

pyJedAI: a Lightsaber for Link Discovery

Konstantinos Nikoletos¹, George Papadakis¹ and Manolis Koubarakis¹

¹National & Kapodistrian University of Athens, Panepistimioupolis 15703, Ilisia, Athens, Greece

Abstract

Link Discovery constitutes a crucial task for increasing the connections between data sources in the Linked Open Data Cloud. Part of this task is Entity Resolution (ER), which aims to identify owl:sameAs relations between different entity descriptions that pertain to the same real-world object. Due to its quadratic time complexity, ER is typically carried out in two steps: first, blocking restricts the computational cost to similar descriptions, and then, matching estimates the actual similarity between them. A plethora of techniques has been proposed for each step. To facilitate their use by researchers and practitioners, we present pyJedAI, an open-source library that leverages Python's data science ecosystem to build powerful end-to-end ER workflows. The purpose of this work is to demonstrate how this can be accomplished by expert and novice users in an intuitive, yet efficient and effective way.

Keywords

Link Discovery, Entity Resolution, Blocking, Matching

1. Introduction

At the core of Semantic Web lies the Linked Open Data (LOD) Cloud, with its constantly increasing size: from 570 datasets in 2014 to 1,255 in 2020 [1]. Yet, the links between its datasets remain low, just 16,174 as of May 2020 [1]. This means that on average, every dataset is connected to just 13 others, i.e., ~1% of all possible links. To increase the connectivity between the LOD datasets, Link Discovery automatically detects relations between their entity descriptions [2, 3].

Entity Resolution (ER) is a subtask of Link Discovery that focuses on detecting owl:sameAs between entity descriptions that represent the same real-world object [4, 5, 6]. ER constitutes a non-trivial task, due to two challenges:

1. its quadratic time complexity, which cannot scale to large volumes of data, and
2. the ambiguity in the entity descriptions.

The former challenge is addressed through *blocking*, which curtails the search space to highly similar descriptions, instead of considering all possible pairs [7]. The latter challenge is addressed through *matching*, which leverages similarity signals in order to categorize every pair of descriptions into matching or non-matching ones [8].

Numerous methods have been proposed for blocking and matching [9]. Yet, the available open-source ER tools offer very few of them, typically the ones proposed by their creators [3].

Woodstock'22: Symposium on the irreproducible science, June 07–11, 2022, Woodstock, NY

✉ sdi1700104@di.uoa.gr (K. Nikoletos); gpapadis@di.uoa.gr (G. Papadakis); koubarak@di.uoa.gr (M. Koubarakis)

🌐 <https://gpapadis.wordpress.com/> (G. Papadakis); <https://cgi.di.uoa.gr/~koubarak/> (M. Koubarakis)

🆔 0000-0002-7298-9431 (G. Papadakis); 0000-0002-1954-8338 (M. Koubarakis)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

The largest variety of methods is implemented by JedAI [10]. However, JedAI, like most Link Discovery tools, constitutes an isolated system, implemented in Java, which cannot be easily extended with existing state-of-the-art techniques from other domains, like Deep Learning and Natural Language Processing (NLP). To address this issue, we present pyJedAI, a new open-source system that implements the same methods as JedAI, but is capable of combining them with any package from Python’s data science ecosystem. We have publicly released the source code of pyJedAI at <https://github.com/Nikoletos-K/pyJedAI> under Apache License V2.0, which supports both academic and commercial applications.

2. System Overview

pyJedAI addresses the following task:

Given a source and a target dataset, S and T , respectively, discover the set of links $L = \{(s, owl : sameAS, t) | s \in S \wedge t \in T\}$.

Its architecture appears in Figure 1. The first module is the *data reader*, which specifies the user input. pyJedAI supports both semi-structured and structured data as input. The former, which include SPARQL endpoints and RDF/OWL dumps, are read by `rdflib`¹. The latter, which include relational databases as well as CSV and JSON files, are read by `pandas`². In this way, pyJedAI is able to interlink any combination of semi-structured and structured data sources, which is a unique feature.

The second step in pyJedAI’s pipeline performs *block building*, a coarse-grained process that clusters together similar entities. The end result consists of a set of candidate pairs, which are examined analytically by the subsequent steps. pyJedAI implements the same established methods for similarity joins and blocking as JedAI, such as Standard Blocking and Sorted Neighborhood, but goes beyond all Link Discovery tools by incorporating recent, state-of-the-art libraries for nearest neighbor search like FALCONN³ and FAISS⁴. In the near future, we will also add support for DeepBlocker [11], the best performing blocking method that leverages Deep Learning without the need to provide any labelled instances – just like all other block building methods.

The next two workflow steps are optional, implementing the same established block and comparison cleaning methods as JedAI. Their goal is to significantly reduce the number of candidate pairs, increasing the overall time efficiency and scalability at a small cost in effectiveness, i.e., by sacrificing recall to an insignificant extent. All methods are efficiently implemented on top of Python’s dictionaries, just like the block building ones.

The *entity matching* step estimates the actual similarity between the candidate pairs. Unlike all other Link Discovery tools, which rely exclusively on string similarity measures like edit distance and Jaccard coefficient [3], pyJedAI leverages the latest advanced NLP techniques, like pre-trained embeddings (e.g., word2vect, fastText and Glove) and transformer language models (i.e., BERT and its variants) [12]. More specifically, pyJedAI supports packages like

¹<https://rdflib.dev>

²<https://pandas.pydata.org>

³<https://falconn-lib.org>

⁴<https://github.com/facebookresearch/faiss>

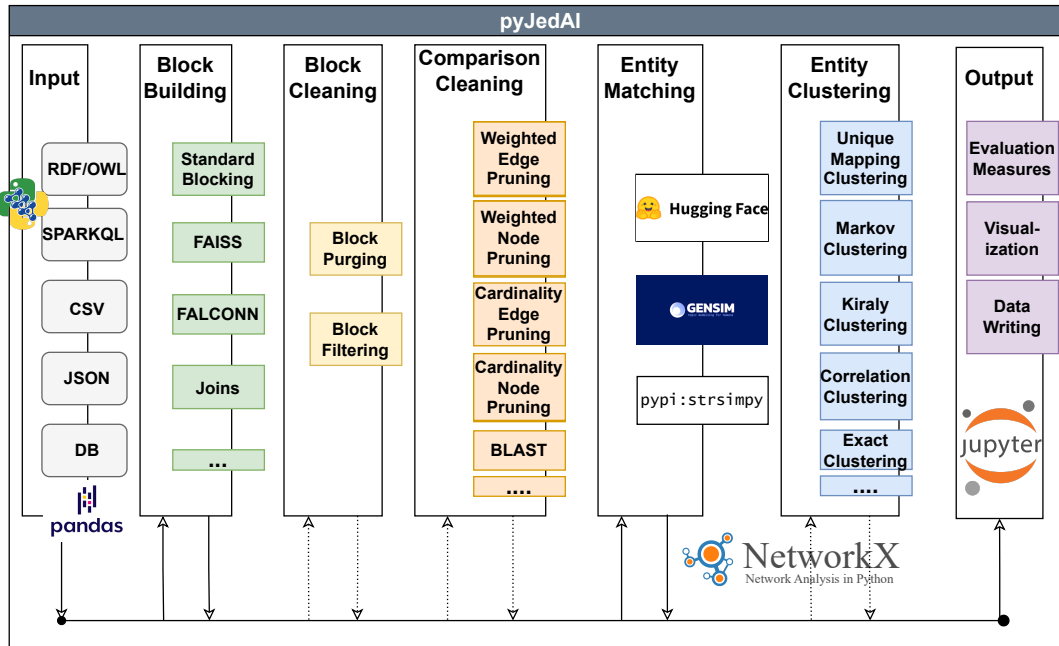


Figure 1: The architecture of pyJedAI. The dotted lines indicate optional steps.

pypi:strsimpy⁵, Gensim⁶ and Hugging Face⁷. This unique feature boosts pyJedAI’s accuracy to a significant extent, without requiring any labelled instances from the user.

The last step performs *entity clustering* to further increase the accuracy. The relevant techniques consider the global information provided by the similarity scores of all candidate pairs in order to take local decisions for each pair of entity descriptions. pyJedAI implements and offers the same established algorithms as JedAI, using NetworkX⁸ to ensure high time efficiency.

Finally, users are able to evaluate, visualize and store the results of the selected pipeline through the intuitive interface of Jupyter notebooks. In this way, pyJedAI facilitates its use by researchers and practitioners that are familiar with the data science ecosystem, regardless of their familiarity with ER and Link Discovery, in general.

3. Demonstration

The purpose of our demonstration is to highlight pyJedAI’s unique capabilities and ease-of-use. To this end, the user is merely asked to select the dataset(s) to be processed and the methods that will form the end-to-end workflow. For the former, the user can select any of the datasets for instance matching from the latest OAEI [13], or any of the four established benchmark

⁵<https://github.com/luozhouyang/python-string-similarity>

⁶<https://radimrehurek.com/gensim>


⁷<https://huggingface.co>

⁸<https://networkx.org>

Comparison Cleaning

```
In [18]: from pyjedai.comparison_cleaning import (
        WeightedEdgePruning,
        WeightedNodePruning,
        CardinalityEdgePruning,
        CardinalityNodePruning,
        BLAST,
        ReciprocalCardinalityNodePruning,
        ReciprocalWeightedNodePruning,
        ComparisonPropagation
        )

In [19]: wep = CardinalityEdgePruning()
        candidate_pairs_blocks = wep.process(filtered_blocks, data)

        Cardinality Edge Pruning: 100%  1076/1076 [00:00<00:00, 5523.70Hz/s]

In [20]: e = Evaluation(data)
        e.report(candidate_pairs_blocks, wep.execution_time)

+-----+
> Evaluation
+-----+
Precision:    6.28%
Recall:       92.19%
F1-score:    11.77%

Total execution time: 0:00:00.244808

True positives: 992
False positives: 14792
True negatives: 1142900
False negatives: 84

Total comparisons: 15784
```

Figure 2: Example of running pyJedAI through a Jupyter notebook.

ER datasets⁹ in any of the supported data formats. Regarding method selection, no labelled instances are required from any of the implemented techniques; the user merely needs to call them in the correct order and to configure their parameters, if the default ones do not yield satisfactory performance.

This is accomplished through a Jupyter notebook that contains detailed instructions for the user, lists all available methods per step and shows the status of every running method through a progress bar.¹⁰ Special care has been taken to assess the performance of every workflow step along with the overall pipeline. Thus, a series of effectiveness and time efficiency techniques is reported after every step. See Figure 2 for an example: command [18] shows the available methods for the optional step of Comparison Cleaning, command [19] applies one of them to the existing set of blocks, while showing its progress, and command [20] reports the performance of the clean set of blocks. Note that it is also possible to collectively report the performance of all tests executed in every session so as to facilitate the comparison between different pipelines and configurations.

4. Conclusions

pyJedAI constitutes the sole open-source Link Discovery tool that is capable of exploiting the latest breakthroughs in Deep Learning and NPL techniques, which are publicly available through

⁹https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution

¹⁰<https://nbviewer.org/github/Nikoletos-K/pyJedAI/blob/main/CleanCleanER-AbtBuy.ipynb>

the Python data science ecosystem. This applies to both blocking and matching, thus ensuring high time efficiency, high scalability as well as high effectiveness, without requiring any labelled instances from the user. In the future, we intend to extend pyJedAI with more capabilities, such as Schema Matching through the Valentine system (<https://github.com/delftdata/valentine-system>).

Acknowledgments

This work has received funding from the European Union’s Horizon 2020 research and innovation programme under GA No 101016798 (AI4Copernicus), EU Horizon Europe GA No 101070122 (STELAR), and from the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the “First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant” (Project Number: HFRI-FM17-2351 GeoQA).

References

- [1] The linked open data cloud, <https://lod-cloud.net/#about>, 2022.
- [2] A. Ferrara, A. Nikolov, F. Scharffe, Data linking for the semantic web, *Int. J. Semantic Web Inf. Syst.* 7 (2011) 46–76.
- [3] M. Nentwig, M. Hartung, A. N. Ngomo, E. Rahm, A survey of current link discovery frameworks, *Semantic Web* 8 (2017) 419–436.
- [4] P. Christen, *Data Matching*, Springer, 2012.
- [5] X. L. Dong, D. Srivastava, *Big Data Integration, Synthesis Lectures on Data Management*, Morgan & Claypool Publishers, 2015.
- [6] V. Christophides, V. Efthymiou, K. Stefanidis, *Entity Resolution in the Web of Data*, Morgan & Claypool Publishers, 2015.
- [7] G. Papadakis, D. Skoutas, E. Thanos, T. Palpanas, Blocking and filtering techniques for entity resolution: A survey, *ACM CSUR* 53 (2020) 31:1–31:42.
- [8] V. Christophides, V. Efthymiou, T. Palpanas, G. Papadakis, K. Stefanidis, An overview of end-to-end entity resolution for big data, *ACM CSUR* 53 (2021) 127:1–127:42.
- [9] G. Papadakis, E. Ioannou, E. Thanos, T. Palpanas, *The Four Generations of Entity Resolution*, Morgan & Claypool Publishers, 2021.
- [10] G. Papadakis, G. M. Mandilaras, L. Gagliardelli, G. Simonini, E. Thanos, G. Giannakopoulos, S. Bergamaschi, T. Palpanas, M. Koubarakis, Three-dimensional entity resolution with jedai, *Inf. Syst.* 93 (2020) 101565.
- [11] S. Thirumuruganathan, H. Li, N. Tang, M. Ouzzani, Y. Govind, D. Paulsen, G. Fung, A. Doan, Deep learning for blocking in entity matching: A design space exploration, *Proc. VLDB Endow.* 14 (2021) 2459–2472.
- [12] Q. Liu, M. J. Kusner, P. Blunsom, A survey on contextual embeddings, *CoRR* abs/2003.07278 (2020).
- [13] M. A. N. Pour, A. Algergawy, et al., Results of the ontology alignment evaluation initiative 2021, in: *Proceedings of the 16th International Workshop on Ontology Matching co-located with ISWC 2021*, volume 3063, 2021, pp. 62–108.