More Accurate Question Answering on Freebase

Hannah Bast, Elmar Haussmann
Department of Computer Science
University of Freiburg
79110 Freiburg, Germany
{bast, haussmann}@informatik.uni-freiburg.de

ABSTRACT

Real-world factoid or list questions often have a simple structure, yet are hard to match to facts in a given knowledge base due to high representational and linguistic variability. For example, to answer "who is the ceo of apple" on Freebase requires a match to an abstract "leadership" entity with three relations "role", "organization" and "person", and two other entities "apple inc" and "managing director". Recent years have seen a surge of research activity on learning-based solutions for this method. We further advance the state of the art by adopting learning-to-rank methodology and by fully addressing the inherent entity recognition problem, which was neglected in recent works.

We evaluate our system, called *Aqqu*, on two standard benchmarks, Free917 and WebQuestions, improving the previous best result for each benchmark considerably. These two benchmarks exhibit quite different challenges, and many of the existing approaches were evaluated (and work well) only for one of them. We also consider efficiency aspects and take care that all questions can be answered interactively (that is, within a second). Materials for full reproducibility are available on our website: `http://ad.informatik.uni-freiburg.de/publications` .

1. INTRODUCTION

Knowledge bases like Freebase have reached an impressive coverage of general knowledge. The data is stored in a clean and structured manner, and can be queried unambiguously via structured languages like SPARQL. However, given the enormous amount of information (2.9 billion triples for Freebase), mapping a search desire to the right query can be an extremely hard task even for an expert user. For example, consider the (seemingly) simple question *who is the ceo of apple*. The answer is indeed contained in Freebase, and the corresponding SPARQL query[1] is:

```
select ?name where {
  Managing_Director  job_title.people_with_this_title  ?0 .
  ?0  employment_tenure.company  Apple_Inc .
  ?0  employment_tenure.person  ?name
}
```

It would clearly be preferable, if we could just ask the question in natural language, and the machine automatically computes the corresponding SPARQL query. This is the problem we consider in this paper.

We focus on "structurally simple" questions, like the one above. They involve k entities (typically two or three, in the example above: *ceo* and *apple* and the result entity), which are linked via a single k-ary relation in the knowledge base. For languages like SPARQL, k-ary relations for $k > 2$ can be represented by a special entity (one for each k-tuple in the relation) and $k - 1$ binary relations (in the example above: the three binary relations in the where clause, all connected to the ?0 entity).

The challenge for these questions is to find the matching entities and relations in the given knowledge base. The entity-matching problem is hard, because the question may use a variant of the name used in the knowledge base (synonymy), and the knowledge base may contain many entities with the same name (polysemy). For example, there are 218 entities with the name *apple* in Freebase, but the right match for the question is actually *Apple Inc.* The relation-matching problem has the same problem, which is even more difficult for k-ary relation with $k > 2$. As a further complication, questions like the above do not contain any word that matches the relations from the sought for query.[2] Note how these problems exacerbate for very large knowledge bases. If we restrict to lexical matches, we will often miss the correct query. If we allow weaker matches, the number of possibilities becomes very large. This will become clearer in Section 3.

1.1 Contributions

We consider the following as our main contributions:

- A new end-to-end system that automatically translates a given natural-language question to the matching SPARQL query on a given knowledge base. Several previous systems factor out part of the problem, for example, by assuming the right entities for the query to be given by an oracle. See Section 3 for an overview of our system.
- An evaluation of our system on two standard benchmarks, Free917 and WebQuestions, where it outperforms all pre-

[1]For the sake of readability, prefixes are omitted from the entity and relation names.

[2]This is typical when the verb *to be* is used in the question.

CIKM'15, October 19–23, 2015, Melbourne, Australia.
 ISBN 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806472.

vious approaches significantly. These two benchmarks exhibit quite different challenges, and many of the existing approaches were evaluated (and work well) only for one of them. See Section 2 for an overview of the existing approaches, and Section 5 for the details of our evaluation.

• Integration of entity recognition in a learning-based approach. Previous learning-based approaches treated this sub-problem in a simplistic manner, or even factored it out by assuming the right entities to be given as part of the problem.

• Using learning-to-rank techniques to learn pair-wise comparison of query candidates. Previous approaches often use parser-inspired log-linear models for ranking.

• We also consider efficiency aspects and take care that all questions can be answered interactively, that is, within one second. Many of the previous systems do not consider this aspect, and take at least several seconds and longer to answer a single query. Again, see Section 5 for some details.

• We make the code of our system publicly available under `http://ad.informatik.uni-freiburg.de/publications` . In particular, this allows reproducing our results. The website also provides various additional useful materials; in particular, a list of mistakes and inconsistencies in the Free917 and WebQuestions benchmarks.

Throughout this paper, we focus on Freebase as the currently largest general-purpose knowledge base. However, there is nothing in our approach specific to Freebase. It works for any knowledge base with entities and (possibly k-ary) relations between them.

2. RELATED WORK

Much recent work on natural-language queries on knowledge bases has focused on two recent benchmarks, both based on Freebase: *Free917* and *WebQuestions*. Section 2.1 gives an overview over this body of work, introducing the two benchmarks on the way. In Section 5, we compare our new method against *all* methods from this section. Section 2.2 briefly discusses work using other benchmarks.

2.1 Work on Free917 and WebQuestions

We consider the works in chronological order, briefly highlighting the relative innovations to previous works and the corresponding gain in result quality. A more technical description of each of the methods is provided in Section 5.3.

In [7], the Free917 benchmark was first introduced. The benchmark consists of 917 questions along with the correct[3] knowledge-base query. All queries have exactly one (possibly k-ary) relation. The basic approach of [7] is to extend an existing semantic parser with correspondences between natural-language phrases and relation names in the knowledge base. The correspondences are learned using weak supervision techniques and from the training portion of the benchmark (70% = 641 questions).

In [15], query candidates are derived by transforming an underspecified logical form of a CCG [21] parse. This form is grounded to Freebase using a set of collapsing and expansion operators that preserve the type of the expression. This has the advantage that it leverages grammatical structure in the question and can adjust knowledge base mismatches, and the disadvantage that it relies on well-formed questions. A linear model is learned to score derivations, which are built using a dynamic programming based parser.

In [2], the WebQuestions (WQ) benchmark was introduced. This benchmark is much larger (5,810 question) but only provides the result set for each question, not the knowledge-base query. This allows gathering more training data more easily (the results were obtained via crowdsourcing). The WQ questions are also more realistic (they were obtained via the Google Suggest API) and language-wise more diverse than the Free917 questions, and hence also harder (e.g. *who runs china in 2011* asking for the former Chinese Premier). The basic approach of [2] is to generate query candidates by recursively generating logical forms. The generation is guided by a mapping of phrases to knowledge base predicates and a small set of composition rules. Candidate scores are learned with a log-linear model.

In a follow-up work [3], the process from [2] is "turned on its head" by again generating a natural-language question from each query candidate. Scores are then learned (again with a log-linear model) based on the similarity between the question representing the query candidate and the original question. This allows leverage of text-similarity information (paraphrases) from large text corpora (unrelated to the queried knowledge base).

In [25], the authors go another step further by not even generating query candidates. Instead their approach tries to identify the central entity of the question, and then iterates over each entity connected (via a single relation) to that central entity in the knowledge base. It is then decided (via a learned model) separately for each such entity whether it becomes part of the result set. In principle, this allows correct answers even when no single relation from the knowledge base matches the question (e.g., asking for a brother of someone, when the knowledge base only knows about siblings). On the downside, this adds a lot of additional features to the learning process (the attributes of the result entities). Quality-wise, the approach does not improve over [2] and [3].

In [19], the authors go yet a step further by not even using the training data. Instead, weak-supervision is used to generate learning examples from natural language sentences. The parsing step itself is conceptualized as a graph-matching problem between the graph of a CCG parse and graphs grounded in Freebase entities and relations. However, their approach was evaluated only on small (and topically narrow) subsets of the two benchmarks.

In [4], the authors try to solve the problem without any natural-language processing (not even POS-tagging). They match the results from [3] but do not improve them.

2.2 Other benchmarks

Another recent notable effort in open-domain question answering is the QALD (Question Answering over Linked Data) series of evaluation campaigns, which started in 2011. See [22] for the latest report. So far, five benchmarks have been issued, one per year. The challenges behind these benchmarks are somewhat different than those behind the Free917 and WebQuestions benchmarks from Section 2.1:

• The biggest and most diverse knowledge base used is DBpedia, which is more than an order of magnitude smaller than Freebase (about 4M vs. about 40M entities).

[3]Actually, a small portion of the queries are incorrect, but this is not a deliberate feature of the benchmark.

• A significant fraction of the questions involves more than one relation or non-trivial comparatives. For example, *what are the capitals of all countries that the himalayas run through* or *which actor was cast in the most movies*.

• The training sets are relatively small (50-100 queries for QALD 1-3). This is mainly due to the fact, discussed in Section 2.1 above, that the ground truth provides not just the correct result sets but also the corresponding SPARQL queries, which requires expensive human expert work. The benchmarks thus give relatively little opportunity for supervised learning. Indeed, most of the participating systems are unsupervised. It is one of the insights from our evaluation in Section 5 that supervised learning is key for results of the quality we achieve.

• QALD 3 and 4 contain multi-lingual versions of the datasets and questions. For QALD 5, the dataset is a combination of RDF data and free text.

For these reasons, and because there is such a substantial body of very recent work on Free917 and WebQuestions with a series of better and better results, we did not include QALD in our evaluation. We consider it a very worthwhile endeavor for future work though, to extend our approach to the QALD benchmarks.

3. SYSTEM OVERVIEW

We first describe our overall process of answering a natural language question from a knowledge base (KB). In the next sections we describe each of the steps in detail. Assume we are trying to answer the following question (from the WebQuestions benchmark):

what character does ellen play in finding nemo?

Entity identification. We begin by identifying entities from the KB that are mentioned in the question. In our example, *ellen* refers to the tv host *Ellen DeGeneres* and *finding nemo* refers to the movie *Finding Nemo*. However, like for the example in the introduction, this is not obvious: *ellen* could also refer to the actor *Ellen Page* and *finding nemo* to the video game with the same name (besides others). Instead of fixing a decision on which entities are mentioned, we delay this decision and jointly disambiguate the mentioned entities via the next steps. Hence, the result of this step is a set of (possibly overlapping) entity mentions with attached confidence scores.

Template matching. Next, we match a set of query templates to the question. Figure 1 shows our templates. Each template consists of entity and relation placeholders. A matched template corresponds to a *query candidate* which can be executed against the KB to obtain an answer.

Our simplest template consists of a single entity and an answer relation (template 1 in Figure 1). One of the query candidates for our example is generated by matching the entity for the tv host *Ellen DeGeneres* and the relation *parents* [4]:

<*Ellen DeGeneres*> <*parents*> <*T*>

This has the (wrong) interpretation of asking for her parents. A slightly more complex template contains two relations connected to the entity via a *mediator object* (template 2 in Figure 1). In our example, this matches a query candidate connecting *Ellen Page* to abstract *film performance* objects, via a *film performance* relation, and from there to all the films she acted in via a *film* relation:

<*Ellen Page*> <*performance*> <*M*>
<*M*> <*film*> <*T*>

This asks for all films *Ellen Page* acted in. Yet another template combines two entities via relations and a mediator entity (m in template 3 in Figure 1). In our example, *Ellen DeGeneres* and *Finding Nemo* are connected via two relations and a film-performance mediator.

<*Ellen DeGeneres*> <*performance*> <*M*>
<*M*> <*film*> <*Finding Nemo*>
<*M*> <*character*> <*T*>

We find this connection using an efficient inverted index (see Section 4.2) and continue matching from the mediator. In particular, we create query candidates asking for the *character* (Dory) and *performance type* (Voice) of *Ellen DeGeneres* in *Finding Nemo*. The final result of this step is a set of all the matched query candidates.

Relation matching. The query candidates still miss the fundamental information about which relations were actually mentioned and asked for in the question. We distinguish three ways of matching relations of the query candidate to words in the question: 1) via the name or description of the relation in the KB, 2) via words learned for each relation using distant supervision, 3) via supervised learning on a training set. Each match has a confidence score attached.

In our example, a word learned for the relations *performance* and *film* connecting an actor to the film she acted in is *play*. This matches in the query candidates asking for all films of *Ellen Page* and for the performance type or character of *Ellen DeGeneres* in *Finding Nemo*. Furthermore, the word *character* matches the relation with the same name, whereas the relation *performance type* doesn't match. Continuing this way, all relations in all query candidates are enriched with information about what words were matched in which way.

Ranking. We now have a set of query candidates, where each candidate is enriched with information about which of its entities and relations match which parts of the question how well. It remains to rank the candidates in order to find the best matching candidate. Note that performing ranking at this final step has the strong benefit of jointly disambiguating entities and relations. A candidate can have a weak match for an entity, but a strong match for a relation, and vice versa. By deciding this at the final stage we can identify these combinations as correct, even when one of the matches seems unlikely when considered separately.

Intuitively, for our example, the candidate covering most words of the question is best. Matching *ellen* to *Ellen Page* does no longer allow matching *Finding Nemo* because these aren't actually related in the KB. On the other hand, asking for the performance type of *Ellen DeGeneres* in *Finding Nemo* doesn't match the word *character*. This leaves us with the correct interpretation of asking for her character in the movie.

4. SYSTEM DETAILS

In this section, we describe the details of our system, called *Aqqu*. Aqqu works by generating query candidates for each

[4]We use SPARQL-like triple (subject, predicate, object) notation, where uppercase characters indicate variables.

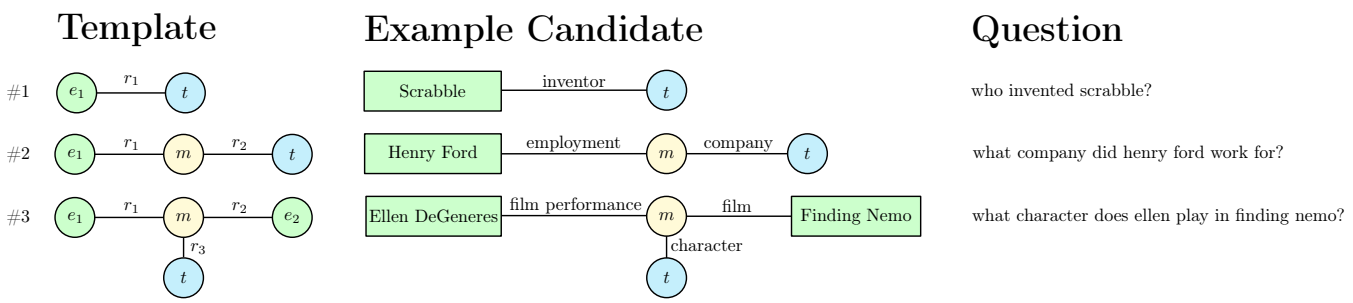

Figure 1: Query templates and example candidates with corresponding questions. A query template can consist of entity placeholders e, relation placeholders r, an intermediate object m and the answer node t.

question. These query candidates are then ranked using a learned model. The top-ranked query is then returned (or "no answer" in case the set of candidates was empty). The following subsections describe the candidate generation and ranking in detail. The previous section explained the process by an example.

4.1 Entity matching

The goal of the entity matching phase is to identify all entities from the knowledge base that match a part of the question. The match can be literal, or via an alias of the entity name.

POS-tagging We POS-tag the question using the Stanford tagger [17]. For entity matching (this subsection), we make use of the tags *NN* (noun) and *NNP* (proper noun). For relation matching (Section 4.3), we also make use of the tags *VB* (verb) and *JJ* (adjective).

Subsequence generation We generate the set S of all subsequences of words from the question, with the following two restrictions. First, a subsequence consisting of a single word must be tagged *NN*. Second, a subsequence must not "split" a sequence of words tagged *NNP*; that is, when it starts (ends) with a word tagged *NNP*, it must not be preceded (succeeded) by a word tagged *NNP*.

Find matching entities For each $s \in S$, we compute the list of all entities from the knowledge base that have s as their name or alias. We use a map from phrases (the aliases) to lists of entities (the entities with the respective alias) obtained from the CrossWikis dataset [20]. CrossWikis was built by mining the anchor text of links to Wikipedia entities (articles) from various large web-crawls. CrossWikis covers around 4 million entities from Wikipedia. Almost all of these entities also exist in Freebase, together with a link to the respective Wikipedia entity. For the remaining Freebase entities, we only consider the literal name match. Overall, we are able to recognize around 44 million entities with about 60 million aliases.

We have also experimented with the aliases provided by Freebase, but they tend to be much more noisy (wrong aliases) and less complete (important aliases missing).

Scores for the entity matches We compute a score for each match s, e computed in the previous step, where s is a subsequence of words from the question and e is an entity from Freebase with alias s. Consider a fixed alias s. CrossWikis also provides us with a probability distribution $p_{cross}(e|s)$ over the Wikipedia entities e with alias s. Let e' be a Freebase entity that is not contained in CrossWikis. Let e_{max} be the CrossWikis entity with the highest $p_{cross}(e|s)$. That is, e_{max} is the most likely Wikipedia entity for alias s. Let $p_{free}(e'|s) = p(e_{max}|s) \cdot pop(e')/pop(e_{max})$, where pop is the (alias-independent) popularity score of an entity, as described in the next subsection. Intuitively, $p_{free}(e'|s)$ estimates the probability that e' has alias s via its relative popularity to the most likely Wikipedia entity for s. We merge $p_{cross}(e|s)$ and $p_{free}(e'|s)$ into one probability distribution by simply normalizing the probabilities to sum 1.

Popularity scores for each entity For each entity, we also compute a (match-independent) popularity score. We simply take the number of times the entity is mentioned in the ClueWeb12 dataset [9], according to the annotations provided by Google [13]. The popularity scores are used for the entity match scores above. They also yield two features used in ranking each candidate; see Section 4.5.

4.2 Candidate generation

Based on the entity matches, we compute a set of query candidates as follows. We generate the query candidates in three (disjoint) subsets, one for each of the three templates shown in Figure 1. Each template stands for a query with a particular kind of structure. These three templates cover almost all of the questions in the Free917 and WebQuestions benchmarks.

Let E be the set of all entities matched to a subsequence of the question, as described in the previous section.

Template 1 For each $e \in E$, find all relations r such that there is some triple $(e, r, \cdot)$ in the knowledge base. We obtain these via a single SPARQL query for each e.

Template 2 For each $e \in E$, find all r_1, r_2, m such that there are two triples (e, r_1, m) and $(m, r_2, \cdot)$ in the knowledge base, where r_1 and r_2 are relations and m is a mediator entity. We obtain these as follows. For each e, we use a single SPARQL query to obtain all matching r_1. For each e, r_1, we then use another SPARQL query to obtain all matching r_2. Note that m remains a variable in the query candidate.

Template 3 For all pairs of entities $e_1, e_2 \in E$ such that the two subsequences matched in the question do not overlap, find all r_1, r_2, r_3 such that there are three triples (e_1, r_1, m), (m, r_2, e_2), and $(m, r_3, \cdot)$ in the knowledge base, where r_1, r_2, r_3 are relations and m is a mediator entity. We obtain these as follows. For each entity e, we precompute the list of all (r, m) such that m is a mediator entity and the triple (e, r, m) exists in the knowledge base. The list is sorted by the ids of the mediator entities. For given e_1, e_2 like above, we then intersect the lists for e_1 and e_2. For each

mediator m in the intersection, we then obtain all r_3 via a simple SPARQL query. In the query candidate, m remains a variable.

4.3 Relation matching

Let C be the set of query candidates computed in the previous subsection. For each query candidate $c \in C$, let RW_c be the set of lemmatized[5] words from the relations from c (there can be one, two, or three relations, depending on the template from which c was generated). We compute how well the words from RW_c match the subset QW of lemmatized words from the question that are not already matched by the entities from c.

We consider four kinds of matches, described in the following: literal, derivation, synonym, context. For each of these four kinds of matches, we compute a non-negative score (which is zero, if there is no match at all). It can happen that all four of these scores are zero. In the basis version of our system, we keep such candidates, in a variant we prune them; see Section 4.7.

Literal matches This score is simply the number of pairs w, q, where $w \in RW_c$ and $q \in QW$ and $w = q$. Almost all questions have no repeated words; in that case, this score is just the number of relation words that occur in the question (and are not already matched by an entity).

Derivation matches This score is the number of pairs w, q, where $w \in RW_c$ and $q \in QW$ and w is derivationally related to q. Here we also consider the POS-tag of w in the question. We precompute a map from POS-tagged words to derivations using WordNet [11]. We extract derivation links for verbs and nouns (e.g. *produce.VB - producer.NN* and vice versa). We also extract attribute links between adjectives and their describing attribute (e.g., *high.JJ - height.NN*). We extend these links with synonyms of the noun in WordNet (e.g. *high.JJ - elevation.NN*).

Synonym matches For each $w \in RW_c$ and $q \in QW$, add s to this score if w is a synonym of q with similarity s. We compute the similarity between two words by computing the cosine similarity between the associated *word vectors*. We use 300-dimensional word vectors that were computed with Google's *word2vec* on a news text corpus of size around 100 billion words.[6] We consider only synonyms, where the score is ≥ 0.4. This threshold is based on observation, but chosen very liberally: many word pairs with score above that threshold are not what humans would call "real synonyms", but almost all such "real synonyms" have a score above that threshold.

Context matches For this score, we precompute weighted *indicator* words for each relation from our knowledge base. These are words which are not necessarily synonyms of words in the relation name, but are used in text to express that relation; see below for an example. The score is then the sum of the weights of all words in QW that are indicators for one of the relations from the query candidate. For templates 2 and 3, we consider $r1.r2$ as one relation.

We learn indicator words using distant supervision [18] as follows. First, we identify entity mentions in Wikipedia using Wiki markup and a set of simple heuristics for coreference resolution, as described in [1]. We also identify dates and values using SUTime [8]. For the 23 million sentences that contain at least two entities (including dates or values), we compute a dependency parse using [17].

For each pair e_1, e_2 of entities occurring in a sentence, we look up all relations r in the knowledge base that connect them. We also treat relations $r1.r2$ that connect the entities via a mediator as a single binary relation r. If the shortest path between e_1 and e_2 in the dependency parse has length at most four, we consider all words along that path as indicator words for r. We also experimented with considering all words in the sentence, or words along longer paths, but these gave considerably worse results.

We find about 4.7 million sentences that match at least one relation this way. For example, we can thus learn that *born* is an indicator word for the relation *place of birth* from the following sentence (assuming that our knowledge base contains the respective fact):

Andy Warhol was born on August 6, 1928 in Pittsburgh.

Note that from the same sentence, we can also learn that *born* is an indicator word for the relation *date of birth*. To distinguish between the two, we need some kind of answer type matching; this is described in Section 4.4.

We compute the weights for the indicator words in the following IR-style fashion. Consider each relation as a document consisting of the words extracted for that relation. Then compute tf.idf scores for all the words in these (relation) documents in the usual way. For each relation, then only consider the top-1000 words and sum up their tf.idf scores. The weight for each word in a (relation) document is then its tf.idf score divided by this sum. This could also be interpreted as a probability distribution $p(w|r)$ over words w given a relation r.

4.4 Answer type matching

For each candidate, we perform a simple but effective binary check based on the relation leading to the answer ($r1$, $r2$, $r3$ for templates 1,2 and 3, respectively). We precompute a list of target types for each relation r by counting the types of objects o in all triples $(\cdot, r, o)$, keeping only the top ten percent of most frequent types. For questions starting with *who*, we check whether the computed target types contain the type person, character, or organization. For questions starting with *where*, we check whether the relation leads to a location or an event. For questions starting with *when* or *since when*, we check whether the type is a date; for all other questions, the check for target objects of type date is negative.

As our evaluation and error analysis shows, these simple heuristics work reasonably well for the Free917 and WebQuestions benchmarks. The reason is that our entity and relation matching already provide ample information for discriminating between candidates. However, as explained in Section 4.3, a question word like *born* alone does not permit discrimination between the two relations *place of birth* and *date of birth*. However, it is exactly those cases that can be easily discriminated with the simple answer-type check from above.

We leave elaborate answer-type detection (which has been addressed by many QA systems) to future work.

4.5 Candidate features

The previous subsections have shown two things. First, how we generate query candidates for a given question. Sec-

[5]For example, *founded* → *found* and *was* → *be*.

[6]`https://code.google.com/p/word2vec/`

ID	Description
1	number of entities in the query candidate
2	number of entities that matched exactly with their name, or with a high probability (> 0.8)
3	number of tokens of all entities that matched literally as per the previous feature
4-5	average (4) and sum (5) of entity match probabilities
6-7	average (6) and sum (7) of entity match popularities
8	number of relations in matched template
9	number of relations that were matched literally via their name
10-13	number of tokens that matched a relation of kind: literal (10), derivation (11), synonym (12), context (13)
14	sum of synonym match scores
15	sum of relation context match scores
16	number of times the answer relation (r_1, r_2, r_3 for templates 1, 2 and 3 respectively) occurs in the KB
17	a value between 0 and 1 indicating how well the relation matches according to n-gram features (Section 4.5)
18	sum of features 3 and 10; that is, the number of tokens matching a relation or entity literally
19	number of tokens that match an entity or relation divided by the total number of tokens in question
20-22	whether the result size is 0 (feature 20), 1-20 (feature 21), or larger than 20 (feature 22); all binary
23	binary result of the answer-type check (Section 4.4)

Table 1: Features used by our ranking approaches. Top/middle/bottom: features for entity matches/features for relation matches/combined or other features.

ond, how we compute various scores for each candidate that measure how well the entities and relations from the candidate match which parts of the question.

In this subsection, we show how we generate a feature vector from each candidate. Most of these features are based on the scores just mentioned. Another important feature, described below, serves to learn the correspondence between n-grams from the question and relations from query candidates. Table 1 provides an overview over all our features. In the description below, we refer to the features by their ID (first column in the table). In Section 4.6, we show how we rank candidates based on these feature vectors.

Entity/Relation matching features Features 1-7 are based on the results from the entity matching described in Section 4.1. Features 8-16 are based on the results from the relation matching step described in Section 4.3. Features 18 and 19 quantify the number of words in the question covered by entity or relation matches (feature 18 = literally, feature 19 = in any way). Features 20-22 quantify the result size. This is important, because some candidates produce huge result sizes or empty results sets, which are both rare. Feature 23 is the binary output of the simple answer-type check from Section 4.4.

N-Gram relation matching feature This feature considers correspondences between words (unigrams) or two-word phrases (bigrams) in the question and the relation in the query candidate. For example, in the WebQuestion benchmark, the question *who is ...* almost always asks for the *profession* of a person. Such a correspondence cannot be learned by any of the mechanisms described in Section 4.3. We learn this feature as follows.

For each query candidate, we generate all unigrams and bigrams of the lemmatized words of the question. The matched entities (Section 4.1) are replaced with a special word *entity*. For each n-gram, we then create an indicator feature by appending the n-gram to the relation names of the candidate. For example, for template 2 from Figure 1, one of the features would be *employment.company+work* for the uni-gram *work* and the relations *employment.company*. We then train an $L2$-regularized logistic regression classifier with all correct candidates as positive examples and all others as negative examples. The value of feature 17 is simply the (probability) output by this classifier.

This feature will be part of a subsequent step to learn a ranking that uses the same training data. To provide realistic feature values (that aren't overfit) we proceed as follows. Split the training data into six folds. In turn, leave out one fold and train the n-gram feature classifier on the remaining folds. Then, for each example in the left-out fold compute the n-gram feature value. Use this computed value as part of the training data for subsequent learning.

4.6 Ranking

For each question, we finally rank the query candidates using the feature vectors described in the previous subsection. The top-ranked query candidate is then used to provide the answer. We say "no answer" only when the set of candidates is emtpy; this is discussed in Section 4.7 below.[7]

We have experimented with state-of-the-art techniques for the learning-to-rank approach from IR [14] [16], including: RankSVM [14], RankBoost [12], LambdaRank [6] and AdaRank [23]. These only lead to moderate results and were outperformed by our approaches described below. We presume that this is because our ranking problem is degenerate. In particular, each query is only associated with a single *relevant* answer. This is different from a typical IR scenario where a query usually has several answers, sometimes with varying degrees of relevance.

We investigate two variants to obtain a ranking: *pointwise ranking* and *pairwise ranking*. These approaches are inspired by the learning-to-rank approaches from IR.

[7]Both benchmarks contain a considerable number of questions starting with *how many ...*, asking for a count. We simply replace *how many* by *what* in these questions, and count the size of the result set (unless the answer already is a count).

Pointwise ranking In the pointwise ranking approach we compute a score for each candidate. Candidates are sorted by this score to infer a ranking. The score is computed by a classifier learned on the candidate features (see Section 4.5) and training data. We create training data by using the correct candidate of each question as positive examples and all other candidates as negative examples.

A drawback of the pointwise approach is that the model "compares" question-independent examples. That is, correct (incorrect) query candidates of questions of different type and difficulty are in the same correct (incorrect) class, when in practice it is not necessary to compare or discriminate between them.

Pairwise ranking In the pairwise ranking approach, we transform the ranking problem into a binary classification problem. The idea is to learn a classifier that can predict for a given pair of candidates, whether one should be ranked before the other.

To infer a ranking, we sort the list of candidates using the learned preference relation. This works very well in practice, although our learning does not guarantee that the learned relation is transitive or anti-symmetric. We have experimented with two alternatives to sorting. Simply computing the maximum turned out to perform badly. This makes sense, because the maximum has to "survive" a larger number of comparisons. Following [10], we have also sorted the candidates by their number of "won" comparisons against all other candidates. The results were identical to those for sorting, but this method requires $\Theta(n^2)$ comparisons for n candidates.

To train the classifiers we create training examples in the following way. For a question with n query candidates, randomly select $n/2$, but at least 200 candidates (or n if $n/2 < 200$) . This is to guarantee that we have enough training examples for questions with few candidates and to avoid putting too much emphasis on questions that have more than 200 candidates.[8] Then, for each randomly selected candidate r_i and the correct candidate c, where $r_i \neq c$, create a positive example pair (c, r_i) and a negative example pair (r_i, c). The feature representation for a pair (a, b) is a tuple of the individual feature vectors and their difference: $\phi_{pair}(a, b) = (\phi(a) - \phi(b), \phi(a), \phi(b))$, where ϕ is a function extracting the features in Table 1.

Both ranking approaches, pointwise and pairwise, require a classifier. Here, we consider two different options.

Linear A logistic regression classifier. In initial experiments, other linear models, such as linear SVMs, have shown similar performance. Logistic regression is also known to output well calibrated probabilities and performs well in high-dimensional feature spaces. We train the model using L-BFGS-B [26]. To avoid over-fitting we apply $L2$-regularization choosing the regularization strength using 6-fold cross-validation on the training set.

Random forest We learn a forest of decision trees [5]. Random forests are able to learn non-linear decision boundaries, require few hyperparameters, are simple to train, and are known to perform very well on a variety of tasks.

[8]Our system generates around 200 candidates on average for a random question, but the exact value had little effect on performance in our evaluation.

4.7 Candidate pruning

Some questions may have no answers in the knowledge base. Our system, as described so far, returns "no answer" only when the set of query candidates is empty. However, as also described, this would rarely happen, since there are matching entities for every question, and we do not require that the relations match any of the words in the question.[9]

We consider two variants of our system to deal with this problem: (1) omitting the n-gram feature, and using hard pruning; and (2) keeping the n-gram feature, and using a pruning-classifier. Note that a nice side-effect of pruning is that it speeds up the ranking process because it needs to consider less candidates.

Without n-grams, with hard pruning When omitting the n-gram feature, there is no reason to keep candidates with the wrong answer type or where features 9-15 are all zero. The natural approach is then to prune such candidates before we do the ranking; this is what we call *hard pruning*. Hard pruning naturally leads to empty candidate sets for some queries. Indeed, on the Free917 benchmark, 10 questions have no answer, and our hard pruning yields an empty candidate set for 7 of them.

With n-grams, with a pruning classifier When keeping the n-gram feature, hard pruning as just described would be counterproductive. As explained in Section 4.5, the answers for the *who is ...* questions from the WebQuestions benchmark are professions. They would be eliminated when hard pruning by answer type. Also, the *profession* relation matches no words from these questions. They would hence also be eliminated when hard pruning if features 9-15 are all zero.

The goal of the pruning classifier is to weed out only the "obviously" bad candidates. For example, candidates that do not match the answer type, have bad relation matches, *and* a weak n-gram feature. We train the pruning classifier in the same way as the pointwise classifiers (see above) with the features from Table 1 using logistic regression. To optimize the classifier for recall we adjust example weights so that positive candidates have twice the weight of negative candidates. Before the ranking step, we apply the classifier to each candidate and only keep candidates classified positively.

5. EVALUATION

We perform an extensive evaluation of our system. In Section 5.1, we provide more details on our two benchmarks. In Section 5.2, we describe the evaluation measures used. In Section 5.3, we describe the systems we evaluate and compare to. In Section 5.4, we provide our main results followed by a detailed analysis in Section 5.5.

5.1 Data

We use all of Freebase as our knowledge base (2.9 billion facts on 44 million entities). Note that our approach is not tailored to Freebase and could easily be adapted to another knowledge base, e.g., WikiData[10].

Datasets We evaluate our system on two established benchmarks: *Free917* and *WebQuestions*. Each benchmark consists of a set of questions and their answers from Freebase.

[9]In that case, features 9-15 are all zero; however, the n-gram features could still be positive.

[10]`http://www.wikidata.org`

The benchmarks differ substantially in the types of questions and their complexity.

Free917 contains 917 manually generated natural language questions [7]. The questions cover a wide range of domains (81 in total). Two examples are *what fuel does an internal combustion engine use* and *how many floors does the white house have*. The most common domains, *film* and *business*, only make up 6% of the questions [7]. All questions are grammatical and tend to be tailored to Freebase. The dataset provides a translation of each question into a SPARQL-equivalent form. We execute the SPARQL queries to obtain a gold answer for each question. [7] also provide an entity lexicon: a mapping from exact text to the mentioned entity for all entities appearing in the questions. This lexicon consists of 1014 different entities. It was used for identifying entities by all systems reporting results on the dataset so far. We only make use of this lexicon where explicitly stated. To report results, we use the original split of the questions by [7] into 70% (641) questions to train and 30% questions (276) to test.

WebQuestions consists of 5,810 questions that were selected by crawling the Google suggest API [2]. Contrary to Free917, questions are not necessarily grammatical and are more colloquial. For example: *where did jackie kennedy go to college* and *what is spoken in czech republic*. Due to how they were selected, the questions are biased towards topics that are frequently asked from Google. According to [19], the *people* domain alone makes up about 7% of questions. Furthermore, the structure of questions tends to be simpler. Most questions only require a single entity with an answer relation [2]. Answers to the questions were obtained by using crowdsourcing. This introduces additional noise; in particular, for some questions only a subset of the correct answer is provided as gold answer. We use the original train-test split of the questions by [2] into 70% (3,778 questions) to train and 30% (2,032 questions) to test.

5.2 Evaluation measures

Given a benchmark and a system, denote the questions by $q_1...q_n$, the gold answers by $g_1...g_n$, and the answers from the system by $a_1...a_n$. Note that an answer can consist of a single value (in particular, a date or a literal) or a list of values. We consider the following two evaluation measures.

Accuracy The fraction of queries answered with the exact gold answer:

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} I(g_i = a_i)$$

where $I(e)$ is an indicator function returning one if expression e is true and zero else. This is reasonable on Free917 which provides perfect gold answers.

Average F1 The average F1 across all questions:

$$\text{average F1} = \frac{1}{n}\sum_{i=1}^{n} F1(g_i, a_i)$$

where the function $F1$ computes F1 in the regular way. This accounts for partially correct results, which is reasonable for WebQuestions, where gold answers are sometimes incomplete.

In our evaluation we focus on accuracy for Free917 and average F1 for WebQuestions. These are the most reported and most intuitive measures for these datasets. We also performed the evaluation with other measures that were used in previous work, e.g., variants of F1 as defined in [15] and [25]. These provided no new insights and strongly correlated with the measures above.

	Free917		WebQuestions
Method	Accuracy+	Accuracy	Average F1
Cai+Yates	59 %	–	–
Jacana	–	–	35.4 %
Sempre	62 %	52 %	35.7 %
Kwiat. et al	68 %	–	–
Bordes et al	–	–	39.2 %
ParaSempre	68.5 %	46 %	39.9 %
Aqqu	**76.4 %**	**65.9 %**	**49.4 %**

Table 2: Results on the Free917 (267 questions) and WebQuestions (2032 questions) test set. For the results in the second column (Accuracy+) a manually crafted entity lexicon was used.

5.3 Systems evaluated

We evaluate and compare the following systems. See Section 2 for a brief description of the systems from previous work. If we (re-)produced results, we explicitly state so. Otherwise, we report existing results.

Cai+Yates The semantic parser developed by [7].

Kwiat. et al The semantic parser by [15].

Sempre The semantic parser by [2]. We produced results for Free917 without an entity lexicon using the provided code.[11]

ParaSempre The semantic parser suggested by [3]. We used the code provided by the authors[11] to produce results on Free917 without an entity lexicon.

GraphParser The semantic parser developed by [19]. We report results obtained from the code provided by the authors[12]. The results from their code slightly deviates from the results reported in their paper.

Jacana The information extraction based approach by [25]. We report updated results from [24].

Bordes et al The embedding-based model by [4].

Aqqu Our system, as described in Section 4. We want to stress that we use the exact same system on both benchmarks. As shown in Section 5.5 below, results can be further improved by adapting the feature set to the benchmark. However, we consider this overfitting. Note that all of the systems above, except Sempre and ParaSempre, were only evaluated on one of the two benchmarks.

5.4 Main results

Table 2 shows the results on the test sets for Free917 and WebQuestions for all the systems from Section 5.3. GraphParser is discussed separately below, because it was evaluated only on a subset of questions.

On Free917, Aqqu improves in accuracy over the best previous systems by 8% with an entity lexicon, and by 14% without entity lexicon. Performance drops considerably for all systems when not using an entity lexicon. This shows

[11] `http://github.com/percyliang/sempre`

[12] `http://github.com/sivareddyg/graph-parser`

	Top-2	Top-3	Top-5	Top-10
Free917	74.3 %	77.2 %	79.3 %	83.7 %
WebQuestions	67.1 %	72.7 %	77.5 %	82.3 %

Table 3: Top-k results on Free917 (top) and WebQuestions (bottom). Percentage of questions with the best answer in the top-k candidates.

	Free917		WebQuestions
Method	Acc+	Acc	Avg F1
Aqqu-point-lin	73.6 %	63.4 %	46.9 %
Aqqu-point-tree	74.3 %	63.0 %	47.9 %
Aqqu-pair-lin	76.4 %	65.2 %	48.3 %
Aqqu-pair-tree	76.4 %	65.9 %	49.4 %

Table 4: Results for different ranking variants on the test sets for Free917 and WebQuestions. For the results in the second column (Acc+) a manually crafted entity lexicon was used.

that addressing entity recognition is an integral part of the problem that cannot be ignored. Overall, we achieve an oracle accuracy (percentage of questions where at least one produced query candidate is correct) of 89.1% and 85.5%, with and without entity lexicon respectively. This indicates that there is still room for improvement for better matching and ranking.

On WebQuestions our system improves the state of the art by almost 10% in average F1. No system uses an entity lexicon. Note that the WebQuestions benchmark is much harder and contains a considerable amount of imperfect or wrong answers. Out of a random sample of 55 questions we found 9 questions that had a wrong answer, and 10 further question that had only a partially correct answer. This suggests that the upper bound for average F1 is roughly around 80%. Our oracle average F1 is at 68.5%. [2] and [3] report 48% and 63% respectively. Hence, we successfully identify most of the entities and relations. However, there is still much room for improvement in ranking and matching.
GraphParser was evaluated only on a subset of Freebase relations. The authors provide a train-test split of questions for WebQuestions. Note that we didn't restrict our system to the specific relations and that GraphParser requires an entity lexicon also on WebQuestions. Our system (without an entity lexicon) scores an average F1 of 66.1 % compared to 40.5 % reported for GraphParser. The selected subset of relations and thus questions seems to be considerably easier to answer for our system.

5.5 Detailed analysis

Top-k results Table 3 shows the top-k results on the two datasets. A large majority of questions can be answered from the top two or three candidates. By providing these interpretations and results (in addition to the top-ranked candidate) to a user, many questions can be answered correctly. Note that on WebQuestions some questions only have an imperfect gold answer with an F1 score smaller than one. Therefore, the percentage of best answers in the top candidates can be slightly larger than the resulting average F1.

Ranking variants As described in Section 4.6, we consider two possible ranking methods: pointwise (*point*) and pairwise (*pair*), each with two different ranking classifiers: logistic regression (*lin*) and random forests (*tree*). This gives a total of four different combinations.

	Free917	WebQuestions
best previous	52.0 %	39.9 %
best now	**69.2 %**	**49.4 %**
no n-grams, all other	69.2 %	39.6 %
no n-grams, no lit-match	65.2 %	39.6 %
no n-grams, no synonyms	61.6 %	28.2 %
n-grams, all other	65.9 %	49.4 %
n-grams, no pruning	64.4 %	49.3 %
n-grams, no synonyms	62.0 %	48.0 %
n-grams, nothing else	18.1 %	43.8 %

Table 5: Feature analysis for Free917 and WebQuestions. No synonyms disables features 11-15 and no lit-match features 2, 3, 9, 10 and 18. When not using the n-gram feature a different type of candidate pruning is performed (see text).

.

Table 4 shows results of all four ranking variants with the full features of Table 1. On both benchmarks, pairwise ranking is more effective than pointwise ranking. This is consistent with our intuition that learning a pairwise comparator is better (see Section 4.6). Furthermore, random forests are slightly more effective than a weighted linear combination. We therefore use pairwise ranking with random forests as a standard choice.

Feature analysis To gain insight into which features are helpful we evaluate our system with different combinations. Table 5 shows the results. Note that, as described in Section 4.7, without the n-gram feature hard pruning is applied. The following main observations can be made.

The n-gram feature is extremely helpful on WebQuestions but slightly detrimental on Free917. The WebQuestions benchmark contains many questions that are hard to answer without this kind of supervision, e.g., *where is reggie bush from?* (asking for the place of birth) or *what to do downtown san francisco?* (asking for tourist attractions). Our system is able to successfully learn important features for these from the training set. On the other hand, the small Free917 benchmark covers a wide range of domains and relations with only few repetitions. N-gram features aren't helpful on this dataset, which is shown by the low performance when only using the n-gram feature (18.1%). Note that the ranking and learning problem is inherently more difficult when the number of possible candidates increases. This is the case when not using hard pruning which goes along with using the n-gram feature (see Section 4.7). This disadvantage cannot be fully compensated by the weak n-gram feature and leaner pruning and, as a result, the score drops by about 3% for Free917. Still, we consider it more important to have a single approach that performs well on different kinds of datasets than to optimize for a single dataset.

Literal features provide a small benefit for Free917 but no benefit on WebQuestions. This is an artefact of the way Free917 was built. Free917 questions are tailored to Freebase, often using words from the relation name as part of the

question. Synonym features are important for both datasets. They give a huge benefit on WebQuestions without the n-gram feature but only a small benefit on top of it.

Finally, the pruning classifier used with the n-gram feature helps on Free917 because it allows to return "no answer" for some questions that have no answer in the knowledge base. The difference on WebQuestions (which always has an answer in the knowledge base) is not significant, and shows that the pruning classifier doesn't negatively affect performance.

Manual error analysis We manually inspected the errors our system makes. Many errors are due to mistakes in the benchmarks (partially or completely wrong gold answers) and inconsistencies in the knowledge base (different relations with contradicting answers on the same piece of information). We provide a list on our website, see the link in Section 1.1.

On that website, we also provide a list of errors due to our system. There is no single large class of errors worth pointing out though.

Efficiency We also evaluated the performance of our system. The average response time for a question is 644 ms for Free917 and 900 ms for WebQuestions.[13] None of the other system from Section 5.3 comes with an efficiency evaluation. For systems that provide code and for which we reproduced results, runtimes are (at least) several seconds per query. Training our system on the large WebQuestions benchmark takes about 90 minutes in total.

6. CONCLUSION

We have presented Aqqu, a new end-to-end system that automatically translates a given natural-language question to the matching SPARQL query on a knowledge base. The system integrates entity recognition and utilizes distant supervision and learning-to-rank techniques. We showed that our system outperforms previous state-of-the-art systems on two very different benchmarks by 8% and more. Aqqu answers questions interactively, that is, within one second.

For around 80% of the queries, the correct answer is among the top-5 candidates. This suggests that a more interactive approach, which asks the user's feedback for critical decisions (e.g., between two relations), could achieve a significantly further improved accuracy.

7. REFERENCES

[1] H. Bast, F. Bäurle, B. Buchhold, and E. Haussmann. Broccoli: Semantic full-text search at your fingertips. *CoRR*, abs/1207.2615, 2012.

[2] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic Parsing on Freebase from Question-Answer Pairs. In *EMNLP*, pages 1533–1544, 2013.

[3] J. Berant and P. Liang. Semantic Parsing via Paraphrasing. In *ACL*, pages 1415–1425, 2014.

[4] A. Bordes, S. Chopra, and J. Weston. Question Answering with Subgraph Embeddings. *CoRR*, abs/1406.3676, 2014.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[13] Answer times are averaged over three runs on a server with Intel E5649 CPUs, 90GB of RAM and warm SPARQL caches.

[6] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, pages 193–200, 2006.

[7] Q. Cai and A. Yates. Large-scale Semantic Parsing via Schema Matching and Lexicon Extension. In *ACL*, pages 423–433, 2013.

[8] A. X. Chang and C. D. Manning. Sutime: A library for recognizing and normalizing time expressions. In *LREC*, pages 3735–3740, 2012.

[9] ClueWeb, 2012. The Lemur Projekt.

[10] W. W. Cohen, R. E. Schapire, and Y. Singer. Learning to order things. *JAIR*, 10:243–270, 1999.

[11] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

[12] Y. Freund, R. D. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *JMLR*, 4:933–969, 2003.

[13] E. Gabrilovich, M. Ringgaard, and A. Subramanya. FACC1: Freebase annotation of ClueWeb corpora, Version 1.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, pages 133–142, 2002.

[15] T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. Scaling Semantic Parsers with On-the-Fly Ontology Matching. In *EMNLP*, pages 1545–1556, 2013.

[16] T. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *ACL*, pages 55–60, 2014.

[18] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *ACL*, pages 1003–1011, 2009.

[19] S. Reddy, M. Lapata, and M. Steedman. Large-scale Semantic Parsing without Question-Answer Pairs. *TACL*, 2:377–392, 2014.

[20] V. I. Spitkovsky and A. X. Chang. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *LREC*, pages 3168–3175, 2012.

[21] M. Steedman. *The syntactic process*, volume 35. MIT Press, 2000.

[22] C. Unger, C. Forascu, V. Lopez, A. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter. Question answering over linked data (QALD-4). In *CLEF 2014*, pages 1172–1180, 2014.

[23] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR*, pages 391–398, 2007.

[24] X. Yao, J. Berant, and B. V. Durme. Freebase QA: Information Extraction or Semantic Parsing? In *ACL, Workshop on Semantic Parsing*, 2014.

[25] X. Yao and B. V. Durme. Information Extraction over Structured Data: Question Answering with Freebase. In *ACL*, pages 956–966, 2014.

[26] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, 1997.