Type Inference on Wikipedia List Pages

Patrick Kuhn, Sven Mischkewitz, Nico Ring and Fabian Windheuser[1]

Abstract: The extraction of information from Wikipedia has led to a huge amount of knowledge made widely available by projects like the DBpedia[2]. So far, most effort is put into extracting explicitly encoded information e.g. infoboxes. However, Wikipedia also contains a huge amount of implicit knowledge. One example for an untouched source of implicit knowledge are Wikipedia's *List of* pages, in which multiple entities with a common type are collected. If this common type is known, it can be added to all entities of the list. Moreover, entities which are part of this list but not yet presented in the DBpedia can be added. This offers a huge potential for extending the DBpedia by adding missing type information. This paper proposes an approach to extract the shared types of a list using statistical methods and natural language processing. For a list entity, it was possible to infer new types with a precision of 86%.

Keywords: Linked Data, Ontology Enrichment, DBpedia, Wikipedia

1 Introduction

In the year 2006 Tim Berners-Lee coined the term linked data, which describes a set of best practices on how to expose and connect data from different sources [BHBL09]. Since then the linked data movement has experienced a remarkable growth. Over the years multiple classical datasets have been integrated. One of the most important techniques for representing and linking entities is the *Resource Description Framework* (RDF) [GB14]. RDF uses subject-predicate-object triples to describe relationships between entities and make them available for automatic interpretation.
Maybe the most important source of information is Wikipedia[3], one of the world's most visited websites. With around 5 million articles in the English version alone[4], it is the largest encyclopedia available. The DBpedia project[5] is one approach to extract the information from Wikipedia and make it accessible in an abstracted form. DBpedia is a community driven database, which aims at extracting structural information from Wikipedia and making it publicly available using the RDF model. Structural information includes "infobox templates, categorisation information, images, geo-coordinates, links to external web pages and links across different language editions of Wikipedia" [Au07]. Currently the English version of DBpedia contains around 3.6 million entities including around 763,000 persons, 572,000 places, and 192,000 organizations. On average an entity has four types associated with it [DB16a].

[1] Hasso-Plattner-Institute, 14482 Potsdam, Germany, {patrick.kuhn, sven.mischkewitz, nico.ring, fabian.windheuser}@student.hpi.de

[2] http://dbpedia.org/

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

[4] https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia/, April 2016

[5] http://wiki.dbpedia.org/

The great majority of extraction has been focused on information, which is explicitly stated in Wikipedia e.g. infoboxes. However there is also information, which can be concluded by studying the relations and structures of resources (implicit knowledge). One example of a currently untouched source of information are Wikipedia's *List of* pages[6]. *List of* pages are collections of multiple entities grouped together under a shared set of types.

After the types describing a list have been identified they can be added to every member of the list. As there are more than 350 thousand *list of* pages[7] in the English Wikipedia this approach offers a huge opportunity to enrich the DBpedia. This information is particularly important because ”proper classification of entities into types is indispensable for any Information Extraction (IE) system” [Po12].

Table 1 shows four entities from the *list of German scientists*[8]. The row *parsed types* shows an excerpt of the associated DBpedia types. As *Franz Aepik* is not represented in the DBpedia, he does not have any type information. *Scientist*, *Person*, *Agent*, and *Thing* can be determined as the shared types for the list and added to every member. In this case the type *dbo:Scientist* can be added to the entity *Roland Benz*. Additionally it is possible to add missing entities such as *Franz Aepik* with all associated types to the DBpedia.

	Leonhard Euler	Carl Friedrich Gauss	Roland Benz	Franz Aepik
Parsed types	dbo:Scientist dbo:Person dbo:Agent owl:Thing	dbo:Scientist dbo:Person dbo:Agent owl:Thing	dbo:Person dbo:Agent owl:Thing	missing in DBpedia
Missing types			dbo:Scientist	dbo:Scientist dbo:Person dbo:Agent owl:Thing

Tab. 1: List of German scientists

This paper proposes an approach to automatically extract type information using the information in Wikipedia's *list of* pages combined with the information from DBpedia. This type information can be used to extend the DBpedia by adding missing types and even adding new entities.

This paper is structured as follows: Section 2 discusses previous work. The approach to extract common types is presented in Section 3. After the entities belonging to a list have been extracted from Wikipedia as described in Section 3.1, candidate types are proposed. These types are then ranked using statistical methods and natural language processing as described in Section 3.3. To evaluate the accuracy of the proposed types a test set has been created as described in Section 4. This test set is used to evaluate the approach as discussed in Section 5.

[6] https://en.wikipedia.org/wiki/List_of_lists_of_lists/
[7] Titles starting with ”List_of” in the title dump: http://wiki.dbpedia.org/Downloads2015-04#titles/
[8] https://en.wikipedia.org/wiki/List_of_German_scientists/

2 Related Work

The goal of this work is to increase the type coverage of DBpedia entities. In linked data, type information is represented by the *rdf:type* predicate. This predicate associates an entity with an ontology. In the case of linked data the term ontology most often refers to a set of classes, their properties, and relations between the class members. The ontology provided by DBpedia is a ”shallow, cross-domain ontology” [DB16b], which covers 685 classes. The DBpedia ontology is pretty small and not all concepts can be expressed by only using classes from it. Therefore multiple other ontologies such as YAGO [SKW07] have been developed. In contrast to DBpedia, the YAGO ontology consists of more than 350,000 classes. These classes are derived from WordNet [Mi95] and Wikipedia categories. The problem with the YAGO ontology is that its ”granularity is often too high” [Ga12]. A result of a larger ontology is that the overlap of types is smaller. Since our algorithm uses this overlap to infer new types, better results can be achieved using the DBpedia ontology.

In general, approaches can be divided into methods which work directly on the information of the Wikipedia page and methods which work on the information provided by linked data.

Tipalo [Nu13] falls into the former category and tries to extract defining statements (e.g. ”XX is a YY”) of an entity from the abstract of its corresponding Wikipedia page. The type information of these statements are extracted and matched against WordNet types. The approach reaches an overall recall of 74% with a precision of 76%. This proves that the abstract can contain useful type information. Because generating information from natural languages is always error prone, we only use the content of Wikipedia pages to further confirm already extracted types instead of generating new ones.

Giovanni et al. introduced a method [Nu12], which makes use of the links between Wikipedia pages to infer type information. The authors report a recall of 86% and a precision of 52%. However the approach is limited to only predict one of the nine top level classes, whereas our approach can be used to infer types from the complete DBpedia ontology.

One of the first paper which only uses linked data to extract new information is discussed by Neville et al. [NJ00]. They train a model on already labeled data and iteratively apply this model to find new types for unlabeled entities. This yields an accuracy of 82%.

Another approach which makes use of the relations of existing linked data sources is presented in [PB13]. The underlying idea of this approach is that each relation is connected to a specific type with a given probability. This can be used to infer new tyes using a weighted voting approach. This leads to F-measure of 88.5%. In contrast to these approaches which only rely on the linked data itself, our approach combines information from Wikipedia and linked data.

This paper extends the approach of Paulheim et al. [PP13], which proposes a two staged process to extract type information from Wikipedia’s *list of* pages. In the first step candidate classes are identified based on statistical methods. Second, these classes are ranked according to information gained by natural language processing frameworks. Paulheim et al. presents this method from a theoretical point of view and list a set of problems, such as fail-safe extraction and suitable scoring functions [PP13], which have to be solved in order to actually implement this process.

3 Extracting Common Types

Given all entities belonging to a *list of* page the main goal is to find a set of DBpedia types which describes the entire list. Figure 1 shows the stages of the approach. In order to achieve this, the first step is to parse and translate the Wikipedia article into an abstract representation. From this representation the actual list members and their types have to be determined. Once this is done the types are ranked according to their relevance for the list. From this ranking a set of fitting types is extracted.

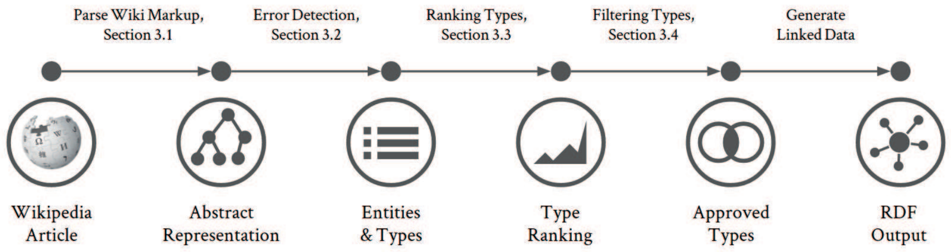
Parse Wiki Markup, Section 3.1
Error Detection, Section 3.2
Ranking Types, Section 3.3
Filtering Types, Section 3.4
Generate Linked Data
Wikipedia Article
Abstract Representation
Entities & Types
Type Ranking
Approved Types
RDF Output

Fig. 1: The linear process from Wikipedia articles to mined data.

3.1 Parsing Wikipedia

Multiple solutions already exist for converting Wiki markup into abstract representations. A flexible solution is provided with the *json-wikipedia parser*[9]. The json-wikipedia parser utilises the *TU-Darmstadt parser*[10] to convert Wiki markup text from a XML dump [Wi16a] into JSON[11]. To satisfy all needs for the extraction of list pages, adaptions[12] have been made to json-wikipedia and the TU-Darmstadt parser. To filter unimportant sections like *External Links*, knowledge about the entity links in the respective section and paragraph on the Wikipedia page has to be preserved. Besides that, there are constructs in the Wiki markup syntax which are not recognized by the original TU-Darmstadt parser e.g. lists composed of multiple columns.

The analysis of 2000 List_of pages[13] showed, that lists can be categorized into three major groups:

[9] https://github.com/Wikilist-Extraction/json-wikipedia/
[10] http://mvnrepository.com/artifact/de.tudarmstadt.ukp.wikipedia/
[11] http://json.org
[12] https://github.com/Wikilist-Extraction/json-wikipedia/
[13] http://windheuser.com/p/random2000.zip/

- **Bullet point lists:** Lists that consist of a sequence of bullet points followed by the entity.
- **Table lists:** Lists containing entities represented in one column of a table.
- **Mixed lists:** Lists without a clear structure or with multiple different representations.

We concentrate on the extraction of knowledge from bullet point and table lists, since they exist more often in Wikipedia. Furthermore bullet point and table lists provide some kind of structure which makes extraction less error prone.

When dealing with bullet point lists the greatest difficulty in parsing arises from nested lists. Furthermore other edge cases have to be considered like multiple entities behind a bullet point.

When extracting table lists, it is often unclear which column is the main column and which are only supportive columns describing the main column. There are several factors affecting the probability of a column being the main one. Features for identifying the main column are its index and the ratio of unique entities. Moreover the list entities in the main column are often described by the other columns. Table 2 shows the list of NBA champions with the respective number of their titles and their division. The entities in the club column are described by the other columns using the *dbp:division* and *dbo:title* predicate. Many relations to another column is a clear indication, that this column contains the list entities. These relations can be extracted using the DBpedia.

This feature is converted to a numerical value by calculating the average number of connections from an entity to the other columns. Once all these features have been normalized between zero and one, the column with the highest weighted sum of these features is chosen as the main column. The weights have been determined using grid search.

Rank	Club	Division	Title
1	Boston Celtics	Atlantic	17
2	Los Angeles Lakers	Pacific	16
3	Chicago Bulls	Central	6
4	San Antonio Spurs	Southwest	5

Tab. 2: List of NBA-Champions[14]

3.2 Error detection

One central problem is, that not every concept, can be mapped to a DBpedia type. The *list of dance pop artists*[15] is an example for a list whithout a direct counterpart in the DBpedia ontology. The types *dbo:MusicalArtist* and *dbo:Band* both describe the content of the list

[14] https://de.wikipedia.org/wiki/National_Basketball_Association/
[15] https://en.wikipedia.org/wiki/List_of_dance-pop_artists

pretty well, but are distinct on a very top level. Musical Artists is a subclass of Person, whereas Band is a subclass of Organisation.

Another problem is that different writing styles in combination with the loose structured syntax make it impossible to take all special cases into account. As a consequence of this a criteria to detect errors is needed. One clear indicator that the wrong entities have been extracted is a great variety of not closely related types in the result set. To detect this a notion to quantify semantic relatedness between two types is needed. For this, an adapted version of the *Leacock and Chodorow similarity* [LC98] is used. The original Leacock and Chodorow similarity is defined by

$$Sim(T1,T2) = -\log(\frac{distance(T1,T2)}{2*depth})$$

where *distance* is defined by the length of the shortest path between two types in the ontology tree and *depth* being the maximum depth of the ontology.

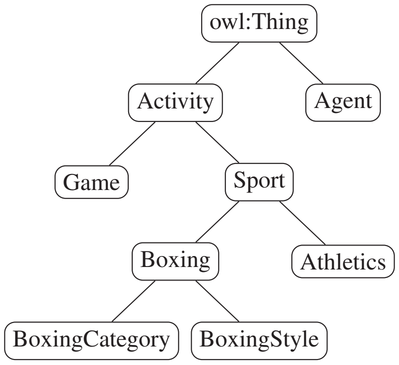

Fig. 2: DBpedia ontology

Figure 2 shows an extract of the DBpedia ontology tree. From this diagram it can be seen that the similarity between two terms is highly dependent on the level of a node. For example *dbo:BoxingCategory* and *dbo:BoxingStyle* are much closer related then *dbo:Activity* and *dbo:Agent*, even though the Leacock Chodorow similarity is the same. For this reason we use the sum of the depths instead of the taxonomy depth for normalization.

$$Sim(T1,T2) = -\log(\frac{distance(T1,T2)}{depth(T1)+depth(T2)})$$

3.3 Ranking Types

The open question is how to select a set of type candidates relevant for the whole list. Therefore identification of types specific to the entire list is required. As proposed by Paulheim this can be achieved by statistical means and textual evidence [PP13].

Statistical means: Many resources are annotated with generic types like *owl:Thing*. If a type occurs with a much higher frequency in a list than in the DBpedia, it is likely that this type is significant for this list. For example having multiple entities with the type *dbo:Scientist* attached is far more expressive than the the type *owl:Thing*. This can be expressed with the TF-IDF statistic. TF-IDF indicates how relevant a single term is to a document in a corpus. In the scope of type properties of a DBpedia entity this means types should occur frequently in the generated type set (term frequency) but should be specific to the considered list as well (inverse document frequency) [PP13]. Term frequency in our case denotes the relative frequency that a type occurs in all list members. Inverse term frequency refers to how common the type is in the whole DBpedia. The TF-IDF weight is computed as follows:

$$tfIdf(type) = tf(type) * idf(type)$$

$$tf(type) = \frac{countInList(type)}{\#entitiesInList}$$

$$idf(type) = log(\frac{\#entitiesInDBpedia}{countInDBpedia(type)})$$

countInList(type) is the number of entities in the list with the given type. *entitiesInList* states how many entites where found in the list. Analogously, *countInDBpedia(type)* and *entitiesInDBpedia* counts the entities and types in the entire DBpedia.

Textual evidence: The statistical measure is further supported by searching for textual appearances of the type in the Wikipedia articles. The list page and the wikipages of the respective entities are used to scan for type labels. Title, abstract, and categories are considered as they are the most accurate description of an entity. Abstracts in Wikipedia articles tend to describe more than the reason why an entity is a member of a specific list. Therefore matches in the title and categories are weighted stronger than matches in the abstract. To improve matching for word variations in the article, the Porter Stemming algorithm [Po80] is used. Each match contributes to the textual evidence score.

Statistical means and textual evidence produce two independent rankings of types. In order to compare the two rankings, they have to be normalized to a scale between zero and one. A combined ranking is obtained by multiplying both scores for each type in the list. A weighting of 85:15 between statistical means and textual evidence is used, based on empirical results.

3.4 Filtering Types

To produce a set of fitting type candidates the ranked list of types has to be filtered according to the computed score. Cutting off at a fixed value does not always fit the underlying data. In the manual analysis of a test set of 2000 lists, we observed that quite often the transition between fitting and non-fitting types is marked by a significant score drop. To take this into account, the last value before the score drop gets determined as a threshold. This divides the ordered list of types into two cluster with the approved and declined types. This approach adapts more flexible to the respective lists.

3.5 Architecture of the Implementation

The previously described approach has been implemented in Scala and Java and is publicly available on Github[16]. Apache Jena, a Java framework[17] for building Semantic Web and Linked Data applications, is used to connect the application to RDF datasources. To process massive data asynchronousy, Reactive Streams[18] are used.

Querying the RDF datasets is the slowest task of the application. Jena TDB[19] is used to minimize the impact of data access to the overall performance. Jena TDB is a native high performance triple store, which supports the full range of Jena APIs. It is possible to use TDB with a multiple reader or single writer policy for concurrent access [Fo16]. This enables parallel queries to the database. Thus slow queries can be compensated by concurrency. Additionally TDB is accessed asynchronously to avoid blocking database queries. Therefor an asynchronous wrapper for Jena SPARQL queries based on Scala Futures[20] was build. The wrapper can be used with SPARQL endpoints, Linked Data Fragments[21] endpoints, and TDB. By preventing busy waiting the pipeline can process 2000 list pages in about 85 seconds [22].

4 Evaluation

The quality of the results is normally evaluated using existing type information by checking whether this information can be reproduced by the system. This was not suitable for this approach, because the aim of this work is to infer new type information and in most cases only top level types are present in DBpedia. As a result a ground truth dataset had to be developed. A total of of 400 randomly chosen lists have been annotated by four different persons. For each list all types of the parsed entities have been annotated as approved or declined. In addition lists could be tagged as incorrectly parsed. The evaluation set is publicly available [Wi16b].

There are multiple cases in which it is hard to decide if a type is appropriate for a certain list. For example one could argue that it would be correct to label Shakespeare as an artist, although in the DBpedia class hierarchy *dbo:Writer* is not a subclass of *dbo:Artist*. In this cases agreements have been made between the annotators.

[16] https://github.com/Wikilist-Extraction/wikipedia-list-extraction
[17] https://jena.apache.org/index.html
[18] http://doc.akka.io/docs/akka-stream-and-http-experimental/current/scala.html
[19] https://jena.apache.org/documentation/tdb
[20] http://docs.scala-lang.org/overviews/core/futures.html
[21] http://linkeddatafragments.org/
[22] Run on a 2 GHz Intel Core i7 Processor and 8 GB 1600 MHz DDR3 Memory

5 Results and Discussion

As a result of the evaluation a precision of 86.19% with a recall of 33.13% has been achieved (F1 score of 0.48). A total number of 60,786 new RDF triples were produced based on the 400 annotated list pages.

Since our approach concentrates on finding the most specific types for a list, more generic types are left out. The recall decreases since these generic types are still correct for most lists. Since DBpedia types are arranged in a hierarchy one could also propose all types of a higher level. This can be done in a post-processing step by also adding all super types and would increase the recall.

The main limitation of the algorithm is still fail safe parsing. When regarding the performance of the algorithm on correctly parsed lists, the precision of our algorithm increases to 94.97%.

Difficulties occur, when the list entities do not contain the shared types. One example for such a list is the *list of German expressions in English*[23]. In this case the common type for the list is *German expressions*. The list entities however have a wide variety of types such as *dbo:Food*, *dbo:MusicalGenre* or *dbo:Weapon*. Since the type *German* or *expression* does not occur it can not be proposed. This reduces the recall, since these lists are filtered out using the error detection mechanism as described in section 3.2.

6 Conclusion and Outlook

In this paper we proposed an approach to extract new type information by combining linked data and natural language processing techniques. On a randomly chosen sample of 2000 randomly chosen lists a total number of 303,934 new type triples have been computed in 85.56 seconds computation time[24].

As solutions to the problems discussed in [PP13] this paper described a reliable method to extract the correct entities from wikipedia list pages by improving the parsing of Wiki markup and introducing a quality measure for the parsed entities. For specifying the correct types, the weighted TF-IDF and textual evidence scores are combined. Moreover, a method for determining the shared types from the obtained results was developed. As the evaluation has shown, this method is able to produce high quality type information (86.19% precision), while still having a feasible run time.

While the focus of this paper is adding DBpedia types to entities, there are a lot of cases where the common type of a list is not contained in DBpedia. For example, there is a list of hills in the Scottish lowlands over 2000 feet, which are called Donalds[25]. To get more specific types for these lists, it would be possible to generate a new type. In this case the type *dbo:Donald* could be proposed. This could be achieved using the list name and an

[23] https://en.wikipedia.org/wiki/List_of_German_expressions_in_English/

[24] Run on a 2 GHz Intel Core i7 Processor and 8 GB 1600 MHz DDR3 Memory

[25] https://en.wikipedia.org/wiki/List_of_Donalds/

advanced textual evidence approach with entity recognition based on the abstracts of the entities.

Another way to extend our approach is to not only extract the type of a list but to also extract its describing axioms. At the moment only the type *scientist* is extracted for the *list of german scientist*. The describing property *German* is not inferred. This could be achieved by extending the statistical ranking proposed in this paper to also include the properties of the entities.

7 Acknowledgment

This work was created based on the seminar *Knowledge Mining* at the Hasso-Plattner-Institute. The authors thank Harald Sack, Jörg Waitelonis and Magnus Knuth for their support.

References

[Au07] Auer, Sören; Bizer, Christian; Kobilarov, Georgi; Lehmann, Jens; Cyganiak, Richard; Ives, Zachary: DBpedia: A Nucleus for a Web of Open Data. In: In Proceedings of the 6th International Semantic Web Conference. Springer, 2007.

[BHBL09] Bizer, Christian; Heath, Tom; Berners-Lee, Tim: Linked data-the story so far. Semantic Services, Interoperability and Web Applications: Emerging Concepts, pp. 205–227, 2009.

[DB16a] DBpedia Data Set Statistics, http://wiki.dbpedia.org/services-resources/datasets/data-set-38/data-set-statistics/.

[DB16b] DBpedia Ontology, http://wiki.dbpedia.org/services-resources/ontology/.

[Fo16] TDB Java API - Concurrency, https://jena.apache.org/documentation/tdb/java_api.html #concurrency/.

[Ga12] Gangemi, Aldo; Nuzzolese, Andrea Giovanni; Presutti, Valentina; Draicchio, Francesco; Musetti, Alberto; Ciancarini, Paolo: Automatic Typing of DBpedia Entities. In: Proceedings of the 11th International Conference on The Semantic Web - Volume Part I. ISWC'12, Springer-Verlag, Berlin, Heidelberg, pp. 65–81, 2012.

[GB14] Guha, Ramanathan; Brickley, Dan: RDF Schema 1.1. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[LC98] Leacock, Claudia; Chodorow, Martin: Combining local context and WordNet similarity for word sense identification. WordNet: An electronic lexical database, 49(2):265–283, 1998.

[Mi95] Miller, George A: WordNet: a lexical database for English. Communications of the ACM, 38(11):39–41, 1995.

[NJ00] Neville, Jennifer; Jensen, David: Iterative classification in relational data. In: Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data. pp. 13–20, 2000.

[Nu12] Nuzzolese, Andrea Giovanni; Gangemi, Aldo; Presutti, Valentina; Ciancarini, Paolo: Type inference through the analysis of Wikipedia links. In: LDOW. 2012.

[Nu13] Nuzzolese, Andrea Giovanni; Gangemi, Aldo; Presutti, Valentina; Draicchio, Francesco; Musetti, Alberto; Ciancarini, Paolo: Tìpalo: A Tool for Automatic Typing of DBpedia Entities. In: The Semantic Web: ESWC 2013 Satellite Events, pp. 253–257. Springer, 2013.

[PB13] Paulheim, Heiko; Bizer, Christian: Type inference on noisy rdf data. In: The Semantic Web–ISWC 2013, pp. 510–525. Springer, 2013.

[Po80] Porter, Martin F: An algorithm for suffix stripping. Program, 14(3):130–137, 1980.

[Po12] Pohl, Aleksander: Classifying the Wikipedia articles into the OpenCyc taxonomy. In: Proceedings of the Web of Linked Entities Workshop in conjuction with the 11th International Semantic Web Conference. volume 5, p. 16, 2012.

[PP13] Paulheim, Heiko; Ponzetto, Simone Paolo: Extending DBpedia with Wikipedia List Pages. In: NLP-DBPEDIA@ISWC'13. pp. –1–1, 2013.

[SKW07] Suchanek, Fabian M; Kasneci, Gjergji; Weikum, Gerhard: Yago: a core of semantic knowledge. In: Proceedings of the 16th international conference on World Wide Web. ACM, pp. 697–706, 2007.

[Wi16a] Wikipedia: Database Download, https://en.wikipedia.org/wiki/Wikipedia:Database_download/.

[Wi16b] Evaluated List Types, https://github.com/Wikilist-Extraction/annotated-lists/.