

CSV2KG: Transforming Tabular Data into Semantic Knowledge

Bram Steenwinckel*, Gilles Vandewiele*, Filip De Turck, and Femke Ongenaë

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. In this paper, we propose a system to generate semantic knowledge, available on DBpedia, from common CSV files. The “Tabular Data to Knowledge Graph Matching” competition, consisting of three different subchallenges, was used to evaluate the proposed methodology.

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 exist that allow enriching raw structured data semantically, they have to be explicitly tailored to the format of the data [4]. Alternatively, systems can be developed that automatically annotate raw data, albeit less accurately [7,3,5,9,2,8]

In this perspective, The “Tabular Data to Knowledge Graph Matching” competition was hosted at the International Semantic Web Conference (ISWC), in 2019 [6]. 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 columns had to be inferred, when possible (Columns-Property Annotation (CPA)). These challenges are illustrated in Figure 1. The competition consisted of four 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

Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

* Both authors contributed equally to this work

B. Steenwinckel, G. Vandewiele et al.

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 three different subchallenges of the competition.

2 System Description

We designed a system to solve these three challenges directly. Our system consists of six phases as visualised in Figure 2. First, crude annotations are made for each cell in the table by generating multiple candidates and disambiguating them by using string similarities on the cell values and the `rdflabel` of each candidate. Afterwards, the column types and properties between columns are inferred using these cell annotations. In a fourth step, the inferred column types and properties are used to create more accurate head cell annotations (the cells in the first column of a table). Phase five uses the new head cells to correct the other cells in the table, using the property annotations. Finally, in phase six, new column types were inferred using all the available corrected cells. The code for this system is written in Python and is made available online¹. In the following section, detailed information of each phase is provided.

2.1 Initial Cell Entity Annotation (cell lookup)

The pipeline used to annotate single cells, during an initial phase, is depicted in Figure 3. For each of the cell values, we first clean them by retaining only the part that comes before a '(' or '[' and by removing all '%', "'", and '\' characters

¹ <https://github.com/IBCNServices/CSV2KG>

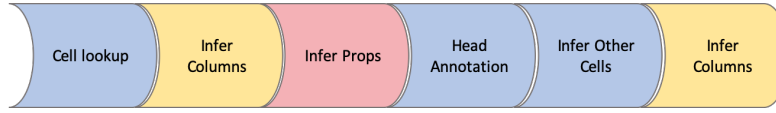


Fig. 2. Overview of each phase in our designed system. The blue phases are used to make cell annotations. The yellow phases provide the column types and the red phase gives property annotations.

(`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 during this initial phase, we do not provide this additional information. If both the `DBPedia lookup` and the `try_url` step did not result in any candidates, the DBPedia Spotlight API is applied. 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.

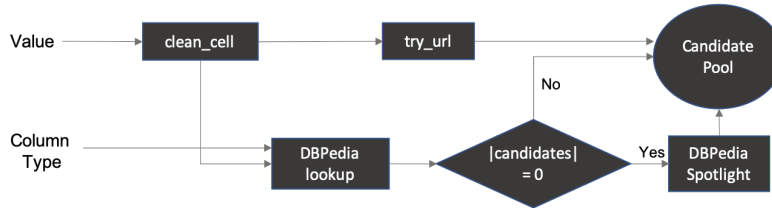


Fig. 3. Cell lookup pipeline: annotations based on the cell value .

2.2 Column Type Annotation (infer columns)

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 find now 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 complete or are 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 2019²) and Shaquille O’Neal his musical endeavours, such as being a rapper and DJ, are not entirely in there³). This makes the inference step far from trivial. Merely taking the type with the highest count, where ties are

² http://dbpedia.org/page/President_of_the_United_States

³ [http://dbpedia.org/page/Shaquille_O’Neal](http://dbpedia.org/page/Shaquille_O'Neal)

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:

$$H(t) = H(\text{SORT}(\{\text{COUNT}(c) \mid c \in \text{CHILDREN}(t)\})[-2 :]) \quad (1)$$

With t the type of which we want to calculate its entropy, $H(\cdot)$ the Shannon entropy, $\text{COUNT}(\cdot)$ a function to get the number of remaining cell annotations of a certain type, $\text{SORT}(\cdot)$ a function to sort a collection in an ascending fashion and $\text{CHILDREN}(\cdot)$ 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 4.

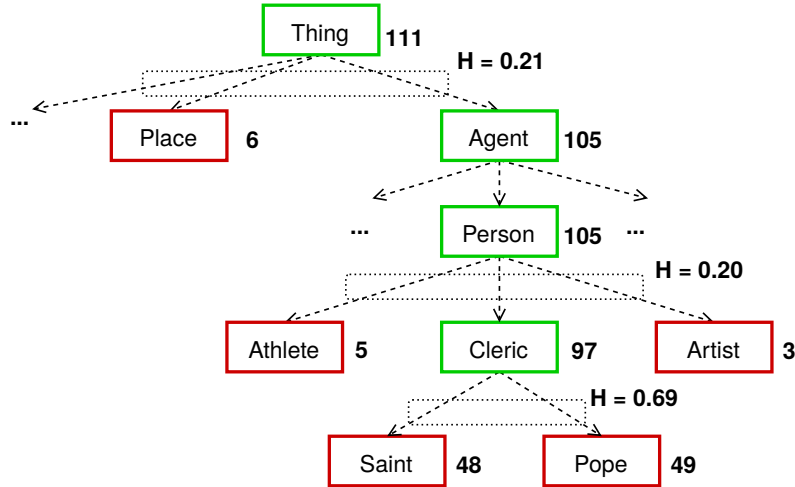


Fig. 4. An example of the column inference heuristic. Majority voting is applied three times until it reaches Cleric. There, the normalized entropy of the two highest counts of its children is too high to continue.

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. Column type information can influence the DBPedia cell lookup as visualised in Figure 3. Therefore, the initial cell lookup was executed again with newly available column information.

2.3 Column Property Annotation (infer props)

The 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 exists 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. To break possible ties, the domain and range of all inferred column types were taken into account from the using a simple query:

```
SELECT ?domain ?range WHERE {
    <pred> rdfs:domain ?domain .
    <pred> rdfs:range ?range .
}
```

with $\langle \text{pred} \rangle$ a possible predicate between the two target columns. So in the case of relationships with equal highest counts, we check the range and domain using the column types. When the range and domain of multiple relationships are valid possibilities, we take the relationships with the most specific range and domain column type (using $\text{depth}(\text{domain}) + \text{depth}(\text{range})$).

2.4 Cell Annotation for the first columns (head annotation)

After obtaining the column type and property annotations, we can create more accurate cell annotations. All properties giving information about the head cells (the cell in the first column of a table) are updated by using the following query:

```
SELECT ?s WHERE {
    ?s <pred> <value> .
}
```

with $\langle \text{pred} \rangle$ the predicates found in Section 2.3 between the head and non-head columns and $\langle \text{value} \rangle$ the cell value of the non-head column as inferred in Section 2.1. For each row in our table, these queries will result in several possible head annotations, and the annotation with the highest count was returned. In the case of an ex aequo, the Levenshtein distance on the `rdf:label` was used.

2.5 Non-head Cell Annotations (infer other cells)

With the corrected head cell annotations, a similar query can be used to correct all the other cells in the table row by row:

B. Steenwinckel, G. Vandewiele et al.

```
SELECT ?o WHERE {  
  <head> <pred> ?o .  
}
```

with `<pred>` the predicates found in Section 2.3 between the head and non-head columns and `<head>` the cell value of the head column as inferred in Section 2.4. Multiple answers are possible when more than one values are annotated with a corresponding property. Levenshtein distances between the `rdf:label` and raw cell value were used to disambiguate the results.

2.6 Final column types (infer columns)

The method from section 2.2 was executed again, but with all new information of the cells.

3 Additional remarks

Some additional elements were provided to boost both the execution and accuracy of our annotation system:

- When names such as G. Vandewiele, B. Steenwinckel were detected, DBpedia lookup was performed given the Person class type, and only the last name was given to the API as a search key. Code was used to check the distance between the cell value and the candidate persons.
- The CTA challenge score was not bounded to 1, so all the parent column annotations were added as well.
- Reasoning was used to find equivalent classes, and these were also added as possible results for the column types.
- Some tables were very similar, and majority voting was used on possible similar tables to correct some of the column types.
- Each phase of the pipeline in figure 2 was parallelized, making it possible to evaluate large amounts of CSV files in a limited amount of time.

4 Evaluation results

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:

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

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:

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

The results of our approach are summarized in Table 1, 2, 3, 4 respectively. It should be noted that 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, therefore, rather low in comparison to our ranking during the other rounds.

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.

Challenge	Primary score	Secondary score	Leaderboard position
CEA	$F_1 = 0.962$	$prec = 0.964$	2 / 8
CTA	$AH = 1.864$	$AP = 0.247$	2 / 8
CPA	$F_1 = 0.841$	$prec = 0.843$	2 / 7

Table 3. The results obtained by our system in the three challenges of round 3.

Challenge	Primary score	Secondary score	Leaderboard position
CEA	$F_1 = 0.907$	$prec = 0.912$	3 / 8
CTA	$AH = 1.846$	$AP = 0.274$	2 / 7
CPA	$F_1 = 0.830$	$prec = 0.835$	2 / 7

Table 4. The results obtained by our system in the three challenges of round 4.

5 Conclusion & Future Work

In this paper, a system to convert a CSV file of raw, structured data to semantic knowledge. The system consists of six phases: three initial annotation phases for all target cells, the types of columns and the properties between columns. A fourth and fifth phase over the target cells utilises the newly inferred properties to create more accurate annotations and a sixth phase which uses all new cell information to improve the column types. The proposed system is rather straight-forward while already achieving promising results. As future work, we would like to improve the system by taking additional resources into account, such as other data sources or embedded values of the occurring triples. 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. Jiaoyan Chen Ernesto Jimenez-Ruiz Oktie Hassanzadeh, Vasilis Efthymiou and Kavitha Srinivas. Semtab2019: Semantic web challenge on tabular data to knowledge graph matching - 2019 data sets. Zenodo. Version 2019.
7. Dominique Ritze, Oliver Lehmborg, 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.
8. Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, 8(6):921–957, 2017.
9. Stefan Zwicklbauer, Christoph Einsiedler, Michael Granitzer, and Christin Seifert. Towards disambiguating web tables. In *International Semantic Web Conference (Posters & Demos)*, pages 205–208, 2013.