Question/Answer Matching for Yahoo! Answers Using a Corpus-Based Extracted Ngram-based Mapping

Stalin Varanasi, Günter Neumann
University of Saarland and MLT-Lab DFKI
D-66123 Saarbrücken, Germany

Abstract

This report describes the work done by the QA group of the Multilingual Technologies Lab at DFKI for the 2015 edition of the TREC LiveQA track. We describe the system, issues faced and the approaches followed considering the time lines of the track.

1. Introduction

The basic functionality of a QA system is to automatically answer a given Natural Language question. The TREC LiveQA 2015 track posed a particularly new challenge from previous QA challenges (like TREC and CLEF) as the questions are taken from the Yahoo! Answers (YA) website and they are far from being factoid questions. We describe briefly how we tried to solve this problem.

A major aspect of the "Live QA" character of the challenge is that one has to deal with real user questions that are extracted from a stream of most recent questions submitted to YA, which have not yet been answered by humans. Furthermore, the allowed processing time of a YA question is limited to 60 seconds, which restricts the development and usage of time consuming complex processing strategies.

A number of recent published approaches in the area of community Question Answering (cQA) explore an existing Q-A archive (for example the YA archive) for efficient retrieval of known answers for new questions, cf. (Figueroa and Neumann, 2014), (Shen et al., 2015). Here, the answer sources (i.e., the answers associated with the questions) are given explicitly, and hence, the main focus lies on the mapping of new questions to known questions. We might consider this as the "question paraphrasing" problem of cQA.

Since for the LiveQA challenge a YA archive cannot directly be used as answer, a major problem is to identify possible answer sources for a question, which might contain the candidate answer. This is in particular difficult, because YA questions can be and usually are about anything from factoid questions like "Who invented the xbox 360?"[1], to non-factoid questions "Why is Fairy Tail Episode 176 not out yet?"[2], to very personal questions like "I hate my dad. anyone else?"[3]. Thus a major challenge is: How can we infer useful answer sources for a YA question?

[1] https://answers.yahoo.com/question/index?qid=20090211135728AArYykb
[2] https://answers.yahoo.com/question/index?qid=20130405112159AAMsPyV
[3] https://answers.yahoo.com/question/index?qid=20080602133015AAvvyBf

In our previous work on Question Answering we have successfully explored and developed a Web-based approach (the implemented QA system was named WebQA), in which the whole Internet was exploited as answer source for answering different kinds of questions, e.g., factoid questions, list-based and definition-based questions (cf. Figueroa and Neumann, 2006, 2007, 2009). The core idea was to use the N-best snippets returned by the search engine Bing[4] as answer source pool, and to apply clustering and extraction methods based on Latent Semantic Analysis (LSA) combined with pattern matching to identify and extract exact answer strings.

Thus, it was a natural first step to exploit WebQA also for answering YA questions such that in a first step a question classification is performed in order to select only those questions, which are covered by WebQA and then to call WebQA to find exact answers. This approach failed for several reasons: 1.) Only a small fraction of the YA questions are basically WebQA-suitable simple factoid questions (e.g., only 22 questions out of 910 questions from the 1k-qids.txt corpus provided by the track organizers), 2.) The found answer strings (whether correct or wrong) were actually too small compared to the answers in the YA development corpus, and finally, 3.) The runtime of WebQA for list-based and definition-based questions was actually too high so that we failed the speed limit of 60 seconds. However, WebQA's pipeline that consists of a number of subcomponents (from snippet identification, sentence extraction, sentence re-ranking, phrase extraction, answer validation etc.) has a high degree of modularity so that we decided to use relevant subcomponents of WebQA for building and testing our **LiveWebQA** system.

2. System Overview

Following our previous work, we assume that the Internet as a whole can serve as reliable answer source for basically any question. However, in contrast to our previous work, we do not assume that the snippets returned by WebQA are sufficient for serving as answer sources but that we have to inspect the whole web documents the snippets point to. The core idea of our approach is now to automatically learn a question-answer mapping from a subset of an available YA-archive (actually we used the ydata-yanswers-all-questions-v1_0 corpus that was made available to the LiveQA participants by the organizers; see also sec. 2.2), and to apply this mapping to YA questions and the whole Web documents identified by WebQA via its snippet identification and ranking step.

Thus, there are 3 main components of our system (called **LiveWebQA**), which dealt with the different aspects of answering a YA question:

1.) Analyzing Questions
2.) Training
3.) Search and score

The major workflow is as follows (cf. also Figure 1):

0. Train a Q-A map in form of a vector space from a training corpus
1. Send the Yahoo question to the 'Retrieve Answers' module of WebQA
2. Send the title and body to the 'Analyze Question' module
3. Send the 'key' question to the 'Retrieve Answers' module

[4] http://datamarket.azure.com/dataset/bing/search

4. Send the 'key' question to the 'From URLs' module
5. Send the web URLs to the 'Search and Score' module
6. Then, identify the best 3 sentences from the searched content
7. Pass the answer to the 'Retrieve Answers' module

Finally, the found answer string is sent to the LiveQA client of the LiveQA track organizers in the required format.

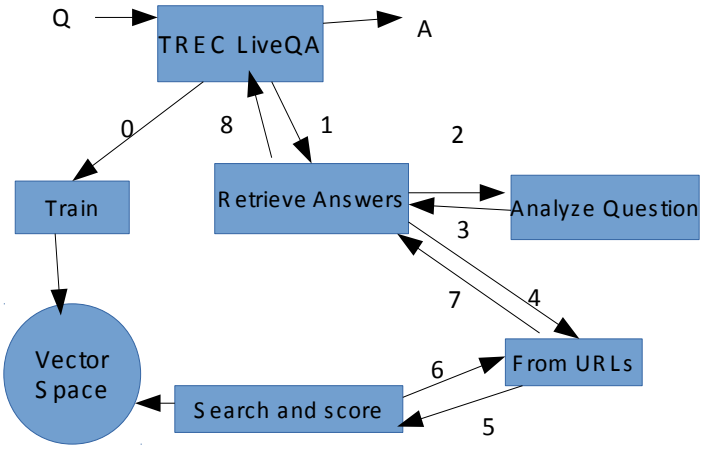

Figure 1: Architecture and main information flow of LiveWebQA

We will now describe the three main components in more details.

2.1 Analyzing questions

To answer a YA question is in general more difficult compared to the answering of a factoid question. For example, in general, the essence of a YA question has to be mined from a text of variable length, since a YA question in general does not exist of single WH-question as, for example, has been the case in the previous TREC and CLEF QA challenges, but a paragraph that contains the main question terms more or less explicitly.

More precisely, a YA question usually consists of a title and a body. Ideally, the title contains the main question. However, often this is not the case, because the relevant question information is distributed across the title and the body. This means, one has to actually identify and extract these question parts. Automatic text summarization is a field of research on its own right, which could be exploited also here. But since the text of a YA question is usually not too large (and because of time constraints during the development phase), we employed a heuristic to just identify the important question fragments based on the following intuitions: if a sentence in the question paragraph (which consists of the question title and body) contains a pronoun (except for first person pronouns) or a question key-word (like ‘who’, ‘what’, ‘how’, ‘where’, ‘why’, ‘?’), then most likely the previous sentence (if it exists) and the sentence itself together have useful information and should not be discarded. In this way, we do not filter out sentences preceding a trivially important sentence (identified by 'help', '?'). Thus, from a new YA question Q, we construct a question Q’ such that Q’

consists of all the direct questions and trivially important sentences of Q and the identified important sentences based on the above criteria preceding these direct questions. Then Q' is used to search the Web using the Bing API.

For example, for the YA question Q with qid=20150713141918AA2OxXt:

title: GearHead mouse button too sensitive?

body: I've been using a GearHead Optical 2.4 GHz Wireless Nano mouse for about a year and have had no problems up until very recently. The right mouse button is way too sensitive. When I open it up the button on the left still has its click but just resting a finger on the right button makes it rigt click. Is there a way to fix this or should I just buy a new mouse?

If I should buy a new one, are there any that you would recommend for gaming?

We compute the following reduced question Q':

GearHead mouse button too sensitive?
The right mouse button is way too sensitive. When I open it up the button on the left still has its click but just resting a finger on the right button makes it rigt click. Is there a way to fix this or should I just buy a new mouse? If I should buy a new one, are there any that you would recommend for gaming?

Here, the underlined sentences are the trivially important sentences and the lines captured preceding those are the identified important sentences. We stop to add more lines if there are no more pronouns in the last sentence that we add.

Although the YA questions to be answered in the LiveQA track 2015 are restricted to a subset of the possible set of YA categories (i.e., only questions are considered from the topics Arts & Humanities, Beauty & Style, Computers & Internet, Health, Home & Garden, Pets, Sports, Travel), we do not make use of them in the question analysis module nor in other modules of our current **LiveWebQA** system.

2.2 Training

The core idea of the training phase is to learn a mapping from YA question patterns to YA answer patterns using an available YA question/answering corpus as basis. In some sense, the learned mapping would express useful indicators for deciding whether a YA question belongs to a certain question type and whether a sentence found in a Web page could serve as answer for this YA question type. Thus, the goal is to learn to guess whether a sentence embeds useful "YA answer sentence typical information" for specific embedded "YA question typical information".

Paragraph embedding is a growing field of research. The method proposed by Mikolov et al. (2013) has proven to get state-of-art results for text similarity tasks. However, the task of QA is usually harder than simple text similarity as it involves the association of a question and answer pair. We had a corpus of questions and answers from YA that was provided to all participants by the organizers of the LiveQA challenge. The corpus (called ydata-yanswers-all-questions-v1_0) contains 4,483,032 questions and their answers from which we used only a small subset of 42K QA pairs (mainly because of development time constraints). In addition of question and answer text, the corpus contains a small amount of meta data, i.e., which answer was selected as the best answer,

and the category and sub-category that was assigned to this question. For more information about form and content, cf. (Surdeanu et al., 2008).
We suspected obtaining question vectors and answer vectors from this corpus during the training phase and associating them may lead to over-fitting using paragraph vectors. Hence, we choose to follow a simpler method to get vectors. Our intuition is that the questions have patterns that can be used to identify the question type and answers associated also have patterns, which are specific to a particular question type. To obtain these generic patterns of questions and answers, we removed the key content words from the questions and answers. We represented a question and an answer in terms of the skipped (by removing rare words) tri-grams that occurred in them. Hence, each question and each answer is a vector whose dimension is equal to the total number of tri-grams that occurred in the corpus. The vectors are then weighted by computing a tfidf-like weight for each cell in the following way:

Value in each dimension = tri-gram_term_frequency * inverse_document_frequency,

where tri-gram_term_frequency is the number of times the tri-gram occurred in the relevant category (question or answer), and inverse_document_frequency is the log of the total number of question answer pairs divided by the number of times tri-gram occurred in the relevant category of question or answer.

We ignored the tri-grams that contained just the function words by putting a frequency threshold, as the function word tri-grams don't give any information specific to the question. For a question-answer corpus, we aimed to train two separate vector spaces one for the questions and one for the answers. These two vector spaces are mapped to each other by the co-occurrences of the question and answer pairs in the corpus.

2.3 Search and Score:

Given a new question, we find the relevant vector in our question-vector space and retrieve the k-nearest neighbors that we have seen in the training corpus, and from this, we determine the associated answer vectors in the answer-vector space. We then predict the best answer-vector as the average of the sum of answer-vectors associated with these k-nearest neighbors. We took the value of k as 50 for a corpus of the 42k Yahoo question and answer pairs.

After having selected the answer vector for the current question, we look up the web pages given by the N-best snippets found by WebQA. For each web page, we identify each sentence s and score it according to the following two measures:

i.) pattern_quality_measure(s): Compute the cosine similarity between the sentence and predicted answer vector.
ii.) relevance_measure(s): Since we are looking through the whole web page, we need a second measure to pick only sentences that are relevant to the question. For this reason, we use the word2vec tool (cf. Mikolov et al., 2013) to compute the cosine similarity between the question and selected sentence. The motivation behind using word embedded vectors here is to weaken the surface-based restriction a direct comparison of the tri-grams between the question and a sentence would have.

By combining both measures, we compute for each sentence s a score as follows:
if the relevance_measure(s) < 0.8 then

score(s) = pattern_quality_measure(s) * relevance_measure(s)

else

score(s) = 0.8 * pattern_quality_measure(s) * relevance_measure (s)

The motivation behind cutting-off the score for sentences with a relevance measure >= 0.8 is that often these sentences would be questions similar to the asked question itself.

We then retrieve the three consecutive sentences from the web pages, which have the highest sum of sentence scores. Such a three-sentence long paragraph is then considered as answer for the current YA question. In order to improve speed, we searched in each web page separately by creating a thread per web page, and send the best answer retrieved so far. Unfortunately, not all web pages were processed within the time limit of 60 seconds because of their size, and hence, the overall quality could be negatively affected.

3. Experiments

We tried with different parameters for thresholds to remove the content words in the training process for patterns. We finally set 300 as frequency threshold for the corpus of the 20 million tokens of our 42K training corpus. We tried using a much larger corpus, but the system didn't perform significantly better when we increased the size.

To get an idea of the potential quality of the results of our approach, we manually verified the answers provided for a test set of the 1000 questions derived from the test corpus 1k-qids.txt during the development of our **LiveWebQA** system. We found that 20% of the times the system gave a useful answer. We considered a predicted answer as useful if parts of it can serve as "answer" to the asked question.

We also automatically evaluated the results by comparing the answers predicted by our **LiveWebQA** system with the answers in the test Q&A corpus using the word2vec (bag of words) cosine similarity. We got 45% of the questions answered with greater than 0.7 cosine similarity measure. Note that this does not automatically mean, that a 0.7 similarity also means that the predicted answer has high accuracy, but only gives an indication of its relatedness on basis of the selected word embedding. The main motivations for using word2vec for our automatic evaluation were twofold: 1) Verifying whether two texts convey the same meaning is a sub-problem to Question-Answering itself. Thus, text similarity between two answers will give us a measure on the quality of the answer even though it doesn't tell us that both have same meaning. 2) We choose word2vec bag-of-models to compare the texts, as it is was easy to integrate also considering the time constraints we had.

4. Results

During the official run, overall, 1087 questions were judged by the TREC organizers and scored using 4-level scale:

4: Excellent - a significant amount of useful information, fully answers the question
3: Good - partially answers the question
2: Fair - marginally useful information
1: Bad - contains no useful information for the question
-2: the answer is unreadable (only 15 answers from all runs were judged as unreadable)

Since we had some delay in starting our server, actually only 1058 questions were processed by our system. 893 of them are scored as 1 (bad); 112 as 2 (fair); 42 as 3 (good); 11 as 4 (excellent).

The score of our system on the official run is given below:

avg score (0-3)	succ@1+	succ@2+	succ@3+	succ@4+	prec@2+	prec@3+	prec@4+
0.211	0.973	0.152	0.049	0.010	0.156	0.050	0.010

If we compare these results with our expectations (about 20% useful answers), we are actually far behind them, which might indicate that our manual analysis was not sufficient or not in line with the official scale.

Below is the official average of scores of all runs:

avg score (0-3)	succ@1+	succ@2+	succ@3+	succ@4+	prec@2+	prec@3+	prec@4+
0.465	0.925	0.262	0.146	0.060	0.284	0,159	0.065

Compared to our results, it means that in average the majority of questions are badly answered, but that the average performance of answering questions as good or excellent is higher compared to our result. It is clear that the overall performance in terms of precision of our system is still poor. However, we are now in a position to dive into the problems of our system using the evaluation results from this initial LiveQA track challenge, so that we are convinced to get a much better result next time.

References

Bogdan Sacaleanu, Günter Neumann and Christian Spurk: DFKI at QA@CLEF 2008. Working Notes for the CLEF 2008 Workshop, 17-19 September, Aarhus, Denmark. (2008)

Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. In NIPS Deep Learning Workshop (2014)

Alejandro Figueroa and Günter Neumann: Language Independent Answer Prediction from the Web. FinTAL 2006: 423-434 (2006)

Alejandro Figueroa and Günter Neumann: Mining Web Snippets to Answer List Questions. AIDM 2007: 61-71 (2007)

Alejandro Figueroa, Günter Neumann, John Atkinson: Searching for Definitional Answers on the Web Using Surface Patterns. IEEE Computer 42(4): 68-76 (2009)

Alejandro Figueroa and Günter Neumann: Category-specific models for ranking effective paraphrases in community Question Answering. Expert Syst. Appl. 41(10): 4730-4742 (2014)

Tomas Mikolov , Kai Chen, Greg Corrado, and Jeffrey Dean: Efficient estimation of word representations in vector space , CoRR, abs/1301.3781 (2013)

Quoc V. Le and Tomas Mikolov: Distributed representations of sentences and documents. ICML (2014)

Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza: Learning to Rank Answers on Large Online QA Collections. Proc. of the 46th Annual Meeting of the Association of Computational Linguistics (ACL) (2008)

Yikang Shen, Wenge Rong, Zhiwei Sun, Yuanxin Ouyang, Zhang Xiong: Question/Answer Matching for CQA System via Combining Lexical and Sequential Information. AAAI 2015: 275-281 (2015)