Question Classification Using Language Modeling

Wei Li
Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts, Amherst, MA 01003

ABSTRACT

Question classification assigns a particular class to a question based on the type of answer entity the question represents. In this report, I present two approaches: the traditional regular expression model, which is both efficient and effective for some questions but insufficient when dealing with others; and the language model, a probabilistic approach to solving the problem. Two types of language models have been constructed: unigram models and bigram models. Several issues are explored, such as how to smooth the probabilities and how to combine the two types of models. As expected, the language model outperforms the regular expression model. An even better result can be achieved by combining the two approaches together.

1. INTRODUCTION

Question answering is a variant of information retrieval, which retrieves specific information rather than documents. A QA system takes a natural language question as input, transforms the question into a query and forwards it to an IR module. When a set of relevant documents is retrieved, the QA system extracts an answer for this question. There are different ways of identifying answers. One of them makes use of a predefined set of entity classes. Given a particular question, the QA system classifies it into those classes based on the type of entity it is looking for, identifies entity instances in the documents, and selects the most likely one from all the entities with the same class as the question. So this approach involves two tasks. First, we should be able to identify named entities. This is a problem in the Information Extraction area [1], and we can make use of an existing entity tagger. Second, we need to classify questions into different classes, and this is the problem I addressed here.

One approach to question classification is to determine the question type based on the sentence structure and key words, which represent syntactic and semantic information respectively. A set of patterns are defined and hard-coded, often with regular expressions. When a new question comes, it is matched against those patterns to find the class it belongs to. As the pattern set gets more complete and accurate, the performance of this approach will become better. So to improve this model, we always have the problem of defining more and more question patterns.

To make the process of question classification more dynamic and automatic, we make use of language modeling, a statistical approach that has gained much attention recently in the IR area [2]. In this approach, the models can be automatically constructed from the training set, and its performance is competitive to other approaches. As for the QA task, we build one language model for every class of questions based on the training data set. To classify a question, the probability of generating it is calculated for each class based on its language model, and the highest probability determines the classification.

For the rest of this report, I will present the implementation of these two approaches and discuss their performance, with the focus on language modeling. In section 2, I will talk about two preparation steps: defining question classes and preprocessing questions before classification; section 3 is about the regular expression model: pros and cons; section 4 discusses language models two experiments and combination with the regular expression model; section 5 examines performance in different cases; section 6 introduces related work and section 7 is the conclusion.

2. PREPARATION

Defining question classes is the first step in classification. One important principal when defining these classes is that all the classes we use to mark questions should be recognizable as entities in the documents. This is because question classification is not an independent job, but a component for the QA task. Two kinds of classes are used. Some entity classes are naturally related to question classes, such as *person*, *location*, *number* and so on. Other classes are created for particular types of questions. For example, a frequently asked type of question is: "Who is sb.?" Typically people want to find quite detailed information about this person with this question. We don't have a good object class corresponding to this type of question, so we use the term *biography* to denote the type of answers for this question and add it into the question class set.

Another preprocessing step is to re-form the question to make its underlying pattern clearer. For example, the "Who is sb.?" questions always ask for a *biography* entity no matter what person's name appears in the question. In other words, the important thing is to know that this question contains a *person* entity. We do not care about the specific entity. So we can safely change questions of this pattern into "Who is <PERSON>?" without losing any information useful in determining the question type. What we actually do is to run an entity recognizer, the major part of which is IdentiFinder [3], against questions and replace all entities with their entity class names.

3. REGULAR EXPRESSION MODEL

The basic idea of this model is to determine a question type based on the sentence pattern, which includes the interrogative word, certain sequences of words and some representative terms of particular question classes. Those patterns are defined with regular expressions. For example, a question starting with "how many" is very likely to be looking for a *number*, and a question starting with "where" is probably a *location* question. For a "what" question, we can look for some key words to make our decision. For example, "agency", "company" and "university" are related to the *organization* class.

Here are some regular expressions used for certain classes of questions:

Questions that start with "what" and ask for a *person* entity:

(*actor*|*actresse*?|*attorne*(*y*|*ie*)|*teacher*|...|*senator*)*s*?

Questions that start with "how" and ask for a *length* entity:

long|*short*|*wide*|*far*|*close*|*big.**(*diameter*|*radius*)

This approach is very efficient and effective on some question patterns, such as "how many" questions. It seldom makes mistakes for this type of question. But there are difficult cases that it can hardly handle. For instance, the answer to a "who" question might be a *person*, an *organization*, and even a *location*. Let's take the question "Who is the largest producer of laptop computers in the world?" as an example. People can easily tell this is asking for an *organization*, but our program cannot decide its type just based on the question pattern. We need additional semantic

information, which is not available in the regular expression model. The same problem occurs with the "where" questions. Many "where" questions are classified as *location* while they are actually *organization* questions. The only way to solve this kind of problem is to build a more complete and accurate pattern set, which involves a great deal of human work. Instead of building a larger and larger question pattern model, we turned to a more automatic and flexible approach: language modeling.

4. LANGUAGE MODEL

The basic idea of language modeling is that every piece of text can be viewed as being generated from a language model. If we have two pieces of text, we can define the degree of relevance between them as the probability that they are generated by the same language model. In the information retrieval area, we build one language model for each document. Given a query, we can decide whether a document is relevant based on the probability that its language model generates such a query. Suppose that the query Q is composed of n tokens: $w_1, w_2, \ldots w_n$, and we can calculate the probability as:

$$P(Q|D) = P(w_1|D) * P(w_2|D, w_1) * \ldots * P(w_n|D, w_1, w_2, \ldots, w_{n-1})$$

So to build the language model on a document, we need to estimate those term probabilities.

Usually, a k-gram assumption is made to simplify the estimation:

$$P(w_i|D, w_1, w_2, \ldots, w_{i-1}) = P(w_i|D, w_{i-(k-1)}, w_{i-(k-2)}, \ldots, w_{i-1})$$

It means that the probability that w_i occurs in the document D will only depend on the preceding (k-1) tokens [4].

Similar ideas have been introduced into the question classification task. We build one language model for each category C of sample questions. When a new question Q comes, we calculate the probability $P(Q|C)$ for each C and pick the one with the highest probability. The major advantage of language model over the regular expression model is its flexibility. The regular expression model is composed of hard-coded rules, which need to be modified to handle new cases. The language model, however, can be automatically maintained. And we believe that, with larger sets of training data, the performance of the language model can be improved.

Two experiments have been conducted, and both of them include two language models: unigram and bigram models. The difference between them is the smoothing technique and the combination method. However, the two experiments provide similar performance. The details will be discussed below.

4.1 EXPERIMENT 1

The unigram and bigram models are the two simplest to construct, where

$$P(Q|C) = P(w_1|C) * P(w_2|C) * \ldots * P(w_n|C)$$

and

$$P(Q|C) = P(w_1|C) * P(w_2|C, w_1) * \ldots * P(w_n|C, w_{n-1})$$

respectively.

For the unigram model, we need to estimate the probability of a token w occurring in the category C, $P(w|C)$. Intuitively, it should be proportional to the term frequency $F(w|C)$. The tricky part is how to deal with tokens that never occurred in this category. We don't want them to have a probability of 0, so some probabilities must be assigned to them and the probabilities for other words will be adjusted accordingly. This kind of smoothing can be done in several ways, and for this experiment, we used an absolute discount method. A small

constant amount of probabilities is assigned to all 0-occurrence tokens, and the probabilities for other tokens will be subtracted accordingly [4]. Here is the formula:

Let *Total0(C)* be the number of 0-occurrence tokens in category C and S be the smoothing discount. So we have:

$$P(w \mid C) =$$

$$\begin{cases} F(w \mid C) * (1 - S) & \text{if} \quad F(w \mid C) \neq 0 \\ S / Total0(C) & \text{if} \quad F(w \mid C) = 0 \end{cases}$$

The bigram model is built similarly, where we need to estimate the conditional probability $P(w_2|C,w_1)$. Let $Total0(C,w_1)$ be the number of tokens that never occur after w_1 in category C, and S be the smoothing discount. There are two cases to consider:

Case 1: $F(w_1|C) \neq 0$, where the probabilities for all unseen w_2 is S. So we have:

$$P(w_2 \mid C, w_1) =$$

$$\begin{cases} F(w_2 \mid C, w_1) * (1 - S) & \text{if} \quad F(w_2 \mid C, w_1) \neq 0 \\ S / Total0(C, w_1) & \text{if} \quad F(w_2 \mid C, w_1) = 0 \end{cases}$$

Case 2: $F(w_1|C) = 0$, where all w_2 are unseen. So $P(w_2|C,w_1)$ should be the same for every w_2, which is calculated as follows:

$$P(w_2 \mid C, w_1) = 1 / Total0(C, w_1)$$

To make the estimation more accurate, we try to combine the two models together. Linear combination is a straightforward way, where

$$P(Q \mid C) = l P_u(Q \mid C) + (1 - l) P_b(Q \mid C)$$

Different values for l have been tested, and the best one is chosen.

4.2 EXPERIMENT 2

In this experiment, we still build the unigram and bigram models. But a different smoothing technique and combination method are used.

For the unigram model, we make use of Good-Turing [5] to estimate the probabilities for tokens that occur small numbers of times or never occur. According to the Good-Turing estimate, $P(w|C)$ should have the following structure:

$$P(w \mid C) =$$

$$\begin{cases} aF(w \mid C) & \text{if} \quad Count(w \mid C) > M \\ q_i & \text{if} \quad Count(w \mid C) = i \text{ and } 0 \le i \le M \end{cases}$$

The choices of a, q_i and M must satisfy:

$$\sum_w P(w \mid C) = 1 \quad \text{and} \quad q_{i-1} < q_i$$

There are several ways to derive the formula, and the result is as follows:

Let N be the size of the corpus, and $n_i(C)$ be the number of tokens that occur i times in C. Actually, we should use $E(n_i(C))$, the expected value for $n_i(C)$. But this value is not available, so we can only use the directly observed one instead.

$$P(w \mid C) =$$

$$\begin{cases} \left(\sum_{i>M+1} i n_i(C) \Big/ \sum_{i>M} i n_i(C) \right) F(w \mid C) & \text{if} \quad Count(w \mid C) > M \\ \left(n_{i+1}(C) \Big/ n_i(C) \right) \left(i + 1 \Big/ N \right) & \text{if} \quad Count(w \mid C) = i \text{ and } 0 \le i \le M \end{cases}$$

M is the largest number that satisfies:

$$(n_i(C))^2 < \frac{i+1}{i} n_{i-1}(C) n_{i+1}(C) \quad i = 1,..., M$$

and

$$\frac{n_{M+1}(C)}{n_M(C)} < \frac{\sum_{i>M+1} i n_i(C)}{\sum_{i>M} i n_i(C)}$$

While the unigram model is built based on the Good-Turing estimate, a Back-Off model [4] is developed for bigrams. The basic idea of the Back-Off model is that $P(w_2|C,w_1)$ should be proportional to $F(w_2|C,w_1)$ only when the occurrence of (w_1, w_2) in C is larger than a certain number. Otherwise, we just use $P(w_2|C)$ to estimate $P(w_2|C,w_1)$. Here's the formula:

$$P(w_2 \mid C, w_1) =$$

$$\begin{cases} aF(w_2 \mid C, w_1) & \text{if } Count(w_2 \mid C, w_1) > K \\ bP(w_2 \mid C) & \text{if } Count(w_2 \mid C, w_1) \le K \end{cases}$$

a is a discount to subtract the probabilities from large-occurrence bigrams, and we used the same discount as in the Good-Turing. ß is chosen for normalization:

$$\sum_{w_2} P(w_2 \mid C, w_1) = 1$$

It is a function of w_1.

K should be a small number, and we found that 0 provides the best performance for our data.

The Back-Off model naturally combines the unigram and bigram models. So to calculate the probability $P(Q|C)$, we can just use the bigram result, i.e.,

$$P(Q \mid C) = P_b(Q \mid C)$$

4.3 COMBINED WITH RE MODEL

Although the language model seems more attractive, it still has drawbacks. One of them is unpredictability. For example, as we do not have any restriction on the classification result of the language model, it is possible to classify a question that starts with "how many" as a *person* question. On the other hand, this kind of pattern is easy to capture by the regular expression model. So we tried to combine them to improve performance. The language model is modified to generate a ranked list of categories based on the belief score, and the regular expression model returns all categories compatible with the question pattern. The combination policy is that the category with the highest rank that is accepted by the regular expression model is the final answer. In this way, the mistake mentioned above will be avoided.

5. EVALUATION

A set of 693 TREC questions has been used for evaluation. They belong to the following classes:

Class Name	# of questions
PERSON	116
LOCATION	126
DATE	73
ORGANIZATION	64
NUMBER	74
OBJECT	121
REFERENCE	119

When testing the language models, we need a training set to build the models. So we did the experiments in the following way. The whole question set was randomly divided into five equally large disjoint parts. One part is chosen to be the test data, while the other four serve as the training data. Accuracy is calculated by comparing the classification result with the manually classified result. The same process has been repeated five times, each time a different test set is chosen. And the average accuracy is used to measure the performance.

Here are the test results for all the models discussed above:

Model		Accuracy
Regular Expression Model only		57.57%
Experiment1	LM only	81.54%
	LM combined with RE Model	85.43%
Experiment2	LM only	80.96%
	LM combined with RE Model	83.56%

The result shows that the language model performs better than the regular expression model, and the performance can be further improved if we combine them together. A little surprisingly, the language model in the first experiment outperforms the second one. We were expecting the reverse result since both the Good-Turing Estimate and the Back-Off Model have been shown to perform well in practice. One possible explanation is that our data set is insufficient to apply Good-Turing Estimate. As discussed above, we used $n_i(C)$ in places of $E(n_i(C))$. These two values should be close when the data set is large enough. But in our case, where there are only around 700 questions, this estimation might be quite bad.

6. RELATED WORK

Question classification is a common part in QA systems. The basic idea is the same: to classify questions and identify corresponding entities in documents, but it can be achieved in different ways. Many systems use techniques that are similar to the regular expression model just mentioned.

MURAX is an earlier QA system that makes use of an online encyclopedia [6]. Its heuristic is simple: to classify questions based on the interrogative words. And for "what" questions, which may ask for several types of entities, the encyclopedia is searched for the noun phrase after "what" and the question type is determined accordingly.

Another QA system using named entities and question classification is the GuruQA system described by Prager [7]. It maintains a set of patterns and compares questions with them to determine their types. The question type is used as a query term and the documents have been processed to add types to the named entities. In this way, the document containing a named entity with the same type of the question is more likely to be retrieved.

7. CONCLUSION

Question answering differs from information retrieval in that it needs to retrieve specific fact information rather than whole documents. This might involve excessive computation if there is no guidance for possible answers. By classifying questions and named entities into the same set of classes, we can eliminate a large amount of irrelevant information.

This report has investigated two approaches for question classification: regular expression model and language modeling. The regular expression model is a simplistic approach and has been put into practice in many systems. Language modeling is a probabilistic approach imported from IR systems. The models are constructed in a more flexible and automatic way. We have built two types of models: a linear combination of unigram and bigram models with an absolute-discount smoothing technique; and a Back-Off bigram model with Good-Turing estimate.

The test result shows that the language model outperforms the regular expression model. And an even better result can be achieved when the two models are combined together. Although Good-Turing and Back-Off models have been proved effective in practice, the

second language model doesn't improve the performance over the first one.

ACKOWLEDGMENTS

This material is based on work supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #EIA-9983215.

The author would like to thank David Pinto, Bruce Croft, Andres Corrada-Emmanuel and David Fisher for their help and support.

Any opinion, findings and conclusions or recommendations expressed in this material are the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] R. Srihari and W. Li, "Information Extraction Supported Question Answering".

[2] J. M. Ponte and W. B. Croft, "A Language Modeling Approach to Information Retrieval".

[3] BBN official site about the IdentiFinder: http://www.bbn.com/speech/identifinder.html.

[4] C. Manning, H. Schutze and H. Schutze, "Foundations of Statistical Natural Language Processing".

[5] F. Jelinek, "Statistical Methods for Speech Recognition".

[6] J. Kupiec, "MURAX: A Robust Linguistic Approach For Question Answering Using An On-Line Encyclopedia".

[7] J. Prager, E. Brown and A. Coden, "Question-Answering by Predicative Annotation".