Memory-based semantic role labeling: Optimizing features, algorithm, and output

Antal van den Bosch, Sander Canisius, Iris Hendrickx
ILK / Computational Linguistics
Tilburg University, P.O. Box 90153,
NL-5000 LE Tilburg, The Netherlands
{Antal.vdnBosch,S.V.M.Canisius, I.H.E.Hendrickx}@uvt.nl

Walter Daelemans, Erik Tjong Kim Sang
CNTS / Department of Linguistics
University of Antwerp, Universiteitsplein 1,
B-2610 Antwerpen, Belgium
{Walter.Daelemans, Erik.TjongKimSang}@ua.ac.be

1 Introduction

In this paper we interpret the semantic role labeling problem as a classification task, and apply memory-based learning to it in an approach similar to Buchholz et al. (1999) and Buchholz (2002) for grammatical relation labeling. We apply feature selection and algorithm parameter optimization strategies to our learner. In addition, we investigate the effect of two innovations: (i) the use of sequences of classes as classification output, combined with a simple voting mechanism, and (ii) the use of iterative classifier stacking which takes as input the original features and a pattern of outputs of a first-stage classifier. Our claim is that both methods avoid errors in sequences of predictions typically made by simple classifiers that are unaware of their previous or subsequent decisions in a sequence.

2 Data and Features

The CoNLL-2004 shared task (Carreras and Màrquez, 2004) supplied data sets for the semantic role labeling task with several levels of annotation apart from the role labels to be predicted. Central to our approach is the choice to adopt the instance encoding analogous to Buchholz et al. (1999) to have our examples represent relations between pairs of verbs and chunks. That is, we transform the semantic role labeling task to a classification task in which we decide for all pairs of verbs and chunks whether they stand in a semantic role relation. Afterwards we consider all adjacent chunks to which the same role label is assigned by our classifier as belonging to the same argument. All results reported below use this task representation. Processing focuses on one verb at a time; verbs are treated independently.

We did not employ the provided Propbank data nor the verb sense information available, nor did we use any other external source of information.

Apart from the provided words and the predicted PoS tags, chunk labels, clause labels, and named-entity labels, provided beforehand, we have considered an additional set of automatically derived features:

1. attenuated words (Eisner, 1996), i.e. wordforms occurring below a frequency threshold (of 10) are converted to a string capturing some of the original word form's features (capitalization, whether it contains numbers or a hyphen, or suffix letters);
2. the distance between the candidate role word and the verb, measured in intervening words, chunks, NP chunks or VP chunks (negative if the word is to the left, positive if it is to the right of the verb);
3. preceding preposition: a feature containing the head word of the previous chunk if that was labeled as preposition;
4. passive main verb: a binary feature which is on if the main verb is used in a passive construction;
5. current clause: a binary feature which is on if the current word is in the same clause as the main verb;
6. role pattern: the most frequently occurring role pattern of the main verb in the training data (contains the order of `V` and `A0-A5`).

For every target verb in every sentence, the data supplied to the learners contains instances for every head word of non-VP chunks and for all words in VP chunks, and all words in all chunks containing a target verb (i.e., more instances than chunks, to account for the fact that some roles are contained within chunks). Here is an example instance for the second chunk of the training data:

```
expect -2 -1 0 morph-cap in IN
NN PP passive clause A0VA1 A1
```

This instance contains 12 features: the verb (1), distance to the verb measured in chunks (2), NP chunks (3) and VP chunks (4), attenuated words (5–6), PoS tags (7–8), a chunk tag (9), passive main verb (10), current clause

(11) and role pattern (12). The final item of the line is the required output class. Our choice of instance format is only slightly harmful for performance: with a perfect classifier we can still obtain a maximal F$_{\beta=1}$ score of 99.1 on the development data.

3 Approach

In this section we describe our approach to semantic role labeling. The core part of our system is a memory-based learner. During the development of the system we have used feature selection and parameter optimization by iterative deepening. Additionally we have evaluated three extensions of the basic memory-based learning method: class n-grams, i.e. complex classes composed of sequences of simple classes, iterative classifier stacking and automatic output post-processing.

3.1 Memory-based learning

Memory-based learning is a supervised inductive algorithm for learning classification tasks based on the k-nn algorithm (Cover and Hart, 1967; Aha et al., 1991) with various extensions for dealing with nominal features and feature relevance weighting. Memory-based learning stores feature representations of training instances in memory without abstraction and classifies new (test) instances by matching their feature representation to all instances in memory, finding the most similar instances. From these “nearest neighbors”, the class of the test item is extrapolated. See Daelemans et al. (2003) for a detailed description of the algorithms and metrics used in our experiments. All memory-based learning experiments were done with the TiMBL software package[1].

In previous research, we have found that memory-based learning is rather sensitive to the chosen features and the particular setting of its algorithmic parameters (e.g. the number of nearest neighbors taken into account, the function for extrapolation from the nearest neighbors, the feature relevance weighting method used, etc.). In order to minimize the effects of this sensitivity, we have put much effort in trying to find the best set of features and the optimal learner parameters for this particular task.

3.2 Feature selection

We have employed bi-directional hill-climbing (Caruana and Freitag, 1994) for finding the features that were most suited for this task. This wrapper approach starts with the empty set of features and evaluates the learner for every individual feature on the development set. The feature associated with the best performance is selected and the process is repeated for every pair of features that includes the best feature. For every next best set of features, the system evaluates each set that contains one extra feature or has one feature less. This process is repeated until the local search does not lead to a performance gain.

[1]We used TiMBL version 5.0, available freely for research from `http://ilk.uvt.nl`.

3.3 Parameter optimization

We used *iterative deepening* (ID) as a heuristic way of searching for optimal algorithm parameters. This technique combines classifier wrapping (using the training material internally to test experimental variants) (Kohavi and John, 1997) with progressive sampling of training material (Provost et al., 1999). We start with a large pool of experiments, each with a unique combination of algorithmic parameter settings. Each settings combination is applied to a small amount of training material and tested on a small held-out set also taken from the training set. Only the best settings are kept; the others are removed from the pool of competing settings. In subsequent iterations, this step is repeated, retaining the best-performing settings, with an exponentially growing amount of training and held-out data – until all training data is used or one best setting is left. Selecting the best settings at each step is based on classification accuracy on the held-out data; a simple one-dimensional clustering on the ranked list of accuracies determines which group of settings is selected for the next iteration.

3.4 Class n-grams

Alternative to predicting simple classes, sequential tasks can be rephrased as mappings from input examples to sequences of classes. Instead of predicting just `A1` in the example given earlier, it is possible to predict a trigram of classes. The second example in the training data which we used earlier, is now labeled with the trigram `A1_A1_A1`, indicating that the chunk in focus has an `A1` relation with the verb, along with its left and right neighbor chunks (which are all part of the same `A1` argument).

```
expect -2 -1 0 morph-cap in
IN NN PP passive clause A0VA1
A1_A1_A1
```

Predicting class trigrams offers two potential benefits. First, the classifier is forced to predict ‘legal’ sequences of classes; this potentially fixes a problem with simple classifiers which are blind to their previous or subsequent simple classifications in sequences, potentially resulting in impossible sequences such as `A1 A0 A1`. Second, if the classifier predicts the trigrams example by example, it produces a sequence of overlapping trigrams which may contain information that can boost classification accuracy. Effectively, each class is predicted three times, so that a simple majority voting can be applied: we simply take the middle prediction as the actual classification of the example unless the two other votes together suggest another class label.

	Prec.	Recall	$F_{\beta=1}$	method
a	51.6%	51.9%	51.8	feature selection
b	57.3%	52.7%	54.9	parameter optimization
c	58.8%	54.2%	56.4	feature selection
d	59.5%	53.9%	56.5	parameter optimization
e	64.3%	54.2%	58.8	classifier stacking
f	66.3%	56.3%	60.9	parameter optimization
g	66.5%	56.3%	60.9	feature selection
h	68.1%	56.8%	61.9	classifier stacking
i	68.3%	57.5%	62.4	feature selection
j	68.9%	57.8%	62.9	classifier stacking
k	69.1%	57.8%	63.0	classifier stacking
	50.6%	30.3%	37.9	baseline

Table 1: Effects of cascaded feature selection, parameter optimization and classifier stacking on the performance measured on the development data set.

3.5 Iterative classifier stacking

Stacking (Wolpert, 1992) refers to a class of meta-learning systems that learn to correct errors made by lower-level classifiers. We implement stacking by adding a windowed sequence of previous and subsequent output class labels to the original input features. To generate the training material, we copy these windowed (unigram) class labels into the input, excluding the focus class label (which is a perfect predictor of the output class). To generate test material, the output of the first-stage classifier trained on the original data is used.

Stacking can be repeated; an nth-stage classifier can be built on the output of the n-1th-stage classifier. We implemented this by replacing the class features in the input of each nth-stage classifier by the output of the previous classifier.

3.6 Automatic output post-processing

Even while employing n-gram output classes and classifier stacking, we noticed that our learner made systematic errors caused by the lack of broader (sentential) contextual information in the instances and the classes. The most obvious of these errors was having multiple instances of arguments `A0-A5` in one sentence. Although sentences with multiple `A0-A3` arguments appear in the training data, they are quite rare (0.17%). When the learner assigns an `A0` role to three different arguments in a sentence, most likely at least two of these are wrong. In order to reflect this fact, we have restricted the system to outputting at most one phrase of type `A0-A5`. If the learner predicts multiple arguments then only the one closest to the main verb is kept.

Features	a-b	c-d	e-f	g-h	i-k
words	-1–0	-2–1	-2–1	-2–1	-2–1
PoS tags	0–1	0–1	0–1	-1–1	-1–1
chunk tags	0	0–2	0–2	-1–1	-1–1
NE tags	–	–	–	–	–
output classes	NA	NA	-3–3	-3–3	-3–3
distances	cNV	cNVw	cNVw	Vw	cNV
main verb	+	+	+	+	+
role pattern	+	+	+	+	+
passive verb	+	+	+	+	+
current clause	+	+	+	+	+
previous prep.	–	+	+	+	–
Total	12	18	24	23	24

Table 2: Features used in the different runs mentioned in Table 1. The numbers mentioned for words, part-of-speech tags, chunk tags, named entity tags and output classes show the position of the tokens with respect to the focus token (0). Distances are measured in chunks, NP chunks, VP chunks and words. In all other table entries, + denotes selection and – omission.

Parameters	a	b-c	d-e	f-k
algorithm	IB1	IB1	IB1	IB1
distance metric	O	M	J	O
switching threshold	NA	2	2	NA
feature weighting	nw	nw	nw	nw
neighborhood size	1	15	19	1
class weights	Z	ED1	ED1	Z

Table 3: Parameters of the machines learner that were used in the different runs mentioned in Table 1. More information about the parameters and their values can be found in Daelemans et al. (2003).

4 Results

We started with a feature selection process with the features described in section 2. This experiment used a basic k-nn classifier without feature weighting, a nearest neighborhood of size 1, attenuated words, and output post-processing. We evaluated the effect of trigram output classes by performing an experiment with and without them. The feature selection experiment without trigram output classes selected 10 features and obtained an $F_{\beta=1}$ score of 46.3 on the development data set. The experiment that made use of combined classes selected 12 features and reached a score of 51.8.

We decided to continue using trigram output classes. Subsequently, we optimized the parameters of our machine learner based on the features in the second experiment and performed another feature selection experiment with these parameters. The performance effects can be found in Table 1 (rows b and c). An additional parameter optimization step did not have a substantial effect (Table 1, row d).

After training a stacked classifier while using the output of the best first stage learner, performance went up from 56.5 to 58.8. Additional feature selection and parameter optimization were useful at this level ($F_{\beta=1}$=60.9, see Table 1). Most of our other performance gain was obtained by a continued process of classifier stacking. Parameter optimization did not result in improved performance when stacking more than one classifier. Feature selection was useful for the third-stage classifier but not for the next one. Our final system obtained an $F_{\beta=1}$ score of 63.0 on the development data (Table 1) and 60.1 on the test set (Table 4).

5 Conclusion

We have described a memory-based semantic role labeler. In the development of the system we have used feature selection through bi-directional hill-climbing and parameter optimization through iterative deepening search. We have evaluated n-gram output classes, classifier stacking and output post-processing, all of which increased performance. An overview of the performance of the system on the test data can be found in Table 4.

Acknowledgements

Sander Canisius, Iris Hendrickx, and Antal van den Bosch are funded by NWO (Netherlands Organisation for Scientific Research). Erik Tjong Kim Sang is funded by IWT STWW as a researcher in the ATraNoS project.

	Precision	Recall	$F_{\beta=1}$
Overall	67.12%	54.46%	60.13
A0	80.41%	70.18%	74.95
A1	62.04%	59.67%	60.83
A2	46.29%	35.85%	40.41
A3	59.42%	27.33%	37.44
A4	67.44%	58.00%	62.37
A5	0.00%	0.00%	0.00
AM-ADV	25.00%	4.56%	7.71
AM-CAU	0.00%	0.00%	0.00
AM-DIR	33.33%	12.00%	17.65
AM-DIS	58.38%	50.70%	54.27
AM-EXT	53.85%	50.00%	51.85
AM-LOC	38.79%	19.74%	26.16
AM-MNR	48.00%	18.82%	27.04
AM-MOD	97.11%	89.61%	93.21
AM-NEG	74.67%	88.19%	80.87
AM-PNC	44.44%	4.71%	8.51
AM-PRD	0.00%	0.00%	0.00
AM-TMP	58.84%	32.53%	41.90
R-A0	80.26%	76.73%	78.46
R-A1	78.95%	42.86%	55.56
R-A2	100.00%	22.22%	36.36
R-A3	0.00%	0.00%	0.00
R-AA	0.00%	0.00%	0.00
R-AM-LOC	0.00%	0.00%	0.00
R-AM-MNR	0.00%	0.00%	0.00
R-AM-PNC	0.00%	0.00%	0.00
R-AM-TMP	66.67%	14.29%	23.53
V	97.93%	97.93%	97.93

Table 4: The performance of our system measured on the test data.

References

D. W. Aha, D. Kibler, and M. Albert. 1991. Instance-based learning algorithms. *Machine Learning*, 6:37–66.

S. Buchholz, J. Veenstra, and W. Daelemans. 1999. Cascaded grammatical relation assignment. In *EMNLP-VLC'99, the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, June.

S. Buchholz. 2002. *Memory-Based Grammatical Relation Finding*. PhD thesis, University of Tilburg.

X. Carreras and L. Màrquez. 2004. Introduction to the conll-2004 shared task: Semantic role labe ling. In *Proceedings of CoNLL-2004*. Boston, MA, USA.

R. Caruana and D. Freitag. 1994. Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 28–36, New Brunswick, NJ, USA. Morgan Kaufman.

T. M. Cover and P. E. Hart. 1967. Nearest neighbor pattern classification. *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, 13:21–27.

W. Daelemans, J. Zavrel, K. Van der Sloot, and A. Van den Bosch. 2003. TiMBL: Tilburg memory based learner, version 5.0, reference guide. ILK Technical Report 03-08, Tilburg University. available from http://ilk.uvt.nl/downloads/pub/papers/ilk.0308.ps.

J. Eisner. 1996. *An Empirical Comparison of Probability Models for Dependency Grammar*. Technical Report IRCS-96-11, Institute for Research in Cognitive Science, University of Pennsylvania.

R. Kohavi and G. John. 1997. Wrappers for feature subset selection. *Artificial Intelligence Journal*, 97(1–2):273–324.

F. Provost, D. Jensen, and T. Oates. 1999. Efficient progressive sampling. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, pages 23–32.

D. H. Wolpert. 1992. On overfitting avoidance as bias. Technical Report SFI TR 92-03-5001, The Santa Fe Institute.