DAIKIRI
Erklärbare Diagnostische KI für industrielle Daten

Project Number: 01IS19085B **Start Date of Project:** 01/01/2020 **Duration:** 24 months

Deliverable 4.2
Scalable Library for Structured Machine Learning including Verbalisation

Dissemination Level	Public
Due Date of Deliverable	Month 27, 31/03/2022
Actual Submission Date	Month 27, 31/03/2022
Work Package	WP4 — Explainable Machine Learning
Tasks	T4.5, T4.6
Type	Report
Approval Status	Final
Version	1.0
Number of Pages	15

GEFÖRDERT VOM

Bundesministerium
für Bildung
und Forschung

This project has received funding from the Federal Ministry of Education and Research (BMBF) within the project DAIKIRI under the grant no 01IS19085B.

History

Version	Date	Reason	Revised by
0.1	31/12/2021	First draft created	Simon Bin
1.0	31/03/2022	Final version created	Stefan Heindorf

Author List

Organisation	Name	Contact Information
UPB	Simon Bin	simon.bin@uni-paderborn.de
UPB	Stefan Heindorf	heindorf@uni-paderborn.de

Executive Summary

We developed a rule-based system to verbalize logical axioms. The first insights and examples will be presented in this deliverable. Furthermore, continued development on the library for Structured Machine Learning will be presented and the preliminary results are a big improvement over the previous deliverable. The data transformation of the first use case data is another major contribution in this document; going from single tabular data to full ontology with relational links. Generating artificial learning problems and exporting concepts from the Structured Machine Learning library is also presented. Finally, we come to a short conclusion.

Contents

1 Introduction **4**

2 Data Set **4**

2.1 Preprocessing of Data Sets . . . 6

2.2 Plausibility of Data . . . 6

2.3 Ontology Modelling . . . 7

2.4 Bridging the Language Gap . . . 7

3 Library for Structured Machine Learning **8**

3.1 Generation of Class Expressions . . . 9

3.2 Data Format for Saved Class Expressions . . . 9

3.3 Scalability . . . 10

4 Library for Verbalisation **11**

4.1 Verbalization Rules . . . 11

4.2 Multilingual Verbalisation . . . 12

5 Evaluation **14**

5.1 Examples of Structured Machine Learning results . . . 14

5.2 Examples of Verbalised Class Expressions . . . 14

6 Conclusion **15**

References **15**

1 Introduction

Structured Machine Learning is one of the building blocks in Explainable Artificial Intelligence. In this deliverable, we continue to improve our library for Structured Machine Learning that was created in tasks 4.1 and 4.2.

In Section 2, we investigate how to transform the project's use case data into a knowledge graph suitable for Structured Machine Learning. Another focus point is to lay the ground for scalability research in the library, based on the transformed data. The limitations in the expressivity supported by our library need to be considered and possible treatments need to be implemented. This shall further strengthen the project goal of making Machine Learning results understandable and explainable.

Notable progress in the design and implementation of the library for Structured Machine Learning has been summarised in Section 3.

Structured machine learning can learn logical axioms. While these are easy to read for logicians, they need to be verbalised in order to make them accessible to the domain experts. Hence, in Section 4, we developed a library for the verbalisation of the structured machine learning results.

Section 5 presents the results of applying the algorithm to the use case data as well as verbalization examples.

Finally, we come to a conclusion in Section 6.

2 Data Set

In the context of this project, we aim to explore the suitability of the Structured Machine Learning approach on the "Smart Logistics" use case presented in work package 6. To that end, the use case data first needs to be transformed into graph data suitable for structured machine learning.

Use case data for "Smart Logistics" is provided by the partners as a CSV (comma-separated values, [Sha05]) document, including documentation. The use case data can be understood to consist of two main classes (event types). Ordering of stock, and measurement of change in existing stock. Furthermore, each of these event types is accompanied by various attributes such as:

Attribute	Description
customer_number, customer_id, customer_name	This meta-data identifies the owner of the smart logistics facility.
site_id	A per-customer unique identifier for each facility site.
site_number	Further meta-data describing the site.
assortment_uuid	Each site contains many assortments. This value provides a unique identifier for assortments.
op_group_id, logistic_type, location_uuid, customer_item_number	These attributes contained in the source CSV document are directly mapped into the structured data as values (data properties), without further interpretation.

Continues on next page.

Continuation from previous page.

Attribute	Description
supplier_id, supplier_item_number, order_number, order_pos, order_priority, physical_address, no_of_boxes, reorder_quantity, deliver_mon, deliver_tue, deliver_wed, deliver_thu, deliver_fri, start_week, every_week, week_of_month	These additional attributes concerning the order are directly mapped into the structured data as values (data properties), without further interpretation.
box_number_in_site	Each assortment contains many boxes—this is a unique number for the box.
replenished_at	Time stamp when this stock was replenished or depleted.
old_stock, new_stock	Absolute measure of the amount of stock in this box or site.
type	Types are order events or stock change events.
ordering_at	A time stamp recording the time when the order was placed.
ordered_qty, delivery_qty	The absolute measure of the amount that was ordered and delivered.
requesting_at, shipping_at, confirmed_at, confirming_at	Further time stamps recording the time of stages of the order process.
reorder_point	The absolute measure of the amount of stock at which a new order should be placed.
positive_stock_change	Whether the change in stock was positive or negative compared to the old stock.
relative_class	A categorical value describing percentage of change comparing the old and new stock values.
anomaly1, anomaly2, anomaly3	Labels about whether this particular event was an anomaly of type 1, 2, or 3, or not.

The use case data consists of approximately 3 million entries. Furthermore, it has been labelled with three different types of "anomalies" by the partner. The goal of the Structured Machine Learning process on the use case data shall be to find explanations describing these anomalies.

2.1 Preprocessing of Data Sets

While modelling the data, we aim at the following principles:

- Identify the OWL Classes contained in the data, and map each source data entry to one or more OWL Individuals (class instances).
- Identify the OWL Properties contained in the data, and create a mapping from source data to OWL knowledge graph.
- Identify the relational links between the individuals.

These are the basic steps required to ensure the use case data can be used in structured machine learning, however they may not be sufficient. Due to the limited expressiveness of OWL (see [Bor96]), we aim do bridge this gap through feature engineering. Our computed features are concerned with two broad categories: *Temporal relations*, created by discrete windows of time difference between events, as well as *mathematical relations* between attributes, foremost the difference between the newly measured stock and the previous stock, as well as the difference between an order delivery and the stock change.

All source code for this task can be found at `https://github.com/dice-group/DAIKIRI/tree/master/usecase1-data-conversion`.

2.2 Plausibility of Data

The use case data has been manually inspected on a random sample for plausibility. According to the documentation of the anomalies, in the use case data there should be a relation between the replenishment of stock from incoming orders and the measured amount of stock. However, when a random sample of the data was plotted (see Figure 1), this relation was not apparent. Hence, a *summary stock change* event had to be introduced as follows:

At each discrete time point t, the $total\_stock(s)$ of a Summary Stock Change s shall be defined as

$$total\_stock(s) := \sum_{e \in B_s} new\_stock(e)$$

E is the set of all stock change events in an assortment, and

$$\begin{aligned} E \supseteq B_s := \{e | replenished\_at(e) \leq replenished\_at(s) \\ \wedge \forall e' \neq e : replenished\_at(e') > replenished\_at(s) \vee replenished\_at(e') < replenished\_at(e) \\ \vee box\_number\_in\_site(e') \neq box\_number\_in\_site(e)\} \end{aligned}$$

All other attributes of the summary stock change are defined identically to the individual stock changes. Using these events, we could correct the relation between orders and stock values.

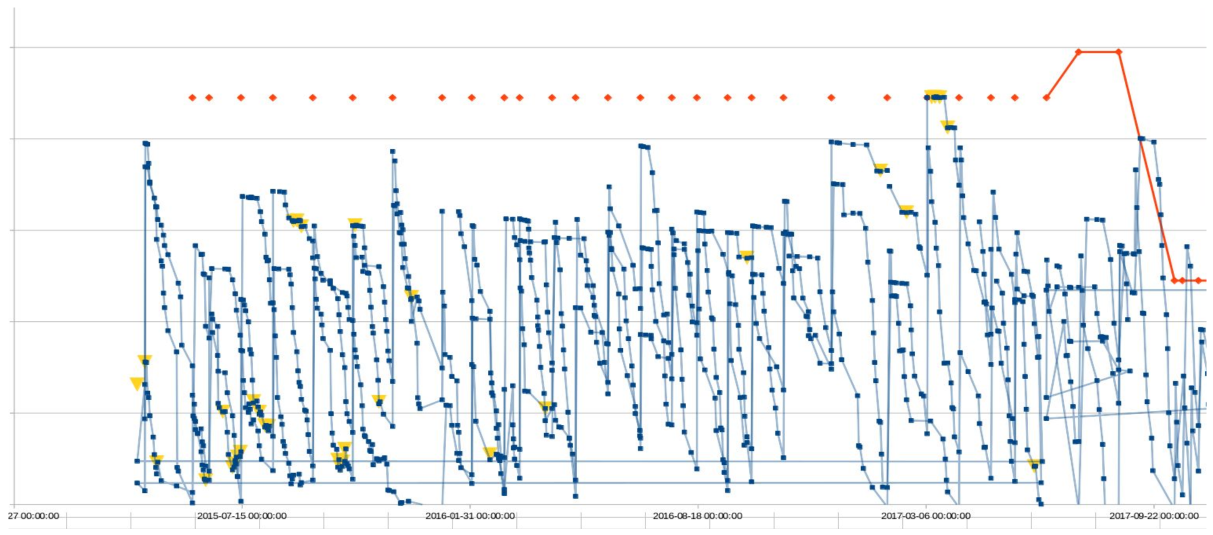

Figure 1: Plot investigating the amount of stock (y axis, blue dots) at a certain time (x axis), compared to the stock replenished from incoming orders (orange dots). Lines connect events within the same "box".

2.3 Ontology Modelling

With all these considered, the first version of the ontology for this use case data could be modelled. A graphical representation is shown in Figure 2. All objectives could be fulfilled:

- Classes have been identified (Customer, Site, Assortment are all hierarchical attributes in the original data).
- The attributes of the original data have been mapped onto ontological properties.
- The events have been linked between each other. The relational links have been added. Each Stock Change happens with regard to a Box, each Box belongs to an Assortment, and so on. The newly introduced Summary Stock Changes have been linked to their contained events (set B). Furthermore, each Stock Change event has been linked to an Order event when such an order could be found within one of the discrete time windows.

2.4 Bridging the Language Gap

Our Structured Machine Learning library was started with support for the description logic $\mathcal{ALC}$ ([BHS08]). Our aim towards the end of the project will be to extend this with support for data properties. To make data properties contained in the use case data accessible to our existing approach, we further mapped discrete attribute ranges to classes. For example, a *delivery_qty* larger than 90 but less than or equal to 100 would be additionally assigned the type *DeliveryQty100*.

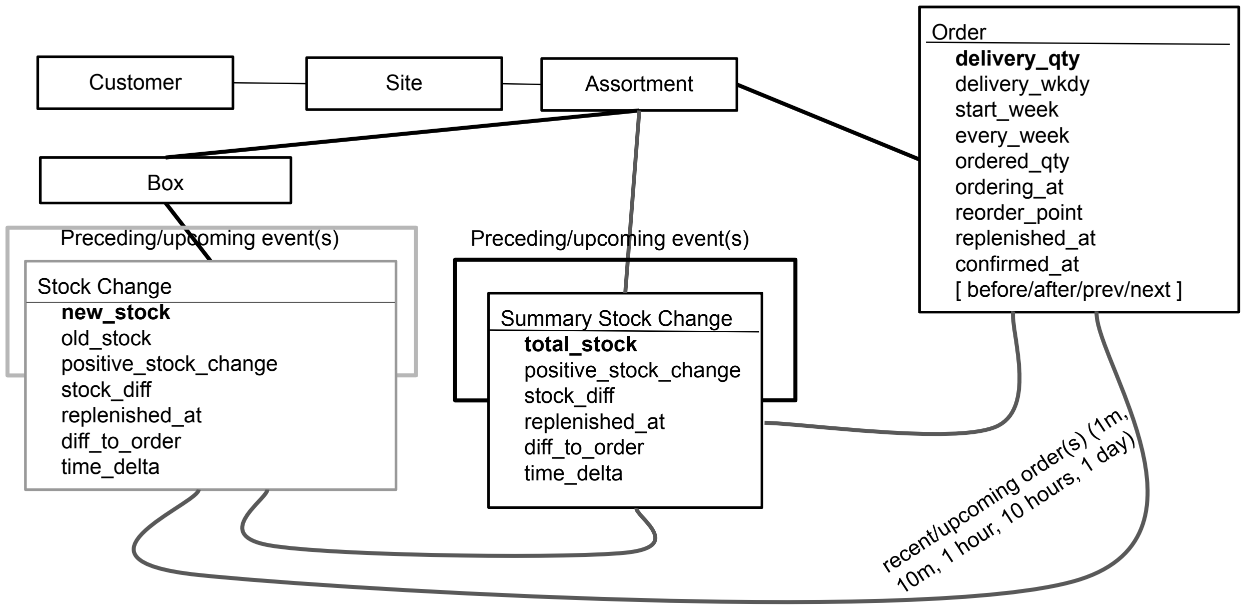

Figure 2: Design of the ontology for “Smart Logistics” use case data

3 Library for Structured Machine Learning

Development of the library for Structured Machine Learning was continued. Support for closed world negation was completed. An exporter for the learning results was implemented. The data format for saved class expression has been specified as RDF/XML[1]. This standardised interchange format will also facilitate the verbalisation of the learned class expressions. Class expression rendering is now also available in Manchester[2] OWL in addition to Description Logics syntax. Simplification algorithms such as the Negation Normal Form ([RV01]) have been added. One major change with regards to the previous version concerns the usage of the *LearningProblem* interface. In order to support learning algorithms that can be trained on multiple problems, the problem is now only used as an argument to the *fit()* method and cannot be configured earlier. Refer to the updated API documentation on our website for more information.

The library is released under an Open Source Licence and can be found on Github at `https://github.com/dice-group/Ontolearn`.

[1] `https://www.w3.org/TR/rdf-syntax-grammar/`
[2] `https://www.w3.org/TR/owl2-manchester-syntax/`

3.1 Generation of Class Expressions

The library for Structured Machine Learning also contains a generator for artificial learning problems. This module can be used to find class expressions with ideal solutions. At the same time, it can also be used as an input to test the verbalisation of class expressions.

To use it, follow the Python code below:

```python
import os

from ontolearn.knowledge_base import KnowledgeBase
from ontolearn.learning_problem_generator import LearningProblemGenerator
from ontolearn.utils import setup_logging

setup_logging("logging_test.conf")

path = 'KGs/Biopax/biopax.owl'

kb = KnowledgeBase(path=path)
lp = LearningProblemGenerator(knowledge_base=kb)
num_inds = kb.individuals_count()
concepts = list(lp.get_concepts(num_problems=5000,
                                num_diff_runs=10,
                                min_num_instances=int(2),
                                max_num_instances=int(num_inds * .95),
                                min_length=4, max_length=40))

lp.export_concepts(concepts, path='example_concepts')
```

The Learning Problem Generator can be customised with several parameters as seen in the listing. The `num_problems` specifies the desired number of class expressions in the result. The `min-` and `max_num_instances` filter out class expressions that match at least/most the required number of instances, and the `min-` and `max_length` select the length of the class expression.

Finally, the *export_ concepts()* method can be used to store the generated class expressions to an RDF/XML document.

3.2 Data Format for Saved Class Expressions

In the listing below you can find an excerpt of an RDF/XML document as exported from the Learning Problem Generator. It is very similar to the document that can be exported from the result of running a learning algorithm. Each suggested problem is encoded in a Class with the sequential name *Pred_ #*. As additional information, the number of individuals covered by the problem is exported as an annotation property *covered_ inds*.

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

(continues on next page)

(continued from previous page)

```
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xml:base="https://dice-research.org/predictions/1629502825.2758381"
         xmlns="https://dice-research.org/predictions/1629502825.2758381#">

<owl:Ontology rdf:about="https://dice-research.org/predictions/1629502825.2758381">
  <owl:imports rdf:resource="file://KGs/Biopax/biopax.owl"/>
</owl:Ontology>

<owl:AnnotationProperty rdf:about="#covered_inds"/>

<owl:Class rdf:about="#Pred_0">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about="http://www.biopax.org/examples/glycolysis
↪#interaction"/>
        <owl:Class>
          <owl:complementOf rdf:resource="http://www.biopax.org/examples/glycolysis
↪#sequenceParticipant"/>
        </owl:Class>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
  <covered_inds rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">47</covered_
↪inds>
</owl:Class>
```

3.3 Scalability

We took several attempts at improving the scalability of the Structured Machine Learning library. We started with the use case data set as described in section 2. The converted data amounts to 50 GiB file size which contains 238 million triples.

At first, it was not possible to process this data set using our initial library implementation, even with 250 GiB of system memory. Later on, we could significantly improve this situation.

For the initial version, we profiled the Python data types and came to realise that loading the full use case data set in our structures would require approximately 50 TiB of system memory.

Consequently, we had to remove the data loading and resorted to data set scanning instead for the second iteration. This enabled us to load and process the use case data, with a speed of 13.53 expression tests per hour.

The performance was still not satisfactory, so in our third iteration we encoded all the triples as their C memory addresses and used Python's FrozenSet data structure to store the object value mappings. This yielded another 42 × performance improvement (see Figure 3).

We have started to work on SPARQL and Tentris [BCB+20] based data access. Initial evaluation of SPARQL-based access [BBLN16] suggests at least a potential doubling of performance. Furthermore,

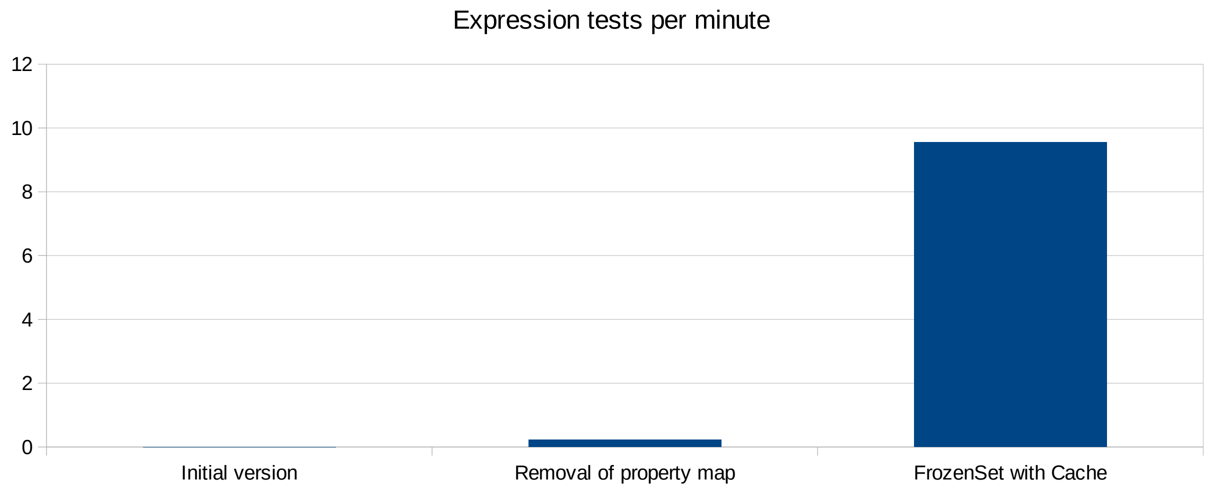

Figure 3: Performance of the different library versions (expression tests per minute)

we want to combine these approaches with parallel search strategies which might result in further scalability enhancements.

4 Library for Verbalisation

This section follows the $\mathcal{SROIQ}^{(\mathcal{D})}$Concepts, Tboxes, and Aboxes definition in [HKS06].

The OWL 2 Web Ontology Language[3] provides the expressiveness of $\mathcal{SROIQ}^{(\mathcal{D})}$[HKS06, HPS04]. A $\mathcal{SROIQ}^{(\mathcal{D})}$ Tbox element t is a general concept inclusion axiom of two concepts C and D. The Library for Verbalization verbalizes Tbox elements $t \in \mathcal{T}$. It builds on SimpleNLG[GR09] version 4.5.0. The library uses given labels in the ontology for the verbalization and in case no label is given for an element, it uses the given class or property name for the verbalization. Anonymous elements, which have no labels (e.g., an inverse property), are not supported and are therefore not verbalized in its current version.

4.1 Verbalization Rules

The corresponding expressions and axioms of t are listed in Table 2 along with their common name and OWL 2 DL syntax[4]. Example verbalizations of these expressions and axioms are listed in Table 3. Additionally, some other OWL 2 classes and axiom are supported by semantics-preserving rewriting to supported classes and axioms of the library (cf. Table 2). Those additional supported elements are partially listed in Table 4.

For instance, an inclusion `SubClassOf(C D)` states that the class expression `C` is a subclass of the class expression `D`. The Verbalization takes the class expressions that represent the subclass `C` and the superclass `D` of an inclusion. Then, for both expressions, it creates a verbalization C_v, D_v. It creates a clause with these verbalizations, using C_v as noun phrase for the subject, "be" as verb phrase and D_v as complement for the object of the clause.

[3] `https://www.w3.org/TR/owl2-overview/`
[4] `https://www.w3.org/TR/owl2-syntax/`

common name	$\mathcal{SROIQ}^{(\mathcal{D})}$	OWL 2 DL
concept name	A	`a:A`
nominal/single individual	$\{i\}$	`IRI, rdfs:label`
top	$\top$	`Thing`
bottom	$\bot$	`Nothing`
conjunction	$C \sqcap D$	`ObjectIntersectionOf(C D)`
disjunction	$C \sqcup D$	`ObjectUnionOf(C D)`
negation	$\neg C$	`ObjectComplementOf(C)`
universal role restriction	$\forall R.C$	`onProperty(R), ObjectAllValuesFrom(C)`
existential role restriction	$\exists R.C$	`onProperty(R), ObjectSomeValuesFrom(C)`
self restriction	$\exists S.Self$	`ObjectHasSelf(S)`
atleast restriction	$(\geqslant nS.C)$	`ObjectMinCardinality(n S C)`
atmost restriction	$(\leqslant nS.C)$	`ObjectMaxCardinality(n S C)`
inclusion	$C \sqsubseteq D$	`SubClassOf(C D)`

Table 2: List of supported elements.

4.2 Multilingual Verbalisation

The Verbalization library translates the English verbalizations with [TTL+20] to German in the current version. The translated examples in Table 3 are

- Jedes Tier ist etwas, das mindestens einen Lebensraum hat.
- Jede Person ohne Kinder ist eine Person, die kein Kind hat.
- Jeder Autobesitzer ist etwas, dessen ein Unternehmen, eine Regierung oder eine Person, die ein Auto besitzt.
- Jedes ruhige Ziel ist ein Reiseziel, das kein Familienziel ist.
- Jeder Mensch ist ein Tier, das sich kennt.
- Jeder binäre Baum ist ein Baum, der als Zweig nur einen binären Baum hat und der höchstens zwei Zweige hat.
- Jeder binäre Baum ist ein Baum.

common name	$\mathcal{SROIQ}^{(\mathcal{D})}$	Verbalizations
top, atleast restriction	Animal $\sqsubseteq (\geqslant 1\text{hasHabitat}.\top)$	Every animal is something that has at least one habitat.
bottom, universal role	PersonWithoutChildren $\equiv$ Person $\sqcap \forall\text{hasChild}.\bot$	Every person without children is a person that has no child.
disjunction, existential role restriction	AutomobileOwner $\sqsubseteq$ Company $\sqcup$ Government $\sqcup$ Person $\sqcap \exists$ own.Automobile	Every automobile owner is something whose a company, a government or a person and that owns an automobile.
negation, conjunction	QuietDestination $\equiv$ Destination $\sqcap \neg$FamilyDestination	Every quiet destination is a destination that is not a family destination.
self restriction	Person $\equiv$ Animal $\sqcap \exists\text{know}.Self$	Every person is an animal that knows oneself.
universal role, atmost restriction top	BinaryTree $\equiv$ Tree $\sqcap \forall$hasBranch.BinaryTree $\sqcap$ $(\leqslant 2\text{hasBranch}.\top)$	Every binary tree is a tree that has as branch only a binary tree and that has at most two branches.
inclusion	BinaryTree $\sqsubseteq$ Tree	Every binary tree is a tree.

Table 3: Verbalized examples.

OWL classes and axioms	Equivalent OWL classes and axioms
`EquivalentClasses(C1 C2 ...)`	`SubClassOf(C1 C2)`, `SubClassOf(C2 C1)`, ...
`DisjointClasses(C1 C2 ...)`	`SubClassOf(C1 ObjectComplementOf(C2))`, ...
`DisjointUnion(C C1 C2 ...)`	`EquivalentClasses(C ObjectUnionOf(C1 C2 ...))`, `DisjointClasses(C1 C2 ...)`
`ObjectOneOf(a1 a2 ...)`	`ObjectUnionOf(ObjectOneOf(a1) ObjectOneOf(a2) ...)`

Table 4: Semantics-preserving rewriting of some OWL constructs that are additionally supported.

5 Evaluation

5.1 Examples of Structured Machine Learning results

As a first proof of concept on the feasibility of using the Structured Machine Learning library on the use case data described and transformed in Section 2, we randomly picked 12 assortments from the use case data (approx. 0.3% of the full data set) and tasked the library to find explanations for *anomaly2*. The following results were obtained:

- When allowing the algorithm to look into the future:

  $$PositiveStockChange \sqcap (\forall timewindow.(\neg Order)),\ \text{Accuracy } 98.761\%$$

- With only the past available, and inverse relations:

  $$PositiveStockChange \sqcap \big(\exists contains\_event^{-}.\ (\forall order\_timewindow.\ (\exists previous.RelativeChangeGE15L25))\big),$$

  F1-measure 98.454%

In the further progress of the project, we aim to extend the processing of the data and the evaluation of the model to the full use case data set.

5.2 Examples of Verbalised Class Expressions

Two verbalized examples of class expressions on the use case data and the corresponding German translations are shown below.

$$\text{Pred\_1} \equiv (\exists \text{contains\_event.NegativeStockChange}) \sqcap (\forall \text{order\_timewindow\_1d.NegativeStockChange})$$

English: Every pred 1 is something that contains as individual event a stock change negative and whose order within 1 Day is a stock change negative.

German: Jedes Pred 1 ist etwas, das als individuelles Ereignis eine Aktienänderung negativ enthält und dessen Bestellung innerhalb von 1 Tag eine Aktienänderung negativ ist.

$$\text{Pred\_2} \equiv \text{RelativeChangeGE8L10} \sqcup (\exists \text{contains\_event}.(\exists \text{timewindow\_1w.NegativeStockChange}))$$

English: Every pred 2 is a relative change of >=8 <10% or that something that contains as individual event something whose event within 1 Week is a stock change negative.

German: Jedes Pred 2 ist eine relative Änderung von >=8 <10% oder etwas, das als individuelles Ereignis etwas enthält, dessen Ereignis innerhalb einer Woche eine Aktienänderung negativ ist.

In the further progress of the project, we aim to extend the processing of the data and the evaluation of the verbalization to the full use case data set.

6 Conclusion

In this deliverable, we have presented our progress on data set transformation, Structured Machine Learning Library and verbalisation of logical expressions. The results of the library applied to the use case data have been heavily improved. As the next steps, we will continue working on improving the scalability of the Structured Machine Learning library, as well as adding support for data types to the library. For the Verbalisation library, we plan to implement data property verbalizations and to improve the fluency of the verbalizations.

References

[BBLN16] Simon Bin, Lorenz Bühmann, Jens Lehmann, and Axel-Cyrille Ngonga Ngomo. Towards SPARQL-based induction for large-scale RDF data sets. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - Proceedings of the 22nd European Conference on Artificial Intelligence*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1551–1552. IOS Press, 2016.

[BCB$^+$20] Alexander Bigerl, Felix Conrads, Charlotte Behning, Mohamed Ahmed Sherif, Muhammad Saleem, and Axel-Cyrille Ngonga Ngomo. Tentris – A Tensor-Based Triple Store. In *The Semantic Web – ISWC 2020*, pages 56–73. Springer International Publishing, 2020.

[BHS08] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, chapter 3, pages 135–180. Elsevier, 2008.

[Bor96] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artif. Intell.*, 82:353–367, 1996.

[GR09] Albert Gatt and Ehud Reiter. SimpleNLG: A realisation engine for practical applications. In *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG 2009)*, pages 90–93, Athens, Greece, March 2009. Association for Computational Linguistics.

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 57–67. AAAI Press, 2006.

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. Reducing owl entailment to description logic satisfiability. *J. Web Semant.*, 1(4):345–357, 2004.

[RV01] Alan JA Robinson and Andrei Voronkov. *Handbook of automated reasoning*, volume 1. Elsevier, 2001.

[Sha05] Yakov Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180, October 2005.

[TTL$^+$20] Yuqing Tang, Chau Tran, Xian Li, Peng-Jen Chen, Naman Goyal, Vishrav Chaudhary, Jiatao Gu, and Angela Fan. Multilingual translation with extensible multilingual pretraining and finetuning. 2020.