QALD-Mini-Project

Lukas Blübaum, Nick Düsterhus, and Ralf Keller

University of Paderborn ,Warburger Str. 100, 33098 Paderborn, Germany
{lukasbl,nduester,rkeller}@uni-paderborn.de
https://github.com/LukasBluebaum/QALD-Mini-Project

Abstract. As part of the lecture "Semantic Web", we developed a template based Question answering engine. The Question answering engine is able to answer different types of questions using DBpedia as the knowledge base to extract those answers. The engine maps keywords to find information that are needed to generate appropriate SPARQL queries. Including keywords that are used to determine the selection of the templates. Those information can be properties and classes. DBpedia Spotlight is used to find the entities of the questions. We iterate over the set of information needed to fill in the templates. The properties get ranked, to improve the probability that our first query returns a correct answer. Our engine achieves an F-score of 0.527 for the QALD8-Test Dataset.

Keywords: QALD · Question Answering · Spotlight · SPARQL.

1 Introduction

The World Wide Web is filled with information for everyone to explore. Most information on the Web is unstructured, what makes it hard to process for humans and computers. The Semantic Web is a approach to provide structured data in the Web that is easy to process by computers and can be processed to be easily understood by humans.

Most Semantic Web Databases use SPARQL as the language to query the database. That means that in order to search the semantic web the user has to learn a query language, that is not very easy to learn. This is not user friendly and can be improved.

The goal of our project is to provide a interface that takes a question formulated in natural language and answers it by querying DBPedia. The interface will be able to be used over the web via HTTP-POST requests. We aim for a F-measure of at least 0.1.

Our project uses a Template-based approach. That means we have defined templates of SPARQL-Queries that are modified at predefined locations based on the question asked. The project consists of three components: The *Question-Answering (QA) Engine*, the *Question-Processor* and the *SPARQLQueryBuilder*.

We use the Library *qa.annotation* to find entities, properties and classes, qa.commons to load and store QALD-datasets and GERBIL QA, a wrapper for web communication.

The *QA Engine* is responsible for providing the interface to users. It reads questions from the Webservice or a predefined dataset and passes the question to the Question Processor. Furthermore the QA Engine is responsible for outputting the answer, i.e sending a HTTP-Response to the user.

Relevant entities contained in the question have to be identified and the question has to be analyzed to determine the type of the question. This is done by the *Question Processor* component.

To get a answer a SPARQL-Query has to be build and executed on a endpoint. That is the responsibility of the *SPARQLQueryBuilder*. This component uses the processed information provided by the Question Processor, builds a SPARQL-Query by using predefined templates and executes the query on an endpoint provided by DBPedia.

2 Simplified Procedure

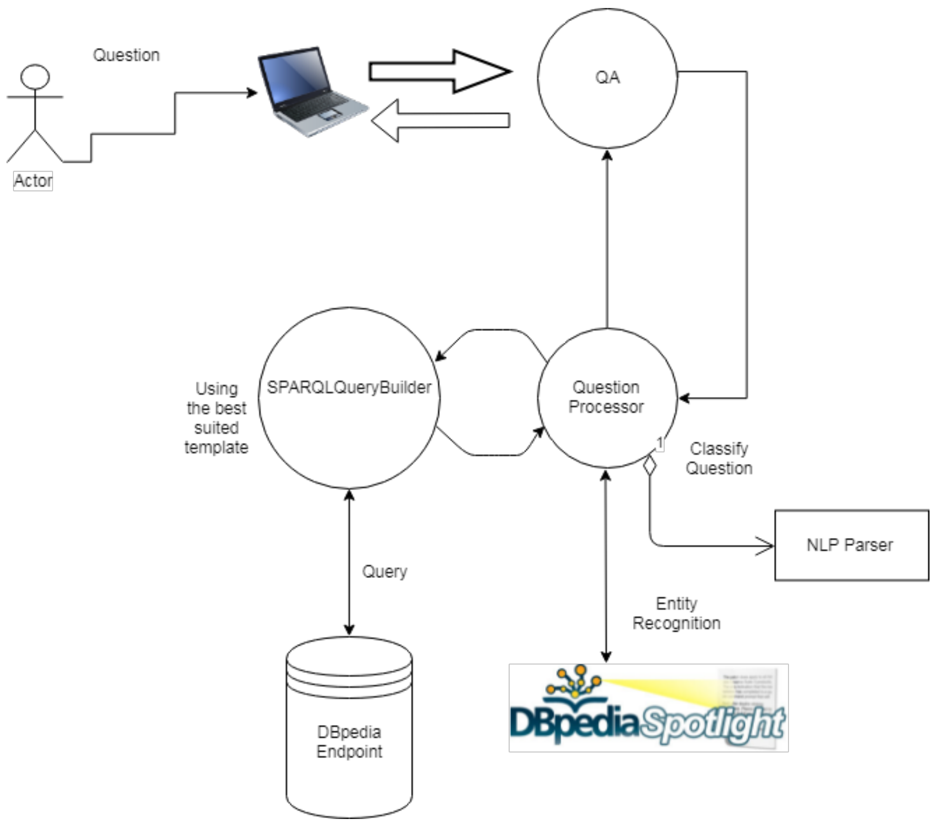


Fig. 1. Architecture.

The Question Answering processing includes multiple steps. After the question have been received, the question will be forwarded to the QuestionProcessor by the QA class.
This class annotates the questions with all additional information that are important to be able to find an answer. The question is sent to DBpedia Spotlight which performs named entity recognition. The found entities are saved in a question object. Following this, we use the Stanford CoreNLP [2] in order to find words that could indicate which DBpedia property we will need to later fill in the templates.
Therefore we pass those keywords to IndexDBO_classes provided by the qa.annotation library.
Furthermore, the IndexDBO_classes maps the nouns of the question to classes if a corresponding class exists in the DBpedia namespace. The IndexDBO class

itself performs a fuzzy search on a predefined file that contains a set of DBpedia properties and keywords using the Apache Lucene library.
All gathered information will be stored in the question object.
Then we classify the question based on the question word (cf. 5 Template overview).
Depending on the classification the SparqlQueryBuilder class, given the question object, runs a method in which a SPARQL template will be filled in with the found entities, classes and properties.
The method itself does some selection of which SPARQL template will be used out of some prepared templates - corresponding to the first classification - based on keywords (e.g. many or comparatives) and the number of entities and classes.
Then we query the DBpedia SPARQL endpoint using apache Jena with all SPARQL queries that seem meaningful given the set of possible properties. The result is returned and converted to an answer JSON object by the provided gerbil wrapper classes.

3 QA System

The QA-System component is the component that reads questions from various datasources and relays them to other components, mainly to the *Question Processor*. It is also responsible for outputting the answer to asked questions.

There are three supported input-modes:

1. Ask a single question stored in a variable
2. Load datasets with multiple questions from a JSON-File
3. Run a queryable webservice

Option 1 is mainly used for debugging purposes and is not of further interest. The second option is used to run practice datasets provided by the dice research group. Our aim of a F-Score of at least 0.1 refers to these datasets. Option 3 is used to provide a webservice a user can use to get their questions answered.

The QA-Component consists of multiple subcomponents. One subcomponent is the *QA* class. This class is the main class of the program, the *main* Method is invoked on startup. It is responsible for checking which mode the program should run in. The mode is defined by a flag in the code that has to be set at compile time. It loads the question and releys them to the Question Processor, based on which mode is selected. If the program runs in *Dataset* mode, the QA class is also responsible for writing the results as a JSON-File to the local storage. To load the dataset, we use the *LoaderController* class provided by the dice research group.

The second component is the *QA-System* component. This class' only purpose is it to take a question and return a *Answer Container*. The Answer Container contains the answer of the question and the query used to get the result.

The QA-System relays the question to the QuestionProcessor to get the Answer Container. The QA-System is mainly used by the webserver to determine the answer to a given question.

3.1 Webserver

The third component is the webserver that listens for questions over HTTP. The webserver is part of the GerbilQA-Benchmarking-Template provided by the dice research group. It utilizes the Spring-Framework to provide the service over the web.

The webserver starts listening for HTTP-POST requests under the */gerbil* path. The parameter of the requests have to contain the question to be answered under the *query* parameter and the language of the question under the *lang* parameter.

The webserver relays the the question to the QuestionProcessor by using the *QA-System* component, which returns an *Answer Container*. The webserver converts the Container to a JSON-String, which is sent to the user as a response to their POST-request.

4 Question Preprocessing

Before we can start to build a SPARQL-Query we need to preprocess the questions. First we classify them depending on their starting word, because by doing so we can already get many information about the expected answer. So for example if we have a "When" question we already know that the answer should be of datatype date or year.

Then we use the Stanford CoreNLP to find keywords inside the questions. Here we extract nouns, verbs and adjectives and later map them to classes and properties from the DBpedia ontology. We also extract superlatives and comparatives to further differentiate the used templates. For the Named Entity Recognition we request the Spotlight [1] demo through the qa.annotation library. Then we remove entities that can also be mapped to classes, because somtimes Spotlight detects something like "Mountain" also as entity (resource) and we normally want to use that as a class and not an entity.

Finally we apply the IndexDBO classes of the qa.annotation library to the extracted nouns,verbs and adjectives to map them to classes and properties from the DBpedia ontology. Here normally verbs, adjectives will be mapped to properties and nouns to classes. In the next step we will then rank the properties, so that we can test the properties with the most hits, triples on DBpedia with that property, first. We also extented the property list in qa.annotations and just added all DBpedia ontology properties.

5 Template Overview

Below in table 1 you can see a quick overview of some of the question types we differentiated. Afterwards we also looked at the number of entites and classes and if there is a superlative or comparative to choose the template accordingly. Finally we requested the DBpedia endpoint using Apache Jena.

Table 1. A quick overview for some of the question types.

Question type	**Expected answer**
boolean (Do, Did, Has, Was, Does ...)	true or false
Where	Place
When	datatypes date or year
Who	Person
List, Name, Show ...	list of resources

5.1 Example

An example for the most basic query consting of <entity> <property> ?answer.

Example 1. "Who was the doctoral supervisor of Albert Einstein?"

First we see that it is a "Who" question so we already know that the answer should be a Person. Spotlight will find one entity: *dbr:Albert Einstein*. Then *doctoral supervisor* will be mapped to *dbo:doctoralAdvisor*.

Example 2.
```
SELECT DISTINCT ?answer WHERE {
?answer a foaf:Person.
<http://dbpedia.org/resource/Albert Einstein>
<http://dbpedia.org/ontology/doctoralAdvisor> ?answer . }
```

5.2 Superlatives, Comparatives

When the user enters a question like *What is the highest mountain in Germany?* the SPARQLQueryBuilder builds a query that fetches all mountains in Germany, sorts them descending by height and selects the first entry. This process is applicable for every other superlative and comparative.

The QueryBuilder has to know which superlatives and comparatives there are, which attributes they compare and if the result has to be descending or ascending. There is no online resource which provides this information, so we created an Enum called *Comparisons* that contains a selection of superlatives and comparatives in connection with the corresponding sort order (ascending for lowest, smallest, descending for highest, longest) and a reference to appropriate DBPedia attributes.

The *SPARQLQueryBuilder* iterates over all comparatives and superlatives and checks if the question contains any of them. If the Builder detects a comparative or superlative it selects the appropriate predefined Query-Template and modifies it according to the data saved in the enum. By using this approach we have a quick and uncomplicated way to handle comparatives and superlatives. The disadvantage of this approach is that we can only handle comparatives or superlatives that we defined on attributes we defined. If the user wants to compare an attribute we have not thought about this approach will not yield any result. See the table 2 below for a few example entries from the Comparison enum.

Table 2. A few example entries from the Comparison enum.

Word	**Uri**	**Order**
largest	dbo:areaTotal	DESC
tall	dbo:height	-
taller	dbo:height	DESC
tallest	dbo:height	DESC
high	dbo:elevation	-
higher	dbo:elevation	DESC
highest	dbo:elevation	DESC

5.3 Example

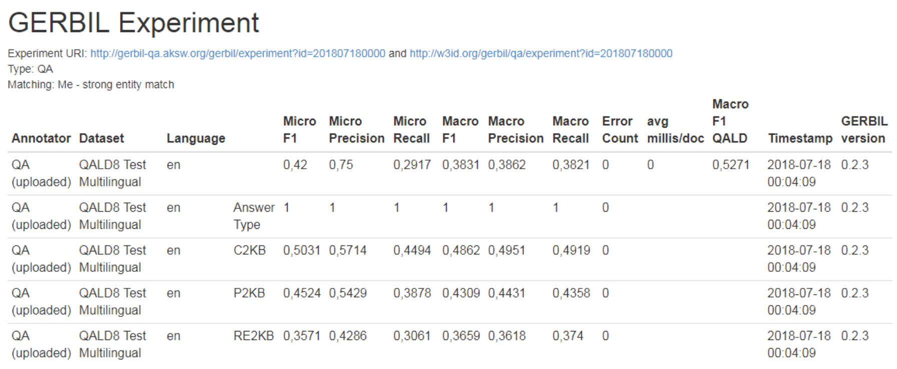
Example 3. "What is the largest country in the world?"

Here Spotlight will find no entity. We detect the class *Country* and from our Comparison enum the word largest (dbo:areaTotal) with the order "DESC".

Example 4.
```
SELECT DISTINCT ?answer WHERE {
?answer rdf:type <http://dbpedia.org/ontology/Country>.
?answer <http://dbpedia.org/ontology/areaTotal> ?area . }
ORDER BY DESC(?area) LIMIT 1 OFFSET 0
```

6 Benchmarking and Evaluation

The GERBIL platform computes an F-score over our answers. We evalutated the QALD8_Train as well as the QALD8_Test set.



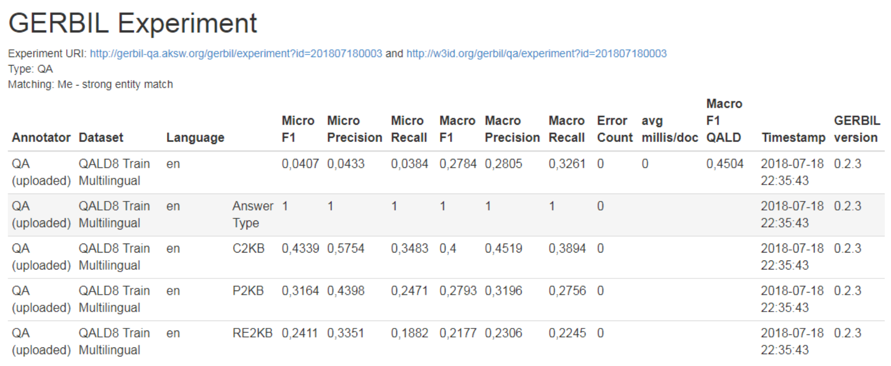
GERBIL Experiment

Experiment URI: http://gerbil-qa.aksw.org/gerbil/experiment?id=201807180000 and http://w3id.org/gerbil/qa/experiment?id=201807180000
Type: QA
Matching: Me - strong entity match

Annotator	Dataset	Language		Micro F1	Micro Precision	Micro Recall	Macro F1	Macro Precision	Macro Recall	Error Count	avg millis/doc	Macro F1 QALD	Timestamp	GERBIL version
QA (uploaded)	QALD8 Test Multilingual	en		0,42	0,75	0,2917	0,3831	0,3862	0,3821	0	0	0,5271	2018-07-18 00:04:09	0.2.3
QA (uploaded)	QALD8 Test Multilingual	en	Answer Type	1	1	1	1	1	1	0			2018-07-18 00:04:09	0.2.3
QA (uploaded)	QALD8 Test Multilingual	en	C2KB	0,5031	0,5714	0,4494	0,4862	0,4951	0,4919	0			2018-07-18 00:04:09	0.2.3
QA (uploaded)	QALD8 Test Multilingual	en	P2KB	0,4524	0,5429	0,3878	0,4309	0,4431	0,4358	0			2018-07-18 00:04:09	0.2.3
QA (uploaded)	QALD8 Test Multilingual	en	RE2KB	0,3571	0,4286	0,3061	0,3659	0,3618	0,374	0			2018-07-18 00:04:09	0.2.3

http://gerbil-qa.aksw.org/gerbil/experiment?id=201807160000

Fig. 2. Gerbil experiment for the QALD8-Test set, with a F-measure of 0.527.



GERBIL Experiment

Experiment URI: http://gerbil-qa.aksw.org/gerbil/experiment?id=201807180003 and http://w3id.org/gerbil/qa/experiment?id=201807180003
Type: QA
Matching: Me - strong entity match

Annotator	Dataset	Language		Micro F1	Micro Precision	Micro Recall	Macro F1	Macro Precision	Macro Recall	Error Count	avg millis/doc	Macro F1 QALD	Timestamp	GERBIL version
QA (uploaded)	QALD8 Train Multilingual	en		0,0407	0,0433	0,0384	0,2784	0,2805	0,3261	0	0	0,4504	2018-07-18 22:35:43	0.2.3
QA (uploaded)	QALD8 Train Multilingual	en	Answer Type	1	1	1	1	1	1	0			2018-07-18 22:35:43	0.2.3
QA (uploaded)	QALD8 Train Multilingual	en	C2KB	0,4339	0,5754	0,3483	0,4	0,4519	0,3894	0			2018-07-18 22:35:43	0.2.3
QA (uploaded)	QALD8 Train Multilingual	en	P2KB	0,3164	0,4398	0,2471	0,2793	0,3196	0,2756	0			2018-07-18 22:35:43	0.2.3
QA (uploaded)	QALD8 Train Multilingual	en	RE2KB	0,2411	0,3351	0,1882	0,2177	0,2306	0,2245	0			2018-07-18 22:35:43	0.2.3

http://gerbil-qa.aksw.org/gerbil/experiment?id=201807170000

Fig. 3. Gerbil experiment for the QALD8-Train set, with a F-measure of 0.45.

7 Discussion

Our system reaches an F-score of about 0.527. This is partly due to the number of properties to which the words of the question can get matched to otherwise, some question that could be answered with a simple SPARQL query could not be answered.

Further factors include the ordering of those properties and a larger number of different templates. The QALD8 set has a focus on questions containing comparative, superlative as well as temporal aggregators, so our templates are mostly designed to answer those kinds of questions.

Presumably, our system will have a slightly worse result on other questions sets. For example the selection of templates for boolean and list questions could be improved. Our system provides the foundations to answer a multitude of questions, so it could easily be extended to further improve the result.

Our system is not able to understand the semantics of the questions.
Nevertheless identifying keywords to map those to classes and properties as well as keywords that enable us to classify the questions leads to an appropriate selection of a SPARQL-Query.

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