Well-Written Knowledge Graphs: Most Effective RDF Syntaxes for Triple Linearization in End-to-End Extraction of Relations from Texts (Student Abstract)

Célian Ringwald[1], Fabien Gandon[1, 2], Catherine Faron[1], Franck Michel[1], Hanna Abi Akl[1, 2]
[1] Université Côte d'Azur, Inria, CNRS, I3S
[2] Data ScienceTech Institute
celian.ringwald@inria.fr, fabien.gandon@inria.fr, faron@i3s.unice.fr,
fmichel@i3s.unice.fr, hanna.abi-akl@inria.fr

Abstract

Seq-to-seq generative models recently gained attention for solving the relation extraction task. By approaching this problem as an end-to-end task, they surpassed encoder-based-only models. Little research investigated the effects of the output syntaxes on the training process of these models. Moreover, a limited number of approaches were proposed for extracting ready-to-load knowledge graphs following the RDF standard. In this paper, we consider that a set of triples can be linearized in many different ways, and we evaluate the combined effect of the size of the language models and different RDF syntaxes on the task of relation extraction from Wikipedia abstracts.

Introduction

The resolution of the relation extraction (RE) task - consisting of retrieving relations from unstructured text - was drastically improved recently by two main changes: (1) the construction of massive corpora aligning texts and facts from Knowledge graphs (KG) e.g. Wikipedia articles with the KGs of Wikidata and DBpedia, and (2) the usage of pretrained language models (PLM) and more recently generative models. However, Wikidata and DBpedia still struggle with coverage and quality issues. In this context, extracting the missing information in these KGs from Wikipedia is an interesting way to fill the gap. To our knowledge, no system was proposed to perform a semantic relation extraction where the extracted triples explicitly follow an RDF syntax. Generative PLMs are very flexible, but the formulation of the task is a really sensitive choice. In this article, we specifically address the following research question: **RQ** – *How does the choice of syntax impact the generation of triples using datatype properties?*

Related Work

Before investing in generative PLMs, the research community focused on systems built on top of encoder-only PLMs (derived from BERT), where relations were decoded by design in a discriminative manner. Since 2021, generative PLMs have gained interest after demonstrating their ability to solve complex tasks in an end-to-end design. Mainly based on the fine-tuning of BART or T5, the triples syntax is learned implicitly from the examples submitted during training (Ye et al. 2022).

The choice of syntax to represent the triples provided to the model is part of the "linearization process". Until now, different methods have been investigated but they were not rigorously compared. The two main solutions proposed were to represent the relation as a list of triples (Wang et al. 2022): $((s1, p1, o1), (s2, p1, o2), ...)$ or via a sequence of tags (Ke et al. 2021) where each element of the triple is preceded by a special token e.g. H, R, T in $\langle H\rangle s1\langle R\rangle p1\langle T\rangle o1\langle H\rangle s2\langle R\rangle p1\langle T\rangle o2$. To limit the number of tokens, (Huguet Cabot and Navigli 2021) proposed a triple linearization method where triples sharing the same subject are grouped to avoid repetition.

As for the representation of triples, the W3C proposed several RDF syntaxes: • RDF-XML which suffers the verbosity of XML • N-Triples, an easy-to-parse line-based format • Turtle, a lighter and easier-to-read syntax; that allows the use of qualified names for URIs to compact their writing and it integrates shortcuts to cram triples sharing the same subjects or the same predicates[1] • JSON-LD, relying on JSON, the now popular Web format.

Experiments

Scope of the study: The scientific community has recently pointed out the "hallucination" problem of PLMs. In practice, this issue may affect the generation of triples with literal values as objects (e.g. attributes such as dates, measures, textual descriptions, etc.). Following the OWL semantics, these triples are represented with datatype properties on which we focused for the first step of our experiments. To evaluate the impact of the triple syntax on the extraction of datatype relations, we compared the results obtained with seven different syntaxes. The core idea is to find the best trade-off between the expressiveness and the conciseness of the syntaxes on one hand, and the ease with which a selected PLM can learn them on the other. The first two selected syntaxes are the ones used in the literature: the list and the tagged sequence. Four syntaxes are from the RDF standard: Turtle, RDF-XML, N-Triples and JSON-LD. The last one is a simplified Turtle syntax where every namespace, prefix, and datatype is considered predefined and thus implied.

[1] https://www.w3.org/TR/turtle/#predicate-lists

Syntax	size (MB)	min. token length	max. token length	avg. token length
JSON-LD	1 104	134	617	317
XML	1 089	177	444	280
NTriples	1 004	98	580	226
Turtle	810	66	285	153
Tags	642	21	351	86
List	623	16	316	67
Turtle light	597	21	171	58

Table 1: Datasets description

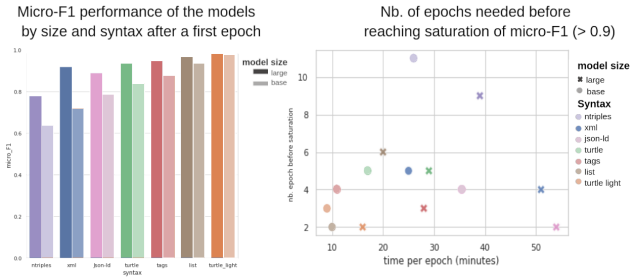

Figure 1: Synthesis of the main results of our experiment.

Dataset: We built a dataset based on the English chapter of DBpedia, focusing on the triples whose subjects are instances of `dbo:Person`, the class having the largest number of datatype properties, especially on the following ones: `rdfs:label`, `dbo:birthDate`, `dbo:deathDate`, `dbo:birthYear`, `dbo:deathYear`. Then we applied two filtering steps on the resulting dataset: a selection of the only triples respecting both of the aforementioned constraints, expressed as a SHACL shape and a lookup process keeping only the triples where the datatypes values are clearly mentioned in the Wikipedia abstracts.

Methodology: We fine-tuned several generative models on the selected syntaxes. The first run of this experiment uses two versions of BART (base and large) configured in a conditional generation mode. We trained these models via cross-validation by choosing during each epoch: a training random subset of our dataset of 10 000 examples; validated and tested via two other random subsets of 3 000 examples. The training was conducted on a single GPU Tesla V100-SXM2-32GB, with an early stop mode: after 5 training steps without loss improvement, the training is stopped. All the material extending the forked REBEL GitHub repository is publicly available[2].

Results and discussions: The results of the experiments are compiled in a Weights and Bias dashboard[3], a digest is provided in Figure 1. The left graph presents the micro-F1 reached after the first epoch and reveals a significant difference in terms of performance between the syntaxes on the BART-base model. Nonetheless, with BART-large, this performance gap is narrowed. On the right graph of Figure 1, we can see that the BART-large takes longer for an epoch but needs less of them to get good results. We can also read the number of epochs needed to reach a micro-F1 greater than 0.9. The number of epochs needed to reach the state of saturation is lower for BART-large. Three syntaxes caught our attention: *Turtle* requires the same number of epochs for BART-base and BART-large; the *List* syntax needs more training epochs to be learned by BART-large than by BART-base; conversely, the *JSON-LD* and the *XML* were easier to learn by BART-large. These results reveal a clear winner: the *simplified Turtle syntax* that outperformed all the other syntaxes in every aspect, and a clear loser: the *N-Triples* syntax that took two or three times longer in terms of training epochs for reaching the micro-F1 saturation compared to other syntaxes.

[2]https://github.com/datalogism/SyntaxBart

[3]https://wandb.ai/celian-ringwald/SyntaxBART?workspace=user-celian-ringwald

Conclusions

Our experiment has shown the significant impact of the RDF syntax chosen on the extraction of datatype properties from texts. In this context, the simplified Turtle syntax proposed is a promising Knowledge Graph linearization solution: fast to learn and economical regarding token length. Moreover, the original Turtle syntax shows its stability on both models (base and large) and could be an interesting option considering its expressiveness and conciseness. In the future, we will extend this experiment with a more robust approach incorporating k-fold cross-validation and other measures more adequate to assess the generated triples.

References

Huguet Cabot, P.-L.; and Navigli, R. 2021. REBEL: Relation Extraction By End-to-end Language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2370–2381. ACL.

Ke, P.; Ji, H.; Ran, Y.; Cui, X.; Wang, L.; Song, L.; Zhu, X.; and Huang, M. 2021. JointGT: Graph-Text Joint Representation Learning for Text Generation from Knowledge Graphs. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2526–2538. Online: Association for Computational Linguistics.

Wang, C.; Liu, X.; Chen, Z.; Hong, H.; Tang, J.; and Song, D. 2022. DeepStruct: Pretraining of Language Models for Structure Prediction. In *Findings of the Association for Computational Linguistics: ACL 2022*, 803–823. Dublin, Ireland: Association for Computational Linguistics.

Ye, H.; Zhang, N.; Chen, H.; and Chen, H. 2022. Generative Knowledge Graph Construction: A Review. In *Proc. of the 2022 Conference on Empirical Methods in Natural Language Processing*, 1–17. ACL.