limited to information technology security logs but can also be used within industrial applications to reveal hidden patterns within such environments. Because of its practical implementation, LAMaLearner can be introduced into any relevant system architecture and can handle large amounts of data by having been designed to scale out form the beginning. The fact that no connection to an external provider is necessary and explanations for labeling decisions can be generated the same way as explained by Eljasik-Swoboda et al. (2019) make this software safe to use under strict privacy legislature like the European Union's GDPR (EU, 2016).

In future works we will use LAMaLearner generated event labeling results as input for time series anomaly detection and regression computation. The current version is limited to identify named entities from single words. As of now, word n-grams cannot be analyzed. Therefore additional ways to create a *named entity recognition* (NER) component minimizing manual effort in its definition will also be researched.

REFERENCES

Chang, C., Lin, C., LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology, volume 2, issue 3, pp 27:1 –27:27, 2011

Chawla, N. V., Bowyer, K. W., Hall, L. O., Keelmeyer, W. P. 2002. SMOTE: Synthetic Minority Over-sampling Technique, Journal of Artificial Intelligence Research, Issue 16, pp. 321-357

Dean, J., Ghemawat S., 2008. MapReduce: simplified data processing on large clusters. In Communications of the ACM issue 51, pp. 107-113.

Dropwizard 2019, Production-ready, out of the box. https://dropwizard.io Accessed September 12, 2019

Elastic, 2019. Open Source Search & Analytics Elasticsearch | Elastic https://elastic.co Accessed September 12, 2019

Eljasik-Swoboda, T., Engel, F., Hemmje, M., 2019. Using Topic Specific Features for Argument Stance Recognition. In: *Proceedings of the 8th international conference on data science, technology and applications (DATA 2019)*, DOI: 10.5220/0007769700130022

Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance); OJ L 119, 4.5.2016, p. 1–88;

Gartner, 2011. Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data, http://www.gartner.com/newsroom/id/1731916 Published June 27, 2011 Accessed May 2, 2016

Graylog, 2019. Industry Leading Log Management | Graylog https://graylog.org Accessed September 12, 2019

IBM, 2019. IBM QRadar SIEM – Overview, https://www.ibm.com/us-en/marketplace/ibm-qradar-siem Accessed September 12, 2019

Jurafsky, D., Martin, J. J., 2009. *Speech and language processing. An introduction to natural language processing, computational linguistics and speech recognition.* 2nd edition, Upper Saddle River, N.J., London: Pearson Prentice Hall (Prentice Hall series in aritificial intelligence), pp 761 ff.

Kent, K., Souppaya, M., 2006. Guide to Computer Security Log Management, Recommendations of the National Institute of Standards and Technology (NIST), DOI: 10.6028/NIST.SP.800-92

Logentries, 2019. Logentries: Log Management & Analysis Software Made Easy. https://logentries.com Accessed September 12, 2019

Loggly, 2019. Log Analysis | Log Management by Loggly https://loggly.com Accessed September 12, 2019

Singh, D., Reddy, C. K., 2014. A survey on platforms for big data analytics, Journal of Big Data. DOI: 10.1186/s40537-014-0008-6

Splunk, 2019. SIEM, AIOps, Application Management, Log Management, Machine Learning, and Compliance. https://splunk.com Accessed September 12, 2019

Sumo Logic, 2019. Log Management & Security Analysis, Continuous Intelligence, Sumo Logic. https://sumologic.com Accessed September 12, 2019

Swift, D., 2010. Successful SIEM and Log Management Strategies for Audit and Compliance, White Paper SANS Institute, https://www.sans.org/reading-room/whitepapers/auditing/paper/33528 Accessed September 5, 2019

Swoboda, T., Kaufmann, M., Hemmje, M. L., Toward Cloud-based Classification and Annotation Support, Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016) – Volume 2, pp. 131-237, 2016

Teixeira, A., 2017. Get over SIEM event normalization. https://medium.com/@ateixei/get-over-siem-event-normalization-595fc36559b4 Accessed Sept. 16, 2019

Varanadi, R., 2003. A Data Clustering Algorithm for Mining Patterns from Event Logs. In: Proceedings of the 2003 IEEE Workshop on IP Operations and Management, ISBN: 0-7803-8199-8

Williams, A. T., Nicolett, M., 2005. Improve IT Security with Vulnerability Management, Gartner Research ID G00127481