Semantic Role Labeling using libSVM

Necati Ercan Ozgencil
Center for Natural Language Processing
School of Engineering and Computer Science
Syracuse University
neozgenc@ecs.syr.edu

Nancy McCracken
Center for Natural Language Processing
School of Information Studies
Syracuse University
njm@ecs.syr.edu

Abstract

We describe a system for the CoNLL-2005 shared task of Semantic Role Labeling. The system implements a two-layer architecture to first identify the arguments and then to label them for each predicate. The components are implemented as SVM classifiers using libSVM. Features were adapted and tuned for the system, including a reduced set for the identifier classifier. Experiments were conducted to find kernel parameters for the Radial Basis Function (RBF) kernel. An algorithm was defined to combine the results of the argument labeling classifier according to the constraints of the argument labeling problem.

1 Introduction and Strategy

The Semantic Role Labeling (SRL) problem has been the topic of the both the CoNLL-2004 and the CoNLL-2005 Shared Tasks (Carreras and Màrquez, 2005). The SRL system described here depends on a full syntactic parse from the Charniak parser, and investigates aspects of using Support Vector Machines (SVMs) as the machine learning technique for the SRL problem, using the libSVM package.

In common with many other systems, this system uses the two-level strategy of first identifying which phrases can be arguments to predicates in general, and then labeling the arguments according to that predicate. The argument identification phase is a binary classifier that decides whether each constituent in the full syntax tree of the sentence is a potential argument. These potential arguments are passed into the argument labeling classifier, which uses binary classifiers for each label to decide if that label should be given to that argument. A post-processing phase picks the best labeling that satisfies the constraints of labeling the predicate arguments.

For overall classification strategy and for suggestions of features, we are indebted to the work of Pradhan et al (2005) and to the work of many authors in both the CoNLL-2004 shared task and the similar semantic roles task of Senseval-3. We used the results of their experiments with features, and worked primarily on features for the identifying classifier and with the constraint satisfaction problem on the final argument output.

2 System Description

2.1 Input Data

In this system, we chose to use full syntax trees from the Charniak parser, as the constituents of those trees more accurately represented argument phrases in the training data at the time of the data release. Within each sentence, we first map the predicate to a constituent in the syntax tree. In the cases that the predicate is not represented by a constituent, we found that these were verb phrases of length two or more, where the first word was the main verb (carry out, gotten away, served up, etc.). In these cases, we used the first word constituent as the representation of the predicate, for purposes of computing other features that depended on a relative position in the syntax tree.

Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL),
pages 205–208, Ann Arbor, June 2005.

We next identify every constituent in the tree as a potential argument, and label the training data accordingly. Although approximately 97% of the arguments in the training data directly matched constituents in the Charniak tree, only 91.3% of the arguments in the development set match constituents. Examination of the sentences with incorrect parses show that almost all of these are due to some form of incorrect attachment, e.g. prepositional attachment, of the parser. Heuristics can be derived to correct constituents with quotes, but this only affected a small fraction of a percent of the incorrect arguments. Experiments with corrections to the punctuation in the Collins parses were also unsuccessful in identifying additional constituents. Our recall results on the development directory are bounded by the 91.3% alignment figure.

We also did not use the the partial syntax, named entities or the verb senses in the development data.

2.2 Learning Components: SVM classifiers

For our system, we chose to use libSVM, an open source SVM package (Chang and Lin, 2001).

In the SRL problem, the features are nominal, and we followed the standard practice of representing a nominal feature with n discrete values as n binary features. Many of the features in the SRL problem can take on a large number of values, for example, the head word of a constituent may take on as many values as there are different words present in the training set, and these large number of features can cause substantial performance issues.

The libSVM package has several kernel functions available, and we chose to use the radial basis functions (RBF). For the argument labeling problem, we used the binary classifiers in libSVM, with probability estimates of how well the label fits the distribution. These are normally combined using the "one-against-one" approach into a multi-class classifier. Instead, we combined the binary classifiers in our own post-processing phase to get a labeling satisfying the constraints of the problem.

2.3 The Identifier Classifier Features

One aspect of our work was to use fewer features for the identifier classifier than the basic feature set from (Gildea and Jurafsky, 2002). The intuition behind the reduction is that whether a constituent in the tree is an argument depends primarily on the structure and is independent of the lexical items of the predicate and headword. This reduced feature set is:

Phrase Type: The phrase label of the argument.

Position: Whether the phrase is before or after the predicate.

Voice: Whether the predicate is in active or passive voice. Passive voice is recognized if a past participle verb is preceded by a form of the verb "be" within 3 words.

Sub-categorization: The phrase labels of the children of the predicate's parent in the syntax tree.

Short Path: The path from the parent of the argument position in the syntax tree to the parent of the predicate.

The first four features are standard, and the short path feature is defined as a shorter version of the standard path feature that does not use the argument phrase type on one end of the path, nor the predicate type on the other end.

The use of this reduced set of features was confirmed experimentally by comparing the effect of this reduced feature set on the F-measure of the identifier classifier, compared to feature sets that also added the predicate, the head word and the path features, as normally defined.

	Reduced	+ Pred	+ Head	+ Path
F-measure	81.51	81.31	72.60	81.19

Table 1: Additional features reduce F-measure for the identifier classifier.

2.4 Using the Identifier Classifier for Training and Testing

Theoretically, the input for training the identifier classifier is that, for each predicate, all constituents in the syntax tree are training instances, labeled true if it is any argument of that predicate, and false otherwise. However, this leads to too many negative (false) instances for the training. To correct this, we experimented with two filters for negative instances. The first filter is simply a random filter; we randomly select a percentage of arguments for each argument label. Experiments with the percentage showed that 30% yielded the best F-measure for the identifier classifier.

The second filter is based on phrase labels from the syntax tree. The intent of this filter was to remove one word constituents of a phrase type that was never used. We selected only those phrase

labels whose frequency in the training was higher than a threshold. Experiments showed that the best threshold was 0.01, which resulted in approximately 86% negative training instances.

However, in the final experimentation, comparison of these two filters showed that the random filter was best for F-measure results of the identifier classifier.

The final set of experiments for the identifier classifier was to fine tune the RBF kernel training parameters, C and gamma. Although we followed the standard grid strategy of finding the best parameters, unlike the built-in grid program of libSVM with its accuracy measure, we judged the results based on the more standard F-measure of the classifier. The final values are that C = 2 and gamma = 0.125.

The final result of the identifier classifier trained on the first 10 directories of the training set is:

Precision: 78.27% Recall: 89.01%
(F-measure: 83.47)

Training on more directories did not substantially improve these precision and recall figures.

2.5 Labeling Classifier Features

The following is a list of the features used in the labeling classifiers.

Predicate: The predicate lemma from the training file.

Path: The syntactic path through the parse tree from the argument constituent to the predicate.

Head Word: The head word of the argument constituent, calculated in the standard way, but also stemmed. Applying stemming reduces the number of unique values of this feature substantially, 62% in one directory of training data.

Phrase Type, Position, Voice, and Subcategorization: as in the identifier classifier.

In addition, we experimented with the following features, but did not find that they increased the labeling classifier scores.

Head Word POS: the part of speech tag of the head word of the argument constituent.

Temporal Cue Words: These words were compiled by hand from ArgM-TMP phrases in the training data.

Governing Category: The phrase label of the parent of the argument.

Grammatical Rule: The generalization of the subcategorization feature to show the phrase labels of the children of the node that is the lowest parent of all arguments of the predicate.

In the case of the temporal cue words, we noticed that using our definition of this feature increased the number of false positives for the `ARGM-TMP` label; we guess that our temporal cue words included too many words that occured in other labels. Due to lack of time, we were not able to more fully pursue these features.

2.6 Using the Labeling Classifier for Training and Testing

Our strategy for using the labeling classifier is that in the testing, we pass only those arguments to the labeling classifier that have been marked as true by the identifier classifier. Therefore, for training the labeling classifier, instances were constituents that were given argument labels in the training set, i.e. there were no "null" training examples.

For the labeling classifier, we also found the best parameters for the RBF kernel of the classifier. For this, we used the grid program of libSVM that uses the multi-class classifier, using the accuracy measure to tune the parameters, since this combines the precision of the binary classifiers for each label. The final values are that C = 0.5 and gamma = 0.5.

In order to show the contribution of the labeling classifier to the entire system, a final test was done on the development set, but passing it the correct arguments. We tested this with a labeling classifier trained on 10 directories and one trained on 20 directories, showing the final F-measure:

10 directories:	83.27
20 directories:	84.51

2.7 Post-processing the classifier labels

The final part of our system was to use the results of the binary classifiers for each argument label to produce a final labeling subject to the labeling constraints.

For each predicate, the constraints are: two constituents cannot have the same argument label, a constituent cannot have more than one label, if two constituents have (different) labels, they cannot have any overlap, and finally, no argument can overlap the predicate.

	Precision	Recall	$F_{\beta=1}$
Development	73.57%	71.87%	72.71
Test WSJ	74.66%	74.21%	74.44
Test Brown	65.52%	62.93%	64.20
Test WSJ+Brown	73.48%	72.70%	73.09

Test WSJ	Precision	Recall	$F_{\beta=1}$
Overall	74.66%	74.21%	74.44
A0	83.59%	85.07%	84.32
A1	77.00%	74.35%	75.65
A2	66.97%	66.85%	66.91
A3	66.88%	60.69%	63.64
A4	77.66%	71.57%	74.49
A5	80.00%	80.00%	80.00
AM-ADV	55.13%	50.99%	52.98
AM-CAU	52.17%	49.32%	50.70
AM-DIR	27.43%	56.47%	36.92
AM-DIS	73.04%	72.81%	72.93
AM-EXT	57.69%	46.88%	51.72
AM-LOC	50.00%	49.59%	49.79
AM-MNR	54.00%	54.94%	54.47
AM-MOD	92.02%	94.19%	93.09
AM-NEG	96.05%	95.22%	95.63
AM-PNC	35.07%	40.87%	37.75
AM-PRD	50.00%	20.00%	28.57
AM-REC	0.00%	0.00%	0.00
AM-TMP	68.69%	63.57%	66.03
R-A0	77.61%	89.73%	83.23
R-A1	71.95%	75.64%	73.75
R-A2	87.50%	43.75%	58.33
R-A3	0.00%	0.00%	0.00
R-A4	0.00%	0.00%	0.00
R-AM-ADV	0.00%	0.00%	0.00
R-AM-CAU	100.00%	50.00%	66.67
R-AM-EXT	0.00%	0.00%	0.00
R-AM-LOC	66.67%	85.71%	75.00
R-AM-MNR	8.33%	16.67%	11.11
R-AM-TMP	66.67%	88.46%	76.03
V	97.32%	97.32%	97.32

Table 2: Overall results (top) and detailed results on the WSJ test (bottom).

To achieve these constraints, we used the probabilities produced by libSVM for each of the binary argument label classifiers. We produced a constraint satisfaction module that uses a greedy algorithm that uses probabilities from the matrix of potential labeling for each constituent and label. The algorithm iteratively chooses a label for a node with the highest probability and removes any potential labeling that would violate constraints with that chosen label. It continues to choose labels for nodes until all probabilities in the matrix are lower than a threshold, determined by experiments to be .3. In the future, it is our intent to replace this greedy algorithm with a dynamic optimization algorithm.

3 Experimental Results

3.1 Final System and Results

The final system used an identifier classifier trained on (the first) 10 directories, in approximately 7 hours, and a labeling classifier trained on 20 directories, in approximately 23 hours. Testing took approximately 3.3 seconds per sentence.

As a further test of the final system, we trained both the identifier classifier and the labeling classifier on the first 10 directories and used the second 10 directories as development tests. Here are some of the results, showing the alignment and F-measure on each directory, compared to 24.

Directory:	12	14	16	18	20	24
Alignment	95.7	96.1	95.9	96.5	95.9	91.3
F-measure	80.4	79.6	79.0	80.5	79.7	71.1

Table 3: Using additional directories for testing

Finally, we note that we did not correctly anticipate the final notation for the predicates in the test set for two word verbs. Our system assumed that two word verbs would be given a start and an end, whereas the test set gives just the one word predicate.

References

Xavier Carreras and Lluìs Màrquez, 2005. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling, Proceedings of CoNLL-2005.

Chih-Chung Chang and Chih-Jen Lin, 2001. LIBSVM : a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Daniel Gildea and Daniel Jurafsky, 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics* 28(3):245-288.

Sameer Pradhan, Kadri Hacioglu, Valerie Krugler, Wayne Ward, James H. Martin, and Daniel Jurafsky, 2005. Support Vector Learning for Semantic Argument Classification, To appear in *Machine Learning* journal, Special issue on Speech and Natural Language Processing.