Matching Web Tables To DBpedia - A Feature Utility Study

Dominique Ritze, Christian Bizer
Data and Web Science Group, University of Mannheim,
B6, 26 68159 Mannheim, Germany
{dominique,chris}@informatik.uni-mannheim.de

ABSTRACT

Relational HTML tables on the Web contain data describing a multitude of entities and covering a wide range of topics. Thus, web tables are very useful for filling missing values in cross-domain knowledge bases such as DBpedia, YAGO, or the Google Knowledge Graph. Before web table data can be used to fill missing values, the tables need to be matched to the knowledge base in question. This involves three matching tasks: table-to-class matching, row-to-instance matching, and attribute-to-property matching. Various matching approaches have been proposed for each of these tasks. Unfortunately, the existing approaches are evaluated using different web table corpora. Each individual approach also only exploits a subset of the web table and knowledge base features that are potentially helpful for the matching tasks. These two shortcomings make it difficult to compare the different matching approaches and to judge the impact of each feature on the overall matching results.
This paper contributes to improve the understanding of the utility of different features for web table to knowledge base matching by reimplementing different matching techniques as well as similarity score aggregation methods from literature within a single matching framework and evaluating different combinations of these techniques against a single gold standard. The gold standard consists of class-, instance-, and property correspondences between the DBpedia knowledge base and web tables from the Web Data Commons web table corpus.

1. INTRODUCTION

Cross-domain knowledge bases such as DBpedia [18], YAGO [17], or the Google Knowledge Graph [36] are used as background knowledge within an increasing range of applications including web search, natural language understanding, data integration, and data mining. In order to realize their full potential within these applications, cross-domain knowledge bases need to be as complete, correct, and up-to-date as possible. One way to complement and keep a knowledge base up to date is to continuously integrate new knowledge from external sources into the knowledge base [10].

Relational HTML tables from the Web (also called web tables) are a useful source of external data for complementing and updating knowledge bases [31, 10, 40] as they cover a wide range of topics and contain a plethora of information. Before web table data can be used to fill missing values ("'slot filling"') or verify and update existing ones, the tables need to be matched to the knowledge base. This matching task can be divided into three subtasks: table-to-class matching, row-to-instance matching, and attribute-to-property matching. Beside the use case of complementing and updating knowledge bases, the matching of web tables is also necessary within other applications such as data search [40, 1] or table extension [41, 8, 21].

Matching web tables to knowledge bases is tricky as web tables are usually rather small with respect to their number of rows and attributes [19] and as for understanding the semantics of a table, it is often necessary to partly understand the content of the web page surrounding the table [41, 20]. Since everybody can put HTML tables on the Web, any kind of heterogeneity occurs within tables as well as on the web pages surrounding them. In order to deal with these issues, matching systems exploit different aspects of web tables (features) and also leverage the page content around the tables (context) [42, 41, 19].

There exists a decent body of research on web table to knowledge base matching [3, 22, 25, 39, 42, 16, 32]. Unfortunately, the existing methods often only consider a subset of the three matching subtasks and rely on a certain selection of web table and knowledge base features. In addition, it is quite difficult to compare evaluation results as the systems are tested using different web table corpora and different knowledge bases, which in some cases are also not publicly available. What is missing is an transparent experimental survey of the utility of the proposed matching features using a single public gold standard covering all three matching subtasks.

Whenever different features are used for matching, a method is required to combine the resulting similarity scores. While certain similarity aggregation methods work well for some tables, they might deliver bad results for other tables. Thus in addition to comparing different features and respective similarity functions, we also compare different similarity ag-

gregation methods. We focus the comparison on matrix prediction methods [33, 5] which are a specific type of similarity aggregation methods that predict the reliability of different features for each individual table and adapt the weights of the different features accordingly.

The contributions of this paper are twofold:

- We provide an overview and categorization of the web table and knowledge base features (together with respective similarity and similarity aggregation methods) that are used in state-of-the-art web table matching systems.
- We analyze the utility of the different matching features using a single, public gold standard that covers all three subtasks of the overall matching task. The gold standard consists of class-, instance-, and property correspondences between the DBpedia knowledge base [18] and web tables from the Web Data Commons table corpus [19].

The paper is organized as follows: Section 2 gives an overview of the overall matching process. Section 3 describes and categorizes the web table and knowledge base features. Section 4 discusses how the features can be used within the three matching tasks and describes the matchers that are employed to exploit the features within the experiments. The aggregation of similarity scores using matrix predictors is discussed in Section 5. Section 6 describes the gold standard that is used for the experiments. Section 7 compares the results of the different matrix prediction methods. Section 8 presents the matching results, compares them with existing results from the literature, and analyzes the utility of each feature for the matching tasks. Conclusions are drawn in Section 9.

2. OVERALL MATCHING PROCESS

We use the model and terminology introduced by Gal and Sagi in [15] to describe the overall process of matching a set of web tables and a knowledge base. Figure 1 shows an exemplary matching process. As input, two sources are required while as output, the process generates correspondences between manifestations of the sources. We consider everything within the sources a manifestation, e.g. manifestations are rows and columns of a table as well as instances, properties, and classes within a knowledge base. The internal components of a process are called first line matchers (1LM) and second line matchers (2LM).

A first line matcher (1LM) takes one feature of the manifestations as input and applies a similarity measure. As an example, a first line matcher gets the labels of the different attributes (columns) of a web table and the labels of the properties of a specific class within the knowledge base as feature, tokenizes both labels, removes stop words, and compares the resulting sets using the Jaccard similarity. The resulting similarity scores are stored as elements in a similarity matrix. In most cases, only considering a single feature is not sufficient for matching two sources. Thus, an ensemble of first line matchers is applied, ideally covering a wide variety of features exploiting different aspects of the web tables and the knowledge base.

Second line matchers (2LM) transform one or more similarity matrices into a resulting similarity matrix. Gal [14] distinguishes decisive and non-decisive second line matchers. Non-decisive matchers do not take any decision about the resulting correspondences, e.g. they only aggregate matrices. Typical aggregation strategies of non-decisive second line matchers are to take the maximal elements that can be found among the matrices or to weight each matrix and calculate a weighted sum. In the example depicted in Figure 1, the aggregation is performed by summing up the elements of both matrices. Non-decisive second line matchers are also referred to as combination methods [9] or matcher composition [11].

In contrast to non-decisive matchers, decisive second line matchers create correspondences between manifestations. For instance, a second-line matcher that applies a threshold is decisive because it excludes all pairs of manifestations having a similarity score below this threshold. It is often desirable that a single manifestation within a web table is only matched to a single manifestation in the knowledge base. To ensure this, so called $1:1$ decisive second line matchers are used. In our example, the $1:1$ matcher decides for the highest element within each matrix row and sets them to 1, all other elements are set to 0.

3. FEATURES

Features are different aspects of web tables and the knowledge base that serve as input for first line matchers. We perceive web tables as simple entity-attribute tables, meaning that each table describes a set of entities (rows in web tables) having a set of attributes (columns). For each entity-attribute pair, we can find the according value in a cell. We require every table to have an attribute that contains natural

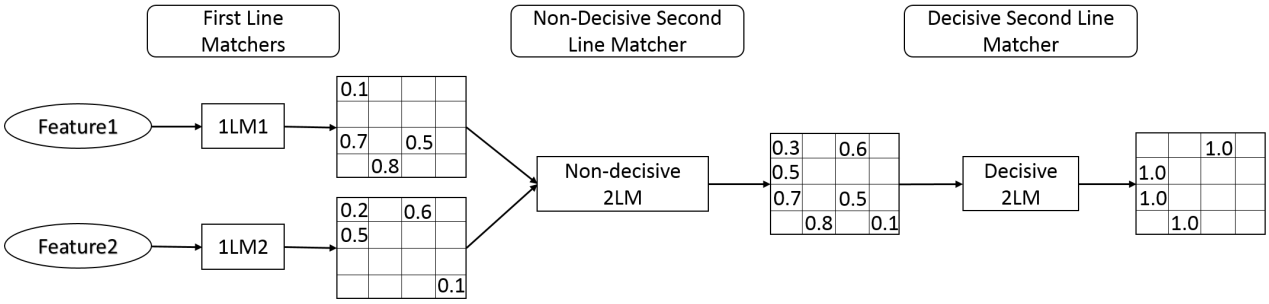

Figure 1: The matching process

language labels of the entities (called entity label attribute), e.g. the entity label of the city Mannheim is "Mannheim". All other attributes are either of data type string, numeric or date. We currently do not consider any other data types like geographical coordinates as for instance taken into account by Cruz et al. [6] or tables with compound entity label attributes [20]. Further, each attribute is assumed to have a header (attribute label) which is some surface form of the attribute's semantic intention. In order to distinguish between web tables and the knowledge base, we use the terms entity and attribute when talking about web tables and instance and property when talking about the knowledge base.

We use the categorization schema shown in Figure 2 for categorizing web tables features. In general, a feature can either be found in the table itself (Table T) or outside the table (Context C). As context features, we consider everything that is not directly contained in the table, e.g. the words surrounding the table. Context features can either be page attributes (CPA) like the page title or free text (CFT). We further divide table features into single features (TS), e.g. a label of an entity, and multiple features (TM), e.g. the set of all attribute labels occurring in a table. Single features refer to a value in a single cell while multiple features combine values coming from more than one cell.

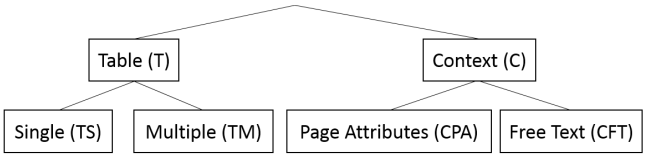

Figure 2: Web table feature categories

Table 1 gives an overview of all features that we consider and classifies them by category. As single table features we use the entity label, the attribute label, as well as the values that can be found in the cells. Multiple features are the entities as a whole, the set of attribute labels, and the table as text. We represent multiple features as bag-of-words. For example, the set of attribute labels can be characteristic for a table, e.g. the attribute labels "population" and "currency" give an important hint that the table describes different countries.

As context features, we use the page attributes title and URL and as free text feature the words surrounding the table. Often, the URL as well as the title of the web page contains information about the content of the table, e.g. the URL http://airportcodes.me/us-airport-codes indicates that a table found on this page might describe a set of airports. Context features are often not directly related to a specific table which makes it tricky to exploit them for matching. Nevertheless Yakout et al. [41] as well as Lehmberg [20] found context features to be crucial for high quality matching. Braunschweig et al. [1] take the surrounding words to extract attribute-specific context in order to find alternative names for attribute labels. The CONTEXT operator of the Octopus system [3] uses context features to find hidden attributes which are not explicitly described in the table.

Most state-of-the-art matching systems only exploit single table features [26, 43, 38, 40, 25, 22, 24, 39, 23, 16]. Multiple table features are considered by Wang et al. [40] (set of attribute labels), by the TableMiner system [42] (set of attribute labels and entities), and by the InfoGather system [41] (set of attribute labels, entities, and tables). Only the systems InfoGather and TableMiner leverage context features.

Table 2 shows the features of the knowledge base (DBpedia) that we exploit within the experiments. Analog to the table features, DBpedia features can either refer to a single triple, e.g. a triple representing the information about an instance label, or to a set of triples like the set of all abstracts of instances belonging to a certain class.

In addition to the web table and knowledge base features, external resource can be exploited for matching, e.g. general lexical databases like WordNet [12]. For matching web tables, systems use external resources which have been created based on co-occurrences [39], that leverage a web text corpus and natural language patterns to find relations between entities [35], or that exploit the anchor text of hyperlinks in order to find alternative surface forms of entity names [2].

4. MATCHING TASKS

The overall task of matching web tables against a knowledge base can be decomposed into the subtasks table-to-class matching, row-to-instance matching, and attribute-to-property matching. In this section, we provide an overview of the existing work on each subtask. Afterward, we describe the matching techniques that we have selected from the literature for our experiments. We employ the $T2KMatch$ matching framework [32][1] for the experiments and implement the selected techniques as first line matchers within

[1]http://dws.informatik.uni-mannheim.de/en/research/T2K

Table 1: Web table features

Feature	Description	Category
Entity label	The label of an entity	TS
Attribute label	The header of an attribute	TS
Value	The value that can be found in a cell	TS
Entity	The entity in one row represented as a bag-of-words	TM
Set of attribute labels	The set of all attribute labels in the table	TM
Table	The text of the table content without considering any structure	TM
URL	The URL of the web page from which the table has been extracted	CPA
Page title	The title of the web page	CPA
Surrounding words	The 200 words before and after the table	CFT

Table 2: DBpedia features

Feature	Description
Instance label	The name of the instance mentioned in the rdfs:label
Property label	The name of the property mentioned in the rdfs:label
Class label	The name of the class mentioned in the rdfs:label
Value	The literal or object that can be found in the object position of triples
Instance count	The number of times an instance is linked in the wikipedia corpus
Instance abstract	The DBpedia abstract describing an instance
Instance classes	The DBpedia classes (including the superclasses) to which an instance belongs to
Set of class instances	The set of instances belonging to a class
Set of class abstracts	The set of all abstracts of instances belonging to a class

the framework. The framework covers all three matching subtasks. Similar to PARIS [37], $T2KMatch$ iterates between instance- and schema matching until the similarity scores stabilize. Correspondences between tables and classes are chosen based on the initial results of the instance matching. Due to this decision, only instances of this class as well as properties defined for this class are taken into account. Thus, the class decision can have a strong influence on the other matching tasks [32].

4.1 Row-To-Instance Matching

The goal of row-to-instance matching is to find correspondences between instances in the knowledge base and entities described by individual rows of a web table. The row-to-instance matching task is tackled frequently by various systems in literature. Some systems purely rely on the label of the entity [26, 43] or on the label enriched with alternatives surface forms [22]. In addition, other systems also take the cell values into account [42, 38, 40, 25]. Most of them have in common that they query APIs to find potential instances, e.g. the Probase, Freebase or Wikiontology API. As a result, a ranked list of possible instances per entity is returned. The ranking function is not always known in detail but often relies on the popularity of an instance. Besides the internal API ranking, other rankings like the page rank of the according Wikipedia page of an instances can be added [26, 38]. As another source of information, Zhang [42] introduced context features (page title, surrounding words).

Within our experiments, we evaluate the utility of all single table features as well as the entity feature for the row-to-instance matching task. For this, we have implemented the following first line matchers within the $T2KMatch$ framework:

Entity Label Matcher: Before the entity label can be matched, we need to identify the attribute of the web tables that contains the entity label (entity label attribute). For determining the entity label attribute, we use a heuristic which exploits the uniqueness of the attribute values and falls back to the order of the attributes for breaking ties [32]. For matching the entity label, we apply the entity label matcher that is included in $T2K$. The matcher compares the entity label with the instance label using a generalized Jaccard with Levenshtein as inner measure. Only the top 20 instances with respect to the similarities are considered further for each entity.

Value-based Entity Matcher: $T2KMatch$ implements a value matcher which applies data type specific similarity measures. For strings, a generalized Jaccard with Levenshtein as inner measure, for numeric the deviation similarity introduced by Rinser et al. [30], and for dates a weighted date similarity is used which emphasizes the year over the month and day. The value similarities are weighted with the available attribute similarities and are aggregated per entity. If we already know that an attribute corresponds to a property, the similarities of the according values get a higher weight.

Surface Form Matcher: Web tables often use synonymous names ("surface forms") to refer to a single instance in the knowledge base, which is difficult to spot for pure string similarity measures. In order to be able to understand alternative names, we use a surface form catalog that has been created from anchor-texts of intra-Wikipedia links, Wikipedia article titles, and disambiguation pages [2]. Within the catalog, a TF-IDF score [34] is assigned to each surface form. We build a set of terms for each label resp. string value consisting of the label/value itself together with according surface forms. We add the three surface forms with the highest scores if the difference of the scores between the two best surface forms is smaller than 80%, otherwise we only add the surface form with the highest score. For each entity label resp. value, we build a set of terms containing the label or value as well as the alternative names. Each term in the set is compared using the entity label resp. value-based entity matcher and the maximal similarity per set is taken.

Popularity-based Matcher: The popularity-based matcher takes into account how popular an instance in the knowledge base is. For example, an instance with the label "Paris" can either refer to the capital of France or to the city in Texas. Both instances are equal regarding the label but most of the times, the city in France will be meant. To compute the popularity of an instance, we count the number of links in Wikipedia that point at the Wikipedia page which corresponds to the instance [7]. Similar methods based on the Wikipedias instance's page rank are applied by Mulwad et al. [26] and Syed at al. [38].

Abstract Matcher: Comparing the entity label and the values can be insufficient if the labels differ too much or if not all information about an instance is covered in the values, e.g. the capital of a country is not contained in the knowledge base as a value but stated in the abstract of the instance. This is especially relevant when thinking about the use case of filling missing values in the knowledge base. Therefore,

the abstract matcher compares the entity as a whole with the abstracts of the instances, both represented as bag-of-words. For each entity represented as bag-of-words, we create a TF-IDF vector and compare it to the TF-IDF vectors constructed from the abstracts where at least one term overlaps. As similarity measure we use a combination of the denormalized cosine similarity (dot product) and Jaccard to prefer vectors that contain several different terms in contrast to vectors that cover only one term but this several times:

$A \bullet B + 1 - (\frac{1}{\|A \cap B\|})$ where A and B are TF-IDF vectors.

4.2 Attribute-To-Property Matching

The attribute-to-property matching task has the goal to assign properties from the knowledge base (both data type and object properties) to the attributes found in web tables. Existing attribute-to-property matching methods often focus on the matching only object properties to attributes[26, 25, 24], also named "relation discovery". As cross-domain knowledge bases usually contain data type and object properties, the goal in this paper is to detect correspondences for both types of properties. Beside exploiting attribute and property values, other methods also take the attribute label into account and compare it to the label of the property [22]. Similar to the instance matching task, the label comparison can be enhanced by including alternative attribute labels, e.g. computed based on co-occurrences [41]. The system introduced by Braunschweig et al. [1] discovers synonymous labels by using the context as well as the lexical database WordNet. Neumaier et al. [27] present a matching approach that explicitly focuses on numeric data which is published via open data portals.

Within our experiments, we evaluate solely single features for attribute-to-property matching and have implemented the following matchers for this:

Attribute Label Matcher: The attribute label can give hints which information is described by the attribute. For example, the label "capital" in a table about countries directly tells us that a property named "capital" is a better candidate than the property "largestCity" although the similarities of the values are very close. We use a generalized Jaccard with Levenshtein as inner measure to compare the attribute and property label.

WordNet Matcher: To solve alternative names for attribute labels, we consult the lexical database WordNet which has also been used by Braunschweig et al. [1]. WordNet is frequently applied in various research areas, e.g. in the field of ontology matching. Besides synonyms, we take hypernyms and hyponyms (also inherited, maximal five, only coming from the first synset) into account. As an example, for the attribute label "country" the terms "state", "nation", "land" and "commonwealth" can be found in WordNet. We again apply a set-based comparison which returns the maximal similarity scores.

Dictionary Matcher: While WordNet is a general source of information, we additionally create a dictionary for attribute labels based on the results of matching the Web Data Commons Web Tables Corpus to DBpedia with $T2KMatch$. As a result, from 33 million tables around 1 million tables have at least one instance correspondence to DBpedia [31]. We group the property correspondences and extract the according labels of the attributes that have been matched to a property. Thus, we are able to generate a dictionary containing the property label together with the attribute labels that, based on the matching, seem to be synonymous. At this point, the dictionary includes a lot of noise, e.g. the term "name" is a synonym for almost every property. A filtering based on the number of occurrences or on the number of web sites is not useful, since the rare cases are most promising. Thus, we apply a filter which excludes all attribute labels that are assigned to more than 20 different properties because they do not provide any benefit. The comparison is the same as for the other matchers including external resources. A related approach is performed by Yakout et al. [41] where synonyms of attribute labels are generated based on web tables that have been matched among each other.

Duplicate-based Attribute Matcher: The duplicate-based attribute matcher is the counterpart of the value-based entity matcher: The computed value similarities are weighted with the according instance similarities and are aggregated over the attribute. Thus, if two values are similar and the associated entity instance pair is similar, it has a positive influence on the similarity of the attribute property pair, see [32] for more details.

4.3 Table-To-Class Matching

The goal of table-to-class matching is to assign the class from the knowledge base to a web table which fits best to the content of the whole table. Assigning the class to which the majority of the instances in the table belong to is most common strategy for table-to-class matching [22, 42, 39, 23, 38, 43, 16]. On top of this approach, methods take also the specificity of a class into account [25], exploit the set of attribute labels [40] or consider the context [42].

We evaluate the utility of features from the categories "table multiple" and "context" for table-to-class matching and have implemented the following matchers for this:

Page Attribute Matcher: We process the page attributes page title and URL by applying stop word removal and simple stemming. The similarity of a page attribute to a class of the knowledge base is the number of characters of the class label normalized by the number of characters in the page attribute.

Text Matcher: Ideally, the set of abstracts belonging to instances of a class contains not only the instance labels and associated property labels but also significant clue words. We use this matcher for the features "set of attribute labels", "table" and "surrounding words". All features are represented as bag-of-words. After removing stop words, we build TF-IDF vectors indicating the characteristic terms of the table and the classes. We apply the same similarity measure which is used by the abstract matcher.

Majority-based Matcher: Based on the initial similarities of entities to instances computed by the entity label matcher, we take the classes of the instances and count how often they occur. If an instance belongs to more than one

class, the instance counts for all of them. Such a matching approach has for example been applied by Limaye et al. [22] to assign classes to attributes covering named entities.

Frequency-based Matcher: Ideally, we want to find correspondences to specific classes over general classes which is not captured by the majority-based class matcher. Similar to Mulwad et al. [25], we define the specificity of a class as following:

$$spec(c) = 1 - \frac{\|c\|}{\max_{d \in C} \|d\|}$$

where c represents a particular class and C the set of all classes in DBpedia.

Agreement Matcher: The agreement matcher is a second line matcher which exploits the amount of class matchers operating on features covering different aspects. Although the matchers might not agree on the best class to choose, a class which is found by all the matchers is usually a good candidate. We propose the agreement matcher which takes the results of all other class matchers and counts how often they agree per class. In this case, all classes are counted having a similarity score greater than zero.

The results of our matching experiments are presented in Section 8.

5. SIMILARITY SCORE AGGREGATION

Each of the previously described matchers generates a similarity matrix as result. Depending on the task, these matrices contain the similarities between the entities and instances, attributes and properties or the table and classes. In order to generate the correspondence, all matrices dealing with the same task need to be combined which is the task of a non-decisive second line matcher. Most approaches in the field of web table matching use a weighted aggregation to combine similarity matrices. While some of them empirically determine the weights, e.g. TableMiner [42], others employ machine learning to find appropriate weights [41, 22]. All existing approaches for web table matching have in common that they use the same weights for all tables. Due to the diversity of tables, one single set of weights might not be the best solution. To overcome this issue, we use a quality-driven combination strategy which adapts itself for each individual table. Such strategies have been shown as promising in the field of ontology matching [5]. The approach tries to measure the reliability of matchers by applying so called matrix predictors [33] on the generated similarity matrices. The predicted reliability is then used as weight for each matrix. Since the prediction is individually performed on each matrix, the reliability of a matcher can differ for each table and in turn we are able to use weights which are tailored to a table.

We evaluate three different matrix predictors: the average predictor (P_{avg}), standard deviation predictor (P_{stdev}) [33] as well as a predictor (P_{herf}) which bases on the Herfindahl Index [29] and estimates the diversity of a matrix.

Average: Based on the assumption that a high element in the similarity matrix leads to a correct correspondence, a matrix with many high elements is preferred over a matrix with less high elements. We compute the average of a matrix M as following:

$$P_{avg}(M) = \frac{\sum_{i,j|e_{i,j}>0} e_{i,j}}{\sum_{i,j|e_{i,j}>0} 1}$$

Standard Deviation: In addition to the average, the standard deviation indicates whether the elements in the matrix are all close to the average. Formally:

$$P_{stdev}(M) = \sqrt{\frac{\sum_{i,j|e_{i,j}>0}(e_{i,j}-\mu)^2}{N}}$$

μ is the average and N is the number of non-zero elements.

Normalized Herfindahl Index: The Herfindahl Index (HHI) [29] is an economic concept which measures the size of firms in relation to the industry and serves as an indicator of the amount of competition among them. A high Herfindahl Index indicates that one firm has a monopoly while a low Herfindahl Index indicates a lot of competition. We use this concept to determine the diversity of each matrix row and in turn of the matrix itself. Our matrix predictor based on the Herfindahl Index is similar to the recently proposed predictor *Match Competitor Deviation* [13] which compares the elements of each matrix row with its average.

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

Figure 3: Matrix row with the highest HHI (1.0)

$$\begin{bmatrix} 0.1 & 0.1 & 0.1 & 0.1 \end{bmatrix}$$

Figure 4: Matrix row with the lowest HHI (0.25)

Figure 3 and Figure 4 show the highest and lowest possible case for a four-dimensional matrix row. At best, we find exactly one element larger than zero while all other elements are zero. Having this ideal case, we can perfectly see which pair fits. In contrast, a matrix row which has exactly the same element for each pair does not help at all to decide for correspondences. We compute the normalized Herfindahl Index for each matrix row which ranges between $1/n$ and 1.0 where n is the dimension of the matrix row. That is the reason why the matrix row in Figure 3 has a normalized Herfindahl Index of 1.0 and the matrix row in Figure 4 of 0.25. To get an estimation per matrix, we build the sum over all Herfindahl Indices per matrix row and normalize it. Formally:

$$P_{herf}(M) = \frac{1}{V} \sum_i \frac{\sum_j e_{i,j}{}^2}{(\sum_j e_{i,j})^2}$$

where V represents the number of matrix rows in the matrix.

Section 7 presents the evaluation results of the different matrix predictors and determines the predictor that is most suitable for each matching task. Further, we discuss how the weights are distributed across the different matrices generated by the matchers.

6. GOLD STANDARD

We use Version 2 of the *T2D* entity-level gold standard[2] for our experiments. The gold standard consists of web tables from the Web Data Commons table corpus [19] which

[2]http://webdatacommons.org/webtables/goldstandardV2.html

has been extracted from the CommonCrawl web corpus[3]. During the extraction, the web tables are classified as layout, entity, relational, matrix and other tables. For the use case of filling missing values in a knowledge base, relational tables are most valuable as they contain relational data describing entities. However, as shown by Cafarella et al. [4], the vast majority of tables found in the Web are layout tables. In addition we have shown in [31] that only a very small fraction of the relational tables can actually be matched to the DBpedia knowledge base. Thus, it is important for a matching algorithm to be good at recognizing non-matching tables. For a gold standard, it is in turn important to contain non-matching tables.

Version 2 of the $T2D$ entity-level gold standard consists of row-to-instance, attribute-to-property, table-to-class correspondences between 779 web tables and the DBpedia knowledge base. The correspondences were created manually. In order cover the challenges that a web table matching system needs to face, the gold standard contains three types of tables: non-relational tables (layout, matrix, entity, other), relational tables that do not share any instance with DBpedia and relational tables for which least one instance correspondence can be found. Out of the 779 tables in the gold standard, 237 tables share at least one instance with DBpedia. The tables cover different topics including places, works, and people. Altogether, the gold standard contains 25 119 instance and 618 property correspondences. About half the property correspondences refer to entity label attributes, while 381 correspondences refer to other attributes (object as well as data type attributes). Detailed statistics about the gold standard are found on the web page mentioned above.

A major difference between Version 2 of the $T2D$ gold standard and the Limaye112 gold standard [22] is that the $T2D$ gold standard includes tables that cannot be matched to the knowledge base and thus forces matching systems to decide whether a table can be matched or not.

7. SIMILARITY AGGREGATION RESULTS

This section describes the experiments we perform regarding the similarity score aggregation using matrix predictors. Following Sagi and Gal [33], we measure the quality of a matrix predictor using the Pearson product-moment correlation coefficient [28]. With a correlation analysis, we can ensure that the weights chosen for the aggregation are well suitable. We perform the correlation analysis for the three matching tasks with the three introduced matrix preditors P_{avg}, P_{stdev} and P_{herf} on the evaluation measures precision P and recall R.

$$P = \frac{TP}{TP + FP} \qquad R = \frac{TP}{TP + FN}$$

While TP refers to the number of true positives, FP represents the number false positives and FN the number of false negatives.

If a predictor has a high correlation to precision respect recall and we use the prediction for weighting the similarity matrix, we assume that the result also has an according precision/recall.

[3]http://commoncrawl.org/

Table 3 shows the results of the correlation analysis for the property and instance similarity matrices regarding precision, e.g. PP_{stdev}, and recall, e.g. RP_{stdev}. All predictor correlations are significant according to a two-sample paired t-test with significance level $\alpha = 0.001$. The analysis of the class similarity matrix predictors is not shown since the correlations are not significant. This results from the fact that only 237 tables in the gold standard can be assigned to a DBpedia class and in turn only for those tables we can compute a correlation with precision and recall. However, in practice the predictor P_{herf} shows the most promising results. The same holds for instance similarity matrices where P_{herf} has the highest correlation with precision as well as recall. In contrast, for property similarity matrices, P_{avg} correlates most. One reason is the comparably low amount of properties that can potentially be mapped to one attribute. Within a single matching task, the choice of the best performing predictor is in most cases consistent. One exception is the correlation of P_{herf} to the recall of the matrix generated by the popularity-based matcher since the most popular instances do not necessarily need to be the correct candidates. Based on the results, we use the prediction computed by P_{herf} as weights for the instance as well as for class similarity matrices and P_{avg} for the property similarity matrices in the matching experiments that we report in the next section.

Figure 5 shows the variations of weights for the similarity matrices coming from different matchers. We can see that median of the weights differ for the various matchers which in turn indicates the overall importance of the features across all tables for a certain matching task. For the instance matching task, the popularity of an instance seems to play a crucial role, followed by the label. Contrary, the values build the foundation for the property matching task. The size of the class as well as the amount of instance candidates belonging to one class, used in the frequency-based resp. majority-based matcher, forms the basis of the class matching task. Adding external resources like Wordnet only leads to slight changes of the weights.

Besides the median, the variations of the weights show that the actual utility of a feature depends on the individual matrix and in turn on the table. This supports our assumption that taking the same aggregation weights for all the tables is not always the best strategy. While the weight variations are very large for all matchers operating on attribute labels (attribute label-, wordnet- and dictionary matcher), this is the opposite for the matchers dealing with bag-of-words. A large variation implies that the reliability is predicted differently for various tables which in turn indicates that the attribute label is a suitable feature for some but not for all of the tables. This finding is reasonable since tables can either have attribute labels that perfectly fit to a property label like “capital” while others do not use any meaningful labels. For the bag-of-words matchers, the reliability is estimated quite similar but low for all the tables. Since they compare bag-of-words, they will always find a large amount of candidates.

8. MATCHING RESULTS

In this section, we report the results of our matching experiments and compare them to results from the literature. After applying the different matchers and aggregating their

Table 3: Correlation of matrix predictors to precision and recall

First Line Matcher	PP_{stdev}	RP_{stdev}	PP_{avg}	RP_{avg}	PP_{herf}	RP_{herf}
	Property Similarity Matrices					
Attribute label matcher	**0.474**	0.415	0.433	**0.448**	0.215	0.209
Duplicate-based attribute matcher	0.048	0.094	**0.086**	**0.106**	-0.074	0.042
WordNet matcher	**0.425**	0.341	0.317	**0.367**	0.120	0.178
Dictionary matcher	0.360	0.274	**0.364**	**0.447**	0.130	0.150
mean	**0.327**	0.281	0.300	**0.342**	0.098	0.145
	Instance Similarity Matrices					
Entity label matcher	-0.167	0.092	-0.160	0.049	**0.233**	**0.232**
Value-based entity matcher	0.361	0.496	0.122	0.311	**0.378**	**0.531**
Surface form matcher	-0.291	-0.094	-0.294	-0.128	**0.241**	**0.238**
Popularity-based matcher	0.136	-0.043	0.112	**-0.038**	**0.263**	-0.236
Abstract Matcher	0.047	0.182	0.134	**0.286**	**0.205**	0.152
mean	0.022	0.158	-0.021	0.120	**0.330**	**0.229**

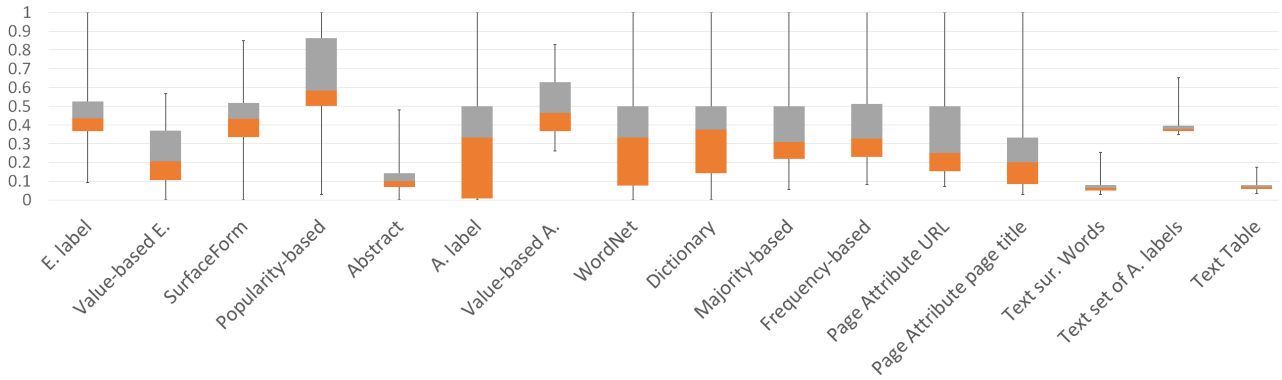

Figure 5: Matrix aggregation weights

similarity scores, we use a $1:1$ decisive second line matcher for generating correspondences. The matcher selects for each entity/attribute/table the candidate with the highest similarity score. This score needs to be above a certain threshold in order to ensure that correspondences are only generated if the matching system is certain enough. The thresholds are determined for each combination of matchers using decision trees and 10-fold-cross-validation. In addition to thresholding, we also apply the filtering rule that we only generate correspondences for a table if (1) a minimum of three entities in the table have a correspondence to an instance in the knowledge base and (2) one forth of the entities in the table is matched to instances of the class we decided for.

We evaluate the matching results according to precision, recall and F1. We compare our results to the results of existing approaches. However, it is tricky to directly compare result as the other systems were tested using different web tables and different knowledge bases and as the difficulty of the matching task is highly dependent on these inputs.

8.1 Row-to-Instance Matching Results

Table 4 presents the results of the row-to-instance matching task for different combinations of matchers. If we only considering the entity label feature, a moderate result with a precision of 0.72 is achieved. Also taking the table cell values into account increases the recall by 0.09 and the precision by 0.08. As expected, based on the weight analysis, considering the values helps to improve the performance but only using the entity label already leads to a decent amount of correct correspondences. By adding surface forms, the recall can again be raised by 0.02 which indicates that we indeed find alternative names for entities in the tables. The popularity-based matcher can slightly increase the precision and recall. Whenever the similarities for candidate instances are close, to decide for the more common one is in most cases the better decision. However, this assumption does especially not hold for web tables containing long-tail entities, e.g. entities that are rather unknown.
Including the only instance matcher relying on a multiple table feature (Abstract matcher), the precision is strongly increased by 0.13 while 0.08 recall is lost. This might be unexpected at first glance since a matcher comparing bag-of-words tends to add a lot of noise. The reason is the choice of the threshold since it needs to be very high to prevent a breakdown of the F1 score. Thus, comparing the entity as a whole with the DBpedia abstracts helps to find correct correspondences but has to be treated with caution to not ruin the overall performance. If we use the combination of all instance matchers, the highest F1 value can be achieved. This shows that the instances found by matchers exploiting different features do not necessarily overlap and that the matchers can benefit from each other by compensating their weaknesses.

In the following, we compare our results to existing results

Table 4: Row-to-instance matching results

Matcher	P	R	F1
Entity label matcher	0.72	0.65	0.68
Entity label matcher + Value-based entity matcher	0.80	0.74	0.77
Surface form matcher + Value-based entity matcher	0.80	0.76	0.78
Entity label matcher + Value-based entity matcher + Popularity-based matcher	0.81	0.76	0.79
Entity label matcher + Value-based entity matcher + Abstract matcher	0.93	0.68	0.79
All	0.92	0.71	0.80

from literature. While Mulwad et al. [26] report an accuracy of 0.66 for a pure label-based instance matching approach, the F1 score achieved by Limaye et al. [22] (web manual data set) is 0.81 when taking alternative names for the labels into account. Extending the label-based method by including the values results in an accuracy of 0.77 [38] resp. a F1 score of 0.82 if the web tables are matched to DBpedia and 0.89 if they are matched against Yago [25]. Very high F1 scores above 0.9 are stated by Zhang [42]. However, the presented baseline that only queries the Freebase API already obtains very close scores such that the good performance is mainly due to the internal API ranking. For other APIs used by the systems, it is not always clear which ranking functions are used and which performance they already achieve without considering any other features.

8.2 Attribute-To-Property Matching Results

Table 5 shows the results of our attribute-to-property matching experiments using different combinations of matchers. In contrast to the row-to-instance matching task, we get a rather low recall (0.49) if we only take the attribute label into account. Based on the weight analysis, we already know that the attribute label is not necessarily a useful feature for all the tables. Including cell values increases the recall by 0.35 but decreases the precision by 0.10. While it provides the possibility to compensate non-similar labels, it also adds incorrect correspondences if values accidentally fit. This especially holds for attributes of data type numeric and date, for example, in a table describing medieval kings it will be quite difficult to distinguish birth dates and death dates by only examining a single attribute at a time. Nevertheless, the values present a valuable feature especially to achieve a decent level of recall, given that the attribute labels are often misleading. Taking WordNet into account does neither improve precision nor recall. This shows, that a general dictionary is not very useful for the property matching task. In contrast, using the dictionary created from web tables increases the recall as well as the precision. With specific background knowledge that is tailored to the web tables, it is possible to enhance the performance. However, the creation of the dictionary requires a lot of smart filtering. Without proper filtering, the dictionary would add only noise. The result of using all matchers together is slightly lower than the best result due to the WordNet matcher.

Our results for the attribute-to-property matching task are difficult to compare to other existing results as many of the existing systems only match attributes to object properties and do not cover data type properties, such as numbers and dates. For this task, Mulwad et al. [26] report an accuracy of 0.25, their advanced system achieves a F1 score of 0.89 [25] while Muñoz et al. [24] report a F1 score of 0.79. Although Limaye et al. [22] additionally include the attribute header, only a result of 0.52 (F1) can be reached. Even without the consideration of data type properties, the property matching task seems to be more difficult than the instance matching task.

8.3 Table-To-Class Matching Results

Table 6 reports the results of our table-to-class matching experiments. Since we need an instance similarity matrix for the class matching, we use the entity label matcher together with the valued-based matcher in all following experiments. When only considering the majority of the instance correspondences to compute the class correspondences, the precision is 0.47 and the recall 0.51, meaning that only for approximately half of the tables the correct class is assigned. One reason for this is the preferential treatment of superclasses over specific classes which are further down in the class hierarchy. All instances that can be found in a specific class are also contained in the superclass and there might be further instances belonging to the superclass that fit. Together with the consideration of the frequency which exactly tackles the mentioned issue, a F1 score of 0.89 can be reached.

In order to see how far we get when solely considering matchers that rely on context features, we evaluate the page attribute matcher and the text matcher independently from the others. Since the differences in the performance are marginal, we do not present the results for the individual features. Whenever the page attribute matcher finds a correspondence, this correspondence is very likely to be correct. However, since the URL and page title are compared with the label of the class, it can happen that no candidate is found at all. Regarding the recall, similar holds for the text matcher but the generated correspondences are not necessarily correct. This is not surprising because we already discovered that matchers using features represented as bag-of-words have a weak ability to differentiate between correct and incorrect candidates due to a lot of noise.

When we combine all previous matchers, a F1 of 0.88 is obtained which is still lower than the outcome of the majority-based together with the frequency-based matcher. If we make use of the number of available class matchers which is transposed by the agreement matcher, we reach a F1 value of 0.92. Thus, taking advantage of features covering the whole spectrum of available information and deciding for the class most of them agree on, is the best strategy for the class matching task.

Due to the fact that the table-to-class matching task has a strong influence on the other two matching tasks in $T2KMatch$, their performance can be substantially reduced

Table 5: Attribute-to-property matching results

Matcher	P	R	F1
Attribute label matcher	0.85	0.49	0.63
Attribute label matcher + Duplicate-based attribute matcher	0.75	0.84	0.79
WordNet matcher + Duplicate-based attribute matcher	0.71	0.83	0.77
Dictionary matcher + Duplicate-based attribute matcher	0.76	0.86	0.81
All	0.70	0.84	0.77

Table 6: Table-to-class matching results

Matcher	P	R	F1
Majority-based matcher	0.47	0.51	0.49
Majority-based matcher + Frequency-based matcher	0.87	0.90	0.89
Page attribute matcher	0.97	0.37	0.53
Text matcher	0.75	0.34	0.46
Page attribute matcher + Text matcher + Majority-based matcher + Frequency-based matcher	0.9	0.86	0.88
All	0.93	0.91	0.92

whenever a wrong class decision is taken. For example, when solely using the text matcher, the row-to-instance recall drops down to 0.52 and the attribute-to-property recall to 0.36.

For the table-to-class matching task, results between 0.43 (F1) [22] and 0.9 (accuracy) [38] are reported in the literature. In between, we find outcomes varying from 0.55 (F1) [43] over 0.65 to 0.7 for different knowledge bases [39]. When taking also the specificity of the classes into account, the performance of 0.57 (F1) is neither higher nor lower than other results [25]. Similar holds for considering the context with a result of 0.63 (F1) [42].

In summary, our matching system is able to distinguish between tables that can be matched to DBpedia and tables that do not have any counterparts. This becomes especially obvious if we look at the results of the table-to-class matching task. The ability to properly recognize which tables can be matched is a very important characteristic when dealing with web tables. Whenever the table can be matched to DBpedia, features directly found in the table are crucial for the instance and property matching tasks. For properties, the cell values need to be exploited in order to achieve an acceptable recall. Adding external resources is useful the closer the content of the external resource is related to the web tables or to the knowledge base. For the table-to-class matching task, the majority of instances as well as the specificity of a class has a very high impact on the performance. While matchers based on page attributes often do not find a correspondence at all, this is the opposite for all features represented as bag-of-words which add a large amount of noise. Ways to handle the noise are either filtering or to only use them as an additional indicator whenever matchers based on other features agree with the decision.

Comparing the related work among each other shows that almost no conclusions can be drawn whether a certain feature is useful for a matching task or not. One reason for this is that the systems are applied to different sets of web tables and different knowledge bases. As indicated by Hassanzadeh et al. [16], the choice of the knowledge base has a strong influence on the matching results. For example, a knowledge base might not contain certain instances (e.g. web tables contain a lot of product data while DBpedia hardly covers products) or properties at all (e.g. product prices) and the granularity of the classes can differ a lot, depending on the structure and the focus of the knowledge base [16, 31].

9. CONCLUSION

This paper studied the utility of different features for task of matching web tables against a knowledge base. We provided an overview as well as a classification of the features used in state-of-the-art systems. The features can either be found in the table itself or in the context of the table. For each of the features, we introduce task specific matchers that compute similarities to instances, properties, and classes in a knowledge base. The resulting similarity matrices, representing the feature-specific results, have been combined using matrix predictors in order to gain insights about the suitability of the aggregation weights. Using matrix predictors, we allow different web tables to favor the features that are most suitable for them.

We showed that a positive correlation between the weighting based on the reliability scores as well as the performance measures precision and recall can be found. However, the best way to compute reliability scores differs depending on the matching task. While predictors based on the diversity of the matrix elements work best for the row-to-instance and table-to-class matching task, an average-based predictor shows a better performance for the attribute-to-property matching task.

The computed weights gave us an idea which features are in general important for the individual matching tasks and how much their significance varies between the tables. While the entity label and the popularity of an instance are very important for the row-to-instance matching task, comparing the cell values is crucial for the attribute-to-property matching task. For the table-to-class matching task, several features are important, while the ones directly coming from the table outperform context features. The largest variation in the weights was discovered for the attribute labels. This

indicates that attribute labels can be a good feature as long as meaningful attribute names are found in the web tables but also that this is not always the case.

We further explored the performance of different ensembles of matchers for all three matching tasks. In summary, taking as many features as possible into account is promising for all three tasks. Features found within tables generally lead to the best results than context features. Nevertheless, taking context features into account can improve the results but particular caution is necessary since context features may also add a lot of noise. External resources proofed to useful as long as their content is closely related to the content of the web tables, i.e. the general lexical database WordNet did not improve the results for the attribute-to-property matching task while a more specific dictionary did improve the results. The performance that we achieved in our experiments for the row-to-instance and the attribute-to-property matching tasks are roughly in the same range as the results reported in literature. For the table-to-class matching task, our results are higher than the ones reported in the related work.

The source code of the extended version of the *T2KMatch* matching framework that was used for the experiments is found on the *T2K* website[4]. The gold standard that was used for the experiments can be downloaded from the Web Data Commons website[5].

10. REFERENCES

[1] K. Braunschweig, M. Thiele, J. Eberius, and W. Lehner. Column-specific Context Extraction for Web Tables. In *Proc. of the 30th Annual ACM Symposium on Applied Computing*, pages 1072–1077, 2015.

[2] V. Bryl, C. Bizer, and H. Paulheim. Gathering alternative surface forms for dbpedia entities. In *Proceedings of the Third NLP&DBpedia Workshop (NLP & DBpedia 2015)*, pages 13–24, 2015.

[3] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data Integration for the Relational Web. *Proc. of the VLDB Endow.*, 2:1090–1101, 2009.

[4] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: Exploring the Power of Tables on the Web. *Proc. of the VLDB Endow.*, 1:538–549, 2008.

[5] I. F. Cruz, F. P. Antonelli, and C. Stroe. Efficient selection of mappings and automatic quality-driven combination of matching methods. In *Proc. of the 4th Int. Workshop on Ontology Matching*, 2009.

[6] I. F. Cruz, V. R. Ganesh, and S. I. Mirrezaei. Semantic Extraction of Geographic Data from Web Tables for Big Data Integration. In *Proc. of the 7th Workshop on Geographic Information Retrieval*, pages 19–26, 2013.

[7] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proc. of the 9th Int. Conference on Semantic Systems*, 2013.

[8] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding Related Tables. In *Proc. of the Int. Conference on Management of Data*, 2012.

[9] H.-H. Do and E. Rahm. COMA: A System for Flexible Combination of Schema Matching Approaches. In *Proc. of the 28th Int. Conference on Very Large Data Bases*, pages 610–621, 2002.

[10] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In *Proc. of the 20th SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 601–610, 2014.

[11] J. Euzenat and P. Shvaiko. *Ontology Matching*. Springer, 2007.

[12] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[13] A. Gal, H. Roitman, and T. Sagi. From Diversity-based Prediction to Better Ontology & Schema Matching. In *Proc. of the 25th Int. World Wide Web Conference*, 2016.

[14] A. Gal. *Uncertain Schema Matching*. Synthesis Lectures on Data Management. Morgan & Claypool, 2011.

[15] A. Gal and T. Sagi. Tuning the ensemble selection process of schema matchers. *Information Systems*, 35(8):845 – 859, 2010.

[16] 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 *Proc. of the 10th Int. Workshop on Ontology Matching*, 2015.

[17] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.

[18] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015.

[19] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer. A Large Public Corpus of Web Tables containing Time and Context Metadata. In *Proc. of the 25th International World Wide Web Conference*, 2016.

[20] O. Lehmberg and C. Bizer. Web table column categorisation and profiling. In *Proc. of the 19th International Workshop on Web and Databases*, pages 4:1–4:7, 2016.

[21] O. Lehmberg, D. Ritze, P. Ristoski, R. Meusel, H. Paulheim, and C. Bizer. The Mannheim Search Join Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 35:159–166, 2015.

[22] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc.of the VLDB Endow.*, 3:1338–1347, 2010.

[23] X. Ling, A. Halevy, F. Wu, and C. Yu. Synthesizing union tables from the web. In *Proc. of the 23rd Int. Joint Conference on Artificial Intelligence*, pages 2677–2683, 2013.

[24] E. Muñoz, A. Hogan, and A. Mileo. Using Linked Data

[4] http://dws.informatik.uni-mannheim.de/en/research/T2K

[5] http://webdatacommons.org/webtables/goldstandardV2.html

to Mine RDF from Wikipedia's Tables. In *Proc. of the 7th ACM Int. Conference on Web Search and Data Mining*, pages 533–542, 2014.

[25] V. Mulwad, T. Finin, and A. Joshi. Semantic message passing for generating linked data from tables. In *Proc. of the 12th Int. Semantic Web Conference*, 2013.

[26] V. Mulwad, T. Finin, Z. Syed, and A. Joshi. Using linked data to interpret tables. In *Proc. of the 1st Int. Workshop on Consuming Linked Data*, 2010.

[27] S. Neumaier, J. Umbrich, J. X. Parreira, and A. Polleres. Multi-level semantic labelling of numerical values. In *Proc. of the 15th International Semantic Web Conference*, pages 428–445, 2016.

[28] K. Pearson. Notes on regression and inheritance in the case of two parents. *Proc. of the Royal Society of London*, 58:240–242, 1895.

[29] S. Rhoades. The Herfindahl-Herschman Index. *Federal Reserve Bulletin*, 79:188–189, 1993.

[30] D. Rinser, D. Lange, and F. Naumann. Cross-Lingual Entity Matching and Infobox Alignment in Wikipedia. *Information Systems*, 38:887–907, 2013.

[31] D. Ritze, O. Lehmberg, Y. Oulabi, and C. Bizer. Profiling the Potential of Web Tables for Augmenting Cross-domain Knowledge Bases. In *Proc. of the 25th International World Wide Web Conference*, 2016.

[32] D. Ritze, O. Lehmberg, and C. Bizer. Matching HTML Tables to DBpedia. In *Proc. of the 5th International Conference on Web Intelligence, Mining and Semantics*, 2015.

[33] T. Sagi and A. Gal. Schema matching prediction with applications to data source discovery and dynamic ensembling. *VLDB Journal*, 22:689–710, 2013.

[34] G. Salton and M. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1983.

[35] Y. A. Sekhavat, F. di Paolo, D. Barbosa, and P. Merialdo. Knowledge Base Augmentation using Tabular Data. In *Proc. of the 7th Workshop on Linked Data on the Web*, 2014.

[36] A. Singhal. Introducing the knowledge graph: Things, not string. Blog, 2012. Retrieved March 19, 2015.

[37] F. Suchanek, S. Abiteboul, and P. Senellart. Paris: Probabilistic alignment of Relations, Instances, and Schema. *Proc. VLDB Endowment*, 5:157–168, 2011.

[38] Z. Syed, T. Finin, V. Mulwad, and A. Joshi. Exploiting a Web of Semantic Data for Interpreting Tables. In *Proc. of the 2nd Web Science Conference*, 2010.

[39] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering Semantics of Tables on the Web. *Proc. of the VLDB Endow.*, 4(9):528–538, 2011.

[40] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu. Understanding Tables on the Web. In *Proc. of the 31st Int. Conf. on Conceptual Modeling*, pages 141–155, 2012.

[41] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *Proc. of the 2012 SIGMOD*, pages 97–108, 2012.

[42] Z. Zhang. Towards efficient and effective semantic table interpretation. In *Proc. of the 13th International Semantic Web Conference*, pages 487–502. 2014.

[43] S. Zwicklbauer, C. Einsiedler, M. Granitzer, and C. Seifert. Towards disambiguating Web tables. In *Proc. of the 12th Int. Semantic Web Conference*, pages 205–208, 2013.