Entity Linking to Knowledge Graphs to Infer Column Types and Properties *

Avijit Thawani, Minda Hu, Erdong Hu, Husain Zafar, Naren Teja Divvala, Amandeep Singh, Ehsan Qasemi, Pedro Szekely, and Jay Pujara

Information Sciences Institute, University of Southern California

Abstract. This paper describes our broad goal of linking tabular data to semantic knowledge graphs, as well as our specific attempts at solving the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching. Our efforts were split into a Candidate Generation and a Candidate Selection phase. The former involves searching for relevant entities in knowledge bases, while the latter involves picking the top candidate using various techniques such as heuristics (the 'TF-IDF' approach) and machine learning (the Neural Network Ranking model). We achieve an F1 score of 0.826 without any training data on the 400000+ cells to be annotated in Round 2 CEA challenge. On CTA and CPA variants, we score 1.099 and 0.790 respectively.

Keywords: Semantic Web · Table Understanding · Knowledge Graphs

1 Introduction

The objective of our work is to investigate approaches to address the three tasks in the ISWC Tabular Data to Knowledge Graph (KG) Matching challenge: **entity linking** (CEA) to map table cells to entities in a knowledge graph, **semantic labeling** (CTA) to map table columns to an ontology class, and **semantic modeling** (CPA) to map column-pairs to an ontology property. While the challenge focuses on DBpedia, we seek general approaches that effective with other knowledge graphs, such as Wikidata.

Figure 1 shows an overview of our approach. For each cell that must be annotated, a **candidate generation** module generates a ranked list of candidate entities in the target KG. In the next step, a **feature generation** module generates a vector of features for each candidate, and in the final step, a **candidate selection** module scores the candidates using the feature vectors. The figure illustrates the most important challenges: inability to generate any candidates (row 4); generated candidates do not include the correct answer (row 5); the

* This material is based upon work supported by United States Air Force under Contract No. FA8650-17-C-7715.

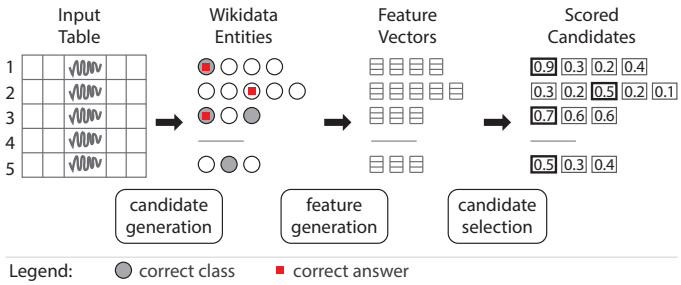

Fig. 1. Overview of our approach

correct answer is not an instance of the semantic label for a column (row 2); multiple candidates have the correct semantic label (row 3); all candidates with the correct semantic label are incorrect (row 5).

Our approaches for CTA and CPA use the components shown in the architecture. For CTA we compute the semantic label using the scored candidates, and for CPA we use the candidates generated in the candidate generation step. In the following sections we describe multiple approaches for each module and report on initial experiments to assess effectiveness.

2 Techniques

2.1 Candidate Generation

We investigated multiple approaches for candidate generation. We use **Wikidata API** to obtain up to 100 candidates for each cell with cell contents and headers. While the results of Wikidata API are of high quality(submission with top candidate of each cell achieves 0.7 F1-score), the maximum achievable recall, which is the ratio of cells having ground truths in their candidates, is still limited. To address this recall limitation of the API, we build a **Wikidata Elasticsearch Index** containing fields for multilingual labels, aliases and descriptions and combine results from multiple queries focusing on different fields. The combined Wikidata candidate generation approach achieves a maximum achievable recall of 85%. In addition, we build a second Elastic Search index using **DBpedia labels**, mapping all DBpedia URIs to their corresponding Wikidata entity identifier (qnode), achieving the maximum achievable recall of 91%. We use normalized reciprocal rank as scores to combine and sort candidates from all queries. Finally, we build a **Name Abbreviations Index** that records abbreviations of person names (initials plus surname) for all instances of Human (Q5) in Wikidata, as this format is common in the round 2 dataset.

2.2 Feature Generation

In this work we investigated a feature engineering approach. *Lexical features* capture the lexical similarity between the contents of a cell and entity labels, and *semantic features* capture semantic coherence among cells in a column.

Lexical features We investigated two lexical features. The first one is the normalized score from all candidate generation modules, which captures a token-based similarity between the tokens in a cell, and the tokens in the labels, aliases and descriptions in the KGs. For performance reasons, we limited the candidate generation to the token-based similarity and designed a second feature to measure at character level. We generate features using several string distance metrics, like inverse Levenshtein distance, word similarity (number of words of the label text which were found in the DBpedia URI), etc.

Semantic features A simple approach to capture semantic coherence is to assume that all entities in a column belong to the same ontology class. The most specific class of all or most cells in a column, as computed in the CTA task is informative, but not specific enough. For example, in Wikidata, all humans are instances of class *Human* (Q5), and the set of athletes is identified using the *occupation* property. We investigated a generalization of this idea to use all properties used to describe entities, in addition to the *instance of* property that identifies the classes of an entity.

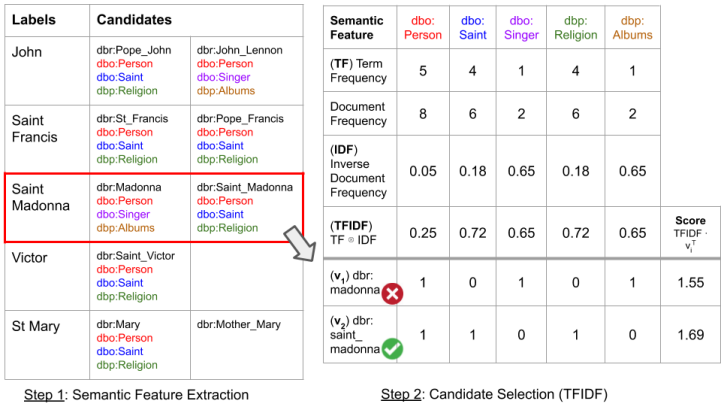

Fig. 2. The TF-IDF approach

For example, consider a column containing names of airports. Candidate entities that are instances of `dbo:Airport` are likely to be correct. The correct set can also be identified by candidates that have values for the property `dbo:runwayDesignation`.

We use candidates for all cells in a column to compile a set of all the classes and properties used to describe them, and for each candidate, we define a uniform-length binary sparse feature vector to record the classes and properties used to describe it. If the features are `dbo:Airport` and `dbo:runwayDesignation`, then the candidate entity `dbpedia:Heathrow` has a feature vector $[1, 1]$. In practice, these vectors may contain thousands of entries.

These features are not equally informative. We seek to maximize *coverage* and *selectivity*. Intuitively, a feature has good coverage if for every cell there is a candidate for which this feature has value 1. In our example, both `Organization`

and `Airport` have good coverage, but `Person` would not as many airports are not named after people. A feature is selective if fewer candidates possess that feature. For our example column, `Airport` is more selective than `Organization` as fewer candidates have a 1 for the `Airport` feature.

We implement this intuition using an adaptation of TF-IDF. In our context, we observe that the first result from candidate generation is more often a correct candidate. Thus, a good measure of 'Term Frequency' of a given semantic feature (eg. `dbo:Person`) is the number of cells for which the first candidate has this feature. In Figure 2, `dbo:Person` occurs in the first candidate of all 5 cells, hence its TF = 5. On the other hand, `dbp:Album` occurs just once in the first candidate (as a feature of `dbr:Madonna`) hence its TF = 1.

Taking forward the analogy, we define the Document Frequency of a semantic feature as the number of total occurrences of the feature (in all candidates). In Figure 2, `dbo:Person` occurs in 8 out of 9 candidates, hence its IDF = $log(9/8) = 0.05$. Once we compute all feature weights accordingly, we can find the 'Score' for each candidate, which is the dot product of the weight vector (TFIDF) and the binary feature vector (v_1 or v_2). As seen in Figure 2, these weights help us select Saint Madonna as the correct entity rather than Madonna, the singer.

2.3 Candidate Selection

We investigated multiple approaches for candidate selection.

Top-1 candidate The simplest approach is to select the top-scoring candidate from the candidate generation module, ignoring all other features. Surprisingly, this approach is a very strong baseline: in round 1 it received an F1 score of 0.809 and a precision of 0.843 and in round 2 it received an F1 score of 0.701 and a precision of 0.768.

Heuristic linear combination We combiner the lexical and semantic features using the following simple formula:

$$S_c = tfidf + 0.5 \times levenshstein\_similarity + 0.5 \times word\_similarity$$

This approach in round 2 received 0.826 F1-score and 0.852 precision. Please note that this submission was the result of concatenating two different submissions, both using the same linear combination as above but differing in that one of them had candidates generated by using a special query in Elasticsearch for handling abbreviations (See Section 2.6 and Table 1 for details).

Neural Network Ranking Model In the heuristic algorithm above, weights among the three features are fixed and not adaptive to the dataset. To solve this problem, we propose a new model that learns weights and relationships from labeled data.

Pairwise Contrastive Loss: Given set $\mathcal{S} = \{(x_{i+}, x_{i-}) \mid x_{i+} \in truth_c,\ and\ x_{i-} \in candidate_c - truth_c, c \in \phi\}$ where (x_{i+}, x_{i-}) is a pair of feature vectors described in *Heuristic linear combination* respectively from ground

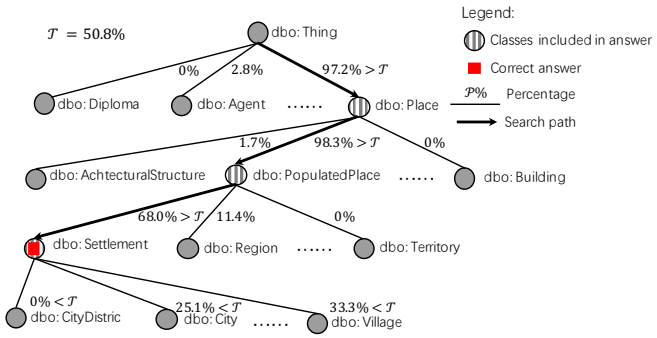

Fig. 3. Demonstration of CTA algorithm

truth $truth_c$ and other wrong answers from $candidate_c$, and c is the cell from training set ϕ. Given a ranking model $\mathcal{F} : x \in \mathbb{R}^{N_f} \rightarrow y \in \mathbb{R}$ where N_f is feature length, the main idea of model training is to enforce one-dimensional model output y_{i+} of ground truth feature vector x_{i+} to be far away from y_{i-} of the wrong candidates x_{i-} and to make y_{i+} greater than y_{i-}. For this goal we propose and optimize a variant of contrastive loss[1] as follows.

$$\mathcal{L}_{contrastive} = \sum_{(x_{i+},x_{i-})\in S} max\{0, m + \mathcal{F}(x_{i-}) - \mathcal{F}(x_{i+})\} + \alpha\|w\|_1$$

where $m > 0$ is margin and $\alpha\|w\|_1$ is L1 normalization.

Model Structure: In this challenge we use a 2-layer neural network activated by ReLU[2]. After training on T2Dv2 dataset[3], scores are obtained from feature of each candidate with ranking model, and candidates with highest score is chosen as answers for each cell. This approach received a high 0.871 F1-score with candidates of 8% cells lacking ground truths in T2Dv2 dataset, and in round 2 it received a lower F1 score 0.808, hinting towards distribution differences between the two datasets.

2.4 CTA

Our approach for the CTA challenge uses the results from the CEA challenge as illustrated in see Fig. 3:

(1) For each column $\mathcal{C}$ our algorithm starts from `dbo:Thing` on level 0 of semantic class tree and calculates the percentages of cells in $\mathcal{C}$ that belong to classes on certain level of the DBpedia class tree. In Fig. 3, on level 2 the percentages of `dbo:Diploma`, `dbo:Agent` and `dbo:Place` are 0%, 2.8% and 97.2%.

(2) The algorithm then picks most common class `dbo:Place` with the highest percentage of 97.2%. Since 97.2% is greater than the threshold percentage $\mathcal{T} =$ 50.8%, `dbo:Place` is regarded as one of classes that $\mathcal{C}$ belongs to. The algorithm records `dbo:Place` in its search path and goes back to step (1), selecting most common classes from the children of `dbo:Place`.

(3) After selecting `dbo:PopulatedPlace` and `dbo:Settlement` respectively on level 2 and 3, there is no class percentage greater than $\mathcal{T}$ on level 4, so the search stops and the algorithm outputs the path below the root: `dbo:Place` → `dbo:PopulatedPlace` → `dbo:Settlement`.

2.5 CPA

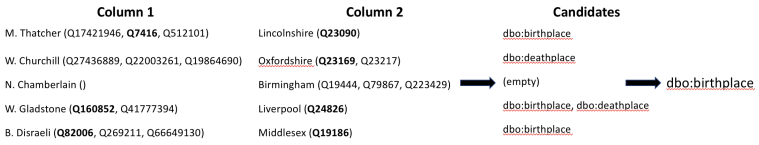

Fig. 4. Example of our CPA system on a single table

In the CPA task, the goal is to find the DBPedia ontology property that best matches the relation between a primary column and other secondary columns. As input, we have the ranked candidates from CEA and sample size N, which is the top N of the candidates we want to consider, to limit the size of our query search space. For each row, we query DBPedia for the properties between all pairs of candidates in the primary and secondary columns. For example in Fig. 4, the candidates for "M. Thatcher" are paired with the one candidate for "Lincolnshire" and we discover `dbo:birthplace` among these properties. We output the most frequent property among all the rows, in the above case `dbo:birthplace`. For secondary columns consisting of literals, we transform the value to address potential differences between the cell value and the values in the KG. Currently we address cases of string, date, and numerical literals.

2.6 Adaptations

In our work, our primary goal was to develop effective algorithms for DBpedia and Wikidata as target KGs. To work with the challenge files we cleaned the input files to address issues such as files with and without headers, empty lines before the first row and a variety of character encoding problems.

Our primary adaptation to the challenge are string similarity metrics to measure distance between cell values and DBpedia URIs. To handle abbreviations we utilized an abbreviated Elasticsearch index to get the candidates for labels that match the regex `^.\.\s.+`. In the future we plan to use metrics that compare the cell values to the labels of candidates.

3 Results

CEA In the upper section of Table 1 i.e. (1) Ablations with Abbreviations, we show the gains achieved using a specialized candidate generation module (Refer to Section 2.6) to handle abbreviations. In the lower section i.e. (2) Experiments on number of rows, we report results by submitting predicted cell annotations

Table 1.

Ablations with/without Abbreviations						
Method	**Candidates**	**Cells**	**# Predictions**	**Precision**	**Recall**	**F1 score**
Heuristic	API, ES	All	419,921	0.766	0.694	0.728
	API, ES	abbreviations only	63,207	*0.201*	0.027	0.048
	API, ES	All except abbreviations	356,714	**0.866**	0.666	0.753
	ES-abbr	abbreviations only	80,451	**0.788**	0.137	0.233
	API, ES, ES-abbr	All	437,165	**0.852**	0.802	**0.826**
Experiments on # Rows						
Method	**Candidates**	**Files**	**# Predictions**	**Precision**	**Recall**	**F1 score**
Heuristic	API, ES, ES-abbr	# Rows > 100	235,914	0.877	0.446	0.591
		100 > # Rows > 10	171,378	0.847	0.313	0.457
		10 > # Rows	29,873	0.675	0.044	0.082

for subsets of tables (a small part of our best achieved results) with restrictions on the number of rows, as described. Our heuristic loses precision when run on smaller tables (less than 10 cells/rows in each column) as compared to larger tables (more than 100 cells/rows). Besides, even for very large tables, our heuristic seems to be only at a precision of 0.877.

CTA In CTA challenge we first did a grid search by submitting results with different $\mathcal{T}$ to get the best performing one. According to the results, CTA algorithm performs best when $\mathcal{T}$ is set to 0.508. However, in Experiment 1 the algorithm discards 1389 columns mistakenly annotated as `dbo:Thing` or `dbo:Agent`. In Experiment 2 we tune another $\mathcal{T}'$ for these columns, increasing the score by 0.008. The result is shown in Table 3.

We studied the sensitivity of our algorithm to the number of cells in a column. We partitioned the collection of columns into multiple datasets and computed the improvement on round2 primary scores between adjacent bands as shown in Table 2. The results show a significant performance increase with tables containing more than 10 rows and better performance with a larger number of cells.

Table 2. CTA performance growth rate on different ranges

Ranges	Columns	Growth rate
(0,2]	0	NA
(2,3]	322	NA
(3,4]	520	3.2%
(4,5]	447	11.6%
(6,10]	2582	5.7%
(10,20]	3394	43.5%
(20,50]	2540	20.6%
(51,100]	1340	13.7%
(101,+∞)	1722	8.4%

Table 3. CTA results finetuning $\mathcal{T}$, $\mathcal{T}'$

Experiment 1			Experiment 2		
$\mathcal{T}$	Primary	Secondary	$\mathcal{T}'$	Primary	Secondary
0.30	1.033	0.258	0.30	-0.031	0.145
0.35	1.057	0.26	0.35	-0.027	0.151
0.40	1.059	0.26	0.40	-0.025	0.152
0.45	1.063	0.259	0.45	-0.023	0.154
0.50	1.059	0.258	0.50	-0.022	0.154
0.508	**1.091**	**0.261**	0.508	0.007	0.242
0.55	1.083	0.259	0.55	**0.008**	**0.243**
0.60	1.063	0.255	0.60	0.007	0.243
0.65	1.044	0.252	0.65	**0.008**	**0.25**
0.70	1.020	0.249	0.70	0.007	0.249

CPA In CPA, our initial experiment directly used the CEA candidates with sample size $N = 10$. The initial experiment performed poorly on primary columns consisting of human names which could not be directly found in our candidate generation, so we augment the candidate list with a human names index

in the second experiment. We found that certain properties such as `dbo:area` had class-specific synonyms such as `dbo:PopulatedArea/area` so in our third experiment we mapped these synonymous properties to their generic analog. These results are shown in Table 4.

Table 4. CPA result

F1	Precision	Experiment Details
0.399	0.679	Basic CPA system without names index
0.786	0.788	CPA with names index
0.79	**0.792**	CPA with names index and synonymous property filtering

4 Conclusion

We proposed an architecture with three modules, *candidate generation*, *feature generation* and *candidate selection* and implemented KG-agnostic approaches for each module, as we are interested in using Wikidata as a target KG. We developed an interesting feature generation module that combines classes and properties in a uniform framework to characterize the semantic coherence of the values in a column, and explored both heuristic and machine learning approaches for candidate selection.

Our round 2 results are limited by insufficient recall in our candidate generation modules, so in round 3 we plan to focus on candidate generation. In round 2 our best results were obtained using a heuristic candidate selection method, so in round 3 we plan to continue our work on the machine learning approaches. We also plan to study sensitivity of our approaches to characteristics of the datasets, including the number of cells in a column, distribution of length of cell values, and effectiveness for different data types (people, organizations, places, etc.) We propose the following modifications to the challenge:

1. Annotation of all cells in a column, with a special *no annotation* marker to identify cells not present in the target KG.
2. Submission of a single class for CTA, changing the scoring function to give partial credit for super-classes or over-specific classes (e.g., $1/2^n$ where n is the number of super-class or sub-class links to the correct answer).
3. Release of a subset of the ground truth for algorithm development.

References

1. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 2, pp. 1735–1742. IEEE (2006)
2. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on machine learning (ICML-10). pp. 807–814 (2010)
3. Ritze, D., Lehmberg, O., Bizer, C.: T2dv2 gold standard for matching web tables to dbpedia. `http://webdatacommons.org/webtables/goldstandardV2.html` (2015), accessed 28-September-2019