K2Q: Generating Natural Language Questions from Keywords with User Refinements

Zhicheng Zheng
Tsinghua University
zhengzc04@gmail.com

Xiance Si
Google Inc.
sxc@google.com

Edward Y. Chang
Google Inc.
edchang@google.com

Xiaoyan Zhu
Tsinghua University
zxy-dcs@tsinghua.edu.cn

Abstract

Garbage in and garbage out. A Q&A system must receive a well formulated question that matches the user's intent or she has no chance to receive satisfactory answers. In this paper, we propose a keywords to questions (K2Q) system to assist a user to articulate and refine questions. K2Q generates candidate questions and refinement words from a set of input keywords. After specifying some initial keywords, a user receives a list of candidate questions as well as a list of refinement words. The user can then select a satisfactory question, or select a refinement word to generate a new list of candidate questions and refinement words. We propose a User Inquiry Intent (UII) model to describe the joint generation process of keywords and questions for ranking questions, suggesting refinement words, and generating questions that may not have previously appeared. Empirical study shows UII to be useful and effective for the K2Q task.

1 Introduction

Keyword search has long been considered an unnatural undertaking task, but it works well with search engines. When entering a question to a Q&A system such as Yahoo! Answers and Quora, however, the keyword paradigm simply does not work. A question must be articulated specifically in the form of natural language. For instance, keywords such as *New York restaurant* is ambiguous in its intent. The user may want restaurants in New York, or want to know the tipping practice at New York restaurants, or something else under the myriad other possible interpretations. For a question to be answered, the asker must articulate her intent with sufficient specificity. Partly because we have been detrained by search engine from writing a question in a complete sentence, and partly because a question is hard to articulate completely when the asker is in a learning mode, a Q&A system should provide tools to users to help them clarify their questions so as to get good answers.

In this paper, we propose K2Q, a system that converts keywords to questions by considering both query history and user feedback. More specifically, given a set of keywords, K2Q generates a list of ranked questions as well a list of refinement words. A user can select a satisfactory question, or she can select a refinement word to generate a new list of candidate questions and refinement words. This process iterates until the user finds a good question matching her intent or quits.

To build an effective K2Q system, there are three aspects that need to be researched. A user issues a question typically because the desired information cannot be found. Our first task, therefore, is in the generation of unseen questions (*unseen questions* refer to those questions which are not known in advance by a K2Q system). After all, popular questions being asked before can be searched via search engines. Second, we must address the challenge of ranking candidate questions. Questions, unlike keywords, rarely repeat. (The same question can be asked in many different forms.) Among the $12,880,882$ questions we have collected, only 1.87% occur more than once. Therefore, a simple ranking scheme based on frequency is not feasible. Third, the suggestion scheme of refinement words must consider maximizing information gain. It is thus desirable to generate diverse refinement words. An intuitive method is to use the most frequent words in candidate questions (after the removal of functional words). However, this method only considers the strength of relatedness between the words and the candidate questions, and may generate several refinement words indicating the same subtopic. For example, when a user inputs *cat feed*, if we simply use the most frequent words as refinement words, we will suggest both *food* and *eat* which indicate the same topic of *feed cat*. Diverse refinement

Proceedings of the 5th International Joint Conference on Natural Language Processing, pages 947–955,
Chiang Mai, Thailand, November 8 – 13, 2011.

words lead to more efficient feedbacks, and thus produce a fewer number of iterations before getting a satisfactory question.

To overcome the aforementioned three challenges, we propose a User Inquiry Intent (UII) model, which describes the joint generation process of keywords and questions. We employ an adaptive language model to describe the process of forming questions, and use automatically induced question templates to create unseen questions. Candidate questions are sorted by their fluency, i.e., the probability of being seen in a Q&A system. We compute the entropy on the distribution of the user's intent and generate refinement words. We then suggest refinement words that maximize entropy gain. Consequently, a satisfactory question can be generated in fewer iterations. Our experiments show that: 1) a template-based approach improves the coverage of suggestions; 2) compared with the baseline method, our UII model improves the ranking of the suggested questions; and 3) the UII model generates better refinement words than the baseline method of suggesting most frequent words. The results suggest that our proposed UII model is effective.

The contributions of this paper can be summarized as follows:

1. We propose a K2Q system, which benefits users to articulate and refine their questions to be more specific and more overtly express their intent.
2. To address the technical challenges of developing K2Q, we propose the UII model, which models the joint generation process of keywords and questions.
3. We show that the UII model is effective in both generating and refining questions by experiments conducted with real-world query logs.

This paper proceeds as follows. Section 2 presents a brief survey of related work. Section 3 details the UII model, which describes the joint generation of both search keywords and natural language questions. Experiments are described in Section 4 and conclusions are given in Section 5.

2 Related Work

To the best of our knowledge, there is no direct research on the K2Q problem. However, query suggestion and question recommendation are among the most relevant tasks that have been researched.

Query suggestion aims to suggest related refined query words to users, and is employed by most modern search engines. Many research efforts have been devoted to query suggestion (Ma et al., 2010; Chirita et al., 2007a) and other, similar tasks such as query expansion (Chirita et al., 2007b; Cui et al., 2003; Theobald et al., 2005; Xu and Croft, 1996) and query refinement (Kraft and Zien, 2004). Applying query suggestion to K2Q is problematic for two reasons: 1) Query suggestion makes heavy use of the query log information such as click sessions. However, less than 1% of queries are questions, and even less of those repeat more than twice. The sparsity of questions renders query suggestion algorithms ineffective when applied to K2Q. 2) To propose new query suggestions, keyword-level editing operations such as *add, delete or change* are used (Jones et al., 2006). These operations do not take grammar into consideration, and are too simple to be applied to whole, complete sentences.

Question recommendation suggests questions which are related to the initial question (Cao et al., 2008; Wu et al., 2008). By treating the initial keywords as a question, question recommendation algorithms can also be used to suggest questions for K2Q. However, question recommendation focuses on proposing existing questions, which fails at the first challenge of generating unseen questions.

Researchers have worked on solving individual challenges posed by K2Q. For question generation, Lin (2008) proposed an "automatic question generation from queries" task in 2008, but no technical approaches were discussed. Kotov and Zhai (2010) proposed to organize search results by corresponding questions. They employed some manually created templates to transform normal sentences into questions. Their work was to generate questions from a paragraph or a sentence, but not according to keywords. In addition, since producing templates requires human effort, it is difficult to achieve high coverage. In IR field, some researchers automatically generate query templates (Agarwal et al., 2010; Szpektor et al., 2011). Inspired by their work, we try to generate templates automatically.

For question ranking, Wu et al. (2008) calculated user-to-question similarity and question-to-question similarity by employing the PLSA model, then ranked candidate questions by combining the two similarity scores. However, it is dif-

ficult to model users in K2Q, so this algorithm is not suitable for our purpose. Cao et al. (2008) proposed an "MDL-based Tree Cut Model" to rank candidate questions. They organized the candidate questions in a tree, then defined question similarity based on both *specificity* and *generality*. They ranked the candidate questions by combining the two similarity scores. Their work aimed to suggest interesting questions to the user, but K2Q proposes to recommend questions with high popularity in order to make correct predictions. Sun et al. (2009) ranked questions with several CQA specific features. However, since the candidate questions in K2Q come from different CQA sites, and some candidates are even unseen questions, it is difficult to apply these features in K2Q. In this paper, we rank questions according to their popularity, which is measured by including an adaptive language model in UII model.

For the generation of refinement words, the most relevant task is query suggestion. Diversity is an important factor in query suggestion, and most of the related works exploited the information of query logs in order to diversify the query suggestion results. Wang et al. (2009) extracted subtopics of a query by mining query reformulations in user session logs. Ma et al. (2010) proposed a method based on Markov random walks and hitting time analysis on a query-URL bipartite graph. Sadikov et al. (2010) clustered query refinements by performing multiple random walks on a Markov graph that approximates user search behavior. However, in K2Q, we cannot get sufficient click information between keywords and questions due to its sparsity in the query logs. In this paper, we use click information to evaluate our methods, but not for training since the total amount is small.

3 User Inquiry Intent Model

When a user inputs a query or posts a question, she wants to get some information. We will refer to the information need as user intent. Both the query and the question are generated from the user intent. We will use an example to illustrate the generative process. A user wants to know what the hot research topics in natural language processing are. If she employs search engines, she might create a query *hot research topics NLP* from her intent. If she wants to post a question in the community, she might choose a way to express her intent. She may choose *What are hot research topics in [subject_areas]*, or *Which research topics are hot in [subject_areas]*. Different ways of posting her intent generate different questions. In this example, let the corresponding final questions be *What are hot research topics in NLP* and *Which research topics are hot in NLP*. The replaceable part (such as *[subject_areas]*) can be considered as slots for concrete words. Generally, a slot can be interpreted as a word or a word cluster. A word cluster is a set of words which can be used in similar contexts. For example, *Beijing* and *Paris* are in the same cluster since they are both suitable to be used in the context *in [cities]*. We obtain the word clusters by using k-means as was shown in (Lin and Wu, 2009).

Here, we propose a User Inquiry Intent model to describe the process of generating queries and questions from the user intent.

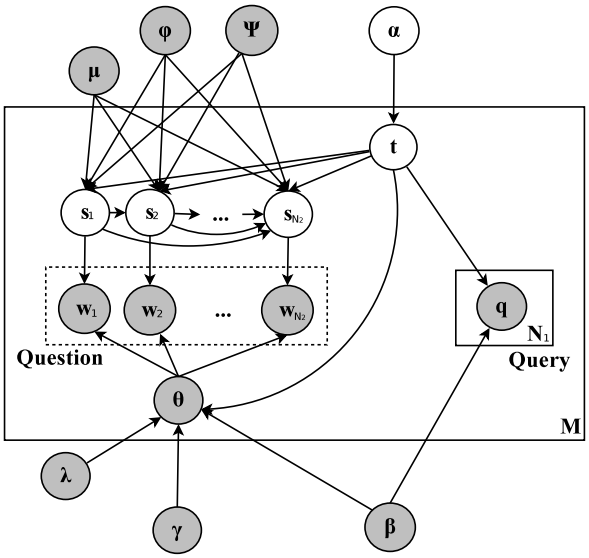

Figure 1: The plate representation of the UII model. (as in Probabilistic Graphical Models)

The plate representation of UII model is shown in Figure 1. In the model:

- t is the index of user intents, ranging from 1 to K (K is the number of different user intents). A user intent not only corresponds to a distribution over all words but also corresponds to a distribution over the slots s.
- w and q are both indices of words, ranging from 1 to W (W is the number of different words). In Figure 1, the left group of w is the question which is able to express the user intent t. The right group of q is the query which is also able to express t.
- s is the index of slots, ranging from 1 to $W +$

L. Besides the W slots which are reserved for exact words, there are an additional L more slots for L word clusters.

- $\vec{\alpha}$ is the prior parameter of the distribution of user intents. $\vec{\alpha}$ is a vector of length K: $\sum_{i=1}^{K} \alpha_i = 1, 0 \leq \alpha_i \leq 1$.
- β is the prior parameter of the word distribution on each user intent t. β is a matrix of size $K \times W$: $\forall i, \sum_{j=1}^{W} \beta_{i,j} = 1, 0 \leq \beta_{i,j} \leq 1$.
- φ is the prior parameter of the slot transition. φ is a matrix. Denote the preceding slots of s as h. Since there are too many possible values of h, in practice we only reserve the frequent N-grams as possible values of h. The size of φ is then $H \times (W + L)$, where H is the size of frequent N-grams. $\forall i$, $\sum_{j=1}^{W+L} \varphi_{i,j} = 1, 0 \leq \varphi_{i,j} \leq 1$.
- γ is the prior parameter of the word distribution on each slot s. γ is a matrix of size $(W + L) \times W$. $\forall i$, $\sum_{j=1}^{W+L} \gamma_{i,j} = 1$, $0 \leq \gamma_{i,j} \leq 1$.
- ψ is the prior parameter of the slot distribution on each user intent t. ψ is a matrix of size $K \times (W + L)$. $\forall i$, $\sum_{j=1}^{W+L} \psi_{i,j} = 1$, $0 \leq \psi_{i,j} \leq 1$.
- θ is a $(W + L) \times W$ matrix. Each row of θ is the word distribution on a slot under a particular user intent.
- λ and μ are two mixture weights of prior distributions. Under user intent t, $\forall i$, $\vec{\theta_i}$ follow the Dirichlet distribution with parameters $\lambda \cdot \vec{\gamma_i} + (1 - \lambda)\vec{\beta_t}$. Under user intent t, the slot transition probability $p(s|h, t)$ is calculated by Eq. 1, which is an adaptive language model (Kneser et al., 1997).

$$p(s|h,t) = \frac{f(s|t)}{z(h,t)}\varphi_{h,s} \qquad (1)$$

$$f(s|t) = (\frac{\psi_{t,s}}{\varphi_{\emptyset,s}})^{\mu}, z(h,t) = \sum_s f(s|t) \cdot \varphi_{h,s}$$

We refer the generative process as Algorithm 1. The process is illustrated with the same example as in the beginning of this section. The user intent has high probabilty on the words *hot*, *research*, *topic*, *NLP*, and hence generates the set of query words *hot research topic NLP*. Assuming that the probability of seeing a slot is determined by two preceding slots, then by considering both the slot distribution on the user intent and the slot transition probability, the user intent first generates *What* from a *START* slot; then *are* from *START What*; *hot* from *What are*; *research* from *are hot*; *topics* from *hot research*; *in* from *research topics*; *[subject_areas]* from *topics in*; and finally an *END* slot from *in [subject_areas]*. Under the user intent, we generate question words from the slots in the slot sequence *What are hot research topics in [subject_areas]*, and form the question word sequence *What are hot research topics in NLP*.

Choose $t \sim Mutinomial(\vec{\alpha})$;
for *each word* q_j *in the query* **do**
  Choose $q_j \sim Mutinomial(\vec{\beta_t})$;
end
Choose $\vec{\theta_i} \sim Dir(\lambda \cdot \vec{\gamma_i} + (1 - \lambda)\vec{\beta_t})$, where $i \in \{1, 2, \ldots, W + L\}$;
for *each word* w_j *in the question* **do**
  Choose $s_j \sim Multinomial(\vec{p}(s|h))$, $p(s|h)$ is calculated as Eq. 1 ;
  Choose $w_j \sim Multinomial(\vec{\theta_{s_j}})$;
end

Algorithm 1: The algorithm for UII model

3.1 Inference

Given prior parameters (for convenience, we denote all prior parameters as π, and all other variables as y), the joint distribution is calculated as:

$$\begin{aligned} p(y|\pi) &= p(t|\vec{\alpha})p(\vec{q}|t,\beta)p(\vec{s}|t,\varphi,\psi,\mu) \\ &\quad p(\theta|t,\gamma,\beta)p(\vec{w}|\vec{s},\theta) \end{aligned}$$

We integrate out θ to calculate $p(t, \vec{q}, \vec{s}, \vec{w}|\pi)$:

$$\begin{aligned} \textstyle\int_\theta p(y|\pi)d\theta &= p(t|\vec{\alpha})p(\vec{q}|t,\beta)p(\vec{s}|t,\varphi,\psi,\mu) \\ &\quad \textstyle\int_\theta p(\theta|t,\gamma,\beta)p(\vec{w}|\vec{s},\theta)d\theta \end{aligned}$$

The right side of the equation is separated into four parts: (1) $p(t|\vec{\alpha}) = \alpha_t$; (2) $p(\vec{q}|t, \beta) = \prod_{i=1}^{N1} \beta_{t,q_i}$; (3) $p(\vec{s}|t, \varphi, \psi, \mu) = \prod_{i=1}^{N2} p(s_i|h_i, t)$, $p(s_i|h_i, t)$ is calculated by Eq. 1; (4) The integration $\int_\theta p(\theta|t, \gamma, \beta)p(\vec{w}|\vec{s}, \theta)d\theta$ is solved by:

$$\begin{aligned} & \textstyle\int_\theta p(\theta|t,\gamma,\beta)p(\vec{w}|\vec{s},\theta)d\theta \\ = \; & \textstyle\prod_{i=1}^{W+L} \int_{\vec{\theta_i}} p(\vec{\theta_i}|t,\beta,\gamma,\lambda) \prod_{j=1}^{W} \theta_{ij}^{n^{(i,j)}} d\vec{\theta_i} \\ = \; & \textstyle\prod_{i=1}^{W+L} \frac{\prod_{j=1}^{W} \Gamma(\lambda\gamma_{i,j}+(1-\lambda)\beta_{t,j}+n^{(i,j)})}{\Gamma(\sum_{j=1}^{W}(\lambda\gamma_{i,j}+(1-\lambda)\beta_{t,j}+n^{(i,j)}))} \cdot \\ & \textstyle\frac{\Gamma(\sum_{j=1}^{W}(\lambda\gamma_{i,j}+(1-\lambda)\beta_{t,j}))}{\prod_{j=1}^{W}\Gamma(\lambda\gamma_{i,j}+(1-\lambda)\beta_{t,j})} \end{aligned}$$

Here, $n = \{n^{(i)}\} = \{\{n^{(i,j)}\}\}$ is a matrix, $n^{(i,j)}$ is the count of slot i generating words j in the question. Function $\Gamma(x)$ is the gamma function.

3.2 Parameter Estimation

As the complexity of the UII model is non-trivial, in this work we choose to estimate the parameters of different components separately instead of performing a global joint optimization.

λ and μ are weight parameters for combining different distributions. We set them empirically.

We estimate the other prior parameters with a given question set (we refer to these questions as known questions). γ is the prior word distribution of slots. There are two kinds of slots, one can be filled with exactly one word, and the other can be filled with any words from the corresponding word cluster. Hence, we estimate γ as:

$$\forall s, i, 1 \leq s \leq W, i \neq s, \gamma_{ss} = 1, \gamma_{si} = 0$$

$$\forall s, i, W < s \leq W + L, \gamma_{si} = p(i|cluster_{(s-W)})$$

Here $p(i|cluster_{(s-W)})$ is the probability of the i-th word in the $(s-W)$-th word cluster. We set the history h to be the l preceding slots, which allows the slot sequence to be an l-order Markov chain. We obtain φ from statistics on the known questions.

In order to estimate $\vec{\alpha}$, β and ψ, we cluster the known questions, where the questions in the same cluster contain the same bag of words (We only exclude the stopwords). Suppose we obtain K question clusters such that each cluster t contains m_t questions, then we estimate $\vec{\alpha}$ as: $\alpha_t = m_t$. According to the statistics of word distributions on each question cluster, we estimate β and ψ as: $\beta_{t,w} = \hat{p}(w|t)$, $\psi_{t,s} = \hat{p}(s|t)$.

3.3 Question Generation

In order to reduce the computational complexity of the model, we do not actually generate all possible questions that the UII model is capable of. Instead, we use a template based method to generate the candidate questions, which are subsequently ranked by the UII model.

Question Templates Generation

To generate unseen questions, we generate question templates from known questions. A question template is a sequence of slots, where each slot is a word or a word cluster. If there are k cluster slots in a template, we refer to it as a k-variable template. Given a set of questions, we initialize a set of 0-variable templates. By replacing one word of a 0-variable template with a corresponding cluster, we obtain a 1-variable template. By merging all the same 1-variable templates, we get a set of 1-variable templates with their support numbers (the support number is the number of 0-variable templates employed to generate the 1-variable template). By increasing the threshold of the minimum support number (denoted as η), we filter out those templates with low quality.

Question Generation from Initial Keywords

Firstly, we search for known questions that contain all the keywords and use them as suggestion candidates (We search for at most $1,000$ questions). For popular keywords such as *New York steakhouse*, we have enough known questions covering all aspects of the inquiry intents. However, for rare keywords such as *Tangshan*[1] *steakhouse*, performing a search in the known questions might yield only a few or no candidates.

Secondly, we generate unseen questions by the use of question templates. By replacing one of the initial keywords with its cluster, we search for 1-variable templates that contain both the cluster and the rest keywords (We search for at most 20 1-variable templates). Then, we replace the cluster slot in the templates with the actual initial keyword, creating a new question.

Both known and generated questions are added into the final candidate question set.

3.4 Question Ranking

With the candidate question set, we use the UII model to rank the candidates by $p(\vec{w}|\vec{q}, \pi)$, the probablity of the question was generated given the keywords. The probability is calculated as:

$$p(\vec{w}|\vec{q}, \pi) = \frac{\sum_t \sum_{\vec{s}} p(t, \vec{s}, \vec{w}, \vec{q}|\pi)}{\sum_{w'} \sum_t \sum_{\vec{s}} p(t, \vec{s}, \vec{w'}, \vec{q}|\pi)} \quad (2)$$

We could perform exact computation (i.e. Dynamic Programming) to compute the sum over all possible $\vec{s}$, however, since each suggestion should be finished in a timely fashion, the complexity of exact computation is not acceptable. Recall the process where we generated candidate questions, we only consider all the question templates that we obtained as all possible slot permutations. Then we only need to sum up all these $\vec{s}$ to calculate the probability, which makes it possible to finish in real-time.

3.5 Refinement Word Generation

The goal of refinement words is to reduce the number of interactions required to reach the desired

[1] A city in China.

question. With the UII model, we use an information theory based approach to select refinement words. We define the entropy on the distribution of the user intents given the initial keywords $Q = \vec{q}$ as:

$$Entropy(\vec{q}) = -\sum_{k=1}^{K} p(t|\vec{q},\pi)\log(p(t|\vec{q},\pi))$$

Here, $p(t|\vec{q},\pi) = \frac{p(t,\vec{q}|\pi)}{\sum_{t'} p(t',\vec{q}|\pi)}$, and $p(t,\vec{q}|\pi)$ is calculated as:

$$p(t,\vec{q}|\pi) = p(t|\vec{\alpha})\prod_{i=1}^{N_2} p(q_i|\beta,t) = \alpha_t \prod_{i=1}^{N_2} \beta_{t,q_i}$$

Then, if the user selects a refinement word q', the resulted entropy gain is:

$$\Delta E(q'|\vec{q}) = Entropy(\vec{q}) - Entropy(\vec{q}\bigcup\{q'\})$$

Entropy gain measures the reduced uncertainty that results by adding q'. As we aim to reduce the number of interaction steps in K2Q, we select those refinement words that maximize the expected entropy gain in each step. The expected entropy gain of a refinement words set R is:

$$\overline{\Delta E(R)} = \sum_{q'\in R} p(q'|R,\vec{q})\Delta E(q'|\vec{q})$$

Here $p(q'|R,\vec{q})$ is the probability that a user will choose q' from R when a set of refinement words R is given to query Q. $p(q'|R,\vec{q})$ is calculated as:

$$\begin{aligned} p(q'|R,\vec{q}) &= \sum_t p(q'|t,\pi)\cdot p(t|\vec{q},\pi) \\ &= \sum_t p(q'|t,\pi)\cdot p(t,\vec{q}|\pi) \\ &= \sum_t \beta_{t,q'}\alpha_t \prod_{i=1}^{N_2}\beta_{t,q_i} \end{aligned}$$

If the user does not select any refinement words from R, then the expectation of entropy gain on R is meaningless. Hence we define our optimization function as Eq. 3:

$$R = \arg\max_R p(R|Q)\cdot\overline{\Delta E(R)} \qquad (3)$$

Here $p(R|Q) = \frac{n(R|Q)}{n(Q)}$, where $n(Q)$ is the number of candidate questions when giving query Q, and $n(R|Q)$ is the number of the candidate questions which contain at least one refinement word in R.

The optimization problem is an NP-complete problem, hence in practice, we use a greedy strategy described by Algorithm 2.

```
Initialize R = ∅, changed = true;
while |R| < MaxNumber and changed do
    Select q' =
    arg max_{q'} p(R⋃q'|Q) · ΔE(R⋃q');
    if p(R⋃q'|Q) · ΔE(R⋃q') >
    p(R|Q) · ΔE(R) then
        R = R⋃q';
        changed = true;
    else
        changed = false;
    end
end
```

Algorithm 2: Greedy strategy algorithm of refinement words generation

4 Experiments

In the section, we use real world questions and Web queries to evaluate the performance of candidate question generation, question ranking and refinement word generation.

4.1 Data Sets

We use a large set of English questions as the training set to generate question templates and estimate the prior parameters in the UII model. There are $12,880,822$ questions in the question set, which are obtained from popular CQA sites such as Yahoo! Answers. Notice that questions from WikiAnswers[2] are intentionally excluded from the training set, as we want to use WikiAnswers' questions as samples of unseen questions to evaluate the performance of question ranking when faced with unseen questions.

To evaluate the performance of K2Q, we need to know the mapping relationship between query and question. To this end, we use Web query logs to create the data set. After the user inputs a query, she will click a result if she thinks it is satisfactory. By only considering the clicked results that are from CQA sites, we collect pairs of queries and their target questions. We employ a one week query log from Google. To avoid data noise, we collect only those query-question pairs which occur at least 10 times in the query log. Under these conditions, we get approximate $500,000$ query-question pairs.

4.2 Evaluation of Candidate Question Generation

First, we show the number of templates under different support numbers (η) in Figure 2. As the figure shows, the number of 1-variable templates drops significantly when we need more supports. According to our observation, $\eta = 3$ suffices as a good balance point between quantity and quality.

[2]http://wiki.answers.com

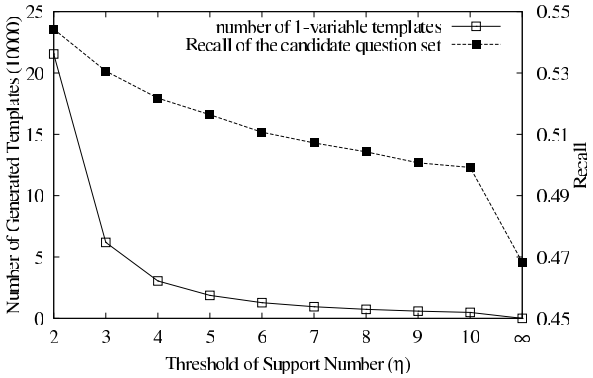

Figure 2: Number of 1-variable templates and recall of candidate questions with different η. The templates help improve the recall (Note: $\eta = \infty$ means no template is used)

Second, we set up an experiment to evaluate the coverage of candidate questions generated by K2Q. We select the top $10,000$ most frequent query-question pairs as the test set. For each query-question pair, we generate candidate questions for the query. We define the query-question pair to be *recalled* in a candidate question set if the set contains the target question in the query-question pair. We use recall to measure the performance of our candidate question generation algorithm, which is defined as:

$$recall = \frac{\#\{\textit{recalled query-question pair}\}}{\#\{\textit{query-question pairs}\}} \quad (4)$$

Recall measures the coverage of the candidate question set. The higher the recall, the more target questions are included in the candidate question sets. Figure 2 also shows the results under different η ($\eta = \infty$ means no 1-variable template is used). Compared to the case when 1-variable templates are not used, our candidate question generation algorithm improved the recall by $6.6\% - 16.2\%$.

4.3 Evaluation of Question Ranking

The baseline method that we compare our approach to is the language model (denoted as LM). The language model ranks the candidate questions according to the probability that it would generate the question. Since a popular question tends to contains popular N-grams, the language model also measures the popularity of the question. Specifically, we use the same question set as that in Section 4.1 to train a 3-gram language model.

To construct the test set, we select $2,000$ query-question pairs that satisfy the following conditions: 1) the target question in the query-question pair is generated as a candidate question; 2) the target question comes from WikiAnswers, as we are more concerned with the ranking performance on unseen questions. The results in Table 1 show that the UII model provides a better ranking of the candidate questions when compared to the language model. We argue this improvement results from two aspects of the UII model: 1) The UII model introduces word clusters, which smooths the probability of rare N-grams. This is important in ranking since the generated unseen questions always contain several rare N-grams. 2) The UII model generates slot sequences via an adaptive langugage model, which helps adjust slot transition probabilities under different user intents.

We select two examples (shown in Table 2) to illustrate the aforementioned aspects of our UII model's superiority to the conventional language model. The target question of the first query is *what is the meaning of advocacy*. Since the 3-gram *meaning of advocacy* never occurs in the training set, LM does not rank the target question as the top 1 best candidate. However, the 3-gram *meaning of WordCluster* occurs frequently in the training set, so the UII model ranks the target question as the top 1 best candidate. The target question of the second query is *what are the different types of Google*. In the training set, the phrases *how many* occurs more frequently than the phrase *what are the different*, so LM does not rank the target question as its number 1 candidate. However, the corresponding user intent has high probability to generate the slot *different*, so the target question is ranked as the best candidate by the UII model.

Recall at	**LM**	**UII model**	Improvement
top 1	37.5%	43.5%	+16.0%
top 2	54.2%	58.0%	+7.0%
top 3	64.1%	66.6%	+4.1%

Table 1: The UII model performs better on the most frequent query-question pairs set

4.4 Evaluation of Refinement Word Generation

We use the query-question pairs to evaluate the performance of the generation of refinement words. We compare the UII model with an intuitive method which selects the most frequent words in the candidate questions as refinements (denoted as MF). The goal of refinement words is to interact with the user in those cases

Query	Question at top 3	
	UII model	**LM**
advocacy meaning	**What is the meaning of advocacy?** The meaning of advocacy? What is meaning of advocacy?	What is the meaning advocacy? **What is the meaning of advocacy?** What is the meaning of the advocacy?
Google types	**What are the different types of Google?** What are the types of Google? How many types of Google?	How many types of Google? Different types of Google? **What are the different types of Google?**

Table 2: Two examples of top 3 suggestions by UII model and LM (The target questions of the queries are shown in bold)

when the initial keywords are too simple to express the user intent. Since one word queries are often too simiple to express the user intent, we use one word queries as the test set. These query-question pairs also satisfy the following two conditions: 1) the target question in the query-question pair is generated as a candidate question; 2) the UII model ranks the target question not to be among the top ten.

We use 623 such query-question pairs as the test set. For each method, we generate a list of 10 refinement words. We use each of the refinement words to refine the query, and then subsequently re-rank the candidate questions and create a new set of 10 refinement words. By exploring all possible choices of refinement words among the list of 10 in each interaction, we find the minimal number of interaction steps that lead to the target questions. We search for at most 5 steps. If K2Q ranks the target question among the top 10 within these 5 interactions, then set the query cost to be the minimal interaction number in which this occured; otherwise, let the query cost be 6. We consider two metrics: 1) the number of query-question pairs where K2Q ranks the target question among the top ten within 5 interactions (denoted as RN); 2) the reciprocal average query cost (denoted as RAQC). The larger RAQC is, the fewer the number of interactions K2Q requires to successfully rank the target question.

	MF	**UII model**	Improvement
RN	363	409	+12.7%
RAQC	0.258	0.265	+2.63%

Table 3: UII model performs better than the baseline method on refinement words generation

From the results in Table 3, we find that: 1) With refinement words, K2Q is more effective of suggesting the target questions. 2) With refinement words generated by the UII model, K2Q gets more efficient feedback, which leads to better performance when compared to the intuitive method. MF only considers the frequency of the words, not the diversity. In contrast, the UII model is effective in suggesting those refinement words that maximally reduce the entropy of the distribution of user intents. It is clear that the UII model suggests a better list of refinement words than the intuitive method.

5 Conclusion

In this paper, we propose the UII model to solve the K2Q problem. The UII model exhibits a simultaneous process of generating both questions and queries based on the user intents. UII model utilizes existing questions on the Web, and leverages templates to generate and rank unseen questions. This model is also equipped with the ability to suggest refinement words that maximize the entropy gain of inferred intent distribution. Experiments show that: 1) using templates improves the coverage of suggestions by $6.6\% - 16.2\%$; 2) the UII model improves the ranking of suggestions over conventional language models with recall at top ranked questions increased over 16%; 3) the UII model suggests better refinement words when compared to a baseline method of suggesting words based on frequency. By interacting via these refinement words, K2Q is able to rank 65.7% target questions within top ten, 12.7% higher than the baseline method. These results show that our proposed UII model yields better suggestions than separated methods in K2Q.

The UII model is a generative model. In this paper, we estimate all the prior parameters without using query-question pairs. Since there are mountains of query-question pairs from query logs, we plan to better estimate the prior parameters with these pairs in future work. Another important direction is to use real-world feedback to evaluate the model, as our current evaluation does not consider real users.

References

G. Agarwal, G. Kabra, and K.C.C. Chang. 2010. Towards rich query interpretation: Walking back and forth for mining query templates. In *Proc. of WWW*. ACM.

Y. Cao, H. Duan, C.Y. Lin, Y. Yu, and H.W. Hon. 2008. Recommending questions using the mdl-based tree cut model. In *Proc. of WWW*, pages 81–90. ACM.

P.A. Chirita, C.S. Firan, and W. Nejdl. 2007a. Personalized query expansion for the web. In *Proc. of SIGIR*, pages 7–14. ACM.

Paul Alexandru Chirita, Claudiu S. Firan, and Wolfgang Nejdl. 2007b. Personalized query expansion for the web. In *Proc. of SIGIR*, SIGIR '07, pages 7–14, New York, NY, USA. ACM.

Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. 2003. Query expansion by mining user logs. *IEEE Transactions on Knowledge and Data Engineering*, 15:829–839.

R. Jones, B. Rey, O. Madani, and W. Greiner. 2006. Generating query substitutions. In *Proc. of WWW*, pages 387–396. ACM.

R. Kneser, J. Peters, and D. Klakow. 1997. Language model adaptation using dynamic marginals. In *Fifth European Conference on Speech Communication and Technology*.

A. Kotov and C.X. Zhai. 2010. Towards natural question guided search. In *Proc. of WWW*, pages 541–550. ACM.

R. Kraft and J. Zien. 2004. Mining anchor text for query refinement. In *Proc. of WWW*, pages 666–674. ACM.

D. Lin and X. Wu. 2009. Phrase clustering for discriminative learning. In *Proc. of ACL*, pages 1030–1038. Association for Computational Linguistics.

C.Y. Lin. 2008. Automatic question generation from queries. In *Workshop on the Question Generation Shared Task*.

H. Ma, M.R. Lyu, and I. King. 2010. Diversifying Query Suggestion Results. In *Proc. of AAAI*.

E. Sadikov, J. Madhavan, L. Wang, and A. Halevy. 2010. Clustering query refinements by user intent. In *Proc. of WWW*, pages 841–850. ACM.

K. Sun, Y. Cao, X. Song, Y.I. Song, X. Wang, and C.Y. Lin. 2009. Learning to recommend questions based on user ratings. In *Proc. of CIKM*, pages 751–758. ACM.

I. Szpektor, A. Gionis, and Y. Maarek. 2011. Improving recommendation for long-tail queries via templates. In *Proc. of WWW*, pages 47–56. ACM.

M. Theobald, R. Schenkel, and G. Weikum. 2005. Efficient and self-tuning incremental query expansion for top-k query processing. In *Proc. of SIGIR*, pages 242–249. ACM.

X. Wang, D. Chakrabarti, and K. Punera. 2009. Mining broad latent query aspects from search sessions. In *Proc. of SIGKDD*, pages 867–876. ACM.

H. Wu, Y. Wang, and X. Cheng. 2008. Incremental probabilistic latent semantic analysis for automatic question recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 99–106. ACM.

J. Xu and W.B. Croft. 1996. Query expansion using local and global document analysis. In *Proc. of SIGIR*, pages 4–11. ACM.