The Use of Paraphrase Identification in the Retrieval of Appropriate Responses for Script Based Conversational Agents

Jerome L. McClendon, Naja A. Mack, Larry F. Hodges

Abstract

This paper presents an approach to creating intelligent conversational agents that are capable of returning appropriate responses to natural language input. Our approach consists of using a supervised learning algorithm in combination with different NLP algorithms in training the system to identify paraphrases of the user's question stored in a database. When tested on a data set consisting of questions and answers for a current conversational agent project, our approach returned an accuracy score of 79.15%, a precision score of 77.58% and a recall score of 78.01%.

Introduction

In this paper we concentrate on the problem of retrieving an appropriate response for a conversational agent based on the user's natural language input. Previous studies done on conversational agents found that, if the agent is unable to produce an appropriate response to the user's last statement, the user may become frustrated and is less likely to use the system (Bloodworth et al. 2012). The difficulty in retrieving an appropriate response for a working conversational agent is determined by the complexity of the inputs that the system will accept. A straightforward approach is to have the conversational agent control the conversation by asking questions and provide the human user with a list of possible keyword responses. A more general approach is to create a set of questions paired with answers (a script or conversational corpus) stored in either an XML file or a database (Dickerson et al. 2005; Leuski et al. 2009). The conversational agent then tries to match a freeform natural language question to the most similar question contained in the conversation corpus. The paired answer is then returned as the most appropriate response. If the question by the user is outside of the expected domain of the conversation and there is no appropriate answer stored in the database, then the appropriate response should be equivalent to "I do not understand that question."

The difficulty in matching the user's question to the most similar question in the corpus is due to the variability of natural language (Dickerson et al. 2005). The conversational agent must be able to identify that a question stored in the corpus is a variation of the user question. Question variations or question paraphrases are questions that have been derived by changing the syntax of the original question by rearranging words, removing words, inserting words, and/or replacing words with synonyms while leaving the meaning of the question unchanged (Bernhard and Gurevych 2008). An example of this would be the two questions shown in Table 1.

User Question	Questions stored in Corpus
What does the pain feel like when you are eating ?	Explain the pain you feel when eating.

Table 1: The user's question and a variation of that question stored in the corpus.

In our system, known as Caesar, we used a supervised learning algorithm in combination with different natural language processing(NLP) algorithms in training the system to identify paraphrases of the user's question stored in the corpus.

Related Work

Supervised Learning Algorithms used in paraphrase identification consist of two phases: training and classification. During the training phase, a supervised learning algorithm produces an output function (the classifier) based on labeled examples (the training set). The labeled examples are sentence pairs labeled *1* or *0*. A *1* represents a paraphrase (positive example) and a *0* indicates a non-paraphrase (negative example). Using the sentence pairs, a feature vector is created using NLP modules that produce similarity scores that measure the similarity between the two sentences. Once the system has been trained, the classifier takes in two sentences and creates a feature vector based on those sentences. The classifier then uses this feature vector to decide if the unseen instance is a positive example or negative example of a paraphrase (Brockett and Dolan 2005; Zhao, Zhou, and Liu 2007; Chitra and Kumar 2010). We chose to use a support vector machine(SVM) because of its ability to work with noisy training data (Brockett and Dolan 2005; Chitra and Kumar 2010).

Brockett & Dolan, use a SVM along with several NLP features that include: string similarity modules, morphological variants, WordNet lexical mappings and word association pairs to extract paraphrases from a corpus of news clusters for the creation of a large monolingual paraphrase corpora (Brockett and Dolan 2005). In order to train the system the authors of the paper used a training set consisting of 10,000 sentence pairs labeled *1* (positive example) or *0* (negative example) by human judges. Of the 10,000 sentence pairs 2,968 sentences were labeled *1* and 7,032 were labeled *0*.

Chitra & Kumar also uses a SVM with a combination of lexical and semantic features to identify paraphrases. The lexical features measured the amount of word level overlap found between two input sentences. The semantic features measured the amount of meaning or concepts shared between the two sentences. Results from their experiments revealed that the features that relied on common consecutive or insequence matches were able to identify 70% of the paraphrases (Chitra and Kumar 2010).

The implementation of our system Caesar uses a SVM in combination with different NLP algorithms in training the system to identify paraphrases of the user's question stored in the corpus. An advantage to using machine learning over other techniques is its ability to analyze and incorporate different information sources (NLP algorithms) such as morphologic, syntactic, and semantic when determining paraphrases (Chitra and Kumar 2010).

Modules Descriptions

For our system we use ten different NLP modules when determining whether two sentences are paraphrases. These modules can be broken into three different categories, each examining different aspects of the sentence: word overlap, word order and n-grams. Each module produces a score between 0 and 1, where 0 indicates that the module found no commonality and 1 indicates maximum commonality. These scores define the feature vector that is later used in the training and classification stages.

Word Overlap

Word overlap modules compute commonality scores based on what words or parts of speech the two sentences have in common. We have implemented six word overlap scoring methods: Jaccard Coefficient, Common Noun Similarity, Proper Noun Similarity, Verb Similarity, Subject Similarity, and Object Similarity.

Let $|s|$ represent the number of items in the set.

In set theory, the Jaccard Coefficient is defined as the number of unique items two sets have in common over the total number of unique items in the two sets (Achananuparp, Hu, and Shen 2008). This is computed as:

$$JC = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} \tag{1}$$

If sentences are viewed as sets then the JC can be defined as the number of unique words two sentences have in common over the total number of unique words in the two sentences. We have also modified JC by using a multiset approach that computes for a word appearing in the same sentence multiple times. For the remainder of the paper, we will use the term, *array*, when referring to a multiset.

The Common Noun Similarity module finds all the common nouns for each sentence and places them into arrays, cn_1 and cn_2 . CNS is thus defined as the number of common nouns the two sentences share over the length of the largest array. CNS is computed as:

$$CNS = \frac{|cn_1 \cap cn_2|}{max(|cn_1|, |cn_2|)} \tag{2}$$

The Proper Noun Similarity module finds all the proper nouns for each sentence and places them into arrays pn_1 and pn_2 . PNS is thus defined as the number of proper nouns the two sentences share over the length of the largest array(Chitra and Kumar 2010). PNS is calculated using the following:

$$PNS = \frac{|pn_1 \cap pn_2|}{max(|pn_1|, |pn_2|)} \tag{3}$$

The Verb Similarity module finds all the verbs for each sentence and places them into arrays, v_1 and v_2. VS is thus defined as the number of verbs the two sentences share over the length of the largest array. The formula for calculating VS is below:

$$VS = \frac{|v_1 \cap v_2|}{max(|v_1|, |v_2|)} \tag{4}$$

The Subject Similarity module finds all the common subjects for each sentence and places them into arrays, sub_1 and sub_2. SS is thus defined as the number of subjects the two sentences share over the length of the largest array. SS is computed as:

$$SS = \frac{|sub_1 \cap sub_2|}{max(|sub_1|, |sub_2|)} \tag{5}$$

The Object Similarity module finds all the common objects for each sentence and places them into arrays, obj_1 and obj_2. OS is thus defined as the number of objects the two sentences share over the length of the largest array. OS is computed as:

$$OS = \frac{|obj_1 \cap obj_2|}{max(|obj_1|, |obj_2|)} \tag{6}$$

Word Order

When determining if two sentences are similar it is also important to examine the structure of the sentence. Although two sentences share the same words, this does not mean that they are paraphrases of each other. We implemented two word order scoring modules: Maximum Common Subsequence and Longest Common Subsequence. We define a subsequence as a sequence of words that have been selected from a sentence and placed in an array in the same order as they appeared in the sentence.

The Longest Common Subsequence module finds every consecutive subsequence that two sentences have in common and places the length of these sequences into an array

of $scores$. LCS is thus defined as the maximum value in the scores array over the length of the longest sentence (Chitra and Kumar 2010). LCS is calculated as the following:

$$LCS = \frac{max(scores)}{max(|s_1|, |s_2|)} \quad (7)$$

The Maximum Common Subsequence module finds every subsequence that two sentences have in common and places the length of these sequences into an array of $scores$. It differs from the LCS in that it allows for words to be skipped as long as the common words in the resulting subsequence are in the same order. MCS is thus defined as the maximum value in the scores array over the length of the longest sentence (Hirschberg 1977). MCS is computed as:

$$MCS = \frac{max(scores)}{max(|s_1|, |s_2|)} \quad (8)$$

N-Grams

N-grams are a neighboring subsequence of n words from a given sentence. N-Gram modules compute commonality scores based on what n-grams they have in common using approximate string matching, a method for finding strings that match a particular pattern. We have implemented two N-gram scoring modules: Word N-Gram Overlap and Skip-grams.

WNGO is defined as 1 over N times the summation from n to N: where the numerator is defined as the number of n-grams that the two sentences have in common for a particular n and the denominator is defined as the length of the sentence with the most n-grams for a particular n. WNGO is computed as:

$$WNGO = \frac{1}{N} \sum_{n=1}^{N} \frac{|G_n(s_1) \cap G_n(s_2)|}{max(|G_n(s_1)|, |G_n(s_2)|)} \quad (9)$$

Where $Gn(s)$ is the set of n-grams of length n for sentence s and N is equal to 4 (Bernhard and Gurevych 2008).

Skipgrams use n-grams but allow words to be skipped within the subsequence (Chitra and Kumar 2010). Therefore, k-skip-n-grams allow a total of k or less skips to construct the n-gram. 2-skip-bi-grams were used in implementing this module.

$$SKIP = \frac{|sk_1 \cap sk_2|}{|sk_1 \cup sk_2|} \quad (10)$$

Classifier

To determine if the user's question and a candidate question taken from the conversation corpus are paraphrases we implemented a SVM classifier to label the sentence pair as a positive or negative example. The SVM was implemented using the LibSVM library using the RBF kernel (Chang and Lin 2011). Based on the user's question and the candidate question a feature vector is created using the NLP modules discussed earlier. The classifier uses this feature vector to determine whether the sentence pair is a paraphrase. Along with a class output label the classifier also generates a confidence score that gives us the probability that this particular instance belongs to the predicted class. As it relates to our system, the confidence score is a value between 0 and 1 and represents how confident the system is in its decision that the two sentences are positive or negative examples of a paraphrase. Using that confidence score we rank the questions from the corpus that have been labeled as a paraphrase when compared to the user's question. We then use the highest ranked question's paired answer as the response. If there are no questions classified as a paraphrase, meaning all the questions were classified as dissimilar, then the answer of "I do not know" is returned.

Trainer

To train the system to identify paraphrases we needed a large set of positive and negative sentence pairs to use in the training set. Since there exists a limited number of paraphrase training sets we created our own training set using multiple English translations of the book of Genesis in the Bible. The Bible was chosen for the following reasons:

- Among different versions of the Bible, semantically similar text existed.
- The organizational structure of the Bible allowed us to automate the process of aligning similar and dissimilar verses.

Using the 1,533 verses from Genesis we were able to create positive and negative sentence pairs. Our final training set consisted of 24,416 samples pairs labeled either *1* or *0* along with the feature vector for that sentence pair. The feature vector is a vector of scores computed using the modules described in the previous section. Out of the total 24,416 sentence pairs:

- 20,668 sentences were labeled *1's*
- 3,748 were labeled *0's*

To create the positive examples (*1's*), we took five different versions of the Bible stored in a text file and inserted verses into a database. We then aligned the different versions of the Bible by chapter and verse number. Chapter 1 verse 2 of the American Standard Version would align with Chapter 1 verse 2 of the King James Version. Since the two sentences were taken from two different versions of the Bible we knew that a large amount of the verses gathered would be expressed differently. Meaning, the words in the sentence would be rearranged, removed, inserted, and/or replaced with synonyms. However, the essential meaning of the two verses would be the same. We also knew that in using this technique we would also produce identical sentence pairs. From previous experiments, we learned that having identical sentence pairs in our training set helped in identifying paraphrases.

To create negative examples (*0's*) we aligned passages together that had different verse numbers or had the same verse number but came from a different chapter of Genesis. Chapter 26 verse 11 of the American Standard Version would align with Chapter 26 verse 14 of the King James Version.

Soft Filtering

During the classification step, most of the algorithm's computation time is spent in creating the feature vector that will be used by the classifier. In order to increase the speed of our system, we used an approach called *soft filtering* to perform classifications on only a subset of questions from the corpus. Soft filtering is a technique used to reduce a large set of potential answers to a smaller set of answers before passing the answers to more time consuming operations (Ferrucci et al. 2010). Since our questions and answers were stored in a MySQL database, we used the Full Text Search option in the database to return a ranked list of ten questions sorted in descending order of relevance as determined by the database. Only the top ten questions were passed to the classifier to be ranked based on their class output label and probability estimates.

System Evaluation

In order to evaluate the system's ability to return appropriate responses from the conversation corpus, we measured the system's overall accuracy, defined as the number of questions answered correctly over the number of questions asked in total. The number of questions answered correctly was separated into two categories: true positives (TP) and true negatives (TN). A TP occurs when the system returns an appropriate answer from the corpus. A TN occurs when the system returns "I do not know" and the appropriate answer to that question was not present in the corpus.

The number of questions answered incorrectly were also separated into two categories: false negatives (FN) and false positives (FP). A false negative occurs when the system returns "I do not know" when an appropriate answer was stored in the corpus. A FP occurs when the answer returned from the system is inappropriate. The FPs were separated into two categories, Type I and Type II false positives. Type I FPs occur when the system returns an incorrect answer from the corpus when an appropriate answer is present in the corpus. Type II FPs occur when the system returns an answer from the corpus when it should not have, because the true value was "I do not know". Accuracy is defined as the total of number of questions answered correctly out of the total number of questions asked. Accuracy is computed as:

$$Accuracy = \frac{TP + TN}{TP + TN + FPI + FPII + FN} \quad (11)$$

Precision is defined as the total of number of questions answered correctly out of the total number of answers the system returned that were from the corpus. Precision is computed as:

$$Precision = \frac{TP}{TP + FPI + FPII} \quad (12)$$

Recall is defined as the total of number of questions answered correctly out of the total number of questions asked where there was a correct answer stored in the corpus. Recall is computed as:

$$Recall = \frac{TP}{TP + FN + FPI} \quad (13)$$

For our data set, we began with a set of questions gathered from a group of nurses for a related project involving the development of a conversational agent. We chose this data set because we wanted to test Caesar on data created by real application area professionals. The agent was designed to help student nurses learn how to interview patients. From that original set of questions, variations were created from each question by potential users of the system. This process of taking an original set of questions and generating variations is similar to process performed by Leuski et al., when testing their conversational agent SGT Blackwell (Leuski et al. 2009). When creating a question variation words were rearranged, removed, inserted and/or replaced with synonyms.

Having the users create variations from an original set of questions produced semantically similar categories. Meaning, questions from the same category had similar meanings and therefore the same answer could be returned for each question in that category.

Before inserting variations of the original question into the corpus, we verified that the questions being inserted and the original question:

1. Had identical meanings
2. Could both return the same answer
3. Both presented alternate wording

These standards were derived from Bernhard and Gurevych's definition of a question paraphrase (Bernhard and Gurevych 2008). If a variation failed to meet any of these standards then it was removed from the category.

The nursing data set consisted of 172 unique question/answer pairs plus paraphrases of every unique question that, altogether, totaled 1,509 question/answer pairs. The number of questions in each category varied from 5 to 68. For each category there were a minimum of five different variations.

Using the questions from the different categories as test data, a program was written that went through each category and randomly shuffled the order in which the questions appeared in the file. After rearranging the order of the questions, the first half of questions was placed into the test data set and the second half into the conversational corpus. If there were an odd number of questions in a category, then the number of questions in that category was divided by two; with the ceiling of the total number of questions going into the corpus and the floor of the total number of questions going into the test set.

During the experiment, when the system returned a question from the same category as the input question, the output was marked as a correct response. Fifty different test sets were produced, creating 50 trials. Each trial randomly put approximately half (plus/minus 1 since categories could

Category	Caesar M(SD)	FTS M(SD)	p-value
True Positive	539.82 (9.99)	538.64 (9.26)	p >.05
True Negative*	87.04 (2.98)	0.4 (0.61)	p <.05
False Negative	9.08 (2.46)	0	p >.05
False Positive I*	143.10 (10.17)	153.36 (9.29)	p <.05
False Positive II*	12.96 (2.98)	99.6 (0.61)	p <.05
Accuracy*	79.15% (1.30)	68.06(1.18)	p <.05
Precision*	77.58% (1.46)	68.04 (1.17)	p <.05
Recall	78.01% (1.44)	77.84 (1.34)	p >.05

Table 2: Relative Performance of Caesar Compared to MySQL Full Text Search over 50 Trials. '*' signifies significant difference.

have an odd number of entries) of the questions in each category into the corpus and used the rest of the questions as the test set. Also, for each trial the system was asked 100 random questions for which there was no answer present in the corpus. This was done to test Caesar's ability to recognize when no answer was present in the corpus. These questions were chosen from a set of 1,000 questions that were picked from the web and were not related to the questions contained in the corpus. Thus, each test set consisted of 792 questions where 692 questions were randomly selected from the 172 categories and 100 questions were randomly selected from 1,000 questions chosen from the web .

Results and Discussion

In each of the 50 trials, 792 questions were asked. The means and standard deviations over all 50 trials for each category are shown in Table 2.

Caesar correctly recognized that a question was not in the corpus 87.04% of the time. Inappropriate answers to questions primarily consisted of False Positives of Type I. The small relative standard deviation for recognizing TPs and TNs also indicate that Caesar's results were very consistent over the 50 trials.

From Equation 11 the mean accuracy over all 50 trials was 79.15%. Caesar returned appropriate responses to questions whose answer was in the corpus 78.01% of the time (Equation 13). When selecting a response from the corpus 77.58% of the responses selected were correct (Equation 12).

There is no universal test set or universal standard that is used to judge the relative performance of conversation agents. There are also few papers in the literature that report implementation details and accuracy values for their approach to retrieving appropriate responses in a conversational agent. We could not make direct comparisons between Caesar and other systems since we did not have access to their test sets or even their method for judging accuracy. Since we used the MySQL Full Text Search (FTS) option to narrow down the candidate matches to ten before we sent them through the classifier, we compared our results

Category	Caesar M(SD)
False Negative	1.92(1.31)
False Positive I	16.64(4.17)
Total Error	18.56(4.35)

Table 3: Average number of errors caused by soft filtering over 50 trials.

to the top ranked question returned by FTS alone as a baseline comparison. These results are shown in Table 2.

The most obvious result was that FTS was extremely poor at recognizing that a question was not in the corpus (TNs) while Caesar correctly made this distinction 87.04% of the time. We found this to be an important result because it is better for the conversational agent to say it does not understand what is being asked and gather more information than to return a random answer which could potentially frustrate the user.

Using a Matched Pairs t-test on all categories, we found that Caesar improved performance significantly in recognizing TNs and significantly decreased the number of FPs of both types. When looking at accuracy and precision Caesar also significantly improved performance when compared to FTS. There was no significant difference between Caesar and FTS when it came to TPs and recall. Caesar returned significantly more FNs due to the fact that FTS almost never returned a FN response. Most (86.65%) of Caesar's inappropriate answers were Type I FPs. After analyzing the data we determined that the majority of the Type I FPs occurred when the user's question and the incorrect question it was matched to shared more words in common than the user's question and the appropriate question stored in the corpus.

As discussed in the previous section we used the technique of soft filtering to reduce the number of potential candidates to ten before moving to the machine learning classification stage. Since we used the soft filtering to reduce the amount of candidates it is possible for an appropriate answer not to be present in the top ten causing the machine learning algorithm to produce an error. If an appropriate answer is not contained in the top ten the system will either produce a Type I FP selecting the wrong response from the ten candidates or the system will decide that an appropriate answer was not present in the database producing a FN. Table 3 shows the average number of FN errors , Type I FP errors and the total number of errors caused by an appropriate answer not being present in the top ten. These numbers were calculated by counting the number of times an appropriate answer did not appear in the top ten for all questions that had an appropriate answer stored in the database. This was done for all 50 trials.

Using the data from Table 3 we adjusted the accuracy, precision and recall based on how well the machine learning algorithm performed alone. The adjusted accuracy is computed as:

$$Accuracy = \frac{TP + TN}{TP + TN + FPI + FPII + FN - TE} \quad (14)$$

Category	Caesar M(SD)
Adjusted Accuracy	81.05% (1.33)
Adjusted Precision	79.48% (1.50)
Adjusted Recall	80.16%(1.48)

Table 4: Adjusted Performance of Caesar over 50 Trials .

Let TE represent the Total Error.

Adjusted Precision is computed as:

Precision =

$$\frac{TP}{TP + FPI + FPII - SoftFilteringErrorFPI} \quad (15)$$

Adjusted Recall is computed as:

$$Recall = \frac{TP}{TP + FN + FPI - TE} \quad (16)$$

Table 4 shows the results from the adjusted accuracy , precision and recall scores .

On average it took Caesar four seconds to identify and select a response from the conversation corpus. Lester et al., suggested the total response time should be between one and two seconds (Lester, Branting, and Mott 2004). To reduce the total time it takes to return a response from the corpus we will modify the system so that actions will be performed in parallel which will lead to an in increase the speed of the system.

Conclusion

Our approach of creating a training set, in which we took five different versions of the book of Genesis from the Bible to create positive and negative paraphrase is novel in the area of paraphrase identification. We were able to use these paraphrases to create a training model which was used in later stages to test on a unrelated data set. When tested on 50 test sets consisting of 792 questions, Caesar returned a correct response 79.15% of the time (Equation 11). This included Caesar returning an appropriate response 78.01% of the time when an answer was present in the corpus (Equation 13). It also included Caesar correctly recognizing when a question was not in the corpus 87.04% of the time. When selecting a response from the corpus 77.58% of the responses selected were correct (Equation 12). We found that the majority (86.65%) of Caesar errors occurred when the system returned an incorrect answer from the corpus when appropriate answers were present (Type I FPs).

In analyzing Type I FPs we found that a large amount of these errors were due to the classifier's understanding of a positive and a negative example of a paraphrase. The modules used by the classifier are based on the number of words two sentences share. Therefore two sentences can be semantically similar and be labeled as not a paraphrase by the classifier because of the few words they share in common. The classifier can also label two sentences as a paraphrase because of the large amount of words they share in common when the sentences are actually dissimilar in meaning. To increase accuracy, our current work now focuses on incorporating semantic techniques that examine the meaning of words and how they relate to each other.

References

Achananuparp, P.; Hu, X.; and Shen, X. 2008. The evaluation of sentence similarity measures. *Data Warehousing and Knowledge Discovery* 305–316.

Bernhard, D., and Gurevych, I. 2008. Answering learners' questions by retrieving question paraphrases from social q&a sites. In *Proceedings of the Third Workshop on Innovative Use of NLP for Building Educational Applications*, 44–52. Association for Computational Linguistics.

Bloodworth, T.; Carico, L.; McClendon, J.; Hodges, L.; Babu, S.; Meehan, N.; and Johnson, A. 2012. Initial evaluation of a virtual pediatric patient system. In *Carolinas Women in Computing (CWIC)*.

Brockett, C., and Dolan, W. 2005. Support vector machines for paraphrase identification and corpus construction. In *Proceedings of the 3rd International Workshop on Paraphrasing*, 1–8.

Chang, C., and Lin, C. 2011. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2(3):27.

Chitra, A., and Kumar, C. 2010. Paraphrase identification using machine learning techniques. In *Proceedings of the 12th international conference on Networking, VLSI and signal processing*, 245–249. World Scientific and Engineering Academy and Society (WSEAS).

Dickerson, R.; Johnsen, K.; Raij, A.; Lok, B.; Hernandez, J.; Stevens, A.; and Lind, D. 2005. Evaluating a script-based approach for simulating patient-doctor interaction. In *Proceedings of the International Conference of Human-Computer Interface Advances for Modeling and Simulation*, 79–84.

Ferrucci, D.; Brown, E.; Chu-Carroll, J.; Fan, J.; Gondek, D.; Kalyanpur, A.; Lally, A.; Murdock, J.; Nyberg, E.; Prager, J.; et al. 2010. Building watson: An overview of the deepqa project. *AI Magazine* 31(3):59–79.

Hirschberg, D. 1977. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)* 24(4):664–675.

Lester, J.; Branting, K.; and Mott, B. 2004. Conversational agents. *The Practical Handbook of Internet Computing*.

Leuski, A.; Patel, R.; Traum, D.; and Kennedy, B. 2009. Building effective question answering characters. In *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*, 18–27. Association for Computational Linguistics.

Zhao, S.; Zhou, M.; and Liu, T. 2007. Learning question paraphrases for qa from encarta logs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1795–1801.