Data-driven question answering system based on knowledge graphs

Somov O.D (oleg.somov@phystech.edu)
MIPT, Moscow, Russia

Abstract

Today most of the articles about question answering systems being published are about one side of the problem – the natural language processing part, but not about the data. Usually data is taken as it is, for example Wikipedia articles or publicly available knowledge graphs. But it is hard to build up a valid and full QA system without working with the data. In this study we are working with both sides of question answering system – data and natural language processing. The part of system connected with data is based on knowledge graphs, a representation of a structured data, which is formed from subject-predicate-object triples. The part of system that accounts for the NLP part translates a natural language question into some structured query. NLP part consist out of the following tasks - named entity recognition, entity matching, query classification. In this study we will show how to build up a multi domain question answering system. The solution is developed for two domains – movies and music. The system is built for Russian language, but the same pipeline can be applied to any language.

Key words: question answering, knowledge graphs, BERT

1 Introduction

The goal of this study is to build scalable multi domain QA system. The things that we want to consider is that same questions but with different formulations(paraphrases) should be answered in same way. The data have to be easily modified. Data structure and the way the data is navigated should be naturally explained to the people not immersed in the topic. The system should work fast and be scalable to new domains easily.

The knowledge graph is a solution for such problem. In this study we will show how to build a system that can traverse the graph by forming structured query from natural language question.

2 Knowledge graph build pipeline

Main part of this study is the knowledge graph (KG), the program is a way to navigate the knowledge graph using natural language to find answer. To implement such system, we would like to have such structure of KG, that the questions to such data structure can be rewritten as a linearized graph.

It is hard to build QA system for the open domain area of knowledge, because there is a lot of redundant, duplicated and irrelevant data in open access sources. Because of this we decide to separate open domain into close domains and begin to construct the open domain solution domain by domain. Small domains can be much easily maintained and cleaned. You can also easily add new sources of data to small domain graph, without having to do a lot of name resolution, entity matching and etc.

In this paper we are working with an open-domain knowledge graph WikiData as a base graph. It is a suitable resource, which is primarily correct and updated in time. We will extract small domain subgraph from base graph. In order for our domain graph to contain relevant, useful and demanded information about the chosen domain, it is important to understand the areas of interests that are most popular in domain and focus on the entities and predicates associated with such areas.

In our work we are focusing on two domains – films and music. A great way to find out what interests' people about chosen domains, is to offer them to ask questions about it. To get a representative info about such areas of interests we have posted an assignment to Yandex Toloka and have collected about 4000 hundred questions per domain. After collecting users' questions about music and movies domain, we have grouped questions into a number of areas, that people have interest in. We call such areas of interest – view. View is a subgraph of a graph domain, which holds prepared and aggregated information from our base graph. View subgraph is a transformed subgraph from base graph, which has only domain-relevant entities and predicates.

For example, in movie domain we picked out 16 views:

- Director or Actors details
- Film Characters
- Person films
- Popular films in country or genre
- …

For music – 24 views:

- Release info
- Upcoming releases
- Couples together in one band
- …

Now, having such info we can construct a domain graph. Views are subgraphs in KG. Entities in such subgraphs are connected in 1 to 2 hops in average, while in our base graph the amount of hops between such entities would be from 3 to 4 hops, if we did not aggregate graph nodes in subgraph structure.

We are working with huge public open-domain KG and it is possible to collect all these subgraphs with SPARQL. SPARQL is a query language which was designed to navigate RDF model data.

RDF is data which consists out of triples – subject-predicate-object. Number of SPARQL queries (from 2 to 5) can extract information that we need about one subgraph of the domain.
As mentioned before, we have picked out with Toloka the subgraphs that we need and we know the entities that build up corresponding subgraphs. Once we have all the data about subgraph, we are building a new entity in RDF model format. For example, we have built a subgraph "person films", which is on Fig.1.

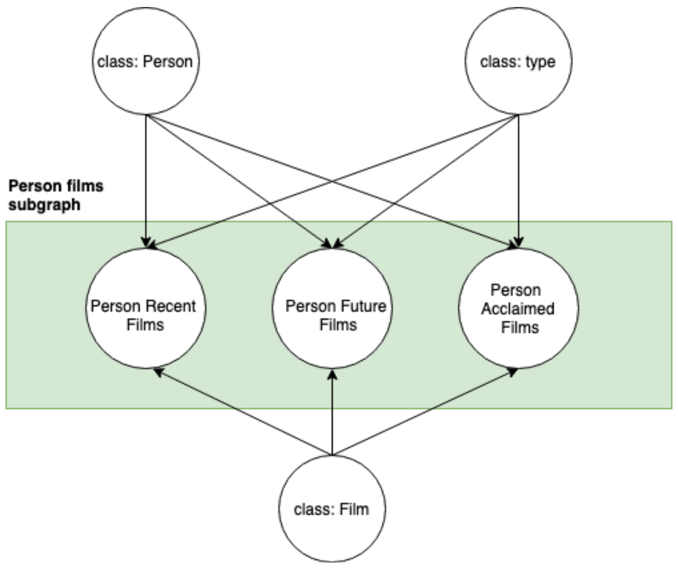

Fig. 1. View subgraph

Person films subgraph contains 3 types of nodes, each one is connected to graph entities of type class, type and person. For every Person films subgraph entity there is different person with his role(director/actor) and different set of movies.
Such graph structure helps us to later to easily build up SPARQL queries, since we already know which part of our graph we will be traversing.

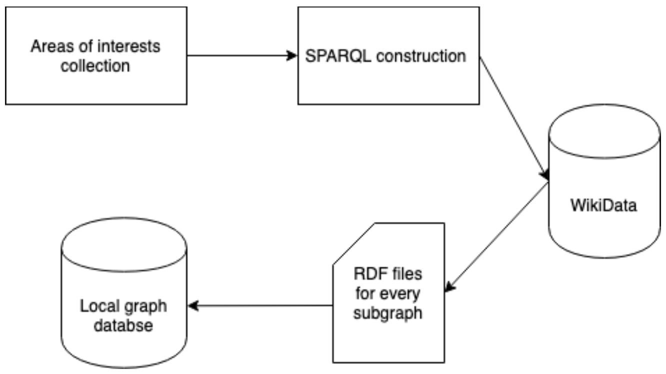

Fig. 2. Graph construction pipeline

When subgraphs that correspond to areas of user's interest are formed, we filter it. After this step we are having a structured and informative domain subgraph of base open-domain graph. We have two domains, so we are following this procedure one more time for another domain. Since we are working with WikiData ID's our two graphs are inter connected with WikiData entities and predicates and in the end, we have clean music and films knowledge graph.

3 Selection of knowledge graph storage

Knowledge graph should be stored in a graph database in order to quickly traverse the graph. There are number of graph databases in the world and most of them fall in two categories – RDF graph databases and Property Graph graph databases. The difference is in how these two graph models are structured.

RDF is presented as a triplet store. It is subject-predicate-object structure. Each node has more than one adjacent node if it is a non-terminal node. Terminal nodes are text values, which usually serve as the text description of a node and normally do not connect to more than one node.
Property graph also has nodes and predicates(connections), but each node and each connection have its own set of properties. Each node can be described with a number of other entities presented as property of a node, whereas in RDF it would be number of nodes connected to the node being described with different predicates. Due to this differences, one can conclude that RDF model is more suitable for knowledge graphs, since each piece of information is linked to another.
There are number of RDF databases – Graph DB, Open Link Virtuoso, Blazegraph, Stardog, AllegroGraph etc. All of these databases have its own pros and cons, but the thing that should be considered is that a graph database could be used efficiently as a knowledge graph storage. It means that there should be quick read from database. The write possibilities do not interest us, because our writes to system are rare.
For this project we have evaluated 3 RDF databases and benchmarked them. For evaluation of the triple stores we have used the Iguana Stress Test Framework (Conrads at al. 2017). The work gives an execution environment, which helped to measure the performance of chosen triplet stores.
Our main criterion – fast graph traversal for our SPARQL queries. We benchmarked SPARQL queries to our subgraphs and have measured the stats.
Here will be presented 3 query graph database benchmarks. The table on Fig. 3 gives us the average time, taken by graph database to return the answer. Let's give examples of a queries which we use in order to evaluate a database. Each query is a template where we vary entities and predicates.

1. What movies were directed by Quentin Tarantino?

```
SELECT ?xLabel
WHERE {
          ?x wdt:P57 wd:Q3772.
          ?x rdfs:label ?xLabel.
}
```

3.What movies will come out in winter?

```
Select ?xLabel
WHERE {
          ?upcomingFilms a dr:upcoming_films_subgraph ;
                              dr:has_film_info [ dr:has_film ?x ] ;
                              dr:has_period "winter" .
                              ?x rdfs:label ?xLabel.
}
```

4. What are the age restrictions of movie Avatar?

```
Select ?xLabel
WHERE {
          ?film a dr:film_rating_subgraph ;
                dr:has_film wd:Q24871;
                dr:has_min_age ?xLabel.
}
```

	Open Link Virtuoso	Ontotext GraphDB	Blazegraph
Query #1	**2**	4	3
Query #2	**10**	12	13
Query #3	**7**	9	11

Fig. 3. RDF databases graph traversal benchmark

Best databases to read from was Open Link Virtuoso, because of highly optimized SPARQL to SQL transformation. WikiData gives the possibility to get RDF dump, which we can process and upload to database. RDF dump is parsed by built in Apache Jena engine.

4 Question to query system overview

Now let's look at the system that can navigate knowledge graph using natural language. The idea of the system is a pipeline that will translate natural query into structured query language.
First thing there is to mention, is that since we have created our subgraphs, we do not need to resolve some of taxonomy problems, like that the extracted actor name entity is type of human and is an actor. In solving ODQA problem that would have been necessary step, but here we will be working mostly with classification problems. The approach that is presented is this paper is a sequence of classification and specifying steps, where every step helps us to build up a final SPARQL query.
The system is composed from following components:

1. **Domain classification** – it is a domain classification stage which help us to gain more insight which area of the graph we will be traversing. It will help to understand which NER model should we use to tag entities.
2. **NER tagger** – helps us to identify all entities at the beginning. Every domain has its own NER model and its own set of entities.
3. **Entities look up** - after we have extracted entities from the input question, we use matching algorithms (Cohen., 2003) to match extracted entities with knowledge graph ID. Our collections are named by the entities they keep – movies, books, genres, actors etc. By matching some entity to another, with some score above threshold, we can understand the type of extracted entity.
4. **View classification** – this step helps us to understand which subgraph in our KG we should traverse in order to find the answer.
5. **Handler searcher** – helps us to identify which path in selected subgraph we should choose.
6. **Handler** – is a parser, that extracts predicates and decides which modifying operators (COUNT, ORDER BY etc.) should we add to the result query.
7. **Answer generation** - after the response from graph database was received, the returning node value is being transformed to the nice and understandable answer with entities from the question.

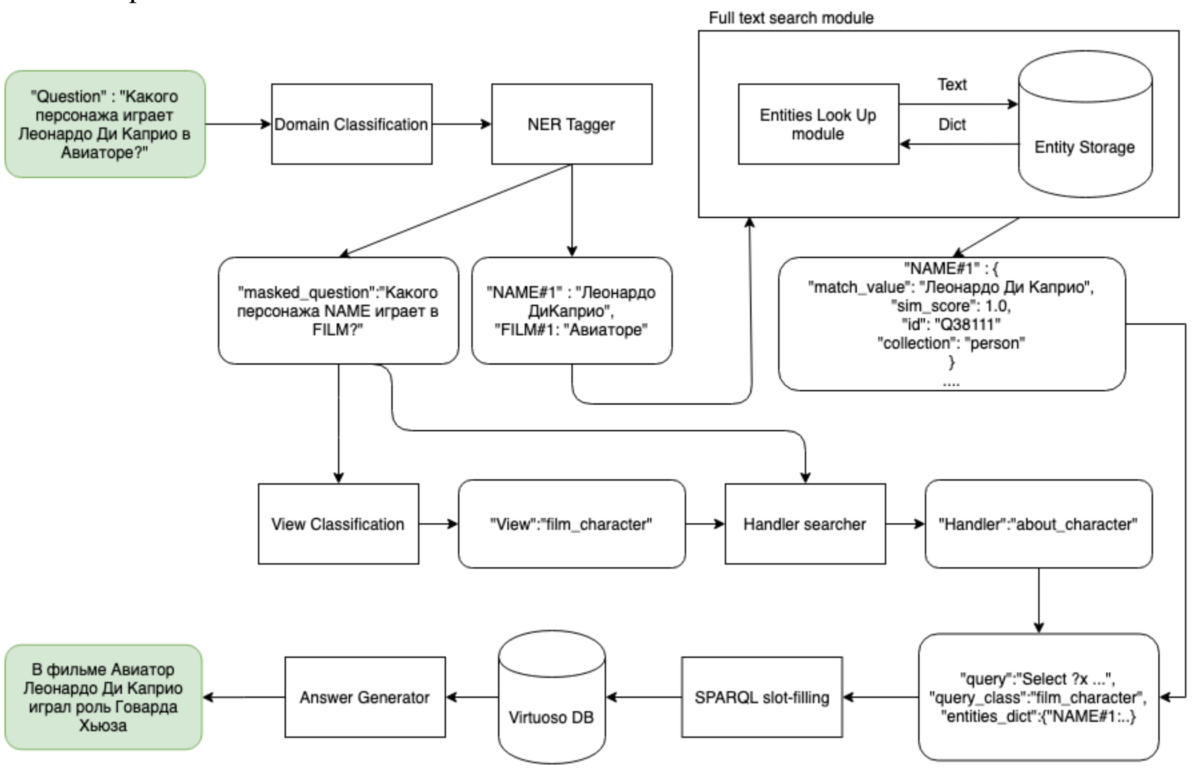
Full text search module
"Question" : "Какого персонажа играет Леонардо Ди Каприо в Авиаторе?"
Domain Classification
NER Tagger
Entities Look Up module
Text
Dict
Entity Storage
"masked_question":"Какого персонажа NAME играет в FILM?"
"NAME#1" : "Леонардо ДиКаприо", "FILM#1: "Авиаторе"
"NAME#1" : { "match_value": "Леонардо Ди Каприо", "sim_score": 1.0, "id": "Q38111" "collection": "person" } ....
View Classification
"View":"film_character"
Handler searcher
"Handler":"about_character"
В фильме Авиатор Леонардо Ди Каприо играл роль Говарда Хьюза
Answer Generator
Virtuoso DB
SPARQL slot-filling
"query":"Select ?x ...", "query_class":"film_character", "entities_dict":{"NAME#1:..}

Fig. 4. Data Flow System Diagram

Later in this work we are presenting the algorithms which solve given problem using machine learning and rule-based algorithms. In every section there will be a description of algorithms and heuristics used and also the metrics presented.

4.1 Domain classification, named entity recognition in multidomain QA system and entity matching

First step in system is domain classification. It helps us to decide which NER model to use. It is a 2-layer dense net with Adam optimizer and cross-entropy loss (binary in our case). As vectorizer, from here and on, we will be using Bidirectional Encoder representing transformers (BERT) (Devlin et al., 2018) for most NLP problems we face. The BERT we are using was trained on Russian Wikipedia, without fine tuning to any specific domain and task.

After we have decided which domain we are working with, we are using domain specific NER neural model. For every domain we are training new model with the same architecture. Each domain has its own set of entities. Films have film titles, names, genres, books, countries, characters. Music – genres, performers, countries, musical releases.

Every model has the following architecture – Bi-LSTM followed by CRF (Lample et al., 2016). We train using Adam optimizer and regularize using L2 regularization and dropout. Model is trained for maximum of 100 epochs with early stopping. The input of the model are BERT embeddings for tokens. Each model solves multiclass problem by tagging each token with one of the classes.

Bidirectional LSTM is a type of RNN, which is capable of learning long-term dependencies, where the input is given to forward and backward LSTM's to capture both sides of a context. Next a hidden layer is placed on top of Bi-LSTM and the context representation vector h_t of Bi-LSTM is projected on the hidden dense layer. This layer produces a score matrix which is passed to CRF loss function.

Such multilabel approach proved to be much more efficient than having one multiclass model, which has a huge number of classes. For the domain dataset with 10k samples, where each class has from 300 to 3000 samples we archive the macro F1 score of 0.73. In comparison, the multiclass model had the F1 score on the same data 0.62. With our experiments, we can conclude that it is a working approach, to do NER tagging by training domain models instead of doing one multiclass model. Other extracted entities – year, season, movie types etc. are handled by rule-based algorithms.

After we have tagged all the named entities in the input question we are passing named entity dictionary to the entity matcher module, where we match entities to the database entity ID's.

We have multiple collections of different entities – person, films, releases etc. Each collection we vectorize with BERT model. We vectorize input query and using KNN algorithm we get **N** closest candidates in the collection above some **T** threshold. Then we specify the closest entity with **specific string distance algorithm**. Every collection has own string distance algorithms, like film collection is better searched over with Levenstein algorithm while name collection is better with Jaro-Winkler algorithm.

N, **T** and **string-matching algorithm** are the parameters which have to be picked individually for every collection of entities. The resulting entities then ranked based on hand-picked heuristics like wiki views count. It is useful, since for example there are two films named "Titanic", one of 1953 and the famous one of year 1997. When the year is not specified, we want to give out the more popular one. The problem that we face here is the unequivocal search of entities. We always search for the entity that is close to the extracted one by string distance independently from other entities in question, without taking in account the relationship between entities inside question. For example, "Who played Jean Valjean in Les Misérables?" we would not be able to answer such question, because there are multiple "Les Misérables" and if we get the most popular one, we would not find there the character by a name Jean Valjean. That problem will be tackled in our future work.

4.2 Subgraph classification

Next step is in question answering system is to determine the subgraph of a knowledge graph that we need to look into in order to find out the answer to the question. This is a text classification task. We have 40 classes, each class(subgraph) contains question paraphrases to this subgraph. The thing to notice is that some questions in some subgraph look alike with other questions from different subgraph.

We use BERT embeddings for masked questions, and also, we add NER features and domain features to our sentence embeddings in order to better distinguish between classes. In this part we will describe the method which worked the best for our problem.

Best approach for this task is proved to be metric learning (Schroff et al., 2015). In this approach, we will learn a non-linear function with a DSSM net, denoted as $f(v)$ and new embeddings for out input vectors $v_1, \dots, v_n$ and then arrange these embeddings in a R^M, where M is the dimension we want to operate in.

When a new sample v_t comes, we will get its BERT embedding, pass it through a learned function $f(x)$,get a new $\tilde{v_t}$ and then using k Nearest Neighbors algorithm we will classify the input sample into one of 40 classes.

The formal problem statement is to maximize the interclass distance d_m, whilst keeping the intraclass distance below some margin.

$$\max \sum_{(x_i, x_j) \in D} d_M(x_i, x_j) \tag{6}$$

$$s.t. \sum_{(x_i, x_j) \in S} d_M(x_i, x_j) \leq \alpha \tag{7}$$

The logic of a model – pass 3 samples into the network – anchor sample(A), positive sample(P) and negative sample(N) and if the anchor sample and positive samples are far from each other, whilst anchor and negative are close, tune the model weights. Positive sample is another sample from the anchor class. Negative sample is sample from a class different from anchor class. We will be using triplet loss function and as p we will be using cosine measure.

$$Triplet\ loss = \max\left(p(v_{anchor}, v_{positive}) - p(v_{anchor}, v_{negative}) + \alpha, 0\right) \tag{8}$$

$$F(x, y) = \frac{(x, y)}{||x|| * ||y||} \tag{9}$$

We implement this model by using DSSM neural network with two hidden layers. DSSM neural network is network where the architecture of all nets is the same and the weights are shared. In this article we are using metric learning approach with a triplet loss function, so we have three inputs to neural net – anchor vector, positive vector and negative vector.

On Figure 4 we are building the input vectors into our model, and on figure 5 present the architecture for the triplet loss DSSM neural net.

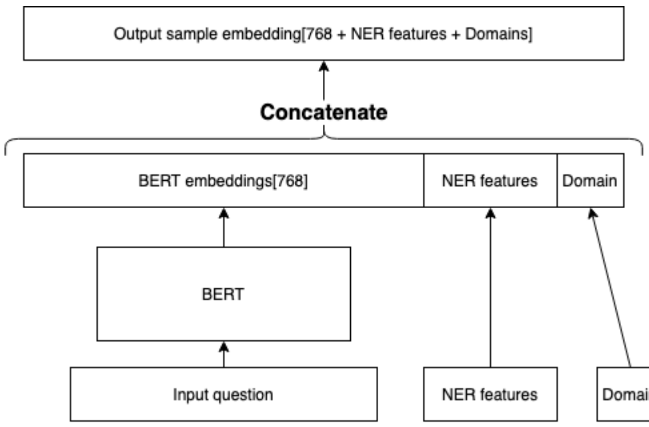

Fig. 5. Creation of input vector

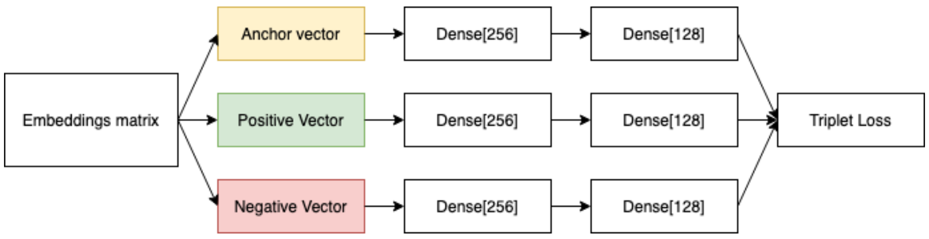

Fig. 6. DSSM Model Architecture

We want to create such metric space, where the classes are separable from one another. But in order for the neural net to separate classes, it must learn how they differ from one another, even if in initial vector space they are close to each other. The solution is hard negative sampling.
We should sample from embedding matrix on each iteration 3 samples – two of one class and one from another. The embeddings from one class are close to each other by the cosine measure to each other and from different classes they are far apart. When we are sampling from embedding matrix we are extracting anchor vector and randomly extracting positive sample from the same class, but as negative sample we are getting the closest vector by the cosine measure from another class. On the picture we can see the difference of the embeddings arrangement after t-SNE dimensionality reduction in two-dimensional space. Metric learning embeddings started to form clusters from the classes, while simple BERT embeddings with features are not simply separable.

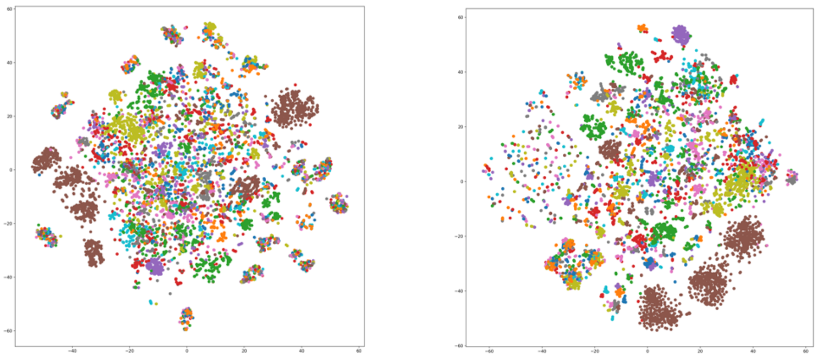
Fig. 7. BERT (left) and metric learning (right) embeddings

The benefit of such model is that we do not need to train a large neural net and retrain it with new samples. Also, the network itself is not big and therefore is stiff to the overfit.
We have the following metrics on our test dataset:

Metrics	**Scores**
Accuracy	0.93
Recall	0.89
Precision	0.92
F1-score	0.91

Fig. 8. Classification metrics

4.3 Handler searcher & Query modification

After we know which subgraph holds the answer to input question, we need to understand which path in graph corresponds to input question. Each subgraph contains from 1 to N handlers. Each handler is responsible for its path in subgraph. For every path in graph, there are a number of natural language question paraphrases. After input question was classified into some subgraph, we decide to which path in this subgraph we can relate input question. This is done with a KNN algorithm on BERT embeddings. The path from the masked question to final SPARQL is the following:

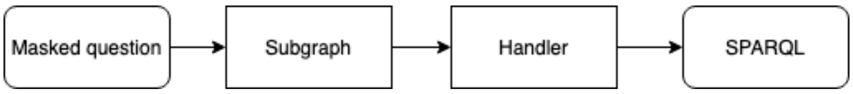

Fig. 9. From masked question to SPARQL query path

When we know which handler corresponds to the input question, we are going to construct a query. The query itself is a path, which leads to answer, based on the initial information we have inferred from the input question. We can see the example on the picture.

```
Select ?height Where
?entity wdt:typeOf wd:mountain;
        wdt:locatedIn wd:Spain;
        wdt:height ?height.
}
ORDER BY DESC(?height)
LIMIT 1
```

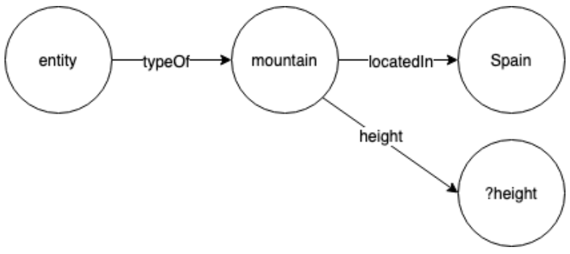

Fig. 11. On the left is SPARQL query for the question "What is the highest mountain in Spain? and to the right is path of graph the query is building

The template of a query is known, because we have classified the subgraph, we are traversing, so its structure is known, and in this step, we are going to modify the template query with predicates, select type, date calculations and etc.
As we discussed in Graph Construction, we have collected dataset of possible questions to the subgraph. Using context-free grammar we can build such rules, which will identify, which modifications to the template query we should do. In dataset we have a lot of samples, and details that modify query in question are finite and the number of paraphrases of such details is very low. Here we will give the things that we do using free grammar and regular expressions:

- Predicate selection – we know the kind of entity we have and the predicates to such entity are known. We just have to understand which predicate is mentioned in question. To understand it, we are using context-free grammar.

- Switch Select statement to Ask – Some questions do not particularly ask for some node value, sometimes the question only intended to receive Boolean value of yes or no. Same as in previous point we are using context free grammar and regular expressions.
- Age calculation
- Revert triples – revert from S-P-O structure to O-P-S
- Modification words – COUNT, ORDER BY.

These intentions are easy to capture with rule-based algorithms, so we use such approach.

5. Overall system metrics and comparison to similar systems

There are not so many QA based on knowledge graphs for Russian language. Most famous one is KBQA from DeepPavlov (Burtsev, 2015) laboratory, which focuses itself on answering open-domain questions. We are going to compare our system to the DeepPavlov KBQA.

We have asked the scribes of Yandex Toloka to ask any question about movie and music domain and have collected the dataset of 250 movie questions and 250 music question. We check the quality of a QA system by two points:

1. Correct intent but wrong answer – it means that overall QA pipeline did work and the answer is of the expected type. For example, the question was about person and the answer contains relevant information or the question was about some number (date, budget) and answer of expected type is returned.
2. Correct answer – how many questions were correctly answered by the system.

System name	Correct intent but wrong answer	Correct answer
DeepPavlov KBQA	0.35	0.15
ours	0.74	0.59

Fig. 12. Systems comparison

DeepPavlov KBQA performs worse than the presented system, because of the following:

- We are building open-domain system domain by domain and currently have 2 domains, while DeepPavlov solutions tries to find answers in a whole WikiData graph.
- DeepPavlov finds only one-hop questions answers, while our system can process more complex queries because of the subgraph logic, handler classification and handler mechanism.
- In our study we also work with the graph structure and internals, which helps us to reduce natural language processing.
- We add flexible handler mechanism that helps us to extract predicates from the question and add modifying words to final SPARQL (like ORDER BY, COUNT).

The thing that are work has in common is the intent classification in a question. DeepPavlov KBQA does it by classifying question predicates, we propose a more fine-grained approach by a two-level classification and handler mechanism, but the thing we both struggle to resolve are questions which have implicit entity naming:

- When was born the director of Titanic (James Cameron)?
- How tall is the king of beasts (Lion)?
- Who played Jean Valjean in Les Misérables? – There are multiple works which go by the name of Les Misérables and in order to give a right answer to this question, we have to know which work the user talks about.

This problem requires a more complex entity matching task to be solved, since we have to match entities that somehow connected to another entity in question. Also, we should work more with syntactic and morphological features in order to obtain the relation type between entities to resolve this case.

A lot of errors that both system encounter are caused by missing data in our knowledge graph. We are working with movies and music domain and most of the errors that we make are associated with the fact that the questions, that we have collected on Yandex Toloka, mention such entities, that are known in Russia, but they are not presented in WikiData knowledge graph.

6. Conclusion and future work

The authors have gone a long way of creating an information retrieval system and have gained a lot of insight how to approach QA problems. In this article, we have presented a pipeline for creating QA systems for close domains and presented the related algorithms which help to solve problems that we face. We explored the tasks associated with a natural language processing as well as with a data itself. We made use of open domain knowledge graphs and showed how to use them in constructing such systems. We have studied graph databases and chose the most appropriate one for knowledge graphs using graph databases relevant benchmarks. We encountered and studied the problems that will have to be solved in future in order to build a qualitative question answering system.

Literature List

1. Adel Tahri, Okba Tibermacine (2013) DBPedia based factoid question answering system // International Journal of Web & Semantic Technology (IJWesT) Vol.4, No.3
2. Christina Unger, André Freitas, Philipp Cimiano (2014) An introduction to Question Answering over Linked Data // In Proceedings of the 2014 Reasoning Web Summer School
3. William W. Cohen, Pradeep Ravikumar, Stephen F. Fienberg (2003) A comparison of string distance metrics for Name-Matching Tasks // IIWeb. – 2003. – T. 2003. – C. 73-78.
4. Daniel Hernández, Aidan Hogan, Markus Krötzsch (2015) Reifying RDF: What Works Well With Wikidata, ISWC, USA, pp. 32-47
5. Daniel Jurafsky, James H.Martin (2018) Speech and Language Processing. 558 c.
6. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018) Bert: Pretraining of deep bidirectional transformers for language understanding, Available at: arXiv:1810.04805
7. Felix Conrads , Jens Lehmann , Muhammad Saleem , Mohamed Morsey, Axel-Cyrille Ngonga Ngomo (2017), Iguana: A Generic Framework for Benchmarking the Read-Write Performance of Triple Stores, Available at: https://svn.aksw.org/papers/2017/ISWC_Iguana/public.pdf
8. Florian Schroff, Dmitry Kalenichenko, James Philbin (2015), FaceNet: A Unified Embedding for Face Recognition and Clustering, Available at: arXiv:1503.03832
9. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. Available at: arXiv:1603.01360.
10. Juan Luis Suárez, Salvador García, Francisco Herrera (2019) A Tutorial on Distance Metric Learning: Mathematical Foundations, Algorithms and Software, Available at: arXiv: 1812.05944
11. Khriyenko Oleksiy, (2019), RDF data Serialization and Storing, Available at: http://users.jyu.fi/~olkhriye/ties4520/lectures/Lecture02.pdf
12. Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, Gustavo Publio (2018) Neural Machine Translation for Query Construction and Composition, Available at: arXiv:1806.10478
13. Wei Shen, Jianyong Wang, Jiawei Han (2014) Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions, IEEE, Vol. 27, pp. 443-460
14. Yuqing Gao, Jisheng Liang, Benjamin Han, Mohamed Yakout, Ahmed Mohamed (2018) Building a Large-scale, Accurate and Fresh Knowledge Graph, KDD, UK, available at: https://kdd2018tutorialt39.azurewebsites.net/KDD%20Tutorial%20T39.pdf

15. Burtsev M. et al. DeepPavlov: Open-Source Library for Dialogue Systems //Proceedings of ACL 2018, System Demonstrations. – 2018. – C. 122-127.