Proceedings Paper:
Fallatah, O., Zhang, Z. and Hopfgartner, F. (2021) KGMatcher results for OAEI 2021. In: Shvaiko, P., Euzenat, J., Jiménez-Ruiz, E., Hassanzadeh, O. and Trojahn, C., (eds.) CEUR Workshop Proceedings. 16th International Workshop on Ontology Matching co-located with the 20th International Semantic Web Conference (ISWC 2021), 25 Oct 2021, Virtual conference. Technical University of Aachen, pp. 160-166. ISSN: 1613-0073.

KGMatcher Results for OAEI 2021

Omaima Fallatah[1,2], Ziqi Zhang[1], and Frank Hopfgartner[1]

[1] Information School, The University of Sheffield, Sheffield, UK
{oafallatah1,ziqi.zhang,f.hopfgartner}@sheffield.ac.uk
[2] Department of Information Systems, Umm Al Qura University, Saudi Arabia
oafallatah@uqu.edu.sa

Abstract. *KGMatcher* is a scalable and domain independent matching tool that matches the schema (classes) of larger Knowledge Graphs by following a hybrid matching approach. *KGMatcher* is composed of an instance-based matcher which only uses annotated instances of knowledge graph classes to generate candidate class alignments, and a string-based matcher. This year is the first OAEI participation of *KGMatcher*, and it is the best performing system on the common knowledge graph track. Although *KGMatcher* results are promising, further improvements of the matching techniques' and matcher combination can be introduced.

Keywords: Knowledge Graphs · Instance-based Ontology Matching · Machine Learning · Schema Matching.

1 Presentation of the system

1.1 State, purpose, general statement

Combining different matching techniques is a common practice in ontology matching tools. Matching techniques are divided into three main categories [10]: (1) *Element level* techniques which discover similar entities by processing textual annotation of ontologies entities, (2) *Structural level* techniques which study the relation between ontologies entities to generate candidate similar pairs, and Finally (3) *Extensional* or *Instance_based* techniques which utilize populated instances, i.e., (ABox) data to generate alignments at schema level (TBox).

The matcher proposed here is a hybrid approach that combines an element level matcher to an instance-based one. The first matcher uses entity labels to generate candidate pairs, and the latter produces candidate class alignments by only using annotated instance names. While conventional ontologies mainly focus on modelling classes and properties, Knowledge Graphs (KGs), particularly those available on the web, are large-scale and describes numerous instances [3]. Following a self-supervised and two-way classification approach, the presented matcher trains a classifier using each KG instances as training data. Each trained

KG classifier can then be used to classify any instance name into one of the classes from that KG. This approach is domain independent and is capable of coping with KGs with unbalanced populations (for details, see [4]). This makes the matcher particularly useful for matching large KGs with numerous populated and overlapping instances such as DBpedia [1] and YAGO [12].

1.2 Specific techniques used

Given two input knowledge graphs, $\mathcal{O}$ and $\mathcal{O}'$, where $\mathcal{O}$ contains a set of classes $\mathcal{O} = \{C^0_{\mathcal{O}}, C^1_{\mathcal{O}}, .., C^i_{\mathcal{O}}\}$, and each class contains a set of instances $C^i_{\mathcal{O}} = \{e^i_0, e^i_1, ..., e^i_n\}$. Similarly, $\mathcal{O}'$ contains a set of classes such that $\mathcal{O}' = \{C^0_{\mathcal{O}'}, C^1_{\mathcal{O}'}, .., C^j_{\mathcal{O}'}\}$ where $C^j_{\mathcal{O}'} = \{e^j_0, e^j_1, ..., e^j_m\}$. *KGMatcher* has two main components: an **instance-based matcher** and a **name matcher**. The workflow of *KGMatcher* is illustrated in figure 1.

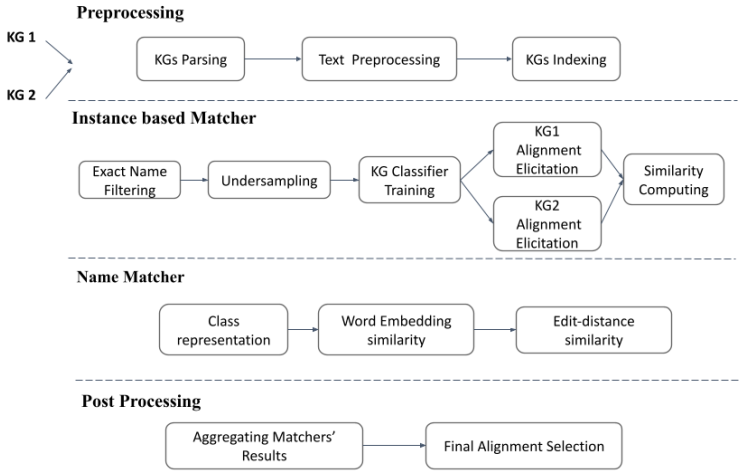

Fig. 1: An overview of *KGMatcher* process

Preprocessing Given the two input KGs, the matcher starts by parsing and indexing the lexical data of the two KGs separately. Following the standard free text search/index approach, an index is created for each KG where each class is treated as a document and the content of each 'document' is the concatenation of the class's instance labels. In addition to the standard text cleaning processes, a word segmentation method is applied in order to separate multi-word entities, e.g., `academicfield`. Using a dictionary, the method is able to infer the spaces between words and replace them with a space.

Instance-based Matcher The first component of *KGMatcher* belongs to the extensional matcher category. It uses a self-supervised machine learning approach to map KG classes based on their instances overlap. The matching is done in a two-way classification method where a *KG classifier* is trained using one KG's instances data. Later on, that classifier is used to classify any instance name into one of the classes from that KG.

- *Exact Name Filtering.* The matcher starts by applying an exact name filter to exclude class labels that exist in both input KGs. Given the large number of classes in typical KGs, this step works as a blocking strategy which reduces the search space of the instance-based matcher.

- *Undersampling.* As the instance distribution in KGs tends to be highly imbalanced, the goal of this step is to balance the number of populated instances in the input KGs to avoid biased classification results. *KGMatcher* follows a resampling strategy aimed at undersampling classes with instances number exceeding the average number of instances per class in that KG, i.e., majority classes. The standard TF-IDF [11] method which often deployed to measure word relevance in a collection of documents is used to undersample KGs classes. Here, the TF-IDF of a word in each class represents the relevance of that word in a particular class in comparison to other classes in the KG. Therefore, for each majority class, the most frequent words in terms of TF-IDF score are used to undersample its instance names. Therefore, instance names that do not compose any of the words with high TF-IDF scores are discarded. As a result, a balanced and indicative set of a KG instance names are obtained to be used as training data. For details about this step, see [4].

- *Training KG classifiers.* Here, a KG classifier will be separately trained for each input KG using the previously undersampled data. Pre-trained word embeddings are used here as features to capture and present the semantics of KG instance names. Compared to traditional feature representation methods, word embedding and language models are recognized as effective ways to capture the semantic similarity of words. *KGMatcher* is able to train two types of classifiers, a Deep Neural Network (DNN) model[3] [4] similar to other successful NLP tasks such as [9] and [2], and a pre-trained BERT model [8]. *KGMatcher* will automatically opt to using the BERT model if a GPU is available during runtime. The output of this phase is two classifiers, $CLS_{\mathcal{O}}$ and $CLS_{\mathcal{O}'}$ trained using the instances from the two input KGs $\mathcal{O}$, and $\mathcal{O}'$ respectively.

[3] The parameters selected for the DNN model: an input layer of pre-trained word embedding model followed by four fully connected hidden layers with 128, 128, 64, 32 rectified linear units. A dropout layer of 0.2 is added between each pair of dense layers for regulation. Finally, a *softmax* layer for multi-class classification, taking the total number of classes in the KG we are training a classifier for.

[4] The input layer is the Google News token-based model `https://tfhub.dev/google/tf2-preview/nnlm-en-dim128/1`

- *KG1 alignment elicitation* is the process of generating candidate pairs using the classifier trained on the first input KG. Candidate pairs are generated by iteratively applying the classifier $CLS_{\mathcal{O}}$ to instances in the other KG's classes. As a result, each instance name in $C^j_{\mathcal{O}'}$ is now classified into a class in $\mathcal{O}$. The candidate pair ($C^i_{\mathcal{O}}$,$C^j_{\mathcal{O}'}$) is added to the first candidate alignments set $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ if the majority of $C^j_{\mathcal{O}'}$ were classified as instances of $C^i_{\mathcal{O}}$. A similarity score between [0,1] is obtained using the percentage of instances that voted for a particular class. Therefore, if 600 out of 1000 instance names in $C^j_{\mathcal{O}'}$ were voting for, $C^i_{\mathcal{O}}$ the similarity score of that pair will be 0.6.

- *KG2 alignment elicitation* is similar to the above illustrated elicitation process. However, the roles of the two KGs are reversed where $CLS_{\mathcal{O}'}$, i.e., the classifier trained on the second KG ($\mathcal{O}'$), is applied to $\mathcal{O}$ instances in order to obtain the second candidate alignment set $A_{\mathcal{O}'\rightarrow\mathcal{O}}$.

- *Similarity computing* is where *KGMatcher* combines the two candidate alignment sets resulted from the two-way classification method. First, the matcher separately stores each directional alignment in an alignment matrix of a $|\mathcal{O}|.|\mathcal{O}'|$ dimension. The two matrices are then aggregated into one matrix by taking the average similarity score of each pair. For example, if ($C^6_{\mathcal{O}}$,$C^3_{\mathcal{O}'}$,0.88) in $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ and ($C^5_{\mathcal{O}'}$,$C^3_{\mathcal{O}}$, 0.64) in $A_{\mathcal{O}'\rightarrow\mathcal{O}}$) their aggregated similarity value will be 0.76. Consequently, the final alignments for this matcher are chosen by following the state-of-the-art automatic final alignment selection approach introduced in [5]. Given an alignment matrix, this method iteratively select the pair with the highest similarity score for each class in both KGs.

Name Matcher The second component of *KGMatcher* is an element-level matcher, which measures the similarity of KG class labels. First, the edit distance of each class pair is measured, and then their semantic similarity is measured by a word embedding method. For the edit distance, *KGMatcher* calculates the levenshtein distance for each class pair. Regarding the word embedding similarity, a pre-trained `word2vec` model is used to represent class labels before measuring their cosine similarities. The semantic similarity is measured in a Vector Space Model, where words with high semantic relations are often represented closer to each other. In the case of multi-words labels, the vector representation of each word composing the label is aggregated with an element-wise average of the composing word vectors. Finally, the maximum of the two similarity measures is chosen as the name similarity of that pair. The threshold value of the name matcher is set to 0.8. To illustrate, if the word embedding similarity of (`RailwayStation`,`TrainStation`) is 0.83 while their levenshtein distance is 0.56, the maximum similarity value, i.e., the word embedding similarity which is also higher than 0.8. Nonetheless, in case that the two similarity scores are lower than the threshold, that pair will be excluded from the candidate alignment.

Post Processing *KGMatcher* combines the results generated from the two component matchers by following the same method described earlier in 2 to combine the two instance classification alignments.

Instance Matching For the OAEI participation, we have adapted *KGMatcher* to also match the instances of KGs. The instance matching component is very simple. First, standard text preprocessing techniques such as lowercasing, and removing stopwords and non-alphanumeric characters are applied. Then, *KGMatcher* generates candidate instance pairs based on the existence of the label in the opposite knowledge graph.

1.3 Adaptations made for the evaluation

KGMatcher is mainly developed with Python. To facilitate reusing and evaluating *KGMatcher* and for the OAEI submission, *KGMatcher* was packaged using a SEALS client. The wrapping services from the Matching EvaLuation Toolkit (MELT) [7] was used to warp *KGMatcher*'s Python-process, and to generate the SEALS package.

2 Results

In this section, we present and discuss the results for each of the OAEI tracks where *KGMatcher* was able to produce a non-empty alignment file. The results include the following OAEI tracks: Conference, Knowledge Graph, and Common Knowledge Graphs track.

2.1 Conference

In the Conference track, when following the rar2-M3 evaluation, *KGMatcher* F1 score (0.52) is slightly lower than both baselines, i.e., *StringEquiv* (0.53) and *edna* (0.56). This particular evaluation, i.e., M3 takes into consideration both class and property matches. The fact that *KGMatcher* does not match property justifies the negative impact of the undiscovered property alignments on the matcher performance on this task. Further, given that the Conference track datasets do not include enough number of instances to apply the instance-based matcher, the name matcher is the only matcher applied to map classes. In terms of the new experimental cross-domain test case of mapping DBpedia and OntoFram, *KGMatcher* performance (0.55) better than both baselines *StringEquiv* and *edna* which have the scores 0.42 and 0.45 respectively.

2.2 Knowledge Graph

In the Knowledge Graph track, *KGMatcher* was able to generate results for all the 5 test cases at both classes and instances level only. In terms of class matching, the matcher yields satisfactory results, with 0.79 for F1 score. The added instance matcher has positively impacted the overall matcher result on this task, with a precision of 0.94, a recall of 0.66 and F1 of 0.82.

2.3 Common Knowledge Graphs

Along with other 6 matchers, *KGMatcher* was able to complete the task of matching the classes from two cross-domain and common KGs. *KGMatcher* obtained a precision of 0.97 and a recall of 0.91. With an F1 score of 0.94, *KGMatcher* is the best performing matcher on this track.

3 General comments

The results of *KGMatcher* has been very encouraging. In the common knowledge graph track, it achieves outstanding results. This indicates that our hybrid approach, utilizing instances data to map KG classes, is able to outperform systems that use other matchers' combination. It is important to note that the performance of *KGMatcher* instance-based component depends on the dataset nature. Since *KGMatcher* is learning KG classifiers by using general pretrained word embedding models, the more representative the KG instances of real-world entities, the better are the instance classification results. Figure 2 shows the different between the performance when classifying instances from common KGs, e.g., NELL, compared to a single domain KG from the knowledge graph track. Note that the latter mainly annotates classes in the entertainment domain [6].

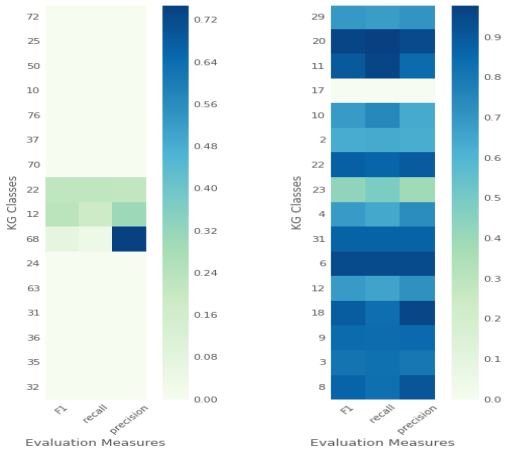

Fig. 2: The instance classification report of a 20 randomly sampled classes from the OAEI KG MemoryAlpha in (a) and NELL in (b). Note that y-axis numbers indicate class IDs.

In future versions of *KGMatcher*, we would like to improve the combination of techniques used within the name matcher. Currently, this component is rather

simple and unable to discover matching pairs with high lexical complexity. This has also affected the matcher's performance on datasets where instance data are not existing or difficult to classify. Additionally, improving the instance-based matcher by further studying other sampling approaches and experimenting with other machine learning methods, would likely improve the overall performance of the matcher.

4 Conclusion

As part of OAEI 2021, this paper presents *KGMatcher*, a matching tool that utilizes instance data annotated within large KGs to map their classes. The process is done by learning KG classifiers, which are able to classify instances into a particular KG class. The results suggest that a hybrid approach that incorporate an instance-based technique can be highly effective for matching large cross-domain KGs.

References

1. Bizer, C., Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mende, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web pp. 1–5 (2012)
2. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of machine learning research **12**(ARTICLE), 2493–2537 (2011)
3. Fallatah, O., Zhang, Z., Hopfgartner, F.: A gold standard dataset for large knowledge graphs matching. In: OM@ISWC (2020)
4. Fallatah, O., Zhang, Z., Hopfgartner, F.: A hybrid approach for large knowledge graphs matching. In: OM@ISWC (2021)
5. Gulić, M., Vrdoljak, B., Banek, M.: Cromatcher: An ontology matching system based on automated weighted aggregation and iterative final alignment. Journal of Web Semantics **41**, 50–71 (2016)
6. Hertling, S., Paulheim, H.: The knowledge graph track at oaei. In: European Semantic Web Conference. pp. 343–359. Springer (2020)
7. Hertling, S., Portisch, J., Paulheim, H.: MELT - matching evaluation toolkit. In: Semantic Systems. The Power of AI and Knowledge Graphs - 15th International Conference. pp. 231–245 (2019)
8. Maiya, A.S.: ktrain: A low-code library for augmented machine learning. arXiv preprint arXiv:2004.10703 (2020)
9. Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J.: Deep learning based text classification: A comprehensive review. arXiv preprint arXiv:2004.03705 (2020)
10. Otero-Cerdeira, L., Rodríguez-Martínez, F.J., Gómez-Rodríguez, A.: Ontology matching: A literature review. Expert Systems with Applications pp. 949–971 (2015)
11. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval, vol. 39. Cambridge University Press Cambridge (2008)
12. Tanon, T.P., Weikum, G., Suchanek, F.: Yago 4: A reason-able knowledge base. In: European Semantic Web Conference. pp. 583–596. Springer (2020)