OLaLa Results for OAEI 2023

Sven Hertling, Heiko Paulheim

Data and Web Science Group, University of Mannheim, Germany

Abstract

This paper presents the results of the OLaLa matching system participating in the OAEI 2023. The system is based on sentence-transformers as well as large language models. The former is used to generate correspondence candidates which is independent of any overlapping tokens because the comparison is only based on embeddings. To finally select the best mappings, a large language model is used to decide if two given textual representations of the source and target concept are equal or not. Based on positive and negative words that the LLM predicts, a confidence is extracted. Still, there are a lot of decisions that heavily influence the final result like (1) how can each concept be verbalized into text, (2) which prompt to use, and (3) which language model to choose. A lot of combinations were executed and the most useful one is submitted and packaged as a matching system.

Keywords

Ontology Matching, Knowledge Graphs, Large Language Model

1. Presentation of the system

Textual descriptions in ontologies are a strong signal for the matching task. With the rise of transformer-based models [1], many systems were developed which have some disadvantages. One of them is the need for huge training data (because the last classification layer is randomly initialized and needs to be trained). Another is the restricted amount of tokens (words/ pieces of text) that can be processed.

Thus in this work, we focus on Large Language Models (LLMs) which should serve as a decision function. One of the famous models is ChatGPT[1] which we do not use in this work because of the following reasons: The model is not open source and thus all results are not reproducible because OpenAI might shut down a model that is not accessible anymore. Furthermore, there is no possibility to access the scores/logits of the model. And lastly, the company charges the user with some cost per query. Larger ontologies require many more queries and thus it is questionable if it is still economically sensible. Therefore we use open source models such as LLaMa 2 [2] to be able to reproduce the results and run the model on our hardware.

OM 2023: The 18th International Workshop on Ontology Matching collocated with the 22nd International Semantic Web Conference ISWC-2023 November 7th, 2023, Athens, Greece

sven.hertling@uni-mannheim.de (S. Hertling); heiko.paulheim@uni-mannheim.de (H. Paulheim)

https://www.uni-mannheim.de/dws/people/researchers/phd-students/sven-hertling/ (S. Hertling); https://www.uni-mannheim.de/dws/people/professors/prof-dr-heiko-paulheim/ (H. Paulheim)

0000-0003-0333-5888 (S. Hertling); 0000-0003-4386-8195 (H. Paulheim)

CEUR Workshop Proceedings (CEUR-WS.org)

[1] https://chat.openai.com/

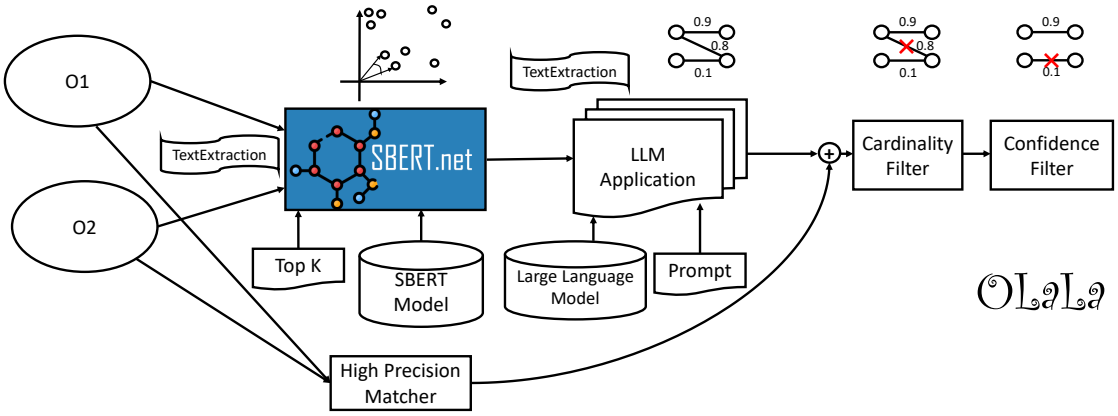

Figure 1: An overview of the *OLaLa* system (from [3]).

Still, a lot of decisions need to be made when designing a full matching system such as (1) how to present each concept/correspondence to the model, (2) which prompt to use, and (3) which model to choose.

There is already a publication about the system in general [3]. The following description is taken from this paper.

Figure 1 shows an overview of the simple architecture of the *OLaLa* system. All components are implemented in MELT [4], a framework for matcher development and evaluation. MELT is also used by the OAEI to package, submit, and evaluate the systems. Thus it is possible for the ontology matching community to use each component in their own matching pipeline and customize them to their liking. The implementation of *OLaLa* is publicly available and we provide a console application[2] which allows to run the system and modify the most important parameters.

At the very beginning, some matching candidates need to be extracted from the two given input ontologies O1 and O2. Afterwards, those candidates are included in the user-defined prompt and presented to the LLM. Two options are possible: 1) each correspondence is analyzed independently of each other 2) given a source entity all possible target entities are presented and the LLM needs to decide which one is correct (or none of them). The output of the high-precision matcher is added to ensure that the simple matches are included as well. Finally, some filters are applied to fulfill the usual requirements for an alignment such as a 1:1 mapping (cardinality filter). The confidence filter at the end ensures that only correspondences with reasonably high confidence are returned. In the following sections, we will describe each step in more detail.

1.1. Candidate Generation

Due to the fact that the LLMs can usually not analyze the input ontologies as a whole (except small ontologies like those in the OAEI conference track, see [5]), some correspondence candidates need to be generated. In this stage only the recall is interesting and the higher the recall the better. Some of the related approaches apply an inverted index to find possible

[2]https://github.com/dwslab/melt/tree/master/examples/llm-transformers

similar entities. This requires some textual overlap of those concepts. In *OLaLa*, the well-known Sentence BERT models (SBERT) are used to generate those candidates. This allows a higher recall because it can also find similar entities without any textual overlap. The trained SBERT models are finetuned siamese BERT models on a huge set of paraphrases [6]. SBERT as well as all LLMs only process text, but the input is an ontology. Thus it is necessary to verbalize the concepts into some natural language text. In MELT they are called `TextExtractors` (see section 1.3).

For the candidate generation step, we use the so-called `TextExtractorSet`. It extracts all texts of a resource which are either labels (e.g. `rdfs:label`, `skos:prefLabel`, `schema:name`) or descriptions (e.g. `rdfs:comment`, `dc:description`, `schema:comment`). In addition to that, the URI fragment is extracted in case it contains not more than 50% numbers. As a last step, all annotation properties are followed recursively and all labels of those resources are added as well.

All those extracted texts for each resource are embedded and a semantic search is executed. It computes the cosine similarity between a list of query embeddings and a list of corpus embeddings and returns the top-k neighbors for each text. From those, we select the top-k best neighbors per resource. This procedure is then repeated but the source and target ontologies are swapped such that both act once as the query and once as the corpus embedding.

1.2. LLM Application

There are two principal approaches how the candidates are presented to the LLM. The first one is *binary decisions*, i.e., deciding whether one candidate is correct or not; the second is *multiple choice* decisions, i.e., selecting the most likely correspondence for one concept from a set of possible targets.

1.2.1. Binary Decisions

Binary decisions are implemented in class `LLMBinaryFilter`. For each candidate correspondence, the source and target entity are verbalized into text and replaced in the prompt given by the user. The output of generative models, such as the ones applied in this work, is always natural language text. To convert this into a binary decision, the following technique is applied: We search for target tokens/words that indicate the result (e.g. true/yes or false/no). If such a token is found, the generation process is directly stopped. Due to the high computation amount, an early stopping approach is useful to process a large number of candidates. Up to now, only the decision is extracted and in case the model generates other texts like “This is a correct match”, we fail at detection.

To overcome this issue and also extract a specific confidence we do the following. If any of the target tokens is detected, then we retrieve the scores of the complete vocabulary and apply the softmax function to it. This corresponds to the probability that the word is generated at this position. We check for all words in a class (e.g. yes, true) the probability and take the maximum value which is normalized by the negative class. Thereby, we get a confidence between zero and one, and every confidence above 0.5 is a predicted positive token.

In case no positive or negative token is generated, the probabilities at the first generated

token are used. All those computations would not be possible with a model accessed by an API such as ChatGPT. Prompt engineering would be another way to get to confidences (e.g. use a prompt such as "and also provide a confidence score with your answer") but usually, the chatbot will respond that it is not able to provide a specific confidence value and even if it does, it is not easy to extract it out of the generated text.

The default generation strategy[3] is greedy such that each token with the highest probability is chosen and the generation process is continued with this text. The implementation also allows to switch to e.g. contrastive search [7] but due to the usual short answers, it is not necessary nor helpful.

1.3. TextExtractors / Verbalizers

In addition to combining all texts from the `TextExtractorSet` explained before, an even simpler extractor called `TextExtractorOnlyLabels` is implemented. It extracts only one text which can originate from the following properties(in decreasing importance): `skos:prefLabel`, `rdfs:label`, URI fragment, `skos:altLabel`, `skos:hiddenLabel`. This means if a `skos:prefLabel` is detected, only this label is used.

Including more context in those examples is achieved by the `TextExtractorVerbalizedRDF`. It selects all RDF triples from the corresponding KG where the resource is in the subject position. Those triples are verbalized - meaning that each subject, predicate, and object is replaced by the text of `OnlyLabels` extractor. All triples with a label-like property are skipped because the information is already included. As an example, the statement":MA_0000002 rdfs:subClassOf :MA_0001112" is converted to "spinal cord grey matter sub class of grey matter".

As a variation of the previous extractor, it is also tried out to provide the triples directly as serialized RDF. The default of the `ResourceDescriptionInRDF` extractor is to serialize to turtle format where the prefixes are used but the prefix definition is excluded from the generated text to make it shorter (other serializations can also be configured). If there are resources in the object position of the triples, they will be also replaced by a literal containing the corresponding label.

1.4. High-Precision Matcher

The high-precision matcher is a simple matcher in MELT that efficiently searches for concepts with the exact same normalized label (or URI fragment if a label is not available).[4] The normalization includes lowercasing, camel case, and deletion of non alpha-numeric characters. If there is only one such candidate for a concept, then it is matched.

1.5. Postprocessing

After the application of the LLM, the resulting alignment is further post-processed by filters. To keep the matching pipeline simple, only two additional filters are applied. The *cardinality filter* ensures a one-to-one mapping which is usually required.

[3] https://huggingface.co/docs/transformers/main/en/generation_strategies
[4] https://dwslab.github.io/melt/matcher-components/full-matcher-list

To further improve the alignment and remove correspondences that are likely to be incorrect, the confidence filter is applied. All correspondences that do not have a higher or the same confidence as a predefined threshold value are excluded.

1.6. Final Configuration

For the final configuration, a lot of parameters need to be fixed. The SBERT model for the candidate generation step is set to `multi-qa-mpnet-base-dot-v1`,[5] and the value k during the top-k neighbors search is set to five. This gives a balance between the number of generated correspondences as well as the achieved recall. The `TextExtractorSet` is used to generate multiple text representations of the resource to run the search in the embedding space.

The LLM model is set to `upstage/Llama-2-70b-instruct-v2`[6] and to generate the text in prompt 7 (see [3]), i.e., a few-shot prompt with three positive and negative examples each[7], `TextExtractorOnlyLabels` is used. With this prompt, the binary decision approach is automatically selected. For the text generation, the maximum number of tokens (`max_new_tokens`[8]) is set to 10 but this number of tokens is usually not reached because a positive or negative word is detected before. The next parameter which is fixed is the temperature. The lower the value, the more deterministic the results are (the token with the highest probability is chosen as the predicted token). With increased temperature, the outputs are more randomized (resulting in more creative texts). We set the temperature to zero such that the results are reproducible. Other generation parameters are set to their default values.

The cardinality filter does not require any parameters, and the value of the confidence filter is set to 0.5. With this setting, we filter out all correspondences where the LLM predicts a negative word (such as “no“ or “false“). Thus we do not need to tune the confidence value and do not require any training alignment for it.

1.7. Adaptations made for the evaluation

OLaLa is available as a docker image provided at figshare. If the image is started, an HTTP endpoint within the container on port 8080 is started. The webserver fulfills the requirements of the REST interface described in the MELT user guide [9].

1.8. Link to the system and parameters file

OLaLa can be downloaded from
https://doi.org/10.6084/m9.figshare.24150846.v2. [8]

[5] https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1
[6] https://huggingface.co/upstage/Llama-2-70b-instruct-v2
[7] The positive and negative examples are taken from the anatomy track and used across all tracks.
[8] https://huggingface.co/docs/transformers/main/en/main_classes/text_generation#transformers.GenerationConfig
[9] https://dwslab.github.io/melt/matcher-packaging/web

2. Results

This section discusses the results of OlaLa for each track of OAEI 2023. The system is not able to produce meaningful results on the multiform track because it is not yet designed for multilingual input. But there are also open-source LLMs out there that support it well and it is worth testing them in the future.

In this year, it is also possible to submit final alignment files in case the system requires substantial hardware resources. OLaLa requires two GPUs with at least 40 GB RAM. Thus we also submitted the produced alignment files.

2.1. Anatomy

When comparing the F1 measure in the anatomy track, the system is in second place with 0.91. This is a rather good value for the track without explicit background knowledge. Nevertheless, one needs to take care take also the pretrained model is trained on a huge amount of text that may cover similar topics.

To improve the results, the recall needs to be higher. Furthermore, the alignment could be coherent with respect to the given ontologies by using an explicit check and filtering of correspondences.

2.2. Conference

In the conference track, OLaLa (F1 of 0.6) is way better than the string equivalence approach (F1 of 0.53). But there are still four matchers that are better than this value. In this track, the precision value is quite low at 0.59. Due to the fact that each correspondence has an associated confidence, a higher threshold would make sense here.

2.3. Bio ML

In this track, OLaLa performs quite well. Only SORBETMtch and LogMapBio are better in most test cases. We still need to look into SNOMED-FMA (Body) test case where OLaLa is the worst system in terms of F1 unsupervised measure.

2.4. Biodiv

In biodiv track, the system scores are on the first place for the following test cases:

- NCBITAXON-TAXREFLD Plantae
- NCBITAXON-TAXREFLD Fungi
- NCBITAXON-TAXREFLD Chromista
- NCBITAXON-TAXREFLD

For NCBITAXON-TAXREFLD Animalia LogMapLt is a bit better.

2.5. Common Knowledge Graphs

For the test case Nell-DBpedia, the proposed system performs best even without instance information because for class matches it does not require any instance information.

For Yago-Wikidata most other systems (except LsMatch) are better in terms of F1.

2.6. Knowledge Graph

OLaLa is very good at property matching but for classes some systems like SORBETMtch or LSMatch are better. In the future, we also plan to include instance matching as well which currently takes too much time to execute.

3. General comments

3.1. Discussions on the way to improve the proposed system

The system can be optimized in multiple ways.

The main issue is scalability because the prediction of new tokens by the LLM takes a lot of time (including the initial processing of the prompt). To reduce this time, the generation process can be sped up (e.g. batch processing). Additionally, we could only process the correspondences with the highest confidence first, and only in case the LLM predicts a negative token, we process other candidates.

Another extension would be to differentiate between class, property, and instance matches. Up to now, the prompt and text extraction are the same for all types of resources. Maybe it is helpful to change either the prompt, the text extractor, or both for each type.

4. Conclusions

In this paper, we have analyzed the results of the OLaLa system. In many tracks, it can achieve very good results and shows that the textual information is very helpful when generating correspondences.

Most of the components that are used in this system are included in the MELT framework[4] which allows other researchers to reuse and compose components in their systems.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv:2307.09288 (2023).

[3] S. Hertling, H. Paulheim, Olala: Ontology matching with large language models, in: Knowledge Capture Conference 2023 (K-CAP '23), December 5–7, 2023, Pensacola, FL, USA, 2023. doi:10.1145/3587259.3627571.

[4] S. Hertling, J. Portisch, H. Paulheim, MELT - matching evaluation toolkit, in: International conference on semantic systems (SEMANTICS), 2019, pp. 231–245.
[5] S. S. Norouzi, M. S. Mahdavinejad, P. Hitzler, Conversational ontology alignment with chatgpt, arXiv preprint arXiv:2308.09217 (2023).
[6] N. Reimers, I. Gurevych, Sentence-bert: Sentence embeddings using siamese bert-networks, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2019. URL: http://arxiv.org/abs/1908.10084.
[7] Y. Su, T. Lan, Y. Wang, D. Yogatama, L. Kong, N. Collier, A contrastive framework for neural text generation, Advances in Neural Information Processing Systems 35 (2022) 21548–21561.
[8] S. Hertling, OLaLa for OAEI (2023). URL: https://figshare.com/articles/software/OLaLa_for_OAEI/24150846. doi:10.6084/m9.figshare.24150846.v2.