RuLeS: Synthetic Datasets for Rule Learning

Submission # 2728

Abstract

Logical rules are a popular knowledge representation language in many domains, they represent background knowledge and encode information that can be derived from given facts in a compact form. But rule formulation is complex and requires domain expertise, and today's knowledge graphs represent additional challenges. Several approaches for automation, learning rules based on example facts, have been proposed over time, including neural systems recently. Yet, the area is missing adequate evaluation approaches: existing datasets (i.e., facts and the rules to be learned) resemble toy examples that neither cover the various kinds of dependencies between rules nor allow for testing scalability. We present a tool for generating datasets and for evaluating rule learning systems.

1 Introduction

Logical rules are a popular knowledge representation language in many domains. They represent domain knowledge, encode information that can be derived from given facts in a compact form, and allow for logical reasoning. For example, given facts parent(ann, bob) and parent(bob, dan), the *Datalog rule* $\text{grandparent}(X, Z)$:- $\text{parent}(X, Y), \text{parent}(Y, Z)$, encodes the fact grandparent(ann, dan) and describes its dependency on the other facts. Moreover, if the data grows and new facts are added, we can automatically derive new knowledge. Since rule formulation is complex and requires domain expertise, *rule learning* [Raedt, 2008; Fürnkranz *et al.*, 2012] has been an active research in AI for a long time and recently revived with the application of deep learning.

In the area of inductive logic programming (ILP) [Muggleton, 1991], the goal is to induce logical rules (also, logic programs) based on example facts. Classical ILP systems such as FOIL [Quinlan, 1990] and Progol [Muggleton, 1995] usually apply exhaustive algorithms to mine rules for the given data and either require negative facts as counter-examples or assume a closed world (for an overview of classical ILP systems see Table 2 in [Stepanova *et al.*, 2018]). The *closed-world assumption* (CWA) basically states that all facts that are not explicitly given as true are assumed to be false.

Today, however, knowledge graphs (KGs) with their often incomplete, noisy, heterogeneous, and, especially, large amounts of data raise new problems and require new solutions. For instance, real data most often only partially satisfies the CWA and does not contain counter-examples. Moreover, in an open world, absent facts cannot be considered as counter-examples either since they are not considered as false. Therefore, the successor systems, with AMIE+ [Galárraga *et al.*, 2015] and RDF2Rules [Wang and Li, 2015] as the most prominent representatives, assume the data to be only partially complete and focus on rule learning in the sense of mining patterns that occur frequently in the data. Furthermore, they implement advanced optimization approaches that make them applicable in wider scenarios. In this way, they address already many of the issues that arise with today's knowledge graphs. Yet, their processing is still exhaustive.

Recently, neural rule learning approaches have been proposed [Yang *et al.*, 2017; Rocktäschel and Riedel, 2017; Evans and Grefenstette, 2018; Minervini *et al.*, 2018; Omran *et al.*, 2018; Campero *et al.*, 2018; Dong *et al.*, 2019] and they might be a good alternative since deep learning has coped with vast amounts of noisy and heterogeneous data already. The proposed solutions consider vector or matrix embeddings of symbols, facts and/or rules, and model inference using differentiable operations such as vector addition and matrix composition. However, the proposed methods are still premature: they only learn certain kinds of rules or lack scalability (e.g., because they search the entire rule space) and hence cannot compete with established rule mining systems such as AMIE+ yet, as shown in [Omran *et al.*, 2018], for example. Further, the reported results are questionable in terms of generalization: the test datasets (i.e., facts and the rules to be learned) either resemble toy examples that neither cover the various kinds of possible dependencies in rule sets nor allow for testing scalability, or they do not contain any rules at all because there simply are no rules yet for the popular, publicly available KGs (see, e.g., the evaluations in [Evans and Grefenstette, 2018] and [Omran *et al.*, 2018]).

Veronika: not 1000% sure (doesn't seem so), can Dong et al output the rules? if not, delete reference above

We are hence in a chicken-and-egg situation missing rules for KGs in practice in order to learn new rules over them. Observe that the general non-availability of rules is also the reason why there are no neural rule learning approaches as we would expect: systems trained on both a number of KGs and relevant rules over them, which are able to suggest rules over new KGs, containing facts that were not seen during training; the existing systems are trained on a single KG alone and suggest rules for exactly that KG.

In this paper, present the tool RuDa (**Ru**le Learning **Da**tasets) for generating datasets containing both facts and rules, and for evaluating rule learning systems. RuDa is highly parameterizable; for instance, the number of constants, predicates, facts, consequences of rules (i.e., determining to which extend a dataset satisfies the closed-world assumption), and noise (e.g., wrong consequences) in the data and the kinds of dependencies between rules can be selected freely. Moreover, RuDa allows for assessing the performance of rule learning systems given their results by computing classical metrics, such as Herbrand distance, and measures that are popular with today's neural approaches, such as Hits@n.

In summary, our contributions are as follows:

- We propose new, synthetic datasets for rule learning which have been generated by our tool RuDa and overcome the above mentioned shortcomings of existing datasets (Section 3).
- We describe RuDa in detail (Sections 4 and 5) and make it available for the public: <link-in-final-version>.
- We evaluate representatives of the different types of rule learning systems on our datasets (Section 6).

2 Rule Learning

In this section, we describe the task of rule learning in more detail and motivate our work by showing the difficulties and problems with current rule learning evaluations. We assume the reader to be familiar with first-order logic (FOL).

2.1 Preliminaries

We consider standard datalog *rules* of the form

$$\alpha_0 \text{ :- } \alpha_1, \dots, \alpha_m. \tag{1}$$

of *length* $m \geq 1$ where all *atoms* α_j, $0 \leq j \leq m$, are of the form $p(t_1, \dots, t_n)$ with a predicate p of arity $n \geq 1$ and terms t_k, $1 \leq k \leq n$. A *term* is either a constant or a variable. α_0 is called the *head* and the conjunction $\alpha_1, \dots, \alpha_m$ the *body* of the rule. All variables that occur in the head must occur in the body. A *fact* is an atom that does not contain variables.

Note that several classical ILP systems also consider more complex function-free Horn rules, which allow for existential quantification in the rule head or negation in the body, but most recent systems focus on datalog rules or restrictions of those [Galárraga *et al.*, 2015; Evans and Grefenstette, 2018; Rocktäschel and Riedel, 2017]. In particular, reasoning systems for knowledge graphs [Yang *et al.*, 2017; Omran *et al.*, 2018] often consider only binary predicates and *chain rules* of the form $p_0(X_1, X_{m+1}) : -p_1(X_1, X_2), \dots, p_m(X_m, X_{m+1})$.

We define the problem of rule learning in the most general way: Given a *target predicate* p and background knowledge in the form of facts, including facts on p, called *positive examples*, the goal is to learn rules that can be used to infer the positive examples from the background knowledge, based on standard FOL semantics.

2.2 On the Evaluation

Veronika:
- maybe extend this section?
- eg, for existing datasets list dataset category, size, and other (what?) interesting parameters - if we are lucky this shows that the existing data is not variable/exhaustive. maybe also check the facts, for kinds of noise they contain

Together with the learning approaches the evaluation of rule learning has changed over time. While the classical ILP approaches often focused on tricky problems in complex domains [ilp, ; Quinlan, 1990] and proved to be effective in practical applications [**?**], current evaluations can be divided into two categories. Some consider very small example problems with usually less than 50 facts and only few rules to be learned [Evans and Grefenstette, 2018; Dong *et al.*, 2019]. Often, these problems are completely defined, in the sense that all facts are classified as either true or false, or that there are at least some negative examples given. Hence, the systems can be thoroughly evaluated based on classical measures such as accuracy. The others regard (subsets of) real KGs such as Wikidata[1] or DBpedia[2], recently also really large ones with millions of facts [Galárraga *et al.*, 2015; Omran *et al.*, 2018; Ho *et al.*, 2018]. Since there are no rules over these KGs, the rule suggestions of the systems are usually evaluated using metrics capturing the precision and coverage of rules [Galárraga *et al.*, 2015] based on the facts in the KG. However, since the KGs are generally incomplete, the quality of the rule suggestions is not fully captured in this way. For instance, [Omran *et al.*, 2018] present an illustrative example rule, $\text{gender}(X, \text{male}) \text{ :- } \text{isCEO}(X, Y), \text{isCompany}(Y)$, which might well capture the facts in many existing KGs but which is heavily biased and does not extend to the entirety of valid facts beyond them. Furthermore, we cannot assume that the few KGs considered completely capture the variety of existing domains and especially the rules in them. For example, [Minervini *et al.*, 2018] propose rules over WordNet[3] that are of very simple nature – together, they contain only a small number of the predicates used in WordNet and all have only a single body atom – and very different from the ones suggested in [Galárraga *et al.*, 2015] for other KGs.

In summary, we claim that the existing rather small number of exemplary datasets is not sufficient to cover the rules that could be mined from arbitrary data and the possible variety of that data. We therefore propose to complement the existing evaluations based on real KGs by considering synthetic, automatically generated datasets that may model different practical scenarios by varying, for example, in the number and kinds of symbols considered, in the structure of rules (e.g., only simple rules such as in WordNet, or only chain rules), and in the ways the rules depend on each other (i.e., the head

[1]https://www.wikidata.org/wiki/Wikidata:Main_Page

[2]https://wiki.dbpedia.org/

[3]https://wordnet.princeton.edu/

of one rule can be part of the body of one or multiple other rules).

Veronika: I am not sure if I missed definitions we need in the next two sections. need to go over it at some point again

3 The Generated Datasets

We have generated several example datasets that reflect different possible scenarios. Specifically, they vary in the sorts and quantity of facts and in the structures and sizes of the rule sets (example rule sets are depicted in Figure 1). Each dataset consists of two files, one for the facts and one for the rules, both come in standard Prolog format. The datasets described in this section were generated under the open-world assumption. Specifically, we configured the generator such that they contain only 70% of the consequences that can be inferred from a basic set of so-called *support facts* (this is detailed in Section 4).

Rules and facts. Our datasets are domain independent, which means that we consider synthetic names p_i for predicates, c_i for constants, and X_i for variables with $i \geq 0$. We generate datalog *rules* as introduced in Section 2. For practicality, we require the head to contain some variable, and set both the maximum rule length and arity of atoms to two (these are parameters configurable in the generator).

We divide rule sets into three different categories depending on the dependencies between rules: *Chain*, *Rooted DG*, and *Disjunctive Rooted DG*; and generated complete datasets for each. Figure 1 shows the generated rule sets. The dependencies between the rules are represented as edges in a directed graph (DG) where the rules are the nodes. That is, an incoming edge at a node shows that the facts inferred by this rule might be relevant for inference with the rule at the connected node. The node at the top is called the *root*. In the following, we use (rule) graph and DG interchangeably.

Category Chain. Each rule apart from the one at the root infers facts relevant for exactly one other rule (i.e., every node has at most one parent node) and, for each rule, there is at most one such other rule which might infer facts relevant for the rule (i.e., every node has at most one child node). However, recursive rules represent an exception, they are relevant for themselves and for one other rule (i.e., the graph has a small loop at each node representing a recursive rule).

Category Rooted DG. It generalizes category Chain in that every rule can be relevant for several others (i.e., every node has at most one parent node). Furthermore, for each rule, there may be several other rules which might infer facts relevant for the rule (i.e., a node may have several child nodes). However, for each predicate occurring in the body of the former rule, there must be at most one other rule with this predicate in the head; that is, there are no alternative rules to derive facts relevant for a rule w.r.t. a specific body atom.

Category Disjunctive Rooted DG. It generalizes category Rooted DG in that we do not have the restriction regarding the latter alternatives.

Figure 1 illustrates the differences between the categories. In (a), for every rule, there is at most one child node with a rule relevant for its derivations. In (b), there might be multiple, but the child nodes contain different predicates in their heads. In (c), the latter does not hold anymore. That is, for given facts, there may be alternative derivations leading to positive examples.

Table 1 shows statistics about the datasets we generated. Observe that, in addition to the datasets whose rules are shown in Figure 1, we present one larger dataset for each category. The dataset size describes the dimension of the fact sets. All the datasets were generated such that they are missing 30% of all consequences, 20% of the original support facts, and contain 20% facts that are irrelevant for the derivation of positive examples. We hence simulated an open-world setting and considered noise. Observe that, in terms of size, these datasets are comparable to existing most ones such as .... We also generated corresponding datasets where all rule graphs have depth two; and we have larger datasets, but the results ... (see Section 6).

Veronika: complete above explanation, mention pred nbrs, comment on difficulty - first want to show that even simple rules learned in other papers in combination tricky see c, mention var indexes, or node

4 Dataset Generation

RuDa contains an easy-to-use generator of ILP datasets as presented in Section 3. It is written in Python. In this section, we describe the generation of the rules and facts in detail.

Configuration. RuDa is parameterizable in many dimensions, most important are the following:

- maximal number of predicates and constants
- maximal arity of predicates
- dataset size: XS, S, M, L, XL
- open-world degree n_{OW} (percentage)
- amount of noise in the data n_{Noise} (percentage)
- minimal and maximal number of DGs in the rule set
- category of DGs: Chain, R-DG, DR-DG, Mixed
- number and maximal length of rules
- maximal depth of rule graphs

Observe that the latter is a bound for the number of inference steps, and that the number of available predicates and constants influences the variability and number of generated facts, rules, and noise. An XS dataset contains about 50-100 facts, an S dataset about 101-1,000, an M dataset about 1,001-10,000, an L dataset about 10,001-100,000, and an XL dataset about 100,001-500,000. Note that RuDa can easily be extended to allow for new sizes like XXL, etc. The open-world degree specifies how many of the consequences from a basic set of support facts are missing in the dataset, and the amount of noise determines how many support facts are missing and how many facts irrelevant for deriving the target facts are contained in the dataset. By number of DGs in a rule set, we mean the number of connected components in the graph of a rule set, where the rules are considered as nodes in a graph and the edges represent the dependencies

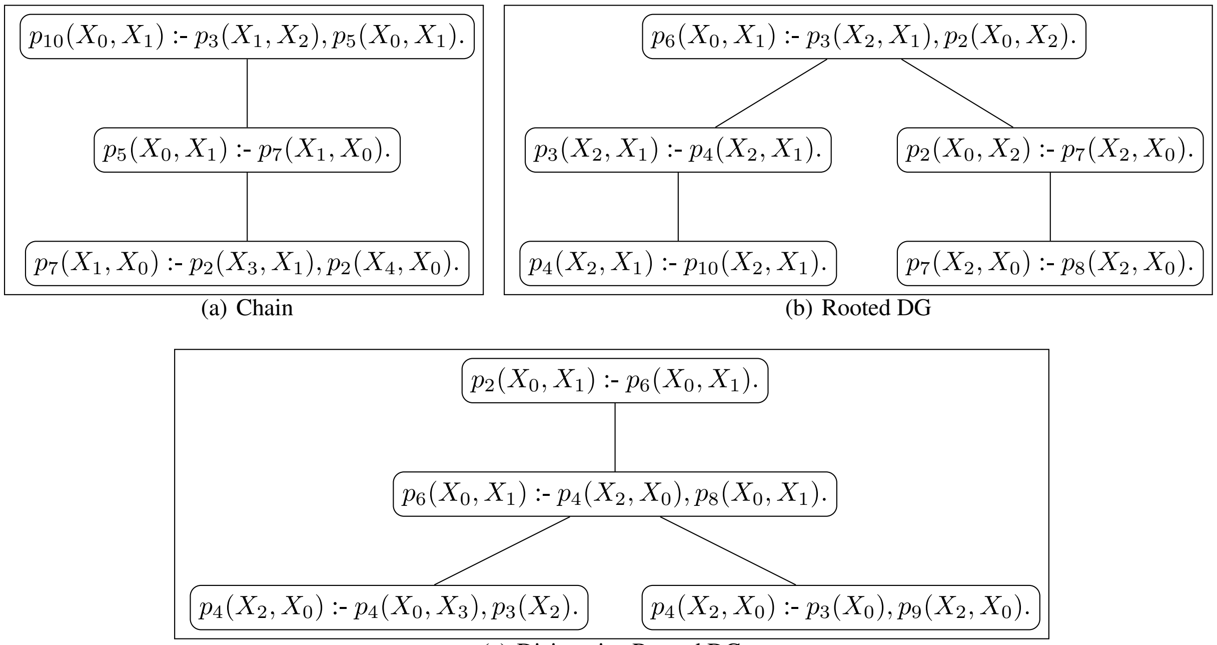
$p_{10}(X_0, X_1)$:- $p_3(X_1, X_2), p_5(X_0, X_1)$.
$p_5(X_0, X_1)$:- $p_7(X_1, X_0)$.
$p_7(X_1, X_0)$:- $p_2(X_3, X_1), p_2(X_4, X_0)$.
(a) Chain
$p_6(X_0, X_1)$:- $p_3(X_2, X_1), p_2(X_0, X_2)$.
$p_3(X_2, X_1)$:- $p_4(X_2, X_1)$.
$p_2(X_0, X_2)$:- $p_7(X_2, X_0)$.
$p_4(X_2, X_1)$:- $p_{10}(X_2, X_1)$.
$p_7(X_2, X_0)$:- $p_8(X_2, X_0)$.
(b) Rooted DG
$p_2(X_0, X_1)$:- $p_6(X_0, X_1)$.
$p_6(X_0, X_1)$:- $p_4(X_2, X_0), p_8(X_0, X_1)$.
$p_4(X_2, X_0)$:- $p_4(X_0, X_3), p_3(X_2)$.
$p_4(X_2, X_0)$:- $p_3(X_0), p_9(X_2, X_0)$.
(c) Disjunctive Rooted DG

Figure 1: Example rule sets generated for the different categories (the rules of datasets (a) CHAIN-S, (b) RDG-S, (c) DRDG-XS).

Name	Size	Category	#Facts	#Rules	#Predicates	#Constants
CHAIN-XS	tiny	Chain	64	3	9	37
RDG-XS	tiny	R-DG	93	4	8	42
DRDG-XS	tiny	DR-DG	76	4	11	34
CHAIN-S	small	Chain	572	3	11	205
RDG-S	small	R-DG	375	5	11	215
DRDG-S	small	DR-DG	945	6	11	406

Table 1: Overview of our generated datasets; all rule graphs have depth three.

between them (see Figure 1). Category mixed specifies that these connected components may be of different categories. The implementation of a given configuration is described in detail in the following.

Preprocessing. The number of DGs that is being generated and their depths are determined randomly; though, we ensure that at least one graph is of the given maximal depth. Note that all random selections we mention are within the bounds given in the configuration under consideration. Also the arities of the predicates are selected at arbitrarily. Predicates are named $p_0, p_1, \ldots$ and constants $c_0, c_1, \ldots$.

Rule generation. According to the rule set category specified and graph depths determined, rules (nodes in the graph) of form (1) are generated top down breadth first, for each of the rule graphs to be constructed. The generation is largely at random, that is, w.r.t. the number of child nodes of a node and which body atom they relate to, respectively; the number of atoms in a rule; and the predicates within the latter, including the choice of the target predicate in the very first step. Though, RuDa also offers the option that all graphs have the same target predicate. In order to allow for more derivations, we currently only consider variables as terms in head atoms; the choice of the remaining terms is based on probabilities as described in the following. Given the atoms to be considered (in terms of their number and predicates) and an arbitrary choice of head variables, we first determine a position for each of the latter in the former. Then we populate the other positions one after the other: a head variable is chosen with probability 0.2 ($[1/5]$); for one of the variables introduced so far, we have probability 0.6 ($[4/5] * [3/4]$); for a constant, 0.02 ($[4/5] * [1/4] * [1/10]$); and, for a fresh variable, 0.18 ($[4/5] * [1/4] * [9/10]$); note that the probabilities can be changed easily.

Fact generation. The fact generation is done in three phases: we first construct a set $\mathbb{D}$ of relevant facts in a closed-world setting, consisting of support facts $\mathbb{S}$ and their consequences $\mathbb{C}$, and then adapt it according to n_{OW} and n_{Noise}.

The (natural) idea is to generate facts by instantiating the rule graphs multiple times, based on the assumption that ILP systems need positive examples for a rule to learn that rule, and to stop the generation when the requested number of facts has been generated. Though, we actually stop later because we need to account for the fact that we subsequently will delete some of them according to n_{OW}. More specifically, we continuously iterate over all rule graphs, for each, select an arbitrary but fresh variable assignment σ, and then iterate over the graph nodes as described in the following, in a bottom-up way. First, we consider each leaf n and corresponding rule of form (1) and generate support facts $\sigma(\alpha_1), \ldots, \sigma(\alpha_m)$. Then, we infer the consequences based on the rules and all facts generated so far. For every node n on the next level and corresponding rule of form (1), we only generate those of the facts $\sigma(\alpha_1), \ldots, \sigma(\alpha_m)$ as support facts which are not among the consequences inferred previously. We then again apply inference, possibly obtaining new consequences, and continue iterating over all nodes in the graph in this way. We further diversify the process based on two integer parameters, n_{DG} and n_{Skip}: in every n_{DG}-th iteration the graph is instantiated exactly in the way described; in the other iterations, we skip the instantiation of a node with probability $1/n_{\text{Skip}}$ and, in the case of DR-DGs, only instantiate a single branch below disjunctive nodes.

In the open-world setting, we subsequently construct a set $\mathbb{D}_{\text{OW}}$ by randomly deleting consequences from $\mathbb{D}$ according to the open-world degree given: assuming $\mathbb{T} \subseteq \mathbb{C}$ to be the set of target facts, we remove $n_{\text{OW}}\%$ from $\mathbb{C} \setminus \mathbb{T}$, and similarly $n_{\text{OW}}\%$ from $\mathbb{T}$. In this way, we ensure that the open-world degree is reflected in the target facts. Though, note that there is the option to have it more arbitrary by removing $n_{\text{OW}}\%$ from $\mathbb{C}$ instead of splitting the deletion into two parts.

The noise generation is split similarly and focuses on two kinds of noise, missing and irrelevant facts. Specifically, we construct a set $\mathbb{D}_{\text{OW+Noise}}$ based on $\mathbb{D}_{\text{OW}}$ by arbitrarily removing $n_{\text{Noise}}\%$ from $\mathbb{S}$, and by adding arbitrary fresh facts that are neither in $\mathbb{C}$ (i.e., we do not add facts which we have removed in the previous step) nor contain the target predicate such that $\mathbb{D}_{\text{OW+Noise}} \setminus \mathbb{T}$ contains $n_{\text{Noise}}\%$ of noise. In addition, we add arbitrary fresh facts containing the target predicate and that are neither in $\mathbb{T}$ such that the set of facts within $\mathbb{D}_{\text{OW+Noise}}$ on that predicate finally contains $n_{\text{Noise}}\%$ of noise.

Output. The dataset generation produces the following:

- The rules and a *training* set ($\mathbb{D}_{\text{OW+Noise}}$) which, is of the requested size, and fulfills n_{OW} and n_{Noise}.
- A *validation* and a *test* set as required by most of the current systems containing the consequences which were removed from the original, closed-world version $\mathbb{D}$ using a split of 1:2.
- A custom *evaluation* set $\mathbb{S}'$ for our evaluation tools that is generated in the same way as $\mathbb{S}$.

  Veronika: still open if we need it for our evaluation, if not remove here

For further experiments, RuDa actually also writes out $\mathbb{D}$, an adaptation of that set which contains noise (but all of $\mathbb{C}$), $\mathbb{D}_{\text{OW}}$, $\mathbb{S}$, $\mathbb{C}$, and files containing the predicates and constants occurring in $\mathbb{D}$.

5 Evaluation Tools

Veronika: introduce preliminaries specific for measures here. they are not needed if you just read the paper to learn about the datasets...

Evaluation tool to compute distances between logic programs: There are 2 (or more) types of distances defined on logic programs

- distance between Herbrand interpretations
- directly on rules (logic programs)

we don't count auxiliary pred

the distances can be considered "fuzzy" in the sense that we consider facts with weights

Veronika: I think we should also compute measures they considered so far - like Hits@... AUC , ... – and the discuss how the results vary

6 Experiments

We ran experiments with representatives of the different types of rule learning systems using the datasets described in Section 3. In this section, we report and discuss the results.

6.1 Systems

We evaluated the following systems.

FOIL. [Quinlan, 1990] A traditional ILP system whose rule construction is completely driven by positive and negative examples in that the rule bodies are iteratively refined (i.e., atoms are added) until none of the negative examples can be derived by the rule anymore – though exceptions are possible. The process is guided by custom heuristics based on the examples.

AMIE+. [Galárraga *et al.*, 2015] A system mining rules systematically by considering rules with a single body atom and then iteratively refining them based on custom confidence and coverage measures. For medium-sized KBs, it has been shown to be several orders of magnitude faster than other state-of-the-art systems.

Neural LP. [Yang *et al.*, 2017] One of the first end-to-end differentiable approaches.

Veronika: one sentence about processing (matrix multiplication?). I do not fully understand it, find the description somehow vague "based on tensorlog"..

Neural LP is special in that it learns a distribution over logical rules, instead of an approximation to a particular rule, such that its inference is based on rules that have these confidences attached as weights.

NTP. [Rocktäschel and Riedel, 2017] Similarly, a recent neural approach that recursively constructs a network which derives the training examples through backward chaining. It is based on learning vector embeddings of symbols and represents symbolic unification by computationally combining them.

Veronika: neural lp:is 'one of the first' true?

Veronika: FOIL: I only coarsely read the paper and did not find the point where it describes which head predicates are considered for rule construction. all possible ones? it says it starts with the target predicate

Name	Size	Category	#Facts	#Rules	#Predicates	#Constants
CHAIN-XS	tiny	Chain	76	2	9	51
RDG-XS	tiny	R-DG	63	3	9	35
DRDG-XS	tiny	DR-DG	97	3	11	67
CHAIN-S	small	Chain	677	2	11	287
RDG-S	small	R-DG	634	3	11	379
DRDG-S	small	DR-DG	790	3	11	301

Table 2: Overview of generated datasets with rule graphs of depth two.

	EVEN	S-2-1 N	S-2-2 N	XS-1 N	S-2-1 E	S-2-2 E	XS-1 E
FOIL	**1.0**	0.0052 [**1.0**]	0.0	0.0	0.575 [**0.982**]	0.0333 [**1.0**]	**0.816 [1.0]**
Amie	-	**0.955**	**1.0**	0.7029	**0.955**	**0.714**	0.7029
Neural LP	-	0.0	0.0	0.234	0.0	0.0	0.7029
NTP	**1.0**	0.382	0.03	**1.0**	0.382	0.031	0.0

Table 3: Herbrand accuracy for simple datasets (CHAIN).
Standard confidence is reported in parenthesis when different from Herbrand accuracy.

	CHAIN-S-2	CHAIN-S-2-0	CHAIN-S-3	CHAIN-XS-2
FOIL	0.608 [**0.954**]	0.354 [0.784]	0.654 [**0.958**]	0.329 [0.350]
Amie	**0.6262**	**0.6981**	**0.8860**	**0.8667**
Neural LP	0.0021 [0.0022]	0.0	0.0312 [0.0318]	0.0533
NTP	0.0	0.0064	0.0225 [0.0256]	0.1610 [0.1755]

Table 4: Herbrand accuracy for binary datasets (CHAIN)
Standard confidence is reported in parentesis when different from Herbrand accuracy.

	DRDG-S-2	DRDG-S-3	DRDG-XS-2	RDG-S-2	RDG-S-3	RDG-XS-2
FOIL	0.193 [0.783]	0.0	**0.703 [0.734]**	0.0	0.515 [**0.911**]	**0.496 [0.624]**
Amie		**0.4898**	0.0064	**0.7137**		0.2208
Neural LP	0.0080 [0.0083]	0.0816 [0.0857]	0.0649 [0.0671]	0.0741 [0.0831]	0.0482 [0.07958]	0.0029 [0.0031]
NTP		0.1801 [0.2068]	0.1236	0.1429	0.0	0.0073

Table 5: Herbrand accuracy for binary datasets (DRDG and RDG)
Standard confidence is reported in parentesis when different from Herbrand accuracy.

6.2 Implementation

We ran all systems on the same machine, a ...

Veronika: mention specific configuration params we use with below links?

FOIL[4] (version 5) AMIE+[5] (version of 2015-08-26) And code of following (current date)... Neural LP[6] NTP[7] The scripts for running the evaluation are available in our GitHub repository.

Veronika: mention negative examples given to FOIL and how we handle validation facts for ntp... and in general

Since Neural LP and AMIE+ only support binary atoms, we represent unary atoms of forms $p(X)$ and $p(c)$ by $p(X, X)$ and $p(c, c)$, respectively, as it is usual [Evans and Grefenstette, 2018].

NTP only learns rules whose general format corresponds to so-called *templates* it is given as input. They are abstracted rules, such as $\#1(X, Y)$:- $\#2(X, Z), \#2(Z, Y)$, which specify the number and variables of atoms and which predicates are shared among them (#2 in the example). In order to not restrict ntp, we provide it with all templates that are necessary to learn the rules in our datasets.

6.3 Results

6.4 Discussion

Veronika: add general discussion

Note that FOIL generally learns rules as introduced in Section 2, but all other systems consider only a restricted subset.

Veronika: the 2 of us need to discuss if this is true wrt amie and ntp. do they really NOT consider existential quantification in rule heads? I think they do not.

As a consequence, we can directly determine which of our rules cannot be (fully) detected by specific systems at all.

[4]http://www.cs.cmu.edu/afs/cs/project/ai-repository-9/ai/util/areas/learning/systems/foil/foil5/0.html

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

[6]https://github.com/fanyangxyz/Neural-LP

[7]https://github.com/uclmr/ntp

Consider, for example, the rules in Figure 1. AMIE+ does not consider reflexive rules and requires rules to be connected and closed. The latter condition yields that it can find only one rule in CHAIN-S and only two rules in DRDG-XS; in RDG-S, all rules are connected and closed.

Veronika: do the results confirm this? mention!

Veronika: mention somewhere else (where it fits) that this nicely shows the variability of our datasets

Neural LP only supports chain rules (without constants),[8] which do not occur in CHAIN-S, only twice in DRDG-XS, but four times in RDG-S.

Veronika: I have trouble with the chain rule example in the neural lp paper:
query(Y,X):-Rn(Y,Zn),...,R1(Z1,X)
shouldn't it be
query(Y,X):-Rn(Y,Zn-1),...,R1(Z1,X)?
so that we really get a chain as
query(Y,X):-Rn(Y,Z3),R3(Z3,Z2),R2(Z2,Z1),R1(Z1,X)?
where $n \neq 3$!

Nevertheless, rules that are not fully supported can still be recognized partially. For instance, Neural LP learns chain rules $p_{10}(X_0, X_1)$:- $p_5(X_0, X_1)$ and $p_6(X_0, X_1)$:- $p_8(X_0, X_1)$ for datasets CHAIN-S and DRDG-XS, respectivly; though, their confidences are not the highest but rather average among the rules learned.

Recall that we provide all necessary templates for NTP, thus it is not restricted in this regard.

Veronika: do the result measures confirm this?

7 Conclusions

In this paper, we have presented RuDa, a system for generating datasets for rule learning and for evaluating corresponding systems, in order to complement the existing evaluations, which are based on real KGs but missing rules. Our experiments on new, generated datasets have shown that

Veronika: .

There are various directions for future work. The dataset generation can be extended to more complex rules by including, for example, negation, existential quantification in rule heads, and functions. Probabilistic rules as learned in [Manhaeve *et al.*, 2018], for example, are also interesting. In the evaluation, we want to consider other measures, such as

Cristina: . . .

Our generated datasets can also be used – and we plan to do so – to develop neural rule learning approaches of a new kind, which are trained on several KGs and rules over them in order to suggest rules for previously unseen KGs later. To the best of our knowledge, this direction has been unexplored to date, which means that the full potential of DL for learning rules has probably not been exploited yet.

[8]The rule specification in [Yang *et al.*, 2017] is slightly ambiguous w.r.t. the predicate and variable names but the system's output shows that it learns chain rules as they are defined in Section 2.

Cristina: TODO: future work
- probabilistic reasoner
- more powerful logic (Full first order)

References

[Campero *et al.*, 2018] Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, and Sebastian Riedel. Logical rule induction and theory learning using neural theorem proving. *CoRR*, abs/1809.02193, 2018.

[Dong *et al.*, 2019] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *International Conference on Learning Representations*, 2019.

[Evans and Grefenstette, 2018] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

[Fürnkranz *et al.*, 2012] Johannes Fürnkranz, Dragan Gamberger, and Nada Lavrac. *Foundations of Rule Learning*. Cognitive Technologies. Springer, 2012.

[Galárraga *et al.*, 2015] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6):707–730, 2015.

[Ho *et al.*, 2018] Vinh Thinh Ho, Daria Stepanova, Mohamed H. Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. Rule learning from knowledge graphs guided by embedding models. In *The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Proceedings, Part I*, pages 72–90, 2018.

[ilp,] ILP Applications and Datasets. https://www.doc.ic.ac.uk/~shm/applications.html. Accessed: 2019-01-30.

[Manhaeve *et al.*, 2018] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018.*, pages 3753–3763, 2018.

[Minervini *et al.*, 2018] Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, and Sebastian Riedel. Towards neural theorem proving at scale. *In Neural Abstract Machines Program Induction v2, NAMPI*, 2018.

[Muggleton, 1991] Stephen Muggleton. Inductive logic programming. *New Generation Comput.*, 8(4):295–318, 1991.

[Muggleton, 1995] Stephen Muggleton. Inverse entailment and progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[Omran *et al.*, 2018] Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. Scalable rule learning via learning representation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2149–2155, 2018.

[Quinlan, 1990] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Raedt, 2008] Luc De Raedt. *Logical and relational learning*. Cognitive Technologies. Springer, 2008.

[Rocktäschel and Riedel, 2017] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 3791–3803, 2017.

[Stepanova *et al.*, 2018] Daria Stepanova, Mohamed H. Gad-Elrab, and Vinh Thinh Ho. Rule induction and reasoning over knowledge graphs. In *Reasoning Web. Learning, Uncertainty, Streaming, and Scalability - 14th International Summer School 2018, Tutorial Lectures*, pages 142–172, 2018.

[Wang and Li, 2015] Zhichun Wang and Juan-Zi Li. Rdf2rules: Learning rules from RDF knowledge bases by mining frequent predicate cycles. *CoRR*, abs/1512.07734, 2015.

[Yang *et al.*, 2017] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 2316–2325, 2017.

Veronika: change style in example predicate and constant macro

Veronika: remove option hypersetup draft in the end

Veronika: we still can shorten the bib by replacing the proceeding names with abbreviations: In NIPS 17, ... and maybe also make the rule graph pic more compact (but I don't know how to do that) - I already deleted or nodes