Semantic Web 0 (2022) 1–0
IOS Press

INK: Knowledge graph representation for efficient and performant rule mining

Bram Steenwinckel [a,*], Filip De Turck [a] and Femke Ongenae [a]
[a] *Internet and Data Lab, Ghent University, Technologiepark-zwijnaarde 126, Gent, Belgium*
E-mail: bram.steenwinckel@ugent.be

Abstract. Semantic rule mining can be used for both deriving task-agnostic or task-specific information within a Knowledge Graph (KG). Underlying logical inferences to summarize the KG or fully interpretable binary classifiers predicting future events are common results of such a rule mining process. The current methods to perform task-agnostic or task-specific semantic rule mining operate, however, a completely different KG representation, making them not suitable to perform both tasks or incorporate each other's optimizations. This also results in the need to master multiple techniques for both exploring and mining rules within KGs, as well losing time and resources when converting one KG format into another. In this paper, we present INK, a KG representation based on neighbourhood nodes of interest for improved decision support. By selecting one or two sets of nodes of interest, the designed rule miner on top of this INK representation will either mine task-agnostic or task-specific rules. In both subfields, the INK miner outperforms the currently state-of-the-art semantic rule miners on 14 different benchmark datasets within multiple domains.

Keywords: Knowledge representation, Semantic rule mining

1. Introduction

Knowledge graphs (KGs) are increasingly used as data structures to combine domain expertise with raw data values [1]. In general, a KG is a multi-relational directed graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E})$, where $\mathbb{V}$ are the vertices or entities in our graph and $\mathbb{E}$ the edges or predicates. The example KG represented in Figure 1 shows eight interlinked nodes describing four members of the band Coldplay. Three of these members have a common subgraph as they all studied and were born in England. One member was born in Scotland which is, at time of writing, still a part of the United Kingdom (UK).

Numerous applications are built upon these KGs, covering various domains such as industry 4.0, pervasive health and smart cities [3–5]. These applications interact with the KGs directly or transform the graph into a vector representation to perform Machine Learning (ML) related tasks [6]. Rule mining is also such a KG application, where the goal is to find logical rules in a given KG. For example, a rule mining task for the given example KG in Figure 1 could find the logical rule: *If X has Alma mater Y and Y is Located In Z, Then X is born in Z.* Such logical rules will come with a certain confidence score, defining the general applicability of the rule. The more reliable rules can then be used to complete the KG, perform downstream applications such as fact prediction, fact checking or anomaly and error detection.

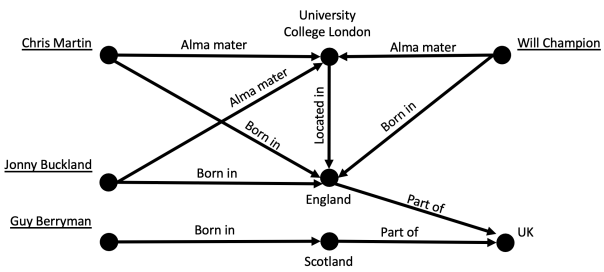

Fig. 1. Simple example of a KG, extracted from DBpedia [2]. Eight nodes are defined, linked to each other by four unique labelled edges.

*Corresponding author. E-mail: bram.steenwinckel@ugent.be.

Rule mining is part of data mining in general, where two broad subfields exist. On the one hand, there is task-agnostic or descriptive mining, where one wants to mine some general information about the KG or some general facts, which hold beyond this provided KG and are generally applicable.

This subfield was originally created to discover hidden knowledge from transactional data, such as relational databases. Association Rule Mining (ARM) is the best-known descriptive technique. A transaction in ARM is an observation of the co-occurrence of a set of items. ARM can be applied to semantic data or KGs by converting the internal representation to a set of transactions. ARM thus identifies the transactions that identify co-occurrences of items that appear frequently in the KG by calculating the associated metrics to quantify this, such as the confidence and the support of the transactions. The logical rule defined, *If X has Alma mater Y and Y is Located In Z, Then X is born in Z*, is such a possible hidden rule that could be mined with an ARM application. ARM for KGs is used in data integration and KG completion tasks [7, 8].

On the other hand, prescriptive mining is more task-specific and performs inferences on the current data, to make predictions in the future [9]. Inductive Logic Programming (ILP) is the best-known paradigm in this subfield. The ILP techniques deduce logical rules from a positive set of nodes and require some (generated) negative set of counter-examples. An example ILP task could be to find one general rule to describe all four members of the Coldplay band in Figure 1. The positive set of nodes selected for this mining task are underlined in Figure 1. One possible rule could state the *born In ?x and ?x Part of UK* relationships hold for all members.

Both subfields are complementary. ILP performs the task-specific cases, as specific facts are needed for those cases. Therefore, ILP directly captures the available related information in the KG to generate the rules. The ILP program will immediately use the available predicate-object information to discriminate between the provided positive and negative set. ILP can, however, be relatively slow and can therefore not handle the huge amount of data that KGs provide today. ARM can handle large KGs and generate rules for fully task-agnostic problems. It is fast and scales to large graphs. This often results in the fact that ARM generates a lot of nonsense or too generally applicable rules as it considers all triples or facts. The generated ARM rule for our example KG is such a rule with limited effect, because people do not always study where they were born.

There does not exist a technique which can perform both prescriptive (task-specific) and descriptive (task-agnostic) rule mining for KGs. The main reason, to our knowledge, is that the current techniques available for both tasks require a different internal representation of the KG. These transformations are performed in relation to the subfield they are operating on. ARM mainly requires the KG to represent as transactions, which reduces the linked aspects of the existing KG. ILP directly works on the graph representation itself, leading to the earlier discussed performance issues.

In this work, we propose a technique to perform rule mining on KGs by Instance Neighbouring using Knowledge (INK). INK is a new way to represent a KG by analysing the neighbourhoods of selected nodes of interest. In the case of mining rules over the whole KG, the neighbourhood of all nodes in the KG are used inside the representation. When a more specific task is given, neighbourhoods of the nodes which have to be considered are only taken into account. In this perspective the INK representation combines the best of both worlds, as it can be used both to mine specific rules in a descriptive rule mining task and reduce the time needed to perform prescriptive mining tasks. In both cases, INK is able to capture the complexity of the KG in an efficient manner.

The remainder of this paper is structured as follows. Section 2 gives an overview of the current available semantic rule mining techniques and discusses their internal representation of the KG. Section 3 details the INK KG representation. Section 4 shows how INK can be incorporated into a rule mining system. Both the implementation of INK and the accompanying rule miner are discussed in Section 5. In Section 6, we evaluate INK for both the task-agnostic and task-specific rule mining, and compare the results with the current state-of-the-art. Section 7 discusses the advantages and drawbacks of INK in the perspective of task-agnostic and task-specific mining. At last, the conclusion of this work is provided in Section 8.

2. Related work & background

Rule mining has a long history, but the existing techniques can be either based on ARM or ILP. ARM searches for implication (if ... then ...) rules, such as *"If a person X has an Alma mater Y, and Y is located in Z, then X is born in Z"*. ILP techniques deduce logi-

cal rules from ground facts and require negative statements as counter-examples. Both techniques were already applied in the context of KGs. This section discusses the two state-of-the-art techniques for KGs, together with some recent advances, namely the ARM technique AMIE and the ILP technique DL-Learner.

2.1. AMIE

AMIE mines task-agnostic rules in the form of horn clauses [10]. Horn clauses are denoted as horn rules when they contain an implication. They usually consist of a head and a body, where the head is a single atom:

$$B_1 \wedge B_2 \wedge ... \wedge B_n \Rightarrow r(x,y)$$

with head $r(x,y)$ and body $B_1 \wedge B_2 \wedge ... \wedge B_n$. The latter is frequently represented as $\vec{B}$. The rules state that if all instantiated body atoms appear in the KG, the head atom can be derived. AMIE reduces the search space by searching for connected and closed rules that are not reflexive:

- A rule is connected if every atom is connected transitively to every other atom of the rule.
- A rule is closed if all its variables are closed. A variable is closed when it appears at least twice in the rule.
- A rule is reflexive if it contains atoms of the form r(x,x)

The example rule *"If a person X has an Alma mater Y, and Y is located in Z, then X is born in Z"* is an example of a connected and closed, not reflexive rule.

2.1.1. Pruning of AMIE candidate rules

Additionally, support and head coverage measures are defined to further prune candidate rules. The support of a rule quantifies the number of correct predictions in the existing KG. AMIE defines the support of a rule as the number of distinct pairs of subjects and objects in the head of all the instantiations that appear in the KG. More in general, the support of a rule R in a KG $\mathcal{G}$ is the number of true derivations r(x, y) (with r(x, y) a the head atom as explained above) that the rule makes in the KG:

$$\text{support}(R) = |\{r(x,y) : (\mathcal{G} \wedge R \models r(x,y)) \wedge r(x,y) \in \mathcal{G}\}|$$

By providing a threshold on this support value, rules and facts which are less common can be pruned.

Support is an absolute number, such that one has to know the absolute size of the KG to give a meaningful value for the threshold. To avoid this, AMIE proposed the head coverage measure to quantify the ratio of the known true facts that are implied by the rule. It is the ratio of instantiations of the head atom that are derived by the rule:

$$hc(\vec{B} \Rightarrow r(x,y)) = \frac{\text{support}(\vec{B} \Rightarrow r(x,y))}{|\{(x',y') : r(x',y') \in \mathcal{G}\}|}$$

Based on the example KG of Figure 1, AMIE mined the following rule:

$$alma\_mater(?x,?y) \wedge located\_in(?y,?z) \implies born\_in(?x,?z)$$

This rule has a support of 3 because three band members fulfil this rule. The head coverage of this rule is 3/4 as one member has the *born_in* relation, but without a link to the *alma_mater* relation.

Both the support and head coverage only consider the correct predictions of the generated rule. Confidence is a measure that also takes false predictions into account. The standard confidence of a rule is the ratio of all its predictions that are in the KG. All facts that are not in the KG are seen as negative evidence.

$$conf(\vec{B} \Rightarrow r(x,y)) = \frac{\text{support}(\vec{B} \Rightarrow r(x,y))}{|(x,y) : \vec{B}|}$$

However, it must be treated with care within the context of KGs. ARM operates under the closed world assumption: a statement that is true is also known to be true and conversely, what is not currently known to be true, is false and introduces negative evidence for those cases which are not available in the dataset. KGs can, however, follow the open world principle: the truth value of a statement may be true irrespective of whether or not it is known to be true [11]. As ARM does not take the difference between "*unknown*" and "$false$" into account, this can lead to surprising results when mining rules inside a KG.

AMIE resolves this problem by generating negative examples for a rule by means of the Partial Completeness Assumption (PCA). Here, they assume that if one y for a given x and r in r(x,y) head is known, then all the y's are known for that x and r. Under the PCA, the denominator of the confidence formula is not the size

of the entire set of conclusions derived from the body of the rule, but the number of facts that we know to be true together with the facts that we assume to be false.

$$conf_{pca}(\vec{B} \Rightarrow r(x,y)) = \frac{\text{support}(\vec{B} \Rightarrow r(x,y))}{|(x,y) : \vec{B} \wedge r(x,y')|}$$

This PCA has been applied in the Google Knowledge Vault and is more commonly known as the "local completeness assumption" [12].

In our example, the confidence of our rule is 1, as no negative facts can be found that contradict the rule. If we would add an additional relationship in our example KG, which states that Guy also studied at the University College London, one negative fact would be available and the confidence score would drop to 3/4.

2.1.2. Implementation AMIE Rule Mining

Mining rules with AMIE is an iterative process. The algorithm maintains a queue of possible rules, initially filled with all head atoms of length 1. It then iteratively dequeues a possible rule. If it meets certain criteria, i.e. closed and of high quality regarding the used pruning technique, it is pushed to the output. If the rule does not exceed the maximum number of atoms maxLen, it goes through a refinement process which expands the rule (the parent) to produce a set of new rules (the children). This refinement procedure is an efficient way to explore the search space as enumerating all possible combinations of conjunctions of atoms is infeasible for large KGs. Hence, they explore the search space by iteratively extending rules using one of the following refinement operators:

- Add Dangling Atom: Add an atom that uses a fresh variable for one of its two arguments. The other argument is a variable that is shared with the rule.
- Add Instantiated Atom: adds a new atom to a rule that uses an entity for one argument and shares the other argument (variable) with the rule.
- Add Closing Atom: adds a new atom to a rule so that both of its arguments are shared with the rule.

The rules who are not already used before or are not pruned to single atoms after this refinement process are pushed into the queue and can be extended in a next iteration.This process is repeated until the queue is empty.

2.2. Optimizations of AMIE approach

To efficiently calculate the pruning metrics, such as support and head coverage, AMIE first used multiple count projection queries inside the mining operators, which translate into very inefficient queries in both SPARQL and SQL. To resolve these inefficiencies, an in-memory database was explicitly designed geared towards this type of queries. This database stores for the S, P and O index of the triple (S,P,O), a nested hash table providing the information of the two other triple parts. More specifically, for each subject, two hash tables are created: (a) one with as keys the predicates and as values the objects, and (b) one with as keys the objects and as values the predicates. This process is repeated for all six combinations. Searching the relevant values in these hash maps can then be performed in constant time.

Several improvements were made to increase the efficiency of AMIE even further. A new version of AMIE, called AMIE+ [13], speeds up 2 different parts of the main rule mining algorithm: the refinement phase and the confidence evaluation. Recently, the AMIE framework was even further upgraded, becoming AMIE3 [14], and incorporates new pruning strategies, parallelization, and a lazy computation of confidence scores to allow the system to scale effortlessly to large KGs. When we refer to AMIE in the next sections of this paper, we refer to the most recent version AMIE3, unless indicated differently.

2.3. DL-learner

DL-Learner is defined in an open software framework, employing methods that are able to use the complex structure of available background knowledge when learning hypotheses [15]. The rule mining capabilities of DL-Learner can solve three types of learning problems:

- **Standard Supervised Learning:** Given a set of positive and negative labels, DL-Learner tries to find an OWL class expression such that all, or as many as possible, positive examples are instances of C and none, or as few as possible, negative examples are instances of C.
- **Positive Only Learning:** Similar as the above, but searching for a class expression that covers the positive examples without providing negative samples. The goal is to still generalise sufficiently.

– **Class Learning:** Given an already existing class A within the ontology, DL-Learner finds an expression to describe A, using the instances of the class as positive examples. Both the existing knowledge about A in the ontology and the positive examples are used to describe A.

The algorithms to solve these learning problems are derived from ILP and genetic programming, extended with theoretical and algorithmic foundations going from learning complex definitions in ontologies to generic schema enrichment and fuzzy description logics.

Almost all of these algorithms are also, similar to AMIE, based on the principle of refinement operators. Within DL-Learner, learning can be seen as the search for a correct concept definition in an ordered space. In such a setting, one can define suitable operators to traverse the search space. The goal is to use those operators that have many useful properties like finiteness, non-redundancy, properness and completeness, while still allowing to efficiently traverse through the search space in pursuit of good hypotheses.

2.3.1. Implementation DL-Learner

The OWL Class Expression Learner (OCEL) is such a common algorithm, which uses a proper and complete refinement operator to build a search tree, while using heuristics that control how the search tree is traversed [16]. In this tree, the nodes are annotated with a score and the number of times they have been expanded. Based on these two values, OCEL defines weak nodes, i.e. when the number of uncovered positive examples is above a given threshold. They are never visited, which allows the algorithm to ignore those parts of the search space resulting in improved efficiency. All the other algorithms made available by DL-Learner follow the same procedure to traverse the search space, but are optimised for a certain problem or use Description Logics to infer more knowledge within the KG. The Class Expression Learner for Ontology Engineering (CELOE) is built on the OCEL algorithm, but uses a different heuristic and introduces some bias to shorter class expressions. All algorithms return the expressions with the highest scores and sort them based on the number of expansions until a certain time threshold parameter is exceeded [16].

If we for example want to learn the class expressions that describe the four band members in our KG (underlined nodes), we can instruct DL-Learner's CELOE algorithm to mine rules for only these specific nodes. Fives rules are mined and ordered based on their accuracy scores:

1. born_in some Thing 100.00%
2. (alma_mater some (university_college_london)) or (born_in some (Scotland)) 100.00%
3. (alma_mater some (university_college_london)) or (born_in some Thing) 100.00%
4. (alma_mater some Thing) or (born_in some (Scotland)) 100.00%
5. (alma_mater some Thing) or (born_in some Thing) 100.00%

2.3.2. Optimizations of DL-Learner approach

As the DL-Learner algorithms are developed to, preferably, only output the optimal solutions, some optimizations such as divide and conquer search techniques are provided to reduce the number of suboptimal solutions within a certain time interval. In all cases, the goal is to restrict the set of nodes for expanding a fixed size of candidates within a set.

To operate efficiently, DL-Learner requires to store the inferred knowledge within memory as the reasoning procedures afterwards check the provided instances based on this inferred knowledge. This storage in memory is needed to speedup the whole process, but restricts DL-Learner from being run on large and complex KGs. Statistical sampling approaches were used to resolve these memory issues and to test hypotheses in the presence of a large number of nodes within the KG [17].

2.4. Other semantic Rule miners

The recent advances in the area of embeddings and KG vector representations resulted in some additional semantic rule mining methods. The main goal of such miners is to deal with the possible incompleteness or large scale of the KGs, which reduces the need for partial completeness calculations as discussed in Section 2.1. One such miner is RuLes [18]. It iteratively constructs rules over a KG and collects feedback for assessing the quality of (partially constructed) rule candidates through specific queries issued to a precomputed embedding model. Within the Rules framework, the confidence measures capture the rule quality better than other techniques because they now reflect the patterns in the missing facts. The improved confidence measures, therefore, improve the ranking of rules. An embedded version of the KG is used here to define the quality of the rule and is not used to mine the task-agnostic rules themselves.

Another such technique is RLvLR (Rule Learning via Learning Representations) miner, an embedding-based approach to rule learning focusing on descriptive rule mining [19]. This miner specifies a target predicate in a KG to mine quality rules whose head has that predicate. The combination of the technique of embedding in representation learning together with a new sampling method results in more quality rules than major systems for rule learning in KGs such as AMIE+. The main focus of the RLvLR miner is defined in the scope of only mining specific rules for a given predicate. The RLvLR miner is, however, not made publicly available, except for an compiled executable to reproduce the fixed experimental setup.

3. INK representation

While AMIE and DL-learner use refinement operators to traverse the search space, INK builds its internal representation by transforming the neighbourhood of the nodes of interest into a binary matrix representation. With this binary matrix representation, column operations based and comparisons of columns for a large number of nodes of interest can be easily performed to reveal new patterns or rules. To explain how this binary representation is built, we use the example KG visualised in Figure 1 throughout this section.

3.1. Neighborhood dictionary

INK operates by selecting nodes of interest. This can be both all nodes within a graph (for task-agnostic mining), as well as some nodes specified upfront (task-specific mining). In our example KG, we select two nodes of interest: `Chris Martin` and `Guy Berryman`. INK will first query the neighbourhood of a given depth for all these nodes of interest. If we define the depth parameter K to be two, the neighbourhood for `Chris Martin` will exist of the ALMA MATER and BORN IN relations, together with the neighbourhood of the *University College London* node providing the LOCATED IN relation and the neighbourhood of the *England* node with the PART OF relation. To store these neighbourhoods efficiently, a dictionary representation is used. For a given node of interest, this dictionary is built in an iterative fashion. The predicates in a neighbourhood of depth one are inserted first into our dictionary, together with their corresponding objects as values. These dictionary values are lists, as a single predicate can occur multiple times with different objects in the neighbourhood of a node. For our given example node `Chris Martin`, we represent the neighbourhood at depth one by:

$$\{\text{ALMA MATER} \rightarrow [\textit{University College London}],$$
$$\text{BORN IN} \rightarrow [\textit{England}]\}$$

To add the neighbourhoods of depths > one, INK concatenates the predicates together. By concatenating these relations, INK provides a path from the node of interest to another node within our graph without providing detailed information about all intermediate nodes on that path. However, this information is still available in the (key, value) pairs added to our dictionary at the lower neighbourhood's depths. In our example node, the previous dictionary will be extended with the following (key, value) pairs at depth two:

$$\text{ALMA MATER.LOCATED IN} \rightarrow [\textit{England}]$$
$$\text{BORN IN.PART OF} \rightarrow [\textit{UK}]$$

Here, we see a link from the node of interest to the *UK* node over the BORN IN relation. The BORN IN object value is not represented in this (key, value) pair, but was specified at the previous depth 1.

In several cases, it is also beneficial to indicate that the relationship itself within the neighbourhood of a node of interest is provided. The current dictionary structure does not indicate this presence explicitly in the dictionary values list. To ensure multiple nodes of interesting with similar relationship edges within their neighbourhoods can be compared against each other on a predicate level, the transformation step will also explicitly state particular relationships are available:

$$\text{ALMA MATER} \rightarrow [\textit{True}],$$
$$\text{BORN IN} \rightarrow [\textit{True}],$$
$$\text{ALMA MATER.LOCATED IN} \rightarrow [\textit{True}]$$
$$\text{BORN IN.PART OF} \rightarrow [\textit{True}]$$

Again, lists were used as values for our dictionary as a single node can have the same predicate multiple times, but with different object values.

Completely similar, the dictionary representation of `Guy Berryman` until depth 2 is:

$$\{\text{BORN IN} \rightarrow [\textit{Scotland}],$$
$$\text{BORN IN.PART OF} \rightarrow [\textit{UK}]$$

BORN IN → [*True*],

BORN IN.PART OF → [*True*]}

More in general, combined for all nodes of interest $\mathcal{N}$, the initial data structure of INK uses the following format:

$$list(tuple(n, neighbourhood(n,k))), \forall \text{ n in } \mathcal{N}$$

where the $neighbourhood(n,K)$ is the function which outputs the dictionary representation for our node *n* till a defined depth K.

3.2. Binary format

As the $list(tuple(n, neighbourhood(n,k)))$ representation is 3 dimensional (one axis for the nodes of interest, one for the dictionary relation keys and one for dictionary object list values), an additional transformation is required to provide a binary representation of this data. All of the object's values inside our neighbourhood dictionary are combined using a delimiter § to their corresponding key. In the strict sense, the binary format is created by unravelling the value lists within our dictionary by concating them with the corresponding dictionary key. For the example nodes `Chris Martin` and `Guy Berryman`, we transformed both depth one and depth two neighbourhoods into:

ALMA MATER§*University College London* (1)

BORN IN§*England* (2)

BORN IN§*Scotland* (3)

ALMA MATER.LOCATED IN§*England* (4)

BORN IN.PART OF§*UK* (5)

ALMA MATER§*True* (6)

BORN IN§*True* (7)

ALMA MATER.LOCATED IN§*True* (8)

BORN IN.PART OF§*True* (9)

These transformations can now be easily represented as a binary matrix. The rows are defined by the nodes of interest, such that each cell indicates whether or not the subject of interest contains the relation(s)§object value. The binary INK representation for the example above is visualised in Table 1. Here, (3) is a specific relation for `Guy Berryman` and could be of interest to differentiate `Guy Berryman` from the other team members.

Table 1
INK's binary representation of `Chris Martin` and `Guy Berryman` nodes in the example graph of Figure 1

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
Chris Martin	1	1	0	1	1	1	1	1	1
Guy Berryman	0	0	1	0	1	0	1	0	1

3.3. Extension modules

While the binary representation of INK reflects the whole KG, it can derive additional information based, e.g. datatype properties or the amount of relationships that are available. This subsection describes two optional extension modules which are available in INK.

3.3.1. Numerical inequality

To deal with numerical data, a preprocessing module will check if the values corresponding to a specific relation are all floats or integers for all the corresponding objects and nodes of interest. When such a relation is found, we build a set of all possible inequalities using all the found objects for that relation. In our example KG, we could add the birth year of all our Coldplay members, which would be an integer value. When this extension module is enabled, all these integer values will be stored inside a set. INK compares for each node of interest the value of the BIRTH YEAR relation with all possible values in our set and adds a new entry to our neighbourhood dictionary as follow:

BIRTH YEAR $< l \rightarrow$ [*True or False*] &

$l >=$ BIRTH YEAR $\rightarrow$ [*True or False*],

$\forall$ l in inequality set

Concrete, eight new entries for will be added, describing if the BIRTH YEAR of `Band Member X` is smaller than the BIRTH YEAR of the `Band Member Y`, or if BIRTH YEAR of `Band Member X` is greater than or equal the BIRTH YEAR of the `Band Member Y`, with both X and Y $\in$ `Chis Martin, Will Champion, Guy Berryman, Jonny Buckland`.

3.3.2. Relation count

Another preprocessing module is available in INK to deal with relations having more than one object value. It can be beneficial to indicate how many similar relations are available for a given node of interest. Therefore, a counting module adds new entries to the neighbourhoods dictionary when two or more similar relations are available. More specifically, if in our example graph `Chis Martin` would have a second ALMA MATER relation, this module would add the following entry:

$$\text{COUNT.ALMA MATER} \rightarrow [2]$$

This entry can be directly transformed to

$$\text{COUNT.ALMA MATER}\S 2$$

as described above. The previous inequality module can also use these counting values, as they are stored as integer values.

4. INK Rule Mining

The matrix representation discussed above can be used to perform both task-specific and task-agnostic rule mining. More in general, rules will be built based on the column description of our binary matrix, as shown in Table 1. The fourth column in this example, i.e. ALMA MATER.LOCATED IN§*England*, already introduced implicitly a variable to ignore the specific alma mater located in England. This fourth column states that there is a relation from our nodes of interest about a non-specified university, school, or college that one formerly attended which is located in England. This column can be interpreted more formally by:

$$\text{ALMA MATER}(?i, ?x) \wedge \text{LOCATED IN}(?x, \textit{England})$$

with $?x$ and $?i$ a variable.

As both task-specific and task-agnostic techniques use this representation, the only difference between them is how they interact and extract the relevant information. The task-agnostic miner operates on the columns themselves, comparing the Boolean values to build so-called frequent itemsets and create the rules. The task-specific miner operates on the rows to differentiate between the nodes of interest. The task-specific miner uses the columns as features within its model to define a more specific rule given the task it wants to solve.

4.1. Task-agnostic mining

Similar to all ARM techniques, the INK task-agnostic mining component defines frequent itemsets. Here, the frequent itemsets are, however, based on the columns of INK's binary representation. The most important aspect to calculate these frequent itemsets is the support level. But in order to build these frequent itemsets, we first have to extract the neighbourhood for all subject nodes containing a fact in our KG. For our example graph in Figure 1, this means that not only the neighbourhoods for `Chris Martin` and `Guy Berryman` will be extracted, but also for all other nodes in our KG as they are also a subjects of facts.

While the relation§object columns were are by default extracted by INK, the task-agnostic miner is more interested in the relation only columns. These columns already introduce a variable near the end, for example we can write ALMA MATER(?X) to indicate that those columns contain an, not specified, alma mater, indicated through the variable X. Based on these relation columns, the frequent itemsets are determined by all the possible combinations of a predefined length.

In traditional frequent itemset miners, the number of items inside the set must be defined upfront. For INK, we fix the number of items within an itemset to two, as the depth parameter of the neighbourhood already implicitly introduces additional preconfigured items in our itemsets with possible lengths greater than one. In our example graph, the relation ALMA MATER.LOCATED IN is already such a predefined item, combining the ALMA MATER and LOCATED IN relationship. As this combined relation can already be found in the neighbourhood of the nodes of interest, there is a high chance that they can occur frequently together. When we add to this combined relation, a second, single relation, we implicitly have a frequent itemset of length 3. For example, if we combine ALMA MATER.LOCATED IN, with BORN IN, we get an itemset stating (ALMA MATER(?X, ?Y) $\wedge$ LOCATED IN(?Y, ?Z), BORN IN(?X, ?Z)). However, pure algorithmic, the length of the itemset remains to two. We just provided additional variables within the items themselves to chain relationships, occurring frequently together, within the KG. INK is in this perspective also not limited to mine closed rules while the rule can still be connected. The head atom can also contain additional free variables due INK's item representation within the frequent itemsets. Both advantages eventually lead to more and more advanced rules.

Whether two items within an itemsets represent an interesting rule, depends on the calculated support value and corresponding threshold. The support for each itemset within INK is calculated using the following rule:

$$|\forall_{R_1} \text{ in } C_{ink}, \forall_{R_2} \text{ in } C_{ink} \sum_{\forall \text{ o in } O_{R_1 \cap R_2}}^{R_1 != R_2} R_1 \S o \ \ \& \ \ R_2 \S o|$$

Where C_{ink} are all the relation-only columns of our INK representation, $O_{R_1 \cap R_2}$ contains the intersection of all object values of the two relations R_1 and R_2 and & is the bitwise and operator. Note that both R_1 and R_2 can be a chain of relationships as discussed above. The individual support measures for each of these items within the itemsets is calculated by counting the number of true values for each relation. Based on these itemsets, we can select both an antecedent and consequent to get rules of interest. Measures such as confidence, lift and conviction are calculated from the support values and can be used to filter these rules.

4.2. Task-specific mining

The task-specific mining approach is based on the Bayesian rule set mining technique described by Wang et al. [20]. In this approach, the model consists of a set of rules and each rule is a conjunction of conditions. The model predicts that an observation is in a positive class when at least one of these rules is satisfied. In contrast, the observation belongs to the negative class if none of the rules apply. The approach described by Wang et al. optimises the search for these rule sets by relying on Bayesian analysis. A globally optimised rule set is learned by considering both the accuracy and the interpretability of a model, while keeping computation simple. By controlling the parameters of the Bayesian prior, large models are penalised. For the Bayesian Rule Set model, this results in a a smaller number of rules. Since a small number of rules must cover the positive class, each rule in this model must cover as many observations as possible. To enforce this, a threshold on the number of examples satisfying the rule, more commonly known as the support of a rule, is introduced. It is due to this threshold that a significant reduction of the rule set's search space can be made.

The required input for this Bayesian rule set mining is a binary matrix, which fit with the proposed INK representation of Section 3. All the extension modules enrich this binary representation, such that all of them can be used as well. To train this model, INK will extract the neighbourhoods from two sets of nodes of interest, for a given depth parameter. One set contains all the positive nodes, the other set contains all negative ones. For task-specific cases, it is therefore required to specify these sets upfront. The labels for each node are stored in a different array. Optional parameters, such as the support, maximum length of the concatenation and the maximum number of rules in the rule set can be provided as input for the algorithm.

The output of the Bayesian rule set mining module contains both the rules learned on the given training dataset to discriminate both positive and negative nodes of interest, as well the mechanism to evaluate the rules on the new unseen nodes.

5. Implementation

The INK representation described in Section 3 and the INK rule mining module of Section 4 are both implemented in Python. To extract the neighbourhoods for a given set of nodes of interest, a component was implemented which can query these relations iteratively. Two options are currently available inside this component: either a KG or a SPARQL endpoint is given as input. When a KG file is given, RDFLib [21] will be used to load the graph in the internal memory of the operating system. However, some large KGs can be hard to fit within the internal memory. Therefore, INK can use RDFLib in combination with the Header, Dictionary Triples (HDT) file format [22]. HDT compresses big RDF datasets while maintaining basic search operations, such as providing the neighbourhood of a node of interest. Listing 1 shows the query used for both options to extract the neighbourhood nodes and relation. The variable subject <ind> starts with the nodes of interest, but differs in each iteration given the graph. The datatype of the object within this query is used to determine if queried objects can be used as subjects in the next iteration (when the neighbourhood depth is not reached yet). The predicates and objects in each iteration are stored as described in Section 3. Python's internal multiprocessing library is used to speed up the extraction of the neighbourhoods, as this operation can be performed over multiple processors given the amount of nodes of interest.

To transform the initial representation into a binary matrix, we used the Scikit-learn DictVectorizer [23] with the sparse option set to true and specifying the data type to be Boolean. This is necessary when we want to deal with large KGs and a large number of nodes of interest.

If target values are specified together with the nodes of interest, the INK miner assumes a task-specific mining operation can be executed. Code from Wang et. al. [20] was adapted to operate on our representation.

When a target array is not provided, task-agnostic mining is executed on the neighbourhoods of all nodes of interest. The task-agnostic code uses the MLxtend library [24] to produce the rules based on the calculated frequent itemsets, based on the INK representation.

The whole INK package is made available on GitHub[1].

```
SELECT ?p ?o ?dt
WHERE {
 <ind> ?p ?o.
 BIND (datatype(?o) AS ?dt)
}
```

Listing 1: SPARQL query

6. Evaluation set-up & results

Both the task-specific and task-agnostic mining capabilities are evaluated on multiple benchmark datasets as specified below. To extract the neighbourhoods of interest, all benchmark datasets were transformed to an HDT format such that the SPARQL query of listing 1 can be executed performant. All evaluations were performed on an Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz processor with 32 cores and 128gb RAM.

6.1. INK vs AMIE

To compare the task independent rule mining capacities, we made a comparison between INK and AMIE on five benchmark datasets, which were already frequently used during various AMIE evaluations. All these datasets are available in a TSV format (preferred by AMIE) and are transformed into HDT (after being transformed into NTRIPLES) for INK. YAGO (2 and 2s) is a semantic knowledge base derived from Wikipedia, WordNet and GeoNames [25]. The latest version, YAGO2s, contains 120M facts describing properties of 10M different entities. The DBpedia datasets (2.0 and 3.8) are a subset of the crowd-sourced community effort to extract structured information from Wikipedia [2]. DBpedia 2.0 and DBpedia 3.8 contain 6.7M and 11.02M facts respectively. The Wikidata dataset is a Wikidata dump from December 2014 and contains 8.4M facts. Wikidata is a free, community-based knowledge base maintained by the Wikimedia Foundation with the goal to provide the same information as Wikipedia but in a computer-readable format [26]. All 5 benchmark datasets are made available by the Max Planck Institute[2].

Both AMIE and INK prune rules based on both the default support level of 100 and a default max rule length of 3. To mine rules of length 3, the INK neighbourhood's depth parameter was set to 2. This could result in rules containing atoms for both the head and body of length 2. When the support level of those atoms is above the provided thresholds, they can both be combined into a rule which implicitly results in a rule with length of 4. To make a fair comparison towards the mined AMIE rules of length 3, we filtered all those length 4 rules.

For all datasets, the cofidence of the top 10, top 25 and top N, with N the smallest number of rules from either INK or AMIE are compared as shown in Table 2. The total number of filtered rules is also listed for both AMIE and INK.

For all benchmark datasets, INK mined a lot more rules compared to the AMIE results. The confidence values are also higher, indicating that the additional rules can be of high value.

6.2. INK vs DL-Learner

To compare the INK miner in the context of task-specific mining, we used the Structured Machine Learning benchmark framework (SML-Bench) [27]. This framework enables some specific tasks where structured hypotheses are learned from data with a rich internal structure or knowledge representation, usually in the form of one or more relations. The systems within this framework might differ in the knowledge representation languages they support and the

[1] https://github.com/IBCNServices/INK

[2] https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/amie

Table 2
Comparison between INK and AMIE3 on 5 benchmark datasets. Both the confidences measures for the top 10, 25 and top min number of rules of either INK or AMIE3 is visualized

	Confidence Top 10		Confidence Top 25		Confidence Top N		# Rules	
	INK	AMIE3	INK	AMIE3	INK	AMIE3	INK	AMIE3
Yago2	**0.553 (0.09)**	0.507 (0.08)	**0.421 (0.13)**	0.353 (0.15)	**0.12 (0.15)**	0.086 (0.13)	**294**	166
Yago2s	**0.927 (0.05)**	0.898 (0.08)	**0.787 (0.14)**	0.707 (0.18)	**0.31 (0.24)**	0.221 (0.23)	**754**	405
DBpedia 2.0	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	**0.329 (0.27)**	0.238 (0.3)	**16957**	8963
DBpedia 3.8	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	**0.162 (0.22)**	0.126 (0.21)	**16499**	9383
Wikidata	1.0 (0.0)	1.0 (0.0)	0.998 (0.0)	0.998 (0.0)	**0.287 (0.29)**	0.223 (0.3)	**3993**	2121

programming languages they are written in. By incorporating INK within this framework, a fair comparison between INK and DL-Learner, which has already been used in many evaluations within the SML-Bench framework. The SML-Bench framework already evaluated seven strucuterd machine learning models. The code to incorporate INK within the SML-Bench framework is also provided online[3] such that INK can also be used in future evaluations.

In total, nine different datasets are available in the SML-Bench 3.0 version, all containing an OWL knowledge base and a single task based on two sets of files indicating the positive and negative nodes of interest.

The SML-Bench framework also contains other learning systems beyond DL-Learner and INK, but those systems do not take any semantic knowledge into account and were therefore neglected. The default SML-Bench configuration options were used within all our evaluations: 10-fold cross validation was used with a maximum execution time of 15 minutes for each fold. DL-learner version 1.5 was used with the by SML-Bench default parameters for each learning task: For all tests, the CELEO algorithm was used to traverse the search space guided by the Pellet reasoner. INK was initialised with a maximum neighbourhood depth of 3 such that the neighbourhoods of the neighbours from our start nodes were taken into account during the rule generation phase. The numerical levels and relation count extension modules described in Section 3.3 were also enabled. The OWL datasets were transformed into the HDT format, which was used as input to generate the INK representation. Four different metrics are reported in Table 4:

- **Accuracy score:** The number of correct predictions divided by all predictions. All learning tasks are binary classification problems, but can be unbalanced. We report the average accuracy score between 0 and 1 together with the standard deviation across the 10 folds.
- **F1 score:** The harmonic mean of the precision and recall: $F1 = 2 * \frac{precision*recall}{precision+recall}$. Again, the average and standard deviation over 10 folds are reported.
- **Matthews Correlation Coefficient (MCC):** This metric takes into account the true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes: $MCC = \frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$. MCC returns a value between -1 and $+1$: A coefficient of $+1$ represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. Again, averages and standard deviations over 10 folds are reported.
- **Duration:** The time needed to find the most descriptive rule, based on the nine out of 10-folds + the time to evaluate this rule on one holdout fold. Averages and standard deviations over 10 folds are reported in seconds.

The results for each learning task are provided in Table 4. The task-specific rule mining results showed that INK is highly competitive with DL-Learner.

- For the carcinogenesis dataset, DL-Learner founds a rule describing the node of interest has minimal five HASATOM relationships which are not linked to the `Iodine-95` object node (HASATOM min 5 (not (Iodine-95))). The INK rule triggers when at least three `Hydrogen-3` objects nodes are linked to the HASATOM relationship (count.hasAtom.type.Hydrogen-3 >= 3).
- The DL-Learner rule defined for the hepatitis dataset states the sample must be either Male or having a COLISTERASEACTIVITYLEVEL <= 5.

[3] https://github.com/IBCNServices/INK

Table 3

Overview of the datasets that are part of SML-Bench with their number of axioms (#A), classes (#C), object properties (#O), datatype properties (#D) and their description

Dataset	#A	#C	#O	#D	N.o.i.	pos/neg	Prediction of
Carcinogenesis	74,566	142	4	15	298	1.19	Carcinogenic drugs
Hepatitis	73,114	14	5	12	500	0.70	Hepatitis type based on patient data
Lymphography	2,187	53	0	0	148	1.21	Diagnosis class based on lymphography patient data
Mammographic	6,808	19	3	2	961	0.86	Breast cancer severity
Mutagenesis	62,066	86	5	6	42	0.44	Mutagenicity of chemical compounds
NCTRER	92,861	37	9	50	224	1.41	Molecule's estrogen receptor binding activity
Prem. League	214,566	10	14	202	81	0.97	Goal keepers based on player statistics
Pyrimidine	2,006	1	0	27	40	1.0	Inhibition activity of pyrimidines and the DHFR enzyme
Suramin	13,506	46	3	1	17	0.70	Suramin analogues for cancer treatment

Table 4

SML-Benchmark comparison between INK and DL-Learner for 4 metrics on 8 benchmark datasets. The results show both the average and standard deviation for a 10-fold cross validation.

	Accuracy (std)		F1 (std)		MCC (std)		Duration (std)	
	INK	DL-Learner	INK	DL-Learner	INK	DL-Learner	INK	DL-Learner
carcinogenesis	**0.56 (0.12)**	0.54 (0.02)	0.47 (0.13)	**0.70 (0.01)**	**0.18 (0.28)**	0.00 (0.09)	**191.4 (23.31)**	888.3 (0.46)
hepatitis	**0.78 (0.03)**	0.49 (0.06)	**0.73 (0.07)**	0.61 (0.03)	**0.56 (0.09)**	0.21 (0.07)	**55.4 (1.28)**	879.0 (0.0)
lymphography	0.80 (0.09)	**0.82 (0.1)**	0.82 (0.07)	**0.86 (0.07)**	0.64 (0.15)	**0.67 (0.18)**	**33.3 (1.27)**	873.7 (0.46)
mammographic	**0.83 (0.04)**	0.49 (0.02)	**0.80 (0.05)**	0.64 (0.01)	**0.66 (0.07)**	0.12 (0.1)	**85.8 (2.52)**	874.1 (0.3)
mutagenesis	**0.98 (0.06)**	0.94 (0.13)	**0.97 (0.1)**	0.93 (0.13)	**0.96 (0.12)**	0.9 (0.2)	**31.4 (0.8)**	883.0 (0.0)
nctrer	**0.99 (0.2)**	0.59 (0.04)	**0.99 (0.02)**	0.73 (0.02)	**0.98 (0.04)**	0.01 (0.12)	**201.9 (3.14)**	885.2 (0.87)
premierleague	**0.99 (0.04)**	DNF	**0.99 (0.04)**	DNF	**0.98 (0.07)**	DNF	**167.9 (2.7))**	DNF
pyrimidine	**0.95 (0.1)**	0.82 (0.16)	**0.93 (0.13)**	0.84 (0.14)	**0.92 (0.17)**	0.69 (0.3)	**28.0 (0.89)**	874.0 (0.0)
Suramin	0.65 (0.32)	**0.71 (0.25)**	0.33 (0.42)	**0.71 (0.33)**	0.2 (0.4)	**0.43 (0.49)**	**26.7 (0.9)**	875.0 (0.0)

The mined INK rule is more precise: the sample must have either a COLISTERASEACTIVITYLEVEL <=5 or less or equal to 19 zincSulfateTurbidityTestLevel.2 screenings, more than 8 and less or equal to 12 totalBilirubinLevel.2 screenings with an in total of less than 24 screenings.

- For the lymphography dataset, DL-Learner founds a rule describing the sample should be of the class CIN14_Lac_Margin or (CIN14_Lacunar and (not (SF16_Vesicles))) or (NON19_n0-9 and (not (LNE11_3))). INK only founds the rule stating the sample should of the class CIN14_Lac_Margin.
- In the mammographic dataset, the DL-Learner mined rule states the sample must have an HASAGE >= 23.5. The INK rule miner generates the following rule: HASBIRADS >= 5.0 or HASSHAPE of type irregular with HASAGE >= 69.
- The rules for the mutagenesis dataset for the DL-Learner and INK technique are quite similar: act >= -0.22 for DL-Learner, act >= 0.3 for INK.
- For the nctrer dataset, the DL-Learner rule states the sample should have minimal two atoms with minimal three bindings. The INK rule looks at the coordinate positions of the FIRST_BOUND_ATOM and SECOND_BOUND_ATOM in combination with ACTIVITYSCORE_NCTRER>=32.
- The INK rule in the premierleague dataset is defined as a sample having an action with a GOALKEEPER_DISTRIBUTION >= 5.
- For the pyrimidine dataset, DL-Learner found a rule where a sample must have a P1_SIZE >= 0.34 or P3_SIZE >= 0.233 and P2_SIZE >= 0.18. The mined INK rule defines a similar rule where the sample must have a P1_POLARIZABLE>=0.367, P1_PI_DONOR<=0.5 and a P2_SIZE>=0.26.
- DL-Learner finds the following rule for a sample within the Suramin dataset: HASATOM with minimal 4 (Hydrogen-8 or Nitrogen-32). The INK rule states the sample should have the HASBOND relationship with an INBOND CHARGE <= -0.622.

7. Discussion

Based on the results provided in Section 6, the INK representation and defined INK miners show for both task-specific and task-agnostics rule mining interesting results. In this section, we provide an explanation on why INK is beneficial in the context of rule mining, but also discuss some of the obtained results to show its limitations.

7.1. INK within task-agnostic rule mining

Compared to AMIE3, INK mined in all datasets more rules and most of these rules have also a high confidence level. The larger number of rules are mainly due to the fact that INK is also capable of analysing head atoms with more than 2 variables. While in previous works the perception raised that those rules could be neglected as their confidence level should be extremely low, INK showed that some of these rules do occur quite frequently in large datasets. An example of such a rule in DBpedia 3.8: *?a isCitizenOf ?b* $\Rightarrow$ *?a wasBornIn ?x* $\wedge$ *?x isLocatedIn ?b* (confidence: 0.27). INK also mines non-closed or open rules here as a variable inside the rule can only occur only once.

Beyond the scope of the current evaluation, INK does have some disadvantages compared to AMIE3. INK consumes a large amount of RAM in order to build the internal representation and to generate the frequent itemsets. AMIE3 is designed to mine rules iteratively and therefore uses less RAM to obtain rules. INK's configuration settings are currently also limited as AMIE can also take into account constants, PCA confidence, removals of perfect rules, etc. INK does however have the capability to mine long rules (rules with a large amount of atoms) without expanding the frequent itemsets. The internal INK representation concatenates relationships within the neighbourhood of the nodes together and represent them already as sub rule parts within the binary INK representation (crf the ALMA MATER.LOCATED IN path or sub rule within our example graph of Figure 1). The support of those rule parts can be easily calculated by analysing how many nodes have this sub part in its neighbourhood. A frequent itemset can combine this rule part directly with another rule part to generate a confident rule. By following this approach, INK always has frequent itemsets with maximum 2 items within each set. This simplifies a large amount of calculations as only 2 rules (one for A$\Rightarrow$ B and B $\Rightarrow$ A) have to be generated and evaluated within each itemset.

7.2. INK within task-specific rule mining

The INK miner holds both a predictive and time advantage compared to DL-Learning in the context of task-specific rule mining, given a large enough positive and negative set of instances. DL-Learner typically traverses the search space until the predefined amount of time is reached. This is a first advantage of INK over DL-learner, as it doesn't need to tune this time parameter.

More in depth, within the Carcinogenesis dataset, the accuracy measures for both INK and DL-Learner are similar. DL-Learner, however, optimises its rules to benefit the instances of the majority class. These cases are reflected in a MCC score close to zero, which indicates that the used rules hold the same predictive performance as a random classifier. For both the rules of the Carcinogenesis and NCTRER datasets, DL-Learner obtained such a MCC score of zero. INK obtains a positive MCC score for these datasets.

In contrast, for the Lymphography and Suramin dataset, INK's MCC scores is lower than the MCC scores of DL-Learner. The explanation is two-fold. First, DL-Learner introduces negation within its rules. By explicitly stating within a rule, a concept must not be available, DL-Learner is able to obtain a predictive advantage. Within the INK transformation steps, no such negation operator is defined for now. By doing this, INK postpones the decision to close the world (crf. Closed-World Assumption) to the learning task and accompanying learning model. INK opens up the possibility to adjust future rule miners to deal with these open-world cases based on its representation. Second, some of the benchmark datasets have a too small set of nodes of interest for INK to be operational. While DL-Learner is able to correctly define generic rules for the Suramin dataset, INK's strengths lie within larger datasets, with more nodes of interest to mine rules from.

For the Prem. League dataset, DL-Learner was unable to finish the training procedure within the time limit of 15 minutes. In contrast, the most interesting rules generated from the INK miner were available within less than 3 minutes. In general, DL-Learner always searches for better, more descriptive and generic rules when enough time is left. This behaviour is also stated in the obtained results of Table 4. Here, nevertheless the used dataset, the duration of the DL-Learner training and evaluation phase is almost always the same. The difference in time across multiple datasets is due to the loading phase of the dataset

itself before the actual rule mining starts. The maximum execution time is a parameter within the DL-Learner configuration file. INK does not have such a timing constraint, but is constrained in rule mining's search space by limiting the neighbourhood's depth. As shown in the performed experiments, high quality rules can already been found when limiting the neighbourhood depth to three.

The INK rules for the Hepatitis, Mammographic, Mutagenesis and Pyrimidine datasets extend in some sort the obtained DL-Learner rules. In most of these cases INK finds additional information within the neighbourhood and adds one or two extra rule atoms or sub rules to achieve a better predictive performance. INK is also able to better define the numerical properties within a rule. DL-Learner tries to minimise the full integer or floating point range when mining such rules, while INK uses the available data within the neighbourhood to already limit the ranges upfront in the rule mining process.

8. Conclusion

In this work, we addressed the current problems of both task-specific and task-agnostic semantic rule mining and the need for one technique which can perform both. The main contribution to fulfil this need is the development of an internal representation benefiting both techniques. INK is such a representation, where the neighbourhood of nodes in a KG are represented as a binary matrix. Combining this INK representation with a Bayesian Rule miner resulted in outperforming the current state of the art methods to perform structured machine learning, both in prediction performance and in time. The same representation can be used to mine frequent itemsets of nodes of interest and build general rules filtered by confidence and a given support level. Compared with the filtered results of AMIE, more confident and new rules were mined by INK for several benchmark datasets.

The INK representation resembles a binary vector matrix, and can be used in several other situations going beyond the general purpose of rule mining. Future work will adapt INK to mine rules with both constants or a wider range scalar data in combination with a temporal aspect. This would enable INK to mine temporal rules, originated from a e.g sensor or more broader, Internet of Things (IoT) streaming data domain.

References

[1] L. Ehrlinger and W. Wöß, Towards a Definition of Knowledge Graphs., *SEMANTiCS (Posters, Demos, SuCCESS)* **48** (2016).

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, Dbpedia: A nucleus for a web of open data, in: *The semantic web*, Springer, 2007, pp. 722–735.

[3] B. Steenwinckel, D. De Paepe, S. Vanden Hautte, P. Heyvaert, M. Bentefrit, P. Moens, A. Dimou, B. Van Den Bossche, F. De Turck, S. Van Hoecke and F. Ongenae, FLAGS: A methodology for adaptive anomaly detection and root cause analysis on sensor data streams by fusing expert knowledge with machine learning, *Future Generation Computer Systems* **116** (2021), 30–48. doi:https://doi.org/10.1016/j.future.2020.10.015. https://www.sciencedirect.com/science/article/pii/S0167739X20329927.

[4] G. Vandewiele, F. De Backere, K. Lannoye, M.V. Berghe, O. Janssens, S. Van Hoecke, V. Keereman, K. Paemeleire, F. Ongenae and F. De Turck, A decision support system to follow up and diagnose primary headache patients using semantically enriched data, *BMC medical informatics and decision making* **18**(1) (2018), 1–15.

[5] P. Bonte, R. Tommasini, E. Della Valle, F. De Turck and F. Ongenae, Streaming MASSIF: cascading reasoning for efficient processing of IoT data streams, *Sensors* **18**(11) (2018), 3832.

[6] S. Ji, S. Pan, E. Cambria, P. Marttinen and S.Y. Philip, A survey on knowledge graphs: Representation, acquisition, and applications, *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[7] T. Ebisu and R. Ichise, Graph pattern entity ranking model for knowledge graph completion, *arXiv preprint arXiv:1904.02856* (2019).

[8] S. Ortona, V.V. Meduri and P. Papotti, Robust discovery of positive and negative rules in knowledge bases, in: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, IEEE, 2018, pp. 1168–1179.

[9] N. Jain and V. Srivastava, Data mining techniques: a survey paper, *IJRET: International Journal of Research in Engineering and Technology* **2**(11) (2013), 2319–1163.

[10] L.A. Galárraga, C. Teflioudi, K. Hose and F. Suchanek, AMIE: association rule mining under incomplete evidence in ontological knowledge bases, in: *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 413–422.

[11] J. Minker, On indefinite databases and the closed world assumption, in: *International Conference on Automated Deduction*, Springer, 1982, pp. 292–308.

[12] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun and W. Zhang, Knowledge vault: A web-scale approach to probabilistic knowledge fusion, in: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 601–610.

[13] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+, *The VLDB Journal* **24**(6) (2015), 707–730.

[14] J. Lajus, L. Galárraga and F. Suchanek, Fast and Exact Rule Mining with AMIE 3, in: *European Semantic Web Conference*, Springer, 2020, pp. 36–52.

[15] J. Lehmann, DL-Learner: learning concepts in description logics, *Journal of Machine Learning Research* **10**(Nov) (2009), 2639–2642.

[16] J. Lehmann, *Learning OWL class expressions*, Vol. 22, IOS Press, 2010.
[17] L. Bühmann, J. Lehmann, P. Westphal and S. Bin, Dl-learner structured machine learning on semantic web data, in: *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 467–471.
[18] V.T. Ho, D. Stepanova, M.H. Gad-Elrab, E. Kharlamov and G. Weikum, Rule learning from knowledge graphs guided by embedding models, in: *International Semantic Web Conference*, Springer, 2018, pp. 72–90.
[19] P.G. Omran, K. Wang and Z. Wang, An embedding-based approach to rule learning in knowledge graphs, *IEEE Transactions on Knowledge and Data Engineering* **33**(4) (2019), 1348–1359.
[20] T. Wang, C. Rudin, F. Velez-Doshi, Y. Liu, E. Klampfl and P. MacNeille, Bayesian rule sets for interpretable classification, in: *2016 IEEE 16th International Conference on Data Mining (ICDM)*, IEEE, 2016, pp. 1269–1274.
[21] D. Krech, RDFLib: A Python library for working with RDF, Online https://github. com/RDFLib/rdflib, 2006.
[22] J.D. Fernández, M.A. Martínez-Prieto, C. Gutiérrez, A. Polleres and M. Arias, Binary RDF Representation for Publication and Exchange (HDT), *Web Semantics: Science, Services and Agents on the World Wide Web* **19** (2013), 22–41–. http://www.websemanticsjournal.org/index.php/ps/article/view/328.
[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., Scikit-learn: Machine learning in Python, *the Journal of machine Learning research* **12** (2011), 2825–2830.
[24] S. Raschka, MLxtend: providing machine learning and data science utilities and extensions to Python's scientific computing stack, *Journal of open source software* **3**(24) (2018), 638.
[25] T. Rebele, F. Suchanek, J. Hoffart, J. Biega, E. Kuzey and G. Weikum, YAGO: A multilingual knowledge base from wikipedia, wordnet, and geonames, in: *International semantic web conference*, Springer, 2016, pp. 177–185.
[26] D. Vrandečić and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Communications of the ACM* **57**(10) (2014), 78–85.
[27] P. Westphal, L. Bühmann, S. Bin, H. Jabeen and J. Lehmann, SML-Bench–A benchmarking framework for structured machine learning, *Semantic Web* **10**(2) (2019), 231–245.
[28] H. Sahu, S. Shrma and S. Gondhalakar, A brief overview on data mining survey, *International Journal of Computer Technology and Electronics Engineering (IJCTEE)* **1**(3) (2011), 114–121.
[29] Q. Wang, Z. Mao, B. Wang and L. Guo, Knowledge graph embedding: A survey of approaches and applications, *IEEE Transactions on Knowledge and Data Engineering* **29**(12) (2017), 2724–2743.
[30] Q. Zhao and S.S. Bhowmick, Association rule mining: A survey, *Nanyang Technological University, Singapore* (2003), 135.
[31] S. Džeroski, Relational data mining, in: *Data Mining and Knowledge Discovery Handbook*, Springer, 2009, pp. 887–911.
[32] L.D. Raedt, K. Kersting, S. Natarajan and D. Poole, Statistical relational artificial intelligence: Logic, probability, and computation, *Synthesis Lectures on Artificial Intelligence and Machine Learning* **10**(2) (2016), 1–189.
[33] M. Nickel, K. Murphy, V. Tresp and E. Gabrilovich, A review of relational machine learning for knowledge graphs, *Proceedings of the IEEE* **104**(1) (2015), 11–33.
[34] L. Galárraga, Applications of rule mining in knowledge bases, in: *Proceedings of the 7th Workshop on Ph. D Students*, 2014, pp. 45–49.
[35] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, *Semantic web* **8**(3) (2017), 489–508.
[36] L. De Raedt and K. Kersting, *Statistical Relational Learning*, in: *Encyclopedia of Machine Learning and Data Mining*, C. Sammut and G.I. Webb, eds, Springer US, Boston, MA, 2017, pp. 1177–1187. ISBN 978-1-4899-7687-1.
[37] F. Ongenae, M. Claeys, T. Dupont, W. Kerckhove, P. Verhoeve, T. Dhaene and F. De Turck, A probabilistic ontology-based platform for self-learning context-aware healthcare applications, *Expert Systems with Applications* **40**(18) (2013), 7629–7646.
[38] K. Colpaert, E. Hoste, S. Van Hoecke, D. Vandijck, C. Danneels, K. Steurbaut, F. De Turck and J. Decruyenaere, Implementation of a real-time electronic alert based on the RIFLE criteria for acute kidney injury in ICU patients, *Acta Clinica Belgica* **62**(sup2) (2007), 322–325.
[39] K. Steurbaut, K. Colpaert, B. Gadeyne, P. Depuydt, P. Vosters, C. Danneels, D. Benoit, J. Decruyenaere and F. De Turck, COSARA: integrated service platform for infection surveillance and antibiotic management in the ICU, *Journal of medical systems* **36**(6) (2012), 3765–3775.
[40] P. Bonte, F. Ongenae and F.D. Turck, Towards Optimizing Hospital Patient Transports by Automatically Identifying Interpretable Causes of Delays, *International Journal of Software Engineering and Knowledge Engineering* **29**(06) (2019), 819–847.
[41] S. Asadifar and M. Kahani, Semantic association rule mining: A new approach for stock market prediction, in: *2017 2nd Conference on Swarm Intelligence and Evolutionary Computation (CSIEC)*, 2017, pp. 106–111. doi:10.1109/CSIEC.2017.7940158.