Mining Semi-Structured Online Knowledge Bases to Answer Natural Language Questions on Community QA Websites

Parikshit Sondhi
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
sondhi1@illinois.edu

ChengXiang Zhai
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
czhai@illinois.edu

ABSTRACT

Over the past few years, community QA websites (e.g. Yahoo! Answers) have become a useful platform for users to post questions and obtain answers. However, not all questions posted there receive informative answers or are answered in a timely manner. In this paper, we show that the answers to some of these questions are available in online domain-specific knowledge bases and propose an approach to automatically discover those answers. In the proposed approach, we would first mine appropriate SQL query patterns by leveraging an existing collection of QA pairs, and then use the learned query patterns to answer previously unseen questions by returning relevant entities from the knowledge base. Evaluation on a collection of health domain questions from Yahoo! Answers shows that the proposed method is effective in discovering potential answers to user questions from an online medical knowledge base.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Retrieval Models

1. INTRODUCTION

Community QA (cQA) websites such as Yahoo! Answers, are highly popular. However, many questions are either left unanswered, partially answered or are answered with a significant delay. Past research has tackled this problem by finding questions from the archive, similar to the one posed by the user [7]. In many cases however, similar questions may not already exist or may be hard to find.

In this paper we propose a novel and complementary solution. We believe that many of the questions may be answerable via online knowledge-base websites such as wikipedia[1] or eMedicinehealth[2]. The following example shows that a question about lowering blood pressure can be potentially automatically answered by mining an answer paragraph from eMedicineHealth:

Question: *What is a good way to lower blood pressure? I would like to go off of my meds if possible. I would like to know the right types of foods and drinks and self care i can do. Thanks in advance.*

Answer value from eMedicineHealth: *Lifestyle changes are important to help control high blood pressure. Even if your doctor has prescribed medicine for you, you can still take many steps at home to lower your blood pressure and reduce your risk. Some people can even take less medicine after making these changes. Make these lifestyle changes to help lower your blood pressure:Lose extra weight,Eat healthy foods....*

Such knowledge-base websites tend to follow a standard structure. Each webpage discusses different aspects of a particular entity. For example an article on Bronchitis[3] on emedicinehealth, discusses the various aspects of the disease entity "Bronchitis", like "causes", "symptoms", "treatment", "prognosis" etc. Moreover, each aspect generally appears as a section heading (eg. "Bronchitis Treatment") followed by its text description.

One can view this data as being organized in a relational database with schema represented by the section headings. For example a relation with attributes "(Disease,Symptoms)" will store tuples of the form "(Bronchitis,<Text describing symptoms of Bronchitis>)". Note that while the values stored in the database are predominantly verbose text descriptions, there are no restrictions on the kind of information that can be stored. For example one could define more complex relations that store numeric values eg. "(Disease,Drug,Dosage)", where the attributes "Disease" and "Drug" store text values while "Dosage" is numeric.

Our goal in this paper is to answer a new question by mining the most suitable text value from the database. We will subsequently refer to such a domain specific relational database as a knowledge-base to distinguish it from a regular database. We will also refer to this novel problem of answering questions by mining text values from it, as

[1]http://www.wikipedia.org

[2]http://www.emedicinehealth.com

[3]http://www.emedicinehealth.com/bronchitis/article_em.htm

CIKM'14, November 3–7, 2014, Shanghai, China.

http://dx.doi.org/10.1145/2661829.2661968.

knowledge-based Question Answering (kbQA). We will also frequently use the terms "document", "value" and "database value" interchangeably to refer to a specific atomic value in the knowledge-base.

The kbQA is a novel text mining problem which has clear difference from the existing information retrieval and question answering tasks. First, it differs from regular text retrieval in that rather than retrieving documents based only on keyword/semantic similarity, the presence of relations between text values also offers us the opportunity to perform limited "reasoning" via sql queries. Thus the challenge in kbQA is to identify relevant sql queries that can help retrieve the answer to the question. In this regard our problem is closer to research in Natural Language Interfaces (NLI), since both attempt to return values from a database [12, 11, 15]. However the nature of questions posed, databases used and the answers expected are all quite different from those in our case. In NLI applications, the natural language input from a user are often short and precise commands or requests making them relatively easy to parse. The databases also generally contain precise values which are easy to match with the keywords in the input command. Consequently, the methods used in NLI involve parsing the question text and directly trying to infer an underlying well-formed relational database query. In contrast, questions posted on community QA websites are uniformly real questions, and tend to be long, noisy and verbose, with a lot of background information, which makes NLI parsing methods unsuitable. Moreover our goal is not to find a unique sql query. Infact since the knowledge-base values are also verbose, the question will partially match many different values and there will likely be multiple sql queries capable of retrieving the answer. More details differentiating our work from other related areas such as question answering are provided in section 2.

To solve this novel text mining problem, we propose a general probabilistic framework that would generate the answer in two steps. First we generate candidate sql queries along with a confidence score on their likelihood of generating the answer. We then execute the high scoring queries to generate candidate answer values. The final score of each candidate value is based on the confidence scores of all queries that generated it.

We further use the framework to derive a specialized model for answering questions on cQA websites, that takes into account the noisy descriptive questions and verbose values. In particular, we show how the set of relevant sql queries may be mined, and the parameters of the model estimated, by automatically generating a training set from questions already available on cQA websites.

We evaluate our model by building a kbQA system for answering healthcare questions on Yahoo! answers, using wikipedia as the knowledge-base. Results show that it is indeed feasible to answer cQA questions using our method.

To summarize, this paper makes the following major contributions:

1. We introduce and study the novel knowledge-based Question Answering (kbQA) problem, where online knowledge databases are leveraged to automatically answer or facilitate answering unanswered questions posted on community QA websites.

2. We propose a general probabilistic framework for solving the kbQA problem and show how it may be instantiated to build a QA system for web communities.

3. We show how relevant queries may be mined and model parameters estimated using an automatically generated training set.

4. We evaluate the proposed framework in healthcare domain and demonstrate the proposed new strategy of KBQA and effectiveness of the proposed QA algorithms.

2. RELATED WORK

Prior work in automatically answering cQA questions has centered around finding similar questions from within a QA archive. These approaches tend to treat a question thread as a structured document with question and answer fields and focus on identifying the semantic similarity between questions. For example, Jeon et. al. [7] propose a retrieval model which exploits similarity between answers of existing questions to learn translation probabilities, which allows them to match semantically similar questions despite lexical mismatch. A subsequent work by Xue et. al. [16] combines a translation-based language model for the question field with a query likelihood model for the answer. Other related approaches include finding answers from frequently asked questions on the web [14, 8]. Our work differs from these approaches in that our answers come from a knowledge base. Matched database values in our case only serve as constraints to query with and are not themselves sufficient to uniquely identify an answer. We must in addition also identify an appropriate target to retrieve.

Another related area is that of structured document retrieval, where the focus is on developing retrieval models for documents with well defined fields. Many field based models have been developed in the past including BM25F [13, 9, 10]. The principle idea behind them is that each query keyword may have been intended to match the content of a specific field. Hence they tend to assign different weights to document fields while estimating relevance.

Extensive research has also been done in developing question answering systems. Over the past few years, both open [1] and closed domain systems [2] have been proposed. However most of these approaches are designed to answer only a restricted set of questions such as short and precise factoid [3] or definitional [5]. In addition they tend to require deep natural language processing steps such as generation of parse trees to analyze the syntax and semantics of the question, which are unlikely to work well with noisy cQA data.

3. PROBLEM DEFINITION

In this section we provide a formal definition of the kbQA task and the notations that will be used in subsequent sections.

Problem: Given a knowledge database D and a question q, our goal is to return a database value v_a as the answer.

Input: There are two input variables q and D.

q may be (semi-)structured or unstructured. While our proposed framework does not restrict the nature of q, questions encountered in community QA websites tend to be predominantly unstructured.

The database D comprises a set of relations $R = \{r_1, .., r_n\}$. The schema of each r_i is represented via the set of attributes $A_i = \{a_{i1}, a_{i2}, ..., a_{iK_i}\}$. Each attribute a_{ij} represents the jth attribute in the ith relation and is considered unique. Finally we define $A_D = \cup_{i=1}^{n} A_i$ as the set of all database attributes and use $a \in A_D$ to represent some attribute in D. For example in a knowledgebase with two relations $R = \{Dis\_Treat, Dis\_Symp\}$ with attributes $(Disease, Treatment)$ and $(Disease, Symptoms)$ we get

$$\begin{aligned}
r1 &= Dis_Treat \\
A_1 &= \{Dis_Treat.Disease, Dis_Treat.Treatment\} \\
a_{11} &= Dis_Treat.Disease \\
a_{12} &= Dis_Treat.Treatment \\
r2 &= Dis_Symp \\
A_2 &= \{Dis_Symp.Disease, Dis_Symp.Symptoms\} \\
a_{21} &= Dis_Symp.Disease \\
a_{22} &= Dis_Symp.Symptoms \\
A_D &= A_1 \cup A_2 \\
&= \{Dis_Treat.Disease, Dis_Treat.Treatment, \\
&\quad Dis_Symp.Disease, Dis_Symp.Symptoms\}
\end{aligned}$$

Note that even though the attributes $Dis\_Treat.Disease$ and $Dis\_Symp.Disease$ have the same semantic interpretation, we treat them as two different and unique attributes. Consequently the total attributes in a database is just the sum of number of attributes in individual relations i.e. $\sum_{i=1}^{n} |A_i| = |A_D|$.

A database value v_{ijk} represents the atomic value appearing in the kth tuple of the ith relation under the j attribute. We define $Attr(v) \in A_D$ to represent the attribute of the value v i.e. for $v = v_{ijk}$, $Attr(v_{ijk}) = a_{ij}$. Finally we use V_D to represent the set of all knowledge-base values.

Output: The expected output is a value $v_a \in V_D$ such that v_a forms a plausible answer to q. We assume the question is answerable through a single value in the database.

We view the core computation problem as that of ranking knowledge-base values based on their probability of being the answer and return the one that scores the highest. Alternatively depending upon the application needs and probability scores, we can also return more than one or no values as well.

4. A GENERAL PROBABILISTIC FRAMEWORK FOR KBQA

Intuitively, our approach involves the following steps:

1. First we identify values in the knowledge-base that are similar to the question. These values can be considered as information the user has already provided.

2. Next we "reason" with these values by incorporating them as constraints in sql queries. The knowledge-base values returned by executing these queries become candidate answers.

3. Finally we rank the values returned by the sql queries based on their probability of being the answer.

Consider an example in the healthcare domain. For simplicity we assume, our knowledge-base only contains a single relation *Rel* with three attributes (Disease, Symptoms, Treatment), and stores values for only two diseases i.e. two tuples $\{(dis1, symp1, treat1), (dis2, symp2, treat2)\}$. Now let a user asks a question describing a set of symptoms and expects a treatment description in response. In the first step we match the question to all values stored in the knowledge-base. Each of the 6 values in the knowledge-base will have some similarity score, with the symptom descriptions $symp1$ and $symp2$ likely having the highest. Consequently, a subset of queries we can execute will be (here we refer to $symp1$ and $symp2$ as constraints)

```
select Symptoms from Rel where Symptoms = symp1
select Symptoms from Rel where Symptoms = symp2
select Treatment from Rel where Symptoms = symp1
select Treatment from Rel where Symptoms = symp2
```

The first two queries return the matched symptom value itself. This is equivalent to the behavior of a traditional retrieval method. The next two queries will return the treatment text value corresponding to the matched symptom. These queries are more likely to retrieve the answer the user expected. Our confidence in the query to be executed will depend on a) how relevant the query is to the question and b) how well its constraint matches the question.

In a more general case, we may have to consider 100s of matched knowledge-base values as constraints and potentially 1000s of candidate queries of arbitrary complexity. This significantly complicates the problem of finding the most relevant answer value. In subsequent discussion we present a principled way of approaching the problem through a general probabilistic framework for modeling the uncertainties.

We begin with the conditional probability $P(V = v|Q = q)$, which is the probability that a value v in the knowledge-base is the answer to the question q. The best answer v_a would then be given by maximizing over the set of all possible values in the database V_D:

$$v_a = \arg max_{v \in V_D} P(V = v|Q = q)$$

We now decompose this expression into two conditional probabilities "bridged" by a sql query that can potentially capture the semantics of the user's question q. Formally, let S_D be the set of legitimate queries that one is allowed to execute on the database D, then the probability of a value v may be obtained by marginalizing over it:

$$P(V = v|Q = q) = \sum_{s \in S_D} P(V = v|S = s, Q = q) P(S = s|Q = q)$$

This decomposition has a quite meaningful interpretation: $P(S = s|Q = q)$ captures the uncertainty in inferring what query to execute for question q. Since a query could potentially return multiple knowledge-base values, $P(V = v|S = s, Q = q)$ captures the uncertainty in deciding the answer among the returned values. In cases where a query only returns a single value $P(V = v|S = s, Q = q)$ trivially becomes 1. On the other hand if v is not among the values generated by s, it is 0. To keep the notation simple, in subsequent text, unless otherwise necessary we will drop the random variables and simply write for example $P(S = s|Q = q)$ as $P(s/q)$.

In theory the set S_D could encompass all possible queries executable on the knowledge-base. However we want to restrict it to only those queries which we feel are relevant in answering questions. To this end, we can make additional simplifying assumptions. We will restrict S_D to only queries

that have a single target attribute and use a single value as constraint. The first assumption is natural since we are trying to retrieve a single value. As we will see later, the second assumption is also not particularly restrictive.

A sql query is a complex object which can be encoded using many different features such as "constraint used","target attribute", "number of relations touched" etc. We will encode a query using two features - its constraint and the target attribute i.e.

$$P(v|q) = \sum_{s \in S_v \subset S_D} P(v|s,q)P(Cons(s), Att(s)|q)$$

Where $Cons(s) \in V_D$ is the constraint used in query s, $Att(s) \in A_D$ is its target attribute and $S_v \subset S_D$ is the set of queries that generate the value v. Assuming that the inference of target attribute and the constraint given a question are independent, we can further simplify the equation as:

$$P(v|q) = \sum_{s \in S_v \subset S_D} P(v|s,q)P(Cons(s)|q)P(Att(s)|q)$$

Finally since any candidate answer value appears only under a single attribute in the knowledge-base, assuming $Att(v)$ to be the attribute of v, we can write the final ranking function as

$$\begin{aligned} P(v|q) &= P(Att(v)|q) \sum_{s \in S_v} P(v|s,q)P(Cons(s)|q) \\ logP(v|q) &= log(P(Att(v)|q)) \\ &\quad + log(\sum_{s \in S_v} P(v|s,q)P(Cons(s)|q)) \end{aligned}$$

Note from the equation that the distribution over the target attribute and the constraint appear as separate components in the log. This is a major benefit of the model. It means that when maximizing the log likelihood over some training data, we can estimate the parameters of these two distributions independently.

Also note that the distribution over constraints, appears in a summation. This means that while calculating $P(v|q)$ we sum over all constraints that can be used in queries to generate v. Thus restricting S_D to only single constraint queries, still allows v to receive contributions from all relevant constraints.

Finally the equation suggests that in order to build a kbQA system, one needs to instantiate the following components

Legitimate Query Set: S_D defines the set of legal queries which the system is allowed to execute to retrieve an answer. For most large knowledge-bases, this will have to be automatically inferred by mining queries from some collection of known QA pairs.

Constraint Prediction Model: $P(Cons(s)|q)$ captures our uncertainty on whether some knowledge-base value can be used as a constraint for the question. Note that $Cons(s) \in V_D$. Hence it is a distribution over all knowledge-base values, conditioned on the question.

Attribute Prediction Model: $P(Att(v)|q)$ captures our uncertainty on the attribute of the value expected in the answer. It is a distribution over all possible attributes $Att(s) \in A_D$, given a question. Its effect is similar in spirit to the question type detection that generally forms an initial step in most QA systems [6].

Value Prediction Model: $P(v|s,q)$ allows us to favor some values over others based on question features, among all values generated by query s. For example, we can use it to ensure some degree of textual similarity between the answer and the question. When no such preferences exist, the distribution can simply be made uniform. For this work we will assume the distribution is uniform i.e.

$$P(v|s,q) = \frac{1}{|Val(s)|}$$

where $Val(s)$ is the set of all values generated on executing the query s.

In the next section we describe how the different components may be instantiated to build a QA system for community QA websites.

5. KBQA FOR WEB COMMUNITIES

In this section we discuss a possible instantiation of general kbQA suitable for web communities and show how its parameters may be estimated. We discuss all components except query mining which is discussed later in section 6.2 after describing the knowledgebase.

5.1 Constraint Distribution ($P(Cons(s)|q)$)

Since both questions and the database values tend to be verbose, one cannot ascertain the existence of a database value merely by looking at whether a question contains it. The constraint model instead needs to incorporate an intelligent similarity function between the question and a database value. In addition, questions tend to contain a number of discourse related background keywords such as "thanks in advance" etc., which we need to filter out. We define the probability of a knowledge-base value $v \in V_D$ of being a constraint for a question q to be

$$P(Cons(s)|q) = \frac{e^{\alpha Sim(Cons(s),q)}}{\sum_{v' \in V_D} e^{\alpha Sim(v',q)}}$$

Where for some v, $Sim(v,q)$ denotes textual similarity between v and q calculated by an intelligent similarity function and α is a parameter. $Sim(v,q)$ is defined by treating the task of finding a matching database value as a retrieval problem, with the question q as a query and a value v as the document. We use the KL-Divergence retrieval model [17] for scoring relevance of values.

$$Sim(v,q) = \sum_{w \in Vocab} P(w|\theta_q) logP(w|\theta_v)$$

where θ_q and θ_v are multinomial word distributions for question and value respectively.

The value model θ_v characterized as the word distribution $P(w|\theta_v)$ is estimated using Dirichlet prior smoothing over the collection [17]:

$$P(w|theta_v) = \frac{c(w,V) + \mu P(w|V_D)}{|D| + \mu}$$

where $c(w, D)$ is the count of word w in the document D, $P(w|V_D)$ is the probability of w in the entire collection. Optimal value of μ is generally set at 4800.

To remove the background keywords while estimating the question model θ_q, we treat the question as being generated from a two component mixture model, with a fixed background model θ_{C_Q} which represents the probability of a word w in a large collection of questions C_Q. Probability of a word in a question is the given by

$$P(w) = (1 - \lambda)P(w|\theta_q) + \lambda P(w|\theta_{C_Q})$$

The model θ_q can then be estimated using Expecation Maximization as in [18]. In effect, the estimated $P(w|\theta_q)$ would favor discriminative words and give them high probabilities.

5.2 Attribute Distribution ($P(Att(v)|q)$)

Attribute prediction can be viewed as a multi-class classification task over question features. We therefore model the distribution $P(Att(v)|q)$ with $Att(v) \in A_D$ being the target attribute, using a maximum entropy model.

$$P(Att(v)|q) = \frac{e^{w^T_{Att(v)} \cdot q_F}}{\sum_{a \in A_D} e^{w^T_a \cdot q_F}}$$

where w_a is the weight vector for attribute a and q_F is the vector of question features.

Question features q_F are defined over n-grams (for $n = 1$ to 5) and information gain is used to select top $25K$ most informative features. Parameters for the attribute prediction model are learnt using the L-BFGS [4] method using the Mallet [4] toolkit. A default gaussian prior with interpolation parameter set to 1 is used for regularization.

Based on our definition of unique attributes in section 3, we treat any two attributes appearing in different relations as different classes for the attribute distribution. For some schema, it may be the case that attributes across two relations are semantically the same. While we do not encounter this case in our knowledge base, in such scenarios one can group together all such attributes into a single attribute class. Alternatively if we know for sure that certain attributes are unlikely to contain an answer value, we can simply ignore them.

5.3 Answer Ranking

Once the set of queries S_D and all component models are estimated, the final ranking algorithm for a new question q works as follows:

1. Constraint Selection: Rank all knowledge-base values based on $sim(v, q)$. Select the top-N values to be used as constraints. We set $N = 10$ for all our experiments.
2. Attribute Selection: Generate features for q and use the attribute prediction model to assign scores to every target attribute.
3. Query Selection: Find all queries in S_D containing one of the selected constraints and a target attribute with a non-zero probability and execute them.
4. Answer Selection: Score each candidate value v by summing over S^e_v the set of all queries that were executed and generated v

$$score = w^T_{Att(v)}.q_F + log(\sum_{s \in S^e_v} \frac{e^{\alpha Sim(Cons(s),q)}}{|Val(s)|})$$

5. Return the highest scoring value as the answer.

[4]http://mallet.cs.umass.edu/

5.4 Training Set Generation

In order to mine the queries and estimate the parameters of the attribute and constraint distributions, we need to construct a training set containing questions and their suitable answers from the knowledge-base. This is achieved by leveraging a collection of $80K$ existing healthcare questions threads from Yahoo! Answers website. For any question q, let $\{a_1, a_2...a_n\}$ be the answers provided for it by the users. We want to find a knowledge-base value $v_a \in V_D$ which is most similar to the user provided answers and treat that as the answer.

More specifically, we use the KL-divergence retrieval model, the same function used for constraint selection, to generate a ranking of values for each answer a_i. Just like the questions, answers also tend to be noisy and may at times be completely irrelevant. Hence we learn the answer model θ_{a_i} in the same manner as we learnt θ_q in constraint prediction, using a background answer model θ_{C_A} over a large collection of answers. We ignore all answers with less than 30 words to ensure quality.

Once the rankings for each answer are generated, we now need to aggregate them. For each retrieved value, we pick the the K best ranks assigned to it by the answers and average them to generate the final score. The value with the lowest score is labeled as the answer. In order to ensure that we only generate training examples for which we are confident, we also reject an instance if:

1. A question has less than K answers available.
2. The lowest scoring value is not ranked at the top by at least two answers.
3. Either two or more values end up with the same lowest score

The parameter K was tuned using a small fully judged validation set discussed in section 8.2.

6. KNOWLEDGEBASE CONSTRUCTION AND QUERY MINING

In this section we discuss how we used wikipedia to construct our healthcare knowledgebase and subsequently mined queries to define our legitimate query set S_D.

6.1 Wikipedia Knowledgebase

We used wikipedia to build our knowledge base. We chose wikipedia because it covers a wide range of healthcare related information, in a language that tends to be at the same level of sophistication as a naive user asking questions on a cQA website. In addition wikipedia is comprehensive enough to contain answers to many questions raised by users.

Finally the strategy used for building the healthcare knowledge base from wikipedia could be used to also build knowledge bases for other domains.

However since there is no fixed set of well defined attributes known beforehand. As a result some effort needs to be put in, *a*) Identifying relevant domain specific wikipedia pages and *b*) Converting the section headings into attributes. In this section we detail the process we followed in converting the raw wikipedia pages into a semi-structured knowledge base.

6.1.1 Identifying the domain related wiki pages:

Most wikipedia articles are assigned one or more high level category tags which loosely identify the high level topic of the page. The category keywords are further organized into a hierarchy based on the generality and "is-a" relationships. For example the category "Health" covers 33 subcategories such as "Diseases and Disorders", "health Law" etc. each of which themselves contain multiple sub-categories. More details regarding wikipedia categories are available here[5]. CatScan[6] is a tool that allows users to browse through the wikipedia category trees.

In all we identified categories covering a total of $29K$ wikipedia pages on healthcare related entities. These spanned pages related to Diseases, treatments, medical techniques, drugs etc.

In order to extract these pages we downloaded the entire english wikipedia dump[7] and processed it using the WikipediaExtractor python script available at[8]. The script allowed us to parse the wiki notation and ultimately obtain plain text with some xml markup identifying section headings.

6.1.2 Identifying the Attributes:

Once the domain relevant pages were identified, our next goal was to identify relations and attributes from the semi-structured wikipages. The section headings in wiki pages are good for this. For example when looking at a wikipage on "Malaria" observe that the section headings consist of "Introduction", "treatment" etc. Other disease related pages tend to have similar section headings. Thus the text under "Treatment" is related to the "Entity" malaria by the "treatment" relation. In other words we can define a treatment relation with two attributes "Entity" and "Treatment Text" which will contain tuples of the form "(Malaria,"<Text Description covering malaria treatment>")".

It is clear that from the standpoint of answering user questions, the attributes that appear frequently i.e. appear in more entities are more likely to be useful. The query templates learnt over these attributes will be more applicable in answering questions across entities.

In order to identify the most useful attributes we rank all section headings based on the frequency of entity pages they appear in. We pick nearly 300 top headings (each appearing in atleast 25 pages) and manually analyzed them. Some irrelevant headings such as "See Also", "References" etc. were eliminated. Of the remaining, many similar headings were merged into the same attribute. For example "Signs and Symptoms", "Symptoms", "Symptoms and Signs" etc. were all grouped into the same attribute "Signs and Symptoms". Similarly "Adverse effects", "Side effects" "Side-effects" "Side Effects" "Health effects" etc. were all merged under the same attribute "Side Effects". Ultimately after pruning and merging we ended up with 59 attributes covering a total of $68K$ text values over the $29K$ entities.

The nature of these relationships is such that each text value is related to only a single entity.

6.1.3 Defining Extended Relations:

In step two we were able to define relationships between entities and various text descriptions. However the entities themselves are also related to each other. For example the drug "Chloroquine" is a "Medication" of the disease "Malaria". Incorporating such relationships into the knowledgebase is critical to answering user questions.

A good way to identify such entity-entity relations is to simply define them based on the attribute in whose text value an entity appeared in. For example "Chloroquine" appears in the medication description of "Malaria", which is a text value related to "Malaria" via the "Medication Text" relation.

To this end we further define 59 additional relations of the form "Entity, <Attribute> Entity" for example the relation medication entity has the attributes "(Entity,Medication Entity)" and a sample tuple "(Malaria, Chloroquine)"

As a result after defining the extended relations our final knowledge base comprised of 118 relations covering $29K$ entities and $68K$ text values. We believe there was a significant overlap between the knowledge-base and the cQA collection.

6.2 Mining Legitimate Query Set (S_D)

Recall a sample sql query we discussed earlier

$s1$: `select Treatment from Rel where Symptoms = symp1`

We refer to "symp1" as a constraint and the remaining query as the template i.e.

$t1$: `select Treatment from Rel where Symptoms = <some symptom value>`

Note that while query $s1$ is only useful in answering questions that match the symptom value $symp1$, the template $t1$ allows us to generalize across questions and is useful for answering any question that provides a symptom description and expects a treatment in the answer. $t1$ intuitively captures a kind of general "reasoning" that may be performed to answer some of the questions that provide symptom descriptions. Hence we can treat any sql query that is generated by adding a constraint with template $t1$ as legitimate. More generally, our main intuition behind defining S_D is to identify a set T of such templates, and assume that any query they generate is in S_D.

Now assuming we have a training set of questions and their answers from the knowledge base available, we can use it to first discover single constraint queries, and then convert them into templates by simply dropping the constraint. More general templates will naturally be found in multiple training question-answer pairs.

[5]http://en.wikipedia.org/wiki/Wikipedia:Categorization
[6]http://meta.wikimedia.org/wiki/CatScan
[7]http://en.wikipedia.org/wiki/Wikipedia:Database_download
[8]http://medialab.di.unipi.it/wiki/Wikipedia_Extractor

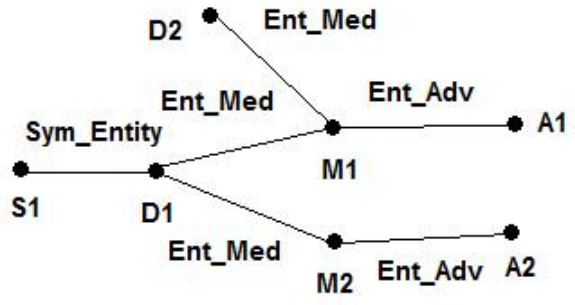

Figure 1: Knowledgebase in Table 1 viewed as a graph

Thus the main challenge in mining such templates is to identify a sql query given a question, its answer value and the knowledgebase. Our approach to solving this problem is to view our entire wikipedia knowledgebase as a graph. Each value in the knowledgebase (either verbose text, or entity name) is treated as a node in the graph. Any two values that appear in the same tuple are connected via an edge. Naturally any two neighboring nodes are bound to have some relationship between them. This relationship is assigned to the edge as a label.

Consider the three relations in Table 1 which are similar to those in our wikipedia knowledgebase. The corresponding knowledge base graph is shown in Figure 1.

Entity	SymptomText
$D1$	$S1$
Entity	MedicationEntity
$D1$	$M1$
$D1$	$M2$
$D2$	$M1$
Entity	AdverseEffectsText
$M1$	$A1$
$M2$	$A2$

Table 1: A Sample Knowledgebase to be mined

Each edge in the graph is assigned a label of the form $Attribute1\_Attribute2$ which represents the relationship between the nodes. Now assuming that our question matched the constraint $S1$ and our answer contained the value $A1$, we first obtain the shortest path between the two nodes. Which in this case is $S1 \rightarrow D1 \rightarrow M1 \rightarrow A1$. Once the path is found, we traverse from the constraint node ($S1$), one step at time towards the answer node ($A1$). In each step a new sql construct of the following form is added in a nested fashion.

```
select Attribute2 from
<Relation containing Attribute1_Attribute2> where
Attribute2=<current node value>
```

More specifically in the first step we generate

```
select Entity from Entity_SymptomText where Symptom-
Text = S1
```

The eventual query template generated is

```
select AdverseEffectsText from
Entity_AdverseEffectsText where Entity = (select Med-
icationEntity from Entity_MedicationEntity where En-
tity = (select Entity from Entity_SymptomText
where SymptomText = <symptom text value>)
```

Note that the queries that get generated in this fashion may at times end up returning multiple values along with the answer value. Especially because multiple entity values may result from one of the nested queries. However our model does take that into account through the value prediction model $P(v|s,q)$ and will favor queries generating fewer values.

To summarize, for each question in the training set, we used the top 10 knowledge base values found by the constraint prediction component as our constraints i.e. start nodes, and the answer value as the end node. Shortest paths were then found in each case and converted into query templates. All templates found in more than 2 questions were considered as legitimate.

7. EXPERIMENTS

7.1 Validation Set for Training Set Generator

In order to validate our training set generation method and tune its parameter K, we needed to create a small fully judged collection containing questions and their answers from a knowledge base. For this we used 21 questions from Yahoo! answers cQA website and manually exhaustively labeled all their answers in a secondary database constructed from eMedicinehealth. We preferred eMedicineHealth, since it is much smaller than wikipedia making it possible to easily create a fully judged collection. The information is mainly centered around diseases and their related properties. The schema comprises 13 attributes such as $Disease$, $Treatment$, $Symptoms$ etc. In all it contained information on 2285 diseases with a total of 8825 values. The judgement set contained an average of 3.5 relevant values per question (out of a total 8825). All judgements were reviewed by a medical expert.

7.2 Automated Evaluation

We processed $80K$ healthcare questions from Yahoo! Answers website using our automated training set generation approach to generate an automated evaluation set. Many questions were filtered out by one or more filtering criteria (see section 5.4). The resulting dataset contained $5.7K$ questions, for each of which a value from our wikipedia knowledge-base was identified as the answer. This dataset served as a large automated evaluation set for training and evaluation of our approaches.

For automatic evaluation we used 5-fold cross validation. In each fold $4.6K$ instances were used for finding the template set S_D, learning the attribute prediction model and tuning the parameters of the constraint prediction model. The learnt model was then evaluated on the remaining $1.1K$ questions.

7.3 Manual Evaluation

We also created a smaller manual evaluation set consisting of 60 manually judged questions. The judgements for this dataset were created by employing a pooling strategy often used in information retrieval evaluation. For each question, we pooled together top 5 values returned by different ranking methods

1. The $Sim()$ function used in constraint prediction. This represented a state of the art information retrieval method.
2. Our kbQA method trained on the automatically generated training set (excluding the 60 manually judged queries)
3. Training Set Generator

Each of these values was then judged by a medical expert as being a relevant or irrelevant answer to the question. The resulting relevant values then became our final evaluation set. In all 116 relevant answers were found.

For manual evaluation, we trained on all $5.7K$ training questions excluding the 60 manually judged questions which were used for evaluation.

7.3.1 Evaluation Metrics

We used Success at 1 (S@1), Success at 5 (S@5) and Mean Reciprocal Rank (MRR) for evaluation. Each criterion allows us to analyze a different aspect of the performance and guess utility for a specific application. S@1 gives us the percentage of questions answered correctly at rank 1 and is relevant if for example we want to automatically post an answer response on some cQA website. On the other hand MRR and S@5 are more useful if we assume the user may be able to look at a short ranked list of answers.

Note that since the automatic trainingset generator is only confident about the best answer, for each question in the automatic evaluation set, we only have one relevant answer.

7.4 Experiment Design

The main goal behind our experiments was to check if it was feasible to answer questions through text mining a knowledge base and whether our kbQA approach would outperform a state of the art baseline. The baseline we primarily want to compare against is the state of the art KL-Divergence retrieval method ($Sim()$) which we use for constraint prediction. Out performing it would ascertain that the kbQA model indeed adds value over baseline retrieval. Other questions critical to success of kbQA, that we are interested in are a) To check if automated trainingset generation was feasible and b) if it was common to find query templates useful in answering multiple questions.

8. RESULTS

8.1 Answer Ranking Performance

Figure 2 compares the 5-fold cross validation performance of our method with the constraint prediction baseline. In each fold, we use $4.6K$ questions for mining templates and estimating parameters. The rest $1.1K$ questions are used for prediction. We observe that kbQA outperforms the state of the art KL-Divergence retrieval method by 39.44% in terms of S@1 and 21.17% in terms of MRR. Both improvements were found to be statistically significant using the Wilcoxon signed rank test with level $a = 0.05$. We also notice that the improvement is greater for S@1 than S@5 suggesting that kbQA is better at pushing the most relevant answer to the top. Overall kbQA succeeded in answering nearly 17% of the answers correctly. A sample answer generated by kbQA in response to a question regarding cramps is shown below. It clearly shows that useful answers for cQA questions do exist in online knowledgebases and our proposed approach can help retrieve them, thereby helping the users.

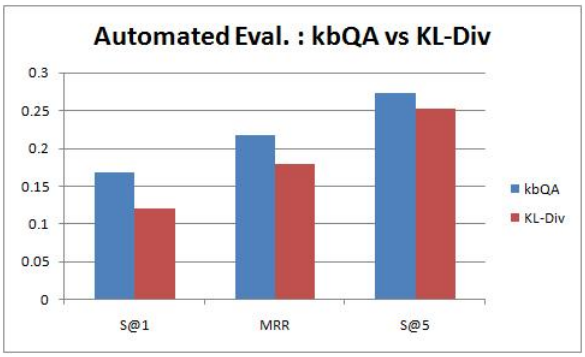

Figure 2: Automated evaluation performance

Question: *How to relieve cramps in feet? I often get cramps in my feet and i have no idea as to why. Does anybody know what could cause them and any way to relieve them when they occur?*

Answer by kbQA: *Skeletal muscles that cramp the most often are the calves, thighs, and arches of the foot.....this kind of cramp is associated with strenuous activity and can be intensely painful.....though skeletal cramps can occur while relaxing..... It may take up to seven days for the muscle to return to a pain-free state..... Nocturnal leg cramps are involuntary muscle contractions that occur in the calves, soles of the feet, or other muscles in the body during the night or (less commonly) while resting.....Potential contributing factors include dehydration, low levels of certain minerals (magnesium, potassium, calcium, and sodium), and reduced blood flow through muscles attendant in prolonged sitting or lying down....Gentle stretching and massage, putting some pressure on the affected leg by walking or standing, or taking a warm bath or shower may help to end the cramp.*

We next analyze the results on the manually judged dataset. The results are shown in Figure 3. We observe that the performance of kbQA is higher than the KL-Divergence method, but lower than the training set generator (TsetGen) which represents a kind of upper bound performance. The kbQA method answers nearly 35% of the questions correctly, suggesting that the automatically generated trainingset was indeed useful for the task.

Finallly Figure 4 compares the automatic evaluation performance of kbQA and KL-Divergence on the 60 questions used for manual evaluation. We assume the results of our training set generator to be the gold standard. kbQA continues to outperform the baseline. We also observe that per-

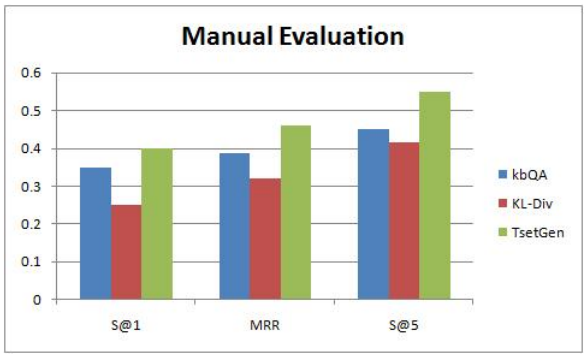

Figure 3: Performance on manually judged set

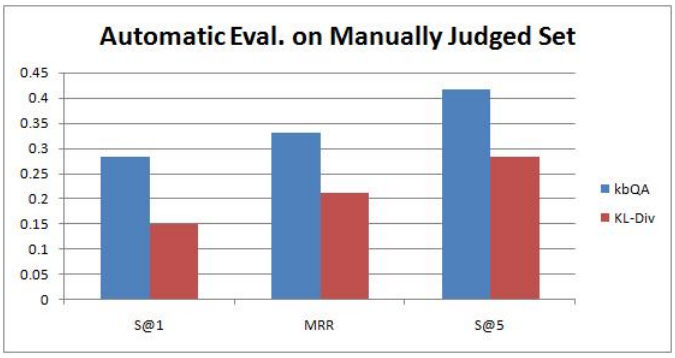

Figure 4: Automatic evaluation on manually judged set

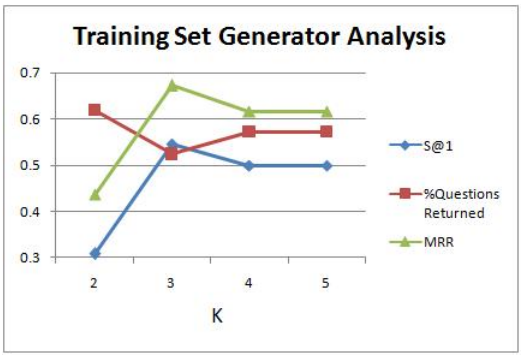

Figure 5: Analysis of training set generator performance on validation set. We choose $K = 3$

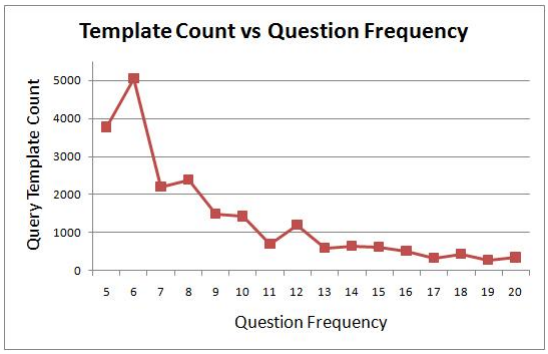

Figure 6: Question frequency analysis

formance is generally lower than the one obtained through manual evaluation because in the automatically generated evaluation set we are unable to generate more than one relevant result per question.

8.2 Training Set Generator Validation

We validate the performance of our automated training set generator on the 21 query validation set. For each question in the validation set, we use our training set generator to find the best answer from the eMedicinehealth knowledge-base and compare it to the manually judged gold standard. Note that many of the questions will get filtered out due to the filtering criteria (see section 5.4). The performance is evaluated only on the questions that are retained. Our goal is to find a parameter setting for K such that we retain a sufficiently high accuracy in terms of S@1 and MRR, without filtering out too many questions. The results are shown in Figure 5.

The figure shows how three different metrics S@1, MRR and Percentage of training questions returned vary with the parameter K. We observe that while $K = 2$ generates the most number of instances (61.9% or 13 out of 21), its accuracy is quite low. On the other hand for values 3 through 5, we retain slightly over 50% of the 21 queries as our training questions. Among these, the accuracy is highest for $K = 3$ which is the value we select for K.

8.3 Analysis of Mined Queries

Our next goal is to analyze whether it is possible to learn query templates that are reusable across questions. We executed our template mining step on all $5.7K$ training questions and plotted two graphs. The first is shown in Figure 6. It plots the question frequency of a query template on the x-axis and the number of query templates that were found with that frequency on the y-axis. The question frequency of a template is the number of questions this template was useful in finding answers to. For example the first data point means that there were nearly 4000 query templates that were useful in answering 5 questions. Naturally, as we increase the question frequency, the number of templates drops. The last data point shows that there were 340 templates that were useful in answering 20 questions. The plot clearly suggests that many templates are useful across questions.

We next need to ensure that the high frequency templates are not concentrated only among a selected few questions. To analyze this, we plot the number of questions that will become unanswerable if we used only query templates above a certain question frequency (see Figure 7). For example the plot shows that if we restrict to query templates with a question frequency of 5 or more, we will be left with no templates capable of answering 142 questions. In general we observe that for most questions there tends to be atleast 1 high question frequency template. Which is why even with threshold of 20, only 10% of the questions become unanswerable.

9. CONCLUSIONS AND FUTURE WORK

We introduced and studied a novel text mining problem, called knowledge-based question answering, where the computational problem is to mine an online semi-structured knowledge base to discover potential answers to a natural

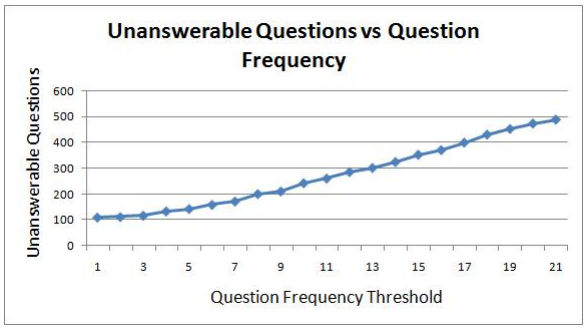

Figure 7: Showing the number of unanswerable questions against question frequency

language question on cQA sites. We proposed a general novel probabilistic framework which generates a set of relevant sql queries and executes them to obtain answers. We presented in detail an instantiation of the general framework for answering questions on cQA sites by leveraging the existing questions and answers as training data. Evaluation has shown that the proposed probabilistic mining approach outperforms a state of the art retrieval method.

Although we only evaluated the our work on healthcare domain, the framework and algorithms are general and can be potentially applicable to other domains as well. Our main future work is to extend our work to additional domains and to refine the different framework components.

10. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Number CNS-1027965.

11. REFERENCES

[1] Trec question answering track. http://trec.nist.gov/data/qamain.html.

[2] S. J. Athenikos and H. Han. Biomedical question answering: A survey. *Computer Methods and Programs in Biomedicine*, 99(1):1–24, 2010.

[3] M. W. Bilotti, J. Elsas, J. Carbonell, and E. Nyberg. Rank learning for factoid question answering with linguistic and semantic constraints. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, CIKM '10, pages 459–468, New York, NY, USA, 2010.

[4] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-newton matrices and their use in limited memory methods. *Math. Program.*, 63(2):129–156, Jan. 1994.

[5] H. Cui, M.-Y. Kan, and T.-S. Chua. Generic soft pattern models for definitional question answering. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '05, pages 384–391, New York, NY, USA, 2005.

[6] Z. Huang, M. Thint, and A. Celikyilmaz. Investigation of question classifier in question answering. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, pages 543–550, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[7] J. Jeon, W. B. Croft, and J. H. Lee. Finding similar questions in large question and answer archives. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 84–90, New York, NY, USA, 2005.

[8] V. Jijkoun and M. de Rijke. Retrieving answers from frequently asked questions pages on the web. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM '05, pages 76–83, New York, NY, USA, 2005.

[9] J. Kim, X. Xue, and W. B. Croft. A probabilistic retrieval model for semistructured data. In *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval*, ECIR '09, pages 228–239, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] J. Y. Kim and W. B. Croft. A field relevance model for structured document retrieval. In *Proceedings of the 34th European conference on Advances in Information Retrieval*, ECIR'12, pages 97–108, Berlin, Heidelberg, 2012. Springer-Verlag.

[11] M. Minock. C-phrase: A system for building robust natural language interfaces to databases. *Data Knowl. Eng.*, 69(3):290–302, Mar. 2010.

[12] A.-M. Popescu, A. Armanasu, O. Etzioni, D. Ko, and A. Yates. Modern natural language interfaces to databases: composing statistical parsing with semantic tractability. In *Proceedings of the 20th international conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[13] S. Robertson, H. Zaragoza, and M. Taylor. Simple bm25 extension to multiple weighted fields. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, CIKM '04, pages 42–49, New York, NY, USA, 2004.

[14] R. Soricut and E. Brill. Automatic question answering using the web: Beyond the factoid. *Inf. Retr.*, 9(2):191–206, Mar. 2006.

[15] V. Tablan, D. Damljanovic, and K. Bontcheva. A natural language query interface to structured information. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pages 361–375, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] X. Xue, J. Jeon, and W. B. Croft. Retrieval models for question and answer archives. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 475–482, New York, NY, USA, 2008.

[17] C. Zhai. Statistical language models for information retrieval a critical review. *Found. Trends Inf. Retr.*, 2(3):137–213, Mar. 2008.

[18] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 403–410, New York, NY, USA, 2001.