Using Simultaneous Training to Improve the Performance of Both a Verb Subcategorization Model and a Word Sense Model

Teg Grenager and Galen Andrew

June 7, 2002

1 Introduction

Linguists have long suggested that the sense, or lemma, of a verb is the appropriate locus of subcategorization. Indeed, Roland and Jurafsky (1998) conclude that for a given lexical token in English, sense is the best determiner of subcategorization frame, far outweighing either genre or dialect. [6, 9]. Conversely, it seems reasonable to expect that the subcategorization frame of a verb instance may provide valuable information for the word sense disambiguation task. Clearly the subcategory alone is highly indicative of the sense, but perhaps even more useful would be the leverage provided by knowing the subcategory in identifying the specific arguments and adjuncts of the verb. Having determined a verb's arguments, one could, for instance, predict sense using a naive Bayes classifier that conditions on words from each argument, or use them to identify other verbs that occur in similar lexico-semantic contexts in a memory-based classifier, as in Lin (1997) [7].

Unfortunately, in practical cases where one of these two tasks is required, we rarely have access to the information provided by the other. Even if we had systems that could predict sense given subcategory and subcategory given sense, neither could function to its full potential unless the other had already performed its task.

Our SynSense system addresses this concern by combining the two systems and using an iterative approach, whereby progressively more accurate estimates of subcategory and sense are given by the two subsystems as they repeatedly pass each other their current best guesses and then recompute their estimates using the new information.[1] Although the initial estimates would be crude, as the subsystems cannot on the first iteration use all of the information they are designed to take advantage of, one would expect that these original estimates would still provide valuable information to the other subsystem, allowing it to produce a better estimate on the next iteration. The process can conclude when

[1] An interesting variation would be for the subsystems to exhange not only their best guess, but their entire probability distribution over guesses.

some termination condition is satisfied, for example if neither system changes its estimate in some iteration, or if a certain fixed number of iterations has passed.

The problem then becomes how to train the two subsystems. Using supervised learning techniques to train either one would require having a large amount of training data that had been marked for both verb-subcategory and sense. Such data sets do exist, for instance the small portion of the Brown corpus that exists in processed form in both Semcor and Treebank.[2] However this small corpus provides nowhere near the quantity of data needed for training a robust system. One might imagine using unsupervised or partially unsupervised methods for training one or both of the subsystems, but instead we took a different approach.

In order to address this issue, we decided to train both of our subsystems simultaneously in a manner that loosely resembles the *co-training* regimen described by Mitchell [2, 1]. This training regimen utilizes two training sets, one marked for marked for verb sense, and one marked for verb-subcategory. In the first step, an initial word sense model which does not condition on verb subcategory is trained on the set that is annotated for sense. This model is then used to classify and label the sense of verbs in the other training set, which is annotated for verb subcategorization. This set can in turn be used to train the verb subcategorization model. Note that the verb subcategorization model can now condition on the senses of the target verbs. We close the loop by using the verb subcategorization model to classify the subcategories of the verbs in the word sense corpus, and extract the arguments and adjuncts of the verb. These are valuable features that the word sense model can condition on during the next training phase. This loop of training on one corpus and marking up the other is repeated until neither of the classifications produced by the model change.

Readers familiar with machine learning may notice that the this training regimen is analogous to the well-known *Expectation-Maximization* (EM) algorithm. EM is often used to estimate the values of unknown parameters or missing data in training sets. As the name suggests, EM consists of two steps. In the first step, the algorithm calculates the expected likelihood of the model for different values of the missing data, given some (perhaps poor) current estimate of the missing data. In the second step, the algorithm computes the values of the missing data that would maximize this expected likelihood, and uses them as the estimated values in the next iteration. Our algorithm operates similarly, but with the "missing data" divided into two (or more generally k) disjoint sets. In each step one of the classifiers estimates the likelihood of its missing data given some (perhaps poor) current estimates for the remainder of the missing data (training). Then it selects the values for this data that maximize these likelihood estimates (classification).

SynSense was not intended to be competitive with state-of-the-art word-sense disambiguators or verb subcategorizers. Our goals are rather

[2]Treebank is not explicitly marked for subcategory, but this information can be easily inferred. Also it is a simple matter to extract verb arguments from a parse tree.

1. to demonstrate that multiple systems that make use of each other's proficiencies can perform better by cooperating than they could working on their own, and

2. to provide further evidence that co-training several specialist systems at once can be an effective way of dealing with the lack of adequate amounts of training data with all of the necessary annotations

Hopefully, these results apply to hybrid systems for any set of NLP problems, not just word-sense disambiguation and verb subcategorization.

2 Algorithms and Design

Our system divides neatly into three interacting parts. There are the two classifying subsystems, and a master system that controls their interaction. Most of these three systems' functionality is independent of the others. Their design is therefore discussed separately (at a medium-high level) below. In addition, there are several utility classes to perform such tasks as selecting training and test sets which will not be discussed further.

2.1 The Verb Subcategorization Subsystem

Determining verb subcategory is tantamount to parsing a verb phrase, which essentially entails parsing the entire enclosing sentence if the boundaries of the verb phrase are not clearly delimited. Not seeing any other way around this difficulty, our approach was to learn individual grammars for each sense of each verb that reflect its various subcategorization probabilities. These tailored grammars are then used by a chart parser (written by Dan Klein and Chris Manning) to parse sentences containing the verb. The verb subcategory and arguments can then be extracted from the resulting parse tree.

Designing the individually tailored grammars is a tricky problem. Essentially, for each verb sense, we need a grammar in which the verb phrase that includes the target verb is parsed according to a learned probability distribution over subcategorization frames. Furthermore, all subtrees that do not contain the target verb should parse as they would with a grammar learned from general English.

We proceed as follows:

Step 1. (off-line) Mark every verb phrase in the training corpus for subcategory. This is done over every verb in *the entire corpus*, not just those specifically under consideration, to obtain more accurate general probabilities of the various expansions of a verb phrase of a given subcategory.

Step 2. (off-line) Propagate markings toward the root of each tree. Thus if we have a tree that uses a transition ($\mathrm{A} \longrightarrow \mathrm{B}\ \mathrm{VP}_2$) where A and B are nonterminals and VP_2 is a verb phrase with subcategory #2, we relabel A as A_2. This records the fact that the nonterminal A can give rise to a verb phrase of the second categorization frame. If there are several verb phrases below some nonterminal, it is tagged for each one (with duplications, if necessary). Also a special tag "START" is put at the top of the tree in order to keep track of the number of times some nonterminal is the root of a tree that contains a verb of subcategory s.

Step 3. (off-line) Compute maximum likelihood estimates of transitions from each nonterminal, considering separately each case where it leads to a particular verb subcategory and the case in which it does not lead to a verb. In test sets, there will be only one verb to subcategorize in each instance. So we essentially consider the training instance multiple times, once for each verb in the sentence. For example, if the subtree rooted at some nonterminal A contains k verbs, all of subcategory s, and there are n verbs total in the sentence, it is considered as k instances of transitions from A_s to nodes marked with s, and $n-k$ instances of transitions from A to nodes not marked for subcategory. Once these weighted counts have been made, the MLE of each transition for each sense is computed in the straightforward way:

$$\Pr(\mathrm{X} \longrightarrow \mathrm{Y}) = \frac{\mathrm{count}(\mathrm{X} \longrightarrow \mathrm{Y})}{\mathrm{count}(\mathrm{X})}$$

Step 4. (on-line) Armed with this "base grammar" which needs to be computed only once, we can proceed to produce a subcategorizing grammar once we've estimated the prior probabilities each subcategorization frame for a particular sense of the target verb. Our original method was simply to add *another* nonterminal, "ROOT", that transitions to each "START" node with probability equal to the (current estimate of the) probability of the corresponding subcategory for a particular sense of the verb. The problem with this approach is that it makes the grammar intractably huge—essentially the size of a standard (nonsubcategorized) grammar times the number of subcategories (in our case 16) plus one! In the end we decided to make individual grammars for each subcategory, and let the parser determine the probability of the best parse under the assumption that the verb subcategorizes a particular way. Then we weight those probabilities by the prior probabilities of each subcategorization to determine the most likely subcategory.

We also compute the MLE that each tagged preterminal gives rise to each verb form and use this to construct an individualized lexicon.

Given a parse of the sentence (either as data in the training set or as a result of parsing a test instance with our custom grammar), the problem of how to determine the subcategory of the verb and extract its arguments remains.

Roland (2001) [8] uses tgrep expressions to cull from Treebank all instances of a verb being used with a certain subcategorization. His expressions are designed so that "non-verb" uses of the word (for example in a gerundive noun phrase), or uses of the verb with a particle that are in fact separate verbs, do not match any expression. Every instance containing a "legal" use of the verb matches exactly one tgrep string corresponding to one of his subcategories, which may be the "other" subcategory.

We used Roland's tgrep expressions to identify the subcategory of a verb in a parsed tree. We had to make a few modifications to rule out cases of "to have", "to be" and "to do" as auxiliaries. (Roland just did not include these verbs in his study. We wanted to use them if only for estimating transition probabilities for each subcategorization frame irrespective of the verb.) Also, Roland's expressions were written for Treebank I style parses, while we were running our tests on Treebank III. This necessitated some other changes since, for instance, Treebank III has a broader definition of sentential components.

Each tgrep expression contains subexpressions for each argument that a verb phrase of a particular subcategory must contain, which provides a convenient way to locate arguments. Our subcategorizer separates the argument subexpressions and first tries to match the sentence to the part without arguments. If it matches successfully, we attempt to match the argument expressions to the children of the matched verb phrase. If all arguments are matched, we have identified the subcategory and located the arguments. We then also identify the subject of the verb (if one exists), which is not required by any subcategory but is almost always present in English. Finally, all children of the verb phrase that have not been identified as arguments are considered adjuncts. All of this matching is *not* performed by the tgrep utility, which would require outputting the trees to disk and converting them to tgrepable format. Instead we have a class that recognizes the subset of tgrep syntax used in our tgrep expressions and can be called on Tree objects in memory.

2.2 The Word Sense Disambiguation Subsystem

Word sense disambiguation programs can use any number of possible feature sets, and a range of possible model representations. Not surprisingly, our word sense disambiguation subsystem is tailored specifically to the task of classifying the sense of verbs for which syntactic information is available. In our system we condition the probability of the sense of the verb on the estimated subcategory of the verb, as well as on the particular words contained in the arguments and adjuncts to that verb. Clearly the verb's sense, subcategory, argument words, and adjunct words can be described best by a giant joint probability distribution. However, representing this full distribution is not tractable, and we instead make some strong independence assumptions. In particular, we represent the following conditional probability distributions as independent.

$$\begin{array}{l}
\Pr(\text{subcat} \mid \text{sense}) \\
\Pr(v_i \mid \text{type}(\text{Arg}), \text{sense}) \forall v_i \in \text{Arg} \\
\Pr(w_i \mid \text{sense}) \forall w_i \in \text{Adj} \\
\Pr(w_i \mid \text{sense}) \forall w_i \in \text{Sent}
\end{array}$$

These are the probability of the verb subcategory given the sense, the probability of a particular word occuring within a verb argument given the type of argument and the verb sense, the probability of a particular word occuring within a verb adjunct given the verb sense, and the probability of a particular word occuring anywhere in the sentence given the verb sense, respectively.

We also strongly considered estimating separately the probability of particular words occuring as the heads of the arguments and adjuncts. However, this proved impractical as incorporating a head-finding algorithm would have required nontrivial effort. (Even though the functionality may be already be implemented in the parsing package we used, understanding the API with limited documentation seemed daunting.) Thus we broadened our model to estimate the probability of *any* word in the arguments and adjuncts to the verbs.

To estimate the true probability of a novel instance given a sense during classification, we combine a subset of these probability distributions. If the instance has complete syntactic information, we multiply together the first three distributions. If instead the instance has not been subcategorized, we use only the latter distribution which considers only the general context of the words in the sentence.

To combat data sparseness problems in the word models, we utilize a form of Good-Turing smoothing [5]. This smoothing is needed to give a non-zero probability of seeing novel words in a particular distribution. Given the small size of the contexts, this smoothing is very important. We revise the probability estimates for unseen words, and for all words that were seen less than $r_{\max}$ times, where $r_{\max}$ is a parameter that is determined dynamically for each distribution. $r_{\max}$ is based on local properties of the count-counts distribution for low frequency words. The probability estimates of the high frequency words are not smoothed, since it can be argued that the frequency counts already form good estimates of the true probabilities.

We also incorporated the potential to weight separately the different probability distributions of the model when performing classification. However, we did not attempt to find optimal settings for these weights.

2.3 The Co-Training Subsystem

The co-training subsystem contains all of the logic for the training and testing of the two classifiers described above. In particular, it implements the co-training algorithm that was described in the introduction. It also implements more conventional training and testing regimens for each of the classifiers independently, giving us something to which to compare the performance of the co-training algorithm.

All of the algorithms begin by loading the files containing the sense and subcategorization training data, and separating them into training and testing instances. Then we initialize the supporting data structures, which involves a surprising amount of work due to the need to initialize the parser. Then, in the co-training algorithm, we create a new subcategory classifier and sense classifier. We first train the sense classifier on the sense training data, and then use it to classify instances in the subcategory training data. We then train the subcategory classifier on the subcategory training data, and use it to classify the subcategory of the instances in the sense training data, while at the same time extracting the arguments and adjuncts to the target verb. This cycle of training is performed iteratively, at each step creating and training new classifiers on the improved data. The loop ends when the training sets have observed no change in their classifications.

To test the co-trained classifiers, we employ a similar iterative algorithm. First the sense classifier classifies the sense, and then the subcategory classifier uses the sense information to classify the subcategory. Next the sense classifier uses the subcategory, argument, and adjunct information to classify the sense. This loop ends when the test sets have observed no change in their classifications.

The conventional training regimens consist of a single iteration of training on a training set and classifying a test set.

The results of the classification on the test set are then printed to the standard output.

3 Code Structure

The classes for SynSense are organized into four packages according to subsystem: `subcatclassifier`, `senseclassifier`, `cotrain` and `util`. The classes in each package and their functions are breifly described here. For further information, please consult the java documentation located at `$SUBMISSIONDIRECTORY/index.html`.

3.1 package synsense.subcatclassifier

`Subcategorizer` is the master subcategorization class. By calling the methods of `TgrepMatcher`, it determines the subcategory of an `Instance` and identifies its arguments and adjuncts, placing this information in the `Instance` for use by the sense classification subsystem. All regular expression strings are `static final` members of this class.

`TgrepMatcher` performs the nitty-gritty of tgrep expression matching. The bulk of the code in this class is concerned with interpreting tgrep strings. There are also procedures for identifying arguments and adjuncts and one for locating the subject.

In the preprocessing phase of custom grammar production, it is necessary to label every verb in a sentence with its subcategory and propagate this information upward. The class `SubcatMarker` perfoms this task, and also has a method for removing those tags after a parse so that the tgrep expressions will still work properly.

3.2 package synsense.senseclassifier

`VerbSenseClassifier` is the only class in this package. As the name suggests, it implements the verb sense disambiguation subsystem. It contains all of the data structures necessary to maintain estimates of the probability distributions that are used in the disambiguation task. It has two important public methods, one to train the classifier on an `InstanceSet` of instances, and one to use the classifier to classify an `InstanceSet` of test instances.

3.3 package synsense.util

`MakeSenseTrainSet` and `MakeTreebankTrainSet` are simple utilities to extract a set of instances from Semcor and Treebank files respectively. Both find only instances that contain some form of the verb (in the syntactic role of a verb) and do not contain any quoted expressions (because our tgrep expressions are not designed to handle quotes). Optionally they can take an argument specifying the maximum number of words in training sentences to cut down on long sentences that take an inordinate amount of time to parse.

`Instance` is a structure used to store all information relevant to either classification subsystem so it can be passed back and forth conveniently as a unit during the training and testing processes. It holds the words in the sentence, the word to be disambiguated/subcategorized (in the same form as it is in the sentence), its subcategory, sense and position in the sentence, and lists of words in each argument or adjunct subtree. Naturally at some points in the process some of these variables hold default, unassigned values. Also the `public static final` codes for subcategory types and argument types are defined here.

Finally `Instance Set` is a container for `Instance`s containing the same verb that also stores the multiple forms that the word might take and keeps track of whether some Instance changed in the last iteration for use in the training termination condition. In addition, if it is being used as a test set, the correct results are stored in a separate vector.

3.4 package synsense.cotrainer

This package at present contains a single class, `Cotrainer`. `Cotrainer` contains the methods for the main training and testing algorithms. It interacts with objects of type `InstanceSet`, `SubcatLexGramMaker`, `VerbSubcategoryClassifier`, `SenseClassifier` to read in from instance files, perform cotraining, and evaluate the trained system.

4 Results and Discussion

First, note that we thoroughly debugged and tested all of the functionality of our program. For very few instances, we have confirmed that the subcategorizer does create an appropriate grammar and lexicon, and does parse sentences correctly given the grammar and lexicon. We have verified manually that the parses seem to reflect accurately the subcategorization probabilities with which the grammar and lexicon were created. We have also observed that the subcategorizer next assigns relatively accurate subcategories to the instances. The verb sense classifier has also been tested extensively, and seems to correctly estimate the probability distributions that it maintains, and classify instances according to them.

Unfortunately, however, we are not able to report results of the co-training algorithm. To perform even a few iterations of the algorithm requires a large number of sentences parses, and empirically we have found that parsing each sentence (even those with less than 20 words) can take several minutes. We attempted to performed co-training and testing runs on several verbs used in studies involving verb subcategory by Garnsey *et al.* [4] and Connine *et al.* [3], in particular the verbs *hear*, *know* and *want*. Even when we restricted the training and test sets to have a minimal number of instances to provide any degree of statistical reliability (i.e., 10) and confined the data sets to sentences of twelve words or less, only a few iterations could be done in the space of several hours. At the time of this submission, we have been running on a small (not statistically relevant) training and test set for over 200 minutes. We have confirmed, however, that the algorithm is making incremental progress, however slow, and is not stuck in an infinite loop. We are disappointed in this lack of results.

References

[1] D. Bikel. A statistical model for parsing and word-sense disambiguation, 2000.

[2] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 1998.

[3] C. Connine, F. Ferreira, C. Jones, C. Clifton, and L. Frazier. Verb frame preference: Descriptive norms. *Journal of Psycholinguistic Research*, 13:307–319, 1984.

[4] S. Garnsey, N. Pearlmutter, E. Myers, and M. Lotocky. The contributions of verb bias and plausibility to the comprehension of temporarily ambiguous sentences. *Journal of Memory and Language*, 37:58–93, 1997.

[5] I. Good. The population frequencies of species and the estimation of population parameters. *Biometrica*, 40:237–264, 1953.

[6] G. Green. *Semantics and Syntactic Regularity.* Indiana University Press, Bloomington, Indiana, 1974.

[7] D. Lin. Using syntactic dependency as local context to resolve word sense ambiguity. In *Meeting of the Association for Computational Linguistics*, pages 64–71, 1997.

[8] D. Roland. Verb sense and verb subcategorization probabilities. Technical report, University of Colorado, 2001.

[9] D. Roland and D. Jurafsky. Verb sense and verb subcategorization probabilities. In *Merlo, P., & Stevenson, S. (Eds.), Volume from CUNY-98. Benjamins, Amsterdam.*, 1998.