The Impact of Imbalanced Class Distribution on Knowledge Graphs Matching

Omaima Fallatah[1,2], Ziqi Zhang[1] and Frank Hopfgartner[3]

[1]Information School, The University of Sheffield, Regent Court, 211 Portobello, Sheffield S1 4DP, UK
[2]Information Systems, Umm Al Qura University, Mecca 24382, Saudi Arabia
[3]Universität Koblenz-Landau, Mainz 55118, Germany

Abstract

Mapping large Knowledge Graphs (KGs) has been a fundamental problem in the semantic web community. Many state-of-the-art methods are not suitable for matching cross-domain, large, and automatically constructed KGs that often suffer from highly imbalanced class distribution. Therefore, recent studies have revisited instance-based matching techniques in addressing this task. This is because such large KGs often lack a well-defined structure and descriptive metadata about their classes, but contain numerous class instances. In this work, we study the problem of imbalanced class distribution in large KG schema matching using instance-based methods. Building on a state-of-the-art method reported in the 2021 OAEI common knowledge graphs track, we study different resampling techniques and propose a new method to address class imbalance in the matching task. We show that our method improves state-of-the-art by up to 11% of recall and 4% in terms of recall. In addition, this work also produces a new public gold standard dataset for mapping large KG classes with over 300 class links, and is by far the largest domain-independent dataset for KG schema matching.

Keywords

Knowledge Graphs Matching, Instance-based Matching, Ontology Matching.

1. Introduction

Over the last decade, there has been a significant growth in the creation and application of knowledge graphs. Knowledge Graph is a unique data structure for representing real-world entities in a structured and connected fashion [1]. With their potential in a wide range of downstream applications such as query answering, recommendation systems and semantic search [2], they are utilized by large companies such as Google, Facebook and Microsoft. Besides such proprietary KGs, there are a number of large common KGs available, including YAGO and Wikidata. Such cross-domain KGs are known for sharing heterogeneous yet highly complementary facts.

Despite the growth in such large-scale KGs, one problem is dealing with the quality of the data generated automatically. This has resulted in continuous efforts to facilitate refining their entities by increasing their coverage (i.e., completion), and detecting errors (i.e., correctness) [3].

OM2022: the International Workshop on Ontology Matching, October 23, 2022, Hangzhou, China
oafallatah1@sheffield.ac.uk (O. Fallatah); ziqi.zhang@sheffield.ac.uk (Z. Zhang); hopfgartner@uni-koblenz.de (F. Hopfgartner)
0000-0002-5466-9119 (O. Fallatah); 0000-0002-8587-8618 (Z. Zhang); 0000-0003-0380-6088 (F. Hopfgartner)

To achieve this, mapping and aligning KGs at both entity and schema level is crucial. Mapping and aligning KG entities has been a significant challenge in the semantic web community. The Ontology Alignment Evaluation Initiative (OAEI[1]) has two tracks dedicated to KGs, including one that particularly evaluates matching common KGs[2]. However, most matchers participating on this task are aimed at matching well-formed ontologies, which is not necessarily the case for cross-domain and semi- or fully-automatically generated KGs [4]. Despite the large number of matching tools, matching the schema of large-scale KGs is far from trivial. Current schema matching systems suffer in terms of balancing between efficiency and effectiveness to solve the task [5]. Recent studies have shown significant improvements over state-of-the-art systems by exploiting an instance-based approach for matching the schema, i.e., classes and properties, of KGs [6, 7]. However, the problem of unbalanced class distribution and particularly under-represented classes remain challenging for instance-based approaches.

In this work, we address this issue by introducing a new method for matching classes from large KGs. Our previous work [8] introduced KGMatcher, an instance-based method that achieved the best results in the recent OAEI 2021 common KG track [6]. The method adopts a data-driven approach where a two-way classification technique is followed to map classes from two KGs based on the extent to which the instances of a class in one KG are classified as instances of classes in another KG. First, a multi-class classifier is trained using instances of classes from each KG. Next, those classifiers are then applied to the other KG to classify its instances. Mapping pairs of classes are then derived based on the classification results of the two classifiers.

While our method achieved good results, it still suffered from unbalanced class distribution. To address this issue, in this work, we look into various sampling techniques and propose a combined approach of over- and under-sampling in classifier training. We call this improved version of our method, KGMatcher+. Specifically, we adapt the sampling component of KGMatcher to better handle the imbalance problem. We introduce and compare six different resampling strategies. We evaluate KGMatcher+ on two large datasets, including the dataset from OAEI common KG track and a new gold standard we make public as part of this work. We show that KGMatcher+ achieves the best results compared to OAEI 2021 participants.

Although many solutions for dataset imbalance has been introduced, we identify that there is no research on addressing the class imbalance issue in KG matching, or even ontology matching in general. Previous research in the context of classification tasks has shown that there is often no one-size-fit-all method, and thus previous findings may not generalise to this task. Combined with the increasing research in KG matching and popularity in instance-based techniques, we argue that it is imperative to further investigate, and develop methods to address the issue of imbalanced distribution in the KG matching task. Section 2 details KGMatcher+. Then, in Section 3 and Section 4 we introduce the datasets, and experiment results. Finally, Section 5 concludes this work.

[1] https://oaei.ontologymatching.org/
[2] http://oaei.ontologymatching.org/2021/

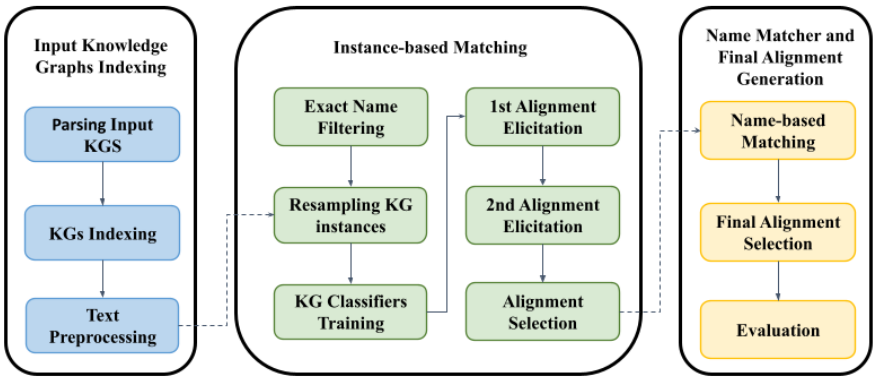


Figure 1: Overview of KGMatcher+ architecture.

2. Approach

Here, we introduce KGMatcher+, specifically focusing on balancing class distribution for matching KG classes. We first give an overview of it in Section 2.1 (readers interested in the details may refer to [8], which is the basis of KGMatcher+). Then, in Section 2.2, we describe how our method uses resampling techniques to address the data imbalance issue.

2.1. Overview of KGMatcher+

2.1.1. Preliminaries

Given two input knowledge graphs $\mathcal{KG}$ and $\mathcal{KG}'$, we define the correspondence between two classes $\mathcal{C} \in \mathcal{KG}$ and $\mathcal{C}' \in \mathcal{KG}'$ as the tuple $< \mathcal{C}, \mathcal{C}', v >$ where $v \in [0, 1]$ is the similarity value of $\mathcal{C}$ and $\mathcal{C}'$. Each class in the two KGs has a set of instances, $\mathcal{C}_n = \{i_0, i_1, i_2..., i_n\}$ and ${\mathcal{C}'}_m = \{i_0, i_1, i_2..., i_m\}$. The following sections describe different modules of the matcher, as illustrated in Figure 1.

2.1.2. Input Knowledge Graph Indexing

The first component of the matcher consists of three steps. It starts by parsing the two input KGs in order to extract and then separately index their lexical annotations. This is followed by text preprocessing to normalise entity labels (e.g., lowercasing, stopwords removal). Preprocessing also separates multi-word entities such as `creativeworkseries` and `dancegroup` by using a word segmentation algorithm based on a dictionary.

2.1.3. Instance-based Matching

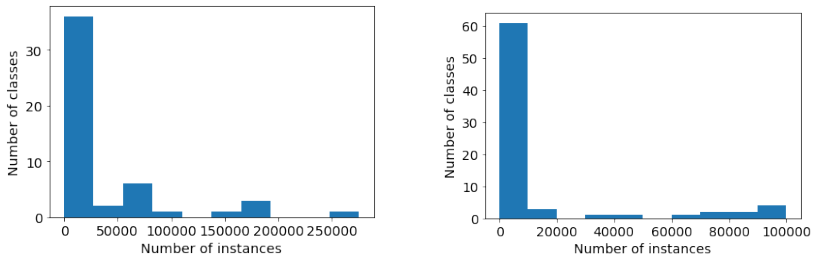
In a two-way classification fashion, this matching component is split into two matching processes. Each is based on generating a multi-class classifier for one of the input KGs using its instance names as training data and class names as classification labels. Therefore, a classifier

trained on $\mathcal{KG}$ instances data will be able to to predict the class C to which a given 'instance' name may belong. In the following, we briefly describe the steps of the instance-based matcher.

- Exact name filter: working as a blocking strategy, this removes class pairs that share exactly the same labels from $\mathcal{KG}$ and $\mathcal{KG}'$. Further, classes with only one instance are eliminated from this process, as our BERT-based classifier will be unable to learn from them.
- Resampling KG instances: the resampling phase is based on applying a data balancing technique in order for the matcher to cope with the imbalance distribution of instances across KG classes. In section 2.2, we discuss different data imbalance solutions that we implement as part of KGMatcher+.
- Instance Classification: here we build a multi-class classifier for $\mathcal{KG}$ and $\mathcal{KG}'$. Class instances are split into 85% for training and 15% for the purpose of evaluating the classifier, which we use a simple BERT-based sequence classifier [9]. The two classifiers $\mathcal{CL}_{\mathcal{KG}}$ and $\mathcal{CL}_{\mathcal{KG}'}$ will be utilized in the two following steps of the matcher.
- Alignment Elicitation: here we derive class mapping candidates based on the classification results. First, in the direction $\mathcal{KG} \rightarrow \mathcal{KG}'$ to generate mapping candidates denoted as $A_{\mathcal{KG}\rightarrow\mathcal{KG}'}$, $\mathcal{KG}$ will be treated as the source KG and $\mathcal{KG}'$ as the target. Subsequently, $\mathcal{CL}_{\mathcal{KG}}$, is applied to all instance names in $\mathcal{KG}'$. By taking the class with the highest probability value, each instance in $\mathcal{KG}'$ will have a predicted class in $\mathcal{KG}$. To generate $A_{\mathcal{KG}\rightarrow\mathcal{KG}'}$. We pair each class $\mathcal{C}'$ with the class $\mathcal{C}$ that receive the most votes, based on applying $\mathcal{CL}_{\mathcal{KG}}$ to instances of $\mathcal{C}'$. As an example, the pair $< \mathcal{C}_4, \mathcal{C}'_2, 0.57 >$ means that 57% of $\mathcal{C}'_2$ instances were predicted to be $\mathcal{C}_4$ when applied to $\mathcal{CL}_{\mathcal{KG}}$. The second elicitation process is done in the opposite direction, repeating the same procedure to create $A_{\mathcal{KG}'\rightarrow\mathcal{KG}}$. The two resulting alignment sets are to be combined in the final step of the instance-based matcher.
- Alignment Selection: $A_{\mathcal{KG}\rightarrow\mathcal{KG}'}$ and $A_{\mathcal{KG}'\rightarrow\mathcal{KG}}$ are unified at this stage to generate one alignment set for the instance-based matcher. Specifically, we use the approach in [10], which firstly creates a $\mathcal{N} \times \mathcal{M}$ table (where $\mathcal{N}$ is the number of classes in $\mathcal{KG}$ and $\mathcal{M}$ the number of classes in $\mathcal{KG}'$), and populates the table based on the two directional alignment sets created before. The algorithm then identifies the highest value in the table cells as the final alignment candidates, and then deletes the corresponding row/column from the table. The process is repeated until all rows/columns are deleted. For further details of this algorithm, please refer to [10].

2.1.4. Name Matcher and Final Alignment Generation

In this step, the terminological similarity of the KG class names is measured to be combined with their instance-based similarity. This matcher combines two similarity measures: one focuses on the string similarity, while the other focuses on semantic similarity. For string similarity, we use the normalised Levenshtein distance. For semantic similarity, class names are represented using a pretrained word2vec model, and then the cosine similarity is calculated. For each pair, the higher value of the two similarity measures is chosen as that pair's similarity value, while higher than a threshold of 0.8. To generate the final alignments of KGMatcher+, the instance-based



(a) YAGO (b) Wikidata

Figure 2: The class distribution of two KGs in the YAGO-Wikidata datasets after applying the exact name filter.

alignments are combined with the name matcher alignments. This is done by following the same alignment selection method used earlier, while treating each as a directional alignment.

2.2. Addressing Imbalanced Data Distribution

In the instance-based matching stage, the training data, i.e., KG class instances, are typically imbalanced. Figure 2 depicts the distribution of instances in two KGs related to our datasets in Section 4. In the following, we introduce six different sampling strategies that make the key components of KGMatcher+. When each strategy is used instead of others, we denote that version of KGMatcher+ as KGMatcher+{SS}, where SS indicates the corresponding Sampling Strategy. Since we are dealing with a classification problem, we can resort to popular methods used for dealing with imbalanced training data in classification. As the goal of data balancing technique is to decrease the bias of classifiers towards majority classes at the expense of the minority classes, we need to define both in the context of KGs. However, there is a lack of consensus on how majority/minority classes are defined in multi-classification settings. Here, we adopt an approach where we firstly calculate the average number of instances per class within a KG, and then classes with fewer instances than this number are treated as minority classes and those above it as majority classes.

2.2.1. KGMatcher+ {Random Undersampling}

In binary classification, this strategy indicates excluding data samples from the majority class to match the size of the minority class. This is a common strategy seen in the literature [11]. Similarly, in multi-class classification tasks, this method is independently applied to each class by randomly sampling an equal sample size of all classes. The sample size matches the size of the class with the least data samples [12].

2.2.2. KGMatcher+ {TF-IDF Undersampling}

As opposed to random undersampling, which randomly discards instances from the majority classes and can result in losing potentially useful samples, this method uses TF-IDF [13] to measure the 'importance' of samples and select them based on this score. This method was first

introduced in our earlier work [8] and used as the only sampling component in KGMatcher [6]. TF-IDF undersampling is applied to majority classes. Briefly, we calculate the TF-IDF of the words from KG instances. Similar to applying TF-IDF for information retrieval tasks, each class here is treated as a 'document' and the concatenation of the labels of its instances is treated as its content. Then, a word with high TF-IDF to a certain class indicates it is more specific to that class. The highest ranked ten words per class are then used to undersample instances in the majority classes by discarding instance names that do not contain any of these words. Although this method is effective in downsampling the majority classes while maintaining the integrity of the data, the problem with classes with fewer instances remains unresolved.

2.2.3. KGMatcher+ {SMOTE}

Synthetic Minority Over-sampling Technique [14] is the most common oversampling method applied in the literature to handle imbalanced data. It randomly oversamples the minority classes by generating syntactic data for each minority class. The algorithm uses the K-nearest neighbours to current instances in a minority class to introduce new synthetic samples from neighbouring samples [15]. This technique is considered as an alternative to random oversampling, which is a non-heuristic approach that balances classes by duplicating the samples in the minority classes to match the size of the largest majority class [12]. However, it is known that random oversampling often leads to model overfitting [16]. Another reason for excluding random oversampling is the severe class imbalance ratio, e.g., in YAGO the smallest class has one instance while some classes have over 100,000 instances as depicted in Figure 2. Random oversampling will produce overwhelmingly redundant instances for small classes, making overfitting much worse.

2.2.4. KGMatcher+ {TF-IDF + Oversampling}

Combining undersampling and oversampling strategies is another approach to handle imbalanced datasets. Following such a hybrid strategy has shown to improve the results of several classification tasks [16, 17]. While earlier work already experimented with other variations of this idea, here we propose a novel method that combines TF-IDF undersampling with random oversampling. We aim to maintain a trade-off between handling the imbalance issue in both majority and minority classes. After applying the TF-IDF undersampling to the majority classes, we apply oversampling to make each class equal-size in terms of their instances. This includes creating repeated samples from minority classes.

2.2.5. KGMatcher+ {TF-IDF + SMOTE}

This strategy is similar to the previous one. However, instead of random oversampling, here, SMOTE is applied as an oversampling technique to handle the minority classes.

2.2.6. KGMatcher+ {Cost-based Learning}

All previous strategies belong to the category of data-level methods, often applied to the datasets prior to training a model. Another type of strategy (i.e., 'algorithm level') aims to modify existing

machine learning models in an effort to reduce their bias towards the majority classes [18]. A common algorithm-level approach is cost sensitive learning [19] which modifies the class weights by assigning larger weights to minority class(es) and smaller weights for majority class(es) to be used during the model learning process. In this work, we evaluate a state-of-the-art approach [20], which gives each class a weight that is equal to its total number of instances divided by the distribution of instances across all classes as depicted in Equation 1, where *dict* is a dictionary of classes and their assigned weights, $bincount(\mathcal{C})$ is umber of instances in the class $\mathcal{C}$.

$$Class\_Weight_{dict} = n_{samples}/(n_{classes} * bincount(\mathcal{C})) \quad (1)$$

3. Datasets

Our first dataset is ***NELL-DBpedia***, which is the OAEI common knowledge graphs dataset. This dataset contains a gold standard mapping of 129 pairs of classes between NELL and DBpedia. They each contain an average of 8,000 and 4,000 instances per class respectively. The second dataset is ***YAGO-Wikidata*** created based on [21]. It specifically maps the Schema.org classes to Wikidata's schema. Different from the original dataset, which only includes the class alignment, we refer to this dataset as YAGO-Wikidata because we retrieved the instances of Schema.org classes from YAGO [3]. The original gold standard includes over 500 mappings, however not all of them are equivalence. For the purpose of our task, and given that the majority of studies on mapping KGs only consider equivalence matches, we only include mappings annotated with the relationship `equivClass`. As a result, the new dataset contains 304 equivalent class pairs. Further, since Wikidata's entities are often represented by their Q indices, e.g., Q1234, we use the Wikidata python API to query their URIs in order to retrieve their labels. The same API was then used to generate a subgraph of Wikidata that includes all the 304 classes and their annotated instances. Similarly, we use YAGO's SPARQL query endpoint to retrieve all schema and instances metadata that are connected to the 304 classes included in the original dataset alignments. On average, this dataset includes over 33,000 and 12,000 instances per class in YAGO and Wikidata respectively. We make the new dataset publicly available in [4]. This includes the two subgraphs in rdf/xml format and the alignments file according to OAEI's standards.

4. Evaluation and Comparative Study

In Section 4.1, we first compare the different sampling strategies in KGMatcher+; in Section 4.2, we compare the results of KGMatcher+ against state-of-the-art methods for mapping KGs. Similar to OAEI standards, we use precision, recall, F-measure to evaluate the accuracy of the resulted alignments. All systems are implemented using the Matching Evaluation Toolkit (MELT)[5], which is also used for recent OAEI campaigns. Our experiments have been executed on a VM with 128 GB of RAM, with 2.4 GHz 16 vCPUs, and a 12 GB GPU.

[3] YAGO 4 https://yago-knowledge.org/downloads/yago-4
[4] https://github.com/OmaimaFallatah/YagoWikiData
[5] https://github.com/dwslab/melt

Table 1
Evaluation results of different variations of KGMatcher+ each utilizing a different data balancing strategy on two KGs datasets.

Resampling Technique	YAGO-Wikidata			NELL-DBpedia		
	P	R	F1	P	R	F1
No Sampling	0.97	0.79	0.87	0.97	0.91	0.94
Random Undersampling	0.98	0.75	0.85	0.98	0.81	0.89
TF-IDF Undersampling	0.96	0.80	0.87	0.97	0.91	0.94
TF-IDF + Oversampling	**0.99**	**0.84**	**0.91**	**0.98**	**0.91**	**0.95**
SMOTE	0.97	0.78	0.86	0.98	0.89	0.93
TF-IDF + SMOTE	0.99	0.80	0.88	0.97	0.91	0.94
Cost-based learning	0.96	0.77	0.85	0.96	0.85	0.90

4.1. Impact of sampling strategies on KGMatcher+

Table 1 shows the precision, recall, and F_1 of KGMatcher+ when using different sampling strategies. As the table shows, in terms of F_1, KGMatcher+{TF-IDF + oversampling} outperforms all other variations with (F_1=0.91) on the YAGO-Wikidata dataset and (F_1=0.95) on the NELL-DBpedia dataset. In terms of undersampling strategies, KGMatcher+{Random Undersampling} fails to improve the overall results on both datasets compared to the results obtained when no sampling was applied. On the other hand, while KGMatcher+{TF-IDF undersampling} does leave the matching results on both datasets unchanged, compared to no sampling, it maintains the same performance while significantly decreasing the matcher processing time (from 55 minutes to 29 minutes on the NELL-DBpedia dataset and from 3 hours to 1.5 on the YAGO-Wikidata). The latter strategy represents our earlier work, KGMatcher [6].

There is a gap in the performance of the two undersampling strategies, as the first one randomly eliminates samples from KG classes. Further, with this strategy, instance samples are reduced to match the size of the class with the least samples, which can be less than 10 instances in some common KGs. This rather aggressive reduction in training data could have badly affected the classifier training. However, using TF-IDF to downsample classes in KGMatcher+{TF-IDF undersampling} does not negatively impact the results, as the elimination process maintains instances with indicative words.

In terms of oversampling strategies, KGMatcher+{SMOTE} decreases the recall on both datasets, which subsequently affects the F_1 score as well. Further, KGMatcher+ {TF-IDF + SMOTE} shows results similar to the best performing strategy on the NELL-DBpedia dataset, i.e., KGMatcher+{TF-IDF + Oversampling}. Nonetheless, this strategy does not perform as well on the YAGO-Wikidata dataset, which is twice the size of the NELL-DBpedia. This shows that combining both undersampling and oversampling is the best strategy for this task. Even though class distribution in the KGs used in the experiments was severe, undersampling the majority classes with TF-IDF helps mitigate this issue. In contrast, generating synthetic data from KG instances seems to introduce noisy samples to the dataset, as indicated by the result of KGMatcher+{SMOTE}. This seems to be consistent with previously reported findings in text

Table 2
Comparison against state-of-the-art methods

Baseline	YAGO-Wikidata			NELL-DBpedia		
	P	R	F1	P	R	F1
atmatcher [22]	**1.00**	0.77	0.87	**1.00**	0.80	0.89
Wiktionary [23]	**1.00**	0.74	0.85	**1.00**	0.80	0.89
LogMap [24]	0.98	0.74	0.84	0.99	0.80	0.88
ALOD2Vec [25]	0.99	0.77	0.86	**1.00**	0.80	0.89
AML [26]	**1.00**	0.80	0.89	**1.00**	0.80	0.89
KGMatcher+	0.99	**0.84**	**0.91**	0.98	**0.91**	**0.95**

classification tasks [15].

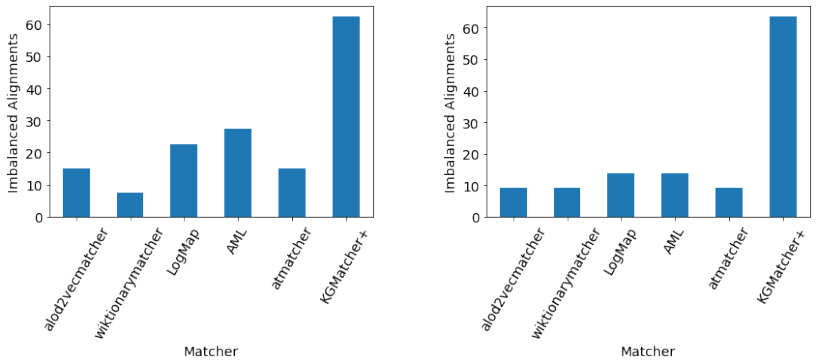
The final data balancing strategy is KGMatcher+ {Cost-based learning}, which adapts the BERT model to handle class imbalance by using class weights. Although the main advantage of this method is to maintain the integrity of the datasets, this did not work well as it achieved the worst precision, recall, and F-measure, which are even lower than the model using no data balancing strategies at all. This is due to over penalising the classifier for incorrectly classifying instances from the minority classes.

4.2. Comparison to State-of-the-art

Table 2 depicts the performance of KGMatcher+{TF-IDF + Oversampling} against multiple OAEI's best performing matchers in the two KG tracks. We can observe that the proposed matcher outperforms all baselines on both datasets, recording the highest recall and F_1. In terms of the NELL-DBpedia dataset, KGMatcher+ outperforms all matchers by a minimum of 6% in F_1 score and by 11% in recall.

On the YAGO-Wikidata dataset, which has worse class imbalance compared to NELL-DBpedia, KGMatcher+ beats all matchers with 4% in terms of recall and 2% in F_1. One can also notice that all matchers score lower on YAGO-Wikidata. This is likely due to the size of YAGO-Wikidata, which is twice the size of NELL-DBpedia. Readers should also note that while all matchers were able to generate class alignments when applied to the full version of YAGO-Wikidata, AML and LogMap were only able to process a smaller version of the dataset, with a small subset of instances per class. Also, both do not utilize instances during the matching process.

In order to analyse the ability of different matching methods to discover alignments containing very imbalanced classes, we conducted a quantitative study of imbalanced pairs in the two datasets. We define a pair $(\mathcal{C}, \mathcal{C}')$ as an imbalance class pair, if one of the classes is a majority class and the other is a minority class, or if both classes are considered as minority classes. We counted around 40 imbalanced pairs in YAGO-Wikidata and 22 in NELL-DBpedia. Figure 3 shows the number of discovered imbalanced alignments by different methods compared. On both datasets, KGMatcher+ was able to discover over 60% of such imbalanced pairs (64% in NELL-DBpedia and 63% in YAGO-Wikidata). On the YAGO-Wikidata, for instance, KGMatchers+ discovered 25 out of the 40 imbalanced pairs, while the next best systems (AML and LogMap)



(a) YAGO-Wikidata (b) NELL-DBpedia

Figure 3: The number of imbalanced class pairs discovered by different methods compared to KGMatcher+{TF-IDF + Oversampling}.

Table 3
Ablation study of KGMatcher+{TF-IDF + Oversampling}

Method	YAGO-Wikidata			NELL-DBpedia		
	P	R	F1	P	R	F1
KGMatcher+	**0.99**	**0.84**	**0.91**	**0.98**	**0.91**	**0.95**
-w.o. resampling	0.97	0.79	0.87	0.97	0.91	0.94
-w.o. instance-based	1.00	0.74	0.85	1.00	0.80	0.89

discovered only 11 and 9 pairs respectively. The results indicate that mapping the schema of large and common KGs is not a trivial task and needs to be carefully handled.

4.3. Ablation Study

Table 3 presents an ablation study for KGMatcher+ on the two datasets. In particular, we look at the effects of the sampling strategy, and the name matcher. In terms of resampling, adapting a resampling strategy does not only improve the processing time as discussed earlier, but also positively affected precision, recall and F-measure on both datasets. This is an inspiring achievement, as the majority of undersampling methods often have a negative effect on the learning process [15]. Regarding the name matcher, combining it with the instance-based method has improved KGMatcher+ results by increasing the recall on both datasets while maintaining a good balance between precision and recall. While the terminological method utilized by KGMatcher+ is the basic edit distance combined with a word embedding based similarity metric, it achieved similar performance to state-of-the-art methods that utilize more complex terminological and structural techniques as shown in Table 2. This further demonstrates that large-scale and cross domain KGs are very different from conventional ontologies, and therefore require more tailored solutions.

5. Conclusion

In this work, we introduced KGMatcher+ which specifically addresses the problem of imbalanced class distribution in the task of matching the classes of common large KGs. To the best of our knowledge, there is a lack of studies in this direction, and our work is the first that addresses this problem in the context of KG matching. We experimented with different sampling strategies, including one that is newly proposed in this work. We show that combining TF-IDF undersampling and oversampling techniques outperforms other strategies. Our work provides empirical reference for future research on large KG matching, which is an increasing challenge due to the typical class imbalance issues. Our future work will expand KGMatcher+ to map KGs properties and utilize the results of schema matching to align KG instances.

References

[1] N. Heist, S. Hertling, D. Ringler, H. Paulheim, Knowledge graphs on the web – an overview, 2020.

[2] D. Obraczka, J. Schuchart, E. Rahm, Embedding-assisted entity resolution for knowledge graphs (2021).

[3] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semantic web 8 (2017) 489–508.

[4] O. Fallatah, Z. Zhang, F. Hopfgartner, A gold standard dataset for large knowledge graphs matching, in: Ontology Matching 2020: Proceedings of the 15th International Workshop on Ontology Matching co-located with (ISWC 2020), 2020.

[5] E. Rahm, E. Peukert, Large-scale schema matching., Encyclopedia of Big Data Technologies, 2019.

[6] O. Fallatah, Z. Zhang, F. Hopfgartner, Kgmatcher results for oaei 2021, in: Proceedings of the 16th International Workshop on Ontology Matching co-located with the 20th International Semantic Web Conference (ISWC), volume 3063, 2022.

[7] D. Ayala, I. Hernández, D. Ruiz, E. Rahm, Towards the smart use of embedding and instance features for property matching, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), IEEE, 2021, pp. 2111–2116.

[8] O. Fallatah, Z. Zhang, F. Hopfgartner, A hybrid approach for large knowledge graphs matching, in: Proceedings of the 16th International Workshop on Ontology Matching, 2021.

[9] A. S. Maiya, ktrain: A low-code library for augmented machine learning, arXiv preprint arXiv:2004.10703 (2020).

[10] M. Gulić, B. Vrdoljak, M. Vuković, An iterative automatic final alignment method in the ontology matching system, Journal of Information and Organizational Sciences 42 (2018) 39–61.

[11] X.-Y. Liu, J. Wu, Z.-H. Zhou, Exploratory undersampling for class-imbalance learning, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39 (2008) 539–550.

[12] G. Lemaître, F. Nogueira, C. K. Aridas, Imbalanced-learn: A python toolbox to tackle

the curse of imbalanced datasets in machine learning, The Journal of Machine Learning Research 18 (2017) 559–563.
[13] J. Ramos, et al., Using tf-idf to determine word relevance in document queries, in: Proceedings of the first instructional conference on machine learning, volume 242, Citeseer, 2003, pp. 29–48.
[14] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, Smote: synthetic minority over-sampling technique, Journal of artificial intelligence research 16 (2002) 321–357.
[15] C. Padurariu, M. E. Breaban, Dealing with data imbalance in text classification, Procedia Computer Science 159 (2019) 736–745.
[16] B. Krawczyk, Learning from imbalanced data: open challenges and future directions, Progress in Artificial Intelligence 5 (2016) 221–232.
[17] H. Feng, W. Qin, H. Wang, Y. Li, G. Hu, A combination of resampling and ensemble method for text classification on imbalanced data, in: International Conference on Big Data, Springer, 2021, pp. 3–16.
[18] Z. Liu, W. Cao, Z. Gao, J. Bian, H. Chen, Y. Chang, T.-Y. Liu, Self-paced ensemble for highly imbalanced massive data classification, in: 2020 IEEE 36th international conference on data engineering (ICDE), IEEE, 2020, pp. 841–852.
[19] C. Elkan, The foundations of cost-sensitive learning, in: International joint conference on artificial intelligence, volume 17, 2001, pp. 973–978.
[20] G. King, L. Zeng, Logistic regression in rare events data, Political analysis 9 (2001) 137–163.
[21] P. Krauss, schemaorg-wikidata-map, https://github.com/okfn-brasil/schemaOrg-Wikidata-Map, 2017.
[22] H. Sven, P. Heiko, Atbox results for oaei 2021, in: Proceedings of the 16th International Workshop on Ontology Matching co-located with the 20th International Semantic Web Conference (ISWC), volume 3063, 2022.
[23] J. Portisch, H. Paulheim, Wiktionary matcher results for oaei 2021, in: Proceedings of the 16th International Workshop on Ontology Matching, volume 3063, 2022, pp. 199–206.
[24] E. Jiménez-Ruiz, Logmap family participation in the oaei 2020, in: Proceedings of the 15th International Workshop on Ontology Matching (OM 2020), volume 2788, CEUR-WS, 2020, pp. 201–203.
[25] J. Portisch, H. Paulheim, Alod2vec matcher results for oaei 2021, in: Proceedings of the 16th International Workshop on Ontology Matching, volume 3063, 2022.
[26] D. Faria, C. Pesquita, T. Tervo, F. M. Couto, I. F. Cruz, Aml and amlc results for oaei 2019, in: Proceedings of the 16th International Workshop on Ontology Matching co-located with the 20th International Semantic Web Conference (ISWC), volume 3063, 2022.