A MAXIMUM ENTROPY SEMANTIC PARSER USING WORD CLASSES

Norbert Pfannerer

SAIL LABS Technology AG, Operngasse 20B, A-1040 Vienna, Austria
norbert.pfannerer@sail-technology.com

ABSTRACT

This paper describes the parser that is used in the Sail Labs Conversational System, which is a spoken dialog system. This parser is a fully statistical, semantic parser. The probability model of the parser is based on the principle of maximum entropy. The maximum entropy framework allows to combine the available information in a fully automatic way, but the training of maximum entropy models is time consuming. Since the parser needs to be retrained when its vocabulary changes, a straightforward application of this model cannot realistically be used in a dialog system. To solve this problem, words can be combined to classes, and the classes can be used instead of the words for the training of the parser. At runtime, words can be added to the classes at no cost.

1. INTRODUCTION

Most current speech understanding systems either use grammars or rules of some sort, often combined with probabilistic methods, or use a fully statistical approach such as decision trees. In syntactic parsing, a fully statistical approach that has gained some attention is the principle of maximum entropy [1,2]. This method allows the available information (encoded into the so called features) to be used in a principled way. The feature set can easily be modified without special treatment depending on the type of feature.

This paper describes the parser that is used in the Sail Labs Conversational System. This is a complete dialog system consisting of a speech recognizer, the parser, a dialog manager, an answer generation and a text to speech system. More information can be found at http://www.sail-technology.com/.

The parser is a fully statistical semantic parser, which uses a maximum entropy probability model. A drawback of maximum entropy models is the large amount of resources (both time and memory) needed to train the models.

This is especially a problem when the parser is used in a dialog system where frequent updates of the vocabulary are required, such as in a name dialing application, where frequent changes to the list of names are to be expected. Each vocabulary update requires the parser model to be retrained, which is impossible in a production environment.

In addition there is the problem of consistency. The training set needs to be representative of the application domain. If names are added later, additional sentences need to be added to the training set. These sentences will be duplications of already existing sentences with only the names changed, or at least they will be similar to already existing sentences. In this scenario time-consuming manual work is required to keep the training set consistent, yet small enough to allow the training in reasonable time.

To overcome these problems the parser was designed to be able to deal with word classes. A word class is simply a set of words, which can be modified without the need to retrain the parser model.

2. BUILDING THE PARSE TREE

The input to the parser is a sentence, i.e. a string of words, which is typically the output of a speech recognizer. The parser then builds a parse tree for this sentence, using semantic tags and labels. The parse tree is then transformed into a sequence of label-value pairs that are further processed by the dialog manager. Fig. 1 shows an example of a parse tree from a name dialing application.

The parse tree is built in a sequence of steps, where each step either assigns a tag or a label to a node in the tree, or builds a new node by joining a sequence of existing nodes to a new sub tree. These steps are applied to the word sequence in a left to right, bottom to top order until the tree is complete.

The probability model assigns a conditional probability to each step $s_i$,

$$p(s_i | context).$$

where $context$ is a partial parse tree.

The probability of the parse tree $T$ is the product of the conditional probabilities (1) of all the steps that build the tree,

$$p(T | <w>) = \prod_{i} p(s_i | context_{i-1}),$$

where $<w>$ is the input word string and $context_i$ is the partial parse tree built up to step $s_i$. The first context is simply the list of input words, $context_0 = <w>$.

3. THE STATISTICAL MODEL

The probability model (1) is based on the principle of maximum entropy [4,5,6]. This method makes use of a set of so called features $f_j$, $1 \leq j \leq k$, which are functions on the space $X$ of pairs of all partial parse trees and steps, i.e. on all pairs of the form ($context$, $s$).

The features usually take as values 0 or 1 (so called binary features), but generalizations are possible. Binary features can
be viewed as a condition that is or is not satisfied. A feature is usually a condition on the current step, i.e. "does the current step assign the label FIRSTNAME?", and a condition on the partial tree that is already built, such as "is the label to the left TRANSFER?". The conditions on the partial trees can make use of the words, the already assigned tags and labels, and the available tree structure.

The probability model is the probability distribution \( p \) on \( X \) that maximizes the entropy

\[
H(p) = - \sum_{x \in X} p(x) \log p(x),
\]

subject to the constraints that the feature expectations remain constant, i.e.

\[
Ef_j = \tilde{Ef}_j, \quad 1 \leq j \leq k,
\]

where the feature expectations are given by

\[
Ef_j = \sum_{x \in X} p(x)f_j(x),
\]

and the observed feature expectations are

\[
\tilde{Ef}_j = \sum_{x \in \text{Training}} \tilde{p}(x)f_j(x)
\]

and \( \tilde{p}(x) \) denotes the relative frequency of the training sample \( x \). The resulting probability model has the form

\[
p(x) = \alpha_0 \prod_{j=1}^{k} \alpha_j^{f_j(x)}.
\]

Note that there is a one to one correspondence between the model parameters \( \alpha_j \) and features \( f_j \). \( \alpha_0 \) is a simple scaling factor. It does not have to be estimated because when conditional probabilities (1) are calculated it drops out. The training algorithm first calculates the \( \tilde{Ef}_j \), then the parameters \( \alpha_j \) can be found by using the Generalized Iterative Scaling method [3], or by Improved Iterative Scaling [6,7]. These algorithms require many iterations over the training corpus, and therefore are costly in terms of running time and memory consumption.

4. THE SEARCH ALGORITHM

The parser tries to search for the tree \( T^* \) that has the maximal probability for the given word sequence \( <w> \).

\[
T^* = \arg \max_T p(T \mid <w>),
\]

where the probability of the parse trees is given by equation (2).

The search algorithm is a breath-first search over all possible sequences of steps, but keeps only the top \( K \) scoring partial parses (sometimes called a beam search). To do this it keeps a list of partial parses that is initialized with \( <w> \). Iteratively all possible steps are carried out on the partial parse trees until the tree is complete. For each step the probability is calculated using equation (7) and the partial parse trees are scored and sorted according to (2).

5. USING WORD CLASSES

The feature set of the parser includes features that take into account the words. Based on some simple feature templates (e.g. "is the word to the left equal to \( x \)?") the training module builds all features that fit the templates and actually occur in the training data.

Dealing with word classes can be done by checking only a substring of the word instead of the whole word, e.g. "does the word end with \( ing \)?", but this method is more useful in a syntactic parser than in a semantic one. Also this does not solve the problems with the names, because names usually don’t follow common spelling features.

To overcome this problem, the parser knows about word classes. A word class is simply a set of words that should be treated the same way, e.g. all last names. To make use of the classes, the training data is modified to contain the class names.

\[\alpha_j\]
instead of the words (see Fig. 2). For example the sentence “can I speak to mister Norbert Pfannerer please” might be replaced by “can I speak to mister [FIRSTNAME] [LASTNAME] please” without change to the tree structure.

It is also possible to use fixed classes that are not used to add words later on, for example a list of weekdays or the names of the months. Fig. 2 uses as an example a class for the salutation. This reduces the amount of training sentences, because without the class all sentences containing one of these words would have to written individually for all words of the class. Therefore the necessary training time and the memory is reduced. This also makes it easier for the dialog designer to create a representative and consistent training set, which is crucial in order to build a robust and working model for real world application.

5.1. Building class features

For the above mentioned feature template and the example sentence in Fig. 1 the parser will generate features such as “is the word to the left equal to speak?”, “is the word to the left equal to to?”, and “is the word to the left equal to Norbert?” When the names are replaced by the classes, instead of the last example the feature “is the word to the left equal to [FIRSTNAME]?” will be generated. This feature is used during the training in the same way a word feature is used.

5.2. Using the class features for the parsing

When parsing an input sentence, the parser needs to evaluate all features at every step to calculate the scores of the partial parse trees according to equations (2) and (7). In addition to the input sentence, the features $f_j$, and the corresponding model parameters $\alpha_j$, the parser needs the set of words that are in each class.

When evaluating a class feature, the parser needs to check whether the corresponding word is in the respective class. As an example see the partial parse tree in Fig. 2. The next action of the parser is to assign a label to the empty node. In considering the label PERSON, the parser will evaluate e.g. the feature “label = PERSON and the current subtree contains the class [FIRSTNAME]”. If the input sentence was “can I speak to mister Norbert Pfannerer please” and the word Norbert is in the class [FIRSTNAME], then the feature will evaluate to true.

The respective word features are active at the same time if the word itself also appeared in the training data. Therefore it is necessary to completely replace the words by the classes in the training treebank. Of course the best way to ensure this is to design the treebank with the classes in mind from the beginning.

5.3. The special class $\{X\}$

In a spoken dialog system the parser has to deal with the special problem of misrecognition. Two types of errors that occur often, but don’t necessarily change the meaning of the utterances, are substitutions of short function words, and insertions that often happen at the beginning or end of the sentence due to hesitations or other non-speech events.

Using class features, it is possible to increase the robustness of the parser against such errors. The method to do this is not to include function words at all in the training set if these words are not necessary for the understanding of the meaning. Instead all those function words and other irrelevant words are replaced by one class.

Sometimes the speech recognizer produces a word that is not in the vocabulary of the parser. To deal with this situation, the class $\{X\}$ behaves differently than normal classes. All other classes contain a fixed list of words, that are either predefined, like the names of the weekdays, or that come from a database. The names of all employees for a name dialer.

In contrast the word list for class $\{X\}$ is not fixed. When the parser reads an input string, it checks each word whether it is in its vocabulary. All new words are added to the vocabulary and to the special class $\{X\}$. All word features are inactive for such words, because they were never seen in training. After the new words have been added to class $\{X\}$, the class features for this class are active, and enable the parser to deal with these words.

The drawback of this method is that good statistics for the class $\{X\}$ must be available. This is only the case if it appears often enough in the training corpus. Often it is necessary to introduce new training sentences that contain one or more instances of $\{X\}$ at the beginning and end.

5.4. Updating the classes

In a real world application the content of the classes needs to be regularly updated. The way to do this obviously depends on the application, but usually the list of words is retrieved from a database.

Examples are the names of people to be used in a name dialer, or the list of pizza toppings for a pizza ordering application. Updating the contents of the classes does not require a retraining of the parser model.
6. EXPERIMENTAL RESULTS

An important question is how to evaluate a statistical parser. One commonly used set of measures is the PARSEVAL measures [8]. These include the measures precision and recall. The parse trees are broken down into the labels and the range of words associated with them (often called brackets).

Precision measures the percentage of labels in the candidate parse that are correct, recall measures how many labels of the correct parse were found. In a spoken dialog system, depending on the application and on the dialog design, it is often only relevant whether a label appear and not which words are associated with it. Therefore two different precision measures were used, one that takes the words into consideration and one that only measures the correct labels without looking at the words.

In general a semantic parser is not easy to evaluate. No measurements on standard corpora were performed with this parser. Measurements on some dialog applications vary largely. These measurements can often be used to judge the quality of the training treebank, and unless the training treebank is consistent and large enough don’t actually measure the quality of the parser.

For the above mentioned reasons no formal evaluation of the parser is available. With various applications, we have seen precision and recall values as high as 99%, but a bad treebank design or too few training sentences with too little variation can easily bring the values down to 50-60%.

Use of classes usually increases the recall, because the treebank gets more consistent, and the parser effectively sees more training examples with fewer sentences.

The effect of the use of the class [X] is difficult to measure. It is necessary to record a test set using real examples of speech recognizer output. Such a test set not only has the problem of being application dependent, it also ages relatively quickly due to improvements to the speech recognizer. In general it seems to improve the accuracy on test sentences, but slight errors in the consistency of the treebank can have the adverse effect.

More tests are needed in this area to guide future research.

7. CONCLUSIONS

The described parser is used in a dialog system for real world applications. This fact and the above mentioned informal results clearly show that a maximum entropy statistical model is a valid approach for a semantic parser.

Despite the drawback that classes have to be used consistently to be of any help, they are crucial in real world applications, where the vocabulary constantly changes.

The advantage of the class [X] is not so clear. Some increase in robustness is observed, but at the increased cost of building the treebank correctly. It still seems often to be an advantage to use it, especially when the recognition rate of the speech recognizer is low, e.g. over noisy telephone lines.

8. REFERENCES