A Hybrid Approach for Large Knowledge Graphs Matching

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. Matching large and heterogeneous Knowledge Graphs (KGs) has been a challenge in the Semantic Web research community. This work highlights a number of limitations with current matching methods, such as: (1) they are highly dependent on string-based similarity measures, and (2) they are primarily built to handle well-formed ontologies. These features make them unsuitable for large, (semi-) automatically constructed KGs with hundreds of classes and millions of instances. Such KGs share a remarkable number of complementary facts, often described using different vocabulary. Inspired by the role of instances in large-scale KGs, we propose a hybrid matching approach. Our method composes an instance-based matcher that casts the schema matching process as a two-way text classification task by exploiting instances of KG classes, and a string-based matcher. Our method is domain-independent and is able to handle KG classes with unbalanced population. Our evaluation on a real-world KG dataset shows that our method obtains the highest recall and F1 over all OAEI 2020 participants.

Keywords: Knowledge Graphs · Machine Learning · Schema Matching.

1 Introduction

In recent years, many public Knowledge Graphs (KGs) have been developed and shared, e.g., DBpedia [1] and NELL [2]. Common KGs are often domain-independent and semi-automatically constructed. Common KGs are highly complementary, therefore, they are often integrated in several web applications such as reasoning and query answering.

KGs have gained more attention in the Semantic Web, which facilitates sharing and reusing knowledge such as those annotated in ontologies. Similar to ontologies, KG entities are highly heterogeneous, since many real-word entities can be described using different vocabulary. Nevertheless, while ontologies primarily focus on modelling the schema of a specific domain, cross-domain KGs are known for describing numerous instances. Due to their nature of being largely generated in a semi-automated manner, KGs are less well-formed compared to manually created and well-designed ontologies.

The problem of ontology matching has been well studied, and matching systems are annually evaluated through the Ontology Alignment Evaluation Initiative (OAEI [3]). A new track for matching KGs has been introduced to OAEI in 2018, where ontology matchers are evaluated on the tasks of matching classes, properties and instances. By design, KGs are known for their large number of instances (ABox). Therefore, the majority of current matchers focus on matching their instances. However, recent studies have shown that the problem of matching KGs schema (TBox) remains a challenging task [4]. Moreover, many KG matchers exploit class matches to generate and refine instance matches [9].

With current matching solutions mainly focusing on well-formed ontologies, the problem of matching automatically curated and large KGs remains significant. While the majority of the state-of-the-art methods are highly dependent on string/language and structural-based techniques [27], KGs often lack some textual descriptions, e.g., comments, required by these methods. In terms of structural-based similarity measures, despite that some KGs lack the schematic information required by such methods, they can be error-prone [19]. This justifies their high-level of dependency on string-based matchers' results.

This work proposes a novel method for mapping classes in large KGs by combining string-based measures with an instance-based method. The latter only uses annotated instance names to generate similar class pairs. Our domain-independent method utilizes the large number of instances in KGs, and is able to cope with unbalanced population of KG classes. This is particularly useful in scenarios of large KGs with rich populated instances, such as DBpedia that is the central linking dataset in the current linked data cloud, and NELL that creates a large-scale KGs in a never-ending machine reading fashion. In addition to OAEI KG benchmark, we conduct an experiment to evaluate the performance of our method on a real-world KG benchmark [4]. We compare the results of our proposed approach against the systems participated in the KG track in OAEI 2020 and show that our method obtains the highest recall and F1 measure in the task of matching common KGs classes.

The remainder of this paper is structured as follows. An overview of the related work is provided in Section 2; Section 3 describes the details of the proposed matcher; Section 4 describes our experiments and 5 discuses the results, followed by a conclusion and future work discussion in Section 6.

2 Related Work

Ontology matching systems often combine different matching techniques [21]. *Element-level* matchers discover similar entities by utilizing the textual annotations defined in the ontology's entities, e.g., URIs, labels, and comments. Other methods leverage lexical databases, such as WordNet[4], as background knowledge to discover semantic similarity. However, recent studies, such as [20], highlights that WordNet lacks sufficient coverage in comparison to word embedding based

[3] `http://oaei.ontologymatching.org/`
[4] `https://wordnet.princeton.edu/`

similarity measures. It is difficult for semantic-based techniques to outperform string-based ones, therefore, combining both measures is a common strategy [9].

Matchers such as the well-known AML [5] and LogMap [13] employ element-level techniques. Both matchers make use of background knowledge bases in order to match biomedical ontologies. Some of recent OAEI KG

6 errors6 warnings track participants have been utilizing other resources. For example, Wiktionary [24], which is an element level-matcher that uses an online lexical resource known as Wiktionary. Similarly, ALOD2Vec [23] utilizes WebIsALOD[5], an automatically generated RDF dataset of hypernym relations, as background knowledge.

In terms of *Structural-level* matchers, they exploit structural information available in well-formed ontologies like disjoint axioms to refine element-level alignments, such as in AML and LogMap matcher family. ATBox [9] is an OAEI 2020 participant which uses similar techniques to filter mappings initially discovered by a string-based matcher. Such an approach requires a well-formed ontology which is not the case in the context of common KGs that lack the schematic richness due to their automatically generated nature [4].

The final matcher category is *Extensional* or *instance-based* matchers that use instances data to generate schema level alignments. The intuition of such a method is that similar classes or properties shares a substantial overlap of their instances. However, according to [12], it is significantly challenging to measure the extension of such an overlap. Previous works that incorporate this method are predominantly domain-dependent [26]. Zhang et al.[27] introduced an instance-based approach that only matches the properties within single LOD datasets, which include some KGs such as DBpedia. Their results are encouraging to apply instance-based methods on cross-dataset settings, particularly with LOD datasets and KGs sharing many similar characteristics.

3 Approach

3.1 Overview

Our matching approach can be formalized as: the **Input** takes two KGs, $\mathcal{O}$ 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 $C^j_{\mathcal{O}'} = \{e^j_0, e^j_1, ..., e^j_m\}$. Our method is composed of an **instance-based matcher** and a **name matcher**. The architecture of the proposed method is illustrated in Figure 1.

The workflow starts with parsing the two input KGs and applying general text preprocessing (Section 3.2). The second component of the method is the matching process which starts with an instance-based matcher (Section 3.3). This matcher is performed in two stages and will result in two directional alignment sets, denoted $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ which is a set of correspondences between classes from $\mathcal{O}$ and $\mathcal{O}'$ respectively, and $A_{\mathcal{O}'\rightarrow\mathcal{O}}$ which is a set of correspondences in

[5] `http://webisa.webdatacommons.org/`

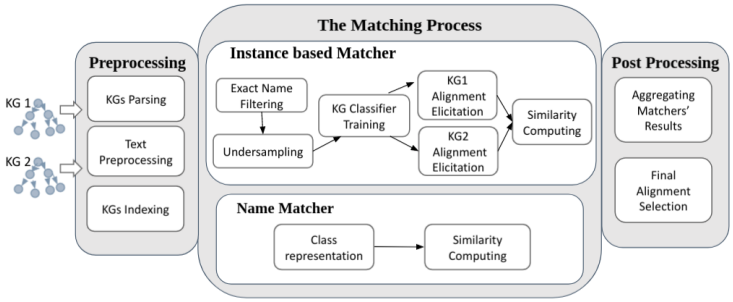

Fig. 1: The architecture of the proposed matcher

the opposite direction. Section 3.3 explains the process of aggregating the two directional alignments in order to obtain one alignment set for this matcher $A_{instance}$. The second matcher is based on string/semantic similarity, which belongs to the element-level matchers category. This matcher uses class labels to generate equivalent class pairs denoted as A_{name} (Section 3.3). The process of selecting the **final alignments** A is described in Section 3.4 (Post Processing in fig. 1).

3.2 Pre-processing

The matcher starts by parsing the two input KGs in order to separately index their lexical data structure. Given a KG, we create an index of its classes by following the standard free text indexing approach for search engines. Here, each class is treated as a document, and the text content of that 'document' is the concatenation of the labels of all the class's instance names. In order to obtain a cleaner version of the datasets, standard text preprocessing techniques are applied. All entity labels are transferred into lowercase and all stopwords and non-alphanumeric characters are removed. Finally, we replace all underscores characters, which are often used to separate multi-word entity labels, with a space character. KGs class names can also be described with multiple words, such as `placeofworship` or by using a camel case (e.g., `ReligiousBuilding`). Therefore, a word segmentation process which utilizes a dictionary is applied to infer the spaces between words while the camel case is replaced with a white space as well.

3.3 The matching process

Instance-based Matcher . This matcher uses a self-supervised approach to map KG classes based on their shared instances. The matching process is divided into a two-way classification process where a *KG classifier* is trained with one KG's instances, and then used to classify a given instance into one of the classes from that KG. As illustrated in Figure 1, this matcher starts by applying an exact name filter then undersamples the datasets from the two KGs in preparation for

the training phase. After the training process, the classifier trained on $\mathcal{O}$ (i.e., $CLS_{\mathcal{O}}$) is used to classify instances from $\mathcal{O}'$. The classification results are then used to elicit the directional alignments $A_{\mathcal{O}\to\mathcal{O}'}$. The second alignment election process is similar to the first one, except that the two KGs roles are reversed to generate $A_{\mathcal{O}'\to\mathcal{O}}$. Finally, the candidate class pairs for this matcher is generated based on the two directional alignments.

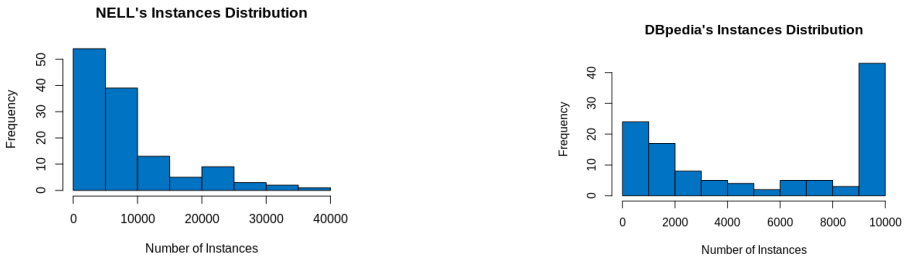

Fig. 2: The distribution of instances across classes in two common KGs

Exact Name Filtering. We start by filtering classes in both KGs with exact names. Therefore, if a class label exists in both input KGs, both classes will be excluded from the instance_based matching process. Our goal here is to use the instance-based matcher to leverage the final alignments with class pairs that are likely to not be discovered by simple string matchers. Further, this also serves as a blocking step which reduces the search space for the matcher, as large KGs can have hundreds of classes to be matched.

Undersampling. Typical large KGs are often very imbalanced. For example, Figure 2 shows the distribution of classes in two common KGs, that we will discuss later in Section 4, i.e., DBpedia[6] and NELL[7]. While some classes in NELL have over 20,000 instances, other classes have less than 10 instances. This imbalance problem can detrimentally affect the learning process and therefore, a sampling process can be useful. The problem of learning from imbalanced datasets has been a thoroughly studied field where different solutions have been developed and analyzed [22]. Solutions often require targeting the *majority classes*, i.e, classes with large number of data points, and the *minority classes*, i.e, classes with few data points [14].

While random undersampling/oversampling [15] are common practices in machine learning applications, they still carry some major limitations. Randomly undersampling the majority classes can result in losing relevant information in the eliminated samples. In contrast, random oversampling, which generates duplicate data points in minority classes, can ultimately result in model overfitting

[6] `https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10`,visited on 14-2-2020

[7] `http://rtw.ml.cmu.edu/rtw/resources`, iteration number 1115, visited on 22-2-2020

as a result of having the multiple samples of same data points. According to [6], datasets with severe class imbalance can be very challenging to train machine learning models and will often require specialized resampling techniques as opposed to generalized solutions.

Our sampling strategy aims to balance KGs instance population by undersampling the majority classes. Here, we define a *Majority* class as a class with a number of instances which exceeds the average number of instances per class in that particular KG. Due to their automated generation process, large KGs often suffer from redundant information in their instances [8]. Our goal is to obtain a smaller yet an indicative instance samples in order to limit the effect of the sparsity problem on the training/learning process.

To help identify a set of indicative instance names, we deploy a TF-IDF [25] based method to resample KG instances. TF-IDF is widely used to evaluate the relevance of words to a collection of documents by weighting their occurrences. In this task, we calculate TF-IDF of tokens for each class. Hence, the weight of a token here represents how relevant a token is to a particular class in comparison to other classes in the KG. Consequently, for each majority class, we use the top $\boldsymbol{k}$ words in terms of TF-IDF score to undersample its instance names. To illustrate, assuming that $C^i_{\mathcal{O}}$ is a majority class in $\mathcal{O}$, and $W = \{w_0, w_1, w_2..w_{\boldsymbol{k}-1}\}$, is a list of tokens with the top $\boldsymbol{k}$ TF-IDF score in $C^i_{\mathcal{O}}$. Then, we discard instance names that do not contain one of the words in W. As a result, we have a set of indicative instance names that will be used to train $CLS_{\mathcal{O}}$. The same process will be applied to the classes in the other input KG, i.e., $\mathcal{O}'$ to train $CLS_{\mathcal{O}'}$.

Training KG classifiers. A KG classifier $CLS_{\mathcal{O}}$ will be trained using the previously undersampled data. We utilize pre-trained word embeddings as features. Word Embeddings (WE) are considered one of the most effective approaches to capture the semantic similarity of words, unlike traditional feature representation methods. As classification method we experiment with two SoA methods (1) a model that uses pre-trained **BERT** model [16], and (3) a **Deep Neural Network (DNN)**. The architecture of this model is inspired by previous high performing models in different NLP tasks such as in [17] and [3].

KG1 alignment elicitation is the process of eliciting candidate directional alignment between a class in one KG (i.e., $\mathcal{O}$) and classes in another KG (e.g., $\mathcal{O}'$), based on the output of $CLS_{\mathcal{O}}$. To perform the elicitation given $C^j_{\mathcal{O}'} = \{e^j_0, e^j_1, ..., e^j_m\}$ we apply $CLS_{\mathcal{O}}$ to classify each instance of $C^j_{\mathcal{O}'}$. As a result, we classify each instance name into a class in $\mathcal{O}$. A correspondence between $C^j_{\mathcal{O}'}$ and $C^i_{\mathcal{O}}$ is added to the candidate alignments set $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ if the majority of $C^j_{\mathcal{O}'}$ instances were classified as instances of $C^i_{\mathcal{O}}$. To generate a similarity value between [0,1], we use the percentage of instances that voted for the majority class. For example, if 700 out of 1000 instances in $C^j_{\mathcal{O}'}$ were classified as, $C^i_{\mathcal{O}}$ the similarity measure of that pair will be 0.7.

KG2 alignment elicitation. Similar to KG1 alignment elicitation, we reverse the roles of the two KGs to generate $A_{\mathcal{O}'\rightarrow\mathcal{O}}$. Candidate alignments are generated based on the output of the classifier trained on $\mathcal{O}'$, i.e., $CLS_{\mathcal{O}'}$.

Similarity computing. This phase of the matcher aims to (1) combine the two directional alignments resulted from the two KGs alignment elections (i.e, $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ and $A_{\mathcal{O}'\rightarrow\mathcal{O}}$), and (2) select the final alignments for the *instance-based matcher*. In order to combine both alignments sets, each directional alignment will be first stored into an alignment matrix of a dimension of $|\mathcal{O}|.|\mathcal{O}'|$. An alignment matrix contains the correspondences for all class pairs from the two input KGs. To aggregate the two matrices, we take the average of the similarity value of each pair. For example, if ($C^4_{\mathcal{O}}$,$C^5_{\mathcal{O}'}$,0.69) in $A_{\mathcal{O}\rightarrow\mathcal{O}'}$ and ($C^5_{\mathcal{O}'}$,$C^4_{\mathcal{O}}$, 0.73) in $A_{\mathcal{O}'\rightarrow\mathcal{O}}$) their aggregated similarity value will be 0.71.

Consequently, the final alignment $A_{instance}$ is generated by following the automated alignment selection approach introduced in [7]. Given an alignment matrix and a threshold $\boldsymbol{t}$, this method goes through each row at a time and selects the maximum correspondence in each row, if the similarity value for that correspondence is beyond t (e.g., bold text in fig. 3). When a class is involved in two correspondences (e.g., $C^3_{\mathcal{O}'}$ in row 1 and 7), only the one with the higher similarity is retained (e.g., $(C^6_{\mathcal{O}}, C^3_{\mathcal{O}'}, 0.92)$), and the previously selected correspondence is deleted. This process takes places iteratively until all classes are selected with a correspondence and no changes are to be made.

	$C^0_{\mathcal{O}'}$	$C^1_{\mathcal{O}'}$	$C^2_{\mathcal{O}'}$	$C^3_{\mathcal{O}'}$	$C^4_{\mathcal{O}'}$	$C^5_{\mathcal{O}'}$	$C^6_{\mathcal{O}'}$
$C^0_{\mathcal{O}}$	0.0	0.01	0.15	**0.55**	0.0	0.22	0.1
$C^1_{\mathcal{O}}$	0.2	0.0	0.0	0.12	0.0	0.13	**0.66**
$C^2_{\mathcal{O}}$	0.11	0.14	**0.72**	0.0	0.0	0.0	0.21
$C^3_{\mathcal{O}}$	**0.88**	0.11	0.0	0.02	0.08	0.0	0.4
$C^4_{\mathcal{O}}$	0.0	0.4	0.55	0.17	0.12	**0.71**	0.0
$C^5_{\mathcal{O}}$	0.13	0.0	0.02	0.0	~~**0.19**~~	0.09	0.0
$C^6_{\mathcal{O}}$	0.0	0.0	0.0	**0.92**	0.16	0.0	0.0

$$Final\_Alignments = \{(C^1_{\mathcal{O}}, C^6_{\mathcal{O}'}, 0.66), (C^2_{\mathcal{O}}, C^2_{\mathcal{O}'}, 0.72), (C^3_{\mathcal{O}}, C^0_{\mathcal{O}'}, 0.88), (C^4_{\mathcal{O}}, C^5_{\mathcal{O}'}, 0.71), (C^6_{\mathcal{O}}, C^3_{\mathcal{O}'}, 0.92)\}$$

Fig. 3: An example of calculating the final alignment using the method in [7]

Name matcher . This matcher calculates the similarity of KG classes based on the string and the semantic similarity of their names. Given a set of all possible correspondences between classes in $\mathcal{O}$ and $\mathcal{O}'$, generated with an exclusive pairwise comparison, we measure the word embedding similarity and the edit distance similarity of the two class names. We only apply the two similarity measures to KGs class names, as not all KGs provides other longer descriptions such as comments. For the edit-distance similarity, we calculate the normalized levenshtein distance for each class pairs. This method normalizes the edit distance value by the length of the longer string to get a value between [0.0, 1.0].

In terms of the word embedding similarity, a Google pre-trained `word2vec` model is used to represent class names and measure their cosine similarities in the Vector Space Model where semantically similar words are represented closer to each other. Following the same approach in Section 3.2, concatenated strings such as `awardtrophytournament` are segmented into multiple words. Thus, in the case

of a multi-word class name, the matcher aggregates the vector representation of each word composing the class name by taking an element-wise average of the vectors of each composing word.

We then choose the maximum of the two similarity measures, if the similarity scores are higher than a threshold t_n. To illustrate, assuming that a pair of the two classes `RailwayStation` and `TrainStation` where their word embedding similarity is 0.83 and their edit distance is 0.56 we select the maximum similarity value, i.e., the word embedding similarity which is also higher than the t_n. However, if the two similarity scores of a pair are lower than the t_n, then that pair will not be added to the candidate alignment set. The output of this matcher is a candidate alignment set A_{name} to be combined with the instance matcher alignments (i.e., $A_{instance}$) to be detailed in the next section.

3.4 Final alignment selection

The above-explained similarity measures will both result in an alignment set. The goal of this final stage of the matching approach is to combine the two matchers results, as well as to select the final alignments for the complete matcher. Given A_{name} and $A_{instance}$, we follow the same method as explained before in Section 3.3 for creating $A_{instance}$ out of the two directional alignment.

4 Experiments

The aim of this experiment is to test our matching approach on the task of matching large KGs and to compare it to OAEI participants. In addition to OAEI participants, we also tested the performance of the baseline matcher KG-baselineLabel[8]. Similar to OAEI, we use precision, recall, F-measure metrics to evaluate the accuracy of the matching methods against the gold standard. The Matching Evaluation Toolkit (MELT) [11] was used to perform this evaluation. Our matcher is implemented with python and was wrapped using MELT external matchers wrapping tool. The evaluation was executed on a VM with 128GB of RAM, 16 vCPUs (2.4 GHz), and 12GB GPU.

4.1 Datasets

The datasets used for evaluation are: (1) The **NELL-DBpedia** dataset is created from the schema of two large public KGs [4]. The gold standard consists of 129 true positive class alignments between NELL and DBpedia. To the best of our knowledge, this dataset is the largest available benchmark for matching KGs classes. This dataset is domain-independent and offers a substantial number of instances, which allow for evaluating instance-based matchers. (2) The **OAEI Knowledge Graphs Benchmark**, which offers five test cases generated from eight different KGs. The largest test cases in terms of class matching have 15 and 14 positive class alignments [10]. Similar to OAEI, in this work, we evaluate and share the average results of the five tasks.

[8] `http://oaei.ontologymatching.org/2020/results/knowledgegraph/index.html`

4.2 Method's parameters

Our method has the following parameters: (1) $\boldsymbol{k}$ which is the number of words ranked by their TF-IDF scores in a class to be used for the undersampling process. In the reported results, we set $\boldsymbol{k}$ to 10. However, we have created three different configurations based on $\mathbf{k} = 5, 10, 20$. We discuss the impact of $\mathbf{k}$ value in section 5.1. (2) For the threshold value of the name matcher $\boldsymbol{t_n}$, we set $\boldsymbol{t_n}$ to 0.8 which is inline with previous element-level methods that combines multiple similarity measures such as [9,20]. (3) In the final alignment process, we apply a threshold ($\boldsymbol{t}$) of 0.22, the value recommended for this method in [7].

5 Results and discussion

As shown Table 1, on the **NELL-DBpedia** dataset, one can observe that our matcher achieves a recall and F1 that are clearly higher than all OAEI 2020 participants. Here, we report the highest result out of the three $\mathbf{k}$ value configurations, i.e., 10. However, our matcher outperforms all OAEI matchers, despite what $\mathbf{k}$ value we use. AML is the best performing matcher on the task of matching classes in OAEI KG track. However, on the task of matching common KG classes, our matcher outperforms AML with 0.06 in F1, and 0.12 in recall. In terms of **OAEI KG dataset**, our matcher does not perform as well, i.e., the F1 is 0.77. We argue that the systems that outperform our matcher on this task implement a variety of matchers that target not only the labels of KG entities, but also consider other entities' metadata. For example, AML incorporates 9 different matchers including structural matchers, and other filters to further improve the quality of the matching results.

Table 1: Performance of the OAEI 2020 participants and the proposed approach on two benchmarks. The best results on each dataset are marked in bold.

Matcher	NELL-DBpedia			OAEI benchmark		
	P	**R**	**F1**	**P**	**R**	**F1**
ALOD2Vec [23]	1.00	0.79	0.88	1.00	0.67	0.8
AML [5]	1.00	0.8	0.89	**0.98**	**0.81**	**0.89**
ATBox [9]	1.00	0.79	0.88	0.97	0.79	0.87
LogMap [13]	0.99	0.79	0.88	0.95	0.76	0.84
LogMapBio	0.99	0.79	0.88	0.95	0.76	0.84
LogMapKG	0.99	0.79	0.88	0.95	0.76	0.84
LogMapLt	1.00	0.59	0.74	0.8	0.43	0.56
Wiktionary [24]	1.00	0.79	0.88	1.00	0.67	0.8
DESKMatcher [18]	0.0	0.0	0.0	0.76	0.66	0.71
KGbaselineLabel	1.0	0.61	0.76	1.00	0.59	0.74
proposed matcher	**0.98**	**0.92**	**0.95**	0.88	0.73	0.77

Other systems also utilize external background knowledge resources such as ALOD2Vec and Wiktionary. However, our matcher only uses the labels of entities to produce matching class alignments. Moreover, the majority of OAEI

systems incorporate multiple string-based techniques such as n-gram, prefixes and suffixes. For instance, one of the string processes implemented by ATBox matcher aimed at finding ontology specific stopwords, which are words that often appear in a certain KG or ontology. It considers such words as stopwords to be removed prior to applying further string matching techniques. This allows to discover the similarity of ⟨`sidebar_starship, starship`⟩ and ⟨`sidebar_novel, novel`⟩, if `sidebar` was a corpus specific stopword. However, those classes will not be matched by word embedding models or by an edit-distance similarity with a high threshold such as the one we apply, i.e., 0.8.

Another contribution to the different performance of our matcher across the two datasets is the datasets' nature. While the first dataset is constructed from common KGs where classes annotate complementary real-world entities, the OAEI datasets have very different nature. Moreover, they are restricted to a single domain (entertainment) where distinguishing entities of book, music and movie can be a difficult task. This is due to the usage of words in naming such entities, which can be very heterogeneous and inconsistent. To illustrate, fig. 4 shows the difference in the performance of our KG classifier when it is trained to classify NELL instances, as opposed to when it is trained to classify the MemoryAlpha KG instances in the OAEI benchmark dataset. Clearly, it was easier to classify instances from multi-domain large KGs like NELL.

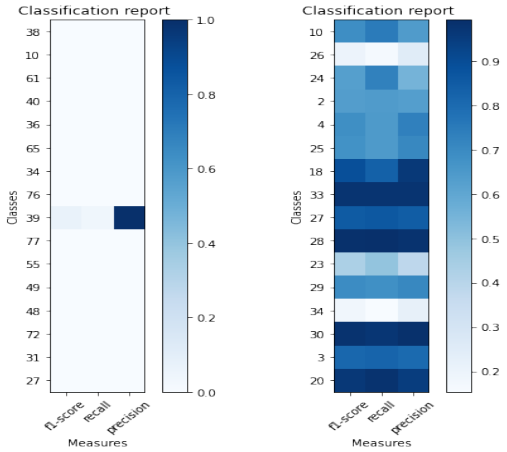

Fig. 4: The instance classification report of a 20 randomly sampled classes from the OAEI KG MemoryALpha in (a) and NELL in (b). Note that y-axis numbers indicate class IDs.

5.1 The impact of components and parameters

In this section, we discuss the impact of several components and parameters of the proposed method. First, in addition to the BERT-based classifier, we also tested our method with another model, based on a simple DNN with 4 density

connected layers. Although both models outperform all OAEI participants on the task of matching common KGs, we use BERT model as it is shown to achieve slightly better F1 (by 0.01). This shows that our approach is generalizable to other machine learning algorithms for the KG classifier. Second, in terms of the threshold **k**, we tested three configurations, 5, 10, and 20. We noticed identical results with k=10 and 20, while slightly lower F1 when k=5, i.e., 0.94 which still significantly outperforms all OAEI participants. Further, setting k to 10 compared to 20 led to a significant reduction in runtime due to more aggressive undersampling (from 55 minutes on the common KGs dataset to 29 minutes). Finally, the exact name filtering had a positive effect on the overall performance of our method, by increasing the F1 by 0.02 on the common KGs dataset.

6 Conclusion and future work

This work reports an ongoing study of utilizing instances to match KGs classes. We proposed a novel domain independent approach for matching classes in large KGs. To the best of our knowledge, our matcher includes the first instance-based matcher for matching KG classes with the ability to handle unbalanced populations. Our findings suggest that a hybrid approach that composes an instance-based matcher can be very effective for matching common KG classes. In future versions of our matcher, we aim to further study the thresholds focusing on the possibility to automate that decision, and to further improve our matcher combination to be able to address different matching tasks. We also aim to extend our matching method to match all KGs entities.

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. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Hruschka, E.R., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Twenty-Fourth AAAI Conference on AI (2010)
3. 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)
4. Fallatah, O., Zhang, Z., Hopfgartner, F.: A gold standard dataset for large knowledge graphs matching. In: OM@ISWC (2020)
5. Faria, D., Pesquita, C., Santos, E., Palmonari, M., Cruz, I.F., Couto, F.M.: The agreementmakerlight ontology matching system. In: OTM Confederated International Conferences. pp. 527–541. Springer (2013)
6. Fernández, A., García, S., Galar, M., Prati, R.C., Krawczyk, B., Herrera, F.: Learning from imbalanced data sets, vol. 11. Springer (2018)
7. 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)

8. Hertling, S., Paulheim, H.: Dbkwik: A consolidated knowledge graph from thousands of wikis. In: 2018 IEEE International Conference on Big Knowledge (ICBK). pp. 17–24 (2018)
9. Hertling, S., Paulheim, H.: Atbox results for oaei 2020. In: CEUR Workshop Proceedings. vol. 2788, pp. 168–175. RWTH (2020)
10. Hertling, S., Paulheim, H.: The knowledge graph track at oaei. In: European Semantic Web Conference. pp. 343–359. Springer (2020)
11. 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)
12. Isaac, A., Van Der Meij, L., Schlobach, S., Wang, S.: An empirical study of instance-based ontology matching. In: The Semantic Web, pp. 253–266. Springer (2007)
13. Jiménez-Ruiz, E.: Logmap family participation in the oaei 2020. In: Proceedings of the 15th International Workshop on Ontology Matching (OM 2020). vol. 2788, pp. 201–203. CEUR-WS (2020)
14. Ksieniewicz, P.: Undersampled majority class ensemble for highly imbalanced binary classification. In: Proceedings of the Second International Workshop on Learning with Imbalanced Domains: Theory and Applications. pp. 82–94 (2018)
15. Liu, A.C.: The effect of oversampling and undersampling on classifying imbalanced text datasets. The University of Texas at Austin p. 67 (2004)
16. Maiya, A.S.: ktrain: A low-code library for augmented machine learning. arXiv preprint arXiv:2004.10703 (2020)
17. 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)
18. Monych, M., Portisch, J., Hladik, M., Paulheim, H.: Deskmatcher. In: CEUR Workshop Proceedings. vol. 2788, pp. 181–186 (2020)
19. Ngo, D., Bellahsene, Z., Todorov, K.: Opening the black box of ontology matching. In: Extended Semantic Web Conference. pp. 16–30. Springer (2013)
20. Nkisi-Orji, I., Wiratunga, N., Massie, S., Hui, K.Y., Heaven, R.: Ontology alignment based on word embedding and random forest classification. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 557–572. Springer (2018)
21. Otero-Cerdeira, L., Rodríguez-Martínez, F.J., Gómez-Rodríguez, A.: Ontology matching: A literature review. Expert Systems with Applications pp. 949–971 (2015)
22. Padurariu, C., Breaban, M.E.: Dealing with data imbalance in text classification. Procedia Computer Science **159**, 736–745 (2019)
23. Portisch, J., Hladik, M., Paulheim, H.: Alod2vec matcher results for oaei 2020. In: CEUR Workshop Proceedings. vol. 2788, pp. 147–153. RWTH (2020)
24. Portisch, J., Paulheim, H.: Wiktionary matcher results for oaei 2020. In: CEUR Workshop Proceedings. vol. 2788, pp. 225–232. RWTH (2020)
25. Schütze, H., Manning, C.D., Raghavan, P.: Introduction to information retrieval, vol. 39. Cambridge University Press Cambridge (2008)
26. Thor, A., Kirsten, T., Rahm, E.: Instance-based matching of hierarchical ontologies. Datenbanksysteme in Business, Technologie and Web (2007)
27. Zhang, Z., Gentile, A.L., Blomqvist, E., Augenstein, I., Ciravegna, F.: An unsupervised data-driven method to discover equivalent relations in large Linked Datasets. Semantic Web pp. 197–223 (2017)