Joint Entity and Relation Linking using EARL

Debayan Banerjee[1,2], Mohnish Dubey[1,2], Debanjan Chaudhuri[1,2], and Jens Lehmann[1,2]

[1] Smart Data Analytics Group (SDA), University of Bonn, Germany
{dubey, chaudhur, jens.lehmann}@cs.uni-bonn.de
debayan@uni-bonn.de
[2] Fraunhofer IAIS, Bonn, Germany
jens.lehmann@iais.fraunhofer.de

Abstract. In order to answer natural language questions over knowledge graphs, most processing pipelines involve entity and relation linking. Traditionally, entity linking and relation linking have been performed either as dependent sequential tasks or independent parallel tasks. In this demo paper, we present EARL, which performs entity linking and relation linking as a joint single task. The system determines the best semantic connection between all keywords of the question by referring to the knowledge graph. This is achieved by exploiting the connection density between entity candidates and relation candidates. EARL uses Bloom filters for faster retrieval of connection density and uses an extended label vocabulary for higher recall to improve the overall accuracy.

Keywords: Entity Linking, Relation Linking, Question Answering

1 Introduction

Accessing information from knowledge graphs (KGs) efficiently has been an important research goal in the last decade. In particular, question answering techniques have been proposed to allow obtaining information from knowledge graphs based on natural language input. A semantic question answering system [4,5] requires (i) entity identification and linking (ii) relation identification and linking (iii) query intent identification and (iv) formal query generation. In this submission, we are concerned with steps (i) and (ii).

In most entity linking systems disambiguation is performed by looking at other entities present in the text. However in the case of short questions there is often no other entity to be found. To deal with such cases we look at not just other entities but also relations in the natural language question. EARL performs the joint linking task by looking for the connections between the candidates in the knowledge graph. Thus, entities help in relation disambiguation and relations help in entity disambiguation. To achieve high recall EARL uses an extended label vocabulary. Additionally use of Bloom filters [1] instead of KB queries results in low response times.

This is an accompanying demo paper for EARL [3], which was accepted at the main ISWC research track and is a component in the QA pipeline that performs joint identification and linking of entity and relation. In addition to the main conference paper, we provide a) an API for using EARL b) description of the usage of Bloom filters to speed up triple queries c) description of the extended vocabulary for DBpedia labels for entities and relations.

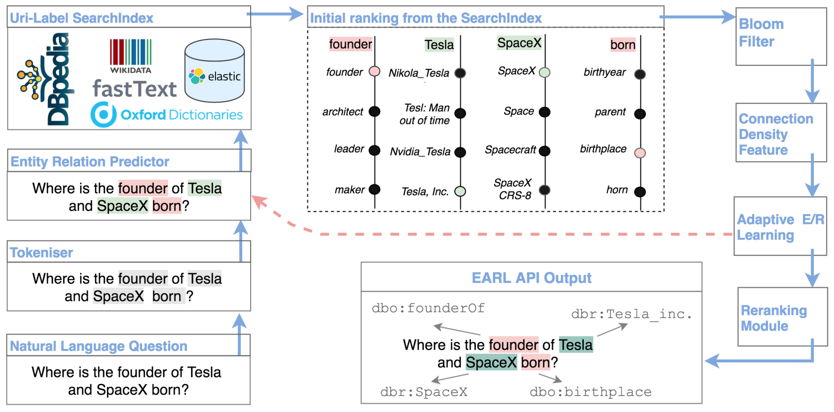

Fig. 1. EARL Architecture

2 Architecture for EARL

2.1 System Overview

Here we give an overview of EARL [3] pipeline to show the components which use the *Bloom filter* and *Extended Label Vocabulary*

Step 1: Shallow Parsing: Earl extracts all keyword phrases from a question using SENNA[3]. E/R prediction module identifies whether the phrase is an entity or relation candidate.

Step 2: Candidate List Generation: EARL uses a Uri-Label index for retrieving candidate URIs for keyword phrases. The *Extended Label Vocabulary* index consists of DBpedia labels extended by Wikidata, Oxford dictionary[4] and FastText[5].

Step 3: Connectivity Detection: We take candidate lists from the previous step and see how well each candidate is KB-connected to other candidates in other lists. This information is distilled into three features, namely Hop-Count $\mathcal{H}$, Connection-Count $\mathcal{C}$ and initial Rank of the List $\mathcal{R}_i$. For checking connectivity among candidates in the knowledge base we use pre-built *Bloom filters* in-memory instead of making network calls to DBpedia. This leads to faster response times.

Step 4: Re-ranking candidate lists: From the top-k candidate list for each entity/relation, we predict the most probable candidate. EARL relies on XGBoost [2] classifiers for re-ranking of the lists based on the three features passed ($\mathcal{H}$, $\mathcal{C}$, $\mathcal{R}_i$).

2.2 Implementation Details

2.2.1 Creating an Extended Label Vocabulary: In the text search phase we need to retrieve a candidate list for each keyword identified in the natural language question by the shallow parser. To retrieve the top candidates for a keyword we create an Elasticsearch index of uri-label pairs. Since EARL requires exhaustive list of labels for a DBpedia uri

[3]https://github.com/torch/senna
[4]https://developer.oxforddictionaries.com/
[5]https://github.com/facebookresearch/fastText

we expanded the labels beyond the dbpedia provided label. We used Wikidata labels using dbpedia "same-as" links for entities. For example label for dbr:BarackObama is only "Barack Obama" in DBpedia, but we expand this set by using Wikidata sameas, thus our index has labels "Barack Obama, Barack Hussein Obama, President Obama, Barack, ... ". For relations we required labels which were semantically equivalent for which we took synonyms from the Oxford Dictionary API. For example dbo:writer has the label "writer" in DBpedia; using OxfordDictionary we expand with labels such as "author, penman, creator, ...". EARL further uses FastText for covering all the inflection forms of these labels, such "write, writer, writing, wrote, written, ...". Our extended vocabulary contains 256K labels for DBpedia relations.

2.2.2 Using Bloom filters for fast querying of the KB: While performing joint connectivity detection EARL checks connectivity and distance between two nodes of the knowledge graph. EARL checks the connection between a candidate of a keyword phrase to all the candidates of all the other keyword phrases. Two DBpedia nodes may have multiple intermediate nodes in the path connecting them and looking at all of them takes a large amount of time as the number of such (pair of nodes) queries are high. We do not directly query the knowledge graph as the execution time for such a query in a large knowledge graph is prohibitively high.
EARL uses *Bloom filters* [1], that are probabilistic data structures which can answer questions of set membership, like "is-a-member-or-not" in constant O(1) time. The user can decide the acceptable error rate when creating the Bloom filter by choosing the appropriate size of the Bloom filter and corresponding number of hash functions. EARL uses Bloom filter parameters so that it has 1 in a million error probability.
There are separate Bloom filters for different hop length pairs. We took each such pair as a string (eg: *http://dbpedia.org/resource/car:http://dbpedia.org/resource/bike*) and added it to the corresponding Bloom filter. Hence we ended up with a 4 Bloom filter with different lengths (upto 4hops), each around 1 GB in size resident in-memory. This method allows us to condense a KG which is several hundred GBs in size into only a few GBs with a low probability of error. When we need to find connection density, we take each pair of candidate URIs from the lists returned by Elasticsearch and ask the Bloom filters if they contain these URI pairs or not. Depending on which Bloom filter answered positively we know how many hops away these two URIs are in the knowledge graph.

2.2.3 Experiment for response time: We empirically show the query-time speed up achieved by the usage of Bloom filter compared to SPARQL over KG in terms of response time. In our experiment we checked the connectivity between two nodes of

Connection check with path-length	**SPARQL infer time**	**Bloom filter infer time**
1-hop e_1 - p_1	3300 μsec	17 μsec
2-hop e_1 - $?x$ - e_2	3300 μsec	17 μsec
3-hop e_1 - $?x$ - $?y$ - p_2	8500 μsec	17 μsec
4-hop e_1 - $?x$ - $?y$ - $?z$ - e_3	11059 μsec	17 μsec

Table 1. Bloom filter's inference time compared to SPARQL

the knowledge graph with a fixed path length. We observe that the SPARQL response time (3300 micro seconds) for such a query is many folds higher then the Bloom filter's response time (17 micro seconds). The results in Table 1 show that the response time of SPARQL increases with the hop length, where as the Bloom filter results are constant. However it must be added that the one-time construction of the Bloom filters as a pre-processing step requires several hours to complete.

3 API and Data Set

EARL API is publicly available at `http://sda.cs.uni-bonn.de/projects/earl/`. The EARL API takes the natural language question as input, and the returns a ranked list of the URIs for the corresponding keyword phrases. A demo of EARL is also accessible via EARL's homepage mentioned above. The ranked list output by EARL also consists of *confidence scores*. EARL's connection density module has a time complexity of $\mathcal{O}(N^2L^2)$, where N is the number of keyword phrases, and L is the size of list retrieved for candidate keyword phrase. The number of bloom queries made for a question is given by the equation $(L^2(N^2 - N)/2)$. EARL API has an average response time of 0.42 seconds per question when queried on the same machine hosting the service. The source code for EARL is available at `https://github.com/AskNowQA/EARL`. Along with the code, we also release the extended label vocabulary. The fully annotated version of the LC-QuAD [6] dataset, where relation and entity in questions are marked with their corresponding URIs is available at `https://figshare.com/projects/EARL/28218`.

4 Conclusion

In this paper, we present an extension of the EARL paper by providing a) a public service and API for EARL b) additional results for Bloom filters and implementation details c) the extended label sets (for DBpedia entity and relations) used in EARL.

References

1. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
2. T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.
3. M. Dubey, D. Banerjee, D. Chaudhuri, and J. Lehmann. Earl: Joint entity and relation linking for question answering over knowledge graphs. In *International Semantic Web Conference*, 2018.
4. M. Dubey, S. Dasgupta, A. Sharma, K. Höffner, and J. Lehmann. Asknow: A framework for natural language query formalization in sparql. In *International Semantic Web Conference*, pages 300–316. Springer, 2016.
5. K. Höffner, S. Walter, E. Marx, R. Usbeck, J. Lehmann, and A.-C. Ngonga Ngomo. Survey on challenges of question answering in the semantic web. *Semantic Web*, (Preprint):1–26, 2016.
6. P. Trivedi, G. Maheshwari, M. Dubey, and J. Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*, pages 210–218. Springer, 2017.