Annotating Web Tables through Knowledge Bases: A Context-Based Approach

Yasamin Eslahi
Zurich University of Applied Sciences
Winterthur, Switzerland
yasamin.eslahi@zhaw.ch

Akansha Bhardwaj
University of Fribourg
Fribourg, Switzerland
akansha.bhardwaj@unifr.ch

Paolo Rosso
University of Fribourg
Fribourg, Switzerland
paolo.rosso@unifr.ch

Kurt Stockinger
Zurich University of Applied Sciences
Winterthur, Switzerland
kurt.stockinger@zhaw.ch

Philippe Cudré-Mauroux
University of Fribourg
Fribourg, Switzerland
philippe.cudre-mauroux@unifr.ch

***Abstract*—The Web has a collection of over 150 million tables, which as a whole represents an invaluable source of semi-structured knowledge. Such tables are commonly referred to as *Web tables*, and are considerably easier to leverage in automated processes than completely unstructured, free-format text. Understanding the semantics of Web tables is important since they are used in various applications like knowledge base augmentation, information retrieval or natural language interfaces for databases. The task of understanding the semantics of a given Web table is known as Web table annotation. In recent years, it has been tackled through methods where the table is enriched using existing knowledge bases containing valuable information on the domain at hand, its entities and their mutual relationships.**

In this paper, we present two novel and unsupervised *Web table annotation* methods, which leverage the context of the tables to better capture their semantics. Our first method is lookup-based and exploits text similarity to find reference entities in the knowledge base. The second method uses distributional vector representations – a.k.a. embeddings – of the Web tables to elicit their context and disambiguate their semantics. Experiments show that our proposed approach outperforms the state of the art in Web table annotation by up to 18%. Another contribution of this work is a manually corrected version of one of the popular gold standard datasets, Limaye, with annotations from DBpedia. Our dataset and code are publicly available[1].

***Index Terms*—Web Table Annotation, Knowledge Base, Embeddings**

I. Introduction

Web tables are a valuable source of semi-structured data. As the Web contains an estimated 154 million HTML tables, understanding the semantics of such data has become an active area of research [2]. The task of understanding the semantics of Web tables is typically accomplished leveraging knowledge bases (KBs). KBs contain rich information about entities, their semantic classes, and their mutual relationships in a given domain. Each entity of the Web table gets a unique reference entity in a KB. Examples of encyclopedic KBs include DBpedia [2], Wikidata [3], or Yago [4].

Figure 1a contains a description of countries, as they are often found in a Web table. The aim of *Web table annotation* is to annotate entries in the Web table to their referent entities in the knowledge base. Annotations can be at the level of a cell value—referred to as instance-level mappings—or at the level of the table—referred to as schema-level mappings. Such annotations can then be leveraged to understand the underlying semantics of the Web tables and to foster interoperability and automated processes across syntactically heterogeneous but semantically related tables, e.g., for Web tables search or knowledge base augmentation. To elaborate on our example in Figure 1a, a number of entities are present in the table, like countries (*France, Germany, India* etc.) in column ‘Country’. Our aim is to map those table values representing entities onto their counterpart in a knowledge base (Wikidata, in our case).

Web table annotation presents several key challenges:

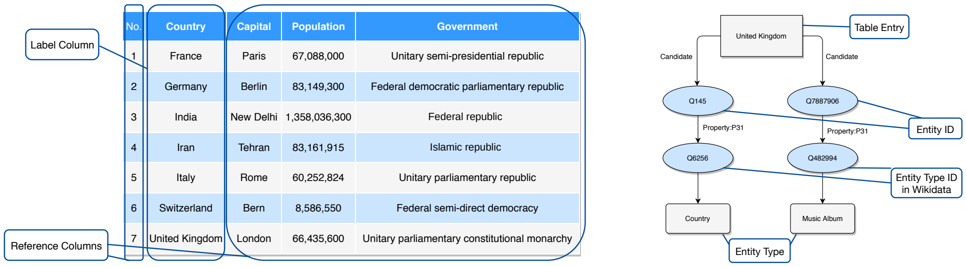
1) The presence of a header row giving a short description (e.g., “No., Country, Capital”) of the various values is not always guaranteed. Typically, each row in the table describes a real-world entity (e.g., in Figure 1a the seventh row describes the United Kingdom), while each column contains the value of the corresponding property, e.g., (“Population”, “66,435,600”), (“Capital”, “London”). The knowledge base often contains information related to these entities and values. For example, United Kingdom is described by node Q145, which is of type country in Wikidata. Hence, it is essential to understand the overall theme and context of a given Web table for correctly matching its entities and values onto the KB.
2) Another important challenge is the disambiguation of polysemous phrases. For example, in Figure 1b, the

[1] https://www.github.com/eXascaleInfolab/sds2020_web_table_annotation

[2] http://www.dbpedia.org/

[3] https://www.wikidata.org/

[4] https://www.mpi-inf.mpg.de/yago-naga/yago/



No.	Country	Capital	Population	Government
1	France	Paris	67,088,000	Unitary semi-presidential republic
2	Germany	Berlin	83,149,300	Federal democratic parliamentary republic
3	India	New Delhi	1,358,036,300	Federal republic
4	Iran	Tehran	83,161,915	Islamic republic
5	Italy	Rome	60,252,824	Unitary parliamentary republic
6	Switzerland	Bern	8,586,550	Federal semi-direct democracy
7	United Kingdom	London	66,435,600	Unitary parliamentary constitutional monarchy

(a) Table of Countries.

(b) Disambiguation of a cell.

Fig. 1: (a) An example of a Web table containing information about countries, their capital, population and type of government. The goal of a Web table annotation task is to annotate each entity of the Web table with its reference entity in a knowledge base such as Wikidata. (b) A common challenge in correctly annotating Web tables is capturing polysemous phrases. This example shows that *United Kingdom* has two possible matches in the KB. It could be a country or an album.

entity *United Kingdom* could be considered as a country or as an album. Another challenge along similar lines is that an entity can also have different names. For instance, the bird *Green sandpiper*, is also known as *Tringa ochropus*.

3) The third challenge is that an entity's name may change over time [5]. If the entity's name in the Web table is different from its equivalent page title in the knowledge base, then it becomes more difficult to annotate it.

As it is clear from the above challenges, the context of a Web table plays an important role in correctly characterizing it. In this work, we present two novel Web table annotation methods that leverage context. Our first method is lookup-based and relies on information matching between labels appearing in Web tables and information stored in KB, while our second method exploits the semantic understanding of the table through embeddings. Specifically, we focus on the problem of Web table annotation [3], [4] (also known as *interpretation* in the literature) [12], [15], the task of mapping a phrase in a Web table to its reference entity in a KB. The purpose of this task is to understand a Web table semantically and to integrate it into a KB such as to enable downstream applications like Web table search, KB augmentation or natural language interfaces to databases. It is important to mention that we focus on instance-level matching only in this work. We leave the utilization of schema-level matching techniques for future work.

In summary, our contributions are as follows:

1) We propose two novel global disambiguation methods for *Web table annotation*. Our proposed *Looping* approach that aims to understand the table semantically outperforms the state of the art by 14% on two gold standard datasets.
2) We present a manually corrected version of the Limaye dataset for the *Web table annotation* task and make our code and data publicly available.

Our paper is structured as follows. Section II describes existing approaches for *Web table* annotation. In Section III, we introduce our approaches. Section IV-A and IV-B describe our dataset and experimental results, respectively. Finally, we conclude and present avenues for future work in Section V.

II. Related Work

Web tables often contain a subject column, also called *label* column, containing labels describing the subject of the table. The other columns, also called *reference* columns, typically represent binary relationships [6]. Supervised Web table techniques require columns and/or rows to be annotated. To annotate a Web table, rows, columns, or individual cells can be mapped onto a KB, for example using a framework to link entities lists onto a KB [9], or by mapping table columns to one or more concepts in the KB [10]. Though supervised methods [4], [9], [11] have shown promising results on Web annotation tasks, they require training datasets of annotated tables.

[5], [12] describe two unsupervised solutions that proceed without annotating the training dataset. Specifically, [12] introduces a method to label columns containing named entity mentions with semantic concepts that best describe the data in the respective column. [5] proposes a heuristic method considering that the leftmost column with the maximum number of distinct and non-numeric values contains the most important attribute of an entity. The solution is based on two techniques: *Lookup-Baseline* and *Embedding-Baseline*. The *Lookup-Baseline* technique is used to match the information from entities appearing in the Web tables to the information of those entities in a KB using a lookup service on the target KB, while the *Embedding-Baseline* technique is used to capture the structure of the neighborhood of each node in the KB. Our proposed method is motivated by their work, which is the state of the art in unsupervised Web table annotation. In the following subsections, we describe in detail the *Lookup-Baseline* and *Embedding-Baseline* approaches.

A. Lookup-Baseline Method

The original *Lookup-Baseline* method [5] is divided into two phases. In the first phase, it extracts the value of each row of the Label Column. This value is the Web table entry that needs to be annotated. Next, the method searches for the entity's candidates in the surface form. The surface form is a collection of key-value pairs, where the key is a label and the values are Wikidata identifiers. For example, for the label *New York City*, the surface form outputs the Wikidata identifiers *Q7013143, Q60, Q3875477*. If, for a Web table entry there exists only one candidate in the surface form, then the Web table entry is annotated with the surface form value. If the Web table entry has more than one candidate, the Lookup-Baseline method filters the candidates based on their entity type. If the candidate entity type belongs to top-3 frequent entity types, it is considered *acceptable*. Additionally, for each *acceptable* candidate, it extracts the most frequent tokens from their description. This is referred to as *strict search*. If the *strict search* does not generate any output or if there are no candidates, the method performs a *loose search*. This is done by selecting one of the extracted binary relations between the label and reference columns.

B. Embedding-Baseline Method

The problem with local disambiguation techniques is that they do not keep track of the context. For example, in a sentence such as *'Sydney cannot be interesting for the kids'*, the embedding method does not know whether *Sydney* is a city in Australia (Q3130), a community in Canada (Q932261), or an American comedy series (Q3979019), since it is not aware of the context around the sentence. In the field of text disambiguation, this issue is mitigated by learning embeddings from the KB. The previous state of the art [5] is inspired by DoSeR [13], where the embeddings are learned using Word2vec [14], a group of related models that are used to produce word embeddings for text. Though Word2vec is mostly used on text data, DoSeR proposes an approach to apply Word2vec on KBs showing good results in terms of scalability and performance. These learnt embeddings are further used to create a K-partite graph as described in [5], where the weight of the edges is the cosine similarity of the entity embeddings. The importance of the nodes is determined using the weighted PageRank [13] ranking algorithm. Candidate nodes with a high PageRank are an indicator of the correct entity as they are closer to the global context of the table.

The *Lookup-Baseline* method presented above uses several KB-dependent heuristics. Note that the *Embedding-Baseline* method that uses global context does not perform at par with *Lookup-Baseline*. In this work, we propose two novel methods for the Web table annotation task, namely *Context-Lookup* (as alternative to *Lookup-Baseline*), and *Looping* (as alternative to *Embedding-Baseline*). Our *Context-Lookup* method uses the context extracted from the whole Web table, while our *Looping* method is an iterative approach that builds a graph used to disambiguate entities with multiple candidates. We describe the details of our proposed approaches in the following section.

III. Methodology

In this section, we describe two methods for solving the *Web table annotation* task. Subsequently, we propose solutions that use their combination.

Algorithm 1: Context-Lookup

```
Input: Table $\mathcal{T}$
Output: Annotated Table $\mathcal{T}'$
$\mathcal{T}' \leftarrow \mathcal{T}$;
allTypes $\leftarrow \phi$
candidateRelations $\leftarrow \phi$
for each row $i$ of $T$ do
    label $\leftarrow T.i.labelColumn$
    candidates $\leftarrow sf\_search(label)$
    if candidates.size $> 0$ then
        allTypes.addAll(candidates.getTypes())
        descr $\leftarrow candidates.getDescriptionTokens()$
        if candidates.size $= 1$ then
            annotate($T'.i$ , candidates)
            for each column $j$ of referenceColumns do
                $v \leftarrow$ T.i.j
                lab_ref_rel $\leftarrow getRelation$(T'.i, v)
                candidateRelations.add(lab_ref_rel, $j$)
acceptableTypes $\leftarrow allTypes.get3MajorTypes()$
freqTok $\leftarrow descr.getMostFrequent()$
rel $\leftarrow candidateRelations.getRel(referenceColumns)$;
for each row $i$ of $T$ do
    if isAnnotated(T'.i) then continue
    label $\leftarrow T.i.labelColumn$
    cand $\leftarrow$search_strict($label, acceptableTypes, freqTok$)
    if cand.size $> 0$ then
        annotate($T'.i$, cand.getFirst())
        continue
    for $j$ in $relations$ do
        r $\leftarrow relations.j$
        results $\leftarrow$search_loose($label, r,$ T.i.j)
        if results.size $> 0$
            annotate($T'.i$ , results.getFirst())
```

A. Context Lookup

The *Lookup-Baseline* method presented by [5] misses a significant portion of the available information. Specifically, their method generates a list of candidates from the surface form but they only consider the entity types of the first element of the list for the annotation. Instead, we propose a systematic majority-based method to benefit from all the candidate entity types from the list, and not just the first element. Intuitively by doing this, we consider the entire context of the table and filter the correct annotations using a majority function. We present our method in Algorithm 1. Elaborating further, after creating the list of candidates, we select the entity types from all the candidates of the whole table. Finally, by a majority function, we choose the top-n most frequent entity types among them. This means that we take other potential candidates into account. Specifically, we define the acceptable type as the three most frequent entity types among all collected entity types. For each entry of a Web table that has only one candidate generated from the surface form, we annotate the entry using that candidate. In case an entry of a Web table has more candidates, we perform a strict search or a loose search similar to [5], but using the acceptable type generated by collecting all the entity types from the whole Web table.

B. Looping Method

Our second technique is based on the global disambiguation technique presented in DoSeR [13], where the similarity between entities is computed as the cosine similarity between their vector representations. These vectors, called embeddings, capture the semantic correlation between entities in the KB. This implies that if the entities are close to each other with respect to their cosine similarity distance, they are also semantically related. However, the embedding-based global disambiguation technique presented in previous work [5] suffers from a major drawback. The proposed weighted graph grows exponentially with the number of candidates, which affects the results adversely. To counter this drawback, we propose an iterative method of *Web table annotation* based on embeddings. Our novel iterative approach gives more weight to the entries with unique candidates that results in much better performance. We explain this further.

Party	05-02-08
Socialist party	60
Social liberal party	92
Christian democrats	17
Danish people's party	133

(a) Parties of Denmark.

Name of party	Seats
the moderate party	4
Christian democrats	1
the green party	1
Sweden democrats	- -

(b) Parties of Sweden.

TABLE I: A Web table of the political parties of (a) Denmark (b) Sweden. A challenging example from T2D dataset, marked in blue, where labels (Christian democrats) and entity types (political party Q7278) are the same but the corresponding reference entities in the KB are different (the Wikidata identifiers of *Christian democrats of Denmark* is Q1789199 and of *Christian democrats of Sweden* is Q213654). Green labels explicitly show the context of the tables.

First, we train a Word2vec model [14] for all the candidate entities and store their vector representations. Next, for each table, we build an initial graph with all unambiguous candidates from the Web table. These are those entities which have a unique candidate. Through our experiments, we find that building a correct initial graph is crucial to the disambiguation of entities with multiple candidates. This can be explained using examples in Table Ia and Ib. Table Ia contains information about political parties in Denmark while Table Ib contains information about political parties in Sweden.

There are several political parties which are polysemous (e.g., *Christian democrats*) and they belong to the same entity type (e.g., *political party*). Our experiments show that in similar cases, building an initial graph with few but correctly annotated entities extracts the right context out of the table from the beginning and helps in the proper annotation of the remaining entities.

Furthermore, at each iteration of the Looping method, we take the initial graph and one ambiguous entity. For k candidates of the ambiguous entity, we build a k-partite graph where the nodes of the graph are the annotated nodes, and all possible candidates of the ambiguous node. For each pair of nodes, we compute the weight of the edge by Equation 1.

$$weight(v1, v2) = \frac{cos(emb(v1), emb(v2))}{\sum_k cos(emb(v1), emb(k))} \qquad (1)$$

An example of such a scenario is presented in Figure 2. In this graph, all the nodes are already annotated except *United Kingdom*. For the *United Kingdom* entity, we have two candidates, a country (UK Country) and a music album (UK Album). We run the *PageRank* ranking algorithm on this undirected graph to generate the PageRank score for each candidate node. The candidate with the highest score is the most probably correct annotation. This is based on the intuition that a higher score for a node from the PageRank ranking algorithm denotes a stronger relationship of this node with other nodes in the graph, which is interpreted as a high likelihood for this node to be closer to the *context* of the graph. This likely correct annotation is added to the initial graph and the above process is repeated for each remaining ambiguous node.

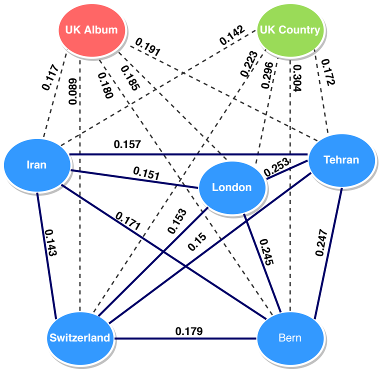


Fig. 2: An example of a looping graph where UK has two possible candidates, type:album (shown in red) and type:country (shown in green). In this case, a 2-partite graph is generated connecting the candidates with the disambiguated entities (shown in blue). The weight of the edges is the cosine similarity between the vector representation of the nodes. A weighted ranking algorithm, e.g. PageRank, helps identify the correct candidate (type:country).

In Figure 2, the node with type country has a higher score compared to the node with type music album signifying a stronger relationship of type:country with the context of the table. Hence, we consider the country (Q145) as annotation of *United Kingdom*. For the next iteration, *United Kingdom* (Q145) is already annotated and therefore, it exists in the base graph. Subsequently, we add the candidates of other ambiguous entities to the looping graph. We continue and annotate one entity in each loop to cover all the entities of the table.

To avoid scalability issues on large graphs, we take only two Web table columns at a time. We make some further refinements in our proposed *Looping* method by leveraging type checking. If we get the same ranking output for multiple candidates, the candidate with the most similar type to the looping graph type is selected. We also study the impact of other ranking methods on our *Looping* algorithm. These experiments and results are detailed in Section IV-C

So far, we presented two distinct methods that can be independently plugged into existing Web table annotation frameworks. Next, we present experimental results based on our methods and their combinations.

IV. Experiments & Results

A. Datasets

We use two publicly available[5]gold standard datasets, T2D[6] and Limaye [4] to empirically validate our *Web table annotation* approach.

The T2D gold standard dataset contains HTML tables for evaluating *Web table annotation* methods. For our task, we use the entity-level gold standard that contains 26,124 rows from a subset of 233 tables. These rows were originally annotated with DBpedia entities.

The Limaye dataset proposed by [5] contains 5,278 annotated rows from a subset of 296 tables mapped to the October 2015 version of DBpedia. We manually corrected some of the labeled column entities as they were wrong. For example, the table entry *Roller Skates* was annotated with the DBpedia URI *resource/NULL* and we replaced it with the DBpedia URI *page/Roller_Skates*. After fixing the DBpedia URIs, we replaced them with Wikidata URIs as Wikidata KB is more up-to-date than DBpedia. Our corrected version of the Limaye dataset along with the corresponding Wikidata annotations, containing 8,400 annotated rows from a subset of 296 tables mapped, is publicly available. We also provide the previous mappings from DBpedia and Wikipedia as reference.

B. Experimental Setup

We present the results of various Web table annotation methods and report the precision, recall, and F-score (reported as Pr, Re, and F1 in the result tables) of the annotation algorithms. We also compare our results to the previous baseline presented by [5].

We implement the *Lookup-Baseline* algorithm [5] using Wikidata and compare it against our *Context-Lookup* approach presented in Section III-A. Table II shows that *Lookup-Baseline* has better results than our *Context-Lookup* on T2D, while our approach outperforms it on Limaye.

T2D dataset has a significant number of entries with unique candidates in the surface form. These entries are essential for achieving better performance as in the next phase, we choose acceptable types based on these values. On the other hand, in the Limaye dataset, we have more difficulty in finding any unique candidates, because in several examples, the cell value in the Web table is different from the entity name in the KB. As a result, the candidate is not accessible using the key in the surface form index. However, our approach achieves better results on Limaye, meaning that our majority function supports the algorithm to annotate those cases where we generate multiple candidates from the surface form. This shows that our approach is more robust to ambiguity.

TABLE II: Results of lookup-based Web table annotation approaches over the T2D and Limaye gold standard datasets.

Method	**T2D**			**Limaye**		
	Pr	***Re***	***F1***	***Pr***	***Re***	***F1***
Lookup-Baseline	0.8784	0.7814	**0.827**	0.788	0.834	0.81
Context-Lookup	0.897	0.72	0.799	0.82	0.82	**0.82**

TABLE III: Results of semantic embedding approaches over T2D and Limaye gold standards

Method	**T2D**			**Limaye**		
	Pr	***Re***	***F1***	***Pr***	***Re***	***F1***
Embedding-Baseline	0.62	0.70	0.66	0.76	0.82	0.79
Looping	0.86	0.82	**0.84**	0.82	0.87	**0.85**

In Table III, we compare the results of our *Looping* method with the baseline (i.e., *Embedding-Baseline*) using PageRank as the ranking algorithm. For the T2D dataset, the Looping method has a significantly better precision of 24% than the *Embedding-Baseline* method and a 18% better F-measure. The results of the Looping method on the Limaye gold standard are 6% better than the *Embedding-Baseline* method in terms of precision and F-measure.

C. Looping Optimization

Our Looping approach presented in the previous section performs better on Web annotation than the state-of-the-art *Embeddings-Baseline* method. When running Looping, we further consider several optimization strategies that we briefly describe below.

a) Initial Levenshtein: For constructing the initial graph in looping, we use two distinct approaches. In the first approach we create the initial graph using the Levenshtein distance between the table entry and the surface form key. In the second approach we create the initial graph without Levenshtein. Table IV presents the performance difference when the Levenshtein distance was used for building the initial graph compared to when it was not. We notice a significant 12% difference in the F1-score.

TABLE IV: Results of using Looping with vs. without Levenshtein in the initial graph

Method	***Pr***	***Re***	***F1***
Looping- Initial Levenshtein	0.67	0.86	0.70
Looping- Without initial Levenshtein	0.84	0.80	**0.82**

b) Ranking Algorithms: In Table III we presented the results for both the *Embedding-Baseline* method and our proposed *Looping* method using PageRank. Now we study the impact of different ranking methods in terms of runtime, since graph ranking methods are often computationally expensive. Table V presents the results of our method Looping with

[5] http://www.cs.toronto.edu/~oktie/webtables/

[6] http://webdatacommons.org/webtables/goldstandard.html

PageRank, EigenVector, and Katz ranking algorithms. Over the Limaye dataset, changing the ranking algorithm does not significantly affect the runtime. Over the T2D dataset, the fastest algorithm is EigenVector while over Limaye, PageRank is faster. Katz has the longest runtime over both gold standards. On the other hand, there was a significant difference in runtime of Katz in comparison to the two other methods over the T2D gold standard.

TABLE V: Run-time comparison of Looping approaches with different ranking algorithms on the T2D and Limaye gold standard datasets

Method	***T2D***	***Limaye***
PageRank Looping	12.1h	**7.28h**
EigenVector Looping	**11.2h**	7.41h
Katz Looping	117h	7.79

c) Hybrid Optimizations: As mentioned above, our two methods are independent of each other and can be used in combination as well. We adopt an optimization approach to increase our performance in the following way. In the first phase, we choose one of the semantic embedding methods, and then we enrich the results by leveraging the lookup-based methods during the second phase. For each phase, we choose the methods with the best-reported results. Table VI shows that there is a remarkable improvement in F1-scores using this approach. The F1-score resulting from using the Katz and EigenVector ranking algorithms are better than our best results achieved so far.

Table VI shows that using *Looping* in the first phase and augmenting it with *Lookup-Baseline* in the second phase yields the best performance. This implies that *Lookup-Baseline* finds correct annotations for the entities where *Looping* could not provide reasonable results. The most striking point is that our *Looping* method (using Eigenvector ranking) combined with *Lookup-Baseline* yields the best results we could achieve from all of the presented methods, and this on both datasets.

TABLE VI: Results of combining approaches over T2D and Limaye gold standards

T2D			
Method	***Pr***	***Re***	***F1***
Embedding-Baseline (PageRank) + Lookup	0.67	0.78	0.72
Looping (PageRank) + Context-Lookup	0.84	0.81	0.826
Looping (Eigenvector) + Lookup	0.854	0.837	**0.846**
Looping (Katz) + Lookup	0.85	0.84	0.845
Limaye			
Method	***Pr***	***Re***	***F1***
Embedding-Baseline (PageRank) + Lookup	0.77	0.86	0.82
Looping (PageRank) + Context-Lookup	0.78	0.85	0.813
Looping (EigenVector) + Lookup	0.823	0.87	**0.847**
Looping (Katz) + Lookup	0.822	0.872	**0.847**

V. Conclusion & Future Works

In this work, we presented two new techniques that perform Web table annotation, namely *Context-Lookup* and *Looping*. More specifically, the *Context-Lookup* method uses the types and relations in the Web table to select the best candidate for annotation. The *Looping* method creates weighted graphs in order to find the best candidate mapping between an entity in the Web table and the corresponding entity in the KB. This weighted graph uses the similarity between entities to find the most related nodes in the context of the Web table. We observe that our proposed approach exceeds the state of the art in Web table annotation methods by up to 18%. For future work, we plan to improve our surface form to generate initial candidates for disambiguating matches between the Web table and the KB. Additionally, we also plan to extend our proposed method to schema-level matching.

Acknowledgment

This work was supported by the Swiss National Science Foundation under grant number 407540_167320. We would also like to thank Michael Luggen and Yann Vonlanthen for their valuable insights.

References

[1] K. Affolter, K. Stockinger, A. Bernstein, ”A comparative survey of recent natural language interfaces for databases” , VLDB J. 28(5): 793-819 (2019)

[2] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, “WebTables: Exploring the Power of Tableson the Web,” Proc. VLDB Endow., vol. 1, pp. 538–549, Aug. 2008.

[3] 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,” OM, 2015.

[4] G. Limaye, S. Sarawagi, and S. Chakrabarti, “Annotating and Searching Web Tables Using Entities, Types and Relationships,” PVLDB, vol. 3, pp. 1338-1347, 2010.

[5] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, and V. Christophides, “Matching web tables with knowledge base entities: From entity lookups to entity embeddings”, in The Semantic Web – ISWC 2017. Springer International Publishing, 2017, pp. 260–277.

[6] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu, “Recovering semantics of tables on the web,” PVLDB, vol. 4, no. 9, pp. 528–538, 2011.

[7] S. Balakrishnan, A. Y. Halevy, B. Harb, H. Lee, J. Madhavan, A. Rostamizadeh, W. Shen, K. Wilder, F. Wu, and C. Yu, “Applying webtables in practice,” CIDR, 2015.

[8] D. Ritze, O. Lehmberg, and C. Bizer, “Matching HTML tables to DBpedia,” in Proc. 5th Int. Conf. Web Intell., Mining Semantics, pp. 1–6, 2015.

[9] W. Shen, J. Wang, P. Luo, and M. Wang, “LIEGE: Link entities in web lists with knowledge base,” in Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, pp. 1424–1432, 2012.

[10] J. Fan, G. Li, B. C. Ooi, K. Tan, and J. Feng, “iCrowd: An adaptive crowdsourcing framework,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, pp. 1015–1030, 2015.

[11] C. S. Bhagavatula, T. Noraset, and D. Downey, “Tabel: Entity linking in web tables,” in The Semantic Web-ISWC 2015. Springer, pp. 425–441, 2015.

[12] Z. Zhang, “Towards efficient and effective semantic table interpretation,” The Semantic Web–ISWC2014, pp. 487–502. Springer, 2014.

[13] S. Zwicklbauer, C. Seifert, and M. Granitzer, “Doser - a knowledge-base agnosticframework for entity disambiguation using semantic embeddings,” In The Semantic Web. Latest Advances and New Domains, ESWC ’16. Springer, 2016.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” In ICLR Workshop Papers, 2013.

[15] V. Mulwad, T. Finin, Z. Syed, and A. Joshi, “Using linked data to interpret tables,” In Workshop on Consuming Linked Data, 2010.