ISWC Challenge: Transforming Tabular Data into Semantic Knowledge

Gilles Vandewiele, Bram Steenwinckel, Filip De Turck, and Femke Ongenae

IDLab, Ghent University – imec, Technologiepark-Zwijnaarde 126, Ghent, Belgium
`{firstname}.{lastname}@ugent.be`

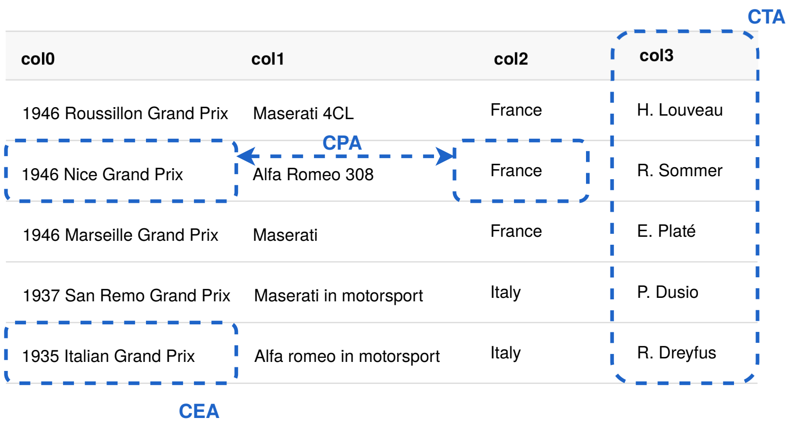
Abstract. A large portion of structured data does not yet reap the benefits of the Semantic Web, or Web 2.0, as it is not semantically annotated. The “Tabular Data to Knowledge Graph Matching” is a competition, consisting of three different subchallenges, in which the participants create systems to automatically annotate tabular data, in the form of CSV files. This paper presents a description of the system submitted by the team IDLab.

Keywords: Tabular Data · Semantic Annotation · Entity Recognition · Type Recognition · Property Recognition.

1 Introduction & Challenge Description

A large portion of both existing and newly produced structured data is not semantically annotated. While solutions exists that allow to easily semantically enrich raw structured data, these have to be tailored specifically to the format of the data [4]. Alternatively, systems can be developed that automatically annotate raw data, albeit less accurately [6,3,5,8,2,7]

The “Tabular Data to Knowledge Graph Matching” is a competition hosted at the International Semantic Web Conference (ISWC), in 2019. Given a Comma-Separated Values (CSV) file, three different challenges had to be tackled: (i) a type from the DBPedia ontology [1] had to be assigned to the columns (Column-Type Annotation (CTA)), (ii) a DBPedia entity had to be assigned to the different cells (Cell-Entity Annotation (CEA)), and (iii) relations between different column had to be inferred, when possible (Columns-Property Annotation (CPA)). These challenges are illustrated in Figure 1. The competition consisted of three different rounds. In each round, a collection of CSV files were provided which had to be annotated. Moreover, the cells, columns and column pairs that had to be annotated were provided as well. This avoids the need to create a pre-processing step that identifies elements in the CSV for which it is possible to create an annotation (e.g. removing cells with numbers). None of the files contained a corresponding ground truth of annotations, allowing only for unsupervised learning approaches to be applied.



col0	col1	col2	col3
1946 Roussillon Grand Prix	Maserati 4CL	France	H. Louveau
1946 Nice Grand Prix	Alfa Romeo 308	France	R. Sommer
1946 Marseille Grand Prix	Maserati	France	E. Platé
1937 San Remo Grand Prix	Maserati in motorsport	Italy	P. Dusio
1935 Italian Grand Prix	Alfa romeo in motorsport	Italy	R. Dreyfus

Fig. 1. The tree different subchallenges of the competition.

2 System Description

Our system consists of three phases. First, crude annotations are made on cell level by generating multiple candidates and disambiguating these using string similarity on the cell value and the `rdf:label` of each candidate. Afterwards, column types and properties between columns are inferred using these annotations. Finally, the inferred column types and properties are used to create more accurate cell annotations.

2.1 Initial Cell Entity Annotation (CEA I)

The pipeline used to annotate a single cells is depicted in the right upper corner of Figure 3. For each of the cells, we first clean them by retaining only the part that comes before a '(' or '[' and by removing all '%', '"', and '\' characters (`clean_cell`). Then, we check whether `http://dbpedia.org/resource/<X>` exists where `<X>` is simply the cleaned cell value with spaces replaced by underscores (`try_url`). Parallel with this, the cell value is provided to the DBPedia lookup API to generate more candidates (`DBPedia lookup`). As no column type annotations are available yet at this time, we simply do not provide this. If both the `DBPedia lookup` and the `try_url` step did not result in any candidates, the DBPedia Spotlight API is applied a final time. In the end, a large pool of possible candidates remain. On this pool, we apply disambiguation by selecting the candidate of which its `rdf:label` has the lowest Levenshtein distance to the actual cell value.

2.2 Column Type Annotation (CTA)

After annotating the different cells, we can query the different types for each of these annotations in the same column and count these. Based on these counts, the goal is to now find the most specific class that matches the entities in the cell. It is important to note that the entities on DBPedia are not guaranteed to be annotated correctly (e.g. Barack Obama, and not Donald Trump, is still the president of the U.S.A according to DBPedia at the time of writing in August 2019[1]) or completely (e.g. Shaquille O'Neal his musical endeavors, such as being a rapper and DJ, are not completely in there[2]). This makes the inference step far from trivial. Simply taking the type with the highest count, where ties are broken by the type that has the highest depth in the hierarchy, would mostly result in a very generic annotation such as `Thing`.

Since the classes of the DBPedia ontology form a hierarchy, they can be represented in a tree. We can now traverse this tree, and apply majority voting (i.e. taking the child with the highest count) on each level of this tree. We continue recursively until the entropy of the two highest counts of its children, is lower than a specified threshold. The entropy is thus defined as:

$$H(t) = H(\textsc{sort}(\{\textsc{count}(c) \mid c \in \textsc{children}(t)\})[-2:]) \tag{1}$$

With t the type of which we want to calculate its entropy, $H(.)$ the Shannon entropy, $\textsc{count}(.)$ a function to get the number of remaining cell annotations of a certain type, $\textsc{sort}(.)$ a function to sort a collection in a ascending fashion, and $\textsc{children}(.)$ a function to obtain the subclasses of a type in the DBPedia ontology. The reason for only looking at the two highest counts of its children is that else it can be susceptible to outliers (i.e. types with very low counts), which can cause the entropy to decrease quickly. This approach is exemplified in Figure 2.

As the maximum score that could be achieved per target column was not bounded by one, we boosted our score by adding all parents in the hierarchy to each column annotation, excluding `Thing` and `Agent`. Moreover, classes that were equivalent according to the DBPedia ontology (e.g. `Location` and `Place`) were added to the collection of annotations per target as well. This resulted in a significant boost in the primary score, at a cost of a reduced secondary score. Moreover, many CSV files shared the same header (first line). For each CSV file, we looked at the column annotations made in CSV files with the same header. In case of conflict, we assigned the column type that occurred most.

2.3 Column Property Annotation (CPA)

The initial cell annotations are used to annotate the properties or relations between pairs of columns. To do this, we iterate over cell pairs from two target columns between which we want to infer the relation. Then, for each of these cell pairs (s, o), we query for all predicates p from the DBPedia ontology that exist between these two entities: $\{p \mid (s, p, o) \in DBPedia\}$. Finally, the predicate that can be found the most between the cell pairs is chosen.

[1] `http://dbpedia.org/page/President_of_the_United_States`
[2] `http://dbpedia.org/page/Shaquille_O'Neal`

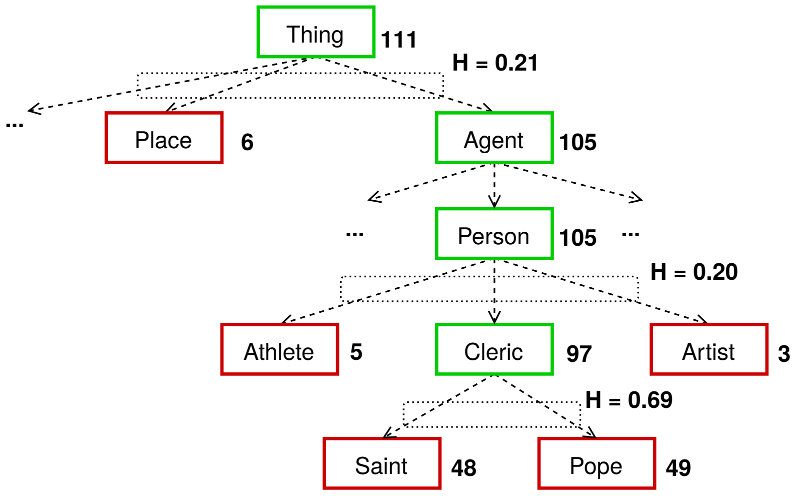


Fig. 2. An example of the column inference heuristic. Majority voting is applied three times until it reaches Cleric. There, the normalized entropy of the two highest counts of its children is too high to continue.

2.4 Final Cell Entity Annotation (CEA II)

After obtaining column type and property annotations, we can apply our CEA pipeline another time to create more accurate annotations. The column type annotation is passed along to the DBPedia lookup. When no candidates are found with that specific column type, the API is called with its parent. The property annotations are used as disambiguation, whenever possible. If a relation is present within the CSV file, we generate candidates for both the head and tail of this relation. Afterwards, we try to find pairs of entities for which the annotated property can be found between them. In case multiple pairs can be formed, disambiguation is performed using Levenshtein distance, as done during the initial cell annotation. The entire pipeline is shown in Figure 3.

3 Competition results

At the time of writing, the first two out of three rounds have been finished. In total, four different metrics were used to evaluate the system. On the one hand, we had the F_1-score, which is a harmonic mean of the precision and recall of our system:

$$\begin{aligned} \text{precision} &= \frac{\#\text{correct annotations}}{\#\text{annotations made}} \\ \text{recall} &= \frac{\#\text{correct annotations}}{|\text{target cells}|} \\ F_1 &= \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \end{aligned} \tag{2}$$

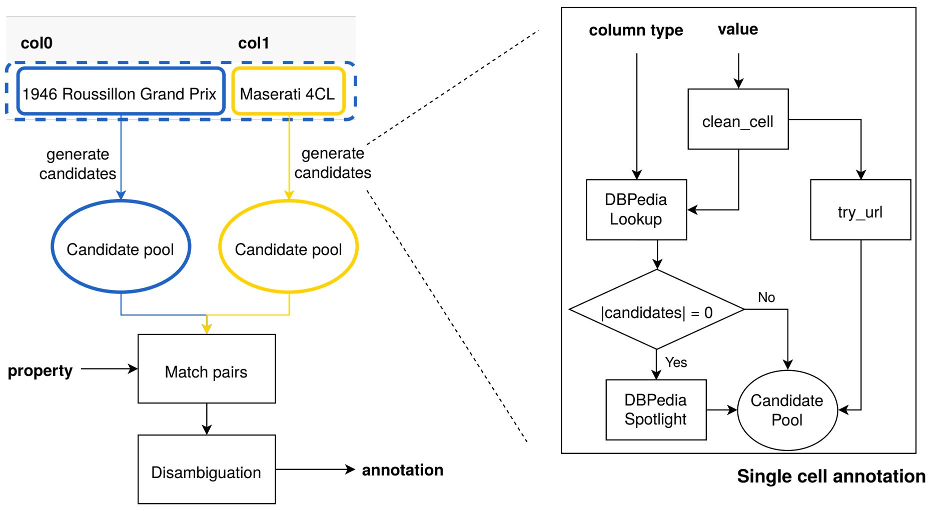


Fig. 3. The pipeline to annotate cells. When no column type is available, it is not provided to the DBPedia Lookup. The pair matching step is also skipped when no property is available.

During the second round, a new metric was introduced for the CTA challenge. An annotation was regarded as *perfect* if it was the most specific class to which all the cells in the column complied to. An annotation was *okay* if it was an ancestor of the actual *perfect* annotations in the DBPedia hierarchy of classes. Based on these two concepts, the primary average hierarchical (AH) and secondary average perfect (AP) score were defined:

$$\begin{aligned} AH &= \frac{\#\text{perfect} + 0.5 * \#\text{okay} - \#\text{wrong}}{|\text{target cells}|} \\ AP &= \frac{\#\text{perfect}}{\#\text{annotations}} \end{aligned} \tag{3}$$

The results of our approaches in round one and round two are summarized in Table 1 and 2 respectively. It should be noted we mostly focused on the CTA challenge during the first round. No submissions were made for the CPA challenge, and we did not annotate all cells for the CEA challenge. Our leaderboard position is rather low in comparison to our ranking during the second round, due to many near-perfect solutions (F_1 close to 1) populating the leaderboard.

4 Conclusion & Future Work

In this paper, a system to convert a CSV file of raw, structured data to semantic knowledge. The system consists of three phases: (i) an initial annotation phase

Challenge	**Primary score**	**Secondary score**	**Leaderboard position**
CEA	$F_1 = 0.448$	$prec = 0.627$	13 / 14
CTA	$F_1 = 0.833$	$prec = 0.833$	6 / 15
CPA	/	/	/

Table 1. The results obtained by our system in the three challenges of round 1.

Challenge	**Primary score**	**Secondary score**	**Leaderboard position**
CEA	$F_1 = 0.883$	$prec = 0.893$	2 / 13
CTA	$AH = 1.376$	$AP = 0.257$	2 / 13
CPA	$F_1 = 0.877$	$prec = 0.926$	2 / 10

Table 2. The results obtained by our system in the three challenges of round 2.

for all target cells, (ii) inferring the types of columns and properties between columns based on these initial cell annotations, and (iii) a third pass over the target cells that utilises the newly inferred column types and properties to create more accurate annotations. The proposed system is rather straight-forward, while already achieving promising results. As future work, we would like to create a system that jointly tackles all three challenges. All possible candidate annotations would have to be stored, and a candidate for each of the three challenges is then chosen that maximizes the likelihood of the data complying to these annotations. Moreover, machine learning systems could be interesting as well, but would require a ground truth.

References

1. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.
2. Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. Tabel: entity linking in web tables. In *International Semantic Web Conference*, pages 425–441. Springer, 2015.
3. Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. Colnet: Embedding the semantics of web tables for column type prediction. AAAI, 2019.
4. Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. Rml: A generic language for integrated rdf mappings of heterogeneous data. *Ldow*, 1184, 2014.
5. Vasilis Efthymiou, Oktie Hassanzadeh, Mariano Rodriguez-Muro, and Vassilis Christophides. Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In *International Semantic Web Conference*, pages 260–277. Springer, 2017.
6. Dominique Ritze, Oliver Lehmberg, and Christian Bizer. Matching html tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, page 10. ACM, 2015.
7. Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8(6):921–957, 2017.

8. Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. Towards disambiguating web tables. In *International Semantic Web Conference (Posters & Demos)*, pages 205–208, 2013.