CS224 Final Project: Answer Extraction

Aria Haghighi, Guy Isley, Alex Williams

June 3, 2004

1 Introduction

As large databases of documents become more and more common, the value of a system that could perform flexible natural language queries that are not domain-specific has become increasingly apparent. In particular, the construction of open-domain, natural language question answering systems has garnered much attention as an application in which to showcase developments in natural language processing techniques. The annual TREC QA contest sponsored by NIST has been used monitor progress of such systems. For the TREC contest, systems are provided with a large database of approximately one million documents from different news sources. In the most recent iterations, systems have been asked to answer three types of questions: list questions ask for a group of entities with a specific property, definition questions request the salient properties of some individual entity, and factoid questions are looking for single answer about some fact of the world. Of these question categories, the factoid questions are by far the easiest to answer using current methodologies, primarily because they provide the most concrete information as far as what the correct answer should look like. Due to the time constraints, we limited our research to the task of answering this category of questions.

Many modern open-domain QA systems can be divided into three main components. First, there is a module that does question classification. Among other things, this module is responsible for determining the expected answer type of the question. Second, there is a module that performs the initial information retrieval on the provided database of documents, selecting passages that are reasonably likely to contain an answer to the question. This is usually done using basic keyword search techniques. It is simply too costly to do deep natural language processing on the initial collection of documents so it is necessary to first narrow the field with shallow IR. Finally, there is a module that extracts a single phrase from the passages provided by the IR, in an attempt to give a correct answer to the given question.

This last module is the focus of our project. In particular, we obtained output from a question answering system developed by a group at the University of Colorado at Boulder[2] that had already done question classification and initial IR on 500 factoid questions from the 2002 TREC contest. For each question, the data provided us with the expected answer type that they had determined for the question as well as the 10 passages that their IR module had selected as the most likely candidates to contain a correct answer. We split the data, randomly selecting 70% of it as a training set and 30% as a test set. We used the training set to train a classifier that decides how likely a

passage is to contain a correct answer and then ranks the passages accordingly. Given the ranked passages, we then decide which entities within individual passages were mostly likely to be actual answers, based on a number of features, and extract our top 5 guesses as specified by the TREC protocol.

2 An Assessment of the Initial Data

2.1 TREC Data

The TREC 2002 main QA contest consisted of 500 factoid questions and the usual large database of documents that may or may not contain an answer to each of the questions. Here are a few sample questions:

- Who is Tom Cruise Married to?
- What was the first spaceship on the moon?
- Where is the oldest synagogue in the United States?
- When did Alexander Graham Bell invent the telephone?
- How many ounces are in a gallon?
- Which U.S. state is the leading corn producer?

In addition to questions like these that require a simple answer of a single noun phrase, there are a few questions that involve more complex answers, such as:

- How did Mahatma Gandhi Die?
- The sun is mostly made up of what two gasses?

Since the vast majority of the questions are of the former type, we decided to concentrate on correctly extracting the type of answers they require. As a result, the task is considerably simplified, because in most cases we can expect to extract a single named entity as the answer.

In addition to the questions, the TREC site provides a file containing a list of question IDs paired with regular expressions that encode the answers. To determine whether or not an answer to a question is correct, the answer can be matched to the regular expression for the corresponding question ID. By examining which answers were accepted as correct in previous years of the TREC competition and observing how well patterns in the file covered answers which we deemed to be correct, we determined that the patterns correctly accepted about 85% of the answers that could be found in the documents. Although this method introduced some noise, we had to accept the patterns as our method of answer verification while we were training the classifier, since we certainly could not hand-grade each answer while training. We did, however, decide to hand-grade the answers our system gave to the final test set in order to get a more accurate evaluation.

2.2 Question Classifications

As noted above, our question classifications were generated by a system developed a group at UC Boulder for the TREC 2002 competition. In their paper on their system, they explain that they "classify the questions according to type by identifying the Name Entity and Thematic Role of the expected answer type."[2] Since we also used the named entity tags that their named entity tagger provided for each candidate passage, this method of question classification supplied us with a convenient and nearly one-to-one mapping between the expected answer types for the questions and named entities in the passages.

However, we noticed certain deficiencies in the question classifications. For example, "Who" questions should be easy to classify since almost all of them expect answers of type `PERSON`. However, a number of "Who" questions where not classified in this manner. For example, the expected answer to "Who is the owner of the St. Petersburg Times?" was listed as type `DATE`, and other "Who" questions were also inappropriately classified with answer types such as `PRODUCT` and `US_STATE`. "What" questions are much more difficult, and there were correspondingly more errors. One particularly baffling misclassification gave the expected answer type of "What is written on the Tomb of the Unknown Soldier?" as `ANIMAL` (admittedly, this question is difficult to classify). Such misclassifications are almost certain to result in an incorrect final answer since they cause our system to attempt to extract answers of the wrong type. Overall, we found that approximately 30% of questions had been misclassified. In addition to these misclassifications, there were also many cases where the expected answer type was too vague to be of much use. A number of questions were classified as `OTHER`, even some in which the question clearly expected a more specific answer type from the list of named entities.

2.3 Candidate Answer Passages

The answer passages—also provided by the UC Boulder group—consisted of 10 passages per question that had been determined to be likely to contain the answer and had been marked-up with named entity tags. To get the passages, the UC Boulder group used a widely available IR engine to rank documents, selected the 500 best documents, segmented the documents into paragraphs, and then ranked and selected the 2500 best paragraphs. Finally, they computed a confidence interval for the confidence that the paragraphs contain some relevant information and returned the top passages according to this measure[2]. Ideally, we would have liked to have more than 10 passages per question and more information on how exactly the 10 passages were selected, but this information was not available to us.

We used the answer patterns to determine what percentage of the questions are actually paired with at least one answer passage that contains the correct answer. We found that only approximately 42% of the questions have any correct answer at all in their passages. Of course, this number is a little bit lower than the true figure because, as noted above, the answer patterns don't provide full coverage of the correct answers that can found in the documents. Still, it is rather disappointing performance, considering that it imposes an upper limit on the overall performance of our system. It may be that it is low because the UC Boulder group does more than basic IR to extract the 10 passages per question. It probably would have been better for our system if we had had more passages

with higher coverage of the correct answers rather than less coverage with fewer passages (i.e. higher recall in exchange for less precision).

We also had to deal with several problems created by the named entity tagging for the passages. As is to be expected, a number of the entities were incorrectly tagged, such as individuals with foreign names being tagged as organizations. In examining results of runs on the test data, we found several instances in which the system would have made the correct choice but for these misleading tags. More troubling were cases where the tagger improperly split single words that represent a single entity into multiple words representing multiple entities. For example, "Clinton" was split into "Clin" and "ton." Not only did this mislead our classifier, but it also made the task of parsing named entities unnecessarily difficult.

3 Choosing a Passage

In order to answer a question, we must first select a passage and then a phrase from that passage. We treated choosing a passage from the collection of passages given to us with the question as a supervised classification problem. Specifically, for a given question, we can generate a set of features for each passage along with a label ({`true, false`}) indicating whether the passage contains the correct answer.

Given a collection of questions, we can generate a training set from each of the feature vectors and labels for the passages from each question. Normally, in supervised learning we assume that all our training examples come from the same underlying distribution, but our strategy does not have this feature since we lump passages from different questions together and are therefore introducing lots of "noise" into the training of our classifier. We reasoned that this wouldn't have a significant effect, because when we are actually doing the classifying, we are only comparing classifier confidences among passages from the same question and choosing those with the highest prediction, so it doesn't matter if we shift all the prediction values up or down with a little noise. We realized belatedly that we probably should have normalized the features according to the question that the passage was initially contained in. This would have eliminated any differences between different questions and their passages. As it stands however, the assumptions we are violating are common victims in machine learning, and so we move on.

The following section describes the features we used for each passage in the creation of the training data. Several of these features are based on features from Pasca and Harabagiu[5].

4 Features

4.1 Question Word Matches

This very simple feature just returns the number of words in the answer passage which also appear somewhere in the corresponding question (comparison is case-insensitive, but matching words must be otherwise identical). Arguably, this is a feature of the passage that is used in the initial IR filtering, and should be replaced by more fine-grained features

that couldn't be carried out in the initial filtering, but we retained this feature because its addition shouldn't *hurt* performance, and could still potentially be useful in some cases.

4.2 Expected Answer Matches

For each question, we are supplied with an expected answer type in the form of a Named Entity Tag. This feature simply gives the number of named entities in the target passage that match the expected answer type. If the value for this feature is zero, then this answer passage is unlikely to be useful, since it contains no examples of the expected answer type at all. However, values of greater than one are not necessarily any more promising than passages with a single expected answer type match.

4.3 Punctuation

This feature adds a score of 1 for each word/phrase of the expected answer type that is followed by a punctuation mark (and 0 when the expected answer is not followed by a punctuation mark) and then returns the total punctuation score divided by the number of expected answer type matches. This takes into account the relatively frequent case in which there is more than one match of the expected answer type in a candidate answer; regardless of the number of expected answer type matches, this feature gives the *average* number of expected answer type matches that are followed by punctuation. We were not sure exactly why this feature would be useful. Perhaps it can be justified by the fact that named entities followed by punctuation are often modifiers that yield key information. For example, in "John Smith, born 1904, ..." the commas indicate the fact that John Smith and 1904 are probably key figures in the passage. Despite lack of better justification, we decided to include this feature since Pasca et al. seem to have obtained good results using it.

4.4 Question Words in Close Proximity

This feature is similar to the "Question Word Matches" feature described above, except that it only counts words in the question that also appear in the answer passage within a certain proximity of a word of the expected answer type. We ended up using the same "closeness" criteria from the Pasca paper, counting a word from the question if it was no farther than 3 words and one comma (and no stronger punctuation, like a period) from a particular word of the expected answer type. Proximity to question words suggest that a candidate answer is likely actually be related to the question words, rather than just appearing in the same passage with them. Punctuation here encodes for the fact the words across phrase and sentence boundaries are less likely to be semantically related to each other. In answers with more than one word of the expected answer type, we consider the number of question words that are close to each individual word matching the expected answer type, and then return the average. This feature can be particularly useful for answers in which a large portion of the question is repeated in the answer.

4.5 Question Words in Order

Here we return the maximum number of words from the question that occur together in the same order in the answer. We only consider uninterrupted strings of words from the question, skipping over any punctuation marks. High scores for this feature can indicate an answer that is especially on-topic and repeats part of the question verbatim. Note that the average expected score for this feature increases as answer passage length increases (which is not true of the features which are based on averages of certain properties of the answer). We assumed this would not be a significant problem since the passages are all approximately the same length.

4.6 WordNet Similarity

It is important to use a feature which uses lexical knowledge in order to identify question text synonyms in an answer passage which gives that passage more relevance to the question than string text matching IR techniques would recognize. Given a measure of similarity between words $sim(w_1, w_2)$ we define the `WordSimilarity` feature for an answer passage P and question Q to be [if we let Q_K be the set of keywords derived from pruning function words in Q]

$$WordSimilarity(P_i, Q) = \frac{1}{|P_i||Q|} \sum_{(w_i, w_j) \in (Q_K, P)} sim(w_i, w_j)$$

We utilized the `perl` module `WordNet::Similarity` in order to get similarity scores derived from the WordNet taxonomy [3]. In particular we used the Resnik measure, which is given by the maximal information content of any common ancestor of two concepts. Formally, if we let c_1, c_2 be concepts and $S(c_1, c_2)$ be the set of common ancestors, then the Resnik measure [6] is given by:

$$R(c_1, c_2) = \max_{c \in S(c_1, c_2)} -\log p(c)$$

where the probability distribution over concepts is empirically estimated from a corpus and the concepts assigned to the words in it. For two words, $sim(w_1, w_2)$ is given by the maximal Resnik measure over pairs of applicable WordNet concepts, or *sys-sets* corresponding to w_1 and w_2.

It is important to note that many of the *sim* scores for a question and passage will be zero if either WordNet does not contain the concept or if the concepts share no common ancestors (the former is much more likely to be the case).

4.7 Google Mutual Information

For each passage P, we will eventually use some named entity—of the same type as the expected answer type—as our final answer. Since named entities tend to be proper nouns, our WordSimilarity feature will typically not tell us anything about how closely related our answers are to the keywords from the question. Our $MutualInformation$ feature tries to capture this similarity/dependence by averaging the mutual information between the named entity phrase and the keyword phrase. The mutual information between two random variables is intended to measure the amount of information that one random

variable's value gives about another variable's value. We should expect that if a named entity is a correct answer, then it will have high mutual information with keywords from the phrase. Formally, mutual information for random variables X and Y is equivalent to

$$MI(X,Y) = H(X) + H(Y) - H(X,Y)$$

where $H(X)$ denotes the shannon-entropy of X. Of course the notion of entropy relies on a probability distribution, so how do we put a probability distribution on a phrase ? We derived a empirical probability distribution using *Google*. We divide the number of hits *Google* gives us divided by the total number of pages it searches[1]. Note that we are not using any information in the retrieved passages in our search other than the counts. We let the intersection of two random variables represent the concatenation of the two phrases for the two random variables.

Given the *Google* probability distribution of a phrase, we define our $GoogleMutualInfomation$ feature for a passage P with Named Entities P_{NE} of the expected answer type of question Q with keywords Q_K is given by:

$$GMI(Q,P) = \frac{1}{|P_{NE}|} \sum_{NE \in P_{NE}} MI(Q_K, NE)$$

So we average the mutual information between each of the potential answer named entities and the keyword phrase from the question .

As noted above our WordSimilarity feature does not cover proper nouns (which aren't in WordNet). It would make sense then to have a feature which tried to measure the similarity among proper nouns. Of course this task might be better accomplished at the IR level. However, from our assessment of the data provided to us, we did not feel very confident in what the IR module had achieved, so we felt justified in using a feature that might normally be employed in the IR phase.

4.8 A Future Feature

One feature which would have liked to add but were unable to due to the time constraint was a measure of how likely a candidate answer is to play the same semantic role as the expected answer type of the question. Since some candidate answers may occur near the question keywords without being in the right semantic relation to them, this feature would be a good way to identify misleading answers. Semantic roles were used by the group UC Boulder, and they emphasize them as one highlights of their approach. In "Automatic Labeling of Semantic Roles," Gildea and Jurafsky[4] explain one approach to determining the semantic role of words in passage. However, their approach is rather complex, require parsing of the passage and semantic analysis—in other words, enough work for a final project in and of itself. We decided to forego this feature in the interest of a more broadly developed approach.

[1]Thanks to Jenny Rose Finkel for giving us her `GoogleCounter.java` code from her and Aria's CS 229 Final Project

5 Which Classifier To Use

We trained and tested several types of classifiers using the Open-Source Java package Weka[7]. Our first hunch was to use a support-vector-machine (SVM) with various kernels. SVMs are one of the best performing classifiers for a wide range of classification problems. Surprisingly, this classifier did terribly, and we are still not exactly sure why. After trying out some misses such as $NaiveBayes$[2] and $LogisticRegression$, we found that the classifiers which did the best was $AdaBoost$ on $J48$ decision trees. The $AdaBoost$ algorithm is a Meta Classifier which takes a collection of weak classifiers (decision trees in our case) and will assign weights to classifiers proportional to their weighted classification error and weights to training instances based on how difficult they are to correctly classify. Its superior performance relative to the other classifiers we investigated stems from the fact that it is capable of memorizing more of the data.

On our testing data, this model achieved 55.5% accuracy (as measured by the percent of passages it correctly classified as containing or not containing an answer) and an F-Measure of 31.3%[3] on the correct passages. Another test we performed was to put all the answer passages back into the original question they came from and see how frequently the best passage chosen by the classifier actually contained the correct answer. This turned out to be 53% of the time in cases where there was correct answer to be found, which (probably) provides an upper bound as to how well we can select answers to a question, since that problem involves correctly selecting a passage as a sub-problem.

6 Extracting Answers from Individual Passages

Once we have ordered the passages according to how likely they are to contain the answer, to complete the TREC question answering task we need to extract 5 answers phrases from the passages.

A basic assumption we made in doing this was to assume that any possible answer must be a named entity of the same type as expected answer type. In retrospect, this probably wasn't necessarily a wise assumption. Due to inaccuracies in the question classifier and the named entity tagger, in some cases a correct answer will be tagged with a different type than the expected answer type, and in other cases a correct answer may not get tagged at all. However, since we could not think of a good way at the time of accounting for these possibilities, we decided to assume correct answers would actually match the expected answer type. However, one method that might have worked would be to measure the semantic similarity (perhaps using WordNet) between the expected answer type and candidate answers, and then to use that measure to weight the chance of selecting a candidate answer.

The way that we actually went about extracting answer was as follows. First we extracted all the named entities that matched the expected answer type from the top five passages for a question. Next we sorted the named entities passages according a comparison function that was basically a decision list heuristic. If an answer was from a passage that our passage classifier gave a confidence of more than 5% higher than the

[2]which we should expect to do poorly since many of our features have very high correlation

[3]The F measure is given by $\frac{2PR}{P+R}$, where P, R are precision and recall respectively

confidence given to the passage for the other answer, then the first answer was always preferred. If the question was a "When" question and the one answer contained digits while the other didn't, the answer with digits was always preferred. The reason for this was somewhat of a hack: phrases such "two hours ago" or "tomorrow" were tagged in our passages as `DATE` entities, which are the expected answer type for most "When" questions, but almost all the "When" questions in TREC refer to fixed times in the past. Thus, digits were a good indicator that a candidate answer also refers fixed date or time period. Next, we preferred answers with fewer tokens over more tokens. The reason for this is that if an answer has too many tokens, it is often a good indication that named the entity tagger went astray and added far too many tokens to a single entity. Finally, if none of the other decisions applied, we chose the answer that had the largest count in co-occurrence with keywords on *Google*. Given that we couldn't do any better deciding between two entries, we figured that this coarse measure of contextual co-occurrence was about the best we could do.

Using this decision list as a comparison function, it was simple matter to sort the answers using a standard sorting algorithm and return the top 5 answers.

7 What We Would Have Liked to Have Done With More Time

After completing the step that extracts an actual answer from an individual passage, we found that there was a significant decline in performance from the level of the passages to the level of individual phrase answers. We realized that we had overlooked many of the important difficulties for answer extraction that occur at the level of extracting answers from a passage. In part, we focused on deciding between passages because we expected that we would have more passages eventually. As it is, our system would probably have better performance if we could test on less-processed passages with a better coverage of the correct answers.

If we could have focused more of our efforts on the lower-level classification problem, we would have been able to use deeper features that don't make sense for the whole passage. For instance, we could have considered features of the parses of sentences in the passage and the relationships of words in the syntactic trees, which is probably too fine-grained to use at the level of choosing between passages. If we were to do it again, we would probably have trained our classifier to select between individual entities in passages, rather than complete passages, and focus our efforts at this level.

One technique that might have worked to improve our performance at the level of choosing between entities in an individual passage would have been the use of weak abduction to attempt to prove our answers by using the passages they came from as premises. This approach is in fact used by QA system developed by Pasca et al. which has been by far the most successful of the QA systems at recent TREC competitions. One method of using abduction costs to decide how a set sentences justifies a particular sentence as a conclusion is described by Hobbs and Stickel.[1] We could use a similar method to determine how well an answer to a question is justified by the passage that the answer came from.

We considering implementing such a procedure as follows: First, we would take that

	Graded Using Pattern File	*Hand-Graded*
Raw Accuracy	18.3%	25%
Confidence Weighted Accuracy	27.4%	38.7%

Table 1: Performance for passages with some answer

	Graded Using Pattern File	*Hand-Graded*
Raw Accuracy	7.76%	10.6%
Confidence Weighted Accuracy	11.6%	16.4

Table 2: Overall performance

candidate answer and use it to fill the gap in the question and turn it into a statement. In some cases this is easy. For example, we can use "Roy Romer" to fill in the gap in "Who is the Governor of Colorado" and get "Roy Romer is the Governor of Colorado" as a statement. In other cases, however, more complex syntactic manipulations would be required. Once a statement has been generated from the question and answer, we could use it as a conclusion to be proved using the passage sentences as the premises.

We considered using an abductive theorem prover currently being developed by Rajat Raina to do the actual proving, but eventually we decided against attempting to implement the whole procedure. We found many difficulties in getting coverage of the questions for our method of converting the questions to statements. Also, we worried about difficulties interfacing with the theorem prover and that the theorem prover might not be able to handle syntax as complex as that in our passages. This would be a good approach to try with more time.

Another possible addition involves the observation that "Where" questions are such that we can often expect our answer to be in a PP phrase. Thus, if we were to parse the passages and find that a candidate answer to a "Where" question is not inside a PP, then we can rank it lower.

Some of the features that compare word equality between the question and the potential answer passages could potentially have benefited from only considering the word stem, rather than the inflected form. However, stemming was of limited utility in the WSD project (our classifier performed better without it in most cases), and because it was actually useful in the Q/A system to find particular passages from the question repeated in the answer verbatim, we did not implement stemming. It would probably still be of some use in certain features, however.

8 Results

A summary of our results is contained in the tables 1 and 2 above.

TREC contestant systems are currently compared using the following system to obtain a confidence-weighted accuracy: If the correct answer is listed as the first answer, the system receives 1 point. If it is listed second, the system receives 1/2 point. Listed third receives 1/3 point, fourth receives 1/4 point, and fifth gets 1/5. If the system does not have a correct answer in the top 5, it does not receive any points. Finally, the system's overall score is divided by the number of questions. The resulting confidence-weighted

score can be viewed as percent accuracy, because it is guaranteed to be less than 100, but it is generally higher than the raw accuracy since systems get points for questions even when they don't list the correct answer as a first guess.

When evaluating our performance on the test data, we used two methods: evaluation based on the pattern matching file provided by the TREC contest, and hand-grading to account for problems and deficiencies in the answer patterns. The hand-graded accuracy is notably higher because patterns fail to account for certain slight differences in the answers in the passages as well as some more complex patterns. For example, in one case a correct answer was marked as incorrect simply because it contained a dash in it. In another, the pattern required "1941" as the answer for "When did the United States Enter World War II?" and our system returned "the early 1940s," which is arguably correct from a historical perspective and similar enough in meaning to be accepted. Admittedly, judgments in the latter case are somewhat subjective, but it is hard to create an objective acceptance procedure without being somewhat arbitrary.

Overall, the performance of our system was not excellent, but it was relatively good considering the quality of the initial data we were given and the timeframe we were alloted in which to implement it. For reference, the complete UC Boulder system that we obtained the data from acheived 22.6% confidence weighted accuracy after substantial longer development and with better integration between their different modules.

References

[1] Jerry R. Hobbs et al. Interpretation as abduction. http://www.isi.edu/ hobbs/interp-abduct-ai.pdf.

[2] Sameer S. Pradhan et al. Building a foundation system for producing short answers to factual questions. In *Proceedings of the Eleventh Text REtrieval Conference (TREC-11)*, 2002.

[3] Ted Pedersen et. al. Wordnet::similarity - measuring the relatedenss of concepts. 2004. http://search.cpan.org/dist/WordNet-Similarity.

[4] Daniel Gildea and Dan Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 99(9), 2001.

[5] Marius Pasca and Sanda Harabagiu. High performance question/answering. *24th Annual International ACL SIGIR Conference on Research and Development in Information Retrieval*, 2001.

[6] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[7] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations.* Morgan Kaufmann, 2000.