AMALGAM: making tabular dataset explicit with knowledge graph

Rabia Azzi[0000−0002−8777−1566] and Gayo Diallo[0000−0002−9799−9484]

BPH Center - INSERM U1219, Team ERIAS, Univ. Bordeaux,
F-33000, Bordeaux, France
first.last@u-bordeaux.fr

Abstract. In this paper we present AMALGAM, a matching approach to annotate tabular dataset with the use of a knowledge graph, developed in the context of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab 2020). The ultimate goal is to provide fast and efficient approach to annotate tabular data with entities from a background knowledge. The approach combines lookup and filtering services combined with text pre-processing techniques. Experiments conducted in the context of SemTab 2020 with both Column Type Annotation and Cell Type Annotation tasks showed promising results.

Keywords: Tabular Data · Knowledge Graph · AMALGAM · Entity Linking · FAIR principles.

1 Introduction

Making web data complying with the FAIR principles [1] has become a necessity in order to facilitate their discovery and reuse. This could be achieved by annotating these data by entities coming from ontologies and structured vocabularies, semantic repositories [2] and Knowledge Graphs (KG). Tabular Data to KG [3] Matching challenge (SemTab 2020[1]) aims at benchmarking systems which deals with the task of annotating tabular data with entities from a KG, referred as table annotation [10]. Over the last years, a set of systems for matching web tables to knowledge bases have been developed [4, 5].

In order to perform a more fine-grained analysis of the SemTab challenge tasks, we categorize them as two main tasks e.g., structure, and semantic annotation. Structure annotation, deals with various tasks including data type prediction and table header annotation [6]. Semantic annotation involves matching table elements into KG, e.g., columns to class and cells to entities [7, 8].

Three tasks that are organised into several evaluation rounds are defined in SemTab 2020: (i) assigning a semantic type (e.g., a KG class) to a column (CTA); (ii) matching a cell to a KG entity (CEA); (iii) assigning a KG property to the

[1] http://www.cs.ox.ac.uk/isg/challenges/sem-tab/2020/index.html

relationship between two columns (CPA). The most popular approaches to deal with these three tasks are a supervised learning setting, where entities candidate are selected by a classification model [9]. However, for the real-time application, obtaining the result as fast as possible is a requirement. As a basis for the current work, the ultimate goal is to provide a fast and efficient approach for a tabular dataset to KG matching task. We have designed and implemented AMALGAM, our proposed approach to realizing CTA and CEA tasks.

2 The AMALGAM approach

To address the above mentioned tasks of the SemTab challenge, AMALGAM is designed according to the workflow depicted in Fig. 1. There are three major phases which consist in, respectively, Pre-Processing, Annotation context and Tabular data to KG matching. The first two steps of the Workflow are identical for the two tasks.

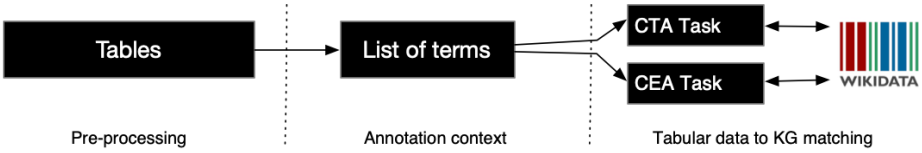

Fig. 1. Workflow of AMALGAM approach. Pre-Processing, which aims to prepare the data inside the table. Annotation context, which aims to create a list of terms that describe the same context. Tabular data to KG matching: (1) Assigning a semantic type to a column (CTA), which aims to identify the attribute label of column; (2) Matching a cell to a KG entity (CEA), which deals with mappings between terms and entities in a KG.

To describe each phase of the AMALGAM approach we consider Table 1, which lists Alberta region towns with additional information, country, elevation above sea level, etc.

Table 1. Table with a list of the Alberta region towns extracted from Round 1 of SemTab'2020.

col0	col1	col2	col3	col4	col5
Grande Prairie	city in Alberta	Canada	Sexsmith	650	Alberta
Sundre	town in Alberta	Canada	Mountain View County	1093	Alberta
Peace River	town in Alberta	Canada	Northern Sunrise County	330	Alberta
Vegreville	town in Alberta	Canada	Mundare	635	Alberta
Sexsmith	town in Alberta	Canada	Hythe, Alberta	724	Alberta

Tables Pre-Processing. As the Table 1 shows, the content of each table can have different types (string, date, float, etc.). The aim of this pre-processing

step is to ensure that the process of loading each table happens without any error. For example, a problem of textual encoding where some characters are loaded as noisy sequences, text field with an unescaped delimiter, will cause the record being processed to have an extra column, etc. Loading incorrect encoding might strongly affect the lookup performance. Therefore, we used the Pandas[2] library to fix all noisy textual data in the tables.

Annotation context. We consider a table as a two-dimensional tabular structure (see Fig. 2) that is composed of an ordered set of x rows and y columns. Each intersection between a row and a column determines a cell c_{ij} with the value v_{ij} where $1 \leq i \leq x, 1 \leq j \leq y$. To identify the attribute label of column also called header detection (`CTA` task), the approach consists in annotating all the items of the column using entity linking. Then, the attribute label row detection is estimated using the random entity linking. In this use case, the annotation context is represented by the list of items in the column. For example, the context of the first row in the Fig. 2 is: [*Grande Prairie, Sundre, Peace River, Vegreville, Sexsmith*]. According to the same logic, we consider that all cells in the same row describe the same context. More precisely, the first cell of the row describes the entity and the following cells the associated properties. For example, the context of the first row in the Fig. 2 is: [*Grande Prairie, city in Alberta, Canada, Sexsmith, 650, Alberta*].

Assigning a semantic type to a column (`CTA`). The `CTA` task can be performed by exploiting the process described in Fig. 2. The Wikidata API allows to look up a Wikidata item[3] according to the title of its corresponding page on a given Wikipedia page, or other Wikimedia family site. In our case, the main information needed from the entity is a list of the *instances of (P31)*, *subclass of (P279)* and *part of (P361)* statements. To do so, a parser is developed to retrieve this information from the Wikidata built request. For example, "Grande Prairie" provides the following results: [list of towns in Alberta:Q15219391, village in Alberta:Q6644696, city in Alberta:Q55440238]. To achieve this, our methodology combines *wbsearchentities* and *parse* actions provided by the API. It could be observed that in this task, there were many items that have not been annotated. This is because tables contain incorrectly spelled terms. Therefore, before implementing the other tasks, a spell check component is required.

As per the literature [11], spell-checker is a crucial language tool of natural language processing (`NLP`) which is used in applications like information extraction, proofreading, information retrieval, social media and search engines. In our case, we compared several approaches and libraries: Textblob[4], Spark NLP[5],

[2] https://pandas.pydata.org/
[3] https://www.wikidata.org/w/api.php
[4] https://textblob.readthedocs.io/en/dev/
[5] https://nlp.johnsnowlabs.com/

Gurunudi[6], Wikipedia api[7], Pyspellchecker[8], Serpapi[9]. A comparison of these approaches could be found in Table 2.

Table 2. Comparative of approaches and libraries related to spell-checking.

Name	**Category**	**Strengths/Limitations**
Textblob	NLP	Spelling correction, Easy-to-use
Spark NLP	NLP	Pre-trained models, Text Analysis, Multilanguage
Gurunudi	NLP	Pre-trained models, Text Analysis, Easy-to-use, Multilanguage
Wikipedia api	Search engines	Search with suggestion, Easy-to-use, Unlimited Access
Pyspellchecker	Spell checking	Simple spell checking algorithm, No pre-trained models, Easy-to-use
Serpapi	Search engines	Limited Access for free

Our choice is oriented towards Gurunudi and the Wikidata API with a post-processing step consisting in validating the output using fuzzywuzzy[10] to keep only the results whose ratio is greater than the threshold of 90%. For example, let's take the expression *"St Peter's Seminarz"* after using the Wikidata API we get *"St Peter's seminary"* and the ratio of fuzzy string matching is 95%.

We are now able to perform the `CTA` task. In the trivial case, the result of an item lookup is equal a single record. The best matching entity is chosen as a result. In the other cases, where the result is more than one, no annotation is produced for the `CTA` task. Finally, if there is no result after the lookup, another one is performed using the output of the spell check produced by the item. At the end of these lookups, the matched couple results are then stored in a nested dictionary [item:claims]. The most relevant candidate, counting the number of occurrences, is selected.

Matching a cell to a KG entity (`CEA`). The `CEA` task can be performed by exploiting the process described in Fig. 3. Our approach reuse the process of the `CTA` task and made necessary adaptations. The first step is to get all the statements for the first item of the list context. The process is the same as `CTA`, the only difference is where the result provides more than one record. In this case, we create nested dictionary with all candidates. Then, to disambiguate the candidates entities, we use the concept of the column generated with the `CTA` task. Next, a lookup is performed by using the other items of the list context in the claims of the first item. If the item is found, it is selected as the target entity; if not, the lookup is performed with the item using the Wikidata API (if the result is empty, no annotation is produced).

[6] https://github.com/guruyuga/gurunudi
[7] https://wikipedia.readthedocs.io/en/latest/code.html
[8] https://github.com/barrust/pyspellchecker
[9] https://serpapi.com/spell-check
[10] https://github.com/seatgeek/fuzzywuzzy

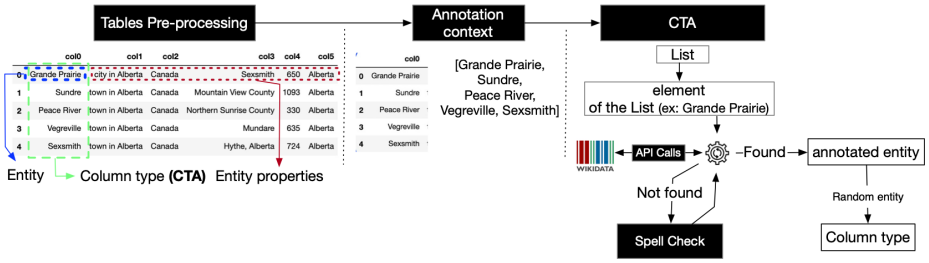

Fig. 2. Assigning a semantic type to a column (`CTA`).

With this process, it is possible to reduce errors associated with the lookup. Let's take the value "650" in row 0 of the table Fig. 3 for instance. If we lookup directly in Wikidata, we can get many results. However, if we check first in the statements of the first item of the list, "Grande Prairie", it is more likely to successfully identify the item.

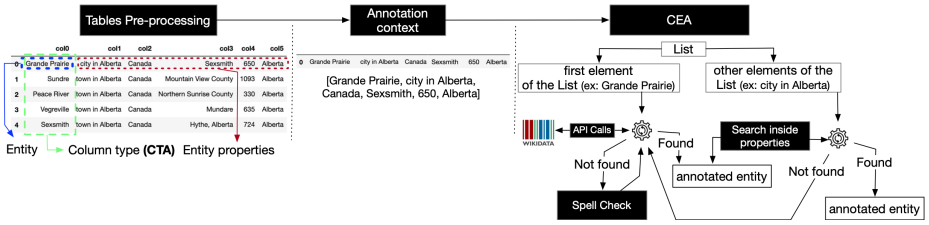

Fig. 3. Matching a cell to a KG entity (`CEA`).

3 Results

This section reports the overall results (The Primary Score) of `AMALGAM` for the two matching tasks in the four rounds of SemTab 2020 [10]. Overall, these results show that `AMALGAM` achieves promising performances for `CTA` and `CEA`.

Table 3. Results of Round 1.

TASK	F1 Score	Precision
CTA	0.724	0.727
CEA	0.913	0.914

Table 4. Results of Round 2.

TASK	F1 Score	Precision
CTA	0.926	0.928
CEA	0.921	0.927

The primary results (F1 score and Precision) of `AMALGAM` for `CTA` and `CEA` in the four rounds are presented in Tables 3, 4, 5 and 6. In the first round,

Table 5. Results of Round 3.

TASK	F1 Score	Precision
CTA	0.869	0.873
CEA	0.877	0.892

Table 6. Results of Round 4.

TASK	F1 Score	Precision
CTA	0.858	0.861
CEA	0.892	0.914

`AMALGAM` achieved the results in Table 3. It could be observed that `AMALGAM` handles properly the two tasks, in particular in the `CEA` task. Regarding the `CTA` task, these results can be explained according to a new revision created in the item revision history and there are probably spelling errors in the contents of the tables. For example, "rural district of Lower Saxony" after 16 April 2020 became "district of Lower Saxony". A possible solution is to retrieve the history of revisions (parsing Wikidata data history dumps) to use them. This could be a perspective to this work.

In Round 2 we particularly focused on the spell check of items to improve the results of the `CEA` and `CTA` tasks. Clearly, our choice tends to use two API services, Wikipedia and Gurunudi, for spelling correction. It could be observed that the achieved results are better than the previous round both in terms of precision and F1-Score. However, these results may be improved too. From previous rounds, we noted that one single term is ambiguous as it refers to more than one entity. In Wikipedia, there is only one article for each concept. However, there can be many equivalent titles for a concept due to the existence of synonyms, etc. For example, the term "Paris" may refer to many concepts such as "the capital and largest city of France", "son of Priam, king of Troy", "county seat of Lamar County, Texas, United States", etc. For the next rounds, disambiguation item is required.

In Round 3 and 4, to overcome the issue related to disambiguation, we have updated our approach by integrating the concept of the column obtained in `CTA` in the linking phase. We showed that the two tasks can be performed relatively successfully with `AMALGAM`, achieving higher than 0.86 in precision and recall values. However, the automatic disambiguation of items proved to be a more challenging task.

A second evaluation is performed with the Tough Tables (2T) dataset which is designed to evaluate table annotation approaches in solving the `CEA` and `CTA` tasks [12]. The results of this second evaluation are given in Table 7. It shows that compared to the four rounds of the traditional challenge, the performance of AMALGAM declined for F1 Score respectively to 0.32 and 0.60 for `CEA` and `CTA` tasks. The same behaviour is observed for all the other participating systems to the SemTab 2020 challenge. That suggests a more challenging issue to annotate 2T[11].

[11] https://www.cs.ox.ac.uk/isg/challenges/sem-tab/2020/results.html

Table 7. 2T Results.

TASK	F1 Score	Precision
CTA	0.606	0.608
CEA	0.323	0.553

4 Conclusion and Future Works

In this paper, we presented `AMALGAM`, a matching approach to making tabular dataset explicit with entities annotation from a knowledge graph model (in the context of the SemTab 2020 Challenge.

Its advantage is that it allows to perform `CTA` and `CEA` tasks in a timely manner. This may be accomplished through the combination of a lookup services and a spell check techniques. It is the first participation of the `AMALGAM` system which is still in the early stages, so there are still rooms from some improvements. However, the experimental results show that our approach achieved promising results.

Our findings during this first participation suggest that the matching process is very sensitive to errors in spelling. Thus, as of future work, an improved spell checking techniques will be investigated. To process such errors the contextual based spell-checkers are needed. Often the string is very close in spelling, but context could help reveal which word makes the most sense. Further more, we will improve the approach by finding a trade-off between effectiveness and efficiency.

References

1. Wilkinson, M., Dumontier, M., Aalbersberg, I. et al. The FAIR Guiding Principles for scientific data management and stewardship. Sci Data 3, 160018 (2016). https://doi.org/10.1038/sdata.2016.18
2. Diallo, G.: Efficient Building of Local Repository of Distributed Ontologies. Proceedings of the 7th International Conference on Signal Image Technology Internet-Based Systems (SITIS'2011), 2011. pages 159-166. doi:10.1109/SITIS.2011.45
3. Ji, S., Pan, S., Cambria, E., Marttinen, P., Yu, P.S.: A survey on knowledge graphs:Representation, acquisition and applications. CoRRabs/2002.00388(2020)
4. Subramanian, A., Srinivasa, S.: Semantic Interpretation and Integration of Open Data Tables. In: Geospatial Infrastructure, Applications and Technologies: India Case Studies, pp. 217–233. Springer Singapore (2018). https://doi.org/10.1007/978-981-13-2330-0_17
5. Taheriyan, M., Knoblock, C.-A., Szekely, P., Ambite, J.-L.: Learning the semantics of structured data sources. Web Semantics: Science, Services and Agents on the World Wide Web **37**(38), 152–169 (2016)
6. Zhang, L., Wang, T., Liu, Y., Duan, Q.: A semi-structured information semantic annotation method for Web pages. Neural Computing and Applications **32**(11), 6491–6501 (2019)
7. Efthymiou, V., Hassanzadeh, O., Rodriguez-Muro, M., Christophides, V.: Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings. In: Lecture Notes in Computer Science, pp. 260–277. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-68288-4_16

8. Eslahi, Y., Bhardwaj, A., Rosso, P., Stockinger, K., Cudre-Mauroux, P.: Annotating Web Tables through Knowledge Bases: A Context-Based Approach. In: 2020 7th Swiss Conference on Data Science (SDS), pp. 29–34. IEEE (2020). https://doi.org/10.1109/sds49233.2020.00013
9. Hassanzadeh, O., Efthymiou, V., Chen, C., Jimenez-Ruiz, E., Srinivas, K.: SemTab2019: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching - 2019 Data Sets (Version 2019) [Data set]. Zenodo. https://doi.org/10.5281/zenodo.3518539
10. Hassanzadeh, O., Efthymiou, V., Chen, C., Jimenez-Ruiz, E., Srinivas, K.: SemTab2020: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching - 2020 Data Sets, October 2020.
11. Shashank, S., Shailendra, S.: Systematic review of spell-checkers for highly inflectional languages. Artificial Intelligence Review **53**(6), 4051–4092 (2019)
12. Cutrona, V., Bianchi, F., Jiménez-Ruiz, E., Palmonari, M. Tough Tables: Carefully Evaluating Entity Linking for Tabular Data. Zenodo, (2020). https://doi.org/10.5281/ZENODO.4246370