Semantic Parsing for Multiple-relation Chinese Question Answering

Jingyao Zhang, Yuncheng Hua, Guilin Qi, and Daiqing Qi

Cognitive Intelligence Lab
Southeast University, Nanjing, China
{230179123, 220174102, gqi}@seu.edu.cn

Abstract. Chinese Open-domain Question Answering is a task of CCKS 2018. For a given Chinese query, the question answering system selects several entities or attribute values of the given knowledge base as the answer to the question. In this paper, we propose a semantic parsing based method to answer the simple and complex questions. This system resolve the ambiguity of natural language questions when matches of query are found. It is easy to expand and more suitable for open field questions. The experiment shows that our system is effective against CCKS-2018 dataset, achieving averaged-F1 measure of 0.5693 which ranked third.

Keywords: Entity linking · Similarity calculation · Subgraph matching · Syntactic dependency tree.

1 Introduction

As more and more structured data become available on the web, factoid question answering (QA) over knowledge bases such as Freebase has been intensively studied recently. Answering a question needs to require references to multiple related facts in knowledge bases or reference to a single fact. Generally, there are two stages in traditional semantic parsing based QA systems[1]: question understanding and query evaluation. Existing systems translate a natural language question into SPARQLs in the first stage, and in the second stage evaluate all SPARQLs translated in the first stage.

In this shared task, Chinese Open-domain Question Answering(COQA) introduced by CCKS 2018, for a given Chinese question, the question answering system selects several entities or attribute values from the given knowledge base as the answer to the question. The questions are objective facts and do not include subjective factors. Sub-tasks such as entity identification and relation extraction are required in the process of understanding and answering questions.

In this paper, we propose a method for the COQA shared task. We divide the problems into two categories: simple and complex problems. Simple problem means that the question could be mapped to one single triplet. Entity linking and similarity calculation are key steps in the system's processing. Meanwhile, complex problems usually require multiple triplets to be combined in order to get the final answer. The system will first perform a syntax tree analysis, and then gradually find the final answer based on the subgraph matching.

2 Methods

2.1 Simple questions

Process overview Figure 1 gives an overview of our method for the simple question " '商鞅变法' 发生于哪个时代?" .we have five main steps: (1)Identify the entity mention in the question; (2)segmentation of other parts of the question; (3)get all the triples that contain the entity through web interface; (4)calculate the similarity between predicates and the results of segmentation; (5)sort those triples based on similarity results to return the correct answer;

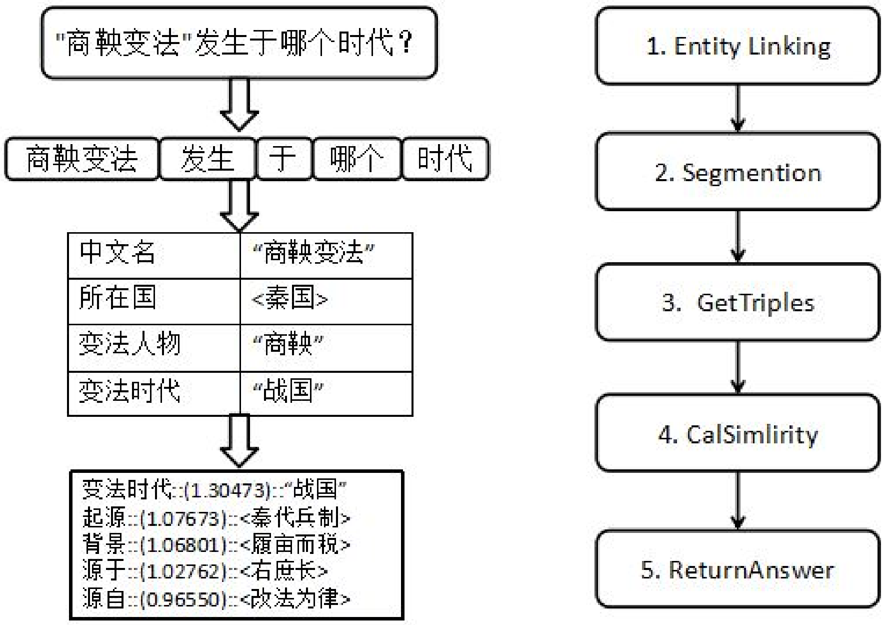
"商鞅变法"发生于哪个时代?
商鞅变法
发生
于
哪个
时代
中文名
"商鞅变法"
所在国
<秦国>
变法人物
"商鞅"
变法时代
"战国"
变法时代::(1.30473)::"战国"
起源::(1.07673)::<秦代兵制>
背景::(1.06801)::<履亩而税>
源于::(1.02762)::<右庶长>
源自::(0.96550)::<改法为律>
1. Entity Linking
2. Segmention
3. GetTriples
4. CalSimlirity
5. ReturnAnswer

Fig. 1. The process of our system for simple questions.

Entity linking Given a question, the entity linker provides a set of top-N entity candidates.In our system,we use passive entity linker[2]:First search for entity candidates by all question words, then use returned entities to guide the mention detection; We perform entity linking by deriving the longest consecutive common subsequence (LCCS) between a question and entity candidates and refer to it as δ. Given a question q and all entity names from PKU-base, we perform the following 3 steps:

1. Use each component word of q to retrieve entities whose names contain this word. We refer to the set of all these entities as C_e.

2. For each entity candidate e in Ce, compute its LCCS δ with the question q. Let p be the position of the last token of δ in q.
3. Compute $a = |\delta|/|q|$, $b = |\delta|/|e|$ and $c = p/|q|$ where $|\cdot|$ is length in words. Finally, entity candidate e is scored by the weighted sum

$$S_e = \alpha a + \beta b + (1 - \alpha - \beta)c \tag{1}$$

   Parameters α and β are tuned on training dataset. Top-N ranked entities are kept for each question.

Similarity calculation After recognizing the entity in the question, we need to get all the triplets **Ts** including the entity, then remove the entity in the query sentence and segment it into tokens **ts**.
(1) The first step in calculating the similarity between predicate in **Ts** and tokens in question is to count the frequency of all triplet predicates f(w);
(2) Then calculate the similarity S between the words of predicates and tokens. If the predicate is less than two words, the following calculation has return value:

$$S = sim_{word} = \frac{\sum_{i=1}^{n}(Ts_i * ts_i)}{\sqrt{\sum_{i=1}^{n}(Ts_i)^2} * \sqrt{\sum_{i=1}^{n}(Ts_i)^2}} \tag{2}$$

If the predicate such as "变法时代" consists of more than two words, meanwhile there is no return value by using the previous calculation, the predicate is then segmented to get sub-words. Calculate the similarity between the results of segmentation and question tokens. The calculation function is as follows:

$$S = \frac{1}{subsize} * \sum_{i=1}^{subsize} sim_{subword} \tag{3}$$

(3) Compute the distance D between the predicate and the token:

$$D = distance = \sum_{i=1}^{n} EditDistance(Ts_i)(ts_i) \tag{4}$$

(4) Similarity and distance summation as sorting indicators F:

$$F = \frac{a}{(f(w) + a)} * (D + S) \tag{5}$$

Hyper parameter **a** is tuned on training set. Using the frequency of predicate word in all triples f(w) and **a** can effectively improve the calculation effect of combined words[3].

2.2 Complex questions

Question structures Figure 2 shows that there are mainly three kinds of complex questions: (1) Multi-hop problems;(2) Multi-entity problems;(3) Multi-predicate problems. We use syntax dependency tree to syntactically analyze complex questions, and then obtain the final answer by subgraph matching in the knowledge base[4].

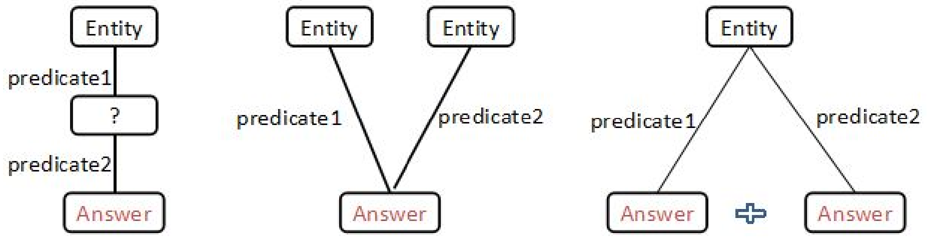

Fig. 2. Structures of complex questions.

Syntax dependency tree The syntactic structure of a sentence is described solely in terms of the words (or lemmas) in a sentence and an associated set of directed binary relations that connect the words[5]. It also contains a root node that explicitly marks the root of the tree,which is the head of the entire structure. Through generating a syntactic dependency tree, we can break a complex question into some simple questions to answer.

Processing flow: (1) We also need to find the entity in the question first, and then (2)backtrack to the root node according to the position of the entity in the syntax dependent tree until the child node of a node has "的" or it is a predicate, then the backtracking is stopped. (3)Take these nodes and their children as a sub-branch. (4) Answer this sub-branch as a simple question, replace this branch with the answer, and make a second round of answer. If the remaining syntax tree is an empty set after removing the stop word and the wh-word, the problem is treated as a simple question, and only one round of processing is enough. Figure 3 shows a typical situation of complex problem.

"Multi" rules : (1) Identifying the entity mention in the query first. If the mention such as "中华人民共和国" contains "和", the current rules do not apply. (2) Otherwise, the query may contains two entities or two predicates. Perform NER on the left and right clauses of "和" to see if two entities can be found; if yes, proceed to step 3, otherwise, proceed to step 6;(3) If there are two entities, such as "成龙" and "周星驰", then find all relevant triples with these two entities to form two sets of triples;(4) perform the Cartesian product on these two sets of triples to obtain the pairwise match. If the predicate and the object of the pair of triples are the same, it is regarded as a candidate ternary.(5) Calculate the similarity (vector similarity combined with text distance) by using the predicate of all candidate triples with the tokens of the non-entity part. Then sorts and returns the answer;(6) If only one entity can be found, and the sentence is like "郭沫若的原名和字号是什么？", then the entity "郭沫若" and left predicate are

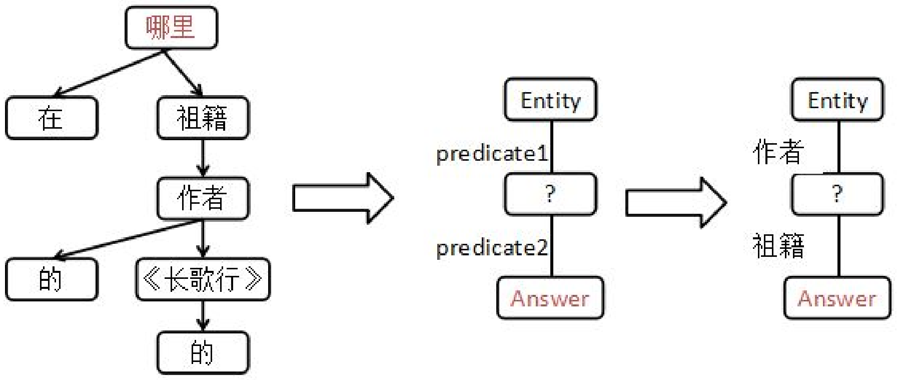

Fig. 3. Examples "《长歌行》的作者的祖籍在哪里？" for multi-hop questions.

composed of a sub-problem "郭沫若的原名", Meanwhile the entity "郭沫若" and the predicate on the right side of "和" are the composition of the question "郭沫若字号是什么？". Calculate the answers to the two sub-questions separately, merge the sub-answers and return. If no answer is found, try to answer it as a complex query.

3 Experiments

We evaluate our system in CCKS2018 shared task four, COQA. 1200 questions with corresponding SPARQLs and answers are provided as training data, while another 800 questions are treated as validation data and test data. The training and testing procedures are carried out on PKU-base. The evaluation indicators for this task include Macro Precision, Macro Recall, and Averaged F1. The final ranking is based on the Averaged F1 value. Let Q be the problem set, A_i is the answer set given by the player to the i-th question, and G_i is the standard answer set of the i-th question. The relevant calculation formula is as follows:

$$MacroPrecision = \frac{1}{|Q|}\sum_{i=1}^{Q} P_i \quad P_i = \frac{|A_i \cap G_i|}{A_i} \tag{6}$$

$$MacroRecall = \frac{1}{|Q|}\sum_{i=1}^{Q} R_i \quad R_i = \frac{|A_i \cap G_i|}{G_i} \tag{7}$$

$$AvagedF1 = \frac{1}{|Q|}\sum_{i=1}^{Q} \frac{2P_iR_i}{P_i + R_i} \tag{8}$$

Table 1 shows the proportion of simple and complex problems in the data set. To deal with simple problems, the correct rate of the system is about 72%.

Table 1. Proportion of simple and complex problems.

	simple	**complex**
training	0.49	0.50
validation	0.50	0.50
test	0.50	0.50

Table 2. Results on different dataset.

	Validation			*Test*		
	Precision	*Recall*	*AveragedF1*	*Precision*	*Recall*	*AveragedF1*
simple	0.713	0.726	0.719	0.709	0.732	0.721
simple+f(w)	0.741	0.758	0.751	0.743	0.770	0.758
complex	0.305	0.347	0.327	0.316	0.342	0.332
complex+multi	0.341	0.382	0.356	0.345	0.386	0.364
total	0.546	0.592	0.563	0.556	0.612	0.569

If the word frequency information about the predicate is used, which is marked as “simple+f(w)” in table 2, the averaged F1 can be increased by about 0.3. After considering the word frequency information, the result of the similarity calculation can be effectively improved. For complex problems, our correct rate is only about 30%. If multi-entity and multi-predicate rules are added, which is marked as “complex+multi” in table 2, the correct rate can be further increased by 5%. Our final averaged F1 has a third place with 0.569.

References

1. Zhu, C., Ren, K., Liu, X., Wang, H., Tian, Y., & Yu, Y.: A Graph Traversal Based Approach to Answer Non-Aggregation Questions Over DBpedia. In Joint International Semantic Technology Conference (pp. 219-234). Springer, Cham(2015).
2. Yin, Wenpeng, et al.: Simple question answering by attentive convolutional neural network. (2016).
3. Arora S, Liang Y, Ma T.: A simple but tough-to-beat baseline for sentence embeddings[J]. 2016.
4. Hu, S., Zou, L., Yu, J. X., Wang, H., Zhao, D.: Answering natural language questions by subgraph matching over knowledge graphs. IEEE Transactions on Knowledge & Data Engineering, PP(99), 1-1.(2018).
5. Marnee, M. C. D., Manning, C. D.: Stanford typed dependencies manual.Technical report, Stanford University(2010).