Knowledge-Graph-Based Semantic Labeling: Balancing Coverage and Specificity

Ahmad Alobaid [a] and Oscar Corcho [a]
[a] *Ontology Engineering Group, Universidad Politécnica de Madrid, 28660 Boadilla del Monte, Madrid, Spain*
E-mails: aalobaid@fi.upm.es, ocorcho@fi.upm.es

Editors: First Editor, University or Company name, Country; Second Editor, University or Company name, Country
Solicited reviews: First Solicited Reviewer, University or Company name, Country; Second Solicited Reviewer, University or Company name, Country
Open reviews: First Open Reviewer, University or Company name, Country; Second Open Reviewer, University or Company name, Country

Abstract.
Many data are published on the Web using tabular data formats (e.g., spreadsheets). This is especially the case for the data made available in open data portals, especially by public institutions. One of the main challenges for their effective (re)use is their generalized lack of semantics: column names are not usually standardized, their meaning and their content are not always clear, etc. Recently, knowledge graphs have started to be widely adopted by some data and service providers as a mean to publish large amounts of structured data. They use graph-based formats (e.g., RDF, graph databases) and often make references to lightweight ontologies. There is a common understanding that the reuse of such tabular data may be improved by annotating them with the *types* used by the data available in knowledge graphs. In this paper, we present a novel approach to automatically *type* tabular data columns with ontology classes referred to by existing knowledge graphs, for those columns whose cells represent resources (and not just property values). In contrast with existing proposals in the state-of-the-art, our approach does not require the use of external linguistic resources or annotated data sources for training, nor the building of a model of the knowledge graph beforehand. In this work, we show that semantic annotation of entity columns can achieve good results compared to the state-of-the-art using the knowledge graph as a training set without any context information, external resources or human in the loop.

Keywords: Semantic Annotation, Knowledge Graph, Semantic Labeling

1. Introduction

An enormous amount of data is currently available on the Web. Efforts in the literature to crawl the web found around 150 million Web tables [1, 2]. And with the increased adoption of open data by public institutions worldwide, the amount of data on the Web is increasing exponentially. Although many recommendations and best practices exist on how such data should be published to encourage more reuse (e.g., the 5-star scheme of open data[1]), most of such data are still being published at most with 2 or 3 stars, that is, using spreadsheets (CSVs or Excel files).

From a practical point of view, this means that such datasets are not semantically annotated[2], making them more difficult to understand and use. Many reasons may exist for this, but one generally-agreed on is that

[1] http://5stardata.info/en/
[2] Note, however, that according to the 5-star scheme, there is no real need for semantic annotations, but only the usage of RDF and the provision of links to other datasets

data providers do not have sufficient tools to help them in publishing data in a more understandable and usable manner.

One possibility to overcome this situation is to create the tools (or services) that are able to generate semantic annotations for those data sources. This process is normally coined as *semantic labeling* [3–6]. The process of semantic labeling is also referred to as *semantic annotation* in the literature [3, 5, 7]. The resulting data (once the semantic annotations are exploited) may be available as virtual or materialized RDF data sources [8].

The resulted semantic annotations may refer to any existing ontologies. In our work, we focus on the (generally lightweight) ontologies that are being used for structuring and annotating knowledge graphs. Knowledge graphs are rich structured data sources that contain data that are usually annotated with semantic types[3]. An example of such knowledge graph is DBpedia [9], which contains knowledge extracted from Wikipedia, by means of a crowdsourced set of mappings that are used to connect Wikipedia infobox templates to the user-generated DBpedia ontology.

The semantic annotation of tabular datasets is usually done manually (e.g., using Open Refine[4] and its RDF plugin) or semi-automatically (e.g., using Karma[5]). Manual annotation is tedious, error-prone, and does not scale. Whilst semi-automatic annotation requires a lot of manual annotation steps so that the machine learning module is able to learn how to perform semantic annotation.

In this paper, we describe our approach for the automatic semantic annotation of tabular datasets, so as to overcome the limitations from manual and semi-automatic approaches, as well as to generate relationships with existing knowledge-graph related ontologies. More specifically, we focus on the annotation of those columns in existing tabular datasets that refer to entities. That is, we refer to columns that contain potential resources: the subject of matters that the source is explaining (sometimes these columns are referred to as subject columns[10]). In summary, we label entity columns[6] of the (input) tabular data sources with *types*[7] from a given knowledge graph.

The main contributions presented in this paper are:

1. A new approach to automatically label subject columns in tabular datasets with semantic types, given an existing knowledge graph and in the absence of context[8].
2. A new set of scoring functions (to determine the *type* applicable to a column), which consider the trade-off between covering as many of the entities as possible while being as specific as possible.

Although our approach out-performs the state-of-the-art, the primary goal in this paper is to show that using the knowledge graph as the training set without using any other context and external sources of information we can semantically annotate entity columns and obtain good results compared to the state-of-the-art.

We validate our claim by testing our approach against three predefined datasets. One that we have gathered in the domain of the Olympic Games, which we manually annotated. The second one is a collection of Web Tables that have been crawled from the web and transformed into CSV files (referred to as Web Data Commons). The third one is a newer version of Web Data Commons, which is cleaner and slightly bigger.

The rest of the paper is organized as follows. In Section 2, we survey the most related research papers to our approach. Then, explain our approach and the used scoring functions in Section 3. We describe the experiment and discuss the results in Section 4. Finally, we conclude the paper and show future lines of work that we would like to explore in the future (Section 5).

2. State of the art

Different approaches have been proposed so far to perform semantic labeling, understood as the process of assigning *types* from knowledge bases to values from any data source. In this section we describe some

[3]As far as we know, there is no definition that it widely used and agreed upon in the literature. When we refer to the term *knowledge graph* here, we are referring to the common understanding and usage of the word in the semantic web community, which is (arguably) analogous to the term *knowledge base*

[4]http://openrefine.org

[5]http://usc-isi-i2.github.io/karma/

[6]In this paper, we use the terms “subject columns” and “entity columns” interchangeably.

[7]We use the term type to refer to ontology classes, since this is a term commonly used in the knowledge graph literature

[8]such as table caption, title, .. etc

of the most relevant approaches focused on tabular datasets.

Cafarella et al. [1] describe an approach for searching and linking Web tables, which exploits an attribute correlation statistics database (ACSDb) that contains the frequency of occurrences of schemas and attributes. Their ranking algorithm uses a linear regression estimator with different features (hits on the table header, leftmost column, table body, and a schema coherency score about how two items are related).

Limaye et al. [11] use probabilistic graphical models for semantic labeling, entity detection and relation extraction using YAGO. They use column type, entity, and the relation between two columns to construct the features, which are based on cosine similarity of the cell and column headers, compatibility of the entity and semantic types, and the compatibility between different column types and entity pairs which are weighted using SVM.

Syed et al. [12] use Wikitology, a knowledge base that contains entity information from Wikipedia. They query Wikitology for each string in the column (each cell), apply a scoring function based on page rank and entity rank (using Wikitology), and pick the most predicted types. They do the same for relation discovery between columns.

Venetis et al. [13] semantically label Web tables using two databases: an isA and a relation database. The isA database is used to identify the class of each column. After that, they inspect the relation between two columns using the relation database, which is in the form of (a, R, b), where a is an instance of class A, b is an instance of class B, and R is the relation between a and b.

Goel et al. [14] semantically label source attributes using Conditional Random Fields, exploiting the latent (hidden) structure within the data. They tokenize the values and apply features depending on the token type (e.g., token length, value, the starting character, whether it is capitalized, whether it is negative, starting digits, unit). They consider the relationship between neighboring labels, tokens and attribute labels.

Zhang et al. [10] match and semantically annotate numeric time-varying attributes in Web tables after splitting them into (n-1) tables, the entity column with each of the other columns. They take into account their headers and context (e.g., surrounding text, web page title, table caption, etc.). They connect tables using manually-added conversion rules (unit, scale).

Ritze et al. [15] present the T2K iterative matching algorithm to match Web tables to knowledge bases. Their approach performs entity linking and schema annotation, each influences (improve) the other. It iterates over Candidate Selection from DBpedia and Value-based Matching (using value-based similarities for the attributes) adjusting and filtering until there is no more change.

Ramnandan et al. [16] present an approach that assigns properties from an already aligned domain ontology to the target data source relying on the data. Their approach treats textual and numeric properties differently (anything that is not a number is treated as text). For textual data they use cosine similarity using *term frequency* (TF) and *inverse document frequency* (IDF). For the numerical values, they compare the distributions using the Kolmogorov-Smirnov (KS) test.

Ermilov et al. [17] detect subject columns using the number of relations between different columns as an indicator (assuming binary relation). They rely on AGDISTIS [18] for entity disambiguation and they use DBpedia as the source of knowledge. For column annotation, they use the header to get potential properties and then rank them according to their frequency in the knowledge base.

Taheriyan et al. [3] build a semantic model that represents the relationship between dataset fields rather than only annotating attributes as semantic types. Data sources are semantically typed, the semantic labeling with confidence intervals is used to construct the semantic model, and a graph with links is built that corresponds to candidate types inferred by the ontology.

Pham et al. [19] propose a semantic labeling approach based on logistic regression. The features they rely on are similarity measures using Jaccard similarity and TF-IDF besides the attribute name (in the header), Kolmogorov-Smirnov and Mann-Whitney tests. The weight of each feature is calculated (which depends on the training data) and is used afterwards for classifying the datasets.

Neumaier et al. [4] aim to create a context for the semantic labels instead of mapping properties only. They represent that as a tree with each children being a context and build a hierarchical background knowledge graph using *rdfs:subClassOf* and property-object pairs. For predicting new data sources, they use the Kolmogorov-Smirnov test and nearest neighbors over the background knowledge graph.

Quercini et al. [20] focus on entity linking of cells using Bing search. They perform text classification on the snippets of the resulted web pages. They train their models with snippets from DBpedia and use SVM for entity linking. Although their algorithm works automatically, the training for the types from DBpedia to get the snippets for text training is performed manually They also use regular expressions to detect specific types (e.g., phone numbers, emails) and TFG's functionality to narrow down the possible types (e.g., Location, Date). They use spatial information to disambiguate the entity linking (utilizing "Google Geocoding API ").

Zhang [21] presents a way to perform semantic annotation for entity and literal columns in web tables taking into account the headers, caption, surrounding text in the webpage, and existing RDFa annotations. The approach uses the digest of the search engine results for entity disambiguation. The author argues that by annotating a subset of a column, the type of the column can be inferred. The idea is to start with an erroneous annotation, and then iteratively improve the annotation taking into account inter-column dependency. It annotates every column that represents entities and detects columns that correspond to properties in the knowledge base. Some of the additions of this paper compared to previous work [22] are: the detection of entity (subject) column, column relation detection, and annotation improvement.

Tonon et al. [23] present an approach to rank entities based on their relevance in a textual context. They use an inverted index for literals matched to their URI in DBpedia using [24, 25] to perform entity linking. They create a single type hierarchy with DBpedia, YAGO, and schema.org using *owl:equivalentClass*, PARIS [26] (which include mappings between DBpedia and YAGO) with some manual tweaking from domain experts. They use an external RDF dataset [27] to retrieve the types. They propose three approaches. The first approach uses the relation between the entity and other entities in the knowledge graph. The second approach considers the occurrence frequency of the entity and its types with other entities in the same context. The third approach relies on the type hierarchy of the entities, favoring deeper types in the hierarchy. They combine the three approaches and weight them based on a decision tree.

Nuzzolese et al. [28] present a tool called Tìpalo. It extracts the definition of the entity from Wikipedia uses FRED [29] to generate RDF of the entity definition and filter candidate types using graph-pattern-based heuristics. Then, it disambiguates candidate types using [30] and aligns them to OntolWordNet, WordNet and DUL+DnS Ultralite.

Dong et al. [31] focus on the performance and adopt a MapReduce-based approach. They explore common similarity functions (e.g., Jaccard, Cosine) and consider a cell value and an entity as similar if they share a common signature. They organize the types into disjoint groups taking into account the type hierarchy and use the hash method to compute the overlap similarity. Then they pick the top-k candidate types for each column. They improve the performance using a partition framework that prunes unnecessary entity type pairs taking into account a bloom filter [32] to represent the entities in the containing partition, which is improved further with bloom filter hierarchy [33].

Hassanzadeh et al. [7] use a MapReduce approach based on the work of [31] and an extension of [34]. They transform the input CSV files and the reference knowledge graph into key-values. The keys are URIs for the column in the case of CSV files and class URIs for knowledge graphs. The values for column URIs are cell values, and values for class URIs are instance labels. They perform overlap similarity analysis between the values of the input CSV and the instance labels.

Ritze et al. [35] present an extended version of their previous work [15]. They use ensemble for different matchings taking into account context features (e.g., Page title, URL) and in-table features (e.g., labels and values). They perform three annotation tasks: row-to-instance, attribute-to-property, and table-to-class. Their approach shows that taking into account all features with weights outperforms all other combinations of the features.

From this initial analysis of the-state-of-the-art in semantic annotation, we can summarize in the following set of observations:

– *Learning from the same set:* using other tabular data (from the same or different dataset) to match the tabular data rather than using a semantic source of knowledge [1, 10]. In other words, these approaches link similar data rather than annotating them with semantic types. Such an approach does not ensure the interoperability and usability of such annotations. In our approach, we annotate datasets with semantic *types* from a given knowledge graph, what makes it much easier to exploit and integrate with other datasets [4, 15].
– *Relying on search engines:* using Web search engines to disambiguate entity linking, such as Zhang [21, 22] and Quercini et al. [20], what makes them dependent on and bound to the search engine used (even if it is not complete reliance, and other features are used as well).

- *Preprocessing and profiling beforehand:* some approaches require building a model before being able to annotate the input sources [4, 12], and sometimes building an external index, such as Tonon et al. [23]. They built an inverted index over the whole DBpedia to increase the performance. This is usually expensive in terms of storage, and time. Our approach creates the model once the input file is provided, and such an approach is feasible as the model is so small that it only contains semantic types related to the provided file.
- *Depending on fixed data sources:* the work by Syed et al. [12] relies on Wikipedia and Wikitology (even though this index may be rebuilt again with other similar sources, it is a complex process) while Nuzzolese et. [28] use the entity definition in Wikipedia. Tonon et al. [23] relies on PARIS [26] for the alignment of the type hierarchy. In our case, we use DBpedia as the learning source, but other sources can be used without any major change in the code. Since our approach does not perform any preprocessing either, changing to another knowledge graph is straightforward. Some approaches share the same advantages as ours, using SPARQL endpoints as learning sources, such as YAGO in [11] and DBpedia in [4, 15], what makes them flexible and applicable to a wider range of training sets.
- *Manual intervention:* despite the fact that these approaches may be automatic or semi-automatic[9], some of them actually require manual actions (e.g., provide predefined conversion rules [10, 15], manually tweak the type hierarchy [23], ontology alignment [16], a black list of properties [4] to improve the accuracy, and abbreviations resolution [10, 15]). Our approach is completely automatic and does not contain such manual crafting, what makes it more easily adoptable to different datasets and knowledge graphs.

3. Approach

In this section, we explain our approach to annotate columns in a given dataset with classes used in an existing knowledge graph. We start with a simplified working example, so as to illustrate the approach, and continue with the description of the algorithm and the scoring functions.

Working Example

Let us consider a tabular data file as shown in Fig. 1a. The entity column that we are interested in is the one with the “Player name" header. The first step of our approach consists in annotating each cell in the entity column, in our case “Facundo Campazzo". We query the knowledge graph for an entity that has the name “Facundo Campazzo". In DBpedia, the entity URI is http://dbpedia.org/resource/Facundo_Campazzo. Then, we query the knowledge graph for the classes to which “Facundo Campazzo" belongs and assign these *types* to “Facundo Campazzo" (Fig. 1b). We do the same for the other cells in the column. After that, we build the class graph that contains all the types of the entities linked to the cells. In our example, we build the graph of the types of “Facundo Campazzo” (Fig. 1c). Note that in that graph, we have smaller circles that do not contain any text, these are just to show that realistically, we may have other types (for other cells) that are not ancestors of *basketball player*. The last step is to score the class graph using Equation (1) and pick the *type* with the highest score, which is *basketball player*.

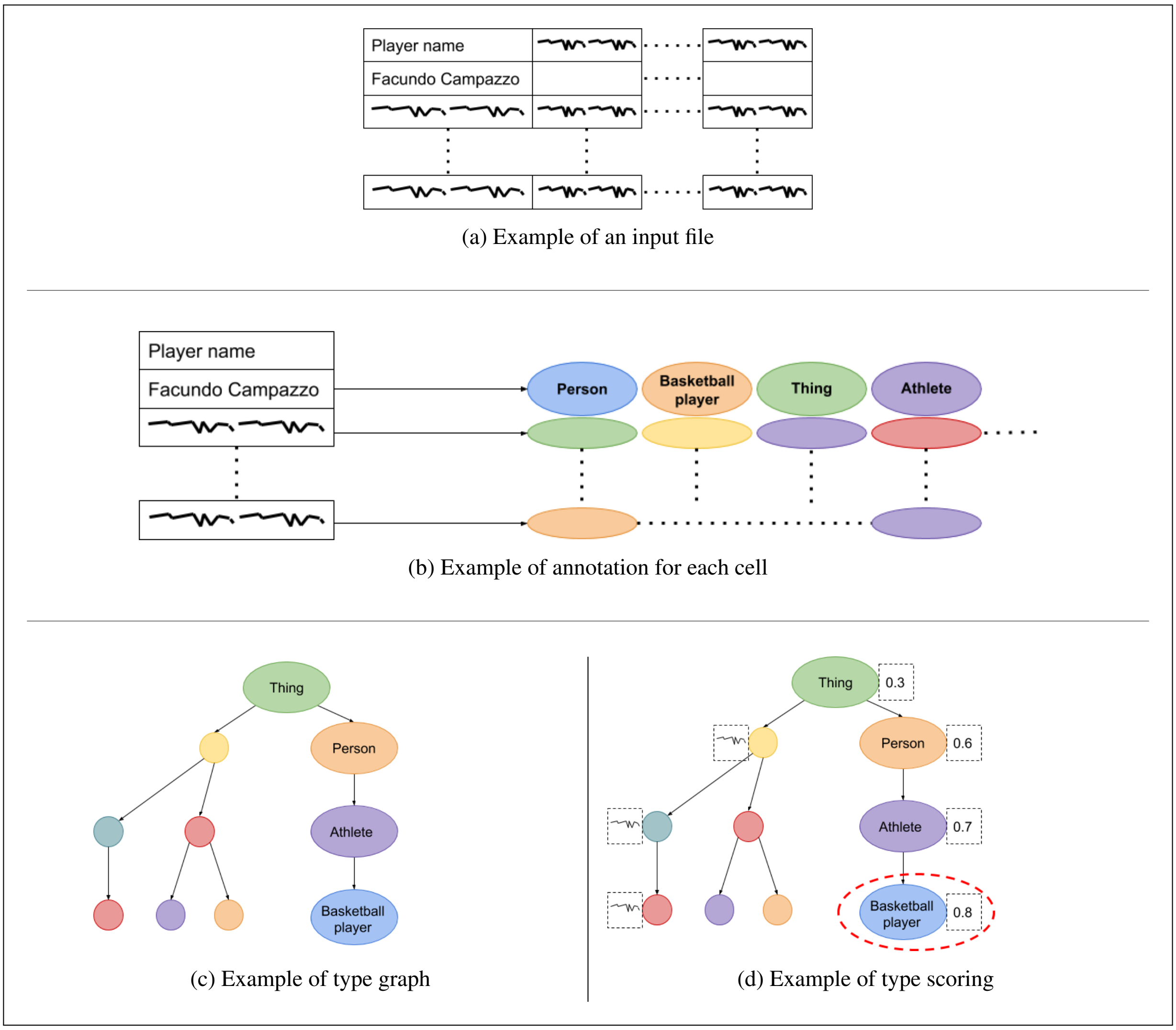
Fig. 2 provides an alternative view of our approach. We approach the problem by first performing entity linking in the tabular datasets to the corresponding ones in the knowledge graph i.e. each cell is *typed* if possible. As in Fig. 2 (step 1), the first step takes the input files (to be annotated) and a knowledge graph to annotate the cells. The next step is to build a class graph of the *types* gathered in the first step. It takes as input the annotated cells and the knowledge graph to build a class graph (hierarchy) from the *types* of the cells (step 2 in Fig. 2). The last step is to score each class in the class graph using the formulas in Section 3. The class with the highest score is picked to be the *type* of the entity column (step 3 in Fig 2).

Assumptions

We consider several assumptions in our work:

- There is a single entity column, and it is not spread over two or more columns (e.g. a column for first name and another column for last name). This is not a major limitation though, and we will consider it as part of our future work.
- The entity column may be identified by the user. In any case, in our automatic process with large

[9]We are not referring here to the gold standards that are built manually for evaluation purposes or the semantic models that are constructed by domain experts



(a) Example of an input file

(b) Example of annotation for each cell

(c) Example of type graph

(d) Example of type scoring

Fig. 1. Example of semantic annotation

corpora, we consider the first column to be the entity column. Nonetheless, our approach works without this assumption without any loss of generality i.e. it would work with entity columns in the middle or at the end of the data source, given that they are identified correctly. Furthermore, Cafarella and colleagues [1] found that the second most heavily weighted features to search for subject columns in Web Tables is the number of hits in the left-most column.

– The input tables are vertical where the header is the first row (if there is one). Nonetheless, this does not limit our approach as horizontal tables can be transposed easily, it is just a matter of detecting whether the table is vertical or horizontal.

– The knowledge graph is a SPARQL endpoint that uses RDF, but this can be easily extended and generalized.

Step 1: Type individual cells

For each cell in the entity column in the input data, we assign a potential list of entities (in the form of URIs). We get the list of entities by querying the knowledge graph for entities having the cell's value as the object of some triple (Listing 1). In this paper, we do exact matching to get entities for a given cell. This may be extended in the future to other types of matching. After that, for each entity matched to a cell, we get

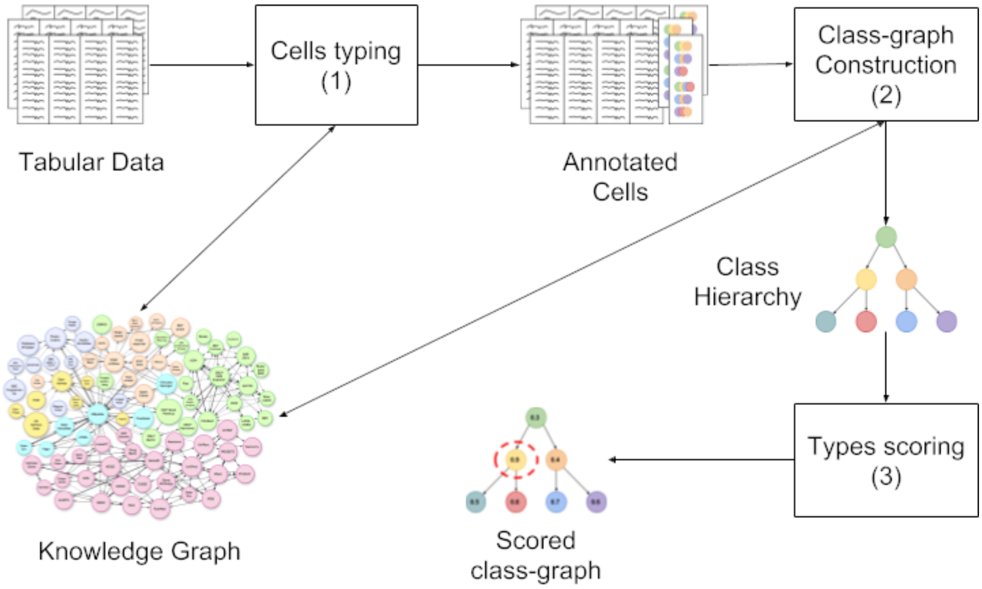


Fig. 2. Semantic annotation workflow

the list of *types* for that entity (Listing 2) and the *types* of a single cell are the *types* of that cell's entities.

```
select distinct ?subject
where{ ?subject ?property "Facundo␣
    Campazzo"@en }
```

Listing 1: Get entities for a cell

```
prefix dbr:<http://dbpedia.org/
    resource/>
select distinct ?class
where{dbr:Facundo_Campazzo a ?class}
```

Listing 2: Get classes to which an entity belongs

Step 2: Build The Class Graph

At this point, we have assigned a list of classes to each entity. Note that given our exact match restriction on labels, we may not find classes for all. However, at this stage this is not too relevant, since we can still obtain good results independently of this.

We build the class graph by first gathering the list of classes for each cell for each entity. We query each class to get its parents (Listing 3). If a class has no parents, then it is a root, otherwise, the parents will also be queried for their parents if not already in the graph and will be linked accordingly.

```
prefix dbo:<http://dbpedia.org/
    ontology/>
select distinct ?parent
where{ dbo:BasketballPlayer
        rdfs:subClassOf ?parent. }
```

Listing 3: Get parent classes for an entity

Step 3: Score Graph Nodes

In choosing the most suitable class in the class graph, there are two contradicting preferences: 1) prefer the most specific types; 2) prefer types that cover the majority of instances. In terms of the class graph, the most specific classes tend to be in the leaves while the nodes (classes) that cover the most are in the top of the graph; typically, the root covers all the instances. We propose a formula to weight the two features and maximize the sum over the nodes in the graph, Equation (1). The coverage score of a *type* (node) t is denoted by $f_c(t)$

and the specificity score by $f_s(t)$. Alpha (α) is used to weight the coverage score and the specificity score of a node t (related proofs are in Appendix A). The specificity score indicates how specific a *type* t is, while the coverage score denotes how much a *type* t covers. Equation (1) maximizes the score f by choosing the *type* t that maximizes the sum of the specificity score and the coverage score.

$$\arg\max_t f(t) = \alpha * f_c(t) + (1-\alpha) * f_s(t) \tag{1}$$

Coverage

Coverage score indicates how many instances a *type* t covers. The higher the coverage score f_c is, the more entities the *type* t covers. The basic idea is to have for each cell a score of 1. This cell score will be shared among the entities of that cell and will be divided evenly. The score of each entity will also be divided evenly among its *types*. This is shown in Equation (2), where v is a text value in the entity column, Z returns the entities of a given text value, e is an entity, and Q returns the classes of a given entity. $\|Z(v)\|$ represents the number of entities from the knowledge graph that are linked to v and $\|Q(e)\|$ denotes the number of classes for e that are returned from the knowledge graph.

$$\begin{gathered} I_c(t) = \sum_v \sum_e \frac{1}{\|Z(v)\| \|Q(e)\|} \\ \forall v, e : e \in Z(v) \text{ and } t \in Q(e) \end{gathered} \tag{2}$$

For the coverage, we also want a *type* t to include the coverage of its children. We represent that in Equation (3), where u is the child of t. If t does not have any child, its value will be $I_c(t)$.

$$L_c(t) = I_c(t) + \sum_u L_c(u) \tag{3}$$

Since the value of L_c will increase as the number of cells increase, we normalize the coverage score by dividing it by m, where m is the number of cells that are matched to at least one entity.

$$f_c(t) = \frac{L_c(t)}{m} \tag{4}$$

Specificity

Specificity score indicates how specific/narrow a given *type* t is. High specificity score means a very narrow/specific *type*. We follow the intuition that a very narrow *type*, has a fewer number of entities compared to another *type*. For example, the number of physicists is much less than the number of scientists and the number of scientists is much less than the number of humans. Saying that a person is a physicist is more specific than saying that this person is a scientist. Another way to look at that is if we picked a human randomly, the probability of this person being a physicists is lower than the probability of this person being a scientist. In accordance with information theory, as the probability decreases, the value of the corresponding piece of information increases.

Therefore, we divide the number of entities of a *type* t which is $\|R(t)\|$ by the number of entities of its parent ($\|R(p)\|$), where p is the parent with the highest number of entities. Note that the number of entities here is the number of entities returned from the knowledge graph. In case the t does not have parents, the value of $I_s(t)$ will be 1.

$$I_s(t) = \frac{\|R(t)\|}{\|R(p)\|} \tag{5}$$

The score I_s is computed from the *type* t to the root; this is done by multiplying I_s for all, along the way for each node. In the case of multiple paths, the lower L_s is chosen. In case a class t has no parents, its L_s value will be set to 1.

$$L_s(t) = I_s(t) * L_s(p) \tag{6}$$

More specific *types* yield lower L_s. But we want to maximize both, f_c and f_s. We need a scoring function that increases as the value of L_s decreases.

We present multiple formulas to compute the specificity of a given type t. All of them follow the intuition we mentioned earlier. Relevant proofs are provided in Appendix A and we also show how we obtain the different functions in Appendix B.

$$f_{s1}(t) = \sqrt{(1 - L_s(t)^2)} \tag{7}$$

$$f_{s2}(t) = -L_s(t)^2 + 1 \tag{8}$$

$$f_{s3}(t) = -L_s(t) + 1 \quad (9)$$

$$f_{s4}(t) = 1 - \sqrt{L_s(t)} \quad (10)$$

$$f_{s5}(t) = (1 - \sqrt{L_s(t)})^2 \quad (11)$$

When we talk about the specificity function f_s, it can refer to any of the specificity functions we mentioned above. In the evaluation section we test all of them and eventually we will pick the best one the yields the highest score.

After computing the coverage score f_c and the specificity score f_s for each *type*, we compute the overall score f for each class in the graph following Equation 1. Finally, the *types* are ordered in decreasing order by the value of f, as the node with the highest score represents the *type* of the entity column; the other top types in the list are picked if the first one is rejected by the user (see Section 4.1)

Implementation

We have built a Web application [37] to semantically annotate tabular datasets automatically using a knowledge graph. Our application takes as an input the URL of the SPARQL endpoint and the CSV files to be annotated. It semantically annotates the entity columns and shows the annotations as a list of classes ordered descendingly from the highest score to the lowest score.

4. Evaluation

We prove the existence of a value α, i.e. Equation (1) results in a correct type. A correct type t will have the highest score among the rest of the types for the entities linked to the cells in the entity column of a given CSV file (see Section A.3 in the Appendix). We also prove that the coverage score of a parent's node has a higher coverage score than any of its children (Section A.1 in the Appendix). The same way, we prove that a child's node (type) has a higher specificity score than its parent (Section A.2 in the Appendix).

In this section, we focus on an experimental evaluation where we aim at evaluating that our approach can automatically assign semantic types to entity columns given a knowledge graph without using other context or external sources of information and yields good results in comparison to the state-of-the-art.

For that reason, we experiment with three different datasets: Olympic Games [38], Web Data Commons version 1 [39], and Web Data Commons version 2 [40]. We refer to Web Data Commons version 1 as T2Dv1 and T2Dv2 for version 2. Thanks to those experiments, we also show that our approach works with realistic datasets. We explain below the details of the experiment and discuss the results.

4.1. Experiments

We experiment with three different datasets. The first one is composed of a collection of CSV files that we gathered about Olympic Games 2020[10]. The second and third datasets are presented in the state-of-the-art as Gold Standards to allow comparison of different annotation approaches. In all of our experiments, we use the English DBpedia because T2Dv1 and T2Dv2 are annotated with the DBpedia classes, so that we can compare the results. We report scores for each dataset, and we calculate the accuracy with different specificity functions.

4.1.1. Olympic Games

We mentioned the details of how we built the dataset in our previous work [5]. We have the data publicly available to allow others to compare [38]. It is a small dataset compared to the other ones; it contains 12 CSV files. All of the subject columns are located in the leftmost (the first column from the left).

Preprocessing: We did a single data transformation, which is merging the columns "FIRSTNAME" and "SECONDNAME" as our approach does not work if the entity column is separated into two columns. We performed that on the files themselves (we updated the files before feeding them to our application). Obviously, this may be easily automated in the future as an additional feature of our implementation.

We apply the approach on each file, performing entity linking and constructing the type graph hierarchy. We compute the coverage and the specificity scores - trying all the specificity functions. We compare the suggested types with the correct one that we handpicked and show the results in Table 1.

[10] https://en.wikipedia.org/wiki/2020_Summer_Olympics

4.1.2. T2Dv1

It is a collection of Web Tables that have been crawled from the Web. They have been manually annotated with DBpedia classes. The dataset is composed of 233 Web Tables. They are not very clean as they were automatically transformed into CSV files (we can see some HTML tags and special characters that are not related to the content of the Tables).

Although this dataset has the annotation of each subject column, we did not find the id of the subject column in the dataset. Since our application expects the id of the subject column, we went through the 233 Tables and manually identified the subject column for each of them. We also eliminate (leave) classes that are not classes of DBpedia (e.g., YAGO classes) as T2Dv1, is annotated with DBpedia classes and we want to compare the scores.

Preprocessing: Performing the test for the first time resulted in a very low recall (less than 0.1). This was due to the fact that the labels in the subject columns were in lowercase. This made it difficult to find the entities from the knowledge graph as we were using a naive exact match. To overcome this, we changed the values in the entity column to *title case*. This is not done on the actual CSV files, but rather included in the application and can be turned on or off by passing a flag.

We ran the application on a MacBook Pro laptop with 2.8 GHz Intel Core i7 with 8 GB of RAM. The application took around 7 hours to compute coverage and the different specificity functions with different values of α.

For a given file, if no annotation matches the one class that is given in the gold standard, we consider it as a wrong annotation, even though some of them looks correct or acceptable to us (e.g., typically people do not use *dbo:AdministrativeRegion*, but they use *place* or *city* instead). We use a testing script to verify if the annotation generated by our application is the same or different than the one generated by T2Dv1. Since different tables can have different values of α that are optimal (which is also reported by Ritze et al. [35] that a single set of weights might not be the best for all tables[11]) the testing script tries with different values of α[12].

[11] Note that our approach does not use the same features or weights.

[12] $\alpha \in [0.45, 0.4, 0.35, 0.3, 0.25, 0.2, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001]$. This turns out to be a good set of values for α to balance coverage and specificity.

Table 1

Semantic-labeling scores for Olympic Games with different specificity functions (and the same coverage function)

f_s	Precision	Recall	F1
f_{s1}	1.0	1.0	1.0
f_{s2}	1.0	1.0	1.0
f_{s3}	1.0	1.0	1.0
f_{s4}	1.0	1.0	1.0
f_{s5}	1.0	1.0	1.0

$f_{s1} = \sqrt{(1 - L_s(t)^2)}$ $f_{s4} = 1 - \sqrt{L_s(t)}$

$f_{s2} = -L_s(t)^2 + 1$ $f_{s5} = (1 - \sqrt{L_s(t)})^2$

$f_{s3} = -L_s(t) + 1$

4.1.3. T2Dv2

This dataset is similar to T2Dv1, but we notice that it is cleaner (we did not see HTML tags in the tables). For this reason, we did not need to perform any transformation for the labels in the subject column (as they were not lowercased). Also, the subject column ids are already provided, so we did not need any detection or to manually identify them. The dataset includes 237 files, slightly bigger than T2Dv1. This dataset has been annotated with DBpedia classes, so in the test, we eliminate any non-DBpedia classes (unless it is a parent of a DBpedia class). The reason is that if we eliminate non-DBpedia classes in the parent classes, the graph will not have a single graph with *owl:Thing* as the root class.

Preprocessing: The dataset is composed of JSON files rather than CSV files, so we transformed them into CSV files with only the subject columns. No further preprocessing was required for T2Dv2.

The experiment was executed on the same machine that we used for T2Dv1, and the time for the experiment was similar (around 6 hours). We use the testing script as the one used in the previous experiment with T2Dv1.

4.2. Results and Discussion

In Table 1, we report the scores for semantic annotation of the subject columns in the Olympic Games dataset. We reach a perfect precision and recall. We have a proof of the scoring functions that we use in Appendix A. Our approach reach perfect score given the fact that these data follow our intuition as we mention in Appendix A and Section 3. Also, we do not expect to have many cases where an incorrect type t_w is more prominent than the correct type t_r given that the

Table 2
T2Dv1 Precision and Recall

Approach	Precision	Recall	F1
T2K	0.94	0.94	**0.94**
TADA-Entity (f_{s1})	0.71	0.96	0.82
TADA-Entity (f_{s2})	0.78	0.96	0.86
TADA-Entity (f_{s3})	0.93	0.97	**0.95**
TADA-Entity (f_{s4})	0.88	0.97	0.92
TADA-Entity (f_{s5})	0.88	0.97	0.92

Table 3
T2Dv2 Precision and Recall

Approach	Precision	Recall	F1
T2K (Majority)	0.47	0.51	0.49
T2K (Majority + Frequency)	0.87	0.90	0.89
T2K (Page attributes)	0.97	0.37	0.53
T2K (Text)	0.75	0.34	0.46
T2K (Majority + Frequency + Page attributes + Text)	0.90	0.86	0.88
T2K Extended	0.93	0.91	**0.92**
TADA-Entity (f_{s1})	0.68	0.96	0.79
TADA-Entity (f_{s2})	0.75	0.96	0.84
TADA-Entity (f_{s3})	0.91	0.97	**0.94**
TADA-Entity (f_{s4})	0.85	0.97	0.90
TADA-Entity (f_{s5})	0.84	0.97	0.90

$f_{s1} = \sqrt{(1 - L_s(t)^2)}$
$f_{s2} = -L_s(t)^2 + 1$
$f_{s3} = -L_s(t) + 1$
$f_{s4} = 1 - \sqrt{L_s(t)}$
$f_{s5} = (1 - \sqrt{L_s(t)})^2$

difference in the depth is close (if not, it can be manipulated with the α value to give more weight to the coverage f_c or specificity f_s). Since this is the case for the Olympic Games dataset, achieving a perfect score is not surprising.

4.2.1. *Web Data Commons v1*

We report the results of our experiment with the Web Data Commons v1 dataset in Table 2. We see that the highest precision that we reach using our approach (TADA-Entity with f_{s3}) is 0.93, against T2K, which reached 0.94, but we achieve a higher recall than T2K in all the different specificity functions. This results in our approach (with f_{s3}) having a higher F1 score.

One of the main reasons that affect the precision score is the wrong entity linking; which is due to the use of naive entity linking.

For example, many companies are named after their creator. An example of that is the famous Jewellery company “Cartier,” which is named after its founder Louis-François Cartier. Another reason is missing data from the knowledge graph. This was the case for lakes labels, which are linked to boxers, sports teams, and places that share the same name. For example, one of them has the name “Molina,” which is linked to a city in Chile, to a soccer club “CF Molina,” to a cyclist “Juan Molina,” to an artist “Ralph Molina,” and not to a Lake labeled “Molina”.We did not take into account that famous labels are more prone to be wrongly annotated if only labels are taken into account. For example, the label “Leon,” which is a city in France, also has as candidate entities: a Japanese wrestler, a music artist, and a scientist. It could be intuitive to take into account what people think when the word “Leon” is first introduced to them. However, that also means that facts or labels not commonly known will be more prone to be misclassified (given that a common fact about an entity with the same label exists). This could be settled using other kinds of insights like properties in the tables. This also can be limited if the properties in the table do not exist in the knowledge graph. Our approach scored 0.01 lower than T2K (in precision), but still outperformed T2K with an F1 score of 0.95 (for the highest specificity function f_{s3}) and recall of 0.97.

4.2.2. *Web Data Commons v2*

This dataset is manually annotated with DBpedia classes. We compare the annotations produced by our approach with the annotations reported in T2Dv2. We compare the performance of our approach with the results reported by Ritze et al. [35] (referred to as T2K Extended). They also show different baselines: Majority (how often the classes occur), Majority+ Frequency (taking into account the specificity[13]), Page attributes (e.g., Page title, URL, ...), Text (abstracts belongs to the classes), Majority+ Frequency + Page attributes + Text[14] . We report the results in Table 3.

We see that the best precision of our approach is 0.91 which is lower than the T2K Extended approach by 0.02.

The precision of T2K (Page attributes) is very high (0.97), but covers only a small set as it has a low recall (0.37). The recall of our approach is 0.97, which is higher than T2K (0.91) and all the other baselines

[13] It is different specificity function than the one we use in our approach.

[14] For more details on the baselines refer to the original paper by Ritze et al. [35].

reported. The F1 of our approach (0.94) is higher than all the T2K approaches.

Looking closer to the wrong annotations, they are similar to the ones reported in T2Dv1. For example, there is a table about "Cricketer," the table is annotated more as "OfficeHolder." This is due to the fact that the entity linking is more linked to Politicians than Cricketers because Politicians are more common in the knowledge graph than Cricketers and also due to common names. We observe that this problem is the most common one in entity linking that are based on labels only. This also agrees with [20], as they reported the confusion of annotating entities with the class *Person*.

5. Conclusion and Future Work

In this paper, we proposed a novel approach for the automatic semantic annotation of entity columns in tabular datasets with classes from a knowledge graph. We experimented with different datasets and used the English DBpedia as our source knowledge graph. We showed that without relying on external context or table headers, we were able to semantically annotate most of the subject columns and outperform the state-of-the-art using a simpler approach with naive entity linking. We also presented how less famous entities can be mistaken for the famous ones. Nonetheless, an open question remains about how to find the optimal value for our parameter α.

For the future work, we plan to use advanced entity linking and entity disambiguation (instead of naive exact match), which may also improve accuracy. We can also experiment including entity attributes to improve accuracy. Another machine learning approach may also be employed here by learning from the user's modifications of the annotations or domain-dependent features.

References

[1] M.J. Cafarella, A. Halevy, D.Z. Wang, E. Wu and Y. Zhang, Webtables: exploring the power of tables on the web, *Proceedings of the VLDB Endowment* **1**(1) (2008), 538–549.

[2] D. Ritze, O. Lehmberg, Y. Oulabi and C. Bizer, Profiling the potential of web tables for augmenting cross-domain knowledge bases, in: *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2016, pp. 251–261.

[3] M. Taheriyan, C.A. Knoblock, P. Szekely and J.L. Ambite, Learning the semantics of structured data sources, *Web Semantics: Science, Services and Agents on the World Wide Web* **37** (2016), 152–169.

[4] S. Neumaier, J. Umbrich, J.X. Parreira and A. Polleres, Multi-level semantic labelling of numerical values, in: *International Semantic Web Conference*, Springer, 2016, pp. 428–445.

[5] A. Alobaid and O. Corcho, Fuzzy Semantic Labeling of Semi-structured Numerical Datasets, in: *European Knowledge Acquisition Workshop*, Springer, 2018, pp. 19–33.

[6] E. Kacprzak, J.M. Giménez-García, A. Piscopo, L. Koesten, L.-D. Ibáñez, J. Tennison and E. Simperl, Making sense of numerical data-semantic labelling of web tables, in: *European Knowledge Acquisition Workshop*, Springer, 2018, pp. 163–178.

[7] O. Hassanzadeh, M.J. Ward, M. Rodriguez-Muro and K. Srinivas, Understanding a large corpus of web tables through matching with knowledge bases: an empirical study., in: *OM*, 2015, pp. 25–34.

[8] W3C, RDF Semantics, 2008.

[9] DBpedia, DBpedia about.

[10] M. Zhang and K. Chakrabarti, Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables, in: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ACM, 2013, pp. 145–156.

[11] G. Limaye, S. Sarawagi and S. Chakrabarti, Annotating and searching web tables using entities, types and relationships, *Proceedings of the VLDB Endowment* **3**(1–2) (2010), 1338–1347.

[12] Z. Syed, T. Finin, V. Mulwad and A. Joshi, Exploiting a web of semantic data for interpreting tables, in: *Proceedings of the Second Web Science Conference*, Vol. 5, 2010.

[13] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao and C. Wu, Recovering semantics of tables on the web, *Proceedings of the VLDB Endowment* **4**(9) (2011), 528–538.

[14] A. Goel, C.A. Knoblock and K. Lerman, Exploiting structure within data for accurate labeling using conditional random fields, in: *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012, p. 1.

[15] D. Ritze, O. Lehmberg and C. Bizer, Matching html tables to dbpedia, in: *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, ACM, 2015, p. 10.

[16] S. Ramnandan, A. Mittal, C.A. Knoblock and P. Szekely, Assigning semantic labels to data sources, in: *European Semantic Web Conference*, Springer, 2015, pp. 403–417.

[17] I. Ermilov and A.-C.N. Ngomo, T AIPAN: automatic property mapping for tabular data, in: *European Knowledge Acquisition Workshop*, Springer, 2016, pp. 163–179.

[18] R. Usbeck, A.-C.N. Ngomo, M. Röder, D. Gerber, S.A. Coelho, S. Auer and A. Both, AGDISTIS-graph-based disambiguation of named entities using linked data, in: *International semantic web conference*, Springer, 2014, pp. 457–471.

[19] M. Pham, S. Alse, C.A. Knoblock and P. Szekely, Semantic labeling: a domain-independent approach, in: *International Semantic Web Conference*, Springer, 2016, pp. 446–462.

[20] G. Quercini and C. Reynaud, Entity discovery and annotation in tables, in: *Proceedings of the 16th International Conference on Extending Database Technology*, ACM, 2013, pp. 693–704.

[21] Z. Zhang, Effective and efficient semantic table interpretation using tableminer+, *Semantic Web* **8**(6) (2017), 921–957.
[22] Z.a. Zhang, Towards efficient and effective semantic table interpretation, in: *International Semantic Web Conference*, Springer, 2014, pp. 487–502.
[23] A. Tonon, M. Catasta, G. Demartini, P. Cudré-Mauroux and K. Aberer, Trank: Ranking entity types using the web of data, in: *International Semantic Web Conference*, Springer, 2013, pp. 640–656.
[24] G. Demartini, D.E. Difallah and P. Cudré-Mauroux, Zen-Crowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking, in: *Proceedings of the 21st international conference on World Wide Web*, ACM, 2012, pp. 469–478.
[25] A. Tonon, G. Demartini and P. Cudré-Mauroux, Combining inverted indices and structured search for ad-hoc object retrieval, in: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2012, pp. 125–134.
[26] F.M. Suchanek, S. Abiteboul and P. Senellart, Paris: Probabilistic alignment of relations, instances, and schema, *Proceedings of the VLDB Endowment* **5**(3) (2011), 157–168.
[27] S. Campinas, D. Ceccarelli, T.E. Perry, R. Delbru, K. Balog and G. Tummarello, The Sindice-2011 dataset for entity-oriented search in the web of data, in: *1st international workshop on entity-oriented search (EOS)*, 2011, pp. 26–32.
[28] A.G. Nuzzolese, A. Gangemi, V. Presutti, F. Draicchio, A. Musetti and P. Ciancarini, Tìpalo: A tool for automatic typing of dbpedia entities, in: *Extended Semantic Web Conference*, Springer, 2013, pp. 253–257.
[29] V. Presutti, F. Draicchio and A. Gangemi, Knowledge extraction based on discourse representation theory and linguistic frames, in: *International Conference on Knowledge Engineering and Knowledge Management*, Springer, 2012, pp. 114–129.
[30] A. Kalyanpur, J.W. Murdock, J. Fan and C. Welty, Leveraging community-built knowledge for type coercion in question answering, in: *International Semantic Web Conference*, Springer, 2011, pp. 144–156.
[31] D. Deng, Y. Jiang, G. Li, J. Li and C. Yu, Scalable column concept determination for web tables using large knowledge bases, *Proceedings of the VLDB Endowment* **6**(13) (2013), 1606–1617.
[32] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, *Communications of the ACM* **13**(7) (1970), 422–426.
[33] K. Shanmugasundaram, H. Brönnimann and N. Memon, Payload attribution via hierarchical bloom filters, in: *Proceedings of the 11th ACM conference on Computer and communications security*, ACM, 2004, pp. 31–41.
[34] S. Duan, A. Fokoue, O. Hassanzadeh, A. Kementsietsidis, K. Srinivas and M.J. Ward, Instance-based matching of large ontologies using locality-sensitive hashing, in: *International Semantic Web Conference*, Springer, 2012, pp. 49–64.
[35] D. Ritze and C. Bizer, Matching web tables to DBpedia-a feature utility study, *context* **42**(41) (2017), 19.
[36] C.E. Shannon, A mathematical theory of communication, *Bell system technical journal* **27**(3) (1948), 379–423.
[37] A. Alobaid and O. Corcho, TADA-Entity, 2018. doi:10.5281/zenodo.1462136. https://doi.org/10.5281/zenodo.1462136.
[38] A. Alobaid and O. Corcho, Olympic Games 2020, 2018. doi:10.5281/zenodo.1408563. https://doi.org/10.5281/zenodo.1408563.
[39] D. Ritze, O. Lehmberg and C. Bizer, T2D Gold Standard for Matching Web Tables to DBpedia, [Online; accessed 26-September-2018].
[40] D. Ritze, O. Lehmberg and C. Bizer, T2Dv2 Gold Standard for Matching Web Tables to DBpedia, 2015, [Online; accessed 16-August-2018].
[41] G. Wu, Y. He and X. Hu, Entity Linking: An Issue to Extract Corresponding Entity With Knowledge Base, *IEEE Access* **6** (2018), 6220–6231.
[42] W. Shen, J. Wang and J. Han, Entity linking with a knowledge base: Issues, techniques, and solutions, *IEEE Transactions on Knowledge and Data Engineering* **27**(2) (2015), 443–460.

Appendix A. Scoring Functions

In this Appendix we prove that the coverage scoring function of a descendant of a *type t* will never obtain a coverage score higher than any of its ancestors and that the specificity scoring function of a descendant of a *type t* will never obtain a specificity score lower than any of its ancestors.

A.1. Coverage

Lemma 1. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2. The coverage score of t_1 is greater than or equal to the coverage score of t_2*

$$f_c(t_1) \geqslant f_c(t_2)$$

Proof. To prove it by contradiction, as assume that $f_c(t_2) > f_c(t_1)$

$$L_c(t_2)/m > L_c(t_1)/m$$

$$L_c(t_2) > L_c(t_1)$$

$$L_c(t_2) > I_c(t_1) + \sum_u L_c(u)$$

since t_2 is a descendant of t_1, then t_2 is in $\sum_u L_c(u)$

$$L_c(t_2) > I_c(t_1) + L_c(t_2) + ...$$

since all the terms are positive, then this proposition is false.

Hence, lemma is proved by contradiction □

A.2. Specificity

Lemma 2. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2. The specificity score f_{s1} of t_2 is greater than the specificity f_{s1} of t_1 i.e. $f_{s1}(t_2) > f_{s1}(t_1)$.*

Proof. To prove that by contradiction,
we assume that: $f_{s1}(t_2) < f_{s1}(t_1)$

$$\sqrt{1 - L_s(t_2)^2} < \sqrt{1 - L_s(t_1)^2}$$

$$1 - L_s(t_2)^2 < 1 - L_s(t_1)^2$$

$$-L_s(t_2)^2 < -L_s(t_1)^2$$

$$L_s(t_2)^2 > L_s(t_1)^2$$

$$L_s(t_2) > L_s(t_1)$$

$$I_s(t_2) * ... * L_s(t_1) > L_s(t_1)$$

Since all the terms (the I_s's and the L_s's) are smaller than 1, this is impossible. Hence, it is proven by contradiction □

Lemma 3. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2. The specificity score f_{s2} of t_2 is greater than the specificity f_{s2} of t_1 i.e. $f_{s2}(t_2) > f_{s2}(t_1)$.*

Proof. Let us assume that: $f_{s2}(t_2) < f_{s2}(t_1)$

$$-L_s(t_2)^2 + 1 < -L_s(t_1)^2 + 1$$

$$-L_s(t_2)^2 < -L_s(t_1)^2$$

$$L_s(t_2)^2 > L_s(t_1)^2$$

$$L_s(t_2) > L_s(t_1)$$

$$I_s(t_2) * ... * L_s(t_1) > L_s(t_1)$$

Since all the terms are less than 1, this is impossible, hence, this lemma is proved by contradiction. □

Lemma 4. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2. The specificity score f_{s3} of t_2 is greater than the specificity f_{s3} of t_1 i.e. $f_{s3}(t_2) > f_{s3}(t_1)$.*

Proof. To prove this by contradiction, we assume that: $f_{s3}(t_2) < f_{s3}(t_1)$

$$-L_s(t_2) + 1 < -L_s(t_1) + 1$$

$$-L_s(t_2) < -L_s(t_1)$$

$$L_s(t_2) > L_s(t_1)$$

$$I_s(t_2) * ... * L_s(t_1) > L_s(t_1)$$

Since all the terms is less than one, this cannot hold; hence it is proven by contradiction □

Lemma 5. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2. The specificity score f_{s4} of t_2 is greater than the specificity f_{s4} of t_1 i.e. $f_{s4}(t_2) > f_{s4}(t_1)$.*

Proof. To prove this lemma, let us assume that: $f_{s4}(t_2) < f_{s4}(t_1)$

$$1 - \sqrt{L_s(t_2)} < 1 - \sqrt{L_s(t_1)}$$

$$-\sqrt{L_s(t_2)} < -\sqrt{L_s(t_1)}$$

$$\sqrt{L_s(t_2)} > \sqrt{L_s(t_1)}$$

$$L_s(t_2) > L_s(t_1)$$

$$I_s(t_2) * ... * L_s(t_1) > L_s(t_1)$$

□

Lemma 6. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2. The specificity score f_{s5} of t_2 is greater than the specificity f_{s5} of t_1 i.e. $f_{s5}(t_2) > f_{s5}(t_1)$.*

Proof. Let us assume that: $f_{s5}(t_2) < f_{s5}(t_1)$

$$(1-\sqrt{L_s(t_2)})^2 < (1-\sqrt{L_s(t_1)})^2$$

$$1-\sqrt{L_s(t_2)} < 1-\sqrt{L_s(t_1)}$$

$$-\sqrt{L_s(t_2)} < -\sqrt{L_s(t_1)}$$

$$\sqrt{L_s(t_2)} > \sqrt{L_s(t_1)}$$

$$\sqrt{L_s(t_2)} > \sqrt{L_s(t_1)}$$

$$L_s(t_2) > L_s(t_1)$$

$$I_s(t_2) * ... * L_s(t_1) > L_s(t_1)$$

Since all the terms are less than 1, this is impossible, and hence proved by contradiction □

A.3. Optimal α

In this section, we explore the possibility of an optimal α for a class hierarchy with a single correct *type*. We explore three cases: 1) t_1 is an ancestor of t_2 and t_2 is the correct type ;2) t_1 is an ancestor of t_2 and t_1 is the correct type; 3) t_1 and t_2 are not on the same path (none of them is an ancestor of the other).

Lemma 7. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2 and t_2 is the correct type. There exists a value α such that $f(t_2) > f(t_1)$ (referred to as a correct α).*

Proof. Given that t_1 is an ancestor of t_2 then

$$1 \geqslant f_c(t_1) > f_c(t_2) > 0 \implies f_c(t_1) - f_c(t_2) = A : A \in (0,1)$$

$$1 > f_s(t_2) > f_s(t_1) > 0 \implies f_s(t_2) - f_s(t_1) = B : B \in (0,1)$$

$$\text{If } \exists\, \alpha : f(t_2) - f(t_1) > 0$$

$$\big[\alpha * f_c(t_2) + (1-\alpha) * f_s(t_2)\big] - \big[\alpha * f_c(t_1) + (1-\alpha) * f_s(t_1)\big] > 0$$

$$\alpha * f_c(t_2) + (1-\alpha) * f_s(t_2) - \alpha * f_c(t_1) - (1-\alpha) * f_s(t_1) > 0$$

$$\Big[\alpha * f_c(t_2) - \alpha * f_c(t_1)\Big] + \Big[(1-\alpha) * f_s(t_2) - (1-\alpha) * f_s(t_1)\Big] > 0$$

$$-\alpha * \Big[-f_c(t_2) + f_c(t_1)\Big] + (1-\alpha) * \Big[* f_s(t_2) - f_s(t_1)\Big] > 0$$

$$-\alpha * A + (1-\alpha) * B > 0$$

$$(1-\alpha) * B > \alpha * A$$

$$(1-\alpha) > \alpha * \frac{A}{B}$$

$$\frac{(1-\alpha)}{\alpha} > \frac{A}{B}$$

$$\frac{1}{\alpha} - \frac{\alpha}{\alpha} > \frac{A}{B}$$

$$\frac{1}{\alpha} - 1 > \frac{A}{B}$$

$$\frac{1}{\alpha} > \frac{A}{B} + 1$$

$$\frac{1}{\alpha} > \frac{A}{B} + \frac{B}{B}$$

$$\frac{1}{\alpha} > \frac{A+B}{B}$$

$$\alpha < \frac{B}{A+B}$$

Since $A \in (1,0)$ and $B \in (1,0)$ hence, exist at least one value α that satisfies that.

□

Lemma 8. *Given two types t_1 and t_2, where t_1 is an ancestor of t_2 and t_1 is the correct type . There exists a value α such that $f(t_1) > f(t_2)$ (referred to as a correct α).*

Proof.

$$f_c(t_1) > f_c(t_2)$$

$$f_c(t_1) - f_c(t_2) = A$$

$$f_s(t_2) > f_s(t_1)$$

$$f_s(t_2) - f_s(t_1) = B$$

$$\text{If } \exists\alpha : f(t_1) > f(t_2)$$

$$f(t_1) - f(t_2) > 0$$

$$\alpha * f_c(t_1) + (1-\alpha) * f_s(t_1) - \alpha * f_c(t_2) - (1-\alpha) * f_s(t_2) > 0$$

$$\Big[\alpha * f_c(t_1) - \alpha * f_c(t_2)\Big] + \Big[(1-\alpha) * f_s(t_1) - (1-\alpha) * f_s(t_2)\Big] > 0$$

$$\alpha * \Big[f_c(t_1) - f_c(t_2)\Big] - (1-\alpha) * \Big[f_s(t_2) - f_s(t_1)\Big] > 0$$

$$\alpha * A - (1-\alpha) * B > 0$$

$$\frac{\alpha}{(1-\alpha)} > \frac{B}{A}$$

$$\frac{(1-\alpha)}{\alpha} < \frac{A}{B}$$

$$\frac{1}{\alpha} - 1 < \frac{A}{B}$$

$$\frac{1}{\alpha} < \frac{A+B}{B}$$

$$\alpha > \frac{B}{A+B}$$

since $A \in (0,1)$ and $B \in (0,1)$, there exists an α that satisfies that, hence this lemma is proved □

Lemma 9. *Given two types t_1 and t_2, where non of them is an ancestor of the other and t_1 is the correct type . There exists a value α such that $f(t_1) > f(t_2)$.*

Proof. <u>Case 1</u>: $f_c(t_1) > f_c(t_2)$ and $f_s(t_1) > f_s(f_2)$ to prove this, we need this to holds: $f(t_1) > f(t_2)$

$$f(t_1) - f(t_2) > 0$$

$$\alpha f_c(t_1) + (1-\alpha) f_s(t_1) - \alpha f_c(t_2) - (1-\alpha) f_s(t_2) > 0$$

$$\alpha\Big[f_c(t_1) - f_c(t_2)\Big] + (1-\alpha)\Big[f_s(t_1) - f_s(t_2)\Big] > 0$$

$$\alpha\Big[f_c(t_1) - f_c(t_2)\Big] + (1-\alpha)\Big[f_s(t_1) - f_s(t_2)\Big] > 0$$

Which holds because $f_c(t_1) - f_c(t_2) > 0$ and $f_s(t_1) - f_s(t_2) > 0$

<u>Case 2</u>: $f_c(t_1) > f_c(t_2)$ and $f_s(t_1) \leqslant f_s(f_2)$

$$f(t_1) - f(t_2) > 0$$

$$\alpha f_c(t_1) + (1-\alpha) f_s(t_1) - \alpha f_c(t_2) - (1-\alpha) f_s(t_2) > 0$$

$$\alpha\Big[f_c(t_1) - f_c(t_2)\Big] + (1-\alpha)\Big[f_s(t_1) - f_s(t_2)\Big] > 0$$

$$\alpha\Big[f_c(t_1) - f_c(t_2)\Big] + (1-\alpha)\Big[f_s(t_1) - f_s(t_2)\Big] > 0$$

Since $f_c(t_1) - f_c(t_2) > 0$ and $f_s(t_1) - f_s(t_2) \leqslant 0$, this will hold for any value $\alpha > 0.5$

<u>Case 3</u>: $f_c(t_1) \leqslant f_c(t_2)$ and $f_s(t_1) > f_s(f_2)$

$$f(t_1) - f(t_2) > 0$$

$$\alpha f_c(t_1) + (1-\alpha) f_s(t_1) - \alpha f_c(t_2) - (1-\alpha) f_s(t_2) > 0$$

$$\alpha\Big[f_c(t_1) - f_c(t_2)\Big] + (1-\alpha)\Big[f_s(t_1) - f_s(t_2)\Big] > 0$$

$$\alpha\Big[f_c(t_1) - f_c(t_2)\Big] + (1-\alpha)\Big[f_s(t_1) - f_s(t_2)\Big] > 0$$

Since $f_c(t_1) - f_c(t_2) \leqslant 0$ and $f_s(t_1) - f_s(t_2) > 0$, this will hold for any value $\alpha < 0.5$

Case 4: $f_c(t_1) \leqslant f_c(t_2)$ and $f_s(t_1) \leqslant f_s(f_2)$

Similar to case 1, this is impossible to hold for any α such that $1 \geqslant alpha \geqslant 0$. This implies that t_1 is probably not the correct type as there are more entities classified as t_2 that are even more specific than t_1. More specifically, it means that most cell annotations are pointing towards t_2; which is not due to a typical mismatch, as typical mismatches do not converge to high specificity type.

□

Appendix B. Constructing the Scoring Functions

In this section, we explain how we came up with each of the scoring functions. The ultimate goal is to have a type for the entity column. Our intuition is to choose the type that is correct for all the cells in the target column that we want to type. Each entity is annotated with one or more types. Given these types, we can construct the type hierarchy from the knowledge graph. We thought of choosing the most common type to cover all the cells. For example, given football players and basketball players, the most common thing is that they all athletes. But in practice, it is challenging to link to the correct entities and to type them correctly from the cells [41, 42]. If we are using DBpedia for example and we got one incorrect type for a cell, we can easily end up in the root of the type hierarchy ("owl:Thing" for DBpedia) as the common type.

We notice further that we need the type also to be as specific as possible. For a column of scientists names, it is more valuable for us to annotate them with the type "scientist" than "person" or "thing." It is also the case that the probability of having a column to have a more specific type (e.g., "scientist") is lower than having the type of the column to be of a more general type (e.g., "person"). Following the information theory, the value of a piece of information increases as the probability decreases.

Following the above intuitions, we aim to have a type that covers most of the cells in the column and also be as specific as possible. These two goals pull in different directions: to increase the *coverage* (to cover as much cells as possible) pulls the type upwards (towards the root) and the *specificity* (to be as specific as possible in the types) pulls the type downwards (here we are picturing the type hierarchy to have the root on the top and the leaves on the bottom of the tree).

Our idea is to find a balance between the two to maximize the score. So, we have this simple function to balance the coverage with α and the specificity with $1-\alpha$. We formalized it here as follow:

$$\max_t f(t) = \alpha * f_c(t) + (1-\alpha) * f_s(t)$$

Where f_c is the coverage function, and f_s is the specificity function. We explain the details for the coverage and specificity functions in the following sections.

B.1. Coverage

We are trying to construct a function that when maximized, picks the type that covers most of the cells. The first thing that we might think of is to use the type that is most common (often referred to as "majority"). But it only works if the types of each cell are almost the same, for example, if the majority of the cells has the type "footballPlayer". It won't work in the case of mixed types (e.g., "footballPlayer" and "basketballPlayer"), which should result in the type "athlete" instead.

Another way we thought of is to have all the types in the path in the type hierarchy from the type of the cell (e.g., "footballPlayer") to the root ("Thing" in the case of DBpedia). This way we have more general types (e.g., "athlete", "person"). We can have something like the majority but for each type in the hierarchy of each typed cell. In other words, the majority for each type in the path to the root.

B.1.1. Uncertainty

Another intuition we can think of is related to uncertainty. A cell can have multiple types due to common names. An example of this is "Scott Arnold," there are multiple players with the same name. In such cases, we assign lower confidence to the types of such cells. We formulate it in a way such that the total value decreases as the number of types increases. Actually, each cell does not just get typed based on its value; we need to get the entities that have the name in the cell. For a given cell, we fetch the entities and then get the types for each of these entities. Each cell will have the types for each entity linked to that cell. We formulate the score as:

$$\frac{1}{||W(v)||}$$

The number of aggregated types for the cell value v is denoted as $||W(v)||$.

B.1.2. Proportional Influence

But, this will not differentiate the influence of a type of an entity if the number of types for that entities is high or low. For example, if we have the polymath "Bertrand Russell," he will be annotated with multiple types: logician, mathematician, historian, writer, and Nobel prize Laureate. Having multiple types, we have less confidence in the intended one for the context. If the input data is about Nobel prize Laureates, then this is the anticipated type. If the other people in the input data are mathematicians, then probably the type "mathematician" is the one that we are looking for. Having multiple types reduces confidence, and we reflect this on the formulation. We have this for each entity proposed for each cell so that entities with fewer types have higher confidence than the ones with more types. To formulate this, we first divide the score of a single cell for each entity. The intuition is that cells with more candidate entities have lower confidence. We will have $\frac{1}{||Z(v)||}$, and then for each entity e, we will have $\frac{1}{||Q(e)||}$ where $||Z(v)||$ is the number of entities for the cell value v and $||Q(e)||$ is the number of types for the entity e. Combining these two, we present the equation ($E(v)$ are the entities for a given cell value v):

$$\sum_{e} \frac{1}{\|Z(v)\|\|Q(e)\|} = 1 \qquad \forall e \in E(v)$$

We illustrate this in Figure 3. Since we want to choose the type to maximize the coverage score, we aggregate the coverage score for each type t as follows:

$$I_c(t) = \sum_{v} \sum_{e} \frac{1}{\|Z(v)\|\|Q(e)\|} \quad \forall v, e : t \in Q(e), e \in Z(e)$$

Note that the t in the equation is for the cells with a value v that has an entity e that has a type t. $I_c(t)$ is the coverage for a single type t.

B.1.3. Inclusion

In the previous equation, we did not take into account that a parent type (in the type hierarchy) actually covers all the cells its children cover. We include this in the below recursive equation:

$$L_c(t) = I_c(t) + \sum_{u} L_c(u)$$

So, the $L_c(t)$ coverage of a type t is the $I_c(t)$ of t plus the coverage L_c of its children.

The $L_c(t)$ coverage increases as the number of entities increase. To overcome this, we normalize $L_c(t)$ by dividing it by the number of cells m . This would make the coverage insensitive to the number of cells in the column. The final coverage score would be:

$$f_c(t) = \frac{L_c(t)}{m}$$

B.2. specificity

Besides, choosing a type that covers as much from the cells as possible, we also want to be as specific as possible. More specific types are more valuable, as the probability decreases the value of the corresponding piece of information increases. This also follows our intuition that we are generally more interested in knowing that a given entity is a basketball player than it being an athlete or a person.

The first intuition that came to our mind is the level of the type in the type hierarchy. The deeper the type node is, the more specific it is. Even though the depth gives us an idea of the specificity of the type, it treats all levels the same way. Knowledge graphs may have more levels (subclass relation) in some domains (in the same knowledge graph) than others, which not necessarily reflect the specificity. As an alternative, we thought of using the number of instances. To know how specific a type t is, we divide the number of instances of a type t ($||R(t)||$) by the number of instances of its parent ($||R(p)||$):

$$I_s(t) = \frac{\|R(t)\|}{\|R(p)\|}$$

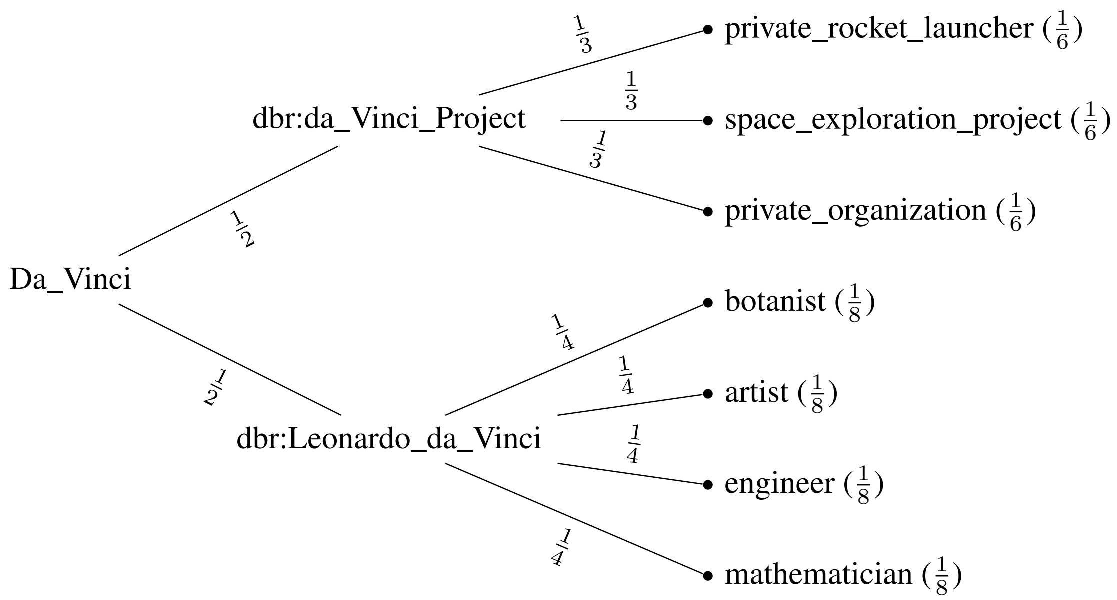


Fig. 3. The coverage score break down for a single cell

We refer to I_s as the instance specificity[15].
This gives us satisfactory results, but it only takes into account the type and its parent. Since the number of entities of a type is less than or equal to the number of entities of its parent, the results will be bounded by 0 and 1. To include the specificity of its parent, we multiply the instance specificity of the type t ($I_s(t)$) by the local specificity of its parent p ($L_s(p)$). The local specificity is computed as:

$$L_s(t) = I_s(t) * L_s(p)$$

Following the local specificity equation, the more specific the type t is, the lower its value becomes. We are looking for a formula f_s that increases as the local specificity decreases. The first thing that came to our mind is an inverted version of the square function, which is a curve. We can also experiment with a straight line as well. Another aspect is that we need is for the function to be bound by 0 and 1. We pick five functions that satisfies these conditions: $\sqrt{(1 - L_s(t)^2)}$, $-L_s(t)^2 + 1$, $1 - \sqrt{L_s(t)}$, $(1 - \sqrt{L_s(t)})^2$, and $-L_s(t) + 1$ (Fig 4).

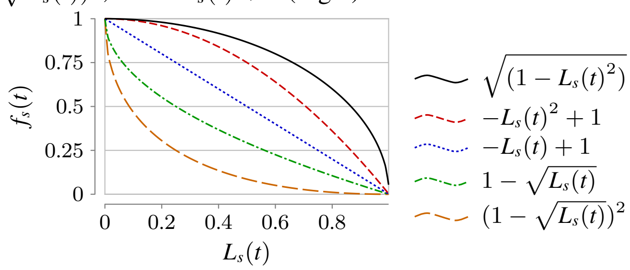


Fig. 4. Different candidate specificity functions

[15]note that *instance specificity* does not refer to the specificity of an entity. It refers to the specificity of a single type t in relation with its parent p