bbw: Matching CSV to Wikidata via Meta-lookup

Renat Shigapov[1 [0000−0002−0331−2558]], Philipp Zumstein[1 [0000−0002−6485−9434]], Jan Kamlah[1 [0000−0002−0417−7562]], Lars Oberländer[1 [0000−0001−7102−5369]], Jörg Mechnich[1 [0000−0002−6406−4906]], and Irene Schumm[1 [0000−0002−0167−3683]]

Mannheim University Library, University of Mannheim,
Schloss Schneckenhof, 68159 Mannheim, Germany
{firstname}.{lastname}@bib.uni-mannheim.de

Abstract. We present our publicly available semantic annotator **bbw** (**b**oosted **b**y **w**iki) tested at the second Semantic Web Challenge on Tabular Data to Knowledge Graph Matching (SemTab2020). It annotates a raw CSV-table using the entities, types and properties in Wikidata. Our key ideas are **meta-lookup** over the SearX metasearch API and **contextual matching** with at least two features. Avoiding the use of dump files, we kept the storage requirements low, used only up-to-date values in Wikidata and ranked third in the challenge.

Keywords: Semantic Annotation · Knowledge Graph · Tabular Data · Meta-lookup · Contextual Matching · Metasearch · Wikidata · SearX

1 Introduction

Motivation. We are developing a domain-specific ontology together with a knowledge graph and the reconciliation services in the project "Business and Economics Research Data Center Baden-Württemberg". We joined SemTab2020 in order to design, implement and test our own semantic annotator.

Challenge. SemTab 2020 was organized within the 19th International Semantic Web Conference and the 15th International Workshop on Ontology Matching. It consisted of 4 rounds and for the first 3 rounds the AICrowd platform was used whereas for the 4th round an offline evaluation was performed. Data in many tables as CSV files were given as input. These raw tables have no metadata (i.e. table name, column names, etc.) and suffer from misprints, mistakes, encoding problems and data changes. The goal was to annotate the column properties (CPA), column types (CTA) and cell entities (CEA) using the properties, types (classes) and entities in the Wikidata knowledge graph, see Fig. 1.

Our contribution. We designed **bbw** using two ideas: meta-lookup and contextual matching. The **bbw** relies on the Wikidata SPARQL endpoint and

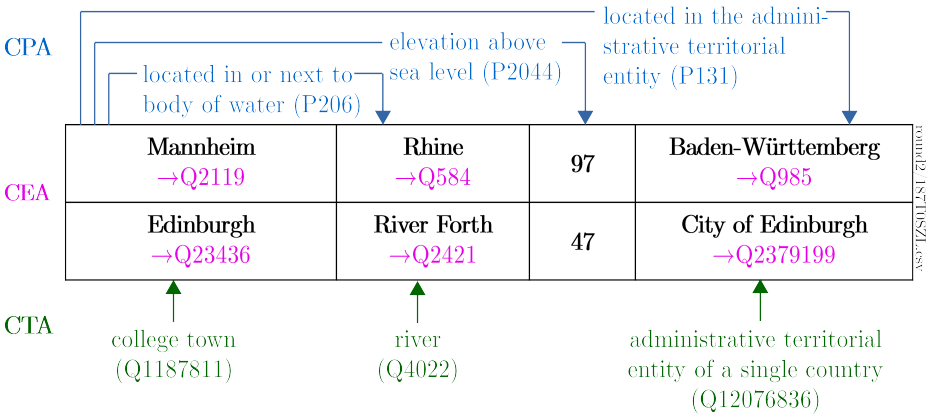

Fig. 1: Tabular data (black) is annotated with the properties (blue), entities (magenta), and types (green) from Wikidata as asked in the CPA, CEA, and CTA tasks respectively.

the locally-deployed **SearX**[1] metasearch API without using the Wikidata dump files.

Relying on Web-APIs releases us from storing large data sets and processing them locally. Moreover, the queries run every time on the most recent data without the need of developing any updating mechanism.

Related work. According to the results of SemTab 2019 [4], the key for a high performance (e.g. MTab[2] [6] and CSV2KG[3] [10]) is the use of 1) multiple lookup services and 2) elaborated lexical matching techniques. To speed up a lookup, the ElasticSearch index of labels and aliases in a knowledge graph is constructed in Tabularisi-system [12], ADOG[4] [7] and DAGOBAH [1]. The importance of contextual matching is showed by MantisTable[5] [2]. The search engine LOD4ALL[6] is used in [5].

Structure. The rest of this work is organized as follows: We describe how **bbw** works, show and discuss our results at SemTab2020, and make conclusions.

2 Implementation

The annotator **bbw** is implemented in Python3 and is distributed as an open source library with MIT License[7]. We added the command-line interface (CLI) using **argparse**[8] library. A simple graphical user interface (GUI) is developed

[1] https://github.com/searx/searx
[2] https://github.com/phucty/MTab
[3] https://github.com/IBCNServices/CSV2KG
[4] https://github.com/danielapoliveira/iswc-annotation-challenge
[5] https://bitbucket.org/disco_unimib/mantistable-tool-3
[6] http://lod4all.net/frontend/index/applicationtop
[7] https://pypi.org/project/bbw
[8] https://docs.python.org/3/library/argparse.html

with **Streamlit**[9]. All scripts are available at the project page[10]. The **bbw**-CLI is parallelized using **GNU parallel** [11] on subsets of the input data. We also implemented a few scripts for merging partial solutions as well as outputting aggregated information used for consistency checks.

2.1 Key concepts: a CSV-file and the data model of Wikidata

Wikidata as a knowledge graph can be easiest thought of simple statements, i.e. triples containing a subject, a property, and an object/literal. For the subject and object the Wikidata items are identified by a QID (Wikidata Q identifier) and the property is identified by a PID (Wikidata property) which can be prefixed accordingly in order to receive an URI. The data types for literals in Wikidata are strings, quantities (i.e. numbers), times, coordinates and more. The actual data model of Wikidata[11] is more complex but we do not focus on it.

We assume that a row in a CSV-file contains data from an item in Wikidata. We refer to that item as the subject or subject item. The first cell in a row usually contains the label (`rdfs:label`) or an alias (`skos:altLabel`) of the subject. The other (tail) cells in a row correspond to the values of properties of that subject. Such a value can either be a literal (e.g. string, number, date) or the label (or an alias) of the object item. Three kinds of annotation tasks for each row can be defined:

Entity annotation. If a cell contains a label or alias of an item in Wikidata, the cell can be annotated via the QID of this item.

Property annotation. If the values of two cells in a row are related via a property in Wikidata, the relation can be identified by its PID.

Type annotation. If a cell contains a label or alias of an item in Wikidata and the item has a statement with the property P31 (instance of), the cell's type can be annotated via the object's QID.

These annotations may not exist, are not necessarily unique and may be false positive due to mistakes in the values. The CPA and CTA tasks from the challenge are then an aggregation over all rows, such that hopefully all possible ambiguity or uncertainty vanishes.

Example: In our annotated example from Figure 1 we see in the first column two subjects Q2119 (Mannheim) and Q23436 (Edinburgh). The second and forth column contain the objects annotated with their QIDs and the third column contains literals of type number. Between the first row with the subjects and the other rows the property relations are annotated with P206, P2044, P131. The type annotation for Baden-Württemberg is Q1221156 (state of Germany) and the type annotation for the City of Edinburgh is Q15060255 (Scottish council area), but in the end the CTA for the whole column will result in Q12076836 (administrative territorial entity of a single country). Both items Q1221156 and Q15060255 are subclasses of Q12076836.

[9] https://github.com/streamlit/streamlit
[10] http://github.com/UB-Mannheim/bbw
[11] https://www.mediawiki.org/wiki/Wikibase/DataModel

2.2 Key ideas: meta-lookup and contextual matching

The data in some input tables have been changed by the challenge organizers. For example, the labels can contain typos or other artefacts: "Monnhem" instead of "Mannheim" or "dinbur" instead of "Edinburgh". The values can differ from the currently known ones in Wikidata, e.g. `9.78` instead of `9`, or `1492-09-11` instead of `1492-10-12`. Simple lookup and matching mechanisms fail when the data contain such errors. We do a meta-lookup first and, if needed, perform several contextual matching steps. However, before going into details, we will explain these two key concepts here.

Meta-lookup. Meta-lookup is a lookup over many search and metasearch engines. SearX can search over more than 80 engines. The use of some of them is constrained technically or legally, see further information in Subsection 2.4. Thus, we use only a subset of the engines: Wikidata, Wikipedia, Wikibooks, Wikiquote, Wikisource, Wiktionary, Wikiversity, Wikivoyage, Startpage, Bing, DuckDuckGo, DuckDuckGo Definitions, eTools, Erowid, Etymonline, MyMemory, Qwant, Mojeek, Naver, DictZone.

We run a docker container locally with the SearX docker image[12]. The **bbw** sends a GET-request to the SearX API. SearX sends the requests to the search and metasearch engines, collects the responses, ranks them and returns the search results, suggestions, corrections and infoboxes as JSON data. The **bbw** collects those results and takes the best matches using the edit distance from **difflib**[13] library. To add more candidates, **bbw** takes the best match in suggestions and corrections, sends them to the SearX API and collects the infoboxes. We use a few out of these candidates as an input for matching.

For example, SearX resolves "Monnhem" and "dinbur" to the correct spellings "Mannheim" and "Edinburgh".

Contextual matching. In contrast to a simple matching, a contextual matching is based on at least two features. For example, it can be 1) a label of a subject item and a label of an object item (or literal), 2) a label and type of an object item, or 3) a label of an object item (or literal) and a property.

The actual matching is performed as follows: We try an exact matching. If it does not give any results, we try a case-insensitive matching for strings. If there are still no results, we match a string with the edit distance. The names are matched ignoring abbreviations. The date, which is nearest to the value, is matched if it is within six months. All the numbers are matched within a 2 % range.

2.3 Workflow: 7 steps

We implemented **bbw** as the 7-steps workflow illustrated in Fig. 2. Step 2 is performed for all rows, but some rows might not be annotated. These rows are then considered in Step 3 including all annotations we already have after Step

[12] https://hub.docker.com/r/searx/searx
[13] https://docs.python.org/3/library/difflib.html

2. The same pattern continues up to Step 6. The postprocessing Step 7 takes the annotations from all steps as input.

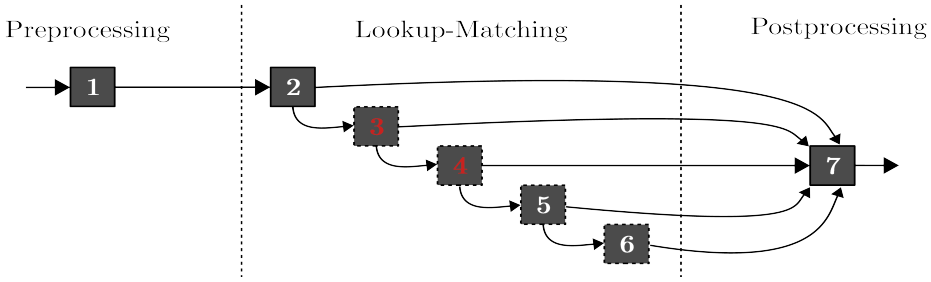

Fig. 2: The 7-steps workflow in **bbb** can be divided in a preprocessing Step 1, five contextual matching Steps 2-6, and a postprocessing Step 7. Steps 5 and 6 are only used in round 3 and 4 and Steps 3 and 4 only in round 3.

Step 1: Preprocessing. Encoding problems are fixed by the **ftfy** library [9] and the **re**gular expressions[14] are used to predict the number, time, name and string data types. A string is identified as a name if it starts with an abbreviation or has an abbreviation between two words.

Step 2: Meta-lookup of the subject's label and querying the candidates. We use the first cell in the row for a meta-lookup, assuming that it is the subject's label. The resulting candidates are used to query the Wikidata SPARQL endpoint and we get all triples for the subjects with a label from the candidates' list. We match the value in a tail cell with object's labels in the resulting triples. If the matching is successful, we take the subject's QID, the subject's type and the property. If, additionally, the object is a literal, we take also the object's QID and the object's type.

Step 2 is performed for each row. We use the most frequent property per column as the input property for Step 3.

Step 3: Querying the objects' labels with their properties. An annotated row is a row with all property annotations found. Otherwise we say that the row is still unannotated. Step 3 is executed only for unannotated rows. We query the endpoint with the values of the tail cells and their properties inferred from Step 2. We match the value in the first cell to the subject's label using the edit distance. Then, we match the resulting triples with the values in the tail cells as in Step 2.

Step 4: Querying the objects' labels of entity columns. We call a column an entity column if it has at least one type annotation inferred from previous steps. In contrast to Step 3, we do not use properties in Step 4. We query the endpoint with the values of the tail cells from all entity columns. It results in the subject's labels and the types of a subject and the objects. The cell values are matched as in Step 2.

[14] https://docs.python.org/3/library/re.html

Step 5: Querying the objects' labels with their types. As the type for this step we use the most frequent type per column inferred from previous steps. For every entity column the endpoint is queried using the value of a tail cell and its type. We take the object's QID from the received results.

Step 6: Querying the subject's types. For unannotated rows the endpoint is asked for the labels of all items with a certain type. We do this for the two most frequent type annotations of the first column. This list of labels is matched with the subject's label via the edit distance. We send another query to the endpoint and receive all triples, which have the subject's label equal to one of the labels from the candidate items. The values in the tail columns are matched as in Step 2.

Step 7: Postprocessing. After Steps 1-6 we might have a few candidates for each task. We choose the most frequent property and entity for CPA and CEA tasks correspondingly. Postprocessing for CTA task is different. If there exists the unique most frequent type, we choose it. If there are multiple most frequent types, we find the first common class w.r.t. to the subclass relation using a special query[15] to the endpoint with the SSSP class[16] in **com.bigdata.rdf.graph.analytics** package[17]. However, if the query returns the generic class (Q35120), we choose the most frequent type. As the final postprocessing step we choose only those annotations which are mentioned in the target files.

2.4 Limitations

The annotator **bbw** has both technical and legal constrains related to the use of multiple API services.

Technical constraints. The current limits of the Wikidata SPARQL endpoint are five parallel queries per IP, thirty error queries per minute and one minute runtime per single query.[18] For example, in round 3 some of our requests were blocked by the endpoint with the error code 429 (Too Many Requests) due to the use of six parallel processes. We reduced the number of CPUs in round 3 after the deadline and added an extra request after waiting the number of seconds returned by the "Retry-After" header.

SearX can throw an error message due to its own technical reasons and due to constraints of a search engine. For example, it happens when an engine asks to resolve a captcha. We experienced that during our initial tests with Google Search. Thus, we did not use Google Search for our submissions.

We found a bug in SearX in the 2nd round and reported it[19]. It was fixed shortly and we used the improved code in the next rounds.

[15] https://w.wiki/h8X. The user Pasleim proposed it.
[16] https://blazegraph.com/database/apidocs/com/bigdata/rdf/graph/analytics/SSSP.html
[17] https://blazegraph.com/database/apidocs/com/bigdata/rdf/graph/analytics/package-summary.html
[18] https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual#Query_limits
[19] https://github.com/searx/searx/issues/2188

Legal constraints. Possible legal restrictions for our usage of the search engines may occur from copyrights and sui generis database rights[20]. In accordance with the jurisdiction of the EUGH[21], these rules do not restrict our usage of search engines, because "if the database maker makes the contents of that database accessible to third parties, [...] his sui generis right does not enable him to prevent such third parties from consulting that database for information purposes"[22]. As described above, we "consult [...] the database"[23] for information purpose in contrast to build "a parasitical competing product"[24] and therefore we do not harm the investment of the database makers. Though, the terms of use of the different search engines may restrict our usage.

The terms of use of the search engines were examined for restrictions concerning metasearch. While some search engines do not have written terms of use at all, and therefore no contractual restrictions affecting their use via metasearch[25] exists, others have terms of use without restrictions on metasearching[26]. Google Search is a special case: Whereas their former terms explicitly prohibited "metasearch[ing]", their current terms merely demand to not use their services "abusively". This change can be understood as a waiver of restricting the use of their service for metasearch. Unfortunately, the terms of use of some search engines include formulations, that can be understood as a refusal of metasearching without permission[27]. However, within the territorial scope of German contract law, even the most extensive of these restrictions could be seen as invalid according to § 307 I, II Nr.1 BGB as they are in violation of § 307 II in relation with basic principle of privileging scientific research purposes stated in § 60d UrhG or at least for privileged scientific use cases unenforceable according to § 60g I UrhG.

3 Results

Table 1 shows our scores and rank in leaderboard at the SemTab2020 challenge. We did not participate in round 1 and round 2 before the first deadline. We started to implement **bbw** on the day of the first deadline in round 2. One CPU was used in round 2, six CPUs were used in round 3 before the deadline, and five CPUs were used in round 3 after the deadline and in round 4.

We applied only Steps 1, 2 and 7 of the workflow in round 2 and this already gave as a very high score in the CPA task. Our low CTA score before the second deadline in round 2 has two reasons: 1) both the target and out-of-the target

[20] Art. 7 Directive 96/9 EC and their national implementations, eg.: §§ 4, 87a ff. UrhG.
[21] EUGH C-203/02 The British Horseracing Board Rn. 55; EUGH C-304/07 Directmedia Publishing Rn. 53.
[22] EUGH C-202/12 Innoweb/Wegener Rn. 46.
[23] EUGH C-202/12 Innoweb/Wegener Rn. 47.
[24] EUGH C-202/12 Innoweb/Wegener Rn. 48; ErwG. 42, Directive 96/9 EC.
[25] For example, Duckduckgo, Etools, Mojeek and Startpage.
[26] For example, Bing, Erowid, Etymonline, MyMemory, Qwant, Wikibooks, Wikiquote, Wikisource, Wikivoyage, Wiktionary, Wikiversity, Wikipedia and Wikidata.
[27] For example, Yandex and Yahoo.

Round	CPA			CTA			CEA		
	F1	P	R	AF1	AP	R	F1	P	R
1	-	-	-	-	-	-	-	-	-
2 (<1st deadline)	-	-	-	-	-	-	-	-	-
2 (>1st deadline)	0.991	0.992	4	0.914	0.929	7	0.892	0.960	10
2 (>2nd deadline)	0.992	0.994	3	0.964	0.978	6	0.944	0.981	7
3 (<deadline)	0.949	0.957	4	0.960	0.966	4	0.954	0.974	5
3 (>deadline)	0.989	0.994	3	0.971	0.974	4	0.976	0.985	4
4 (AG)	0.995	0.996	2	0.980	0.980	2	0.978	0.984	4
4 (2T)	-	-	-	0.516	0.789	6	0.863	0.927	2
4 (avg)	0.995	0.996	2	0.762	0.884	5	0.920	0.955	2

Table 1: Our scores (F1 is F1-score, P is precision, AF1 is approximate F1-score and AP is approximate precision) and rank (R) in leaderboard at SemTab2020. AG – automatically generated dataset, 2T – tough tables, and avg – averaged over both datasets.

annotations were submitted, and 2) a bug in the evaluator at SemTab2020 which counted the out-of-the target annotations. We interpret our low CEA score as a consequence of using only Step 2 in contextual matching.

Our CPA-score dropped in round 3 in comparison to round 2 due to a bug in **bbw** leading to the incorrect counts of the most frequent property. After fixing it, our CPA scores were comparable to our CPA scores in round 2. Steps 3-6 of the workflow increased our CTA and CEA scores in round 3. We continued our tests after the deadline and found out that disabling Steps 3 and 4 increased our scores further.

Only steps 2, 5 and 6 for contextual matching were used in round 4. The dataset contained two parts: AG – automatically generated and 2T – tough tables [3]. Tough tables were easily separated, because they are not mentioned in the CPA target files and have usually higher size than the AG tables. We applied our reduced 5-steps workflow to the AG tables, got our highest scores in all tasks and ranked in second place in CPA and CTA tasks. Our 5-steps workflow is applied to roughly three quarters of the 2T dataset containing relatively small tables. Among those circa 70 % of tables were annotated. This gave us very low CTA scores with the 2T dataset. We decided to improve our CEA scores and to keep the CTA task as it was. We performed meta-lookup and non-contextual matching with the Wikidata API in order to annotate 800 of the most frequently unannotated labels. This improved our CEA score and we ranked in second place with the 2T dataset. Due to time pressure we did not add the corresponding types to our 2T-CTA submission.

4 Conclusions

We presented the semantic annotator **bbw** for matching tabular data to the Wikidata knowledge graph. The annotator is based on 1) meta-lookup over many search and metasearch engines, and 2) contextual matching with at least two features.

We were continuously developing **bbw** between rounds 2 and 4. Starting with the low CEA and CTA scores in round 2, we improved the workflow and ranked in second place in round 4. Thus, a meta-lookup is a competitive approach compared to processing the Wikidata dump files. In contrast to the latter, the meta-lookup keeps the storage requirements low and uses only up-to-date data in the knowledge graph.

Further improvements to **bbw** can be made by using more features [8], fine tuning the hyperparameters and optimizing the SPARQL queries[28].

The annotator **bbw** is distributed as an open source Python library with MIT license: https://pypi.org/project/bbw. It can be easily installed via "pip install bbw". The command line interface, a simple graphical user interface and a script for parallel computations are also provided. Everyone is welcome to adapt **bbw** at http://github.com/UB-Mannheim/bbw.

Acknowledgments

This work was funded by the Ministry of Science, Research and Arts of Baden-Württemberg through the project "Business and Economics Research Data Center Baden-Württemberg".

References

1. Chabot, Y., Labbé, T., Liu, J., Troncy, R.: DAGOBAH: An end-to-end context-free tabular data semantic annotation system. In: SemTab@ISWC 2019. pp. 41–48 (2019), http://ceur-ws.org/Vol-2553/paper6.pdf
2. Cremaschi, M., Avogadro, R., Chieregato, D.: MantisTable: An automatic approach for the semantic table interpretation. In: SemTab@ISWC 2019. pp. 15–24 (2019), http://ceur-ws.org/Vol-2553/paper3.pdf
3. Cutrona, V., Bianchi, F., Jiménez-Ruiz, E., Palmonari, M.: Tough Tables: Carefully evaluating entity linking for tabular data. In: The Semantic Web – ISWC 2020. pp. 328–343 (2020), https://doi.org/10.1007/978-3-030-62466-8_21
4. Jiménez-Ruiz, E., Hassanzadeh, O., Efthymiou, V., Chen, J., Srinivas, K.: SemTab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In: The Semantic Web – ESWC 2020. pp. 514–530 (2020), https://doi.org/10.1007/978-3-030-49461-2_30
5. Morikawa, H.: Semantic table interpretation using LOD4ALL. In: SemTab@ISWC 2019. pp. 49–56 (2019), http://ceur-ws.org/Vol-2553/paper7.pdf

[28] https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/query_optimization

6. Nguyen, P., Kertkeidkachorn, N., Ichise, R., Takeda, H.: MTab: Matching tabular data to knowledge graph with probability models. In: SemTab@ISWC 2019. pp. 191–192 (2019), http://ceur-ws.org/Vol-2553/paper2.pdf
7. Oliveira, D., d'Aquin, M.: ADOG - Annotating data with ontologies and graphs. In: SemTab@ISWC 2019. pp. 1–6 (2019), http://ceur-ws.org/Vol-2553/paper1.pdf
8. Ritze, D., Bizer, C.: Matching Web tables to DBpedia - A feature utility study. In: 20th International Conference on Extending Database Technology. pp. 210–221 (2017), https://doi.org/10.5441/002/edbt.2017.20
9. Speer, R.: ftfy. Zenodo (2019), https://doi.org/10.5281/zenodo.2591652, v. 5.5
10. Steenwinckel, B., Vandewiele, G., Turck, F.D., Ongenae, F.: CVS2KG: Transforming tabular data into semantic knowledge. In: SemTab@ISWC 2019. pp. 33–40 (2019), http://ceur-ws.org/Vol-2553/paper5.pdf
11. Tange, O.: GNU parallel 20200722 ('Privacy Shield') (July 2020), https://doi.org/10.5281/zenodo.3956817
12. Thawani, A., Hu, M., Hu, E., Zafar, H., Divvala, N.T., Singh, A., Qasemi, E., Szekely, P.A., Pujara, J.: Entity linking to knowledge graphs to infer column types and properties. In: SemTab@ISWC 2019. pp. 25–32 (2019), http://ceur-ws.org/Vol-2553/paper4.pdf