Model Tree Learning for Query Term Weighting in Question Answering

Christof Monz

Department of Computer Science
Queen Mary, University of London
Mile End Road, London E1 4NS, United Kingdom
E-mail: `christof@dcs.qmul.ac.uk`
URL: `www.dcs.qmul.ac.uk/~christof`

Abstract. Question answering systems rely on retrieval components to identify documents that contain an answer to a user's question. The formulation of queries that are used for retrieving those documents has a strong impact on the effectiveness of the retrieval component. Here, we focus on predicting the importance of terms from the original question. We use model tree machine learning techniques in order to assign weights to query terms according to their usefulness for identifying documents that contain an answer. Incorporating the learned weights into a state-of-the-art retrieval system results in statistically significant improvements.

1 Introduction

Current question answering systems rely on document retrieval as a means of identifying documents which are likely to contain an answer to a user's question. The documents returned by the retrieval engine are then further analyzed by computationally more expensive techniques to identify an answer to a given question.

The effectiveness of the retrieval component is critical for the performance of a question answering system: If the retrieval system fails to find any relevant documents for a question, further processing steps to find an answer will inevitably fail as well.

In particular, the way queries are formulated has a strong impact on retrieval effectiveness. Boolean retrieval is especially sensitive to query formulation. Using all content words from the question for the query can steer the retrieval process in a wrong direction. For example, consider the question *What is the abbreviation for the London stock exchange?*. It seems natural to include the word *abbreviation* in a boolean query like:

(1) `abbreviation AND london AND stock AND exchange`

However, most documents that contain an answer to this question express it in the form of '. . . London Stock Exchange (LSE). . . ,' not using the term *abbreviation* or one of its morphological variants at all. Hence, in boolean retrieval, a query such as (1) might be too strict and retrieve no documents at all.

Vector-space retrieval is less strict with respect to presence or absence of query terms in documents, but a similar problem arises. Even vector-space retrieval approaches will prefer documents containing the term *abbreviation* over those that do not contain it, although, as discussed above, most documents providing an answer to the question above do not contain it. The reason is that the term *abbreviation* receives a high term weight, because it is much less frequent in the collection than the other terms and is therefore considered a well-discriminating term for this query.

Our approach tries to predict the importance of a term for a given question by applying machine learning techniques. The learned importance weights are then used to improve the retrieval engine. Learning query term weights is appealing in the context of question answering because a user's information need is expressed as a well-formed sentence as opposed to sets of keywords, used in regular information retrieval.

2 Related Work

Previous work on query formulation for question answering has mainly been done for Web question answering. Brill et al. [2] focus on formulating query strings that approximate the way an answer is likely to be expressed. This involves automatically mapping the syntax of an interrogative to the syntax of a declarative sentence. Documents are required to either match one of the strings or the boolean query. However, they do not address the issue of term weighting.

Paşca [11] does address the issue of term selection and term relevance. His work is closely related to the work presented in this paper. For query formulation, he distinguishes between three types of terms: high-relevance, medium-relevance, and low-relevance query terms. Deciding which class a given term belongs to is based on a number of rules, some of which are also integrated in our approach.

Although machine learning techniques have been used before to find answer extraction patterns, see, e.g., [8, 14], they have not been applied before to query formulation in the context of question answering. On the other hand, machine learning has been applied to query formulation in the context of ad hoc retrieval. Cooper et al. [4] use logistic regression to assign weights to matching clues, such as the number of times a query term occurs in the query, the number of times a query term occurs in a document, the idf score of a matching term, and the number of distinct terms common to both query and document. [3] did apply machine learning techniques for selecting query terms, but it was done in the context of relevance feedback retrieval.

The work by Mayfield and McNamee [9] and Agichtein et al. [1] is to some extent complementary to our work as it expands retrieval queries with terms or phrases that are likely to be found in the context of phrases which are of the expected answer type. While the approach in [9] is based on simple co-occurrence statistics, [1] also incorporate the BM25 retrieval weighting scheme to assign weights to the expansion phrases.

3 Optimal Query Term Selection

In this section we estimate the effect query formulation, in the form of term selection, can have on retrieval performance, by using the TREC-9, TREC-10, and TREC-11 data sets consisting of 500 different questions each and the AQUAINT document collection.

For evaluating the retrieval effectiveness, we used NIST's judgment files. Each file indicates for each submitted answer-document-id pair, whether the answer is correct.

Questions with no correct answer were disregarded. The documents that are known to contain a correct answer form the gold standard for our evaluation, which contains 480, 433, and 455 questions for TREC-9, TREC-10, and TREC-11, respectively.[1]

In order to compute the optimal term selection for each question, we compare all possible ways of selecting terms from a question. That is, given a question q in which the set of terms T occurs, we consider all possible subsets of T, and evaluate the respective performances. More formally, the set of term selection variants (*tsv*) is defined as $tsv(q) = POW(T) - \{\emptyset\}$. For each question in the three data sets, we determined the query variant with the highest average precision. Table 1 shows the performance gains that can be achieved by using an oracle to pick the optimal query variant.

	TREC-9		TREC-10		TREC-11	
a@n	Lnu.ltc	opt	Lnu.ltc	opt	Lnu.ltc	opt
a@5	0.700	0.823 (+17.6%)$^\blacktriangle$	0.649	0.749 (+15.4%)$^\blacktriangle$	0.523	0.690 (+31.9%)$^\blacktriangle$
a@10	0.785	0.890 (+13.4%)$^\blacktriangle$	0.734	0.815 (+11.0%)$^\blacktriangle$	0.626	0.767 (+22.5%)$^\blacktriangle$
a@20	0.845	0.921 (+9.0%)$^\blacktriangle$	0.801	0.887 (+10.7%)$^\blacktriangle$	0.705	0.824 (+16.9%)$^\blacktriangle$
a@50	0.914	0.956 (+4.6%)$^\blacktriangle$	0.875	0.924 (+5.6%)$^\blacktriangle$	0.795	0.881 (+10.8%)$^\blacktriangle$

Table 1. Comparison of the a@n scores of optimal retrieval queries to baseline runs.

In the context of question answering, it is common to measure the effectiveness of a retrieval system in terms of answer-at-n (a@n) which is the percentage of questions for which the system returned at least one document containing an answer in the top-n ranked documents. For our baseline we did not use blind relevance feedback, as Monz [10] has shown that simple Lnu.ltc weighting with stemming outperforms approaches using blind feedback for question answering.

As one could expect, query formulation has a significant impact on the overall performance of a retrieval system, even if query formulation is just based on term selection without expanding the queries with semantically related terms. This comparison shows that much can be gained from better query formulation, but, of course the problem of identifying an optimal query without having any relevance assessments remains open. In the remainder we explore ways to solve this issue.

Two retrieval methods a and b are compared by one-tailed statistical significance testing, using the bootstrap method [5]. Improvements at a confidence level of 95% are marked with “$^\vartriangle$” and at a confidence level of 99% with “$^\blacktriangle$”.

4 Computing Query Term Weights

Our approach is to use the different query variants of a question to distinguish between terms that help retrieve relevant documents, and terms that harm the retrieval effectiveness for that particular question.

[1] The original TREC-9 data set contains 243 questions that are re-formulations of the questions in the main TREC-9 set. For our experiments, these variants were disregarded.

In the previous section, we considered only one single best-performing query variant, but often there are several almost equally well-performing query variants. Looking at the ranked query variants, reveals that some terms occur more frequently in higher-ranked variants than other terms.

An analysis of the distribution of query terms over the ranked query variants allows one to assign a weight to each query term: If a term occurs mainly in query variants that have a high average precision it should receive a high weight, whereas a term that occurs mainly in query variants that have a low average precision should receive a low weight. Thus, the weight of a query term depends on two factors: The average precisions of the query variants in which the term occurs (its presence weight: $w_+(t)$), and the average precisions of the query variants in which the term does not occur (its absence weight: $w_-(t)$). Presence and absence weights are normalized by the sum of the average precisions of all query variants, so the weights will range between 0 and 1.

Given a question q and all its query variants $tsv(q)$, the *presence weight* of term t ($w_+(t)$) is computed as:

$$w_+(t) = \frac{\sum\limits_{q' \in tsv(q) \wedge t \in q'} \text{avg\_prec}(q')}{\sum\limits_{q' \in tsv(q)} \text{avg\_prec}(q')} \tag{2}$$

The *absence weight* of term t ($w_-(t)$) is computed as:

$$w_-(t) = \frac{\sum\limits_{q' \in tsv(q) \wedge t \notin q'} \text{avg\_prec}(q')}{\sum\limits_{q' \in tsv(q)} \text{avg\_prec}(q')} \tag{3}$$

The presence and absence weights of a term t, can be combined into a single weight by subtracting the absence weight from the presence weight, which we call the *gain* of term t: $gain(t) = w_+(t) - w_-(t)$. If a query term has a positive gain it should be included in the query, but excluded if its gain is negative.

This approach of computing term weights assumes that terms occur independently of each other. This assumption does not hold in practice, but it is commonly used in information retrieval and allows us to simplify the computation of term weights.

5 Representing Terms by Feature Sets

In the previous section, the computation of the term weights was based on the distribution of the terms themselves over the query variants. This is problematic for two reasons. First, the same term can have a high gain in one query, and a low gain in another. Second, if the learning algorithm is based on the surface terms themselves, it cannot assign weights to terms that did not occur in the training data. The first problem is a direct consequence of the term independence assumption. It could be solved by conditioning the weight of a term on a number of terms that also occur in the question, but then data sparseness becomes even more of an issue.

One way to address both problems is to represent terms and their contexts in a more abstract manner. Here, we use a set of features that represent certain characteristics of a term and its role in a question. The list of features contains information about the term's part-of-speech, whether it semantically includes other terms in the question, the type of question it occurs in, etc. As mentioned above, some of the features capture aspects inherent to a term, such as part-of-speech, while others capture contextual aspects, such as semantic inclusion. Table 2 lists all features. The features *question focus*, *superlative*, *quoted*, *number of leaves*, *modified noun*, and *person name* are based on [11].

Feature	Value	Feature	Value
part-of-speech	Penn Treebank part-of-speech tag	location	Whether the word is part of a location name
question focus	Whether the word is part of the question focus	abbreviation	Whether the word is an abbreviation
superlative	Whether the question contains a superlative adjective	upper case	Whether the word starts with an uppercase letter
question class	A fixed list of question classes	classif. word	Whether the word was used to classify the question
multpl. occurr.	Whether the word occurs more than one in the question	person name	What part of a person's name is the word, if applicable
quoted	Whether the word occurs between quotation marks	honorific	Whether the word is a honorific term (e.g., Dr.)
modified noun	Whether the word is a noun that is preceded (modified) by another noun	no. edges	The number of edges pointing to a word in the dependency parse graph of the question
term ratio	$1/m$, where m is the number of unique terms in the question	hypernym	Whether the word is a hypernym of another question word
no. leaves	The number of hyponyms in WordNet that do not have any further hyponyms	relative idf	The relative idf compared to the other words in the question

Table 2. List of features for question words.

Most of the features in Table 2 are fairly general and self-explanatory, but some are more specific and do require some further explanation.

The *focus* of a question is a phrase describing a semantic type of which the answer is an instance. For example, in the question *In what country did croquet originate?*, the focus is *country*. The answer to this question, which is *France*, is an instance of *country*. Whether a word is part of the question focus affects query formulation, because many documents containing an answer to the question do not explicate the instance relation.

Classifying words help determine the type of a question. E.g., in the question *What province is Calgary located in?*, the word *located* indicates that the question is a location question. However, words that are good indicators for question classification, are infrequent in answers as expressed in documents. For instance, it is rather unlikely that the word *located* is used in a declarative sentence answering this question.

The *number of incoming edges* refers to the dependency parse graph of the question which is generated by MINIPAR [7]. If a word has a larger number of incoming edges, several words in the question are in a modifier or argument relationship with this word, and therefore it is more likely to play a central role in the question. For each word in any question, the features listed in Table 2 are extracted automatically, by using off-the-shelf tools, including a POS tagger, dependency parser, and named-entity recognizer.

6 Learning Term Weights

Instead of using some heuristics for predicting the query term importance we apply machine learning techniques to assign a weight to each term in the question, where the actual query that is used for retrieval will include these weights. The input for the learning algorithm is the set of feature vectors as described in the previous section, and the classes are the terms' gains as described in Section 4.

For the purpose of learning term weights, the machine learning algorithm should learn to predict the degree of the query term's usefulness for query formulation. Decision trees, naive Bayes, and linear regression, all allow for interval classification and generate transparent classification rules.

Naive Bayes classification is known to be well-performing for nominal classification, but performs badly for interval classification [6].

The best-known algorithm for decision tree learning is Quinlan's C4.5 [13], but C4.5 cannot deal with cases where the class to be learned is not a label, but a real number. M5 [12], on the other hand, which is an extension of C4.5, does allow for this type of continuous classification, also referred to as regression.

The M5 algorithm builds model trees combining conventional decision tree learning with the possibility of linear regression models at the leaves of the tree. The resulting representation is transparent because the decision structure is clear and the regression models are normally easily interpretable. The idea of model trees is largely based on the concept of regression trees. The advantage of M5 is that model trees are generally much smaller than regression trees and have proved to be more accurate in a number of tasks; see [12]. M5 is suited for learning query term weights because it combines decision tree learning with linear regression, which allows it to consider dependencies between features. The learning algorithm used here, is M5′ [15], which is a reconstruction of Quinlan's M5 algorithm. M5′ is part of the WEKA ML package [16].

7 Results

For our experiments, the weight of a query term depends on two factors: The frequency of a term in a document, and the collection frequency, i.e., the number of documents containing that term. If we want to integrate the learned term weights, as described above, the computation of the retrieval status value (*RSV*) has to be adapted appropriately. We use the learned query term weights in combination with the original retrieval status value that resulted from computing the similarity between a query q and a document d according to the Lnu.ltc weighting scheme, which results in the new retrieval status value: RSV_L, which is defined as follows:

$$RSV_L(q,d) = \sum_{t \in q \cap d} RSV(q,d) \cdot \mathrm{weight}(fr(t,q)) \cdot \mathrm{idf}(t)$$

Here, $fr(t,q)$ is the feature representation of term t in query q, and weight($fr(t,q)$) is the learned weight, which results from applying the M5′ model tree to t's feature vector. $RSV(q,d)$ is the document similarity according to the Lnu.ltc weighting scheme, and idf(t) is the idf value of term t, i.e., $\log_2(N/df_t)$, where N is the collection size and df_t is the number of documents containing t.

For the evaluation three different model trees were generated, one for each of the TREC data sets. The model tree for assigning weights to terms in the TREC-9 data set was trained on feature representations of words from TREC-10 and TREC-11 (2854 instances), the model tree for the TREC-10 data set used feature representations from TREC-9 and TREC-11 (3167 instances), and the model tree for the TREC-11 data set used feature representations from TREC-9 and TREC-10 (2769 instances).

First, we considered the performance with respect to the answer-at-n (a@n) measure. Table 3 shows the results of using learned query terms weights in contrast to the Lnu.ltc baseline. The improvements are rather modest, although still statistically significant in some cases.

	TREC-9		TREC-10		TREC-11	
a@n	Lnu.ltc	LWR	Lnu.ltc	LWR	Lnu.ltc	LWR
a@5	0.700	0.727 (+3.7%)$^{\triangle}$	0.649	0.654 (+0.1%)	0.523	0.547 (+4.6%)$^{\triangle}$
a@10	0.785	0.806 (+2.7%)$^{\triangle}$	0.734	0.730 (-0.1%)	0.626	0.637 (+1.8%)
a@20	0.845	0.863 (+2.1%)	0.801	0.804 (±0.0%)	0.705	0.732 (+3.8%)$^{\triangle}$
a@50	0.914	0.908 (-0.1%)	0.875	0.859 (-1.8%)	0.795	0.815 (+2.5%)

Table 3. Comparison of the a@n scores of learned-weights retrieval (LWR) to the baseline.

8 Conclusions

In this paper we investigated to what extent it is possible to learn query term weights for better query formulation. As we have seen in Section 3, keyword selection has a strong impact on the performance of the retrieval component. In order to learn query term weights, we considered all possible ways of selecting terms from the original question for query formulation, and used the performance results of each possible formulation in order to determine individual query term weights.

Query terms are represented as sets of features on which the M5′ model tree learning algorithm is trained. The resulting model trees confirm some of the heuristics and intuitions for keyword selection than can be found in the literature; see, e.g., [11]. The improvements are modest for a@n, yet statistically significant in some cases, and stayed behind the potential improvements optimal query selection can yield. On the other hand, our term weight learning approach yields significantly better results than the baseline for mean average precision. Hence question answering systems that are more sensitive to the rank and number of a retrieved document can benefit from using our approach.

In some cases the issue of whether a term is helpful for retrieving answer documents simply depends on idiosyncrasies of the documents that contain an answer, but we do not believe that this had a noticable impact on our results as we used large and varied training data to generalize properly.

References

1. Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning to find answers to questions on the web. *ACM Transactions on Internet Technology*, 4(2):129–162, 2004.
2. Eric Brill, Susan Dumais, and Michelle Banko. An analysis of the AskMSR question-answering system. In *Proceedings of Emperical Methods in Natural Language Processing (EMNLP 2002)*, pages 257–264, 2002.
3. Hsinchun Chen, Ganesan Shankaranarayanan, Linlin She, and Anand Iyer. A machine learning approach to inductive query by examples: An experiment using relevance feedback, ID3, genetic algorithms, and simulated annealing. *Journal of the American Society for Information Science*, 49(8):693–705, 1998.
4. William Cooper, Aitao Chen, and Frederic Gey. Full text retrieval based on probalistic equations with coefficients fitted by logistic regression. In *Proc. of the 2nd Text REtrieval Conference*, pages 57–66, 1993.
5. Brad Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7(1):1–26, 1979.
6. Eibe Frank, Leonard Trigg, Geoffrey Holmes, and Ian H. Witten. Naive bayes for regression. *Machine Learning*, 41(1):5–25, 2000.
7. Dekang Lin. Dependency-based evaluation of minipar. In *Proceedings of the Workshop on the Evaluation of Parsing Systems*, 1998.
8. Lucian Vlad Lita and Jaime Carbonell. Unsupervised question answering data aquisition from local corpora. In *Proceedings of the Thirteenth Conference on Information and Knowledge Management (CIKM 2004)*, pages 607–614, 2004.
9. James Mayfield and Paul McNamee. JHU/APL at TREC 2005: QA retrieval and robust tracks. In Ellen M. Voorhees and Lori P. Buckland, editors, *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*, 2005. NIST Special Publication: SP 500-266.
10. Christof Monz. Document retrieval in the context of question answering. In Fabrizio Sebastiani, editor, *Proceedings of the 25th European Conference on Information Retrieval Research (ECIR-03)*, LNCS 2633, pages 571–579. Springer, 2003.
11. Marius Paşca. *High-Performance Open-Domain Question Answering from Large Text Collections*. PhD thesis, Southern Methodist University, 2001.
12. John R. Quinlan. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.
13. John R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
14. Deepak Ravichandran and Eduard Hovy. Learning surface text patterns for a question answering system. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 41–47, 2002.
15. Yong Wang and Ian H. Witten. Induction of model trees for predicting continuous classes. In *Proceedings of the Poster Papers of the European Conference on Machine Learning (ECML)*, pages 128–137, 1997.
16. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.