Building a Knowledge Graph

Lukas Blübaum, Nick Düsterhus, and Monika Werner

University of Paderborn, Padeborn 33098, Germany
{lukasbl,nduester,mwerner}@uni-paderborn.de
https://github.com/LukasBluebaum/BKG

Abstract. The goal of this proseminar was to create an encyclopetic knowledge graph from a large text corpus like a wikipedia dump. In order to learn and better understand the basics of Semantic Web technologies. For this we were given a wikipedia dump and the core ontology of DBpedia. To facilitate this task we were introduced to several tools including FOX [7] and DBpedia Spotlight [1]. The whole project is written in Java and as license we used the GNU General Public License v3.0. The documented source code, including instructions on how to run the code, can be found on the given GitHub repository.

Keywords: Knowledge Graph · Semantic Web · Coreference Resolution · Named Entity Recognition · Entity Disambiguation · Relation Extraction · FOX · Spotlight

1 Introduction

Websites like Wikipedia contain a large volume of information. Unfortunately this information is mainly unstructured, for example in the form of natural language text. So the detection of complex informationen can be quite difficult. One approach to structure this knowledge is the extraction of triples from the natural language text. These triples consist of a subject, predicate and object and can then be displayed as a knowledge graph. The creation of such a knowledge graph is the main goal of this project.

To carry out this projekt several steps had to be taken. At first we had to process the given wikipedia dump and the core ontology. For the wikipedia dump this meant cleaning the text and for the core ontology we filtered some not required information. The next steps included Named Entity Recognition, Entity Disambiguation and Relation Extraction. As frameworks for the Named Entity Recognition and Entity Disambiguation we used FOX and Spotlight. For the Relation extraction we implemented to approaches. The first approach takes the sentences and sends them to FOX in "re" mode to directly extract the triples. The second one, our own approach, uses Spotlight and the Stanford Core NLP [2]. Finally we evaluated the results in relation to runtime and correctness. Here we measured how many triples from DBpedia we extracted per article, and manually checked the correctness of triples that are not contained in DBpedia. Due to time constraints we chose the category of U.S. Presidents for the evaluation.

1.1 Architecture

Just a few points to the architecture of our project. We used the WikiCleaner and OntologyParser classes to process the given DBpedia ontology and wikipedia dump. Then the RelationExtraction class starts two threads, one for the FOX approach and one for our own approach These two threads then try to extract the triples with the help of the NLPParser class and the webservices classes that query the online FOX demo and Spotlight demo respectively. As the last step the extracted triples will be written to a RDF graph.

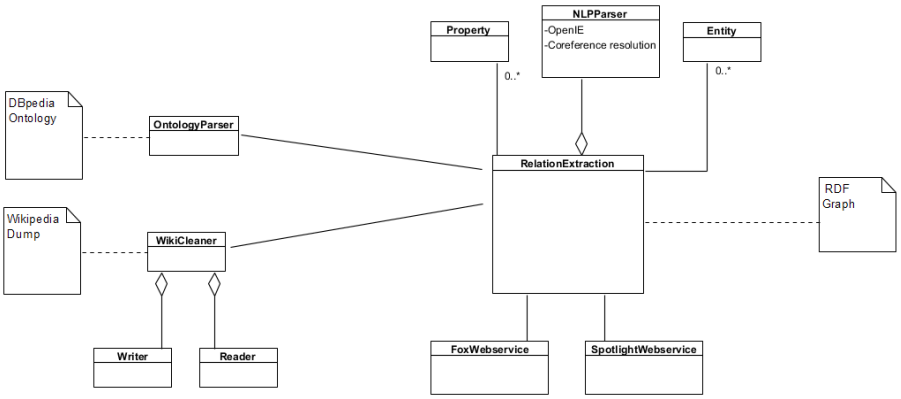


Fig. 1: A quick overview of the architecture of our project.

2 Preprocessing

The preprocessing consists of the cleaning of the wikipedia dump and the coreference resolution.

2.1 Extraction and cleaning of the text from a wikipedia dump

As the foundation we were provided with a wikipedia dump. This wikipedia dump already came without markup, infoboxes or chapter subdivisions. The fact that there are no infoboxes already presented some problems, but more to that during the evaluation and discussion. But there were still some elements that were undesired for the later steps in this process. Such elements include unnecessary URLs, text in parenthesis, whitespace, nullchars and special symbols. We removed the texts passages in parenthesis because the Stanford CoreNLP had some problems with these during the binary relation extraction for our own approach later on. The elements were removed under application of some regular expressions.

The result is a simple text containing only the letters from the english alphabet, numbers, periods, commas and of course simple spaces. In order to speed up the cleaning process, we used a combination of Reader,Worker and Writer threads. To handle the concurreny we used BlockingQueues from the standard Java libraries. One which the Reader thread uses to pass the read lines to the Worker threads and another one which the Worker threads use to pass the cleaned lines to the Writer thread.

2.2 Coreference Resolution

Another step that needs to be taken before the information can be properly extracted is the coreference resolution, so that Spotlight and FOX can also recognize named entities in sentences were previously only a coreference appeared. Therefore we have to replace the pronouns with their representative mention. To actually find this representative mention we used the coref annotator [6] of the Stanford CoreNLP.

Example 1. John met Judy in 1960. He married her during his college year.

In this sentence we obviously want to replace the he, her and his with their corresponding representative mention. The result would look like this:

Example 2. John met Judy in 1960. John married Judy during John's college year.

This example already shows one of the challenges that arise during the coreference resolution. Possesive pronouns have to be detected and handled accordingly. So in this case *his* should not be replaced with *John* but rather with *John's*. To make this distinction we use the dependency parser [5] of the Stanford CoreNLP. Here we take the pronoun in question and then determine if it appears in connection with a possesion modifier relation. If it does, we adjust the representative mention the pronoun will be replaced with accordingly.

3 Named Entity Recognition

For this step we use the DBpedia Spotlight web service. We send the text to DBpedia Spotlight and receive a JSON response. The response is then parsed and the found entities and their information is stored in objects of our Entity class. As information we keep the uri, surface form, offset and the types. Since we send more than one sentence at a time to Spotlight we need to keep the offset to map the entities back to their corresponding sentence where they appeared. The surface form we need later on for our own relation extraction where we have to locate the entities inside binary relations. In addition to that we also keep the types because during our own relation extraction we compare these to the range and domain of some properties.

4 Entity Disambiguation

The Entity Disambiguation itself is already done by Spotlight and FOX. Spotlight for example has two approaches for this. After they have located a candidate they take the information next to the surface form as information and try to determine the best match based on that. They also try to weigh words on their ability to actually disambiguate between candidates [3]. FOX and AGDISTIS [4] [8] were already discussed in the lecture.

5 Relation Extraction

5.1 Using FOX

FOX, can be used additionally to our relation extraction algorithm. FOX and our algorithm run in different threads in order to increase the number of triples with a reduced runtime. Here we just request FOX in 're' mode and then parse the output and write the triples that were found to the graph.

5.2 Own Approach

Our algorithm relies on the coreference resolution as well as Stanford OpenIE. Third person pronouns are replaced by their representative mention. Our algorithm tries to send as many sentences as Spotlight can handle at once to reduce the number of requests which leads to a reduced runtime. Moreover, this provides Spotlight more context to perform disambiguation.

We build an entity class to save all the information we received, including the offset and the surface form to match the entities back to the sentences as well as the types. Those types are triples itself, but they are also needed to determine which properties would be a valid match according to their respective domain and range. In the next step, we had to process Wikipedia sentences to extract binary relations between entities. Since Wikipedia sentences tend to be more complex simple dependency parsing can lead to failure. We decided to use OpenIE, a system that extracts binary relation triples from a text. OpenIE recursively traverses the dependency tree to find independent clauses. It then searches for shorter arguments that are logically entailed by the original sentence. It also uses a proof system to only create logically coherent clauses. We precompute the relations and save them for each sentence.

With the entities and relation stored the next task is to map those to DBpedia properties. Therefore we parse the DBpedia ontology and create Property (Relation) Objects. Each Property Object contains the label, domain, range and a number of keywords that allow us to map the property to natural language text. The first list of keywords is generated from the property labels. The properties are stored as a list, converted to JSON and saved as a file that we could

manually or automatically edit. A keyword String attribute that consists of multiple words, can be interpreted as a vector of which all words have to be found to map a relation to the property. We worked with lemmatized keywords and binary relations to reduce the needed amount of keywords.

We then iterate through all the entities and binary relation for a sentence and try to match the subject and the object of the binary relation to entities. Afterwards we map the keywords to the binary relation and check for the range and type. If everything matches we create a triple and write it to the graph.

For entity to literal relations, we first match the subject of the binary relation to an entity. Then we search for numbers inside the object. In case that we also find a month and two numbers we convert them to the xsd:date format. If we only find a number, we keep this number for the keyword matching step. Often Wikipedia plain text literals are partly written-out. To account for these cases we convert them to the actual number.

Example 3. 82 million $\Rightarrow$ 82000000

6 Example

In the following we will demonstrate the whole process at an example.

Example 4. Obama was born on August 4, 1961, at Kapiolani Medical Center for Women and Children in Honolulu, Hawaii. He graduated from Harvard University.

After the coreference resolution the sentence will look like this:

Example 5. Obama was born on August 4, 1961, at Kapiolani Medical Center for Women and Children in Honolulu, Hawaii. Obama graduated from Harvard University.

Next we use OpenIE to extract binary relations from the sentences, some of them are listed in the following table.

Table 1: Shortened version of the binary relations extracted by OpenIE. (only some of the important ones we can actually use)

Subject	**Predicate**	**Object**
Obama	be bear on	August 4 1961
Obama	graduate from	Harvard University
Obama	be bear at	Kapiolani Medical Center for Women in Honolulu

Then Spotlight will find the following Named Entities.

Example 6. **Obama** was born on August 4, 1961, at **Kapiolani** Medical Center for Women and Children in **Honolulu**, **Hawaii**. **Obama** graduated from **Harvard University** .

Now as already discussed we will try to map the subject, predicate and object to entities, properties and entities or literals. Here we distinguish between entity to literal relation and entity to entity relation.

Table 2: Mappings entity to literal relation. For the first binary relation of Table 1.

Part of binary relation	**Keywords**	**Mapping**
Subject	Obama	dbr:Barack_Obama
Predicate	bear	dbo:birthDate
Object	August 4 1961	1961-08-04

Table 3: Mappings entity to entity relation. For the third binary relation of Table 1.

Part of binary relation	**Keywords**	**Mapping**
Subject	Obama	dbr:Barack_Obama
Predicate	graduate	dbo:almaMater
Object	Harvard University	dbr:Harvard_University

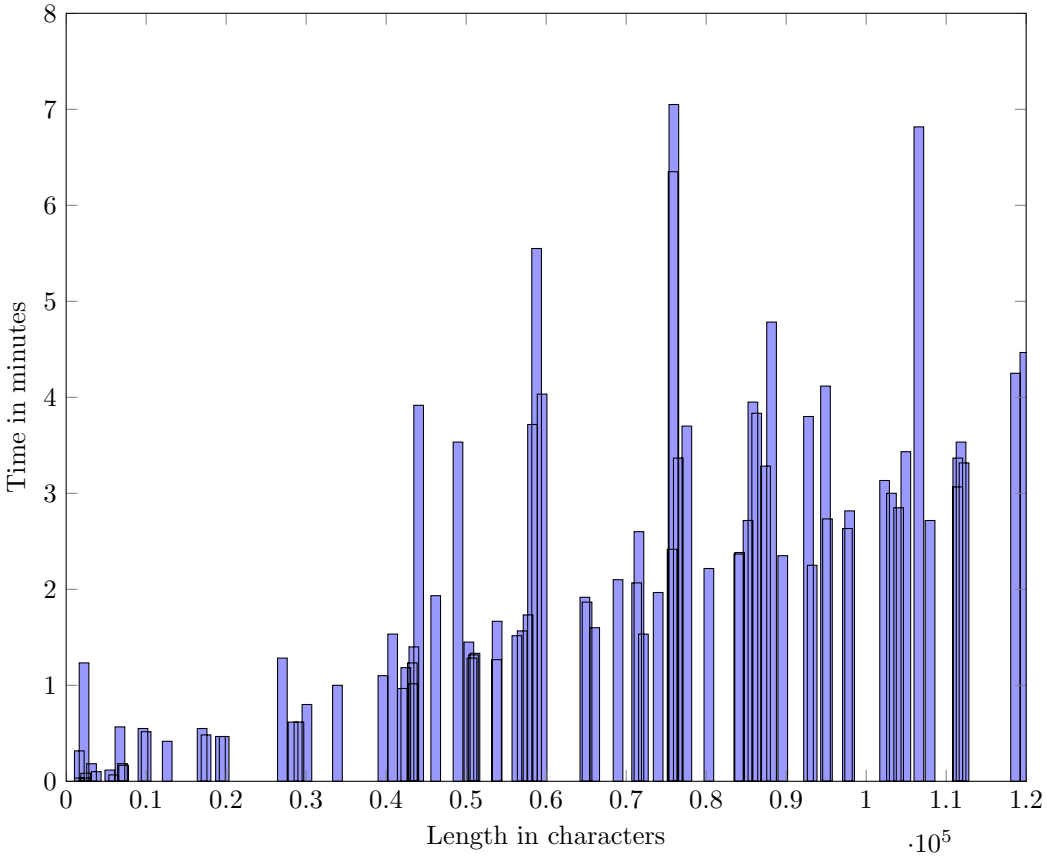
So finally we get these two triples and write them to the graph. In this example we could also find the triple with the birth place of Honolulu.

7 Benchmark and Evaluation

Note that for benchmarking and evaluation we only used our relation extraction approach.

7.1 Runtime

We measured the runtime of 100 articles in length of characters per time in minutes (see the graph below). We considered articles of different lengths to demonstrate how the runtime develops if the articles get longer. A huge share of the runtime is due to the coref annotator of the Stanford CoreNLP. This also explains the outliers because the coref Annotator then just takes longer for these sentences.



Computer Specifications

- OS: Windows 10
- CPU: Intel Core i5-7600K
- SSD

7.2 Correctness

For the evaluation we wrote a Benchmark class that takes a category and one or more dumps as input and then queries all subjects from the given category. Then we compare all triples from our graph that have a subject from the category with those from the dumps.

We selected the category *Presidents of the United States*. First we compared our result to the given dumps, then we selected 100 triples randomly that were not contained in the dumps and checked them manually.

From the 2810 triples contained in the dump - from the chosen category - we also found 556. Note here that this also contains types we received from Spotlight, without the types these numbers are 2320 and 121. From the 100 randomly selected triples 46 were correct. This leads to a recall of 0.197 and precision of 0.46. So the resulting F-measure amounts 0.27. In the evaluation folder on the GitHub repository we provided the used files.

The provided dumps (2016-10)

- disambiguations_en.ttl, instance_types_en.ttl, labels_en.ttl, long_abstracts_en.ttl, mappingbased_literals_en.ttl, mappingbased_objects_en.ttl, persondata_en.ttl, specific_mappingbased_properties_en.ttl, transitive_redirects_en.ttl

8 Discussion

In conclusion, our system can find relations from plain text utilizing two different strategies by FOX and our own algorithm. Our software relies on Spotlight for Named Entity Recognition and Stanford CoreNLP with OpenIE as our main tool to extract logically coherent relations. By coreferencing the articles we can consider each sentence for itself. Our algorithm scales with the amount of keywords that are added to the mentioned property file. It can theoretically find relations for all DBpedia properties, besides Entity to String relations since we lack context to reach a good precision on those.

In the following we will discuss some difficulties and possible improvements. Many triples can only be found in the infoboxes so we cannot find them in the text itself. For example the height of a person. We only focused DBpedia ontology properties - except types and names - so it can distort the benchmarking since our system is not designed to find properties based on the structure of the text like dbo:abstract or other namespaces like <http://purl.org/dc/terms/description>. We also missed some triples or extracted incorrect ones because of the disambiguations. For example Spotlight often falsely recognized parties not as the actual party but rather as the history of that party. In addition to that the disambiguation also fluctuated depending on the amount of text.

The main improvement that could be made, would be to adjust the information including the keywords for the properties. Since sometimes we miss context if the property lacks the domain and/or range. For example dbo:citizenship not having range dbo:Country. A huge number of carefully selected keywords or maybe even negative keywords so words that should not occur would also improve the property selection.

References

1. Daiber, J., Jakob, M., Hokamp, C., Mendes, P.N.: Improving efficiency and accuracy in multilingual entity extraction. In: Proceedings of the 9th International Conference on Semantic Systems (I-Semantics) (2013)
2. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: Association for Computational Linguistics (ACL) System Demonstrations. pp. 55–60 (2014), http://www.aclweb.org/anthology/P/P14/P14-5010
3. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: Dbpedia spotlight: Shedding light on the web of documents. In: Proceedings of the 7th International Conference on Semantic Systems. p. 3. I-Semantics '11 (2011)
4. Moussallem, D., Usbeck, R., Röder, M., Ngonga Ngomo, A.C.: Mag: A multilingual, knowledge-base agnostic and deterministic entity linking approach. In: K-CAP 2017: Knowledge Capture Conference. p. 8. ACM (2017), https://svn.aksw.org/papers/2017/KCAP_MAG/sigconf-main.pdf
5. Nivre, J., de Marneffe, M.C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C.D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., Zeman, D.: Universal dependencies v1: A multilingual treebank collection. In: Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016) (2016)
6. Recasens, M., de Marneffe, M.C., Potts, C.: The life and death of discourse entities: Identifying singleton mentions. In: North American Association for Computational Linguistics (NAACL) (2013)
7. Speck, R., Ngonga Ngomo, A.C.: Ensemble learning for named entity recognition. In: The Semantic Web ISWC 2014, Lecture Notes in Computer Science, vol. 8796. Springer International Publishing (2014)
8. Usbeck, R., Ngonga Ngomo, A.C., Auer, S., Gerber, D., Both, A.: Agdistis - graph-based disambiguation of named entities using linked data. In: 13th International Semantic Web Conference (2014), http://svn.aksw.org/papers/2014/ISWC_AGDISTIS/public.pdf