Implementing Scalable Structured Machine Learning for Big Data in the SAKE Project

Simon Bin
AKSW Research Group
University of Leipzig
Leipzig, Germany
sbin@informatik.uni-leipzig.de

Patrick Westphal
AKSW Research Group
University of Leipzig
Leipzig, Germany
patrick.westphal
@informatik.uni-leipzig.de

Jens Lehmann
Fraunhofer IAIS and
University of Bonn
Bonn, Germany
jens.lehmann@cs.uni-bonn.de

Axel Ngonga*
Data Science Group
University of Paderborn
Paderborn, Germany
ngonga@upb.de

***Abstract*—Exploration and analysis of large amounts of machine generated data requires innovative approaches. We propose a combination of Semantic Web and Machine Learning to facilitate the analysis. First, data is collected and converted to RDF according to a schema in the Web Ontology Language OWL. Several components can continue working with the data, to interlink, label, augment, or classify. The size of the data poses new challenges to existing solutions, which we solve in this contribution by transitioning from in-memory to database.**

***Keywords*-structured machine learning; rdf; semantic web; big data; classification algorithms; open source;**

I. Introduction

A. RDF and Semantic Web

The Resource Description Framework RDF[1] is the corner stone of the Semantic Web. It is a family of W3C[2] standards and we can observe that the available public RDF data is growing in the Linking Open Data Cloud.[3]

This uniform way to represent linked data, combined with related standards such as the Web Ontology Language OWL[4] provides an opportunity for new interesting and challenging tasks, such as detecting patients with particular diseases [1].

In many cases, machine learning on structured data is of central importance to achieve these goals. These learning approaches commonly rely on reasoners to check whether a set of objects in that data belong to a certain hypothesis concept. Many reasoners have been developed to support this task, of which most load all the data they require into memory. For a list of reasoners accessible via the OWL API interfaces see here.[5]

With the growing Data Web, investigation into how to apply machine learning and in particular structured machine learning to such very large data sets is necessary.

Incidentally, in this paper we are going to look at private (non-public) use cases of the web of data. In many cases it makes sense for businesses to keep their data to themselves. Thanks to RDF, this is easily achieved while at the same time offering a perfect way to introduce links into the open (public) data cloud [2].

In that process we have discovered a performance issue with OWL reasoners when scaling to large data sets. Note that the most general profiles of OWL are inherently intractable.[6] However, for machine learning, it can still be useful to work with tractable subsets. On the other hand, it is possible to use the SPARQL[7] query language to retrieve RDF data from graph databases in an efficient manner. Hence, in [3] we created a bridge between SPARQL and OWL, and we are able to answer OWL queries on large amounts of data (under certain assumptions). For that purpose, we ignore the open-world assumption of OWL and argue that a closed-world view and unique names assumption is sufficient for our structured machine learning purposes. Taking those requirements in a machine learning context into account, we have defined a formal rewriting procedure mapping OWL instance checks (as the most frequent reasoning procedure in the refinement operator based machine learning algorithm [4]) to SPARQL queries. This result enables the use of large amounts of existing OWL and RDF data to solve classification problems. We then can also make use of this approach on the event log data that we are going to collect in the next section.

B. Project

In the SAKE project[8] (Semantic Analysis of Komplex Events) the combination of Semantic Web Technologies like RDF and OWL with event logging and monitoring data is explored.

In the premises, we have automated devices which produce large amounts of data such as message logs or sensor

* also at AKSW Research Group, University of Leipzig, Leipzig

[1] http://www.w3.org/TR/rdf11-concepts/
[2] http://www.w3.org
[3] http://lod-cloud.net/
[4] http://www.w3.org/TR/owl2-overview/
[5] http://owlapi.sourceforge.net/reasoners.html
[6] Complexity of OWL 2 DL (a common subset of OWL) is 2NEXPTIME
[7] http://www.w3.org/TR/sparql11-query/
[8] https://www.sake-projekt.de/en

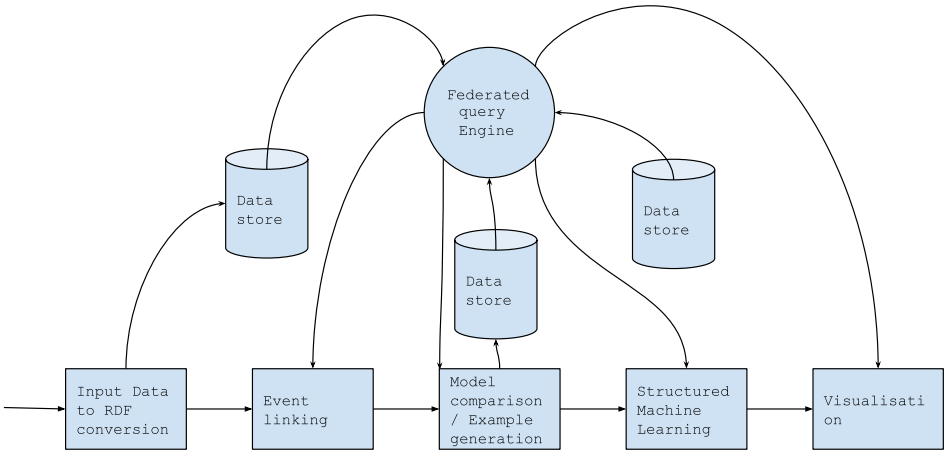

Figure 1: The data processing pipeline (arrows from left to right) in the SAKE project and their interaction with the databases (store, retrieve)

readings. The question we want to answer in this contribution is, (how) can the Symbolic Machine Learning techniques from the domain of Semantic Web be applied to this data?

The goal of doing so would be to get a better understanding of the data, creation of more powerful tools to access, query and explore the data, and derive hidden knowledge such as symbolic classifiers for error events through an automated fashion.

The general concept and vision of the SAKE platform has been described in [5]. Here, we want to look at the individual challenges a bit more in depth and describe their actual implementations.

C. Use case: Event log analysis

First, a look at a real situation at one of the use case partners. Our partner deploys printing machines. Every day, each of their machines produces roughly 20000 messages on the main message bus. At 7.5 kilobytes per message, this amounts to 150 MB per day. In one company, there can be tens to hundreds of machines deployed. And sometimes it may even be worth comparing machine behaviour across companies.

The data needs to be processed and analysed. To understand how to go from the input data to the analysis step, a data processing pipeline has to be built. The final pipeline is presented in Figure 1 with the details of each step described further down.

Analysis of the data provides several benefits: better understanding of the customers, potential for fault detection in machines or processes, early correction of future failures and thus is good for business. However, the data volume is already quite enormous and the human engineers and service technicians can benefit from computer assistance in this task.

The SAKE project also considers other use cases, such as compressor behaviour data in the chemical industry or for business application monitoring in information technology.

While the rest of this paper will describe the steps with the first use case in mind, the basic techniques are the same for our other use cases. Note also, that most components can be exchanged for other components of a similar type.

The paper is structured as follows: In section II, we summarise the challenges of our project. In section III, each step is explained in more detail. Then, the integration of all components is explained in section IV and finally, we conclude with a discussion and point out future work in section V.

II. Problem statements

First we give an overview of the challenges in the SAKE project and our solution to them.

1) Event data is in a custom XML format or other proprietary format.
   - → Develop an ontology describing the input data.
   - → Transform the data into RDF format using a custom built transformation engine.
2) Reasoners cannot load event data into memory (due to size), or not within a specific time.
   - → Store data in graph database designed to handle RDF data instead.
   - → Develop a new adaptor that the algorithms can use to query the database instead of using a reasoner. Understand the implications of omitting the inferencing for which the reasoner was responsible, and design alternative methods to replace this functionality, where needed.
3) Positive examples required for supervised machine learning are not readily available in the collected data.
   - → Unsupervised or human guided classification algorithms derive input examples, based on a variety of possible methods (for example by employing a statistical normal model).

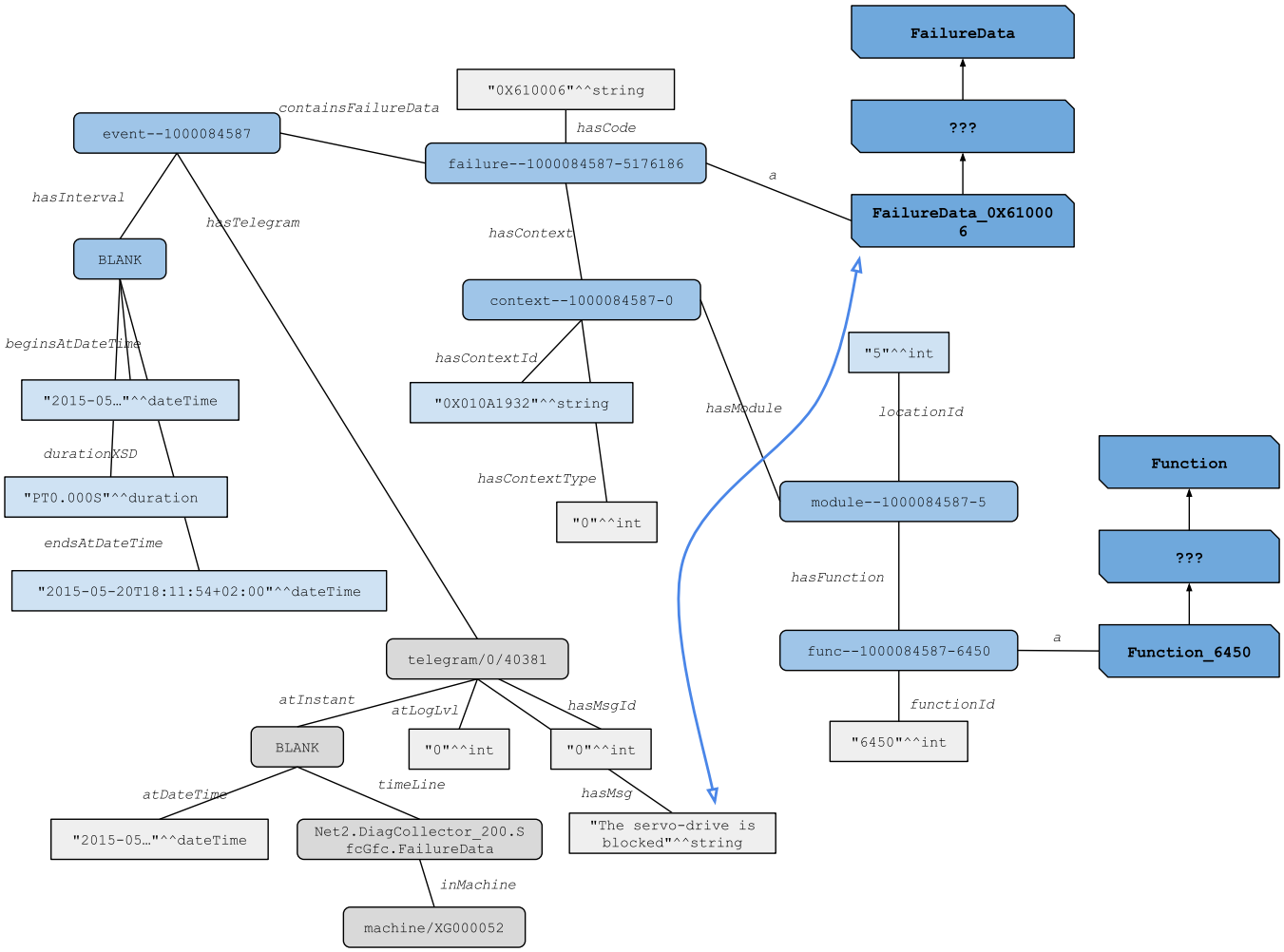

Figure 2: An excerpt from the preliminary ontology schema and example data for one of our use cases, event log analysis of machines (Classes on the right side, in **bold**)

4) Missing explicit relationships between events leaves the structured machine learning less efficient.

   → Linking algorithms are developed to interlink the events based on temporal relations as well as similar events between machines.

5) Data is not distributed and not local to machines of data origin.
   Queries have a speed bottleneck.

   → Develop federated query engines to store the data at the points of collection and optimise the query speed at the same time.

6) The possible machine learning results (models) are expressed as "technical" class expressions which may be difficult to grasp for the service engineers.

   → Offer a visual and exploratory interface to drill down into the result set described by (parts of) the expression.

III. Approach

Now we will briefly expand on each of these topics.

A. Storage

The next step is to put the data where it belongs – into databases. For that, there are already several possibilities. In the project, we decided to deploy the AI4BD Ontos[9] Quad graph database. For the purpose of easier reproducibility, we based the results in this paper on Virtuoso Opensource,[10] since it is freely available. Nevertheless, it cannot be expected that each graph database implements the standard perfectly (similar to the various SQL dialects). For this reason, it may be necessary to specialise the Query implementations inside the machine learning algorithm. We have implemented versions for both Virtuoso and Quad.

[9] http://ontos.com/

[10] Virtuoso open source edition
https://github.com/openlink/virtuoso-opensource

Even after creating the database set up, collecting more and more data would mean increased storage requirements. To better handle this, we developed a federated query engine[11] on top of graph databases [6]. This allows us to distribute the data and also investigate whether there are preferential distribution methods (research is still ongoing on that topic [7]). Furthermore, each of the machine centres is fitted with its own data store containing the data of the local machines.

B. Data Acquisition and Access

For point 1), the solution was to develop a transformation engine which connects to the event log output of the machines, processes the events into RDF data format and then stores them into a RDF graph database. The application is scaling to big data thanks to the usage of Apache Spark.[12] Prior to that, an ontology has been developed to describe the nature of the event data. An example of the XML export provided by the machine log is on display in Listing 1 on the next page. It is then converted into RDF/OWL and parts of the ontology together with some instance data can be seen in Figure 2. The blocks marked with question marks indicate that there may be more semantic knowledge still hidden inside the engineer minds that has not been RDFised yet. This step will have to be repeated for any new, non RDF data source that should be integrated into our approach.

Alternatives and helpful tools exist: if the data is already in a structured format such as a relational database, mapping tools like [8] or Ontology Based Data Access as in [9] can be used. However, the data must be understood properly in any case in order to design the schemata and employ meaningful analysis.

After the transformation has succeeded and the RDF/OWL data has been stored, the next step to use a Structured Machine Learning Framework is to load the data. Due to the logical implications of the ontology axioms (so that the inferences can be correctly calculated), we normally use an OWL Reasoner such as Pellet.[13]

Unfortunately, the Structured Machine Learning framework DL-Learner [10] could not be applied directly on the data because of the size. Even just loading the data of a single day and ten machines into memory (approximately 3 gigabytes of data) would make the framework and reasoner use up more than 60 gigabytes of system memory (at which point our system memory was exhausted). Other frameworks that we investigated had similar problems [11, under review]. More memory could simply be added, but there should be an alternative solution.

In [3], we have shown that it is possible to rewrite description logic queries in a way that matches the canonical

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<session name="Session 2015-04-22 11:30:13">
  <starttime>2015-04-22 11:30:13</starttime>
  <entries>64710</entries>
  <size>3935392</size>
...
  <telegram idx="56029" level="3" type="0" messageId
      ="15792351" param="703110">
    <channel>Net13.CDSC_29.SupervisionDrive3</
        channel>
    <timestamp>2015-04-22 11:41:14.359</timestamp>
    <msg>SapSupervisionSTO ACTIVE -&gt; READY: RESET
         drive torque off ( 0:10:47750 ms )</msg>
  </telegram>
  <telegram idx="56031" level="3" type="0" messageId
      ="15790086" param="66">
    <channel>Net13.CDSC_29.SupervisionDrive3</
        channel>
    <timestamp>2015-04-22 11:41:14.359</timestamp>
    <msg>status word changed (0x42)</msg>
  </telegram>
...
</session>
```

Listing 1: Example of an XML export from the machine log.

interpretation as defined in [12]. We extended the DL-Learner Framework[14] to directly query the RDF graph database and perform the supervised machine learning, thus solving the memory issue.

By replacing the reasoner, it becomes necessary to run some of the inferencing steps required for Class Expression Learning ourselves. For instance, the subclass hierarchy should be present and replicated on all instances. It would also be possible to refine the SPARQL translation [3] and push more knowledge from the Ontology into the query. There is already a range of publications on this topic, for example the rewriter proposed in [13] which implements OWL-QL, although it must be cautioned that the complexity of the rewritten queries overwhelms some databases, thus voiding our gain.

On the other hand, more and more commercial databases are starting to put reasoning capabilities into their databases, like Oracle[15] or Stardog.[16]

C. Event linking and example generation

The core structured learning algorithm works on a set of positive and negative input examples which it will then try to describe using OWL Class Expressions. Depending on the data source, such information or classification may or may not be available. In our use case, it was not available. Sometimes it is possible to learn from human engineers about which events have been problematic (knowledge is present but not yet recorded in a machine readable way). At other times, not much information may exist. Then there are also many different approaches that can be pursued. In our project, we decided for co-occurrence analysis and statistical

[11] CostFed is open source https://github.com/AKSW/costfed

[12] https://spark.apache.org/

[13] Pellet 2 is open source https://github.com/stardog-union/pellet

[14] DL-Learner is open source http://dl-learner.org/

[15] https://www.cs.ox.ac.uk/isg/projects/OracleERO/

[16] http://www.stardog.com/

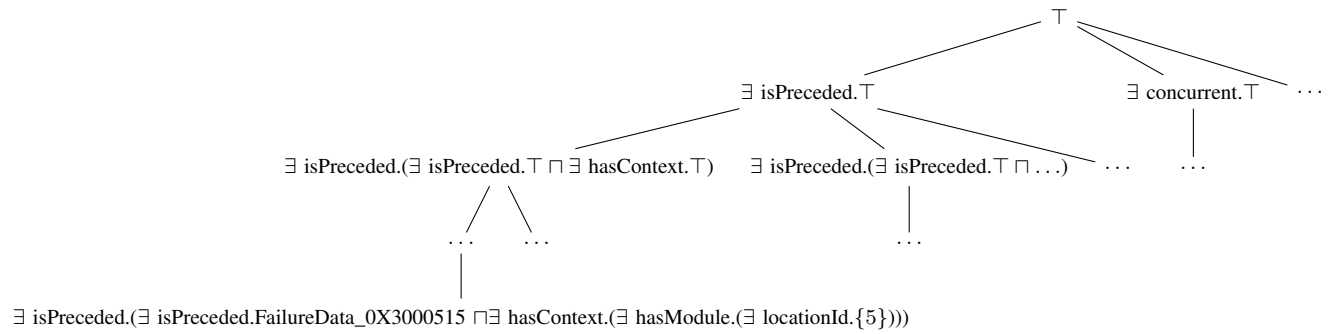

Figure 3: Search tree while describing error causes in machine log files.
The expression on the lower right reads: the second predecessor event group leading to the problem contained events of class 0x3000515 (event class codes are not cross referenced with human readable descriptions at the moment) and occurred in location ID 5 (a specific part of the machine, like the light curtain)

normal model based on the assumption that the majority of the machines do not exhibit any grave problems and that we want to investigate those which depart from the norm. We have described this approach in [14] and [15].

Based on those results, we can identify some positive input examples. The negative examples are generated by uniform sampling on the other events.

It turns out that all our work still does not bear fruit, since there are as yet no relations between the events expressed inside the database. To solve issue 4), we turn to linking. Linking of data in the Semantic Web is an ongoing research area. We use and extend the LIMES[17] framework with temporal linking algorithms such as [16] and topological linking [17]. To deal with the data size, a sliding window approach is used during the link generation. The generated links are then stored in the graph database. By introducing links between concurrent events, predecessor events and successor events, as well as linking together similar modules (but in different machines), we can allow the structured machine learning algorithms to make use of these relations.

D. Structured Machine Learning

To derive descriptions for the positive examples, we can now use algorithms of the DL-Learner framework such as Class Expression Learning for Ontology Engineering [4] or EL downward refinement [18].

A typical search tree during the processing of event log data is seen in Figure 3. The structured machine learning framework uses the concept of refinement operators. It will successively try to match the positive and negative examples against some class expressions, starting from the most general concept "$\top$" which encompasses all instances of the knowledge base. Then it will try to expand on the most promising nodes, that is those yielding better accuracy with regard to the target example sets, as well as any additional weights assigned to the heuristic. One simple possibility is to choose predictive accuracy as the percentage of correctly classified examples: $acc = \frac{\#\text{covered positive example}+\#\text{not covered negative examples}}{\#\text{all examples}}$ at each step.

To be most effective, it is crucial to define any class hierarchies correctly in the ontology. For example, the FailureData codes in the example are all sub-classes of a generic failure data class. In that way, the algorithm can first specialise on failure codes in general and then branch out to a disjunction of multiple specific failure codes.

Once a list of class expressions has been calculated, it can then be cut down at predecessor events and used to visually highlight the results in a graphical explorer. If there are informative links and labels present in the database, the user could also immediately follow them and learn more about the events, components and sensors in the machine (this is not currently implemented and the required data for this has not been collected yet).

IV. Putting it all together

So far we have introduced all the components and explained the requirements of the graph databases, but how does it all fit into pieces? Most of the components can be used individually, but if desired they can be connected using a HTTP REST API. For example they can be registered within the Ontos EIGER[18] platform which can provide a web interface and initiate calls to the API. First of all, each component needs to be configured and deployed on one or several servers. For the Spark-based parts, a Spark Cluster should be running. The graph database should be running and the nodes be registered in the federated engine.

Currently, our solution is based on manual imports. In step 1), the user can import the event logs and submit them for conversion to RDF, as well as choose a database for storage. Furthermore, the linking is configured to run once the import is finished. The linking framework LIMES requires information about the input database and the target

[17]LIMES is open source http://aksw.org/Projects/LIMES

[18]http://ontos.com/products/platform/

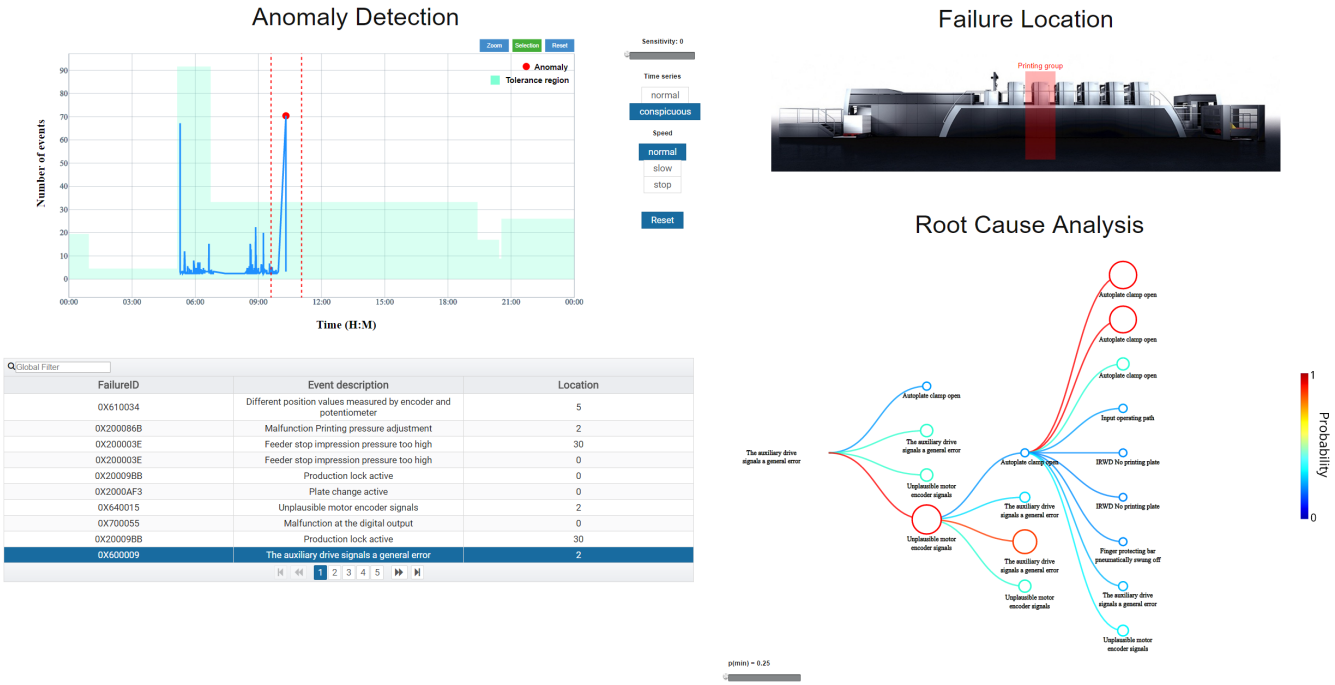

Figure 4: Event Data visualisation. The time plot in the upper left shows the event frequency within a day, and the statistical normal model in the background. The company-internal failure data catalogue has been linked into the lower left and on the right side are the results of the correlation analysis [14] (Screenshot provided by Marco Huber, USU Software AG, Germany)

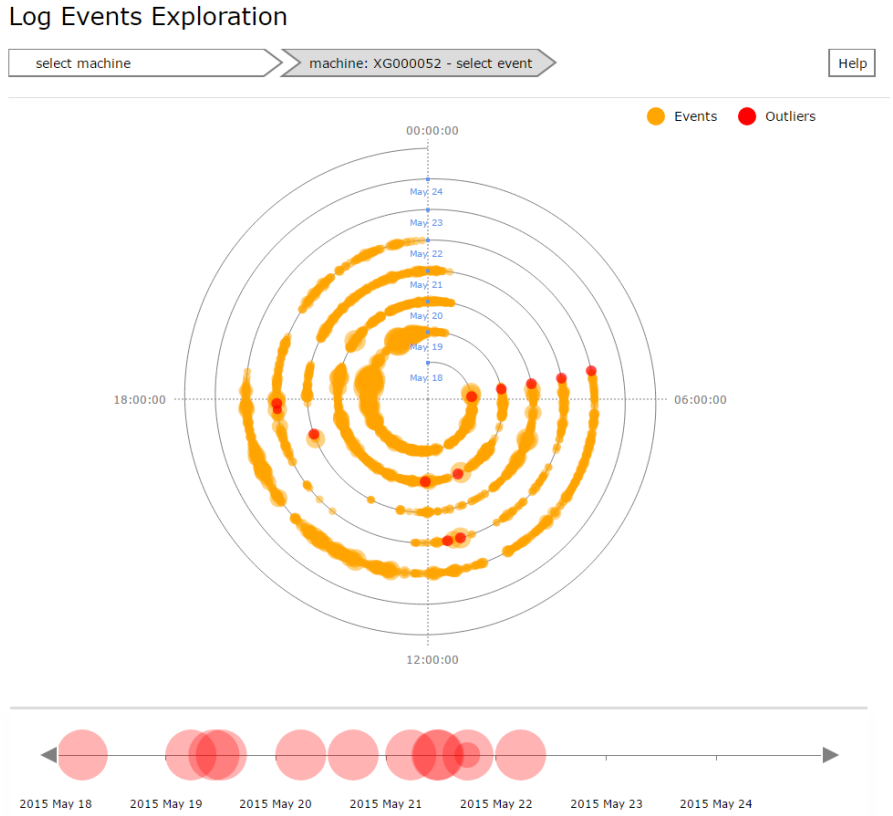

Figure 5: Temporal Event Data visualisation. Events are plotted on a spiral with one ring per day, machine learning results (reboot indicators) are indicated with red dots. (Screenshot provided by Martin Voigt, Ontos GmbH, Germany)

database (or graph within) to store the links to, as well as configuration of the linking rules. For this there can be linking plans created in the GUI. Now the component to generate or retrieve positive examples can be run, after which the class expression learning will take place using all the previous information. The results from that step can then be presented in a visual explorer for further human investigations.

Two interfaces on top of the data are exemplified in Figure 4 and Figure 5. On the top figure, the daily behaviour of a single machine can be analysed, compared to a normal model, as well as use case partner specific information be embedded in the display. After selecting a range of events in the time line, event chains can be uncovered on the lower right by interactively opening and following the nodes.

On the bottom figure, a single machine can be analysed in its weekly performance (weeks can be selected on the bottom part). Error events are shown in red and the machine learning results can be applied using a filter select box (not on screen).

There will also be the possibility to turn the class expressions from the machine learning results into complex event processing rules for future alerts.

V. Conclusion

We have shown that structured machine learning on large scale data is becoming feasible, and provided a description of the required steps to implement it in practice. Much of our work and related work is also being published as open source software for anyone to integrate and experiment with. To facilitate the integration of DL-Learner into the pipeline, a new web service component[19] was created, that can be submitted learning jobs to and query the progress of the learning algorithm.

As can be detailed in the referenced publications, the SPARQL machine learning approach allows DL-Learner to process large data sets that were previously out of reach. Additionally, speed improved by nearly two orders of magnitude when running the learning algorithms using SPARQL when compared to smaller machine learning data sets.

The temporal relations within the event data led to the implementation of new semantic linking algorithms inside LIMES as well as a newly refactored release of the LIMES framework.

Class Expression Learning provides an additional method of generating human understandable classifiers in addition to deep learning methods.

Finally, Ontos GmbH, Germany and USU Software AG, Germany,[20] are integrating results from this research into their commercial offerings and data analysis platforms. An appealing graphical user interface makes it easier for the users to both configure the system and interactively explore the results, leading to improved service time and new complex event processing rules to actively detect common issues in the future.

[19]https://github.com/AKSW/DL-Learner-sake-web

[20]https://www.usu.de/en/

Acknowledgement

This work was supported in part by a research grant from the German Ministry for Finances and Energy under the SAKE project (Grant agreement No. 01MD15006E) and by a research grant from the European Union's Horizon 2020 research and innovation programme under the SLIPO project (Grant agreement No. 731581).

References

[1] M. Saleem, S. S. Padmanabhuni, A.-C. Ngonga Ngomo, J. S. Almeida, S. Decker, and H. F. Deus, "Linked cancer genome atlas database," in *Proceedings of I-Semantics2013*, 2013.

[2] N. Arndt and M. Martin, "Decentralized evolution and consolidation of rdf graphs," in *17th International Conference on Web Engineering (ICWE 2017)*, ser. ICWE 2017, Rome, Italy, Jun. 2017.

[3] S. Bin, L. Bühmann, J. Lehmann, and A.-C. Ngonga Ngomo, "Towards SPARQL-based induction for large-scale RDF data sets," in *ECAI 2016 - Proceedings of the 22nd European Conference on Artificial Intelligence*, ser. Frontiers in Artificial Intelligence and Applications, G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum, and F. van Harmelen, Eds., vol. 285. IOS Press, 2016, pp. 1551–1552.

[4] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp, "Class expression learning for ontology engineering," *Journal of Web Semantics*, vol. 9, pp. 71 – 81, 2011.

[5] M. F. Huber, M. Voigt, and A.-C. N. Ngomo, "Big data architecture for the semantic analysis of complex events in manufacturing," in *GI-Jahrestagung*, 2016, pp. 353–360.

[6] M. Saleem, M. I. Ali, R. Verborgh, and A.-C. Ngonga Ngomo, "Federated query processing over linked data," in *Tutorial at ISWC*, 2015.

[7] J. Lehmann, G. Sejdiu, L. Bühmann, P. Westphal, C. Stadler, I. Ermilov, S. Bin, N. Chakraborty, M. Saleem, A.-C. N. Ngonga, and H. Jabeen, "Distributed semantic analytics using the sansa stack," in *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, 2017.

[8] C. Stadler, J. Unbehauen, P. Westphal, M. A. Sherif, and J. Lehmann, "Simplified RDB2RDF mapping," in *Proceedings of the 8th Workshop on Linked Data on the Web (LDOW2015), Florence, Italy*, 2015.

[9] E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, C. Pinkel, M. G. Skæveland, E. Thorstensen, and J. Mora, "BootOX: Practical mapping of RDBs to OWL 2," in *Proc. of the 14th International Semantic Web Conference (ISWC 2015)*, ser. Lecture Notes in Computer Science, vol. 9367. Springer, 2015, pp. 113–132.

[10] L. Bühmann, J. Lehmann, and P. Westphal, "Dl-learner - a framework for inductive learning on the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 39, pp. 15 – 24, 2016.

[11] P. Westphal, L. Bühmann, S. Bin, H. Jabeen, and J. Lehmann, "SML-bench – a benchmarking framework for structured machine learning," 2017, (under review).

[12] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007.

[13] S. Bischof, M. Krötzsch, A. Polleres, and S. Rudolph, "Schema-agnostic query rewriting in SPARQL 1.1," in *The Semantic Web–ISWC 2014*. Springer, 2014, pp. 584–600.

[14] M.-A. Zöller, M. Baum, and M. F. Huber, "Framework for mining event correlations and time lags in large event sequences," in *15th International Conference on Industrial Informatics, Emden, Germany*. IEEE, 2017.

[15] M. Huber, *at - Automatisierungstechnik*. DeGruyter, 08 2017, vol. 65, ch. Conditional anomaly detection in event streams, p. 233, 4.

[16] K. Georgala, M. A. Sherif, and A.-C. N. Ngomo, "An efficient approach for the generation of allen relations," in *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI) 2016, The Hague, 29. August - 02. September 2016*, 2016.

[17] M. A. Sherif, K. Dreßler, P. Smeros, and A.-C. Ngonga Ngomo, "RADON - Rapid Discovery of Topological Relations," in *Proceedings of The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017.

[18] J. Lehmann and C. Haase, "Ideal downward refinement in the EL description logic," in *Inductive Logic Programming, 19th International Conference, ILP 2009, Leuven, Belgium*, 2009.