Tab2Know: Building a Knowledge Base from Tables in Scientific Papers

Benno Kruit[1,2(✉)], Hongyu He[1], and Jacopo Urbani[1]

[1] Department of Computer Science,
Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
b.b.kruit@cwi.nl, hongyu.he@vu.nl, jacopo@cs.vu.nl
[2] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Abstract. Tables in scientific papers contain a wealth of valuable knowledge for the scientific enterprise. To help the many of us who frequently consult this type of knowledge, we present Tab2Know, a new end-to-end system to build a Knowledge Base (KB) from tables in scientific papers. Tab2Know addresses the challenge of automatically interpreting the tables in papers and of disambiguating the entities that they contain. To solve these problems, we propose a pipeline that employs both statistical-based classifiers and logic-based reasoning. First, our pipeline applies weakly supervised classifiers to recognize the type of tables and columns, with the help of a data labeling system and an ontology specifically designed for our purpose. Then, logic-based reasoning is used to link equivalent entities (via *sameAs* links) in different tables. An empirical evaluation of our approach using a corpus of papers in the Computer Science domain has returned satisfactory performance. This suggests that ours is a promising step to create a large-scale KB of scientific knowledge.

1 Introduction

Often, scientific advancement requires an extensive analysis of pre-existing techniques or a careful comparison with previous experimental results. For instance, it is common for researchers in Artificial Intelligence (AI) to ask questions like "Which are the most popular datasets used for graph embeddings?" or "What is the F1 of BERT on TACRED?". Finding the answers obliges the researchers to spend much time in perusing existing literature, looking for experimental results, techniques, or other valuable resources.

The answers to such questions can be frequently found in tabular form, especially the ones that describe the output of experiments. Unfortunately, tables in papers are made for human consumption; thus, their layout can be irregular or contain specific abbreviations that are hard to disambiguate automatically. It would be very useful if their content were copied into a clean Knowledge Base (KB) where tables are disambiguated and connected using a single standardized vocabulary. This KB could assist the users in finding those answers without accessing the papers or could be used for many other purposes, like categorizing papers, finding inconsistencies or plagiarized content.

J. Z. Pan et al. (Eds.): ISWC 2020, LNCS 12506, pp. 349–365, 2020.
https://doi.org/10.1007/978-3-030-62419-4_20

To build such a KB, we present Tab2Know, an end-to-end system designed to interpret the tables in scientific papers. The main challenge tackled by Tab2Know lies in the interpretation of the table, which is a necessary step to build a KB. In this context, the peculiarities of tables in scientific literature make our domain quite different from previous work (e.g., [3,23,32]), which mainly focused on Web tables. First, the interpretation of Web tables benefits from the existence of large, curated KBs (e.g., DBPedia [5]), which allows the linking of many entities. In our case, there is no such KB. Second, a large number of Web tables can be categorized as *entity-attribute* tables, i.e., tables where each row describes one entity, and the columns represent attributes [23,32,39]. In our context, we observed that many tables are of different types, namely they express n-ary relations, such as the results of experiments. For such tables, existing techniques designed for entity-attribute tables cannot be reused.

With Tab2Know, we propose a pipeline for knowledge extraction that includes both weakly supervised learning methods and logical reasoning. Tab2Know is designed to 1) detect the type of the table; 2) disambiguate the types of columns, and 3) link the entities between tables. The first operation is applied to distinguish, for instance, tables that report experiments from tables that report examples. The second operation recognizes the rows that contain the headers of the table and disambiguates the columns, linking them to classes of an ontology. The third operation links entities in different tables.

We implement the first two operations using statistical-based classifiers trained with bag-of-words and context-based features. These classifiers have an accuracy that largely depends on the quality and amount of training data. Unfortunately, labeling training data is increasingly the largest bottleneck as it often requires an expensive manual effort and/or expertise that might not be readily available. To counter this problem, we propose a weakly supervised method that relies on SPARQL queries and Snorkel [30]. The SPARQL queries are used to automatically retrieve samples of a given class, type, etc., while Snorkel resolves potential conflicts in the prediction with a sophisticated voting mechanism.

After the first two operations are completed, we transform the tables into an RDF KB and apply reasoning with existentially quantified rules to identify and link entities in different tables. Reasoning with existentially quantified rules is a well-known technology for data integration and wrangling [22]. For our problem, we designed a set of rules that considers the types of columns and string similarities to establish links using the *sameAs* relation. Then, we used VLog [8] to materialize the derivations and link the entities across the tables.

We evaluated our approach considering open access CS papers. In particular, we evaluated the performance of our pipeline using gold standards and compared it to another state-of-the-art method. We also applied our method to a larger corpus with 73k scientific tables. In these tables, we found 312k entities, which are linked to the table structure and metadata in our large-scale KB.

We release the datasets, gold standards, and resulting KB as an open resource for the research community at https://doi.org/10.5281/zenodo.3983012.

The code, ruleset, and instructions to replicate our experiments are also publicly available at https://github.com/karmaresearch/tab2know.

2 Related Work

Extracting knowledge from tables is a process that can be divided into *three* main tasks: *table extraction*, *structure detection*, and *table interpretation*. Once a set of tables is interpreted, another problem consists of recognizing whether multiple tables mention the same entities. We call this task *entity linking*, but this is also known as *entity resolution* [28], *record linkage* [10], or *entity matching* [6].

Table Extraction. This task consists of recognizing the parts of a PDF/image which contain a table. Existing methods can be categorized either as heuristic (e.g., [11,27]) or supervised (e.g., [29]). In this paper, we use the system *PDFFigures* [11], which is a recent approach based on heuristics with very high precision and recall ($\geq 90\%$) that is used in Semantic Scholar [1].

Structure Detection. Given as input an image-like representation of a table, some systems focus on recognizing the table's structure so that it can be correctly extracted. A popular system is *Tabula* (https://tabula.technology/), which recognizes the table's structure using rules. More recently, some deep learning methods based on Convolutional Neural Networks (CNN) [34], Conditional Generative Adversarial Networks (CGAN) [37], and a combination of a CNN, saliency and graphical models [20] have been evaluated. The performance of these methods is good ($F_1 \geq 0.95$), but not much different from Tabula, which returns a F_1 between 0.86 and 0.96 and has the advantage that is unsupervised.

Table Interpretation. The goal of table interpretation consists of linking the content of the table to a KB so that new knowledge can be extracted from the table [24]. In this context, most of the previous work has focused on tables that represent *entity-attribute* relations [23]. These tables have rows that describe entities and columns that describe attributes. Thus, their interpretation consists of mapping each row to an entity in the KB, and linking each column to a relation in the KB. Some work has focused only on the first task (e.g., [3]) while others on the second (e.g., [9,14,25]). The work at [9], in particular, is similar to ours as it also uses SPARQL queries to create training data. The difference is that in [9], SPARQL is used to query a rich KB automatically, whereas in our case, we let users specify queries since we lack such a KB. In terms of methodology, current work in this field either relies on statistical models, like PGMs [3,24], or introduces an iterative process that filters out candidates [32,33,39].

As far as we know, the only systems that offer a end-to-end table interpretation are *T2K* [32], *TableMiner+* [39], and *TAKCO* [23], but these are designed for Web tables and rely on a rich KB like DBPedia [5], which we do not have.

The only work that has focused on the interpretation of tables from scientific literature is [38]. The authors describe an approach to automatically extract experimental data from tables based on ensemble learning. Although we view this work as the most relevant to our problem, there are several important differences

between our work and theirs. First, our approach employs a different set of technologies and performs entity linking, which is not considered in [38]. Then, our approach is more general. In fact, [38] focuses only on the extraction of tuples $(method, dataset, metric, score, source)$ while ours extracts a larger variety of knowledge. Finally, our approach yields a better accuracy (see Sect. 6).

Entity Linking. The problem of resolving entities in tables has received considerable attention in database research (96+ papers in VLDB, KDD, etc. in 2009–2014) [10,21,28]. One of the most popular systems is Magellan [21]. Magellan is a tool to help users to perform entity matching, providing different implementations of matching and blocking algorithms. Recently, Mudgal et al. [26] have studied the application of deep learning for entity matching, but concluded that it does not outperform existing methods on structured data. Other works have explored the usage of embeddings for this task: For instance, Cappuzzo et al. [7] have shown how we can construct embeddings from tabular data. Another line of work has been focusing on crowds, e.g., [12] and citations therein, while other works have focused on entity resolution using knowledge bases (e.g., LINDA [6]). Our work differs from the ones above because they either focus on highly structured table sets or require the existence of KBs (which we do not have). Moreover, another important difference is that we take a declarative approach with rules. Rules are useful because they can be easily debugged/extended directly by domain experts, and they can be integrated with ontological reasoning.

Other Related Works. We mention, as further related work, the systems by [13] and *TableNet* [17] which focus on *searching* for tables related to a given query. Other, less relevant works focus on extracting and searching for figures on papers [35,36]. These works complement our approach and can further assist the user to find relevant knowledge in papers.

3 Overview

Our goal is to construct a clean and large KB from the content of tables in scientific papers stored as PDFs. To do so, we need to address two main challenges: first, we must resolve the ambiguities that might arise during the noisy extraction process and reduce the error rate as much as possible. Second, we must counter the problem that we lack both: 1) a pre-existing KB that can guide the extraction process and 2) a large amount of training data. We must, in other words, find a way to build a KB from scratch.

Our proposal is a pipeline with three main tasks, as shown in Fig. 1:

- **Task 1: Table Extraction.** The system receives as input an image-like representation of a table, recognizes its structure, and returns its content as a CSV file. For this task, we use external tools. We provide more details below;
- **Task 2: Table Interpretation.** The system processes the CSV to recognize the headers and the type of the table. Then, it disambiguates the columns by mapping them to ontological classes. We describe this task in Sect. 4;

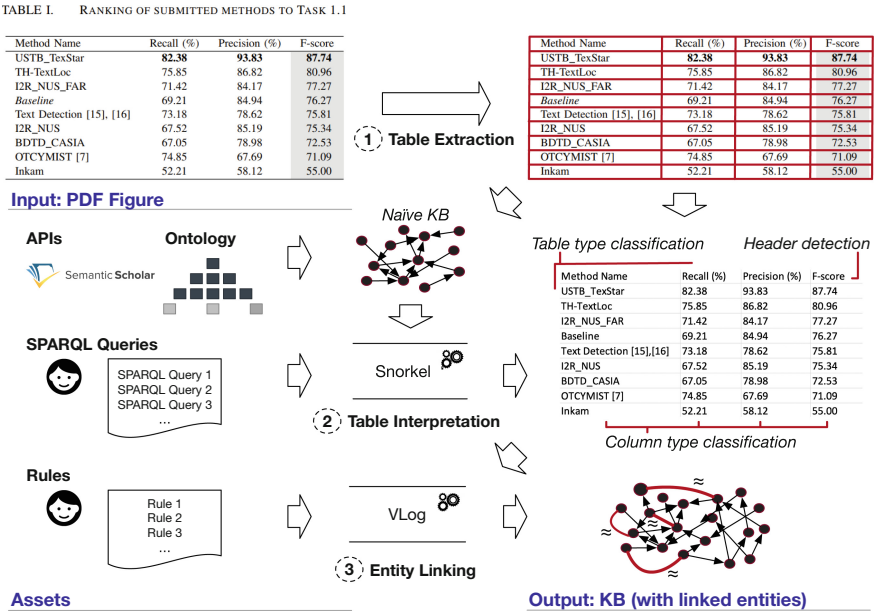


Fig. 1. Tab2Know: system overview

- **Task 3: Entity Linking.** Finally, the system performs logical-based reasoning to link the entities across tables. We describe this task in Sect. 5.

While in principle our method can be applied to scientific papers in any domain, we restrict our analysis to papers in Computer Science, which is our area of expertise. In particular, we consider Open Access papers and have been published in top-tier venues in subfields like AI, semantic web, databases, etc.

Before we describe the components, we describe *two* additional assets that we use for different purposes. The first one is an ontology constructed annotating a sample of random tables. A first version of this ontology contained 44 classes organized in a hierarchy with a maximum depth of 6. After further annotations, we decided to simplify it to a set of 27 classes (depth 3) for which we had substantial evidence in our corpus. The final ontology has 4 root classes: **Example**, **Input**, **Observation**, and **Other**. These classes define general table types. Then, the subclasses describe column types, e.g., **Dataset**, **Runtime**, or **Mean**. As an example, one of the longest chains is **Recall** $\sqsubseteq$ **Metric** $\sqsubseteq$ **Observation** with $\sqsubseteq$ denoting the subclass relation. The ontology is serialized in OWL using WebProtégé [19] and is publicly available as resource.

The second asset is an external KB that contains metadata of the papers, namely Semantic Scholar [1]. We access it using the provided APIs to retrieve the list of authors, the venue, and other contextual data.

Table Extraction. Our input consists of a collection of papers in PDF format. The first operation consists of launching PDFFigures [11] to extract from the

PDFs the coordinates of tables and related captions. We use the coordinates to extract an image-like representation of the tables, see for instance the table reported in Fig. 1. Then, we invoke Tabula, which is a tool also used in similar prior works [38], to recognize the structure of the tables using their coordinates and to translate them into CSV files.

After the images are converted, we perform a naïve conversion of the tables into RDF triples. We assign a URI to every table, column, row, and cell and link every cell, row, and column to the respective table with positional coordinates.

Example 1. Consider the table in Fig. 1. We report below some triples that are generated while dumping its content into RDF.

```
PREFIX : http://xzy/tab2know
:Table1 :hasRow :Table1-r1            :Table1 :hasCol :Table1-c1
:Table1-r1 rdf:type :Row              :Table1-c1 rdf:type :Column
:Table1-r1 :rowIndex 1^^⟨xsd:int⟩     :Table1-c1 :colIndex 1^^⟨xsd:int⟩
:Table1-r1c1 :cellOf :Table1          :Table1-r1c1 rdf:type :Cell
:Table1-r1c1 :rowIdx 1^^⟨xsd:int⟩     :Table1-r1c1 :colIdx 1^^⟨xsd:int⟩
:Table1-r1c1 rdf:value "Method name" :Table1-r2c1 rdf:value "USTB_TexStar"
                                    ...
```

As we can see from the triples in Example 1, the KB generated at this stage is a direct conversion of the tabular structure into triples. Despite its simplicity, however, such a KB is already useful because it can be used to query the n-ary relations expressed in the tables in combination with the papers' metadata. For instance, we can write a SPARQL query to retrieve all the tables created by one author with a caption containing the word "results", or to retrieve the tables containing "F1" and which appear as proceedings of a certain venue.

The main problem at this stage is that we can only query using string similarities, which severely reduces the recall. For instance, a query could miss a column titled `Prec.` if it searches for `Precision`. The next operation, described below, attempts to disambiguate the tables to create a KB that is more robust against the syntactic diversity of the surface form of their content.

4 Table Interpretation

Tab2Know performs three main operations to interpret the tables. First, it identifies the rows with the table's header (Sect. 4.2). Then, it detects the type of the table (Sect. 4.3). Finally, it maps each column to an ontological class (Sect. 4.4). First, we describe the procedure to obtain training data.

4.1 Training Data Generation

Statistical models are ideal for implementing a table interpretation that is robust against noise. However, their accuracy depends on high-quality training data, which we do not have (and it is expensive to obtain such data with human annotators). We counter this problem following the paradigm of *weak supervision*.

The idea is to employ many annotators, which are much cheaper than a human expert but also much noisier. These annotators can deliver a large volume of labeled data, but the labels might be incorrect or conflicting. To resolve these problems, we can either rely on procedures like majority voting or train a dedicated model to computed the most likely correct label. In the second case, we can use Snorkel, one of the most popular models for this purpose [30].

Snorkel's goal is to facilitate the learning of a model θ that, given a data point $x \in \mathcal{X}$, predicts its label $y \in \mathcal{Y}$. Instead of training θ by fitting it to a set of pre-labeled data points, as it would happen in a traditional supervised approach, Snorkel trains an additional generative model with unlabeled data and uses pre-labeled data only for validation and testing. For these two tasks, the amount of pre-labeled data can be much smaller, and thus cheaper to obtain. Then, the generative model can be used to train θ.

Snorkel introduces the term *labeling function* to indicate a data annotator with possibly low accuracy. A labeling function $\lambda : \mathcal{X} \rightarrow \mathcal{Y} \cup \{\emptyset\}$ can encode a heuristic or be a simple predictor. It receives a data point x in input and either returns a label in $\mathcal{Y}$ or *abstains*, i.e., returns $\emptyset$. Given m unlabeled data points and n labeling functions, Snorkel applies the labeling functions to the data points and computes a matrix $M \in (\mathcal{Y} \cup \{\emptyset\})^{m \times n}$.

Then, Snorkel processes M to compute, for each x_i where $i \in \{1, \ldots, m\}$, a *probabilistic training label* $\tilde{y}_i$. The processing consists of creating a generative model using a matrix completion-style algorithm over the covariance matrix of the labels [31]. Then, this model can be used to generate labeled data for training θ. In this work, we considered models such as Naïve Bayes (NB), Support Vector Machine (SVM), and Logistic Regression (LR) [4] to implement θ. We have also experimented with deeper learning models, but we did not obtain improvements because such models are more prone to overfitting if training data is scarce.

The effectiveness of Snorkel largely depends on the number and quality of the labeling functions. In our context, we implemented them using SPARQL queries, which are supposed to be entered by a (human) user. SPARQL queries are ideal because they can assign labels to many data points at once. For each query Q, we create a labeling function that receives in input a column/table x and returns an assigned class label (e.g., a table type, or the class of a column) if x is among the answers of Q. Otherwise, the function abstains.

Example 2. We show below an example of a SPARQL query that labels columns with the class $\mathbf{F_1}$ if they have a header cell with value "f1" and contain any cell with a numeric type.

```
select distinct ?column where {
      ?table :column ?column ; :cell ?cell .
      ?column :hasTitle "f1" . ?cell rdf:type xsd:decimal . }
```

Clearly, this query is not a good predictor if taken alone, but if we combine its output with the ones of many other functions, then the resulting predictive power is likely to be superior. This is the key observation used by Snorkel.

l_1-l_2	#S	#l_1-W	#l_2-W	#l_1-V	#l_2-V
en-de	1.9M	55M	52M	40k	50k
en-fr	2.0M	50M	51M	40k	50k
en-es	1.9M	49M	51M	40k	50k

(a) Input

Models	Rerank size	Beam size	GMV	Latency
miDNN	50	-	2.91%	9%
miRNN	50	5	5.03%	58%
miRNN+att.	50	5	5.82%	401%

(b) Observation

Type	Example Words
Offensive	disgusting, filthy, nasty, rude, horrible, terrible, awful, worst, idiotic, stupid, dumb, ugly, etc.
Non-offensive	help, love, respect, believe, congrats, hi, like, great, fun, nice, neat, happy, good, best, etc.

(c) Example

α_c	DP concentration parameter for each $c \in V$
$P_0(e\|c)$	CFG base distribution
$\boldsymbol{x}$	Set of non-terminal nodes in the treebank
$\mathcal{S}$	Set of sampling sites (one for each $x \in \boldsymbol{x}$)
S	A block of sampling sites, where $S \subseteq \mathcal{S}$
$\boldsymbol{b} = \{b_s\}_{s \in \mathcal{S}}$	Binary variables to be sampled ($b_s = 1 \rightarrow$ frontier node)
$\boldsymbol{z}$	Latent state of the segmented treebank
m	Number of sites $s \in S$ s.t. $b_S = 1$
$\boldsymbol{n} = \{n_{c,e}\}$	Sufficient statistics of $\boldsymbol{z}$
$\Delta n^{S:m}$	Change in counts by setting m sites in S

(d) Other

Fig. 2. Examples of tables of each category

In our pipeline, we execute all the user-provided SPARQL queries and then use their outputs to build the matrix M for a large number of data points. Next, we train the final discriminative model θ. We compute two different θ: One to generate training data for predicting the tables' types (Sect. 4.3) while the other is for predicting the columns' types (Sect. 4.4).

4.2 Table Header Detection

First, we identify the rows that define the headers. To this end, we can either always select the first row as header or employ more sophisticated methods to recognize multi-row headers, like [16]. We observed that a simplified unsupervised version of [16] yields a good accuracy on our dataset. We describe it below.

Our procedure exploits the observation that header rows differ significantly from the rest of the table with respect to character-based statistics. Hence, we categorize characters either as *numeric*, *uppercase*, *lowercase*, *space*, *non-alphanumeric*, or *other*. Then, for each column, we count how many characters of each class (e.g., numeric) appear in its cell. We compute the average count per class across the column and use these values to determine the standard deviation for each cell. The *outlier score* of a row r is determined as the average of the standard deviations of all classes of its cells. If the outlier score or r is greater than τ (default value is 1, set after cross-validation), then r is marked as header.

4.3 Table Type Detection

In scientific papers, tables are used for various reasons. We classified them in the classes **Observation**, **Input**, **Example**, and **Other** (See Fig. 2 for examples).

Knowing the class of a table is useful for reducing the search space when the user is interested in some specific content (e.g., The F_1 measure is typically not mentioned in tables of type **Example**). Moreover, we can also use this information as a feature for the column disambiguation.

We predict the table type with a statistical classifier. As features for the classifier, we selected bags-of-ngrams of lengths 1 to 3 that occurred more than once, weighted by their TF-IDF score. Tables often contain abbreviations and domain-specific symbols that address an audience of experts. These provide strong hints for determining the type of the table; thus we consider the ngram in the content of the cells and the table caption. We also included other numerical features. In particular, we use the fraction of numeric cells in the table and the minimum, maximum, median, mean and standard deviation of numerical columns. This resulted in a total of 5804 features.

To train the models, we first ask the users to specify some SPARQL queries which will be used by Snorkel to create a large volume of training data. Then, we experimented with three well-known types of classifiers: NB, SVM, and LR. Eventually, we selected LR because it returned the best performance on the noisiest dataset.

4.4 Column Type Detection

Finally, the interpretation procedure attempts at linking the columns to one of the available classes in our ontology. The ontology includes popular classes that we identified while annotating a sample (e.g., **Dataset**, **Runtime**,...), while infrequent classes with very few columns are mapped to the class **Other**. In general, we assume that a column is *untyped* if it is mapped to **Other**.

For this task, we also used bag-of-ngram features of lengths 1 to 3, extracted from the table caption, the column header cells, the header cells of the other columns, and the column body. We restricted the set of ngrams to only the top 1000 most frequent per extraction source. Additionally, we added features about the numerical columns, identical to those in Sect. 4.3. This resulted in a total of 3076 features.

Similarly as before, we first rely on user-provided SPARQL queries to generate training data. Then, we considered NB, SVM, and LR as classifiers. Once the models for the table and column types are trained, we use them to predict the types of every table and column in our corpus. Finally, we use the predicted class to annotate the table/column in the KB with a semantic type.

5 Entity Linking

Rationale. Predicting the types of tables and columns is useful to map the table schema into a meaningful n-ary relation. The last operation in our pipeline consists of associating cells to entities so that we can populate the n-ary relations with new instances.

We start by assuming that every non-numerical cell contains an entity mention, which implies the existence of one entity. This assumption is not unrealistic. Indeed, if we look at the table in Fig. 1, then we see that every non-numerical cell that is not in the table's header refers to an entity (e.g., the cell "USTB_TextStar" refers to an algorithm to detect text inside images).

In practice, it is likely that some entities are mentioned multiple times. This consideration motivates us to discover whether two entity mentions (possibly on different tables) refer to the same entity. When we do so, then we gain more knowledge about the entity and reduce the number of entities in the target KB. We call this task *entity linking* because we are linking, with the *sameAs* relation, equivalent entities across tables.

With this goal in mind, we start by assuming that every entity has the content of the corresponding cell as label. For instance, the entity mentioned in the cell with "USTB_TextStar" has "USTB_TextStar" as label. Using the labels to determine equality can be surprisingly effective in practice, but it is not an operation without risks. In fact, there are cases where different entities have the same label, or the same entity has multiple labels. These cases call for a more sophisticated procedure to discover equalities.

Reasoning. Reasoning with existentially quantified rules is an ideal tool to establish non-trivial equalities between entities since it was already previously used for data integration problems [15,18]. For our purposes, we are interested in applying two types of rules: *Tuple Generating Dependencies (TGDs)* and *Equality Generating Dependencies (EGDs)*. We describe those below.

Consider a vocabulary consisting of infinite and mutually disjoint sets of predicates $\mathcal{P}$, constants $\mathcal{C}$, null values $\mathcal{N}$, and variables $\mathcal{V}$. A *term* is either a constant, a variable, or a null value. An *atom* is an expression of the form $p(\vec{x})$ where $p \in \mathcal{P}$, $\vec{x}$ is a tuple of terms of length equal to the arity of p, which is fixed. A *fact* is an atom without variables. A TGD is a rule of the form:

$$\forall \vec{x}, \vec{y}.(B \rightarrow \exists \vec{z}.H) \tag{1}$$

where B is a conjunction of atoms over $\vec{x}$ and $\vec{y}$ while H is a conjunctions of atoms over $\vec{y}$ and $\vec{z}$. Let $x, y \in \vec{x}$. A EGD is a rule of the form:

$$\forall \vec{x}.(B \rightarrow x \approx y) \tag{2}$$

Intuitively, TGDs are used to infer new facts from an existing set of facts (i.e., the database). Their execution consists of finding in the database suitable replacements for the variables in $\vec{x}$ and $\vec{y}$ that render B a set of facts in the database. Then, these replacements and mappings from $\vec{z}$ to fresh values in $\mathcal{N}$ are used to map H into a set of facts, which is the set of *inferred* facts.

EGDs are used to establish the equivalence between terms. Their execution is similar to the one of TGDs, with the difference that whenever they infer that $a \approx b$, where a and b are terms and $a < b$ according to a predefined ordering, then every occurrence of b in the database is replaced with a.

The *chase* [15] is a class of forward-chaining procedures that exhaustively apply TGDs and EGDs to infer new knowledge with the rules. A formal definition of various chase procedures is available at [2]. In this work, we apply the restricted chase, one of the most popular variants. It is known that sometimes the chase may not terminate, but this is not our case since we use an *acyclic* ruleset [15].

We first map the content of the KB extracted from the tables into a set of facts. For example, the first two RDF triples in Example 1 map to the facts `hasRow(Table1,Table1-r1)` and `hasCol(Table1,Table1-c1)` respectively.

Then, we use the two TGDs

$$type(X, \texttt{Column}) \rightarrow \exists Y.colEntity(X, Y) \quad (r_1)$$
$$type(X, \texttt{Cell}) \rightarrow \exists Y.cellEntity(X, Y) \quad (r_2)$$

to introduce fresh entities for every column and cell in the tables. The predicates *colEntity* and *cellEntity* link entities (Y) to the columns and cells respectively. Note that we use null values to represent entities, thus we are simply stating their existence with some placeholders. To reason and discover whether two different entities are equivalent, we employ EGDs. In particular, we use five EGDs, reported below:

$$ceNoTypLabel(X, L), ceNoTypLabel(Y, L) \rightarrow X \approx Y \quad (r_3)$$
$$eNoTypLabel(X, C, L), eNoTypLabel(Y, C, L) \rightarrow X \approx Y \quad (r_4)$$
$$eTableLabel(X, T, L), eTableLabel(Y, T, L) \rightarrow X \approx Y \quad (r_5)$$
$$eTypLabel(X, S, L), eTypLabel(Y, S, M), STR\_EQ(L, M) \rightarrow X \approx Y \quad (r_6)$$
$$eAuthLabel(X, A, L), eAuthLabel(Y, A, M), STR\_EQ(L, M) \rightarrow X \approx Y \quad (r_7)$$

where *ceNoTypLabel*, *eNoTypLabel*, *eTableLabel*, *eTypLabel*, and *eAuthLabel* are auxiliary predicates that we introduce for improving the readability. We describe their intended meaning as follows. The fact $ceNoTypeLabel(X, L)$ is true if $colEntity(Y, X)$ is true and Y is an untyped column with header value L; $eNoTypeLabel(X, C, L)$ is true if X is an entity with a label L that appears in a cell inside an untyped column associated to entity C; $eTableLabel(X, T, L)$ is true if entity X with label L appears in table T; $eTypeLabel(X, S, L)$ is true if entity X with label L appears in a column with type S; $eAuthLabel(X, A, L)$ is true if entity X with label L appears in a table authored by author A.

The rationale behind each EGD is the following:

- **Rule $\mathbf{r_3}$** : This rule is introduced to disambiguate untyped columns. Since we were unable to discover the columns' types and assigned them to the class `Other`, we use the value of the header to determine whether they contain the same type of entities. Thus, the rule will infer that their associated entities are equal if they share the same header.
- **Rule $\mathbf{r_4}$** : This rule infers that two entities are equal if they appear in the same group of columns (created by r_3), and they share the same label.
- **Rule $\mathbf{r_5}$** : This rule encodes a simple heuristics, namely that if two entities with the same label appear in the same table, then they should be equal, irrespective of the type of columns where they appear.
- **Rule $\mathbf{r_6}$** : This rule disambiguates entities in columns of the same type. Here, we no longer consider the header of the column (as done by r_3 and r_4) but

compare the entities' labels. After experimenting with approximate string similarity measures, like the Levenshtein distance, we decided to use a case insensitive string equality (STR_EQ) to reduce the number of false positives. Case-insensitive similarity is more expensive than an exact string match because it requires dictionary lookups. We use it here and not in r_3, r_4, and r_5 because the comparisons are done only between entities of the same type.

- **Rule $\mathbf{r_7}$** : This rule implements another heuristic which takes into account the authors of the paper. It assumes that two entities are equal if they appear in two tables authored by the same author (we used the IDs provided by Semantic Scholar to disambiguate authors) and have the same label.

Once the reasoning has terminated, we introduce a new entity for each different null value and add RDF triples that link them to the corresponding cells and columns. Notice that the list of presented rules is not meant to be exhaustive. The ones that we describe show how we can exploit the predictions computed in the previous step (r_6) and external knowledge (r_7) relying on string similarity when no extra knowledge is available. We believe that additional EGDs, possibly designed to capture some specific cases, can further improve the performance.

6 Evaluation

Inputs. We considered two datasets: A corpus of tables that we manually constructed, and the dataset by [38], which is called *Tablepedia*.

Our corpus of tables contains 142,966 open-access PDFs distributed by Semantic Scholar. These papers appear in the proceedings of top venues in CS (the full list of venues is reported in our data repository). From these papers, we extracted 73,236 tables with PDFFigures and Tabula. These tables have 6.23 rows on average (SD = 6.58), and they have 7.11 columns (SD = 6.27). We converted the tables into RDF, resulting in a KB with 23M triples. We used Blazegraph to execute the SPARQL queries. After adding the table types and column types, we loaded the KB into VLog [8] to perform rule-based reasoning.

Tablepedia contains 451 tables, which have the columns annotated only with three classes: **Method**, **Dataset**, and **Metric**. To use this dataset in our pipeline, we created a graph representation of the tables without the annotations. Then, we translate the 15 *seed concepts* that are used in [38] to create the tables into labelling queries, so that we could apply Snorkel using both datasets. In contrast to Tablepedia, our annotated dataset maps to a much larger number of classes. Notice that the most frequent column types in our dataset (**Observation**, **Accuracy**, and **Count**), do not occur in Tablepedia.

Training Data. To create the training data for weak supervision, two human annotators (one PhD and one bachelor CS student) wrote SPARQL queries for labeling with the aid of a web interface designed for this purpose. The annotators examined the results of these queries on a sample of 400 tables, ensuring that the queries represented heuristics that covered a reasonable amount of the data. The quality of the SPARQL queries is fundamental to produce a good training

Method	Acc.
1^{st} Row	0.71
Ours	**0.76**

(a) Header detection

Model	Prec.	Recall	F1	AUC
SVM	0.71	0.79	0.74	0.86
LR	0.72	0.79	0.74	0.84
NB	**0.80**	**0.82**	**0.79**	**0.91**

(b) Table type prediction on our corpus

Task	MV	Snorkel
Table Types	0.50	**0.71**
Column Types (Our corpus)	**0.56**	0.49
Column Types (Tablepedia)	0.39	**0.65**

(c) MV vs. Snorkel

Model	Prec.	Recall	F1	AUC
NB	0.52	0.48	0.47	**0.87**
SVM	**0.58**	**0.56**	**0.53**	0.83
LR	**0.58**	**0.56**	**0.53**	**0.85**

(d) Column type prediction on our corpus

Model	Prec.	Recall	F1	AUC
Yu et al. [38]	0.82	0.81	0.81	0.90
NB	0.84	0.82	0.81	0.96
SVM	0.90	0.89	0.89	0.97
LR	**0.92**	**0.91**	**0.91**	**0.98**

(e) Column type prediction on Tablepedia

Fig. 3. Table interpretation with Naïve Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LR). MV is Majority Voting, AUC is area under the curve

dataset, and hence return good predictions. It is crucial that the queries have *large coverage* to avoid introducing a bias and to increase the training data size. For instance, if the queries label only a few tables, then the model will not receive enough evidence. To this end, we encouraged them to write queries which also matched a large number of items on the entire set of tables, and that did not excessively overlap. This resulted in 39 queries for labeling 98,570 tables with the corresponding type and 55 queries for labeling 165,302 columns.

Gold Standards. To test the performance, the same human annotators as before manually annotated 400 random tables. The tables in this sample have, on average, 9.92 rows (SD 7.28) and 5.07 columns (SD 3.20). These tables were annotated with the number of header rows, and table and column types. This process resulted in 321 table type and 873 column type annotations (excluding **Other**). Most tables were annotated with the **Observation** class (258), followed by **Input** (50); the smallest class was **Example** (13). The human annotators have annotated the table and column types looking at the images of the tables, the table captions, and possibly the full paper in case it was still not clear. The annotators have annotated the tables independently and resolved the conflicts together whenever they disagreed. After the first round of annotation using the first version of the ontology (44 classes), we marked as infrequent all classes with fewer than 10 annotations. These classes were removed from the ontology and the annotations were redirected to **Other**. For the Tablepedia dataset, we used the annotations provided by the original authors.

We highlight two aspects of our gold standard that have a direct impact on the evaluation. First, in contrast to [38], we decided not to filter out tables that were incorrectly extracted by Tabula. This makes our corpus more challenging because it might contain errors due to incorrect parsing. Second, our choice of merging infrequent column types into the type **Other** ensures that for each type there is always some evidence, but it has the downside that some classes in the long tail are ignored. Interpreting such types is an additional challenge that deserves a thorough study in future work.

6.1 Table Interpretation

Figure 3a reports the accuracy of our header detection heuristic compared to the baseline that consists of always selecting the 1^{st} row. We observe that our technique has superior performance, although it still makes some mistakes.

In Fig. 3b, we report the performance of our table type detection models on our gold standard. In general, we observe that all three models return reasonably high performance. Naïve Bayes (NB) outperformed the others, especially in terms of F_1 and AUC. Thus, we decided to select this as the default one for this task.

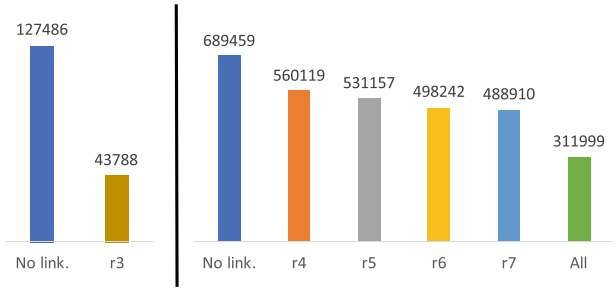
In Fig. 3d, we report the classifiers' performance for the column types on our gold standard, while Fig. 3e reports the same for Tablepedia. In both cases, we see that LR performs best, likely due to the combined importance of textual and numeric features for this task. Additionally, we observe that our model significantly outperforms the model of [38] on their dataset. If we compare the scores between the two datasets, then we see that they are significantly lower with our dataset. The reason is two-fold: First, the authors of Tablepedia have manually removed much noise from the extracted tables while no pre-processing took place on our dataset. Second, our dataset contains many more classes than Tablepedia, which makes it more challenging to predict.

Finally, we studied the added value of using Snorkel and compared it with a simpler majority voting (MV), i.e., labeling a data point using the most frequently predicted class. In Fig. 3c, we report both the accuracy obtained with majority voting and with Snorkel with various types of predictions. While Snorkel outperforms MV for the table type detection and column type detection in Tablepedia, MV is better when detecting the column types of our corpus. This was expected because, in this last case, our labeling functions (i.e., SPARQL queries) have frequently abstained. Consequently, M has a low label density, and whenever this occurs, Snorkel is unable to compute optimal weights that diverge from MV [30].

6.2 Entity Linking

Figure 4a reports the number of entities before and after the execution of the EGD rules. The left side compares the number of entities that refer to columns before and after r_3 was executed. As we can see, r_3 merged many entities, and this reduced the number of distinct entities of 65%. The right side shows the decrease of entities that refer to cells after the execution of rules $r_4, \ldots, r_7$. Here, the bar titled r_i reports the number of entities if only r_i is executed while the right-most column indicates the number of entities when all rules are included. We observe that every EGD contributes to merge some entities, but the best results are obtained when all EGDs are activated: here, the EGDs merged about 55% of the entities.

To evaluate the quality of entity links, we manually evaluated a sample of 100 merged entities. For each sampled entity, we first determined whether the entity was a meaningful one. From this analysis, we discovered that 65% of the entities are correct while the remaining have either nonsensical labels or some



(a) Ablation study. The bar marked with r_i reports the number of entities when only EGD r_i is included in the rule set

	Label	# Links
Good ✓	mnist	288
	knn	211
	wiki	146
	cifar-10	108
	en-es	65
Bad ✗	after	183
	analysis	66
	subset	49
	0/0/0	9
	f4(x)	6

(b) Examples

Fig. 4. Analysis of the performance of entity linking

text resulted from errors of Tabula. In Fig. 4b, we report examples of good and bad entities with their number of links.

Then, we looked at the cells which referred to the entity, which were 541 in total. Since the rules could make a mistake and link two cells to the same entity although they meant different ones, we evaluated, for each entity, the precision of its links. Given the set of n cells that link to the same entity, the precision is computed by taking the cardinality of the largest subset of cells that refer to the same concept and divide it by n. For instance, consider an entity X with label Y which is linked to $n = 4$ cells. Three of these cells contain the text Y but refer to a dataset while one cell contains Y but refers to something else. In this case, the precision for X is $\frac{3}{4}$. In our sample, the average precision over the meaningful entities was about 97%, which is a relatively high value. This indicates that reasoning produced an accurate entity linking.

7 Conclusion

Summary. We presented Tab2Know, an end-to-end system for building a KB from the knowledge in scientific tables. One distinctive feature of Tab2Know is the usage of SPARQL queries for weak supervision to counter the lack of training data. Another distinctive feature is the usage of existentially quantified rules to link the entities without the help of a pre-existing KB.

Our pipeline effectively combines statistical-based classification and logical reasoning, exploiting SPARQL and remote KBs like Semantic Scholar. Therefore, we believe that ours is an excellent example of how semantic web technologies, statistical- and logic-based AI can be used side-by-side.

Future Work. Although our results are encouraging, and the current KB is already able to answer some non-trivial queries, future work is required to improve the performance. First, a more accurate table extraction procedure is needed to improve the accuracy of table interpretation and entity linking. Moreover, our current ontology links classes only via $\sqsubseteq$. It is interesting to study

whether new relations can lead to better extractions. For instance, specifying the range of some classes could be used to exclude mappings to columns with incompatible values. Finally, a natural continuation of our work is to further research whether additional rules can return a better entity linking. In particular, we believe that rules that take into account the context of the table or co-authorship networks will be particularly useful.

To conclude, we believe that Tab2Know represents one more step that brings us closer to solve the problem of constructing an extensive and accurate KB of scientific knowledge. Such a KB is a useful asset for assisting the researchers, and it can play a crucial role in turning the vision of open science into a reality.

References

1. Ammar, W., et al.: Construction of the Literature Graph in Semantic Scholar. In: NAACL, pp. 84–91 (2018)
2. Benedikt, M., et al.: Benchmarking the chase. In: PODS, pp. 37–52 (2017)
3. Bhagavatula, C.S., Noraset, T., Downey, D.: TabEL: entity linking in web tables. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 425–441. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25007-6_25
4. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, New York (2006)
5. Bizer, C., et al.: DBpedia-a crystallization point for the web of data. J. Web Semant. **7**(3), 154–165 (2009)
6. Böhm, C., de Melo, G., Naumann, F., Weikum, G.: LINDA: distributed web-of-data-scale entity matching. In: CIKM, pp. 2104–2108 (2012)
7. Cappuzzo, R., Papotti, P., Thirumuruganathan, S.: Creating embeddings of heterogeneous relational datasets for data integration tasks. In: SIGMOD, pp. 1335–1349 (2020)
8. Carral, D., Dragoste, I., González, L., Jacobs, C., Krötzsch, M., Urbani, J.: VLog: a rule engine for knowledge graphs. In: ISWC, pp. 19–35 (2019)
9. Chen, J., Jiménez-Ruiz, E., Horrocks, I., Sutton, C.A.: ColNet: embedding the semantics of web tables for column type prediction. In: AAAI, pp. 29–36 (2019)
10. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. TKDE **24**(9), 1537–1555 (2012)
11. Clark, C., Divvala, S.: PDFFigures 2.0: mining figures from research papers. In: JCDL, pp. 143–152 (2016)
12. Das, S., et al.: Falcon: scaling up hands-off crowdsourced entity matching to build cloud services. In: SIGMOD, pp. 1431–1446 (2017)
13. Das Sarma, A., et al.: Finding related tables. In: SIGMOD, pp. 817–828 (2012)
14. Efthymiou, V., Hassanzadeh, O., Rodriguez-Muro, M., Christophides, V.: Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 260–277. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_16
15. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theoret. Comput. Sci. **336**(1), 89–124 (2005)
16. Fang, J., Mitra, P., Tang, Z., Giles, C.L.: Table header detection and classification. In: AAAI, pp. 599–605 (2012)
17. Fetahu, B., Anand, A., Koutraki, M.: TableNet: an approach for determining fine-grained relations for wikipedia tables. In: WWW, pp. 2736–2742 (2019)

18. Geerts, F., Mecca, G., Papotti, P., Santoro, D.: That's all folks! LLUNATIC goes open source. PVLDB **7**(13), 1565–1568 (2014)
19. Horridge, M., Gonçalves, R.S., Nyulas, C.I., Tudorache, T., Musen, M.A.: Webprotégé: A cloud-based ontology editor. In: WWW, pp. 686–689 (2019)
20. Kavasidis, I., et al.: A saliency-based convolutional neural network for table and chart detection in digitized documents. In: ICIAP, pp. 292–302 (2019)
21. Konda, P., et al.: Magellan: toward building entity matching management systems. PVLDB **9**(12), 1197–1208 (2016)
22. Konstantinou, N., et al.: VADA: an architecture for end user informed data preparation. J. Big Data **6**(1), 1–32 (2019). https://doi.org/10.1186/s40537-019-0237-9
23. Kruit, B., Boncz, P., Urbani, J.: Extracting novel facts from tables for knowledge graph completion. In: Ghidini, C., et al. (eds.) ISWC 2019. LNCS, vol. 11778, pp. 364–381. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30793-6_21
24. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. PVLDB **3**(1–2), 1338–1347 (2010)
25. Luo, X., Luo, K., Chen, X., Zhu, K.Q.: Cross-lingual entity linking for web tables. In: AAAI, pp. 362–369 (2018)
26. Mudgal, S., et al.: Deep learning for entity matching: a design space exploration. In: SIGMOD, pp. 19–34 (2018)
27. Oro, E., Ruffolo, M.: PDF-TREX: an approach for recognizing and extracting tables from PDF documents. In: ICDAR, pp. 906–910 (2009)
28. Papadakis, G., Ioannou, E., Palpanas, T.: Entity resolution: Past, present and yet-to-come. In: EDBT, pp. 647–650 (2020)
29. Pinto, D., McCallum, A., Wei, X., Croft, W.B.: Table extraction using conditional random fields. In: SIGIR, pp. 235–242 (2003)
30. Ratner, A., Bach, S.H., Ehrenberg, H., Fries, J., Wu, S., Ré, C.: Snorkel: rapid training data creation with weak supervision. VLDB J. **29**(2), 709–730 (2020)
31. Ratner, A., Hancock, B., Dunnmon, J., Sala, F., Pandey, S., Ré, C.: Training complex models with multi-task weak supervision. In: AAAI, pp. 4763–4771 (2019)
32. Ritze, D., Lehmberg, O., Bizer, C.: Matching HTML tables to DBpedia. In: WIMS, pp. 1–6 (2015)
33. Ritze, D., Lehmberg, O., Oulabi, Y., Bizer, C.: Profiling the potential of web tables for augmenting cross-domain knowledge bases. In: WWW, pp. 251–261 (2016)
34. Schreiber, S., Agne, S., Wolf, I., Dengel, A., Ahmed, S.: DeepDeSRT: deep learning for detection and structure recognition of tables in document images. In: ICDAR, pp. 1162–1167 (2017)
35. Siegel, N., Horvitz, Z., Levin, R., Divvala, S., Farhadi, A.: FigureSeer: parsing result-figures in research papers. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9911, pp. 664–680. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46478-7_41
36. Siegel, N., Lourie, N., Power, R., Ammar, W.: Extracting scientific figures with distantly supervised neural networks. In: JCDL, pp. 223–232 (2018)
37. Vine, N.L., Zeigenfuse, M., Rowan, M.: Extracting tables from documents using conditional generative adversarial networks and genetic algorithms. In: IJCNN, pp. 1–8 (2019)
38. Yu, W., Peng, W., Shu, Y., Zeng, Q., Jiang, M.: Experimental evidence extraction system in data science with hybrid table features and ensemble learning. In: WWW, pp. 951–961 (2020)
39. Zhang, Z.: Effective and efficient semantic table interpretation using TableMiner+. Semant. Web **8**(6), 921–957 (2017)