Building a Question Classifier for a TREC-Style Question Answering System

Richard May & Ari Steinberg

Topic: Question Classification

We define Question Classification (QC) here to be the task that, given a question, maps it to one of k classes, which provide a semantic constraint on the sought-after answer [Li02]. The topic of Question Classification arises in the area of automated question-answering systems, such as those created for the TREC question answering competition. Automated question-answering systems differ from other information retrieval systems (ie. search engines) in that they do not return a list of documents with possible relevance to the topic, but rather return a short phrase containing the answer. Moreover, question answering systems take in as input queries expressed in natural language rather than the keywords traditional search engines use.

In order to respond correctly to a free form factual question given a large collection of texts, any system needs to understand the question to a level that allows determining some of the constraints the question imposes on a possible answer. These constraints may include a semantic classification of the sought after answer and may even suggest using different strategies when looking for and verifying a candidate answer. More specifically, knowing the class (or possible classes) of the sought after answer narrows down the number of possible phrases/paragraphs a question-answering system has to consider, and thus greatly improves performance of the overall system. [Harabagiu] divides their QA system into three distinct pieces. At the core of the first module in the system lies a question classification task. Thus, it seems that question classification is an important subtask of automated question answering. An error in question classification will almost undoubtedly throw off the entire QA pipeline, so it is crucial to be able to classifiy question correctly.

In our paper, we will build on the hierarchal classification discussed in [Li02] and experiment with some features of our own design. We expect that by tweaking both the classification algorithms and the choice of features, we can get improvements in this crucial QA subsystem.

Classification

Classification Methodology

Results from [Li02] demonstrated that using a flat classifier performs just as well as a two-layer hierarchical classifier that used a coarse classifier to dispatch the classification task to a second classifier. We too, plan on using a hierarchal classifier, however, ours will differ in that we will also attempt to learn which classes are often confused, whereas [Li02] applied domain knowledge to the task and created six hand crafted super classes that then contained distinct subsets of the true classes.

Classification System

For our code base, we leverage an existing machine learning library, MALLET, discussed in [McCallum]. We designed a hiearchal classifier trainer, which takes in a training set and partitions it into a base train set and an advanced train set. The trainer then uses the base train set to train the coarse classifier over all the possible question types. The trainer then tests the coarse classifier on the advanced training data, to build a confusion matrix. Using a set of threshold parameters, the trainer decides if certain predicted classes have too high a confusion rate and then trains a secondary classifier on the advanced training instances that were predicted to be part of the high confusion rate classes.

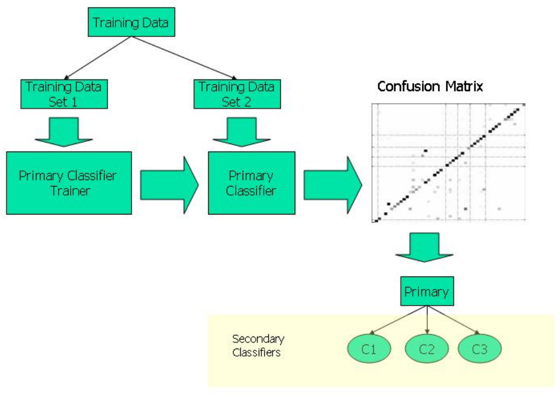

Figure 1: Diagram of how to train a hierarchal classifier

Features

The majority of our creative energies were focused into feature engineering. It is a process that requires a lot of trial and error. We intend to leverage resources in WordNet to improve on semantic understanding of the questions.

As input to our machine learning algorithm, the computer examines each question and derives a representation consisting of numerous features. In the end, a typical question can have as many as 60 features, and our set of 6000 total questions can result in anywhere from 30,000 to 120,000 unique features. While it would be easy to generate far more than this, due to memory constraints we must be careful in selecting only the most useful types of features to include in this set. In addition, adding extraneous features can add noise to the data and result in weaker performance. On the other hand, large performance improvements can be gained by adding useful features to the set.

Basic features

The most basic representation of a question is simply the individual words of that question (ignoring contextual information such as the ordering of the words). While simple, this is also by far the most important part of our program - the best indicators of certain question types are single words, and in particular question words clearly reveal a lot about the type of question being asked. On the other hand, a large amount of information would be lost by stopping here, since words can often mean many different things depending on their context.

We can regain some of this contextual information by examining part of speech tags. We run a parser over the question and take the preterminal nodes of the parser as the parts of speech for each word. These parts of speech alone wouldn't help much, so we add as features word and part of speech pairs, thus helping to disambiguate words which have different senses depending on their part of speech.

Another problem that part of speech tags fail to address is feature sparsity (in fact, part of speech and word pairs suffer even further from this problem). Some words can reveal a lot of information, but do not show up enough times in the training set to allow our classifier to pick up on this. To address this problem, we use a stemmer to create a more generalized form of the word. For example, whether a verb is in present or past tense might not impact how it affects the question being asked.

Another basic feature that we add is bigrams - pairs of words that occur sequentially. However, bigrams inherently suffer even worse from sparsity problems than individual words, so we use the stemmed form of the words to create the bigrams.

One final basic feature that we experimented with is to take conjunctions of any other two basic features. Bigrams do not capture long-distance dependencies - two words that may affect each other's meanings but are not adjacent in the sentence, so we attempted to do that with these conjunctions. The conjunctions did help to improve accuracy, but they resulted in unmanageably large feature set sizes, so we could not run them with the larger training corpuses or with the help of our full feature set.

Parse Signatures

A more advanced feature that we added is something we call a "parse signature", though the term "sentence reduction" may be a more accurate descriptor. Our goal with these signatures was to create a representation of the entire question structure, instead of just small fragments of it. Obviously, though, adding the full sentence would only help to later classify that exact sentence, so we needed a way of generalizing this. Another motivation for creating these signatures was to attempt to have some way to represent the rich grammatical information given to us by the question's parse tree, but again without suffering from the sparsity issues that would come with using the entire tree as a feature.

A parse signature can be thought of as a left-to-right readout of a parse tree. More formally, it is a set of nodes from a parse tree such that every leaf node has exactly one ancestor (or itself) in the set. The question itself would be one such signature, and the tree's root node would be another (though neither of those would be particularly useful for our purposes). It should be clear that the number of parse signatures that can be generated from a question is enormous, so we need a way of limiting this number. Our solution is to parameterize the signature generation with the desired length of the readouts. Choosing to include all signatures of lengths 1 to 5 captures the information that we desire while keeping the number of features at a reasonable level.

We can further reduce the noise of this data by choosing to not traverse below certain nodes. For example, our parse trees always end in a "." node which expands to a "?", but since nearly every question ends with a ? this means we would always be generating two almost-identical versions of each signature - one ending in . and one ending in ?. To avoid this, we tell the signature generator to never traverse below a . or some other parts of speech such as CC whose exact words seem to have little impact.

WordNet

Thus far, all of our features have focused on syntactic considerations, but there is clearly a lot of information to be gained by looking at the semantic information in the questions. In order to accomplish this, we add WordNet information to our features. For every noun, verb, adjective, and adverb, we also add the first synonym that appears in its "synset" (a group of words that WordNet tells us share a meaning). While we could add every synonym, as long as we are careful to choose the same synonym each time something in a particular synset occurs there is no reason to add more than one.

Perhaps more interesting than synonyms, though, are hypernyms. This is the "... is a kind of ..." relationship (going from more specific to more general), e.g. "animal" for "dog". While the idea of a more generalized form of a word is clearly a useful one, just adding a word's direct hypernym probably wouldn't be very helpful, since this may not be general enough (we might get that a dog is a canine and a cat is a feline instead of finding that they are both animals). Going multiple levels up also does not help, because two words can be at different levels in the tree, so while they may share some ancestor, it may be 2 levels up for one word and 3 for another and we would again not see the similarity. One solution would be to add all of a word's hypernyms, but this would result in too much data.

Our solution was instead to always take the hypernyms a certain distance from the "root" node. For example, many nouns have "entity" as their root node, but knowing that a noun is an entity isn't much better than knowing that it's a noun. We start at the root (such as entity) and work down several nodes in the direction of the target word until we get to our fixed level and choose that as the hypernym to add. This way we are more likely to have related words share a hypernym, even if they are at different levels of the WordNet tree. We add such a hypernym to our feature set for every word that falls occurs below the target level.

Finally, this hypernym information fits nicely into our parse signature idea. The signature "What causes pneumonia", for example, is probably too specific, but having "What causes NN" may be too general. If we instead had "What causes HYP_disease" (inserting the HYP_ prefix to distinguish the hypernym from the base word) as our signature this may be more ideal. We insert the hypernyms into the tree between a word's part of speech and the word itself. For words that do not have hypernyms (because they are the wrong part of speech or occur too close to the root node in the WordNet hierarchy) but do have stemmed forms we add the stemmed form instead. Again, this has the potential to dramatically increase the number of parse signatures, so we tell our signature generator to never traverse below a node with an HYP_ or STM_ prefix.

Evaluation & Results

Dataset

In order to compare our results with previous work, we used the same dataset as [Li02]. The data can be obtained at the groups website (`http://l2r.cs.uiuc.edu/~cogcomp/Data/QA/QC/`). The training data consists of 5500 labelled questions. The test data is 500 questions taken from the TREC 10 set. In both the training and test data, there are a total of 50 different question classes. In addition, we count an answer as correct if the best classification label output by our classifier is the true label.

Hierarchal Classification Results

After experimenting with a variety of classifiers (SVM, MaxEnt, NB, Decision Tree) for primary and secondary classification, we decided that a mix of a Maximum Entropy course classifier with a Naive Bayes fine classifier was the best combination. Observations indicated that it was better to mix classifiers than to have the same type of classifier as both the primary and secondary.

As for why a Naive Bayes secondary classifier had the best results, we can offer some intuition. By the time we reached the secondary classification level, our training data's feature sets were extremely sparse in the size of the feature space. Thus we would want a classifier that could take in a lot of features while still being very general. The exponential family of classifiers are known to perform better than some of the other classifiers we experimented with given sparse training data.

Unfortunately our best hierarchal classifiers still under-performed compared to the flat maximum entropy classifier we train. Under the best conditions, we choose to split the training data with 75% used for the course classifier and 25% for the fine classifiers.

Training Set	1000	2000	3000	4000	5500
Flat Classifier	67.6	74.2	77.8	80.2	82
Hierarchal Classifier	64.6	71	75.8	77.8	81

Examining our performance data, we realize that the main loss of performance is due to reduced amount of training data fed into the primary classifier. If we rescale the training set axis of the hierarchal classifier to reflect the use of only 75% of the training data for the course classifier, we see that the hierarchal classifier does quite well in comparison to the flat classifier.

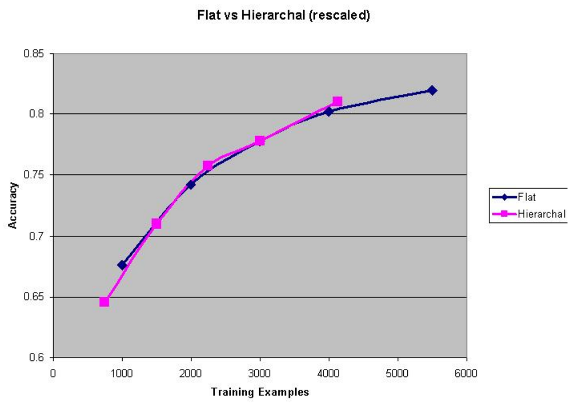

Figure 2: Performance comparison with rescaled hierarchal training set size.

Feature Set Results

Our biggest performance boosts came from using Porter's word stemmer, provided by [Stanford NLP] and the additional semantic categories provided by WordNet. We were disappointed that the parse signatures yielded only a small boost (0.2%) with the full training set, but expect that further performance gains may be achieved by further adjusting the parameters used to generate the signatures, as well as working to further incorporate the WordNet data into the signatures. Our performance without WordNet (80%) exceeds the performance achieved by [Li02] (78.8%) without adding related words, so we expect that it should be possible to exceed [Li02] if we experimented more with the semantic information.

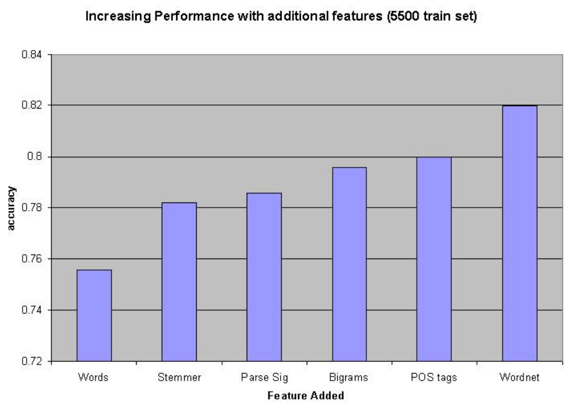

Figure 3: Performance increase with each added feature.

Comparison with Li & Roth

Our system was able to perform better than [Li02] at the 4000 training example level. However, where our system came up short was in its ability to combine features conjunctively or disjunctively. [Li02] demonstrated use of a system for expressing complex conjunctions & disjunctions of features without experiencing the combinatorial explosion often associated with this field, by cleverly selecting only useful pairs. Implementing such a combinatorial system was beyond the scope of our time constraints, since it would require a great deal of hand-tweaking.

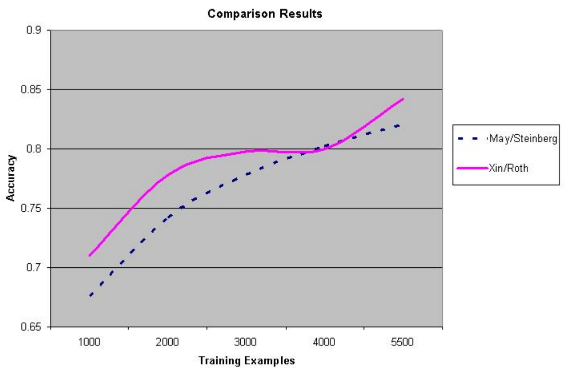

Figure 4: Performance Comparison

Summary & Future Work

With regards toward applying a hierarchal classifier to the task of question classification, both our results and [Li02] indicate that using a flat classifier is better. However, given a sufficiently large training corpus, we still believe that a hierarchal classifier should do no worse than a flat classifier because the secondary classifier could at the very least just always output the same label as the course classifier. At the present levels of training data, however, the difference in the training corpus sizes is too great to overcome.

During initial testing with smaller feature sets, we did experience some performance increases by combining features as pairs. With a greater amount of memory or a clever way of pruning the pairs that get added, we are confident that we could significantly improve the performance of our system. In addition, we believe that the WordNet information in particular has far more potential to be explored. [Li02] was able to achieve an 8% performance increase by adding information about related words, compared to our 2% gain. If we had additional time to improve our utilization of WordNet we could expect to get accuracy greater than that of [Li02].

References

[Li02] Xin Li, Dan Roth. Learning Question Classifiers. COLING. 2002.

[Harabagiu] Sanda M. Harabagiu, Marius A. Pasca, Steven J. Maiorano. "Experiments with Open-Domain Textual Question Answering"

[McCallum] McCallum, Andrew Kachites. "MALLET: A Machine Learning for Language Toolkit." http://mallet.cs.umass.edu 2002.

[Stanford NLP] JavaNLP. http://nlp.stanford.edu