Towards an Approach based on Knowledge Graph Refinement for Answer Type prediction

Azanzi Jiomekong[1], Vadel Tsague[1], Brice Foko[1], Uriel Melie[1] and Gaoussou Camara[2]

[1]Department of Computer Science, University of Yaounde I, Yaounde, Cameroon

[2]Unité de Formation et de Recherche en Sciences Appliquées et des TIC, Université Alioune Diop de Bambey, Bambey, Sénégal

Abstract

This paper presents our contribution to the SMART 3.0 Answer Type (AT) prediction research problem. This contribution consists of a method for Answer Type prediction that can be applied to any Knowledge Graph. Inspired by the TransE algorithm for link prediction in knowledge graphs, we defined the models. Concerning category prediction, we trained the model on 80% of the training dataset furnished by the organizers and tested on 20% by using different models parameters. Applied to the test data provided by the organizers, we obtained the max accuracy of 0.96 for the DBpedia and the category_match value of 0.94 for the Wikidata dataset.

Keywords

Question Answering, Answer Type Prediction, Knowledge Graphs, Knowledge Graphs Refinement, Wikidata, DBpedia,

1. Introduction

Question Answering (QA) systems aim to respond to natural language questions of users. To this end, Knowledge Base Question Answering (KBQA) is required to understand the meaning of natural language questions and furnish reliable responses. Thus, KBQA is a popular task in the field of Natural Language Processing (NLP) and Information Retrieval (IR). It aims to answer a natural language question using the facts in the Knowledge Base [1]. To build efficient QA systems, several subtasks should be considered. While SMART 2020[1] focus on Answer Type (AT) subtask, SMART 2021[2] on Answer Type and Relation Linling (RL) subtasks, SMART 3.0[3] consider Answer Type prediction, Relation Linking and Entity Linking (EL) subtasks. During the SMART 3.0 challenge, we addressed the three subtasks. This paper is devoted to the proposition of a solution for the Answer Type prediction problem.

Answer Type prediction consists of the classification of the expected answer type. In effect, the ability to know the AT enables the system to significantly narrow down the search space for an answer to a given question [1]. For example, given the following question: "Where was born Maurice Kamto?" It is possible to reduce the search space from 67 answers to 7 entities while querying all the 1-hop entities of the Maurice Kamto resource in DBpedia (*dbr:Maurice_Kamto*). The illustration of this example is given by the table 1.

[1]https://smart-task.github.io/2020/
[2]https://smart-task.github.io/2021/
[3]https://smart-task.github.io/2022/

A question with answer type predicted

```
{
"Question": "Where
    was born Maurice
    Kamto?",
"category":"resource
    ",
"type":[
    "dbo:Country",
    "dbo:
        PopulatePlace
        ",
    "dbo:Place",
    "dbo:Location",
    "dbo:Settlement",
    "dbo:village"
    ]
}
```

SPARQL Query over DBpedia

```
SELECT DISTINCT ?type ?o ?p WHERE{
    dbr:Maurice_Kamto ?p ?o .
    ?o rdf:type ?type .
    FILTER strstarts(str(?type),
        str(dbo:))
}
```

Result: 67 results

SPARQL Query with AT predicted

```
SELECT DISTINCT ?type WHERE {
    dbr:Maurice_Kamto dbo:birthPlace ?o
        .
    ?o rdf:type ?type .
    FILTER strstarts(str(?type), str(
        dbo:))
}
```

Result: 7 results

Table 1
SPARQL example on how AT prediction can help to reduce the solution space over DBpedia dataset

SMART 2020 was devoted to the AT task and SMART 2021 to AT and RL tasks. Fourteen papers were proposed for the AT task since 2020 - all these papers were using DBpedia and 6 were using DBpedia and Wikidata. DBpedia dataset were used by all the participants during the year 2020 and 2021. Three participants (37.5%) were using Wikidata in 2020 and 3 (50%) in 2021. Past participants were having good accuracy (some reaching 90%) and good NDCG and mrr (some reaching 80%). Thus, our aim in this research is to augment the repository of solutions to the AT task by proposing new methods.

To solve the AT task, SMART 3.0 provides us with two versions of large-scale dataset: one dataset is the DBpedia dataset (composed of 760 classes) and the second one is the Wikidata dataset (composed of 50K classes). Using these datasets we are proposing a solution to the AT task. This solution is based on Knowledge Graphs refinement techniques [2]. In effect, whatever the method used to construct KGs, it is known that they will never be perfect. Thus, to increase their utility, various refinement methods are proposed. These solutions are used to predict missing knowledge such as missing entities, missing types of entities, and/or missing relations that exist between entities [2]. Internal KG refinement techniques aim to complete the KG by using information hidden in the graph and external methods aim to use external knowledge sources such as text corpora to complete the graph. In this paper, we are proposing the use of internal method for category prediction and for the prediction of literal types (number, string, date) and Booleans types. For the prediction of types of resources, we are using external methods.

To implement our solutions, we needed to set up a development environment. Thus, we used:

- Operating system: Ubuntu,

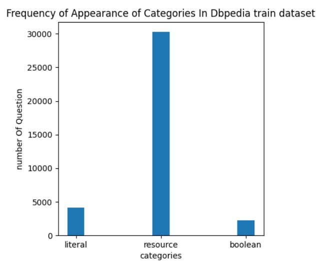

Figure 1: Data distribution for each category and literal for DBpedia

- Programming languages: JavaScript and Python,
- Python libraries: Json[4], CSV[5],
- JavaScript library: Natural[6].

The source code is available on GitHub[7].

We started this work with the analysis of the dataset (Section 2). Thereafter, we processed to the model definition (Section 3) and the model training and use (Section 4). We finalize with the conclusion (Section 5).

2. Analysis of the dataset

The aim of this step was to gather, portray and provide a deeper understanding of the structure of the dataset so as to provide an efficient solution. As presented by the organizers datasets are designed to provide a single answer category either "boolean", "literal", or "resource". They assign the "boolean" category as boolean, "literal" category into an answer type either "number", "date", or "string". For resource categories, DBpedia and Wikidata ontologies classes are used to identify the answer type.

To understand the dataset, we thought it necessary to investigate a subset of this dataset. Thus, fifty questions for each dataset were randomly selected and their annotations automatically removed. Once the annotations were removed, a manual annotation of this subset of the dataset took place. The manual analysis of the dataset gave us the insight that "resources", "literal" and "booleans" are not equally distributed in the dataset. The analysis of the distribution by category presented by the Fig. 1 for DBpedia and Fig. 2 for Wikidata confirmed this insight.

The main finding in this dataset was that:

- Data are not equally distributed amongst the different categories (see Fig. 1 for DBpedia and Fig. 2 for Wikidata);
- The question types can be divided into *Wh-*, *How many*, *Off which*, *Thus*, *Did*, etc. questions.

[4]https://docs.python.org/3/library/json.html
[5]https://docs.python.org/3/library/csv.html
[6]https://github.com/NaturalNode/natural
[7]https://github.com/jiofidelus/tsotsa/tree/SMART_22

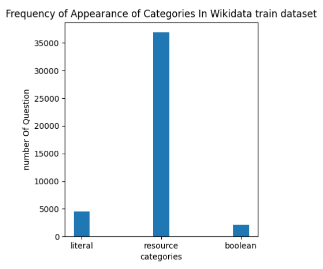

Figure 2: Data distribution for each category and literal for Wikidata

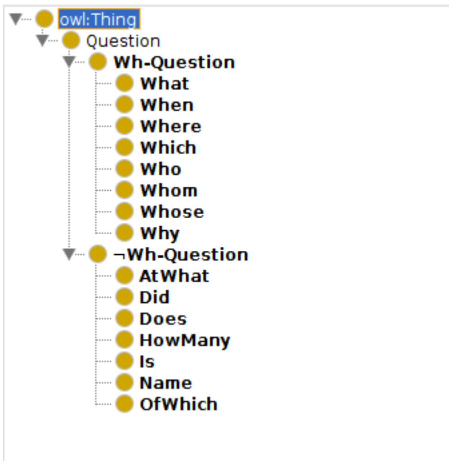

Figure 3: Taxonomy of types of questions built during the analysis of the dataset

Given the finding of the analysis of the dataset, we build the question taxonomy presented by the Fig. 3. This taxonomy will be used to build and refine the model.

3. Model definition

The aim of the model definition step was to define a model that can be applied to any dataset to solve the Answer Type prediction research problem. The first thing we did was to define the graph structure that can be used to represent any Question Answering dataset (see Fig. 4). This consists of matching the taxonomy of the Fig. 3 to the corresponding answer type and answer category. Thereafter, to match the set of questions to the question taxonomy. The idea is to say that, if a question is linked to a node in the taxonomy and this node is linked to a category and/or a question type, thus, this question is linked to this category and/or answer type. Thus, we define the *"hasEmbedding"* relation between a question and its embedding in the taxonomy. To define this relation, knowledge should be extracted from the question. Knowledge extraction is the creation of knowledge from structured (relational databases, XML), semi-structured (source code) and unstructured (text, documents, images) sources [3]. In our case, we are faced with extracting terms from unstructured sources which are questions.

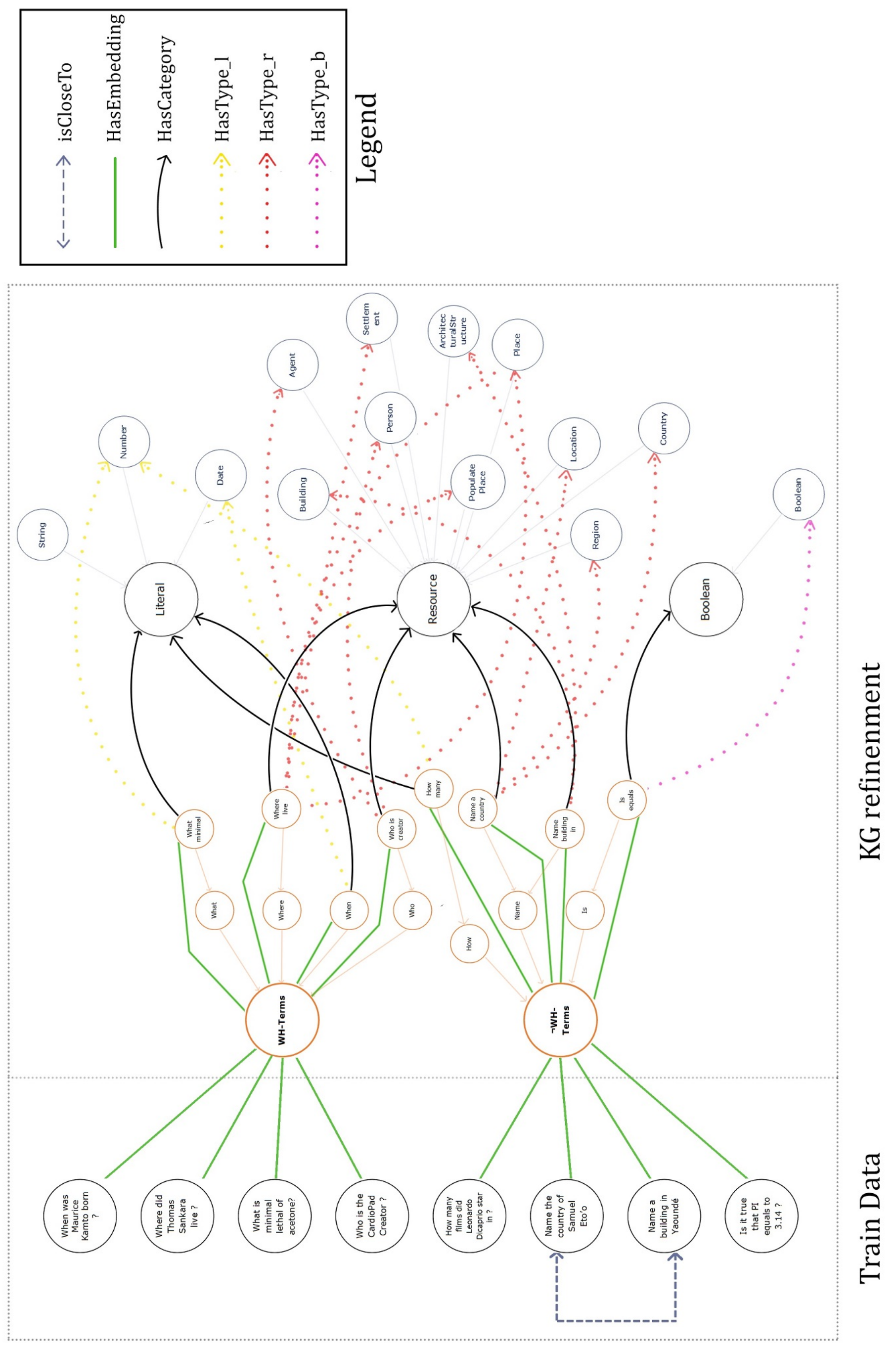

Figure 4: Knowledge graph-based representation of the dataset

From the Fig. 4, we define the Question Answering dataset as a multi-relation data using a Knowledge graph defined by the quadruple $G = (V, E, R, T)$ where:

- $V = \{v_i \in V\}$ the set of nodes. These nodes are questions, embedded vectors, questions categories, and questions types;
- $E = \{(v_i, r, v_j)\}$ the set of edges, which are triples with node v_i connected to node v_j via the relation r. These are the relations between two questions, a question and his type or his category;
- $T = \{t(v_i)\}$ denotes the node types. The type of nodes categories and nodes types are their labels and the types of node questions are their id and the title of the question;
- $R = \{r \in R\}$ denotes the relation types. These are the labels of the relations.

On the other hand, the following relations are used to link the different nodes in the graph of Fig. 4:

- ***hasEmbedding:*** this relation defines how closeness a question is with an embedding vector defined in the taxonomy of question. This is very useful because when two questions have the same embedding in the taxonomy, they will have the same category.
- ***isCloseTo:*** this defines a relation between two questions that have the same embedding. This information will be particularly important during the training of the model. In effect, two questions that are close to each other share the same category.
- ***hasCategory:*** this relation defines the relation between a node (the node can be a question or an embedding vector in the taxonomy) and its category;
- ***hasType_b:*** this relation is used to model the relation of "Boolean" category to its type, which is "boolean";
- ***hasType_l:*** this relation is used to model the relation of "Literal" category to its type, which can be "number", "string" or "date";
- ***hasType_r:*** this relation is used to model the relation of the "Resource" category to its type in the knowledge graph.

Given this new configuration of the training dataset, we reduced the problem of Answer Type prediction to the problem of KG completion. To solve this problem, our goal will be to provide an efficient tool to complete the graph by adding new facts without requiring extra knowledge.

Inspired by the TransE algorithm [4] for learning low-dimensional embedding of entities, we defined our model, presented by the equation 1. We consider that relationships are represented as translations. For instance, if (Q1 isCloseTo Q2) holds, then the embedding of $Q1$ should be close to the embedding of $Q2$. The model is defined as a function which takes a node and a relation and predicts all the nodes that are related to this relation. This prediction is based on the proximity between two nodes.

$$\begin{aligned} f_r :& V \times E \times V \longrightarrow \mathbb{R} \\ & f_r(v_i, r, v_j) = ||v_i + v_r - v_j|| \\ & \mathrm{v}_i + v_r \approx v_j \text{ if } f_r(v_i, r, v_j) \approx \beta \end{aligned} \tag{1}$$

The function f_r is used to verify if the triples in the knowledge graph holds. The triple (v_i, r, v_j) holds if $f_r(v_i, r, v_j) \approx \beta$ - in this case, we can say that $v_i + v_r \approx v_j$. β being the model parameter. If β reaches a certain value (or belongs to a certain interval of values), thus, we can say that v_i and v_j are linked with the relation r.

The model that we just defined is based on node embedding, the nodes being the questions, the questions categories and the questions types. Thus, we should find a way to model these nodes in such a way that they can be used to train our models to predict relations. This task is done by encoding the questions so as to simplify the definition of (Question, hasEmbedding, Vector) triples. We extracted knowledge from the question title and used this knowledge to define the embedding vector of each question. To this end, we suppose that a question can be well represented by some terms (word or group of words) in his title and we defined the $question2vect$ function (see equation 2) which takes a question and return a vector containing the terms used as the embedding of this question.

$$question2vect(Q) = (t_1, t_2, ..., t_n) \tag{2}$$

In equation 2, $(t_i, i <= n)$ denotes the different terms (word or group of words) in the question title that can be representative of the question. For instance, for category prediction, these words can be the "*WH-*", "*Is*", "*How many*" terms that are used to identify the category of a question.

For instance, using the taxonomy of Fig. 3, the size of the vectors returned by $question2vect$ is 15. Using this taxonomy, we provide by the listing 1 some examples of the evaluation of $question2vect$ function using some questions in the dataset. The listing 2 presents the algorithm executed to transform a question into a vector of terms.

Listing 1: Example of the transformation of some questions into a vector using the taxonomy of questions

```
question2vect(What are the awards won by the producer of Puss in Boots?)=
    (What, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
question2vect(Who are the gymnasts coached by Amanda Reddin?)=
    (0, 0, 0, 0, Who, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
question2vect(How many superpowers does wonder woman have?)=
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, How many, 0, 0, 0)
question2vect(Is Azerbaijan a member of European Go Federation?)=
    (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, Is, 0, 0)
```

Listing 2: Question2vect algorithm

```
Input:
    Q //The set of questions
    lessFrequent //The less frequent words in the train dataset
    T //The current taxonomy
Output:
    V //The vector of terms
Steps:
    1- listOfWord = tokenize(Q) //Tokenize the question to obtain the list of
                                //words it contains
    2- for w in listOfWord
         if w in lessFrequent
            delete w
    3- V = buildVector(listOfWord, T) //This method will build the vector by
                                          //replacing the empty space by 0 so that
                                 //the size of the vector correspond to |T|.
```

4. Models training and use

We used the equation 1 to define specific models for each relations.

- **Learning the "hasEmbedding" relation:** the function $f_{hasEmbedding}$ is given by the equation 3.

$$f_{hasEmbedding}(Q, hasEmbedding, V) = |Q - V| = \alpha \tag{3}$$

Globally, a node Q has embedded node the node V if $\alpha \approx \beta$ - β being the model parameter defined during the training process. The listing 3 presents the algorithm used to determine the embedding vector of a question.

Listing 3: hasEmbedding algorithm

```
Input:
    Q //The set of questions
    V //The embedding vector
Output:
    hasEmbedding=false //A boolean to define if the question Q
                    //has embedding vector V
    1) W = question2vect(Q)
    2) If ||W - V|| < beta //Verify the number of different words
                         //between W and V
        hasEmbedding = true
     else
        hasEmbedding = false
```

- ***Learning the "isCloseTo" relation:*** the function $f_{isCloseTo}$ is given by the equation 4.

$$f_{isCloseTo}(Q_i, isCloseTo, Q_j) = |Q_i - Q_j| = \theta \tag{4}$$

Globally, a question Q_1 is closed to the question Q_2 if $\theta \approx \gamma$ - γ being the model parameter defined during the training process. The algorithm 4 is used to link two questions that are closed to each other.

Listing 4: isCloseTo algorithm

```
Input:
    Q_1, Q_2: question //Two questions
Output:
    isCloseTo=false //A boolean to define if question Q_1 is close to
                    //question Q_2 or not
Steps:
    1) theta = 0
    2) Remove the special characters inside Q_1 and Q_2: \,/"#:%
    3) Put all the word inside q1 and q2 in lower case
    4) Tokenization of Q_1 and Tokenization of Q_2
    5) For each Token t_1 of Q_1
        if t_1 is inside the Token's List of q2
            theta = theta + 1
    6) If theta < gamma
        isCloseTo=true
```

- ***Learning "hasCategory":*** To predict "hasCategory", relation, we consider that each embedding node has a probability to belong to a category. The scoring function defining this probability can thus be obtained using statistical learning, Naïve Bayes, etc. For instance, if we consider that we are using statistical learning, the scoring function will be obtained by the equation 5 - to find if a node V hasCategory the node C, we simply count the number of time the node V hasCategory the node C and we divide by the number of time the node V hasCategory any other category in the training dataset.

$$f_{hasCategory}(V, hasCategory, C) = \frac{\sum_{i=1}^{N} freq((V, C), Train)}{\sum_{i=1}^{N} freq((V, *), Train)} \tag{5}$$

  Globally, an embedded vector V has category C if it has the greatest probability to have category C compared to the other categories. The algorithm is presented by listing 5.

Listing 5: hasCategory algorithm

```
Input:
    V //A vector
    C //The set of category
Output:
    c_v //The category of the vector V
Steps:
    for each c_i in C
        1) nb = number of times V has category C in the train dataset
        2) N = number of times V has category any other category in the
            train dataser
        3) beta_i = nb/n //The probability that V has category c_i
        4) if max (beta_i) //verify if beta_i is the max probability
                           //obtained
            c_v = c_i
```

- ***Learning "hasType_b", "hasType_l" relations:*** These relations are predicted using simple rules. In effect, once the category is predicted, we define a set of rules that match each element of the taxonomy to the corresponding type. The algorithms are presented by listing 6 for predicting boolean type and listing 7 for predicting literal type.

Listing 6: hasType_b algorithm

```
 //We use simple rules
Input:
    V //a vector
Output:
    boolCat //A boolean to know if the category is a boolean or not
Steps:
    if question.category = "Boolean":
        boolCat = Boolean
```

Listing 7: hasType_l algorithm

```
Input:
    C //Category of a question
    literals //List of literals
    V //Embedded vector
Output:
    t //Type of the embedded vector
Steps:
    If category(C, V) = Literal //Check if the category of the vector is
                                //a literal
        //Search for the literal in the list of literals that match with
        //the vector V
        t = matchLiteral(literals , V)
```

- ***Learning "hasType_r" relation:*** concerning the *hasType_r* relation, we combine the internal method for knowledge graph refinement with the external method. In effect, we define the embedding vector of the question, so that if it matches a resource provided by the SMART organizers, thus, this resource is saved. If not, we made a SPARQL query on the graph online in order to get the resource that holds. The algorithm is given by the listing 8 and an example of a SPARQL query for DBpedia is given by the listing 9 and for Wikidata is given by the listing 10.

Listing 8: hasType_r algorithm

```
Input:
    V: Vector //a Embedding vector of a
    T: Taxonomy //The taxonomy
Output:
    R: List<type> //List of type of the KG
Steps:
    1) Search inside the taxonomy T, which embedding vector as the same
       value of V
    2) If the vector V found in step {1} has one or multiples type value
        define inside our KG, insert them inside R List
    3) Else
        3.1) define W = ToString(V)
        3.2) Search on the external KG a entities that match W using
            Wikibase API
        3.3) With the result of step {3.2}, execute SparQL query to get
            the types associated
        3.4) insert the types found on {3.2} inside R
    4) Return R
//The ToString(V) function transforms return a String by concatenating
    each term of V
```

Listing 9: SPARQL Query of step (3.2) of Algo hasType_r for Dbpedia KG

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbr: <http://dbpedia.org/resource>
PREFIX dbo: <http://dbpedia.org/ontology>

SELECT DISTINCT ?type
```

```
    WHERE{
        <'+Search_parameter+'>  rdf:type ?type
        FILTER strstarts(str(?type), str(dbo:))
    }
//comment: Search_parameter must be replace by the entities found on
    step {3.1} of hasType_r Algo
```

Listing 10: SPARQL Query of step (3.2) of Algo hasType_r for wikidata KG

```
    query:'#defaultView:Table
    PREFIX bd: <http://www.bigdata.com/rdf#>
    PREFIX mwapi: <https://www.mediawiki.org/ontology#API/>
    PREFIX wdt: <http://www.wikidata.org/prop/direct/>
    PREFIX wikibase: <http://wikiba.se/ontology#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT DISTINCT ?type ?typeLabel
    WHERE {
        wd:'+Search_parameter+' (wdt:P279|wdt:P31) ?type.
        SERVICE wikibase:label {
            bd:serviceParam wikibase:language "en".
        }
    }
    ORDER BY ASC(?type) LIMIT 10'
//comment: Search_parameter must be replace by the entities found on
    step {3.1} of hasType_r Algo
\end{
```

To extract knowledge from questions, we used the TF-IDF technique. We identify the less frequent words in questions and we remove them to obtain the information that can be used to well represent the question. These information were used to define the $f_{hasEmbedding}$ and $f_{isCloseTo}$ models. Thereafter, we defined a function which calculates the distance between two questions. This consists of subtracting the terms that are identical and count the rest of the terms in the result. If the rest is less than β, thus, the two nodes are close.

To train the $f_{hasCategory}$ model, DBpedia and Wikidata datasets were divided into training (80%) and testing (20%). The 20% of the dataset was selected proportionally to the distribution of each data category. From the 80% of the training dataset, we removed all the answers category and answer type that were provided as response to the Answer Type prediction question.

Using the training datasets, we compute the function $f_{hasCategory}$ (see equation 5 and algorithm 5) for each category. This is done by calculating the frequency of apparition of a node embedding V classified as the category C divided by the number of apparitions of the node embedding V with any category.

Once trained on the training dataset, we applied it to the test dataset and obtained the accuracy of 0.89 for DBpedia and 0.84 for Wikidata. These values were very low compared to related work results. Thus, we decided to refine the model.

Given that the first taxonomy (see Fig. 3) did not permit to have good accuracy, we decided to refine the taxonomy. The refinement was done by augmenting the taxonomy of Fig. 3 with more terms that are used to identify the question category. To this end, the $question2vect$

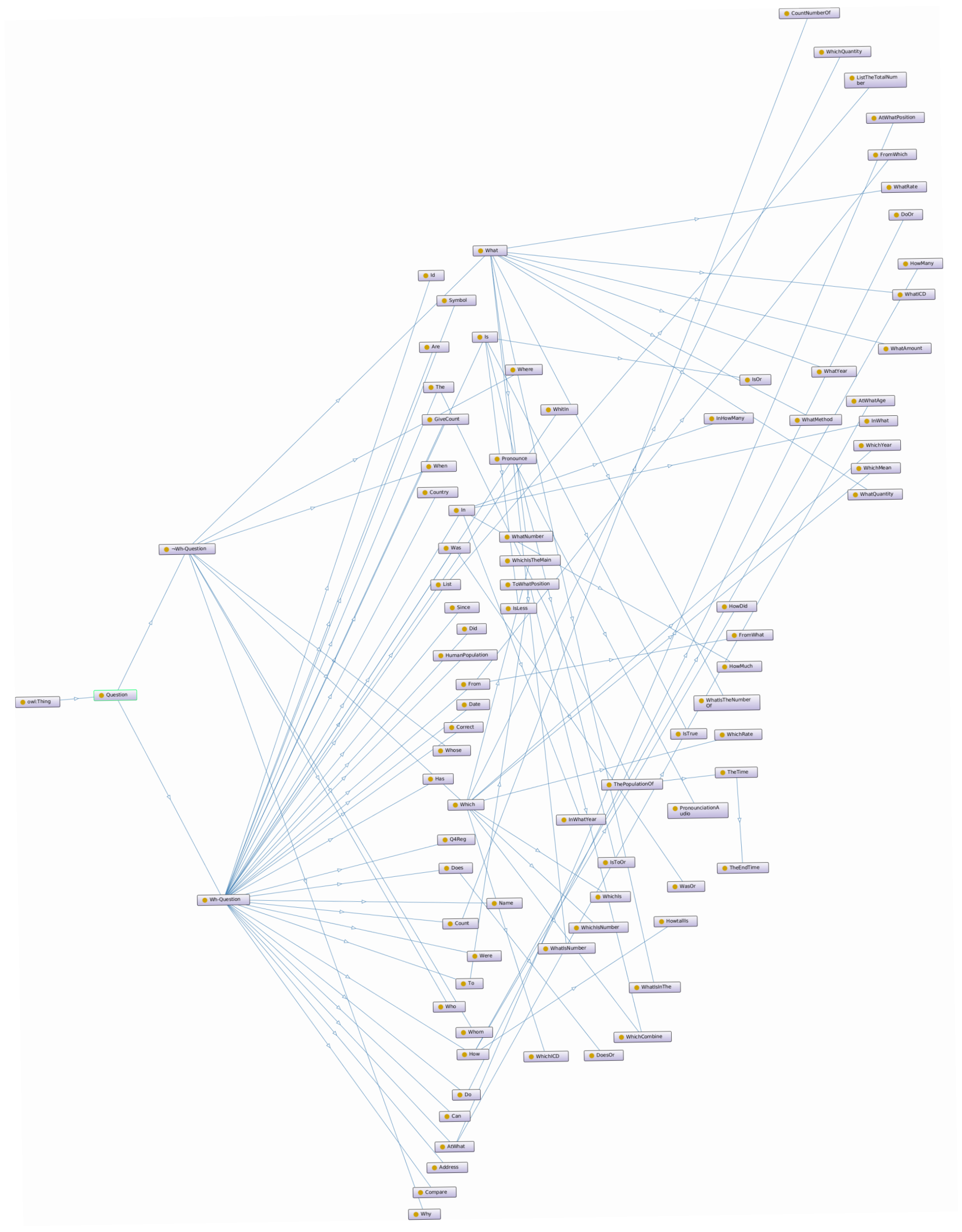

Figure 5: Knowledge graph-based representation of the dataset

Scoring function	Accuracy	Approach for resources type	NDCG@5	NDCG@10
Statistic_-WH (taxonomy = 15 classes)	0.881	Statistic_-WH (taxonomy = 15 classes) + KG matching	0.346	0.303
Statistic_-WH (taxonomy = 15 classes + other=resources)	0.890	Statistic_-WH (taxonomy = 15 classes + other = resources) + KG matching	0.400	0.357
Statistic_-WH (taxonomy = 50 classes + other=resources)	0.957	Statistic_-WH (taxonomy = 50 classes + other=resources) + KG matching	0.320	0.293
Statistic_-WH (taxonomy = 67 classes + other=resources)	0.964	Statistic_-WH (taxonomy = 67 classes + other=resources) + KG matching	0.435	0.392
Bayes	0.954	Bayes + KG matching	0.432	0.389

Table 2
Results of the Answer Type prediction on DBpedia test data

Accuracy	mrr (KG matching approach)
0.94	0.15

Table 3
Results of the Answer Type prediction on Wikipedia test data

algorithm (see listing 2) was used to obtain more larger taxonomy (see Fig. 5. This taxonomy have 67 classes.

On the other hand, we wanted to test another model than the statistical one. Thus, we choose to test the Naive Bayes classifier using this approach.

The statistical scoring function and the Bayes function was applied to the training dataset and we obtained the max accuracy of 0.95% on DBpedia and 0.93% on Wikidata.

To finalize this work, we applied our approach on the test data provided by SMART 3.0 organizers and we got our results analyzed. The result of the different models is presented by the table 2 for DBpedia and table 3 for Wikidata. These tables showed that the more the taxonomy is extended, the better the results are.

The parameter of our models (see table 2) are defined as follow:

- **Statistic_-WH (taxonomy = 15 classes)**: represents the question taxonomy of Fig. 3;
- **Statistic_-WH (taxonomy = 15 classes + other=resources)**: represents the question taxonomy of Fig. 3 in which all the others questions which cannot be embedded to the node in the taxonomy are classified as "resource" category. In effect, during the study of the dataset, we found that most questions (more than 80%) not having an embedding vector was of "resource" category;
- **Statistic_-WH (taxonomy = 50 classes + other=resources)**: this is the used of the question taxonomy extended with more classes;
- **Statistic_-WH (taxonomy = 67 classes + other=resources)**: represents the use of question taxonomy extended with more classes;
- **Bayes**: represents the use of Naive Bayes as the scoring function.

5. Conclusion

To solve the Answer Type task, we propose in this paper a model based on Knowledge Graph refinement techniques. It considers that the prediction of the answer category and answer type can be reduced to the prediction of links in a knowledge graph. This method was applied on DBpedia and Wikidata dataset provided by SMART organizers. To predict the answer category, we used two scoring functions - one built using statistical learning and the second one built using Naive Bayes. The evaluation on the test data provided by the organizers gave the max accuracy of 0.96 for DBpedia, category match of 0.94 for Wikidata. Concerning the prediction of the answer type, we combined Naive Bayes with rule-based graph matching techniques. The evaluation on the test data of DBpedia gave NDCG@5=0.435 and NDCG@10=0.392. Concerning Wikidata, the mrr=0.15. Given that these results are very low compared to the past challenges results, we are currently making a state of the art of machine learning methods and models that can be used to improve the link prediction in knowledge graphs. The results of this state of the art will be used to compare different methods and come up with the best method that can be used for link prediction in question answering systems.

Acknowledgments

We would like to thank the ISWC conference and the SMART challenge organizers and all the reviewers. This work was partially supported by Neuralearn.ai (Start-up Cameroon).

References

[1] N. Mihindukulasooriya, M. Dubey, A. Gliozzo, J. Lehmann, A.-C. N. Ngomo, R. Usbeck, SeMantic AnsweR Type prediction task (SMART) at ISWC 2020 Semantic Web Challenge, CoRR/arXiv abs/2012.00555 (2020). URL: https://arxiv.org/abs/2012.00555.

[2] P. Cimiano, H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, Semant. Web 8 (2017) 489–508. URL: https://doi.org/10.3233/SW-160218. doi:10.3233/SW-160218.

[3] A. Jiomekong, G. Camara, M. Tchuente, Extracting ontological knowledge from java source code using hidden markov models, Open Computer Science 9 (2019) 181–199.

[4] A. Bordes, N. Usunier, A. Garcia-Durán, J. Weston, O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, NIPS'13, Curran Associates Inc., Red Hook, NY, USA, 2013, p. 2787–2795.