Generating High-Quality Query Suggestion Candidates for Task-Based Search

Heng Ding[1,2](✉), Shuo Zhang[2], Darío Garigliotti[2], and Krisztian Balog[2]

[1] Wuhan University, Wuhan, China
hengding@whu.edu.cn
[2] University of Stavanger, Stavanger, Norway
{shuo.zhang,dario.garigliotti,krisztian.balog}@uis.no

Abstract. We address the task of generating query suggestions for task-based search. The current state of the art relies heavily on suggestions provided by a major search engine. In this paper, we solve the task without reliance on search engines. Specifically, we focus on the first step of a two-stage pipeline approach, which is dedicated to the generation of query suggestion candidates. We present three methods for generating candidate suggestions and apply them on multiple information sources. Using a purpose-built test collection, we find that these methods are able to generate high-quality suggestion candidates.

1 Introduction

Query suggestions, recommending a list of relevant queries to an initial user input, are an integral part of modern search engines [8]. Accordingly, this task has received considerable attention over the last decade [1,2,7]. Traditional approaches, however, do not consider the larger underlying task the user is trying to accomplish. In this paper, we focus on generating query suggestions for supporting task-based search. Specifically, we follow the problem definition of the *task understanding* task from the TREC Tasks track: given an initial query, the system should return a ranked list of suggestions "that represent the set of all tasks a user who submitted the query may be looking for" [14]. Thus, the overall goal is to provide a complete coverage of aspects (subtasks) for an initial query, while avoiding redundancy.

We envisage a user interface where task-based query suggestions are presented once the user has issued an initial query; see Fig. 1. These query suggestions come in two flavors: *query completions* and *query refinements*. The difference is that the former are prefixed by the initial query, while the latter are not. It is an open question whether a unified method can produce suggestions in both flavors, or rather specialized models are required. The best published work on task-based query suggestions, that we know of, is by Garigliotti and Balog [4], who use a probabilistic generative model to combine keyphrase-based suggestions extracted from multiple information sources. Nevertheless, they rely

© Springer International Publishing AG, part of Springer Nature 2018

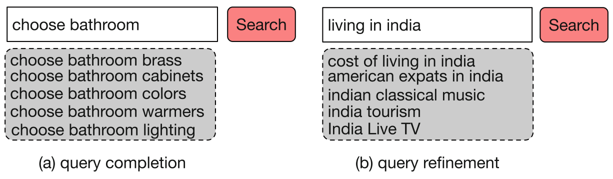

Fig. 1. Examples of query suggestions to support task-based search.

heavily on Google's query suggestion service. Thus, another main challenge in our work is to solve this task without relying on suggestions provided by a major web search engine (and possibly even without using a query log).

Following the pipeline architecture widely adopted in general query suggestion systems [7,12], we propose a two-step approach consisting of *suggestion generation* and *suggestion ranking* steps. In this paper, we focus exclusively on the first component. Our aim is to generate sufficiently many high-quality query suggestion candidates. The subsequent ranking step will then produce the final ordering of suggestions by reranking these candidates (and ensuring their diversity with respect to the possible subtasks). The first research question we address is: *Can existing query suggestion methods generate high-quality query suggestions for task-based search?* Specifically, we employ the popular suffix [7], neural language model [10], and sequence-to-sequence [12] approaches to generate candidate suggestions. The second research question we ask is: *What are useful information sources for each method?* We are particularly interested in finding out how a task-oriented knowledge base (Know-How [9]) and a community question answering site (WikiAnswers [3]) fare against using a query log (AOL [11]). We find that the sequence-to-sequence approach performs best among the tested three methods. As for data sources, we observe that, as expected, the query log is the highest performing one among all. Nevertheless, the other two also provide valuable suggestion candidates that cannot be generated from the query log. Overall, we find that all method-source configurations contribute unique suggestions, and thus it is beneficial to combine them.

2 Query Suggestion Generation

Given a task-related initial query, q_0, we aim to generate a list of query suggestions that cover all the possible subtasks related to the task the user is trying to achieve. For a given suggestion q, let $P(q|q_0)$ denote the probability of that suggestion. Below, we present three methods from the literature. The first two methods are specialized only in producing query completions, while the third one is able to handle both query completions and refinements. Due to space constraints, only brief descriptions are given; we refer to the respective publications for further details.

2.1 Popular Suffix Model

The *popular suffix model* [7] generates suggestions using frequently observed suffixes mined from a query log. The method generates a suggestion q by extending the input query q_0 with a popular suffix s, i.e., $q = q_0 \oplus s$, where $\oplus$ denotes the concatenation operator. The query likelihood is based on the popularity of suffix s: $P(q|q_0) = \mathit{pop}(s)$, where $\mathit{pop}(s)$ denotes the relative frequency of s occurring in the query log.

2.2 Neural Language Model

Neural language models (NLMs) can be used for generating query suggestions [10]. Suggestion q is created by extending the input query q_0 character by character: $q = q_0 \oplus \mathbf{s} = (c_1, \ldots, c_n, c_{n+1}, \ldots, c_m)$, where $c_1, \ldots, c_n$ are the characters of q_0, and $\mathbf{s} = (c_{n+1}, \ldots, c_m)$ are characters generated by the neural model. Given a sequence of previous characters $\mathbf{c} = (c_1, \ldots, c_i)$, the model generates the next character $(i \geq n)$ according to: $P(c_{i+1}|\mathbf{c}) = \mathit{softmax}(h_i)$, where the hidden state vector at time i is computed using $h_i = f(x_i, h_{i-1})$. Here, f denotes a special unit, e.g., a long short-term memory (LSTM) [5]; x_i is the vector representation of the ith character of suggestion q and is taken to be $x_i = \sigma(c_i)$, where σ denotes a mapping function from a character to its vector representation. Finally, the query likelihood is estimated according to:

$$P(q|q_0) = \prod_{j=n}^{m-1} P(c_{j+1}|c_1, \ldots, c_j).$$

Our implementation uses a network of 512 hidden units. We initialize the word-embedded vector with the pre-trained vector from Bing queries.[1] Beam search width is set to 30.

2.3 Sequence to Sequence Model

The sequence-to-sequence model (Seq2Seq) [6] aims to directly model the conditional probability $P(w'_1, \ldots, w'_m|w_1, \ldots, w_n)$ of translating a source sequence $(w_1, \ldots, w_n)$ to a target sequence $(w'_1, \ldots, w'_m)$. Thus, it lends itself naturally to implement our query suggestion task using Seq2Seq, by letting the initial query be the source sequence $q_0 = (w_1, \ldots, w_n)$ and the suggestion be the target sequence $q = (w'_1, \ldots, w'_m)$. Typically, a Seq2Seq model consists of two main components: an encoder and a decoder.

The *encoder* is a recurrent neural network (RNN) to compute a context vector representation c for the original query q_0. The hidden state vector of the encoder RNN at time $i \in [1..n]$ is given by: $h_i = f(w_i, h_{i-1})$, where w_i is the ith word of the input query q_0, and f is a special unit (LSTM). The context vector

[1] https://www.microsoft.com/en-us/download/details.aspx?id=52597.

representation is updated by $c = \phi(h_1, h_2, \ldots, h_n)$, where ϕ is an operation choosing the last state h_n.

The *decoder* is another RNN to decompress the context vector c and output the suggestion, $q = (w'_1, \ldots, w'_m)$, through a conditional language model. The hidden state vector of the decoder RNN at time $i \in [1..m]$ is given by $h^{'}_i = f'(h^{'}_{i-1}, w^{'}_{i-1}, c)$, where $w^{'}_i$ is the ith word of the suggestion q, and f' is a special unit (LSTM). The language model is given by: $P(w^{'}_i | w^{'}_1, \ldots, w^{'}_{i-1}, q_0) = g(h^{'}_i, w^{'}_{i-1}, c)$, where g is a softmax classifier. Finally, the Seq2Seq model estimates the suggestion likelihood according to:

$$P(q|q_0) = \prod_{j=1}^{m-1} P(w'_{j+1}|w'_1, \ldots, w'_j, q_0) \ .$$

We use a bidirectional GRU unit with size 100 for encoder RNNs, and a GRU unit with size 200 for decoder RNNs. We employ an Adam optimizer with an initial learning rate of 10^{-4} and a dropout rate of 0.5. Beam search width is set to 100.

3 Data Sources

We consider three independent information sources. For the PopSuffix and NLM methods, we need a collection of short texts, $\mathcal{C}$. For Seq2Seq, we need pairs of question-suggestion pairs, $\langle \mathcal{Q}, \mathcal{S} \rangle$.

- *AOL query log* [11]: a large query log that includes queries along with anonymous user identity and timestamp. We extract all queries from the log as $\mathcal{C}$. We detect sessions (each session including multiple queries) using the same criterion as in [12]. Then, we pair queries in the same session to obtain $\langle \mathcal{Q}, \mathcal{S} \rangle$, where $\mathcal{Q}$ denotes a set of queries and $\mathcal{S}$ are suggestions paired against $\mathcal{Q}$. In order to obtain more pairs, we extract all proper prefixes from the query, and pair them together. For example, given a query "make a pancake", we can construct two pairs ⟨ "make", "make a pancake" ⟩ and ⟨ "make a", "make a pancake" ⟩. This way, we end up with a total of 112 K prefix-query pairs.
- *KnowHow* [9]: a knowledge base that consists of two and half million entries. Each triple $\langle s, p, o \rangle$ represents a fact about a human task, where the subject s denotes a task (e.g., "make a pancake") and the object o is a subtask (e.g., "prepare the mix"). We collect all subjects and objects as $\mathcal{C}$, and take all (142K) subject-object pairs to form $\langle \mathcal{Q}, \mathcal{S} \rangle$. Additionally, prefixes from tasks (i.e., subjects and objects) are extracted to get more pairs, the same way as it is done for the AOL query log.
- *WikiAnswers* [3]: a collection of questions scraped from WikiAnswers.com.[2] We detect task-related questions using a simple heuristic, namely, that a task-related question often starts with question constructions "how do you"

[2] http://www.answers.com/Q/.

or "how to." These question constructions are removed from the questions to obtain $\mathcal{C}$ (e.g., "how to change gmail password" $\rightarrow$ "change gmail password"). This source can only be used for the PopSuffix and NLM methods, as it does not provide pairs for Seq2Seq.

4 Experiment

We design and conduct an experiment to answers our research questions. First, we collect a pool of candidate suggestions, by applying the methods on different sources. Second, we collect annotations for each of these suggestions via crowdsourcing. Finally, we report and analyze the results.

4.1 Pool Construction

We consider all queries (100 in total) from the TREC 2015 and 2016 Tasks tracks [13,14]. We combine the proposed methods (Sect. 2) with various information sources (Sect. 3) for suggesting candidates. We shall write s-m to denote a particular configuration that uses method m with source s. In addition, we also include the suggestions generated by (i) the keyphrase-based query suggestion system [4], and (ii) the Google Query Suggestion Service (referred to as *Google API* for short. Two pools are constructed, one for query completions (QC) and one for query refinements (QR). The pool depth is 20, that is, we consider (up to) the top-20 suggestions for each method.

4.2 Crowdsourcing

Due to the fact that many of our candidate suggestions lack assessments in the TREC ground truth (and thus are considered irrelevant), we obtain relevance assessments for all suggestions via crowdsourcing. Specifically, we use the Crowdflower platform, where a dynamic number of annotators (3–5) are asked to label each suggestion as relevant or non-relevant. A suggestion is relevant if it targets for some more specific aspect of the original query, i.e., the suggestion must be related to the original query intent. It does not have to be perfectly correct grammatically, as long as the intent behind is clearly understandable. The final label is taken to be the majority vote among the assessors. A total of 12,790 QC and 9,608 QR suggestions are annotated, at the total expense of 692\$. Further details are available in the online appendix.[3]

4.3 Results and Analysis

Table 1 presents a comparison of methods in terms of precision at cutoff points 10 and 20 (P@10 and P@20). Overall, we find that our methods can generate high-quality query suggestions for task-based search. Our best numbers are

[3] http://bit.ly/2BnSjhR.

Table 1. Precision for candidate suggestions generated by different configurations. For QC methods, we also report on recall (R) and cumulative recall (CR).

Method	QC				QR	
	P@10	P@20	R	CR	P@10	P@20
AOL-PopSuffix	0.257	0.245	0.168	0.168	-	-
KnowHow-PopSuffix	0.195	0.170	0.102	0.256	-	-
WikiAnswers-PopSuffix	0.181	0.167	0.101	0.333	-	-
AOL-NLM	0.256	0.241	0.170	0.474	-	-
KnowHow-NLM	0.166	0.147	0.108	0.575	-	-
WikiAnswers-NLM	0.163	0.121	0.088	0.650	-	-
AOL-Seq2Seq	0.283	0.181	0.156	0.765	0.043	0.031
KnowHow-Seq2Seq	0.158	0.111	0.079	0.813	0.206	0.148
Keyphrase-based [4]	0.321	0.239	0.130	-	0.575	0.504
Google API	0.267	0.134	0.078	-	0.289	0.145

comparable to that of the Google API. It should, however, be noted that the Google API can only generate a limited number of suggestions for each query. The keyphrase-based method [4] is the highest performing of all; it is expected, as it combines multiple information sources, including suggestions from Google.

We find that the AOL query log is the best source for generating QC suggestions, while KnowHow works well for QR. AOL-Seq2Seq performs poorly on QR; this is because only 2% of all training instances in $\langle \mathcal{Q}, \mathcal{S} \rangle$ are QR pairs. For QC suggestions, the performance of AOL-PopSuffix and AOL-NLM are close to that of the Google API, while AOL-Seq2Seq even outperforms it. For QR suggestions, the performance of AOL-Seq2Seq is close to that of the Google API in terms of P@20, but is lower on P@10. In general, our system is able to produce more suggestions than what the (public) Google API provides.

Additionally, we also evaluate the recall of each QC method, using all relevant suggestions found as our recall base. We further report on cumulative recall, i.e., for line i of Table 1 it is the recall of methods from lines $1..i$ combined together. We observe that each configuration brings a considerable improvement to cumulative recall. This shows that they generate unique query suggestions. For example, given the query "choose bathroom", our system generates unique query suggestions from different methods and sources, e.g., "choose bathroom marks" (WikiAnswers-NLM), "choose bathroom supply" (AOL-NLM), "choose bathroom for your children" (KnowHow-NLM), "choose bathroom grout" and "choose bathroom appliances" (KnowHow-Seq2Seq), which are beyond what the Google API and the keyphrase-based system [4] provide. Therefore it is beneficial to combine suggestions from multiple configurations for further reranking.

5 Conclusions

We have addressed the task of generating query suggestions that can assist users in completing their tasks. We have focused on the first component of a two-step pipeline, dedicated to creating suggestion candidates. For this, we have considered several methods and multiple information sources. We have based our evaluation on the TREC Tasks track, and collected a large number of annotations via crowdsourcing. Our results have shown that we are able to generate high-quality suggestion candidates. We have further observed that the different methods and information sources lead to distinct candidates.

As our next step, we will focus on the second component of the pipeline, namely, suggestions ranking. As part of this component, we also plan to address the specific issue of subtasks coverage, i.e., improving the diversity of query suggestions.

References

1. Bhatia, S., Majumdar, D., Mitra, P.: Query suggestions in the absence of query logs. In: Proceedings of SIGIR 2011, pp. 795–804 (2011)
2. Cai, F., de Rijke, M.: A Survey of Query Auto Completion in Information Retrieval. Now Publishers Inc., Hanover (2016)
3. Fader, A., Zettlemoyer, L., Etzioni, O.: Open question answering over curated and extracted knowledge bases. In: Proceedings of KDD 2014 (2014)
4. Garigliotti, D., Balog, K.: Generating query suggestions to support task-based search. In: Proceedings of SIGIR 2017, pp. 1153–1156 (2017)
5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**, 1735–1780 (1997)
6. Luong, M., Pham, H., Manning, C.D.: Effective Approaches to Attention-based Neural Machine Translation (2015)
7. Mitra, B., Craswell, N.: Query auto-completion for rare prefixes. In: Proceedings of CIKM 2015, pp. 1755–1758 (2015)
8. Ozertem, U., Chapelle, O., Donmez, P., Velipasaoglu, E.: Learning to suggest: a machine learning framework for ranking query suggestions. In: Proceedings of SIGIR, vol. 12, 25–34 (2012)
9. Pareti, P., Testu, B., Ichise, R., Klein, E., Barker, A.: Integrating Know-How into the Linked Data Cloud (2016)
10. Park, D.H., Chiba, R.: A neural language model for query auto-completion. In: Proceedings of SIGIR 2017, pp. 1189–1192 (2017)
11. Pass, G., Chowdhury, A., Torgeson, C.: A picture of search. In: Proceedings of InfoScale 2006 (2006)
12. Sordoni, A., Bengio, Y., Vahabi, H., Lioma, C., Grue Simonsen, J., Nie, J.-Y.: A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In: Proceedings of CIKM 2015, pp. 553–562 (2015)
13. Verma, M., Kanoulas, E., Yilmaz, E., Mehrotra, R., Carterette, B., Craswell, N., Bailey, P.: Overview of the TREC tasks track 2016. In: Proceedings of TREC 2016 (2016)
14. Yilmaz, E., Verma, M., Mehrotra, R., Kanoulas, E., Carterette, B., Craswell, N.: Overview of the TREC: tasks track. In: Proceedings of TREC 2015 (2015)