Another Sys Called Qanda[1]

Eric Breck, John Burger, Lisa Ferro,
Warren Greiff, Marc Light,[2]
Inderjeet Mani, Jason Rennie

The MITRE Corporation
202 Burlington Road
Bedford, MA 01730

Introduction

This year our primary goal was to improve on the performance of our TREC-8 system. In addition to improving the system directly, we worked on a number of tools to aid our development. We continued our work on a tool for automatic scoring of system responses, a "judge" program. We designed a tool for doing regression testing of question answering systems. We developed a measure of candidate confusability which measures the effectiveness of a set of features for reducing the choices that a ranking system has to make: a coarse form of perplexity. Finally, we performed preliminary work on a method for generating supervised training data.

We began with our system from the TREC-8 competition (Breck et al., 1999). Like many of the TREC-8 systems, it had the system design illustrated in Figure 1. The input question is processed by a question analyzer, which assigns it one of several dozen answer types. The question is also fed to an information retrieval engine, which returns a set of documents. Next, a set of taggers finds entities of the type assigned by the question analyzer and other related types. These entities are then ranked as to how likely they are to be the answer. Finally, answer strings are generated for the top candidates.

For this year, we kept this basic design but improved many of the modules involved. For example, we extended our answer type inventory, improved the question analyzer, and added a temporal dereferencing module to the taggers (Mani & Wilson, 2000). However, much of our effort went into the candidate ranking system. We did this for two reasons. First, an error analysis last year showed that problems in candidate ranking caused a substantial portion of Qanda's error (21%). Second, it seemed to be an appropriate place for a principled method of

---

[1] In keeping with our marine theme, we considered renaming our system Flounder due to its poor performance this year. A trivial bug in the answer candidate ranking system caused candidates to be ranked essentially at random. Perhaps this run should be considered a "chance" baseline.

[2] Principal contact: <light@mitre.org>

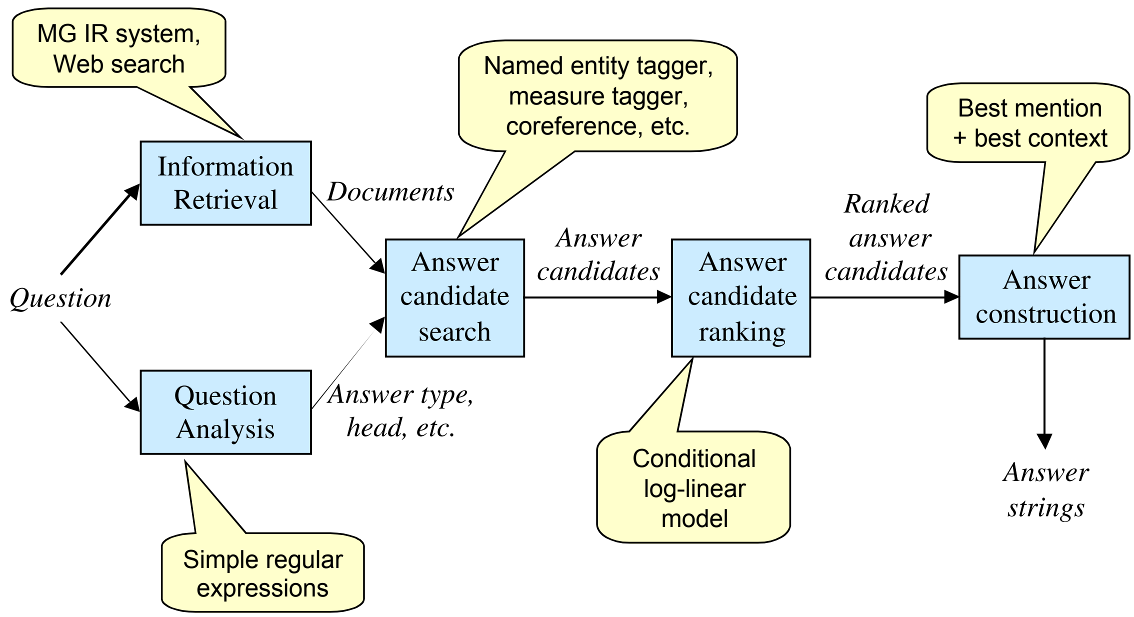


Figure 1: Ubiquitous question-answering architecture

integrating the large number of information sources that we thought were relevant to question answering.

The remainder of this paper is structured as follows: first we describe the development tools just mentioned. Then we discuss our work on candidate ranking and provide a number of relevant results. Finally we talk about our end-to-end experiments and make some concluding remarks.

Development Tools

This year we built a number of tools to aid in system development.

Automatic evaluation with Qaviar and Roe

To keep the development loop tight, we needed a means of automatically evaluating many intermediate system versions and configurations. The methods we use are based on comparing a system's response to a human-developed answer key.

The simplest approach is to judge a candidate answer as correct if it contains, as a substring, any of the alternative reference answers provided in the key. The tool that evaluates candidates in this way is called Roe. In general, we found that this criterion was too strict to use as a means of evaluation, although it is useful for automatically constructing (noisy) training data for our candidate selection component (see below).

The other tool for automatic system evaluation is called Qaviar. The essence of Qaviar's approach is to see if the response has sufficient correct words from the key. The comparison is performed by first normalizing the answer key and the system response. Stop words and duplicates are removed, and the remaining words are stemmed.[3] Recall is calculated as the fraction of words in the answer key found in the system's response, giving a real number between 0 and 1. We use a threshold to produce a binary judgment—in this paper, 0.5. That is, this method judges a response correct if it contains at least half of the stemmed content words present in the human-generated answer key. This method was compared to the judgments of the human assessors at last year's TREC, and found to agree 93–95% of the time. See (Breck et al., 2000) for a variety of comparisons to human judges, as well as a more detailed presentation of the method.

Regression testing with Snapper and Filet

Another tool we found necessary was something to contrast the current version of the system with the previous one, with respect to a development set of questions. An aggregate score, such as RAR, is insufficient for this—what we really wanted was the equivalent of a regression testing infrastructure for question answering systems.

Snapper is a simple tool that provides a snapshot of the system's current output. First, one of the automatic evaluation modules described above is run on the ranked answers to provide a (noisy) judgement of their correctness. For each question in the regression test, we then note the highest rank of a correct answer, the text of this answer, the answer type chosen by the question analyzer, and several other items of interest. Together, these constitute a summary of the system's processing of those questions, which is saved away.

When we wish to compare next week's system, or some other variant, we run Snapper again. We can then compare the two snapshots to see what has changed. Filet is a tool for doing this: It can show only the questions on which the systems differ, as well as provide a number of aggregate statistics such as, for how many questions did the highest rank improve, degrade, etc. These can also be broken down by answer type.

Together, these tools provide us with a useful way to compare various versions of our system. For example, if we improved the named entity tagger's ability to find people's names, we can gauge its effect by comparing the Snapper output, before and after. We would expect to see that questions requiring *Person* or *Agent* answers had improved.

Acquiring question-answer pairs with Piqe

Below, we describe how we used last year's assessments to build a corpus of (noisy) supervised question-answer-passage triplets to train our candidate ranking module. The result of the TREC assessors' efforts is extremely valuable, but rather expensive, data. Because we wanted to train our system on as much data as possible, we also began work on techniques for acquiring more.

---

[3] We used Steven Abney's stemmer from the SCOL toolkit.

The goal of this effort was to develop a semi-automated method for acquiring partially supervised data for candidate ranking, with a corresponding user interface. The basic idea is as follows.

Our Web-based interface presents a data developer with questions from the domain of interest. These are typically culled from a search engine and heuristically filtered to identify actual questions, removing mere keyword queries. If an answer does not yet exist, then the human provides an answer, using whatever resources are appropriate. The rest of the procedure is automatic: using an IR engine, passages are found, either in an appropriate corpus or on the Web, which contain the words in the answer plus some minimum number of words from the question. For instance, if a human identifies *Boston* as the answer to the question *What is the capital of Massachusetts?* then paragraphs that contain the words *Boston*, *capital*, and *Massachusetts* are probably good candidates for passages that answer the question. With a lower threshold, *Boston* with either *capital* or *Massachusetts* might be sufficient.

We have implemented this system but have not evaluated it. Obviously, the approach is not foolproof—there will be some noise in the resulting data. Another issue is the bias inherent in this approach. For example, we will clearly only find training passages that use the same vocabulary as the question. Nonetheless, we believe that we use this methodology to provide a substantial amount of medium- to high-quality data.

Thoughts on evaluating answer-type hierarchies

One thing that seems to be common to the design of many question-answering systems is some sort of answer-type hierarchy. We believe that it is important to understand how the design of this hierarchy affects the difficulty of the task. For example, with respect to a particular set of answer types, we could posit both a perfect question analysis component and a perfect sub-passage retrieval system. The former always correctly identifies the semantic type of the answer, and the later correctly highlights passages (e.g., paragraphs or sentences) containing an answer. Even then, there may be residual error or *confusability* due to multiple entities of the correct type within a highlighted text passage. And so we would like to be able to talk about the degree to which a particular answer-type hierarchy reduces this residual error.

For example, consider the following question and passage containing an answer:

> *Who was Johnny Mathis' high-school track coach?*
> *Many famous people, including O.J. Simpson, Johnny Mathis, and Frank Sinatra, were coached by Vasquez as high-school students.*

The confusability of a type such as *Person* in the second sentence is six, counting the common nouns denoting persons, while a type such as *PersonName* reduces the confusability to four. We have built some simple tools for gauging this notion of confusability, but they depend on humans marking up the data in question. We would also like to take into account the accuracy of the taggers that find entities of the various types. There may be trade-offs between the specificity of the type hierarchy, and a ceiling on the accuracy of the taggers for those types. Since many of our taggers are trained on human-tagged data, the ability of humans to unambiguously tag entities of the proper types is also a factor.

Essentially, what we are groping toward is a metric analogous to *perplexity*, as used in language modeling, a more complete measure that would take into account the answer-type hierarchy, and its discriminatory power, as well as the quality of the taggers used to find entities of those types.

Final comments on system development

Much of the development environment just described was not fully in place during the final weeks leading up to the TREC-9 deadline. We posit that there is a high correlation between effective development tools and system performance. We expect to improve and add to this suite of development tools.

Candidate Ranking

For Qanda, candidate answers are not textual regions such as sentences—they are typed entities such as persons or measurements, with associated contexts. The candidate ranking system is responsible for deciding which candidates to use to construct one or more answers. It does so by evaluating the quality of candidate answers for each question, sorting them, and returning the best. Our TREC-8 system evaluated candidates based on features of the candidate/question pair. The features used last year were: a simple, hard-coded match against the question type; the number of words from the question present in the context of the candidate; the IR rank of the candidate's document; and (for the purpose of tie-breaking) the candidate's position in the document. Candidates were then sorted according to the values of these four features.

For TREC-9, the set of features extracted from the question/candidate pairs was greatly expanded. Also, we have replaced strict sorting according to raw feature values with sorting by the probability of the candidate being judged correct, as given by a conditional log-linear model. The parameters of this model are estimated from supervised training data derived from TREC-8.

Training data

A training corpus of positive and negative question-answer pairs was developed from last year's data (questions 1 through 200) in the following manner:

- We ran Qanda's question analyzer and taggers on every document from which a correctly judged answer was derived (according to the NIST assessors)
- We paired with the question each answer candidate found by any tagger, forming a question-answer-context training instance

- We ran the Roe automatic judgement component described above on the answer candidate
- If Roe judged the answer correct, the question-answer pair was labeled a positive training instance, else it was a negative instance

In this way, we generated 306,000 training instances from last year's data, of which approximately 2% were positive instances. This is the data that was used to train the probabilistic models described below.

Log-linear models for candidate ranking

Our primary candidate-ranking component uses a conditional log-linear model (Berger et al., 1996) trained on the corpus just described. These models take the following form:

$$\mathrm{P}(y \mid x) = \frac{1}{Z_x} e^{\sum_i \lambda_i f_i(y,x)}$$

For Qanda, x is a question-candidate pair and y is a Boolean variable indicating whether the candidate answers the question. Z_x is a normalizing constant, and the f_i are features of the candidate. Training the model amounts to acquiring the maximum likelihood set of weights (the λ_i) with respect to the training data. The 306,000 training instances, as well as the actual candidates produced at run-time, were characterized by the following:

- Features of question/context overlap: noun overlap, verb overlap, proper noun overlap, overlap of the question with a proposed candidate, and bigram overlap
- Type match features: perfect match and various degrees of semantic distance between the question type and the candidate type.

Additional features were added conjoining each of the basic features listed above with one categorizing the question as seeking a temporal phrase, a number, an explanation, an agent, a location or simply a noun or verb phrase. Altogether, the log-linear model uses several hundred Boolean features to characterize answer candidates.

In TREC-8 we used a radix sort to combine these features: candidates were ordered by type match features, then overlap, document rank, and finally distance from beginning of document. One way to think about radix sorting is that a feature has a weight greater than the sum of all the weights of the features that follow it. It is a very strong constraint on the relation between features and we suspected that it was sub-optimal: that features are much closer in weight than radix sorting allows.

Feature Selection

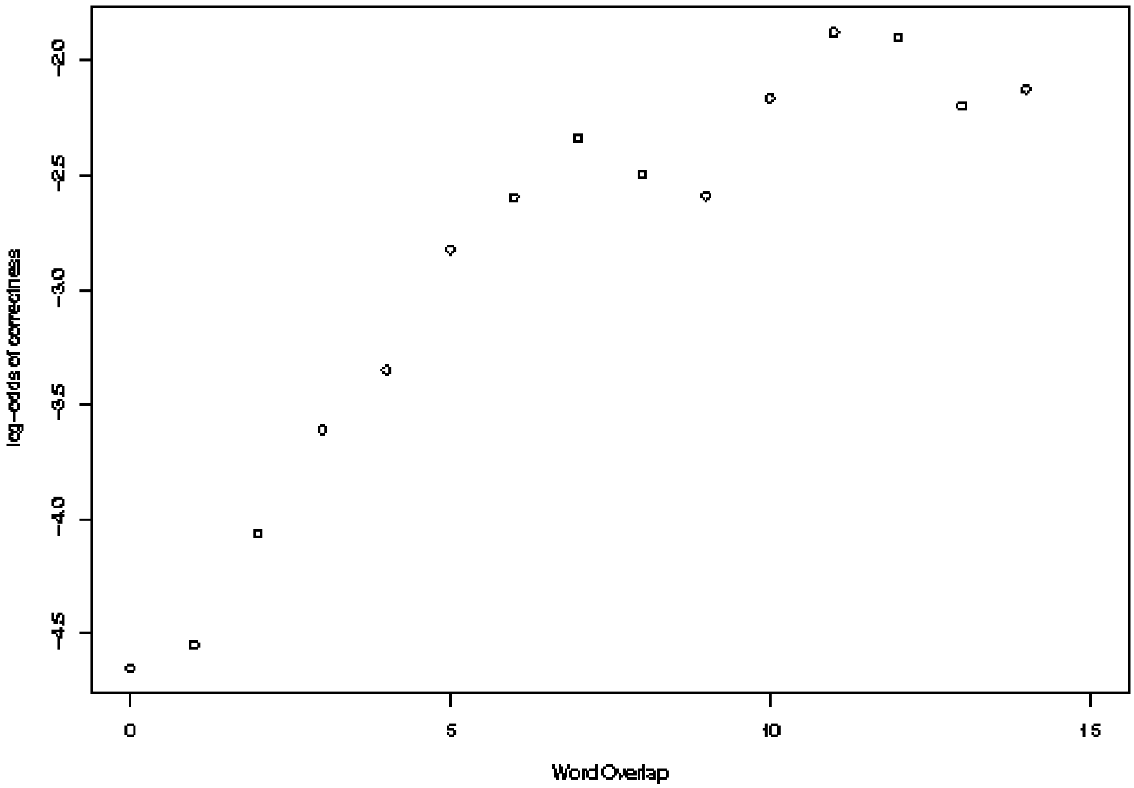
We performed a substantial amount of statistical analysis to learn which features and feature combinations are good predictors of candidate correctness. Once a useful feature set for the model is chosen, model parameters are determined according to the criterion of maximum likelihood. The statistical analysis was performed on questions and documents from TREC-8. The questions were run through Qanda, and then each candidate was judged to be correct or incorrect by the automatic scoring system described in the previous section.

An important advantage of the statistical modeling approach is the ability to analyze the predictive value of features that are being considered for inclusion in the ranking scheme. For example, in Figure 2 shows a graph that was used for gaining initial insight into the behavior of the word overlap feature. The graph shows the coefficient for each possible value of the overlap feature that results from the estimation of a simple model based on only this one feature, treated, for this purpose, as a categorical variable. These coefficients have a direct interpretation in terms of the probability of correctness. In each case the coefficient is equivalent to the log-odds ($\log(p/(1-p))$) of correctness conditioned on the overlap feature assuming a given value.

For example, for an overlap value of 5, the odds of correctness is given by $e^{-2.82} = .059$, equivalent to a probability of .056. We can see from this graph that the probability of correctness when overlap is 1, is only slightly greater than the probability when there is no overlap at all. For greater values of overlap, the probability rises, tapering off to a stable value as overlap approaches approximately 10. The variability of the coefficient estimates for overlap values beyond 7 can be attributed to the sparseness of data available for the estimation at these values. It is important to emphasize that the coefficients shown in Figure 2 are those for a model based exclusively on the single overlap feature. A more complete model, based on the overlap feature in conjunction with a number of other features, can result in very different values for the coefficients. Nonetheless, the analysis based on the isolated overlap feature is useful for obtaining initial insights into its behavior as a predictor of candidate correctness.

Another example of how statistical analysis can be used to determine the best way to model the data for ranking is shown in Figure 3. The four curves correspond to the coefficients of the proper-noun-overlap feature for four different models. For each model, the data was restricted to a given question type. The top line corresponds to questions that are looking for *Locations,* as determined by the question analyzer; the two in the middle correspond to questions looking for *Agent*, and the bottom curve to *Quantities*. The similar shape of the bottom three curves gives reason to believe that these three categories can be combined without problem for the purposes of modeling the increase in probability of correctness corresponding to increased values of the proper noun overlap feature. The dip for high values in the bottom curve is not of great concern, being suspect because the estimates are based on somewhat sparse data. Note that the lower absolute values associated with the bottom curve are due to the (prior) probability of correctness being lower for quantity questions, in general. This is not an issue, since it is only the relative values of the probabilities that are relevant for a given question. The curve for location

Figure 2: Log-linear model coefficients for simple word overlap



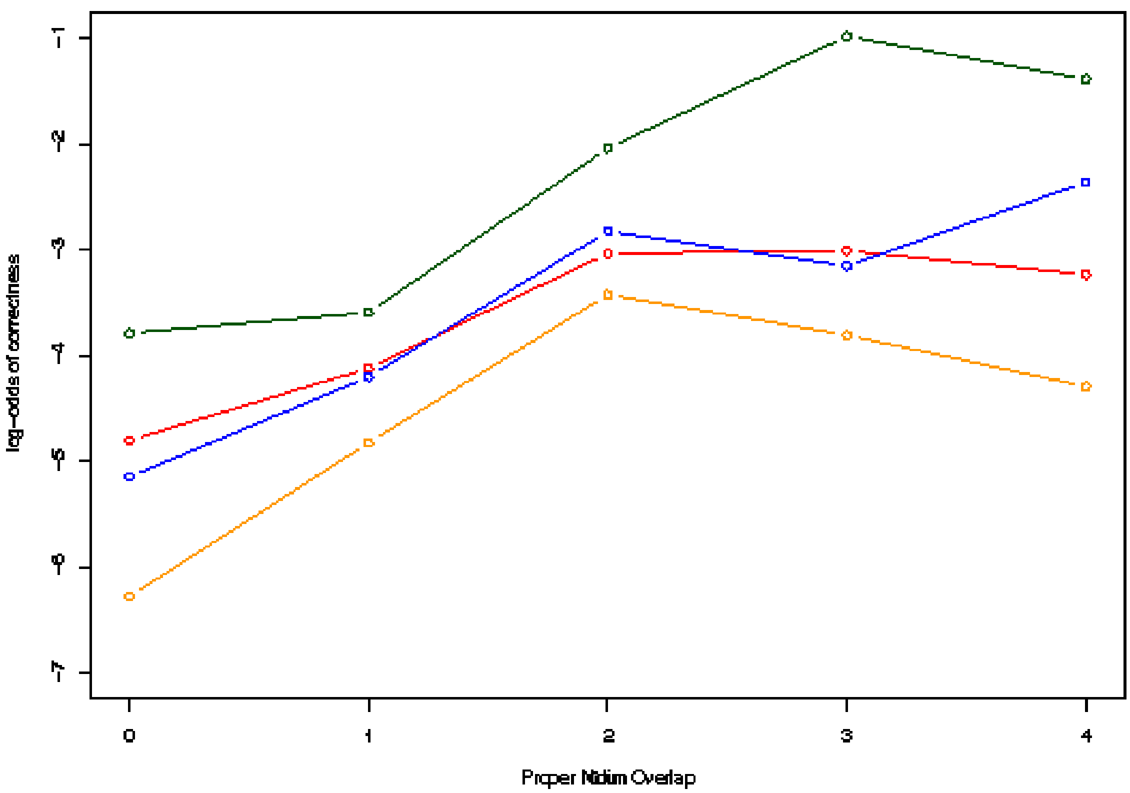


Figure 3: Proper-noun overlap coefficients for four answer types

questions, however, rises more sharply, obtaining greater values relative to the coefficient associated with no overlap. This suggests that ranking might be improved by using a model for which the interaction between question type and proper-noun-overlap is taken into account, at least insofar as to whether the question is of type location or not.

Experiments

As noted, our official entries this year suffered from a bug that rendered our results meaningless. After the official evaluation, we ran our system on 200 of the 500 unique questions from the TREC-9 data. Our chief annotator developed an answer key for these questions, informed partially by examining the other system responses that were assessed favorably by the TREC judges. We scored Qanda's unofficial responses against this answer key using the Qaviar automated scoring system[4] Our results are summarized in Figure 4. The **Radix** score was produced by running Qanda with a simple radix ranking module, sorting on type-match features first followed by overlap features. The **LLM** score was produced by running Qanda with a log linear model trained on the data described above using 344 features.

Conclusion

For TREC-9 we concentrated on development tools and candidate ranking. We worked on tools for automated scoring, regression testing, measures for candidate confusability, and methods for finding supervised training data. For candidate ranking we employed statistical tools to study a number of features predictive of candidate correctness.

	Right #1	*Right top5*	*RAR*
Radix 250-byte	11.6%	30.7%	0.187
LLM 50-byte	13.1%	33.2%	0.206
LLM 250-byte	20.6%	41.7%	0.282

Figure 4: Unofficial results of several candidate rankers

[4] As noted above, this approach agreed 93–95% of the time with the TREC assessors on last year's data (Breck et al., 2000).

References

Adam Berger, Steven Della Pietra and Vincent Della Pietra (1996). “A maximum entropy approach to natural language processing”, *Computational Linguistics* 22-1.

Eric J. Breck, John D. Burger, Lisa Ferro, David House, Marc Light , Inderjeet Mani (1999). “A sys called Qanda”, *Proceedings of the Eighth Text REtrieval Conference.*

Eric J. Breck, John D. Burger, Lisa Ferro, Lynette Hirschman, David House, Marc Light, Inderjeet Mani (2000). “How to evaluate your question answering system every day … and still get real work done”, *Second International Conference on Language Resources and Evaluation*.

Inderjeet Mani and George Wilson (2000). “Robust temporal processing of news”, *38th Annual Meeting of the Association for Computational Linguistic*.