PERQ: Predicting, Explaining, and Rectifying Failed Questions in KB-QA Systems

Zhiyong Wu† Ben Kao† Tien-Hsuan Wu† Pengcheng Yin* Qun Liu §
†The University of Hong Kong, *Carnegie Mellon University §Huawei Noah's Ark Lab, Hong Kong, China
{zywu,kao,thwu}@cs.hku.hk pcyin@cs.cmu.edu qun.liu@huawei.com

ABSTRACT

A *knowledge-based question-answering* (KB-QA) system is one that answers natural-language questions by accessing information stored in a knowledge base (KB). Existing KB-QA systems generally register an accuracy of 70-80% for simple questions and less for more complex ones. We observe that certain questions are intrinsically difficult to answer correctly with existing systems. We propose the PERQ framework to address this issue. Given a question q, we perform three steps to boost answer accuracy: (1) (Prediction) We predict if q can be answered correctly by a KB-QA system S. (2) (Explanation) If S is predicted to fail q, we analyze them to determine the most likely reasons of the failure. (3) (Rectification) We use the prediction and explanation results to rectify the answer. We put forward tools to achieve the three steps and analyze their effectiveness. Our experiments show that the PERQ framework can significantly improve KB-QA systems' accuracies over simple questions.

ACM Reference Format:
Zhiyong Wu, Ben Kao, Tien-Hsuan Wu, Pengcheng Yin and Qun Liu. 2020. PERQ: Predicting, Explaining, and Rectifying Failed Questions in KB-QA Systems. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371782

1 INTRODUCTION

A *knowledge-based question-answering* (KB-QA) system answers natural language questions with the information stored in a knowledge base (KB). Most existing KB-QA systems are powered by curated KBs (e.g., Freebase). A curated KB is manually created, in which factual knowledge is encoded in entities and relations with well-structured schemas using the Resource Description Framework (RDF). Figure 1 shows a snippet taken from Freebase that centers at an entity *Survival* (a 2006 movie). A KB-QA system processes a question through a number of steps. These include parsing the question to identify named entities and relations; locating relevant resources in a KB; composing a structured query; and ranking answers.

KB-QA is a very active area of research. This is evidenced by the numerous KB-QA systems invented in the past few years, which use different sophisticated techniques to improve the various steps of the QA processing pipeline. Table 1 summarizes 12 KB-QA systems

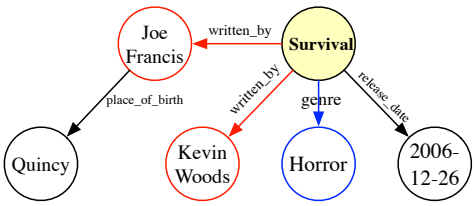

Figure 1: A Freebase snippet

QA Systems	Accuracy	QA Systems	Accuracy
Bordes et al. (2015) [4]	62.7	Yu et al. (2017) [23]	77.0
Bao et al. (2016) [2]	72.8	BuboQA (2018) [14]	74.9
Golub and He (2016) [8]	70.9	AR-SMCNN (2018) [16]	77.9
CFO (2016) [6]	75.7	Hao et al. (2018) [9]	80.2
AMPCNN (2016) [22]	76.4	APVA-TURBO (2018) [18]	81.5
Lukocnikov et al. (2017) [12]	71.2	AEQA (2019) [10]	74.9

Table 1: Answer accuracies(%) of some KB-QA systems.

that were published in the past 5 years (2015-19). The numbers displayed in the table show the accuracies of the systems when they are evaluated with the SimpleQuestions (SimpleQ) [4] benchmark. From the table, we see that the accuracies of existing systems are largely limited to the range of 70-80%. In particular, there are questions that are intrinsically difficult to answer. Our objective is to discover and to understand these questions. The knowledge obtained allows us to better apply existing KB-QA systems and to improve their accuracies.

We propose a prediction-explanation-rectification (PERQ) framework that addresses the following questions: (1) What are the common features shared by the "difficult questions" (those that are not answered correctly by most (if not all) KB-QA systems)? (2) If these features could be identified, is it possible to predict whether a system is likely to fail to answer a question? (3) If such a prediction can be made, can we explain how we arrive at the prediction? (4) Can we devise actions that rectify question processing in search of a correct answer? The major contributions of this paper are:

- We formulate the problem of predicting if a KB-QA system S would correctly answer a question q as a classification problem. We design a *predictor* tool for the classification task. We implement and install the predictor in three representative KB-QA systems and evaluate the predictor's performance. The predictor's accuracy ranges from 87.8% to 89.6%, which shows that it is highly accurate.
- We design an *explainer* tool. Given a system S that is predicted to fail to answer a question q, the explainer identifies *culprit features*, which are those of S and q that sway the prediction to "S fails q" the most. By identifying and analyzing culprit features, we are able to discover the weaknesses of existing KB-QA systems.

WSDM '20, February 3–7, 2020, Houston, TX, USA

ACM ISBN 978-1-4503-6822-3/20/02…$15.00
https://doi.org/10.1145/3336191.3371782

• We design two *rectifier* tools that improve KB-QA systems' performance. The first tool, called iQA, is an interactive system that identifies ambiguities in questions and requests minimal clarifying actions from users. The second tool, called fQA, surgically fuses together the most effective components that are taken from different KB-QA systems and avoids employing the components that are suspected as unreliable to boost the success rate in answering questions. Both iQA and fQA make use of the analyses given by the predictor and the explainer to intelligently determine what interactions and how fusion should be done.

The rest of the paper is organized as follows. In Section 2 we describe KB-QA system basics. Section 3 describes related works. Section 4 describes the PERQ framework and discusses the designs of the predictor, the explainer, and the two rectification tools. Section 5 evaluates the tools. Finally, Section 6 concludes the paper.

2 PRELIMINARIES

In this section we describe a typical question answering pipeline using BuboQA [14] as an example. Figure 2 illustrates the key steps when BuboQA processes the question q_1: "What type of genre is Survival?" This question is taken from the SimpleQ benchmark and it intends to ask about the 2006 movie shown in Figure 1. The answer given in the benchmark is "horror". The first step that BuboQA takes is to detect entities named in the question. From Figure 2, we see that BuboQA correctly recognizes the named entity "Survival", which we will call the *entity mention* (box (a)). BuboQA links the entity mention to a number of candidate entities in the KB, which are shown in box (b). This step is called *entity linking*. In the example, we see that there are multiple KB entities (with different IDs) that share the same or similar names as the entity mention. In particular, the top-ranked KB entity (ID: *m.027fzns*) refers to a music album titled "Survival". The movie "Survival" (ID: *m.027vq10*) is the second one in the list, which is highlighted in red. The next step is *relation prediction*, which identifies relations in the KB that match those in the question. Box (c) shows a ranked list of candidate relations identified by BuboQA. We see that the intended relation *film.film.genre* is ranked third while the unintended one *tv.tv_program_genre* ranks first. The system then evaluates all combinations of the linked entities and the predicted relations to determine the most likely context. The ranked combinations are shown in box (d), from which we see that BuboQA ranks the intended context (movie) second. Finally, box (e) shows the SPARQL query composed, which returns "horror" as an (incidentally) correct answer.

From this example we make two observations. First, although BuboQA gives the correct answer, the process is not perfect. In particular, neither the intended entity nor relation is ranked top in their respective lists. Second, question q_1 is inherently ambiguous. There are other entities with the name "Survival" (including a music album and a TV series) that could have been the intended entity and in which case the answer returned by BuboQA would have been wrong (e.g., the music album "Survival" has genre "rock music").

In this paper we focus on *simple questions* such as those that are listed in the SimpleQ benchmark. We assume that each question can be answered by looking up a single fact (an RDF tuple) in the KB and that the answer is a single entity. For example, question q_1 can be answered by the fact (*Survival*, *film.film.genre*, *horror*). We remark that previous studies on existing benchmarks [19], community QA sites [7], and real-world commercial platform's usage logs [17] have

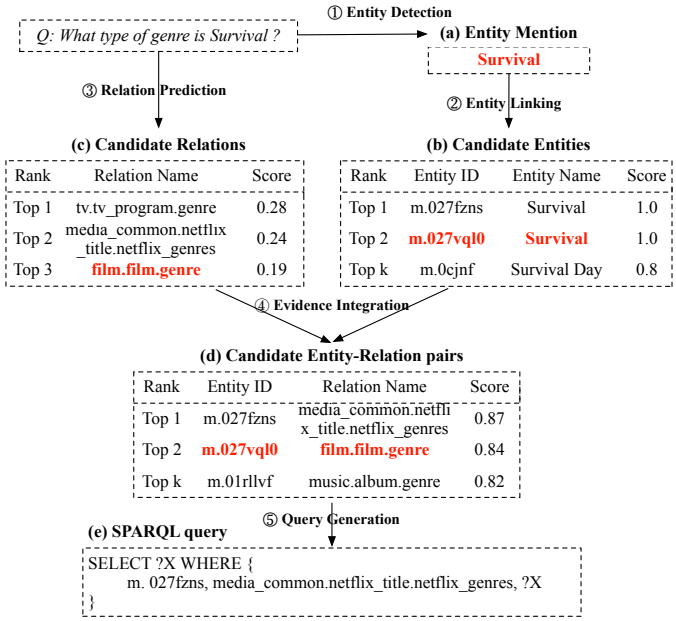

Figure 2: A question processing example.

shown that simple factoid questions sufficiently cover and satisfy a broad range of information needs.

Given a question q and a KB-QA system S, we use $A_S(q)$ to denote the answer to q given by S. With simple questions, we assume that the gold answer of q, denoted by $A_*(q)$, is a single entity. We further assume that $A_S(q)$ is also a single entity. In case S retrieves multiple entities as candidate answers, we assume that the candidates are ranked, and $A_S(q)$ is the top-ranked answer. In the following discussion, we say that "S *correctly answers* q" if $A_S(q) = A_*(q)$; otherwise, we say that "S *fails* q".

3 RELATED WORK

Recent research in KB-QA systems focuses on techniques that improve the various steps of the QA processing pipeline, notably in entity linking and relation prediction. Here, we mention a few exemplary works.

Bordes et al. [4] first propose to design a KB-QA system based on memory network. Golub and He [8] is the first work that introduces a character-level representation of questions, entities, and relations. Their approach is based on the attention-enhanced encoder-decoder architecture recently developed for machine translation [1]. The character-level representation reduces vocabulary size and makes the model more robust in handling the out-of-vocabulary (OOV) problem. They apply an n-gram pruning strategy that speeds up similarity computation between questions and KB entities.

CFO [6] encodes questions at word-level only and utilizes pre-obtained entity/relation encodings (e.g., pre-trained entity/relation embeddings and type vectors). To further reduce search space, CFO uses n-grams that are derived from entity mentions for similarity computation pruning. AMPCNN [22] employs a character-level CNN to encode entity mentions and entity names. For relation matching, however, a word-level CNN with attentive max-pooling is used to better capture word-level semantic concepts. Lukovnikov

et al. [12] present an end-to-end neural network approach for generating <entity, relation> pairs as a single process. Inspired by the success of residual learning in computer vision, [23] uses a hierarchical residual BiLSTM for relation prediction. It applies AMPCNN's entity linking method to derive a questions' answers.

There are also systems that enhance previous models in generating candidate entity-relation pairs. For example, [9] uses a joint (entity and relation) matching method. APVA-TURBO [18] introduces a verification mechanism that re-ranks questions' answers. The "turbo training" mechanism proposed interactively trains the ranking model. AR-SMCNN [16] uses a similarity-matrix-based CNN with two-directions pooling to capture the rich semantics in questions. Finally, KEQA [10] applies knowledge graph embedding (e.g., TransE [5]) in entity linking and relation prediction.

Another line of research focuses on answering more complex questions, for example, those that require multiple facts to answer and/or those whose answers are sets of entities. Example systems include SEMPRE [3] and STAGG [20]. These systems are often evaluated with the WebQuestions (WebQ) benchmark [3]. While our idea can be applied to systems that handle complex questions, we focus on simple questions in this paper. The reason is that our objective is to analyze the factors that affect the accuracy of a KB-QA system. Hence, the availability of a good-quality benchmark (with questions and their correct answers) is necessary for us to construct a reliable model. The WebQ benchmark, however, is far from perfect. First, it is estimated in [21] that only around 2/3 of the questions in WebQ have correct answers in the benchmark. Second, large numbers of WebQ questions are ill-formed because they are extracted from the Google Suggest API. Third, the number of questions in WebQ is 18 times smaller than that of SimpleQ. This makes model training difficult. Finally, we note that around 62% of the questions in WebQ are actually simple questions. We thus leave the study on complex questions as future work.

4 PERQ

In this section we discuss the PERQ framework, which consists of three components. We first discuss the designs of the predictor and the explainer in Sections 4.1 and Section 4.2, respectively. After that, we describe the two rectification tools, iQA and fQA, in Section 4.3.

4.1 Predictor

The predictor's task is to predict if a KB-QA system S would correctly answer a question q. We model the prediction problem as a classification problem. Specifically, we consider a set of features $\mathcal{F} = [f_1, \ldots, f_d]$, whose values are derived from q and the execution of S on q. These values are encoded in a *feature vector* $\boldsymbol{x}_{S,q}$.

Definition 1. *A predictor, denoted by C, is a function that takes a feature vector $\boldsymbol{x}_{S,q}$ and outputs a confidence score $p \in [0, 1]$ that expresses the likelihood of S correctly answering q, i.e., $C(\boldsymbol{x}_{S,q}) \rightarrow \langle p \rangle$. If $p < 0.5$, we interpret the prediction being "S fails q"; otherwise, the prediction is "S correctly answers q".*

We consider the following two categories of features:

Question Features. We perform textual analysis on an input question q to obtain features that are related to its structure, function, and content. Examples include: Question length — a longer question (with more words) potentially mentions more entities and relations and is thus more complex. On the other hand, a longer question may be more specific and hence less ambiguous; Question interrogatives — such as the wh-word (what, which, who, how) of q; Structural complexity of q, which can be captured by the depth of q's constituency parsing tree [11]. For example, the question "Which gaming company worked on the Legend of Zelda : Twilight Princess" is much more complex than the question "Is Sami Garam male or female". While the parse tree of the former has 7 levels, the latter has only 3. In total, we have 22 question features.

System Features. These are features related to the processing of q by S, which can be collected from the output of the various components of S. One example is the number of candidate answers S retrieves from the KB. Intuitively, we are less confident in S's answer if there are many candidate answers and so it is more likely that S fails q. As another example, in the entity-linking step, S identifies a list of candidate KB entities (Figure 2 box (b)). Two features we can derive are (1) whether the intended entity is likely included in the list and (2) the confidence score assigned by S to the identified entity. We consider a total of 23 system features.

Table 2 shows the list of features. For example, the system feature *Entity-Confidence* refers to the confidence score $p(\hat{e}_1)$ that the system assigns to the top-ranked candidate entity; The feature *Linking@k* refers to the likelihood that the intended entity is one of the top-k entities identified by the system. This feature is measured by the sum of the confidence scores of the top k entities ($\sum_i^k p(e_i)$).

Let $\mathcal{F}_q$ and $\mathcal{F}_{sys}$ be the sets of question features and system features, respectively. We build a classifier C with the feature set $\mathcal{F} = \mathcal{F}_q + \mathcal{F}_{sys}$ to serve as the predictor.

4.2 Explainer

Given that a question q is predicted to be failed by a KB-QA system S, the explainer's task is to provide the reasons behind such a prediction. Specifically, we design an explainer that identifies "culprit features," which are those in $\mathcal{F}$ that exert the biggest influence in causing the predictor to arrive at the "S fails q" outcome.

Definition 2 (Culprit Feature). *Given a KB-QA system S, a question q that is represented by the feature vector $\boldsymbol{x}_{S,q} = [x_1, \ldots, x_d]$, where x_i is the value of feature $f_i \in \mathcal{F}$, and a predictor C, we derive a feature attribution value ϕ_i for each feature f_i. We define ϕ_i as the change in the predictor's prediction ($C(\boldsymbol{x}_{S,q})$) conditioned on f_i. Features that are associated with large negative ϕ's are the culprit features of question q. The one with the largest negative attribution is called the top-culprit.*

We apply SHAP (**SH**apley **A**dditive ex**P**lanations) [13], which computes feature attributions based on Shapley regression values in game theory. To compute the effect of a feature $f_i \in \mathcal{F}$, for each subset of features $X \subseteq \mathcal{F} \setminus \{f_i\}$, we train two models C_X and $C_{X \cup \{f_i\}}$. The former is trained with features in X and the latter includes feature f_i as well. ϕ_i is computed by

$$\phi_i = \sum_{X \subseteq \mathcal{F} \setminus \{f_i\}} \frac{|X|!(|\mathcal{F}| - |X| - 1)!}{|\mathcal{F}|!} [C_{X \cup \{f_i\}}(\boldsymbol{x}_{S,q}) - C_X(\boldsymbol{x}_{S,q})].$$

Intuitively, the feature attribution ϕ_i is the weighted sum of the differences in prediction results for the question q given by classifiers constructed with and without using feature f_i.

As an illustrative example, we predict the correctness of BuboQA's answer to the question q_2 = "*What type of genre is Survival?*" Figure 3 illustrates the explanation of the prediction. Features with

Notations		
mention: entity mention recognized	$\mathbb{E}$: the set of all candidate entities in KB	e_i: the i-th ranked entity in $\mathbb{E}$
$\mathbb{R}$: the set of all candidate relations in KB	r_i: the i-th ranked relation in $\mathbb{R}$	$\mathbb{ER}$: the set of all candidate entity-relation pairs
er_i: the i-th ranked entity-relation pair in $\mathbb{ER}$, let $er_1 = \langle \hat{e}, \hat{r} \rangle$	$Sim(*, *)$: Levenshtein similarity of two strings	$name(*)$: name of $*$ in KB
$\mid * \mid$: number of elements in $*$	$p(*)$: confidence of $*$	
$\mathcal{F}_q$: **Question features (22)**		
▹ Q-Length: word count of the question	▹ Stopwords: number of stopwords	▹ Structure Complexity: height of constituency tree
▹ Interrogatives: one-hot encoding of 7 interrogatives, including WHO, WHAT, WHERE, WHEN, WHICH, HOW, NAME		
▹ Parts of Speech: Number of 12 parts of speech tags (defined in [15]) in the questions, including NOUN (nouns), VERB (verbs), ADJ (adjectives), ADV (adverbs), PRON (pronouns), DET (determiners and articles), ADP (prepositions and postpositions), NUM (numerals), CONJ (conjuctions), PRT (particles), ⁘ (punctuation marks), X (other categories, such as foreign words)		
$\mathcal{F}_{sys}$: **System features (23)**		
⇒ **Entity Detection Features (2)**		
▹ Mention Similarity: $Sim(mention, name(e_1))$	– String similarity between the entity mention and the name of the top-ranked entity in KB	
▹ Mention Length: word count of *mention*	– Length of the entity mention	
⇒ **Entity Linking Features (9)**		
▹ Entity Found: $\sum_{e \in \mathbb{E}} p(e)$	– Likelihood of the intended entity being in the entity list	
▹ Entity Confidence: $p(\hat{e})$	– Likelihood of the intended entity ranked first in the relation list	
▹ Linking@K: $\sum_{i=1}^{k} p(e_i)$, $k \in [1, 3, 5, 10, 20, 50]$	– Likelihood of the intended entity being top-K in the entity list	
▹ #Linking: $\mid \mathbb{E} \mid$	– Length of the entity list	
⇒ **Relation Features (6)**		
▹ Relation Found: $\sum_{r \in \mathbb{R}} p(r)$	– Likelihood of the intended relation being in the relation list	
▹ Relation Confidence: $p(\hat{r})$	– Likelihood of the intended relation ranked first in the relation list	
▹ Relation@K: $\sum_{i=1}^{k} p(r_i)$, $k \in [1, 3, 5]$	– Likelihood of the intended relation being top-K in the relation list	
▹ #Relations: $\mid \mathbb{R} \mid$	– Length of the relation list	
⇒ **Ranking Features (3)**		
▹ E-R Found: $\sum_{er \in \mathbb{ER}} p(er)$	– Likelihood of the intended entity-relation pair being in the candidate entity-relation list	
▹ E-R Confidence: $p(er_1)$	– Likelihood of the intended entity-relation pair ranked first in the candidate entity-relation list	
▹ #E-R: $\mid \mathbb{ER} \mid$	– Length of the entity-relation list	
⇒ **KB features (3)**		
▹ #Homonyms: $Count(t)$, where t is an entity in KB and $name(\hat{e}) == name(t)$	– Number of homonyms of the intended entity	
▹ In/Out Degree: in/out degree of e_1	– In/Out degree of the top-ranked entity in the KB	

Table 2: Features

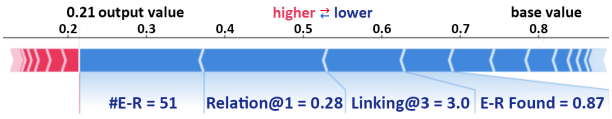

Figure 3: Explainer's output for question q_2

positive (negative) attributions are shown in red (blue). The length of the bar associated with a feature indicates the absolute value of the feature's attribution. From Figure 3, we see that the prediction score is 0.21, which indicates a low confidence in BuboQA correctly answering q_2. Our predictor thus predicts that BuboQA fails this question. The explainer identifies four culprit features. For example, the culprit "*#E-R=51*" means that BuboQA found 51 candidate entity-relation pairs (see Figure 2, box (d)), which will lead to 51 possible answers. The culprit "*Relation@1=0.28*" means that BuboQA has low confidence (0.28) that the intended relation is ranked first in the candidate relation list (see Figure 2, box (c)). There are other features with negative attributions that are not shown because they have smaller (absolute) attribution values. All these suggest that BuboQA is likely to fail q_2.

4.3 Rectifiers

In this section we discuss two approaches that take advantage of the analyses given by the predictor and the explainer to rectify the answer of a question. The first approach, called iQA (for *interactive QA*), aims at disambiguating a question by requesting simple feedback from the enquirer. The second approach, called fQA (for *fusion QA*), intelligently fuses the components of a set of KB-QA systems to improve answers' accuracy.

4.3.1 iQA. We applied the predictor and the explainer to analyze how well existing KB-QA systems answer questions. We observe that a major reason of failing a question is the intrinsic ambiguity in the question. For example, there are 16 KB entities matching the entity mention "Survival" for question q_2. The idea of iQA is to resolve ambiguity by asking an enquirer to choose from a short list of relevant contexts.

There are existing KB-QA systems that interact with enquirers during the question-answering process. Two prominent systems are Improve-QA [24] and Zheng's [25]. These systems, however, require certain level of user sophistication. For example, Improve-QA requires enquirers to distinguish among correct, incorrect, and missing answers of questions, while Zheng's requires enquirers to interpret and refine query graphs. In contrast, iQA limits user interaction to *few-and-simple actions* that can be easily taken by *casual unsophisticated* enquirers, who have limited knowledge of their questions' answers. We follow three basic principles.

[P1: Avoid unnecessary interactions] For simple factoid questions, existing KB-QA systems can achieve an accuracy of up to around 80% (see Table 1). In case the system can answer a question correctly, enquirers should not be bothered with unnecessary interactions. Existing interactive systems, however, are rather indiscreet in engaging users in interactions.

[P2: Minimality] For questions that require additional user input, the information requested should be minimal and should be restricted to only those that can most effectively refine the questions.

[P3: Simplicity] Interactions should easily carried out by unsophisticated enquirers that do not have prior knowledge of the technical aspects of the systems or the KBs. For example, we do not assume enquirers have the skill to help compose a query graph (cf. Zheng's). The amount of information that an enquirer has to consume during the interaction should also be minimal.

Figure 4 shows iQA's architecture. Given a KB-QA system S, iQA utilizes S to answer a user query q. In our current implementation, we use BuboQA [14] as system S, which accesses information stored

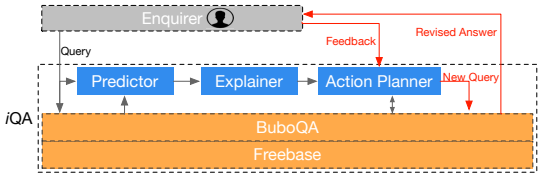

Figure 4: iQA's architecture

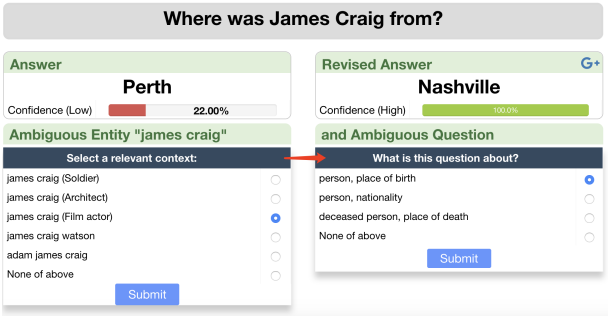

Figure 5: iQA Example

in Freebase. iQA employs a predictor and an explainer, which were discussed in the previous sections. Note that if the predictor predicts that S correctly answers q, then no user interaction is needed; the answer obtained by S can be directly returned to the enquirer. This satisfies Principle P1 in avoiding unnecessary interactions. In case it is predicted that S fails q, the explainer determines what needs to be fixed (*culprit feature*). This satisfies Principle P2 in that only relevant interactions with the enquirer will be carried out.

We analyzed BuboQA's answers to the questions in the SimpleQ benchmark. We found that among all failed questions, 99.6% have one of the following as the top-culprit feature: *#Homonyms*: the number of entities in the KB that share the same name as the entity mentioned in q is large. *Linking@1* and *Relation@1*: the confidence of S that it has identified the correct entity/relation in the KB is low. *ER-Found* and *ER-Confidence*: S's confidence in finding the correct entity-relation pair is low.

The heart of iQA is the *Action Planner*. Based on the culprit features identified by the explainer, the action planner determines a set of actions. To satisfy Principle P3, these actions are limited to three simple types:
A1: display a short list of entities and request the user to pick one,
A2: display a short list of relations and request the user to pick one,
A3: display a short list of *categories* and request the enquirer to pick a category to help filter entities.

For each question that is predicted *failed*, at most 2 actions are requested. iQA provides an action plan for each feature. These action plans will be illustrated shortly. We remark that multiple KB entities could share the same surface name (homonym). For example, 16 movies/songs/albums etc. in Freebase are all named *Survival*. Hence, simply displaying the KB entities' surface names (which could all be the same) to enquirers and asking them to choose is a confusing action. iQA addresses this issue by selecting a suitable *context* for each displayed entity. A *context* could be a category or a distinguishing relation of an entity. iQA employs an information theoretic approach to compose such contexts. Due to space limitation, we omit further discussion on context composition in this paper. We illustrate iQA's action plans with an example.

Figure 5 shows question q_3: "*Where was James Craig from?*" This question is difficult for BuboQA because it is ambiguous in both entity and relation — there are multiple KB entities with the name *James Craig* and the relation that expresses the idea of "where someone is from" is uncertain (e.g., it could refer to nationality or birthplace). The predictor gives a very low confidence (22%) in BuboQA's answer ("*Perth*"). The explainer identifies that both entity and relation need to be disambiguated. The action planner thus picks the plan of requesting Action A1: select an entity, followed by Action A2: select a relation. After the enquirer responds with the intended entity (James Craig the actor) and the intended relation (birthplace of a person), iQA re-executes BuboQA. The revised answer is "*Nashville*".

4.3.2 fQA. Consider a question q answered by a set $\mathcal{S} = \{S_1, \ldots, S_k\}$ of KB-QA systems such that $A_{S_i}(q)$ is the answer given by S_i. Let p_i be the confidence score given by the predictor on the answer $A_{S_i}(q)$. In our study, we consider three KB-QA systems, namely, AR-SMCNN, BuboQA, and CFO as the S_i's. We found that about 17.1% of all SimpleQ questions are incorrectly answered by ALL three systems. These questions are the toughest ones and for them our predictor gives low confidence scores (p_i's). Given a question q, if $p_i < 0.5$ for all S_i's, we call q a "*tough question*". Our next rectifier, fQA, attempts to better handle tough questions.

Given a tough question q, fQA fuses the S_i's via a *component-wise* integration. Referring to Figure 2, we see that most KB-QA systems generate a candidate entity list and a candidate relation list. fQA consults the explainer to see if the two lists constructed by an S_i can be trusted. Specifically, the explainer identifies the top-culprit feature in S_i's processing of q. Let us denote this culprit by $\hat{f}_i$. fQA determines if $\hat{f}_i$ is relevant to the entity list, or to the relation list, or both. Such list(s) is/are then labeled "untrusted" and are discarded. fQA then collects from the KB-QA systems all lists that are not untrusted and integrate them to derive a ranked list of entity-relation pairs (see Box (d) in Figure 2). To integrate, for example, the entity lists from different systems, the entity scores (such as those shown in Box (b) in Figure 2) are scaled by the Sigmoid function before the lists are merged. The integration of relation lists is similarly done. After that, we apply BuboQA's algorithm to compute the ranked list of entity-relation pairs from which the final answer to q is determined.

Note that the 23 $\mathcal{F}_{sys}$ features are divided into 5 groups (see Table 2). Features in categories Entity-Detection, Entity-Linking, and KB-Features are relevant to the entity list; those in category Relation are relevant to the relation list. Finally, features listed under the Ranking category are relevant to both lists.

5 EXPERIMENTS

We conducted experiments to evaluate the effectiveness of PERQ's four components, namely, the predictor, the explainer, iQA, and fQA. In this section we report the results and further illustrate the components with examples.

	Accuracy	"Correctly-answered" label			"Failed" label		
		Precision	Recall	F1	Precision	Recall	F1
C_{AR}	89.8%	90.9%	96.6%	93.7%	84.1%	65.1%	73.4%
C_{CFO}	89.6%	90.9%	96.4%	93.6%	83.3%	65.2%	73.2%
C_{BuboQA}	87.8%	88.9%	95.8%	92.2%	83.1%	63.6%	72.0%
$\mathcal{F}_{sys}$	87.6%	88.8%	95.6%	92.1%	82.4%	63.2%	71.5%
$\mathcal{F}_q$	75.5%	75.9%	99.1%	85.9%	55.2%	3.6%	6.7%

Table 3: Predictors' performance for three KB-QA systems

5.1 Experimental Setup

We use SimpleQ as our question benchmark. SimpleQ consists of 108,442 questions. Each question is associated with a gold standard fact: ⟨subject; relation; object⟩ that can be found in **FB2M**, where **FB2M** is a Freebase subset with 2 million entities. The gold answer to a question can be directly deduced from the associated fact. The questions in SimpleQ are conventionally divided into training/validation/test sets with 79,590/10,845/21,687 questions, respectively. We denote them by Q_1, Q_2, Q_3, respectively.

We evaluate PERQ by installing it in three existing KB-QA systems, namely, AR-SMCNN, BuboQA, and CFO. In particular, these three systems form the ensemble of fQA. We individually trained, validated, and tested them using Q_1, Q_2, Q_3, respectively. The accuracies of our installations of the three systems using the SimpleQ benchmark are respectively AR-SMCNN: 72.8%; BuboQA: 75.2%; CFO: 69.7%. Note that the numbers slightly differ from those reported in the original papers of the systems. Similar issues in reproducing the original results are reported in [10].

5.2 Predictor

We train a predictor C_S for each system $S \in$ {AR-SMCNN, BuboQA, CFO}. To train C_S, we label questions in Q_2 as training data. Specifically, for each question $q \in Q_2$, we label q according to whether q is correctly-answered/failed by system S. Predictor C_S is then evaluated by questions in Q_3, which are similarly labeled. We have applied different classification methods to build the predictors. These include Naïve Bayes, CART, SVM, LightGBM, and XGBoost. We find that XGBoost gives the best overall performance. In the following, all predictors are constructed using XGBoost.

Table 3 shows the performance of the three predictors C_{AR}, C_{BuboQA}, C_{CFO}, for the three KB-QA systems AR-SMCNN, BuboQA, and CFO, respectively. For each predictor C_S (row), we show its overall accuracy, followed by the precision, recall, and F1 scores of the two classes of questions: those that are answered correctly by S (labeled "correctly-answered") and those that are failed by S (labeled "failed"). From the table, we see that the predictors are highly accurate — the overall accuracies of the predictions range from 87.8% to 89.8% over different KB-QA systems that use different techniques and models for question-answering. This shows that the prediction task is viable and is applicable to a range of different systems. Comparing each predictor's performance with respect to the two classes of questions, we observe that the predictors are especially strong in identifying "correctly-answered" questions (with very high precision and recall values). For "failed" questions, the predictors are able to capture around 2/3 of them (recall) with very good precision (in the 80's).

The last two rows in Table 3 show the results of an ablation analysis that compares question features ($\mathcal{F}_q$) vs. system features ($\mathcal{F}_{sys}$) in terms of their effectiveness. The results are obtained with predictor C_{BuboQA}. The "$\mathcal{F}_{sys}$" and the "$\mathcal{F}_q$" rows show the performance when only system features or only question features are used, respectively. From the table, we see that system features alone are mostly sufficient to construct a good predictor. Adding question features only marginally improve the performance numbers. On the other hand, question features alone are ineffective. In particular, the recall of "failed" questions is extremely low (3.6%). This shows that failed questions do not have very distinguishing features in the question construct.

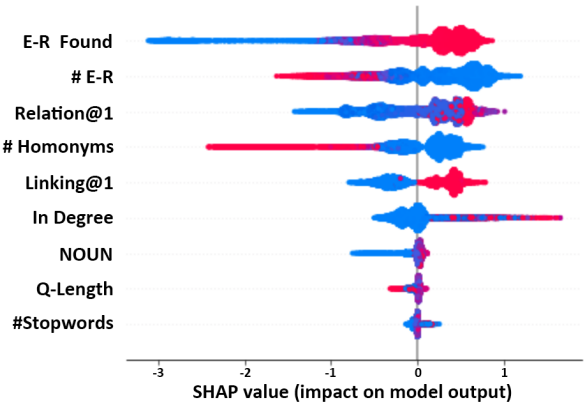

Figure 6: Features' attribution distributions

By inspecting and comparing a large number of correctly-answered and failed questions that share similar question structures, we observe that the main determining factor of whether a question is likely to be failed is *ambiguity*. For SimpleQ's questions, the main source of ambiguity come from the KB's content instead of the question structure. To illustrate, consider the following questions taken from SimpleQ, q_4: "Which country is Spring Hill located in?" and q_5: "Which country is Lake Wakuach located in?". Even though the questions have the same structure, BuboQA correctly answers q_5 but fails q_4. The reason is that there are dozens of places named "Spring Hill" but "Lake Wakuach" is unique.

Next, we compare the effectiveness of each individual feature. In Figure 6, we plot the attribution value distributions of 9 selected features. (Due to space limitations, the plots of the other 36 features are not shown.) Each question $q \in Q_3$ contributes one pixel in each distribution plot. Given a feature f_i, let x_i be the value of f_i for question q. The pixel's color indicates the value x_i. Specifically, the spectrum of blue-to-purple-to-red corresponds to the range of small-to-large values. The location of the pixel indicates the attribution ϕ_i of f_i for question q. The scale of the attribution range is shown in the x-axis of Figure 6. We have negative ϕ_i on the left and positive ϕ_i on the right in the plots. Intuitively, the wider a feature's plot extends, the more influential is the feature in predicting if a question is correctly answered or failed.

As an example, the top feature is *E-R Found*. This feature measures BuboQA's confidence in whether the correct E-R pair is included in its E-R list. From the plot, we see that all blue pixels are located on the (far) left, which means that whenever this feature value is small (blue), the predictor is swayed much towards predicting a "failed" label (large negative attribution). Hence, whenever BuboQA has low confidence in its E-R list, chances are that it

Category of the identified top culprit	Precision	Recall
Entity-related	97.5%	98.3%
Relation-related	86.1%	86.1%

Table 4: Explainer's accuracy

would fail the question. As another example, consider the feature *#Homonyms*. From its plot, we see that all red pixels are located on the far left. This indicates that whenever the number of homonyms is large (red), the question is likely failed. As an example of a non-influential feature, consider *Q-Length*, which measures the number of words in a question. The feature's attribution plot has a very small spread. Moreover, the negative attribution region is a mixture of red and blue pixels. *Q-Length* thus gives a very weak prediction signal. Similar observations can be made for other question features. In contrast, the distributions for features at the top of Figure 6 show strong prediction signals. For example, from our data, the average values of *#E-R* for correctly-answered/failed questions are 11 vs 32; those of *#Homonyms* are 12 vs 163.

5.3 Explainer

In this section we evaluate the effectiveness of the explainer. As we have discussed, system features are more influential in the prediction. Hence, we focus on system features in the following discussion. The 23 system features belong to 5 groups (see Table 2), which can be further put under 3 categories. The first category includes features in groups *Entity-Detection*, *Entity-Linking*, and *KB*. They are related to entity linking. The second category includes features in group *Relation*, which are related to relation prediction. Features in group *Ranking* are related to the E-R pairs found by the KB-QA systems. They form the last category. Let us first focus on the first two categories, i.e., entity-related and relation-related.

We collect questions in the set Q_3 that are failed by BuboQA and are predicted failed by our predictor into a question set Q_4. There are 3,390 questions in the set. We investigate whether our explainer can identify the correct categories of the failure. Table 4 shows an analysis of the explainer's output if the identified top culprit belongs to either the entity-related category or the relation-related category. For each category, Table 4 shows the accuracy of the explainer in terms of precision and recall. For example, the precision of the entity-related category is 97.5%. This means that if the top culprit feature the explainer identified for a question belongs to the entity-related category, 97.5% of the time, BuboQA retrieves the wrong entity in answering the question. Also, the recall shown in the table is 98.3%. This means that when BuboQA retrieves the wrong entity in answering a question, the explainer can identify a top culprit that is entity-related. From the table, we see that the explainer is highly effective.

Next, we show a few cases that further illustrate the explainer. Table 5 shows four questions taken from SimpleQ. For each question, we show the answer given by BuboQA, the gold answer, the prediction score given by the predictor and its prediction ("correctly-answered" or "failed"), and the top-culprit identified by the explainer. For q_6, the predictor has very high confidence (0.97) that BuboQA's answer is correct, so it predicts the label "correctly-answered". For q_7, the predictor has very low confidence (0.01) in BuboQA's answer, so it predicts the label "failed". The explainer reveals that there are 565 entities in the KB that share the same name of the identified entity ("And I Love Her") and suggests that *#Homonyms* = 565 is the culprit of BuboQA failing q_7. Note that "And I Love Her" is a song originally recorded by the Beatles in 1964. Since then, there are many cover versions of the song, which contribute to some of the homonyms found in the KB. Finally, for q_8, the predictor again has very low confidence (0.08) in BuboQA's answer. The explainer suggests that the entity recognized by BuboQA is likely incorrect (*EntityFound* is small (0.6) compared with typical values for correct answers (which range from 0.8 to 1.0)). On further inspection, we found that BuboQA extracted (correctly) the entity mention ("*Neusoft group*") from q_8. However, the real name of the company is "*Neusoft*", which is what is recorded in the KB. BuboQA links the entity mention to other entities in the KB, such as "*mnet group*" due to its shorter string-matching distance from "*Neusoft group*". *MNet group* is a media company and hence the incorrect answer (*media*) is returned by BuboQA.

5.4 iQA

iQA improves question-answering by engaging enquirers to resolve ambiguity. In this section we evaluate our iQA implementation using BuboQA as the underlying KB-QA engine. We focus on two aspects, namely, efficiency (the amount of work asked of the users) and effectiveness (answer accuracy improvement).

We consider the 21,633 questions in the set Q_3 that BuboQA processes. In the experiment, BuboQA correctly answers 16,301 of them. 15,610 of these 16,301 questions are correctly predicted by our predictor. Hence, for a question that is correctly answered by BuboQA, 15,610/16,301 = 95.8% of the time iQA requests no actions from the enquirer. Hence, iQA is highly effective in avoiding unnecessary interactions.

Table 6 shows the distribution of the number of actions requested per user question. We see that 17,552 questions require no actions (these are questions whose labels are predicted as "correctly-answered" by the predictor), which amounts to 81.1% of all questions. For those that the predictor requests actions, we have a fairly even split between 1 vs. 2 actions. Note that iQA limits interactions to at most 2 actions for each question. The average number of actions per question is 0.28. As a comparison, with Zheng's [25] (see Section 4.3.1), an enquirer interacts with the system 3.6 times on average. The number of actions requested by iQA are very small.

With iQA, each action involves the enquirer picking an entity/relation/category from a short list. We limit each list to display at most N_d items. Note that an enquirer may read 0, at most N_d, or at most $2 \times N_d$ items depending on whether 0, 1, or 2 actions are asked of him. Let N_r be the average number of items read by an enquirer. Table 7 shows the system's accuracy if iQA is used to rectify BuboQA's answers under different values of N_d. BuboQA's accuracy without iQA rectification is also shown as a reference. From the table, we see that the more items we display per list, the higher is the system's accuracy. This is because a longer list helps improve the chances of the correct entity/relation being displayed and selected by the enquirer. With at most 20 items displayed, we can significantly improve answer accuracy by 7 percentage points. Even with very short lists (N_d = 5 in our baseline implementation), we register a significant improvement of 4.2 percentage point. Also shown in Table 7 is N_r, the average number of items that an enquirer has to read. We see that the work requested is very minimal. Even for N_d = 20, each enquirer is expected to read 3.6 items only.

	Question	BuboQA answer	Gold answer	Prediction score (predicted label)	Explanation (culprit features)
q_6	Which color represents Golden Gate University?	Blue	Blue	0.97 ("correctly-answered")	-
q_7	Who recorded And I Love Her?	Bobby Womack	José Feliciano	0.01 ("failed")	#Homonyms = 565
q_8	What type of organization is Neusoft group?	Media	Limited Company	0.08 ("failed")	EntityFound = 0.6

Table 5: Prediction and explanation example cases

Number of actions requested	0	1	2
Number of questions	17,552	2,020	2,061
Percentage	81.1%	9.3%	9.5%

Table 6: Distribution of number of actions per question

	BuboQA	iQA with N_d =			
		5	10	15	20
Accuracy	75.2%	79.4% (+4.2%)	80.9% (+5.7%)	81.6% (+6.4%)	82.2% (+7.0%)
N_r		1.1	2	2.8	3.6

Table 7: Accuracy improvement given by iQA vs N_d

	AR-SMCNN	BuboQA	CFO	Random	fQA
Accuracy	72.8%	75.2%	69.7%	72.6%	**79.4%(+6.8%)**

Table 8: Accuracies of single systems and fQA. Bold typeface indicates a statistically significant result.

5.5 fQA

We evaluate the rectification tool fQA. We use $\mathcal{S}$ = {AR-SMCNN, BuboQA, and CFO} as the KB-QA systems whose results are fused by fQA. We use Q_3 as the question set for evaluation. Table 8 shows the accuracies of the systems. The system labeled "Random" refers to the one that randomly picks a system from $\mathcal{S}$ to answer a question. Its accuracy is the average of the three systems in $\mathcal{S}$ and it serves as a reference for comparison; From the table, we see that fQA significantly improves accuracy; It achieves a +6.7 percentage point improvement over Random. And the improvement is statistically significant compared with all four baselines using paired t-test with p-value < 0.05.

To understand how fQA helps improve accuracy, we conduct the following measurement. For each system $S \in \mathcal{S}$, we partition the set of test questions Q_3 into two sets, $\mathcal{A}(S)$ and $\mathcal{D}(S)$. Intuitively, $\mathcal{A}(S)$ is the set of questions for which fQA *agrees* with the answers given by system S, i.e., $\mathcal{A}(S) = \{q \in Q_3 \mid \text{fQA and } S \text{ give the same answer to } q\}$. $\mathcal{D}(S) = Q_3 - \mathcal{A}(S)$ is the *disagree set*. Figure 7 shows the accuracies of each system on its agree/disagree sets. In the figure, the respective systems' accuracies over the whole test set Q_3 are illustrated by the dotted lines. For example, CFO's accuracy is 69.7%. However, given a question q for which fQA *agrees* with CFO's answer (i.e., $q \in \mathcal{A}(\text{CFO})$), there is a 82.6% chance that CFO correctly answers q. In contrast, if fQA does not agree with CFO on its answer to a question q (i.e., $q \in \mathcal{D}(\text{CFO})$), CFO is very likely to fail q (with an accuracy of only 10.4%). The big difference in the accuracies for the agree set and the disagree set shows that fQA's consideration of the predictor output is very effective in determining whether an answer given by a system should be trusted or not. This leads to the good performance of fQA.

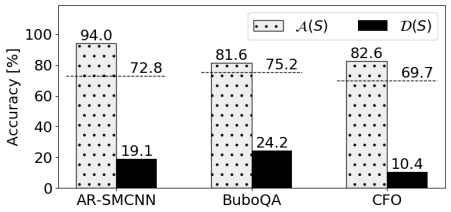

Figure 7: Systems' accuracies over their respective $\mathcal{A}(S)$ and $\mathcal{D}(S)$

	AR-SMCNN	BuboQA	CFO	fQA
n	64	300	141	460
$n/\lvert Q_t \rvert$	3.0%	13.9%	6.6%	**21.4%**

Table 9: Systems' performance in handling tough questions. (n = # of questions in Q_t that are correctly answered.)

As discussed in Section 4.3.2, the objective of the fQA is to handle "*tough*" questions. We define *tough questions* to be those that the predictor gives < 0.5 confidence scores for ALL systems in the set $\mathcal{S}$. We found that out of the 21,687 questions in Q_3, 2,705 of them are *tough*. Furthermore, we inspected all the entity/relation lists retrieved by the systems for the tough questions. We found that for 553 of the tough questions, their correct entities/relations are missing in all the lists. Since no integration methods can answer those 553 questions, we omit them in our evaluation. That leaves us 2,152 tough questions. They are collected in the set Q_t.

Table 9 shows the number of tough questions that are correctly answered by each system (n). From the table, we see that the single systems are weak in answering tough questions, especially for AR-SMCNN and CFO; BuboQA is the best among the three. fQA shows a statistically significant boost in answering 460 tough questions correctly, which is 7 times more than what AR-SMCNN can handle.

6 CONCLUSIONS

In this paper we investigate the intrinsic reasons of certain questions being difficult to answer by existing KB-QA systems and study techniques that improve question-answering accuracy. We put forward the PERQ framework, which consists of three components: a predictor, an explainer, and a rectifier. Through experiments, we show that our predictor and explainer are highly effective in assessing whether a question is likely failed by a KB-QA system, and if so, the main cause of the failure. We proposed two tools: iQA and fQA to rectify failed questions. Our experiments show that both tools can significantly improve system accuracy.

ACKNOWLEDGMENTS

This research is supported by Hong Kong Research Grant Council GRF grants 17254016.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *ICLR* (2015).

[2] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *COLING*. ACM.

[3] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *EMNLP*. ACL.

[4] Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *arXiv:1506.02075* (2015).

[5] Antoine Bordes, et al. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*. 2787–2795.

[6] Zihang Dai, Lei Li, and Wei Xu. 2016. CFO: Conditional focused neural question answering with large-scale knowledge bases. *ACL* (2016).

[7] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. 2013. Paraphrase-driven learning for open question answering. In *ACL*, Vol. 1. ACL.

[8] David Golub and Xiaodong He. 2016. Character-Level Question Answering with Attention. In *EMNLP*. ACL.

[9] Yanchao Hao, Hao Liu, Shizhu He, Kang Liu, and Jun Zhao. 2018. Pattern-revising Enhanced Simple Question Answering over Knowledge Bases. In *COLING*.

[10] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *WSDM*. ACM, 105–113.

[11] Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *ACL*, Vol. 1. Association for Computational Linguistics, 423–430.

[12] Denis Lukovnikov, et al. 2017. Neural network-based question answering over knowledge graphs on word and character level. In *WWW*. ACM.

[13] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *NIPS*. Curran Associates, Inc.

[14] Salman Mohammed, et al. 2018. Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks. *NAACL-HLT* (2018).

[15] Slav Petrov, Dipanjan Das, and Ryan McDonald. 2011. A universal part-of-speech tagset. *arXiv:1104.2086* (2011).

[16] Yingqi Qu, et al. 2018. Question answering over freebase via attentive RNN with similarity matrix based CNN. *COLING* (2018).

[17] Ferhan Ture and Oliver Jojic. 2017. No Need to Pay Attention: Simple Recurrent Neural Networks Work!. In *EMNLP*. ACL.

[18] Yue Wang, Richong Zhang, Cheng Xu, and Yongyi Mao. 2018. The APVA-TURBO Approach To Question Answering in Knowledge Base. In *COLING*. 1998–2009.

[19] Xuchen Yao. 2015. Lean question answering over Freebase from scratch. In *NAACL-HLT*. ACL.

[20] Scott Wen-tau Yih, et al. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. *ACL* (2015).

[21] Wen-tau Yih, et al. 2016. The value of semantic parse labeling for knowledge base question answering. In *ACL*, Vol. 2. ACL.

[22] Wenpeng Yin, et al. 2016. Simple Question Answering by Attentive Convolutional Neural Network. In *COLING*. ACM.

[23] Mo Yu, et al. 2017. Improved neural relation detection for knowledge base question answering. *ACL* (2017).

[24] Xinbo Zhang and Lei Zou. 2018. IMPROVE-QA: An Interactive Mechanism for RDF Question/Answering Systems. In *SIGMOD*. ACM.

[25] Weiguo Zheng, et al. 2019. Interactive Natural Language Question Answering Over Knowledge Graphs. *Information Sciences* (2019).