Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)
Demonstrations Track

# PyClause - Simple and Efficient Rule Handling for Knowledge Graphs

**Patrick Betz**[1] , **Luis Galárraga**[2] , **Simon Ott**[3] ,
**Christian Meilicke**[1] , **Fabian Suchanek**[4] and **Heiner Stuckenschmidt**[1]
[1]University of Mannheim
[2]INRIA
[3]Austrian Institute of Technology GmbH
[4]Telecom Paris, Institut Polytechnique de Paris
{patrick, christian, heiner}@informatik.uni-mannheim.de, luis.galarraga@inria.fr

## Abstract

Rule mining finds patterns in structured data such as knowledge graphs. Rules can predict facts, help correct errors, and yield explainable insights about the data. However, existing rule mining implementations focus exclusively on mining rules – and not on their application. The PyClause library offers a rich toolkit for the application of the mined rules: from explaining facts to predicting links, scoring rules, and deducing query results. The library is easy to use and can handle substantial data loads.

## 1 Introduction

Knowledge graphs (KGs) such as Yago [Suchanek *et al.*, 2007] or Wikidata [Vrandečić and Krötzsch, 2014] represent knowledge about the world via structured facts of the form `predicate(subject, object)` [Suchanek *et al.*, 2019]. For example, a general-purpose KG could contain the fact `livesIn(Eleonor Roosevelt, NewYork)`. Rule mining approaches learn symbolic rules of the form $H \Leftarrow \boldsymbol{B}$ with body $\boldsymbol{B}$ and head $H$ from a KG. In our example, we could mine:

```
citizen(X,Y) ⇐ livesIn(X,A), locatedIn(A,Y)
```

where $X$, $A$, and $Y$ are variables and the terms $r(\cdot,\cdot)$ are called atoms. This rule says that a person $X$, who lives in a city $A$ located in a country $Y$ is most likely a citizen of that country. Such rules serve several purposes: they can be used to explain facts (why is Roosevelt likely to be a citizen of the United States? Because she lived in New York!); to predict facts (if Roosevelt's nationality is unknown, it can be guessed); and to correct mistakes (if Roosevelt has another nationality than American, this statement should be checked by a human). Rules can also help understand the patterns in the KG (compute every rule that predicts Roosevelt to be a citizen of the United States). In domain-specific settings, rules can be used for recommendation, e.g., to guide research efforts. A domain-specific KG such as Hetionet [Himmelstein *et al.*, 2017], for example, contains biomedical facts such as `treats(Ibuprofen, Headache)`. On such a KG, we could mine that similar compounds should be considered as treatments for the same medical conditions:

```
treats(X,Y)  <= resembles(A,X), treats(A,Y)
```

Such a rule can be used, e.g., in the context of a *drug repurposing* application [Liu *et al.*, 2021], in which new treatment possibilities are searched for given compounds. Finally, rules can also be used in combination with embedding-based prediction methods to endow them with explainability [Boschin *et al.*, 2022; Betz *et al.*, 2022a; Galárraga, 2023].

Rule mining thus has useful applications in explainable data interpretation. And indeed, there exist several approaches that can mine rules, including AMIE [Galárraga *et al.*, 2013; Galárraga *et al.*, 2015; Lajus *et al.*, 2020], AnyBURL [Meilicke *et al.*, 2023], Rudik [Ortona *et al.*, 2018], and differentiable learning methods [Yang *et al.*, 2017; Sadeghian *et al.*, 2019; Qu *et al.*, 2021]. These approaches can mine rules on large real-world KGs in minutes (the above rules were mined on YAGO by AMIE, and by AnyBURL on Hetionet, respectively). While the mining can be performed efficiently, the functionality to actually *apply* the rules is hardly accessible. It is usually not possible to use the rules to predict facts, explain facts, or compute the precision of these rules on another KG. Rather, the user has to invest additional coding effort.

In this paper, we present PyClause, a library that makes general rule application easily accessible in Python. Applications using the rules can be incorporated into existing Python projects with a few lines of code. PyClause can flexibly switch between different input and output modes, e.g., from Numpy arrays containing integers to lists of strings. Computations can be performed by user defined options and all implementation details are documented in a default configuration file. PyClause supports multithreading for all the implemented features and its core mechanics are implemented in C++ for maintaining runtime efficiency. PyClause, as well as all data of our examples is publicly available at https://github.com/symbolic-kg/PyClause.

## 2 Rule Handling with PyClause

The PyClause library is structured in two Python packages `clause` and `c_clause`. Figure 1 shows the library components when using different features. The `c_clause` package is written in C++, and implements most of the library core features such as data storing and rule grounding. It is accessible from within Python via a high-level API that is structured

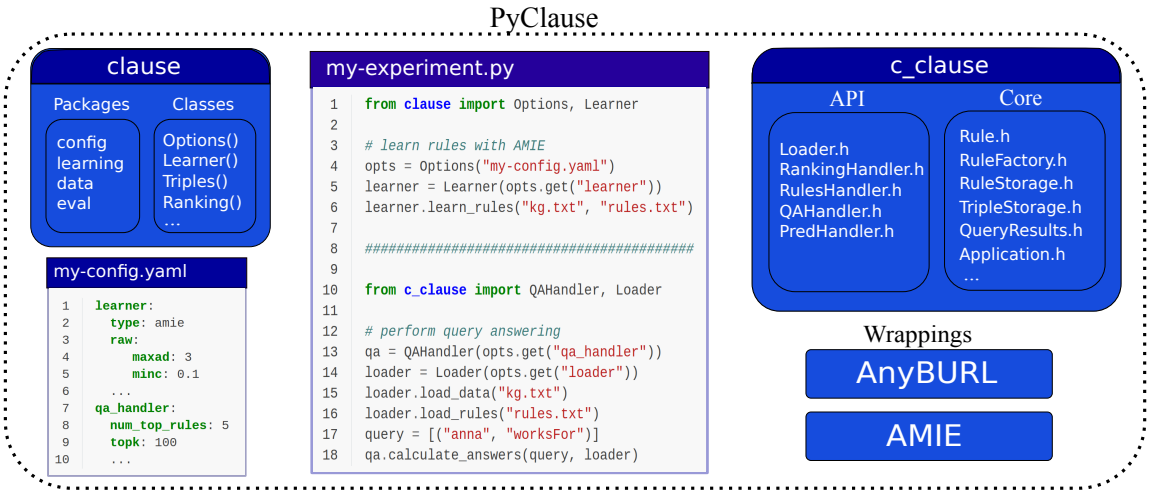


Figure 1: Usage-centric library overview. The example code mines rules with AMIE, which are used to predict the employer of a person.

into multiple handler classes providing access to different features. The `clause` package is pure Python and provides different utilities such as option handling and rule learning.

PyClause supports horn rules without negation on binary predicates given by the respective KG. In the following, we briefly define some terminology that is used throughout the descriptions of this section. Given a rule $H \Leftarrow \boldsymbol{B}$, a substitution $\sigma$ is a mapping from variables in the rules to constants. Applying a substitution to the atoms of the rule produces grounded atoms, i.e., facts. If the body groundings are facts from the KG, the head grounding is a prediction. If this prediction is also in the KG, we call it a true prediction. The support of a rule is the number of true predictions made by the rule. If we normalize this number by all the predictions of the rule, we obtain the standard confidence. Depending on what we consider a valid prediction, we can obtain different confidence metrics [Galárraga *et al.*, 2013; Tanon *et al.*, 2017].

## 2.1 Features

PyClause provides convenience wrappers to two rule learners, AMIE and AnyBURL. Both systems can be run from Python, and can be fully configured by the PyClause option handling. The AMIE system was especially adapted to conform to a common interface. Figure 1 (center top) shows how a rule mining system can be called easily from within PyClause. In what follows, we assume that the rules have been mined (by AMIE, AnyBURL, or any other system), and focus on the applications of the rules.

A typical workflow can be seen in Figure 1 (center bottom): we first load a KG and a rule set into memory by help of the `c_clause.Loader`. It is given as a reference to a particular handler class, that additionally takes feature specific input (e.g., facts) to perform its task. Then, PyClause supports the following operations on the KG and set of rules.

**PREDICT.** Three different prediction modes are provided. 1) Given a KG and one or more input queries, i.e., facts with a missing object or subject, find candidates for the missing entity and provide a likelihood score for each candidate. 2) Given a set of complete facts, calculate a likelihood score for each fact, which will be zero if it is not predicted by any rule. Otherwise the likelihood scores are computed by aggregating the confidences of the predicting rules. Different aggregation functions can be configured in the options, see, e.g., [Betz *et al.*, 2023]. 3) Given an additional target KG we aim to complete, form a head and a tail query from every existing fact and calculate candidates for all created queries. This results in a complete ranking in the realm of knowledge graph completion [Rossi *et al.*, 2021] (KGC). PyClause also provides utilities to directly calculate fine-grained ranking-based evaluation metrics.

**EXPLAIN.** For the prediction modes described above, the library can additionally compute and output all the rules that produced the respective candidate or fact predictions. These constitute the reasons why a prediction was made and they could, e.g., be used as features for neuro-symbolic models. Finally, for the given input facts and their predicting rules, all the body groundings of each rule can be returned. For example, an explanation for the prediction `citizen(Tom Hanks, US)` can be given by the first rule of Section 1 together with its body groundings {`livesIn(Tom Hanks, LA)`, `locatedIn(LA, US)`}.

**SCORE.** Given a set of rules (whether mined or defined by the user), PyClause can compute different metrics such as the number of predictions, the support, the rule confidence or precision, on any specified KG.

**MATERIALIZE.** Given a set of rules (whether mined or defined by the user), PyClause can find all predictions of the rule, and materialize them into a new KG. This is useful to compute the deductive closure of the KG under the rules, and can be used to add new facts.

For all of these functionalities, PyClause supports different types of data representation. Inputs and outputs can be read or written from and to files or they can be directly given and accessed from code. Furthermore, they can be either represented as Numpy arrays containing indices or as lists of strings. The full list of PyClause operators and their descrip-

tion as well as further details such as the supported rule types (the language bias) are provided in the library documentation.

### 2.2 Efficiency

PyClause maintains the runtime efficiency of symbolic approaches, even though it is used from Python. In the following experiments, we focus on different application tasks, which cover all computationally relevant aspects of the library. We use the FB15k-237 [Toutanova and Chen, 2015] and the WN18RR [Dettmers *et al.*, 2018] KG's and their common training and test splits, which are frequently used in the KGC community (e.g., [Broscheit *et al.*, 2020]). For FB15k-237 (WN18RR), the training split on which rules are learned and applied, contains 272k (86k) triples. The test splits contain 20k (3k) triples. The rule sets are taken from [Meilicke *et al.*, 2021] and they contain more than 5 million (97k) rules for FB15k-237 (WN18RR).

The experiments are performed on an Intel(R) Xeon(R) Silver CPU. When possible, we compare against AnyBURL as it is recently shown to be runtime efficient compared to neural models [Meilicke *et al.*, 2023]. We measure the pure computation times and exclude rule loading, which only has to be performed once in PyClause and is similar for both methods, e.g., roughly 25 (0.5) seconds for FB15k-237 (WN18RR).

Table 1 shows results for different features. In the first two rows, we use the default options for PyClause and AnyBURL and create a ranking file for the FB15k-237 test split when applying the rules on the training split (predict mode 3 from Section 2.1). In the next two rows, we re-calculate the preprocessing used in [Ott *et al.*, 2023] on the training splits of the respective KG's. This means calculating a ranking file for the training split itself, and storing all the rules that predicted every computed candidate[1]. The results show the runtime benefit when using PyClause. Finally, we show in the last row the runtimes when assigning triple scores given by the predicting rule with the highest confidence of the test splits of the KGs.

| Task | System | FB15k-237 | WN18RR |
|---|---|---|---|
| Ranking (6 threads) | PyClause | 47.3s | 6.3s |
| | AnyBURL | 155s | 36s |
| Rule Retrieval (16 threads) | PyClause | 147s | 50s |
| | AnyBURL | 3.4h | 819s |
| Triple Scoring (6 thr) | PyClause | 5.4s | 0.5s |

Table 1: Rule application runtimes for different tasks in seconds (s) or hours (h). The rule set for FB15k-237 contains more than 5 million rules and 97k rules for WN18RR.

Rule application tasks on KGs do not involve full fledged reasoning. Systems like ProbLog [De Raedt *et al.*, 2007] focus on inference using model theoretic entailment on general probabilistic logic programs. This type of inference is not feasible for larger KGs. To demonstrate this, we translated the rules and the train and test splits from WN18RR into a ProbLog program to perform the task of the first row. It is one of the smallest KGs used for evaluation tasks. ProbLog did not terminate after two hours. Restricting the rules from 97k to 400 cyclical rules with one or two body atoms and reducing the amount of queries from 6000 to 20 (2 queries per fact), ProbLog did still not terminate after two hours. ProbLog did terminate in few minutes when using 32 rules, i.e, only cyclical rules with one body atom, and 20 queries.

General purpose rule-based programming systems such as CLIPS [Riley, 1991], on the other hand, can be considered as efficient alternatives. PyClause, however, is more tailored towards KGs and KGC and provides specific functionalities such as confidence aggregation functions in this context.

[1]This feature is only available in an unofficial version of AnyBURL provided in the supplementary materials of [Betz *et al.*, 2022b].

## 3 Application: Drug Repurposing

We will briefly continue the example given in Section 1 regarding the drug repurposing problem; a longer version is provided in the examples section on Github. The Hetionet KG contains more than 2 million facts and we use a rule set learned with AnyBURL on the KG. We assume that the KG and the rule set is loaded with a `Loader` and that a `QAHandler` (qa) and `PredictionHandler` (ph) are instantiated. We consider the given query `treats(Isoetarine, ?)`, i.e., we seek for ailments that can be treated with the compound `Isotarine`.

```
queries = [("Isoetarine", "treats")]
qa.calculate_answers(queries, loader, "tail")
answers = qa.get_answers(as_string=True)
answers[0][0] # return first candidate
['asthma', 0.449] #output
```

The first candidate prediction is `Asthma` with a likelihood of 0.449. We can now analyze this prediction further:

```
fact = [("Isoetarine", "treats", "asthma")]
ph.calculate_scores(fact, loader)
targets, pred_rules, groundings \
  = ph.get_explanations(as_string=True)
```

The variable `pred_rules` holds the predicting rules for the fact and `groundings` the respective body groundings. One of the predicting rules is the rule that we have already seen in Section 1. We show the output when accessing the first grounding of this rule (code suppressed for brevity):

```
['Salbutamol', 'resembles', 'Isoetarine']
['Salbutamol', 'treats', 'asthma']
```

We can thus see that one reason for the prediction `Asthma` is that the resembling drug `Salbutamol` is already known to treat `Asthma`.

## 4 Conclusion

PyClause is a library for efficiently handling association rules in the context of knowledge graphs. It provides tools for using the rules as predictors and also as explanations for factual predictions. The current implementation includes two state-of-the-art rule miners but rules from any mining system can be used. Future work can integrate rules and their predictions generated by PyClause into machine learning pipelines, including neural architectures and embedding-based models.

## References

[Betz *et al.*, 2022a] Patrick Betz, Christian Meilicke, and Heiner Stuckenschmidt. Adversarial explanations for knowledge graph embedding models. In *IJCAI*, 2022.

[Betz *et al.*, 2022b] Patrick Betz, Christian Meilicke, and Heiner Stuckenschmidt. Supervised knowledge aggregation for knowledge graph completion. In *ESWC*, 2022.

[Betz *et al.*, 2023] Patrick Betz, Stefan Lüdtke, Christian Meilicke, and Heiner Stuckenschmidt. On the aggregation of rules for knowledge graph completion. *KLR Workshop@ICML*, 2023.

[Boschin *et al.*, 2022] Armand Boschin, Nitisha Jain, Gurami Keretchashvili, and Fabian M. Suchanek. Combining embeddings and rules for fact prediction (invited paper). In *International Research School in Artificial Intelligence in Bergen, 2022*, pages 4:1–4:30, 2022.

[Broscheit *et al.*, 2020] Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. Libkge-a knowledge graph embedding library for reproducible research. In *EMNLP*, 2020.

[De Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, 2007.

[Dettmers *et al.*, 2018] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

[Galárraga *et al.*, 2013] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

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

[Galárraga, 2023] Luis Galárraga. Effects of locality and rule language on explanations for knowledge graph embeddings. In *International Symposium on Intelligent Data Analysis*, 2023.

[Himmelstein *et al.*, 2017] Daniel Scott Himmelstein, Antoine Lizee, Christine Hessler, Leo Brueggeman, Sabrina L Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E Baranzini. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *Elife*, 6, 2017.

[Lajus *et al.*, 2020] Jonathan Lajus, Luis Galárraga, and Fabian Suchanek. Fast and exact rule mining with amie 3. In *ESWC*. Springer, 2020.

[Liu *et al.*, 2021] Yushan Liu, Marcel Hildebrandt, Mitchell Joblin, Martin Ringsquandl, Rime Raissouni, and Volker Tresp. Neural multi-hop reasoning with logical rules on biomedical knowledge graphs. In *ESWC*. Springer, 2021.

[Meilicke *et al.*, 2021] Christian Meilicke, Patrick Betz, and Heiner Stuckenschmidt. Why a naive way to combine symbolic and latent knowledge base completion works surprisingly well. In *AKBC*, 2021.

[Meilicke *et al.*, 2023] Christian Meilicke, Melisachew Wudage Chekol, Patrick Betz, Manuel Fink, and Heiner Stuckeschmidt. Anytime bottom-up rule learning for large-scale knowledge graph completion. *VLDB*, 2023.

[Ortona *et al.*, 2018] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust Discovery of Positive and Negative Rules in Knowledge Bases. In *ICDE*, 2018.

[Ott *et al.*, 2023] Simon Ott, Patrick Betz, Daria Stepanova, Mohamed H Gad-Elrab, Christian Meilicke, and Heiner Stuckenschmidt. Rule-based knowledge graph completion with canonical models. In *CIKM*, 2023.

[Qu *et al.*, 2021] Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *ICLR*, 2021.

[Riley, 1991] Gary Riley. Clips: An expert system building tool. In *National Technology Transfer Conference and Exposition*, volume 2, 1991.

[Rossi *et al.*, 2021] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data*, 15(2), 2021.

[Sadeghian *et al.*, 2019] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, 2019.

[Suchanek *et al.*, 2007] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *WWW*, 2007.

[Suchanek *et al.*, 2019] Fabian M. Suchanek, Jonathan Lajus, Armand Boschin, and Gerhard Weikum. Knowledge Representation and Rule Mining in Entity-Centric Knowledge Bases. In *Reasoning Web Summer School*, 2019.

[Tanon *et al.*, 2017] Thomas Pellissier Tanon, Daria Stepanova, Simon Razniewski, Paramita Mirza, and Gerhard Weikum. Completeness-aware rule learning from knowledge graphs. In *IJCAI*, 2017.

[Toutanova and Chen, 2015] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

[Vrandečić and Krötzsch, 2014] Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledge Base. *Communications of the ACM*, 57(10), 2014.

[Yang *et al.*, 2017] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, 2017.