TREC Factoid Question Answering: An Exploration of Redundancy-Based Question Answering, Web-Based Boosting and Classification Techniques

Marie-Renée Arend
University of Washington
rcarend@uw.edu

Josh Cason
University of Washington
casonj@uw.edu

Anthony Gentile
University of Washington
agentile@uw.edu

Abstract

This paper outlines the evolution of a redundancy-based factoid question answering system with integrated web-based boosting. The main content is focused on the final version of the system which contains an implementation of an SVM classifier for both question and answer classification tasks. Of the changes we made to our question-answering system, our results clearly show that the treatment of factoid question answering as a classification task was our most effective approach. In addition to discussing our final system and its various modules, we also present some possible avenues for further improvement.

1 Introduction

When examining the problem of question answering, there are three dimensions to the problem to take into account: analysis of the question, the answer search, and the selection and presentation of the answer. To study the effect of differing approaches on these dimensions to the problem of factoid question answering, we built three versions of a question-answering (QA) system, each building upon the previous version and incorporating different techniques and strategies based on the results of previous research into this field.

The first version of our QA system essentially served as a baseline to ensure that our pipeline was functional. We prepared factoid queries from TREC 2004 and 2005 for web search using the most basic shallow processing techniques and used the Bing search engine to retrieve the top 100 candidates for each query. Next, we filtered and re-ranked the ngrams for each query based on the method and formula outlined in Lin's work (2007). Finally, our top 20 answer candidates were projected onto the AQUAINT corpus, and PyLucene, a Python wrapper of the core Lucene project, was used to retrieve the most relevant documents for these answer candidates.

In the second version of our QA system, we expanded upon our baseline approach by testing the effect of query expansion using word sense disambiguation to add synonyms, and adding some answer pattern detection to our pipeline. We found that our synonym-based query expansion process hurt our results, though the answer pattern detection helped. To exploit the answer pattern detection, we first applied a weight to our initial web results based on their ranking from Bing, and then we searched for answer patterns based on our query. If the answer pattern was found, we gave that result an additional weight boost that was used in our ngram filtering and re-ranking module.

The final version of our QA system incorporated some major changes to our first two versions, such as the implementation of a classifier system and caching. The discussion of our final system will be the main focus of this paper.

2 Question Classification

As mentioned in the introduction, the most important change to our system that we incorporated for the final version was the implementation of a classifier. From scikit-learn's Python package (Pedregosa et al., 2011), we built an SVM classifier with a radial basis function kernel and utilizing chi-squared feature selection.

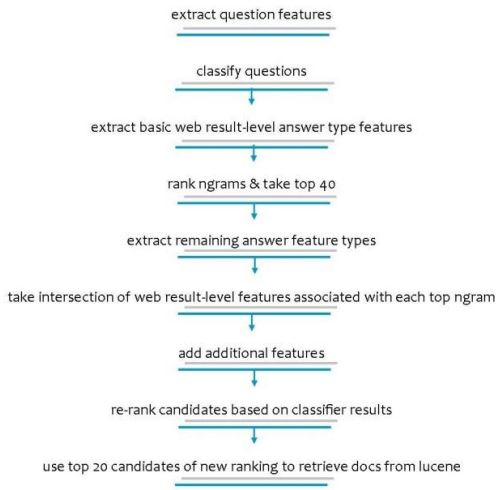

Fig. 1: New System Pipeline

2.1 Question Feature Extraction

To train our classifier, we experimented with a number of different features and *k* values, which limit the number of features to *k* best according to the chi squared algorithm. The features we chose are loosely based on Li and Roth's (2007) syntactic features, and include stemmed ngrams, weights for temporal, location or numerical question words, POS-tagged tokens from the question and target with stopwords removed, head NP and VP chunks, and the question word(s). The training labels were drawn from Li and Roth's publicly available labeled factoid questions.

Despite performing around sixty tests with various features and *k* values, we found that the overall accuracy of our classifier did not vary much. Our lowest accuracy was around 78% and our highest was just over 80%. In our final system, we used the optimal combination despite the only small increase it afforded.

Though we neither tested nor included semantic features from the questions in the final version of our system, the lack of a significant increase in classifier accuracy from the addition of syntactic features suggests that incorporating different types of features, such as semantic ones, could be a worthwhile avenue to pursue in future work.

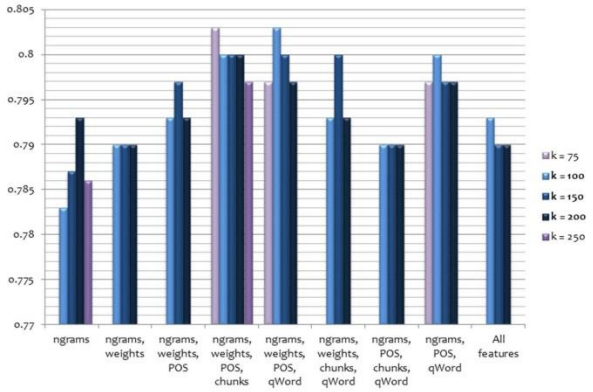

Fig. 2: Some Feature Combinations and Accuracy Results

3 Web Answer Classification

When web results are retrieved, an initial pass of logic is done to apply a weight to the web results. This logic includes consideration for the initial ranking of the results as well as detection for location, temporal, and numerical contexts. Later in the pipeline we use classified question types to further filter and weigh our corresponding web results.

3.1 Initial Ranking and Context Filtering

When web results are retrieved, we apply an initial weighting to the results that is based on the ranking of the result set and on particular context filters. First, all web results start with a weight of 1.0. We then adjust the weights for each result based on its initial ranking. The goal is to have lowest ranking result in the set have a weight of 0.1. To do this we iterate over the result set and decrement the weight by $\frac{0.9}{N} \times (rank - 1)$, where N is the size of the result set.

After this initial weighting is complete, we then look for weighting that we can apply based on question context. If a question starts with 'When', 'What Year', or 'What Month', we can conclude that the answer is going to be some kind of temporal pattern. Likewise, when a question starts with 'Where', 'In what country', 'In what state', 'What country', or 'What State' we can conclude we are looking for an answer in a location pattern. Lastly, when we see a question start with 'How many', 'How much', 'At what age', or 'How old' then we can conclude that we will be looking for

an answer with a numerical pattern. So when a question like 'In what state was he born?' is seen, the location context detection is triggered, and if a web result has a state pattern in it, such as 'North Carolina', then the web result will receive a weight boost of 20%.

3.2 Basic Web Result Level Answer Feature Extraction

The question classification type is used to trigger a set of regular expression detection for the subclasses of the given type. For example, if a question is classified as being 'ENTY', we then attempt to detect the subclasses (sports, religion, colors, etc.) in the web results. When these subclasses are detected, corresponding features are added back to the classifier.

Web result feature generation was restricted to patterns that depended on punctuation or other feature of the text that would be removed when extracting ngram answer candidates. The choice of patterns to search for is limited to those related to label produced in the question classification module.

3.3 N-gram Generation and Filtering

The next module generates ngrams from the web results and filters them following some of the techniques mentioned in Lin (2007). Results are gleaned from the Google Search API, tokenized, and, from the title and snippet, unigrams, bigrams, trigrams, and quadrigrams are counted. Multi-token ngrams are then filtered according to the following criteria. An ngram is removed from the ranking if the ngram begins or ends with a stopword, contains words from the question, or has other problems like if there is punctuation leftover from tokenizing.

Ngrams are then reranked with the following formula.

$$S_c = S_c + \sum_{t \in c} S_t$$

The score, S_c, of a candidate answer in increased by the sum of the scores of its unigram tokens. This combats the tendency to reward shorter answers due to their natural higher frequency.

3.4 Extraction of Remaining Answer Type Features

Once ngrams are extracted and ranked, features are again generated. Only the top 40 candidates are retained. Patterns which do not rely on punctuation are used to generate features from them. The search is again limited to those patterns related to the question classification label.

Once these answer candidate level features are generated, other features such as top web result rank, the question classification label itself, and the features used in question classification are added. In order to incorporate web result level features, the intersection of all web result level features associated with each top 40 ngram, *n*, is added:

$$\bigcap_{w \in W} f(n, w)$$

where *f* returns the set of features for web result *w*, if *n* appeared there.

3.5 Answer Feature Re-Ranking

In the training stage, all features collected so far are then labeled yes or no for whether the answer candidate contains an acceptable answer. During testing the classifier is called on to predict a ‘yes’ or ‘no’ judgment. However, in order to re-rank the 40 candidates, the classifier generates the probability that a candidate is a ‘yes’ answer as well. Answer candidates are then re-ranked according to how high its probability is supposed to be. Lastly, the top twenty of the re-ranked candidates are selected as our final results.

4 Document Retrieval

When we search the AQUAINT index, we strip Lucene keywords out such as "+', '||', and ':'. Then we do a simple wildcard search that turns a given query into query* OR query. This essentially means that a search for the term 'cook' would turn into a Lucene query of "cook* or cook" which would not only match the word cook, but also cookies, cookbook and so on.

4.1 Query Modification for Retrieval

Often when searching the web or a document collection, it is difficult to get the correct results

because of mismatch between the wording of a user's question and the wording of potential answers. To alleviate this to some degree, we incorporate a query processing unit. In our baseline iteration of our system, this module only performed the most basic processing tasks, such as stemming, removing stopwords and adding the target if not already present in the question.

In the second version of our system, we attempted to use our own version of the modified Lesk algorithm as described by Huang et al. (2008). Instead of just disambiguating the head word in the query, we chose to disambiguate every word in the query that can be found in WordNet. Once the best sense for these words was generated by the algorithm, we wanted to use those best senses to find synonyms to add to the original query.

The word sense disambiguation and synonym-generating code did not work as intended for that version of our QA system. Perhaps the most troublesome problem with the code's results was that the words were not properly disambiguated, leading to a large number of both relevant and irrelevant synonyms being added to the reformed query. This created too much noise in the query and, as a result, many questions that were answered in our baseline system were not answered the second time around. A second major problem is that the disambiguation algorithm takes too much time to process each query. As such, we decided to abandon the use of synonyms and word sense disambiguation for the final version of our system.

For the final version of our query processing unit, we went back to our baseline version which incorporates only shallow processing techniques. Since these techniques were enough to allow us to retrieve relevant web results, we saw no reason to further slow our system's running time by adding in other, extraneous modifications.

4.2 Answer Projection

The top twenty answer candidates are chosen and are used to project the answer onto the AQUAINT corpus (Graff, 2002). AQUAINT is used for evaluating QA systems, and it is necessary to find a document in it to support our web retrieved answer. We use a standard IR system Lucene (Hatcher et al., 2004) in order to retrieve the most relevant documents.

5 Results

	D2	D3	D4 (2006)	D4 (2007)
Lenient MRR	0.0455	0.0475	0.1938	0.1909
Strict MRR	0.0039	0.0070	0.0479	0.0656

Table 1: MRR scores

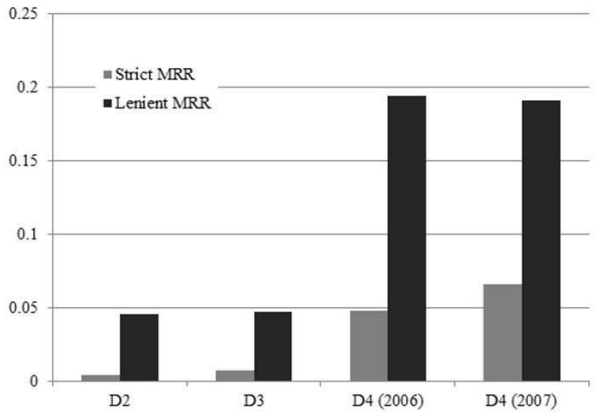

Fig. 3: MRR scores

The final version of our QA system, labeled D4 in the table above, was trained on TREC 2004 and 2005 questions, tuned using 2006 questions, and, finally, objectively evaluated using TREC 2007 questions. No tuning was performed for the 2007 question set.

Our results are encouraging in that as we explored the process of question answering by changing and improving our approach, our lenient and strict MRR scores improved. The score improvement from our first version to our second is most likely due to the addition of weights for the web result ranking and answer pattern detection, since this was the most significant change we made at that point over our baseline system. The score improvement from our second to our final version can best be explained by the implementation of our SVM classifier. Previous research has shown the classifier approach to factoid question answering is effective, and its implementation necessitated many changes to our pipeline. Had those changes

not been efficacious, we would not have seen such a boost in our scores.

6 Other Implementation Details

Tokenizing, stemming, POS tagging, a WordNet (Miller et al., 1990) interface, and a stopwords list in these modules are supported by the open-source Natural Language Toolkit (Bird, et al., 2009).

7 Conclusion and Future Directions

This paper presents an approach to the task of factoid question answering from a machine learning perspective. In all, we created three versions of our QA system, each improving upon the other, but it is clear that the machine learning perspective, implemented in the form of an SVM classifier, was the most effective. We are lucky to have had access to a decent amount of training and evaluation data in the form of previous years' TREC questions and accurate labels from Li and Roth's publicly available research.

Our experiments with various syntactic features and their lack of effect on the overall accuracy of our classifier over basic ngram features showed that the incorporation of semantic features could be a worthwhile pursuit for further improvement to our system. In addition, the gain in both strict and lenient MRR scores in the second version of our system that could best be attributed to the incorporation of answer pattern detection reveals another avenue for further experimentation. Finally, we learned from our classmates' work that even seemingly random changes, such as using the log of the *idf*, can lead to surprisingly strong score boosts, so truly, there are numerous small changes we could try making in order to improve our results, in addition to large changes such as semantic feature extraction.

References

Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. O'Reilly Media.

Graff, D. (Ed.). (2002). *The AQUAINT corpus of English news text*. Linguistic Data Consortium.

Hatcher, E., Gospodnetic, O., & McCandless, M. (2004). Lucene in action.

Li, X., & Roth, D. (2005). Learning question classifiers: The role of semantic information. *Natural Language Engineering*, *1*(1), Retrieved from http://l2r.cs.uiuc.edu/~danr/Papers/LiRo05a.pdf

Lin, J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems (TOIS)*,*25*(2), 6.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, *12*, 2825-2830.

Resnik, Philip. (1995). Disambiguating Noun Groupings with Respect to WordNet Senses. *Third Workshop on Very Large Corpora*. Retrieved from http://acl.ldc.upenn.edu/W/W95/W95-0105.pdf