Emory University at TREC LiveQA 2016: Combining Crowdsourcing and Learning-To-Rank Approaches for Real-Time Complex Question Answering

Denis Savenkov
Emory University
dsavenk@emory.edu

Eugene Agichtein
Emory University
eugene@mathcs.emory.edu

Abstract

This paper describes the two QA systems we developed to participate in the TREC LiveQA 2016 shared task. The first run represents an improvement of our fully automatic real-time QA system from LiveQA 2015, *Emory-QA*. The second run, *Emory-CRQA*, which stands for Crowd-powered Real-time Question Answering, incorporates human feedback, in real-time, to improve answer candidate generation and ranking. The base *Emory-QA* system uses the title and the body of a question to query Yahoo! Answers, Answers.com, WikiHow and general web search and retrieve a set of candidate answers along with their topics and contexts. This information is used to represent each candidate by a set of features, rank them with a trained LambdaMART model, and return the top ranked candidates as an answer to the question. The second run, *Emory-CRQA*, integrates a crowdsourcing module, which provides the system with additional answer candidates and quality ratings, obtained in near real-time (under one minute) from a crowd of workers When *Emory-CRQA* receives a question, it is forwarded to the crowd, who can start working on the answer in parallel with the automatic pipeline. When the automatic pipeline is done generating and ranking candidates, a subset of them is immediately sent to the same workers who have been working on answering the questions. Workers than rate the quality of all human- or system-generated candidate answers. The resulting ratings as well as original system scores are used as features for the final re-ranking module, which returns the highest scoring answer. The official run results of the tasks indicate promising improvements for both runs compared to the best performing system from LiveQA 2015. Additionally, they demonstrate the effectiveness of the introduced crowdsourcing module, which allowed us to achieve an improvement of ∼20% in average answer score over a fully automatic *Emory-QA* system.

1 Introduction

The results of the previous iteration of TREC LiveQA shared task demonstrated that there is still significant room for improvement for automatic QA systems to be able to satisfy various user information needs. Even the winning system was able to return a fair or better answer to only half of the questions, which means that the other half of the users would get no benefit from system responses [1]. There are a number of directions to try to bridge this gap, and improve the core algorithms inside question answering systems, such as data sources, candidate extraction, ranking and answer generation. The analysis of answer ratings revealed that automatic QA systems often return a response that is covering a different topic and is non-relevant to the user question [2]. We, as humans, on the other hand have no problems understanding such questions, and can usually make relevant suggestions or recommendations even if we do not know the right answer to the question. For TREC LiveQA 2016 we tried to approach the problem of improving question answering performance of our system by both improving the technical aspects and exploring if it is possible to put human workers in the loop and leverage the power of crowdsourcing for real-time question answering. As a basis we took the architecture of Emory QA system, built for the last year shared tasks [2].

The base automatic system, *Emory-QA* was improved in a number of ways:

- Extended the set of data sources, which now includes Yahoo! Answers, Answers.com, WikiHow and general web search.

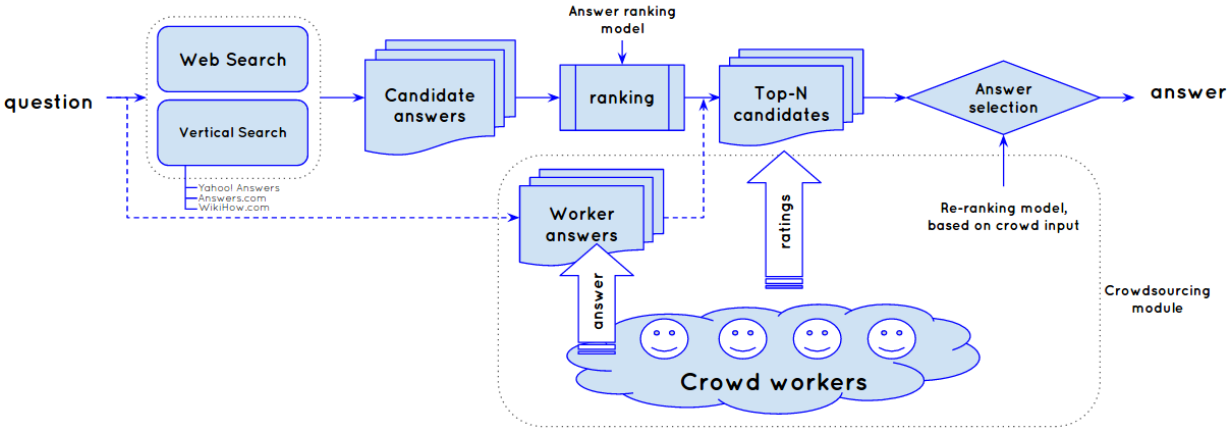

Figure 1: Architecture of our question answering system

- Improved passages representation by including the topic and context from original web pages.
- Extended a set of features to match question against candidate answer text, topic and context.
- Switched to a non-linear ranking model (LambdaMART), trained on the qrels available from LiveQA 2015.

Additionally, we introduced a new system, *Emory-CRQA*, which aims to target the problems of lack of good answer candidates and ranking errors by incorporating crowdsourcing into the ranking process. For each question, we asked workers to provide their own answer if possible, and in addition to rate a subset of candidates generated by the system or other workers. In our preliminary study, the crowd was able to provide reliable input for both types of feedback under a one minute time-limit [3]. The following sections describe the two Emory QA systems in more detail.

2 Approach

The architecture of our question answering system is presented on Figure 1. For the official TREC LiveQA 2016 run we submitted two variations of our question answering system: with and without the crowdsourcing module. Similarly to the previous year our fully automatic system, *Emory-QA*, uses a set of vertical and general web search components to generate a set of candidate answers, which are represented by a set of features and ranked by a trained model, and the top scoring candidate is returned as the answer to the question. While this approach works reasonably good, there are still a big number of questions, where no appropriate candidates are generated or the trained model was not able to rank them appropriately. Therefore, we decided to employ crowdsourcing and test whether it is possible to improve performance of near real-time question answering system using worker contributions.

2.1 Candidate generation

Each question issued to a QA system in TREC LiveQA consists of 3 main parts: title, body and category (Figure 2). The analysis of our system performance from last year TREC LiveQA shared task [2] demonstrated the effectiveness of reusing answers from similar questions, previously posted to CQA websites. Therefore, we decided to extend a set of vertical platforms and included Answers.com[1] and WikiHow[2]. To generate candidate answers from these platforms we first formulate one or more queries to the corresponding platforms search interfaces. The questions posted by users vary from short and concise to very

[1]http://www.answers.com/
[2]http://www.wikihow.com/

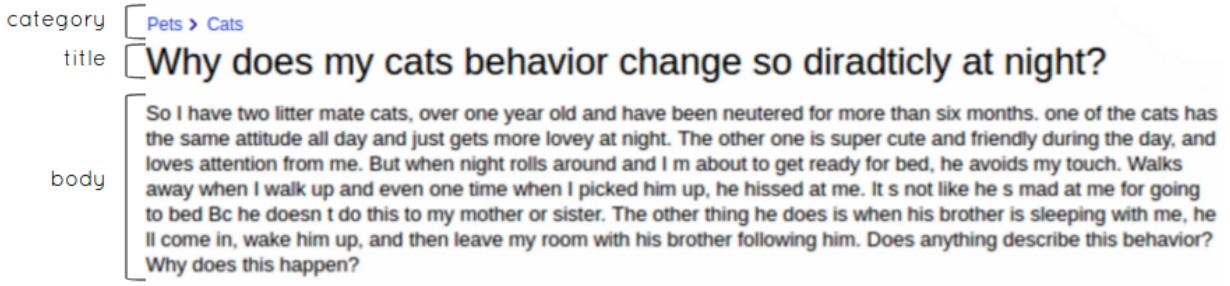

Figure 2: Example of a question from Yahoo! Answers community question answering platform

verbose. Therefore, we use multiple query generation strategies, designed to increase the chances to find a good match:

- Question title, which most often captures the gist of the question
- Two longest question sentences (detected by the presence of the question word at the beginning or question mark at the end of a sentence) from the title and body of the question. In some cases the real user question is hidden inside the body, while the title provides only the overall topic of the question.
- Concatenation of the question word, verbs and top-5 terms from the question title by inverse document frequency[3]. This strategy targets over-specific questions (Figure 2), which often retrieve few if any search results.

Each of the generated queries is issued to a CQA platform search interface and top-10 retrieved questions along with the corresponding answers are added to the candidate answers pool.

Many user information needs are still unique or have unique details, which makes it hard to find a similar record in a CQA archive. Therefore, as in the previous year, we use standard web search[4]. The above-mentioned strategies are used once again to generate search queries, that are issued to a web search API to retrieve top-10 potentially relevant documents. To extract the content blocks of the web page our system uses a method similar to [4]. Such blocks of web page content are added to the candidate answers pool along with certain meta-information.

The meta-information we extract for CQA- and web-based candidates helps the ranking module to score candidate answers relevance. For regular web page paragraphs, it is useful to know the topic of the page (*e.g.,* its title) and the context (*e.g.,* text that immediately precedes the paragraph in the document). For CQA answers, our system stores the text of the corresponding question title, body and category. For convenience, we will refer to this question title and web page title as *"answer topic"*, while the body of the retrieved question and the preceding text block for web candidates as *"answer context"*.

2.2 Candidate ranking

To estimate the relevance of each answer candidate, we represent them with a set of features (Table 1). On the highest level, we had three groups of features: answer text statistics, candidate source and textual match features. In the first group, we computed the lengths of candidates in characters, words and sentences; average length of words; number of non-alphanumeric characters; number of question marks and number of verbs. These features are designed to capture the readability of the question, as well as its informativeness in general (*e.g.,* a candidate with many question marks is likely to just state a question). The answer source features are binary features, that can help the model to distinguish passages extracted from Yahoo! Answers, Answers.com, WikiHow or regular web search, and possibly learn to prefer one type to the other in certain circumstances. Finally, the match-based feature group is designed to capture semantic similarity between the question and a candidate answer. We have multiple pieces of text on both question and answer sides,

[3]IDF of terms are estimated using Google N-gram corpus: https://catalog.ldc.upenn.edu/LDC2006T13
[4]https://www.microsoft.com/cognitive-services/en-us/bing-web-search-api

e.g., title and body of the question; text, topic and context of an answer. All our match-based features are computed for different combinations of these texts. This solution is inspired by the clues-based approach of CMU OAQA system from TREC LiveQA 2015 [5], and confirmed with analysis of the results from last year. Fragments with high textual similarity to the question often simply restate it and do not provide any useful information, which can often be found in the following passage. Our text matching features can be split into two subgroups: n-gram overlap and IR score features. Particularly, we compute uni-, bi- and trigram based cosine similarity, as well as the longest span of matching terms for text fragments. As IR metric we chose to use BM25 score, which was demonstrated to be a strong baseline in previous research [6].

Answer statistics
— Length in chars, words and sentences — Average number of words per sentence — Fraction of non-alphanumeric characters — Number of question marks — Number of verbs
Answer source
— Binary feature for each of the search verticals: Web, Yahoo! Answers, Answers.com, WikiHow.com
N-gram matches
— Cosine similarities using uni-, bi- and tri-gram representations of the question title and/or body, and answer text, topic or context — The lengths of longest spans of matched terms between question title and/or body, and answer text, topic or context
Information Retrieval score
— BM25 scores between question title and/or body, and answer text, topic or context

Table 1: The list of candidate answer ranking features used by the automatic module of our CRQA system

The next step is to rank candidate answers, and for that we used a LambdaMART model [7]. In a fully automated scenario the top ranked candidate was returned as the final answer to the given question.

2.3 Model Training

To train our ranking LambdaMART model we used the labeled data from TREC LiveQA 2015 [5]. This dataset contains 1087 questions along with labeled responses submitted by automatic system during the shared task last year. Each answer was rated by professional NIST assessors on a scale from 1 (bad) to 4 (excellent). Most of the rated answers also include source URLs, which we used to download the original web pages and extract topic and context for the candidate. We trained LambdaMART model to optimize the NDCG metric using RankLib library[6].

2.4 *Emory-CRQA*: Crowdsourcing module

TREC LiveQA 2015 demonstrated that automatic systems still struggle to answer many of real user questions, *e.g.,* the winning system was able to return a fair or better answer to roughly half of the questions, and only ∼18% of the answers were excellent. Complimentary to improving the technical side of the systems, we decided to explore crowdsourcing as one of the ways to help the system to deal with these questions. Preliminary analysis [3] showed that under a limited time crowd workers can provide reliable answer quality ratings and in some cases even write answers to the questions. Therefore, we integrated the crowdsourcing module into our QA system. *Emory-CRQA* utilizes a crowd of workers to generate additional and rate existing answer candidates, while still operating under a one minute time limit.

To handle the real-time aspect of the task, *Emory-CRQA* starts a timer after it receives a question and waits to get all worker contributions for 50 seconds to leave itself 10 seconds to prepare and send back the

[5]https://sites.google.com/site/trecliveqa2016/liveqa-qrels-2015
[6]https://sourceforge.net/p/lemur/wiki/RankLib/

final response. The pipeline of *Emory-CRQA* is pictured on Figure 1 and Figure 3 presents the user interface of our crowdsourcing module.

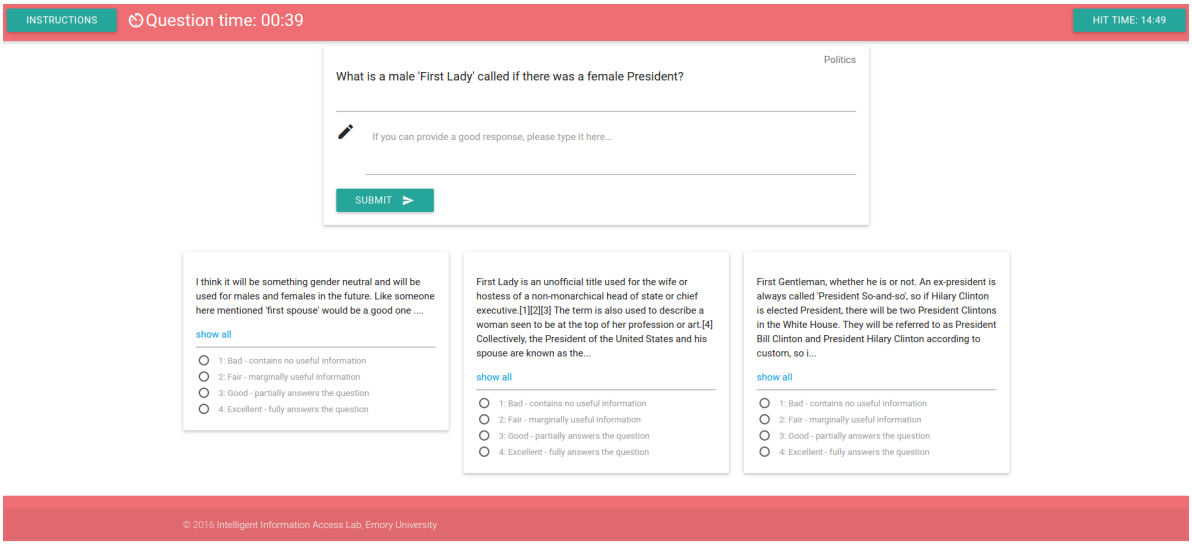

Figure 3: User Interface for workers in our Crowd-Powered Question Answering system

The overall algorithm for obtaining crowdsourcing input is:

1. When a system receives a question, it is posted to workers, who have 50 seconds to provide their input
2. Workers are asked to write an answer if they can provide one (optional)
3. Otherwise they need to wait for answer candidates to appear
4. When a system is done generating and ranking candidates, it posts top-7 answers[7] for rating (which usually happens $\sim$ 15 seconds after the question is posted)
5. Workers receive a list of answers and rate them until the timer expires. Answers, provided by the workers, are also rated by other workers. Each answer is rated on a scale from 1 to 4, using the official TREC LiveQA rating scale:
   - 1 – Bad: contains no useful information
   - 2 – Fair: marginally useful information
   - 3 – Good: partially answers the question
   - 4 – Excellent: fully answers the question
6. The worker interface displays 3 answers at a time, and when an answer gets rated, it disappears and its place is taken by another answer from the pool. The interface displays only the first 300 characters of the answer, which was experimentally shown to be enough on average to make a good judgment. Full answer can be revealed upon clicking the "show all" link.
7. When the question expires, it disappears, and workers wait for the next question

The workers were hired on Amazon Mechanical Turk[8]. Since the latency of hiring new workers for the task can be high, we adapted the retainer model [8, 9], *i.e.,* workers were paid by time, during which they had to stay on our interface and complete the tasks for 15 minutes. After 15 minutes expire, the task is submitted and the worker got paid. During 24 hours of the official shared task run we posted 10 tasks every

[7]The number is chosen based on the average number of answers workers could rate under 1 minute in our preliminary analysis [3]

[8]http://mturk.com

15 minutes, with the goal to have around 10 workers for each question. However, not all tasks are accepted immediately, therefore the actual number of workers per question fluctuates and can be greater than 10. When a worker first gets to our crowdsourcing interface, she is shown task instructions (Table 2) and asked to wait for the questions to arrive. The workers were paid $1.00 for a 15 minutes task, no matter how many questions they received. Each 15 minutes we had 10 assignments for different workers, which translates to $15.00 per 15 minutes. Since not all assignments were accepted, overall, our experiment cost *$0.88 per question*, and we show some ideas to decrease these costs in our HCOMP paper [11].

We should note, that the setup of TREC LiveQA shared task was favorable for the retainer crowdsourcing model, as the questions were arriving almost every minute uniformly over 24 hour period, which diminishes the waiting and worker idleness problems [10].

Instructions
1. This HIT will last exactly 15 minutes
2. Your HIT will only be submitted after these 15 min.
3. In this period of time you will receive some questions, that came from real users on the Internet
4. Each question has a time limit after which it will disappear and you will need to want for the next one
5. If you know the answer to the question, please type it in the corresponding box
6. At some point several candidate answers will appear at the bottom of the page
7. Please rate them from 1 (bad) to 4 (excellent)
8. Do not close the browser or reload the page as this will reset your assignment.

Table 2: Crowdsourcing task instructions, displayed to the user when she first gets to the task

2.5 *Emory-CRQA*: Answer re-ranking and selection

The last stage in *Emory-CRQA* is answer re-ranking, which aggregates all the information received from the crowdsourcing and produces the final answer to the question. The input of the re-ranking module is a set of candidate answers with quality ratings provided by the crowd workers. During the TREC LiveQA 2016 run we used a simple heuristic model, which ordered the answers by the average rating, and returned either the top candidate, if its average score was ≥ 2.5, or the longest worker contributed answer. This heuristic was inspired by the observation that user contributed answers are usually relevant to the question and are better preferred to automatically generated answer, which covers completely different topic. However, some workers add "noise" to the candidate pool by submitting short and non-informative answers like "yes", "no" and "I don't know", *etc.* After the shared task was over, we collected all the questions, candidate answers and all crowdsourcing contributions and conducted a series of experiments with more sophisticated re-ranking techniques. The details as well as additional analysis can be found in our HCOMP paper [11].

3 Evaluation

From the final 24 hour run of the systems, responses to 1015 questions were judged by the organizers on a scale from 1 to 4:
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)
Similar to the previous year, the reported metrics were:

- **avg-score(0-3)**: average score over all questions, where scores are translated to 0-3 range. This metric considers "Bad", unreadable answers and unanswered questions as having score 0
- **succ@i+**: the fraction of questions with the answer score of i or greater (i=1..4)
- **p@i+**: the fraction of provided answers with score i or greater (i=2..4)

	# ans	avg score (0-3)	succ@2+	succ@3+	succ@4+	p@2+	p@3+	p@4+
Results from TREC LiveQA 2016								
Emory-QA	995	1.0542	0.5192	0.3547	0.1803	0.5296	0.3618	0.1839
Emory-CRQA	976	1.2601 (+19.5%)	0.6197 (+19.2%)	0.4207 (+18.6%)	0.2197 (+21.8%)	0.6445 (+21.7%)	0.4375 (+20.9%)	0.2285 (+24.2%)
Average results	771	0.5766	0.3042	0.1898	0.0856	0.3919	0.2429	0.1080
Results from TREC LiveQA 2015								
CMU OAQA'15	1064	1.081	0.532	0.359	0.190	0.543	0.367	0.179
Emory-QA'15	884	0.608	0.332	0.190	0.086	0.408	0.233	0.106

Table 3: Results of the TREC LiveQA 2016 evaluation of Emory University QA systems and average results of all systems. For convenience we also provide our and best overall results from LiveQA 2015 shared task, but since the data is different the results between years are not directly comparable. Percents show the improvements of crowdsourcing over fully automatic Emory-QA system

Table 3 provides the results of our fully automatic and CRQA systems as well as average performance over all submitted systems.

One significant improvement over the last year was made towards system reliability, which resulted in higher number of answers, and therefore higher average score. The quality of the answers are not directly comparable between the years as the data is different, however, higher numbers for all the metrics adds some optimism. Crowdsourcing module turned out to be quite effective and allowed to boost the system average answer score by $\sim 20\%$ from 1.0542 to 1.2601. We will add more detailed analysis of the system improvements in the final version of the report.

4 Conclusions

During TREC LiveQA 2016 we submitted two runs with fully automated system and a system that used crowdsourcing to collect and rate candidate answers. The results of the shared task clearly demonstrated the effectiveness of crowdsourcing even in near real-time scenario, which boosted the answer scores by $\sim$20% on average. This initial success opens up a lot of opportunities to explore different kind of inputs and feedback from the crowd workers to improve the performance of a general QA system. In the future research there are a number of problems that needs to be addressed, *i.e.,* how to optimize the costs associated with the crowdsourcing module [11], how to use crowdsourcing feedback more efficiently, *e.g.,* make the system learn from crowd feedback over time and what kind of feedback from the workers is more efficient, *etc.* One can imagine the system that automatically decides when and what type of information to request from the workers (or from the user herself in a chat scenario), based on the predictions about different component performances, and then adjust its internal models to learn from the feedback provided.

References

[1] Eugene Agichten, David Carmel, Donna Harman, Dan Pelleg, and Yuval Pinter. Overview of the trec 2015 liveqa track. In *Proceedings of Text Retrieval Conference*, 2015.

[2] Denis Savenkov. Ranking answers and web passages for non-factoid question answering: Emory university at trec liveqa. 2015.

[3] Denis Savenkov, Scott Weitzner, and Eugene Agichtein. Crowdsourcing for (almost) real-time question answering: Preliminary results. In *2016 NAACL Workshop on Human-Computer Question Answering*, 2016.

[4] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM.

[5] Di Wang and Eric Nyberg. Cmu oaqa at trec 2015 liveqa: Discovering the right answer with clues. In *TREC*, 2015.

[6] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383, 2011.

[7] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11:23–581, 2010.

[8] Michael S Bernstein, Joel Brandt, Robert C Miller, and David R Karger. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 33–42. ACM, 2011.

[9] Jeffrey P Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, et al. Vizwiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 333–342, 2010.

[10] Walter S. Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F. Allen, and Jeffrey P. Bigham. Chorus: A crowd-powered conversational assistant. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 151–162, 2013.

[11] Denis Savenkov and Eugene Agichtein. Crqa: Crowd-powered real-time automatic question answering system. 2016.