Finding Answers in Semi-Structured Data

Joe DiFebo Sean Massung

December 19, 2012

Abstract

Using Wikipedia [5] as a knowledge base is a relatively new branch in the thought process of the Question Answering (QA) domain. Most notably in [1], the team used Wikipedia as an information facet for the famous Watson QA system [2]. We expand upon their ideas as we delve into the Wikipedia structure.

Using our system, we show using additional semi-structured knowledge (such as infoboxes) can improve the accuracy of QA retrieval. This approach would be useful in any domain that has any sort of data table scheme reminiscent of Wikipedia's infoboxes, such as IMDB, Facebook, and Amazon. Each of these also has a "title" for each of its pages that can be used in the search, such as movie, person, or product.

Although extracting infobox information is not a new concept [4], we can combine its utility with a information retrieval techniques to create a novel QA system.

Our results show that this can be a useful feature to add for any question answering service.

1 Introduction

As the world's data grows exponentially, it's hard to keep it organized in a quickly accessible, machine-readable way. Databases excel in fetching information with a known schema, but most world knowledge is unstructured. However, some knowledge can be considered "semi-structured". That is, human-understandable organization exists, but it's not friendly enough for computers to query with predetermined fields.

Modern IR techniques deal with these issues by defining some similarity measure between the query and each indexed document. They return a ranked list of documents that are most similar to the query. This does not, however, deal with answering specific questions: "Who won the world cup in 1998?" might return a document entitled *1998 FIFA World Cup*, and it's up to the user to find the sentence containing the answer on that document.

Question Answering (a subfield of IR), deals with analyzing sentence structure and uses NLP tools to figure out the exact meaning in a question. However, with the abundance of semi-structured data online, being able to filter through a premade ontology would greatly improve the accuracy of a QA system. Less reliance on NLP tools reduces the complexity of the system and alleviates any errors introduced by less than perfect language processing. Additionally, this develops a stronger dependency on inadvertent human structure–which is desired; answers will be presented to human askers.

2 Previous Work

2.1 QA with Wikipedia

As mentioned in the abstract, one of the Watson system's main features was using Wikipedia as a knowledge base. They classified the question into various categories signifying where a Wikipedia article was in relation to the question:

1. *Title in Question*: the Wikipedia article title containing the answer is part of the question
2. *Title in Answer*: the answer to the question is a Wikipedia article title, not mentioned in the question
3. *Title in Neither*: as the name suggests, the article title is in neither the question or answer, and it's doubtful whether the answer can even be found on Wikipedia.

In light of the third point, the authors found that 98% of the TREC QA dataset answers were an article title. Because of this fact, we decided to ignore the third case, and only focus on TQ (title in question) and TA (title in answer). We only trained on TQ and TA questions, giving a relatively accurate classifier due to the two-class classification. Although this sets the system up for failure on the remaining 2% of questions, we rationed the accuracy gained by reducing a class would offset this.

The authors then ran different parts of the system depending on the supposed article location. After

receiving a list of top documents likely to contain the answer, they ran a few candidate generation schemes to extract answers from the documents. We will use this same general approach in our system.

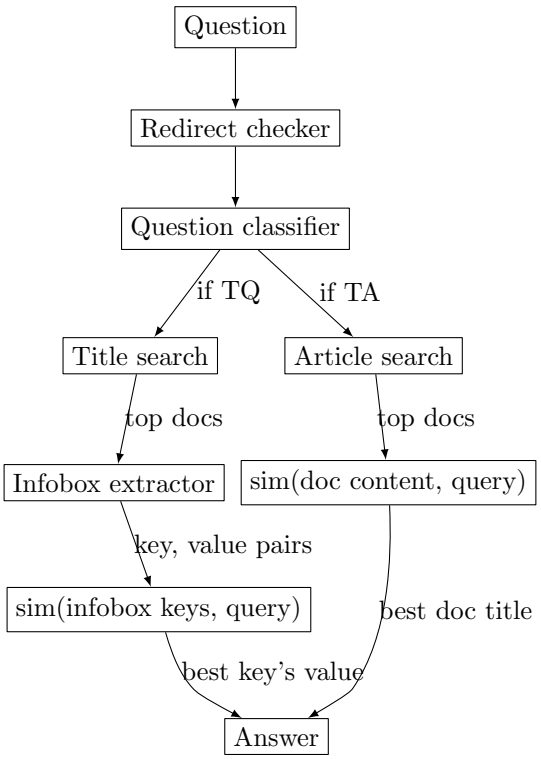
2.2 Extracting Semi-Structured Information

The treatment of Wikipedia's infoboxes is a distinguishing difference between our work and the work in [1]. As described in [4], using tables to represent (attribute, value) pairs is a fairly common practice. There was a wide variety of websites mentioned in [4] from which the authors extracted these pairs, but notably Wikipedia was not one of them.

Wikipedia's wiki templates evaluate to a relatively basic HTML format allowing us to employ a simple heuristic to extract this information, unlike the complicated unsupervised method presented in the above paper. However, if expanding our system to different domains, we would be able to use this paper's work as a pluggable component called upon for various sites.

3 Methodology

Our system flow is kept as simple as possible and is loosely based on the Watson framework. It contains two main indexes: one for article titles and one for article content.



3.1 Redirect Checking

First, before any intensive processing is done, we check to see if the question is similar to an existing Wikipedia redirect. The reasoning for this is best seen by example. Look at the following Wikipedia pages and their redirects:

- First president of the United States → George Washington
- First man on the moon → Neil Armstrong
- Third planet from the sun → Earth

Using this observation, if a user asks "What is the third planet from the sun?", the answer should clearly be "Earth". We formalize this intuition as follows:

- Check if the question begins with a combination of question words such as "who is", "when is", "what are", etc.
- If so, stem the question and remove stopwords.
- Check to see if the cleaned question matches any redirects (which have also been cleaned in the same way)
- If a matching redirect is found, return the page title that it redirects to as the answer.
- Otherwise, continue on to the main system.

In our example, we would stem "What is the third planet from the sun" to "third planet sun", which is an exact match with "Third planet from the sun" after stemming and stopword removal. Hence, we would return "Earth" as the answer.

3.2 Question Classifier

The first step in the main system is to classify the question as either TQ or TA. We used Learning-Based Java [6] as our machine learning framework, training it on annotated TREC '04, '05, '06 QA datasets. We used Stanford Named Entity Recognition [3] to replace named entities with their tags. We then stemmed and removed stopwords from these sentences and used a simple unigram language model to generate the features for LBJ. We were able to get 85% accuracy.

While 85% may seem low for two-class classification, it is hard to quantify what separates a TQ from a TA. We suspected a lack of named entities indicated a TA, but this will not be true for each case. Question words such as "who" and "what" are equally useless since they appear in almost every question. We were therefore justified in removing stop words

(which would contain all the question words) as well as stemming.

We used 272 questions for training and 100 questions for testing. Manually labeling the questions was a bit time consuming, and added to the sparsity of examples. Given more time, we could have labeled much more, most likely resulting in a higher accuracy.

3.3 TQ Path

Once we decide that the answer to the question is contained in an article found in the question, we search the title index for the question. This returns a ranked list of documents that match the question.

Say a user searched for "When was George Washington born?". This will cause the system to search the title index for "georg washington born", returning the pages

1. Washington_Georges
2. George_Washington
3. George_Washington_Manypenny
4. George_Washington_Hosmer
5. George_Washington_Carroll
6. ...

Then, the infobox extractor is run on the top k result pages, extracting infobox information and accumulating (key, value) pairs. This works by locating table tags and extracting their content.

The Jaccard similarity is calculated between the query and key for each pair. The pair with the highest similarity is then returned, with the answer to the question being the value of the pair.

	article	field	value
1	George Washington	Born	2/22/1732
2	George Washington Morse	Born	6/24/1816
3	George Washington Foster	Born	12/1866
4	George Washington Watts	Born	1/18/1851
5	...		

In this case, the Jaccard similarity between the processed query "georg washington born" and the article + field for an infobox entry is 1.0 since they are exactly the same. Hence 2/22/1732 is returned.

3.4 TA Path

On the other hand, if we think the answer to the question is a Wikipedia article title not mentioned in the question, we apply a different approach. We search the full-text article index for the top documents matching the question.

Say we search for "Who wrote the book *Planning Algorithms* at the University of Illinois", and it's classified as TA. We receive the results

1. Steven_M._LaValle
2. David_E._Goldberg
3. Carl_W._Condit
4. Robert_J._Vanderbei
5. Neil_Chriss
6. ...

The matching text similarity from the first article matches the query the best. Thus, "Steven M. LaValle" is returned.

4 Evaluation

Experimental results from our system are divided into two main phases: document retrieval (picking documents that contain answers), and answer selection (choosing which information from the documents is the actual answer).

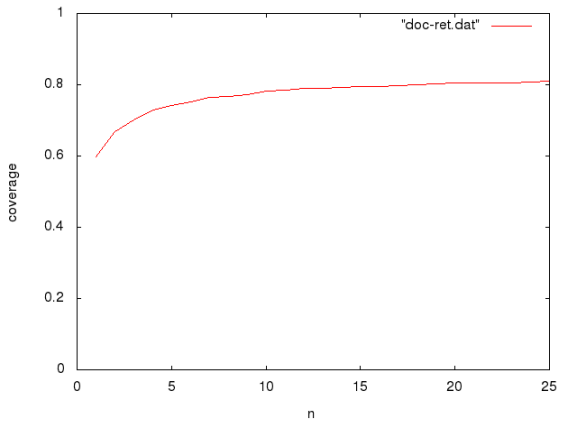
4.1 Document Retrieval

For document retrieval evaluation, we use the measures suggested by [7]. In short, we will use the *coverage* statistic as opposed to the typical IR metrics precision and recall, since the effectiveness of a QA system selecting candidate documents is different than a typical search engine:

$$coverage(Q, D, n) = \frac{|\{q \in Q | R_{D,q,n} \cap A_{D,q} \neq \emptyset\}|}{|Q|}$$

where Q is the question set, D is the document collection, $A_{D,q} \subseteq D$ that contains answers to the question $q \in Q$, and $R_{D,q,n} \subseteq D$ is the n top-ranked documents in D retrieved for query q.

We vary the value of n, showing how the system performs after retrieving various numbers of top documents for 1000 TREC queries.

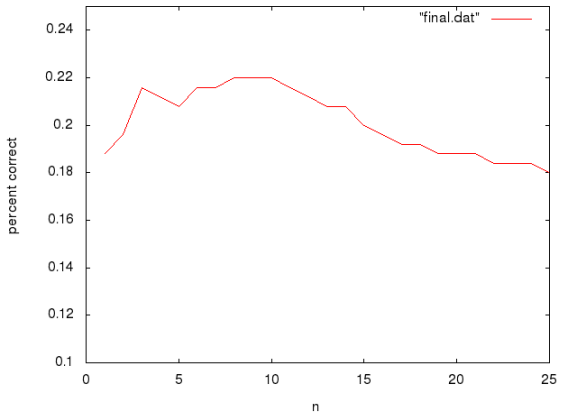


Clearly, increasing n brings better accuracy, but having more candidates for the second stage makes answer selection more difficult.

4.2 Answer Selection

How well does the system match the query to the appropriate infobox? We use a similar methodology presented in the previous section. Since the infoboxes viewed are based on which top documents are returned, returning more documents will increase the number of potential infoboxes. However, the accuracy is not proportional to the number of infoboxes seen. In fact, comparing a large number of infoboxes creates data whose noise is proportional to the number of documents retrieved.

To evaluate this component, we run the same queries again, varying n from 1 to 25. We let the system select the best infobox match from the current selection of n documents. We increment a tally for each n every time the answer is correct. This should indicate a preferential range for n.



We see that there is a peak around the $n \in [7, 10]$ range.

4.3 End-to-end System

We found that while increasing the number of top documents returned increased the document coverage, it decreased the infobox selection accuracy since more and more noise was being added. This allowed a small number of documents (usually containing the answer) to be returned while narrowing the search for the correct infobox. Setting $n = 9$ allowed a maximum of 22% accuracy.

5 Conclusions

In the end, we are very happy with our results. Question Answering is known to be a very hard task, and while we come nowhere close to the performance of systems like Watson (59.3%), our 22% is quite remarkable given that the project was completed by two students in less than a semester. While the system is not state-of-the-art, many of the techniques can be improved upon and possibly incorporated into other systems to improve them as well. As a whole, this project was a success.

6 Future Work

This system can be improved in several ways. The system incorporates a pipeline design, where the output from one part is used as the input to the next part. As a result, even if every subsystem performs relatively well, the performance of the overall system may suffer greatly from cascading errors. Thus, an obvious improvement to the system would be to improve on every subsystem individually.

Both the "title search" and "answer search" parts work relatively well, but they can easily be improved by making the algorithms more specialized to this domain. This could be as simple as replacing the BM25 formula with a better one, or it could potentially incorporate other information as well.

The infobox extractor is, at this point, a very simple tool. In order to match a question with an infobox field, a crude form of query expansion is used. If the query expansion technique is improved, we will certainly find more potentially correct fields. Also, having better similarity criteria will result in picking fewer incorrect fields, allowing us to find the correct field more often. Improving this would improve the overall system by a large amount.

In addition to using infoboxes to obtain semi-structured data, it would also be useful to be able to parse other data, such as large tables (which map pairs of keys to a value), or even bulleted lists, or ranked lists, which would be able to identify at-

tributes such as the "tallest" or "heaviest". This information is easily accessible to a human, but unless special consideration is taken, very difficult for an automated program to parse.

The largest improvement can be made by incorporating some type of question classification (QC) and document classification techniques. If the system can detect that the question is, say, asking for a name, and the system can reliably determine if a phrase is a name or not, the results could be narrowed down and many incorrect answers could be eliminated. Existing QC systems could probably be used without any modification. The system would also need some sort of potential answer classification to make use of this, and such a system would probably perform best if it were made specifically for Wikipedia. Wikipedia metadata might be useful to this end as well. This would most likely make the largest impact on results.

References

[1] Jennifer Chu-Carroll and James Fan. Leveraging wikipedia characteristics for search and candidate generation in question answering. In *AAAI*, 2011.

[2] David A. Ferrucci, Eric W. Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John M. Prager, Nico Schlaefer, and Christopher A. Welty. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.

[3] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 363–370, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[4] Yingqin Gu, Jun Yan, Hongyan Liu, Jun He, Lei Ji, Ning Liu, and Zheng Chen. Extract knowledge from semi-structured websites for search task simplification. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, CIKM '11, pages 1883–1888, New York, NY, USA, 2011. ACM.

[5] Meta. Data dumps — meta, discussion about wikimedia projects, 2012. [Online; accessed 15-December-2012].

[6] Nick Rizzolo and Dan Roth. Learning based java for rapid development of nlp systems. In *LREC'10*, pages –1–1, 2010.

[7] Ian Roberts and Robert Gaizauskas. Evaluating passage retrieval approaches for question answering. In *In Proceedings of 26th European Conference on Information Retrieval*, pages 72–84, 2003.