Improving sequence segmentation learning by predicting trigrams

van den Bosch, A.; Daelemans, W.

Published in:
Proceedings of the Ninth Conference on Natural Language Learning, CONLL-2005, June 29-30

Publication date:
2005

Improving sequence segmentation learning by predicting trigrams

Antal van den Bosch
ILK / Computational Linguistics and AI
Tilburg University
Tilburg, The Netherlands
Antal.vdnBosch@uvt.nl

Walter Daelemans
CNTS, Department of Linguistics
University of Antwerp
Antwerp, Belgium
Walter.Daelemans@ua.ac.be

Abstract

Symbolic machine-learning classifiers are known to suffer from near-sightedness when performing sequence segmentation (chunking) tasks in natural language processing: without special architectural additions they are oblivious of the decisions they made earlier when making new ones. We introduce a new pointwise-prediction single-classifier method that predicts trigrams of class labels on the basis of windowed input sequences, and uses a simple voting mechanism to decide on the labels in the final output sequence. We apply the method to maximum-entropy, sparse-winnow, and memory-based classifiers using three different sentence-level chunking tasks, and show that the method is able to boost generalization performance in most experiments, attaining error reductions of up to 51%. We compare and combine the method with two known alternative methods to combat near-sightedness, viz. a feedback-loop method and a stacking method, using the memory-based classifier. The combination with a feedback loop suffers from the label bias problem, while the combination with a stacking method produces the best overall results.

1 Optimizing output sequences

Many tasks in natural language processing have the full sentence as their domain. Chunking tasks, for example, deal with segmenting the full sentence into chunks of some type, for example constituents or named entities, and possibly labeling each identified chunk. The latter typically involves disambiguation among alternative labels (e.g. syntactic role labeling, or semantic type assignment). Both tasks, whether seen as separate tasks or as one, involve the use of contextual knowledge from the available input (e.g. words with part-of-speech tags), but also the coordination of segmentations and disambiguations over the sentence as a whole.

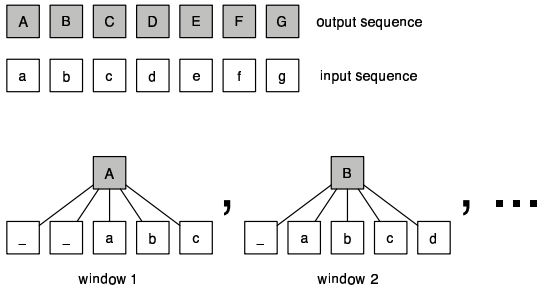

Figure 1: Standard windowing process. Sequences of input symbols and output symbols are converted into windows of fixed-width input symbols each associated with one output symbol.

Many machine-learning approaches to chunking tasks use windowing, a standard representational approach to generate cases that can be sequentially processed. Each case produces one element of the output sequence. The simplest method to process these cases is that each case is classified in isolation, generating a so-called point-wise prediction; the sequence of subsequent predictions can be concatenated to form the entire output analysis of the sentence. Within a window, fixed-width subsequences of adjacent input symbols, representing a certain contextual scope, are mapped to one output symbol, typically associated with one of the input symbols, for example the middle one. Figure 1 displays this standard version of the windowing process.

The fact that the point-wise classifier is only trained to associate subsequences of input symbols to single output symbols as accurately as possible is a problematic restriction: it may easily cause the classifier to produce invalid or impossible output sequences, since it is incapable of taking into account any decisions it has made earlier. This well-known problem has triggered at least the following three main types of solutions.

Feedback loop Each training or test example may represent not only the regular windowed input, but also a copy of previously made classifications, to allow the classifier to be more consistent with its previous decisions. Direct feedback loops that copy a predicted output label to the input representation of the next example have been used in symbolic machine-learning architectures such as the the maximum-entropy tagger described by Ratnaparkhi (1996) and the memory-based tagger (MBT) proposed by Daelemans et al. (1996). This solution assumes that processing is directed, e.g. from left to right. A noted problem of this approach is the *label bias problem* (Lafferty et al., 2001), which is that a feedback-loop classifier may be driven to be consistent with its previous decision also in the case this decision was wrong; sequences of errors may result.

Stacking, boosting, and voting The partly incorrect concatenated output sequence of a single classifier may serve as input to a second-stage classifier in a stacking architecture, a common machine-learning optimization technique (Wolpert, 1992). Although less elegant than a monolithic single-classifier architecture, this method is known to be capable of recognizing recurring errors of the first-stage classifier and correcting them (Veenstra, 1998). Boosting (Freund and Schapire, 1996) has been applied to optimize chunking systems (Carreras et al., 2002), as well as voting over sets of different classifiers (Florian et al., 2003). Punyakanok and Roth (2001) present two methods for combining the predictions of different classifiers according to constraints that ensure that the resulting output is made more coherent.

Output sequence optimization Rather than basing classifications only on model parameters estimated from co-occurrences between input and output symbols employed for maximizing the likelihood of point-wise single-label predictions at the output level, classifier output may be augmented by an optimization over the output sequence as a whole using optimization techniques such as beam searching in the space of a conditional markov model's output (Ratnaparkhi, 1996) or hidden markov models (Skut and Brants, 1998). Maximum-entropy markov models (McCallum et al., 2000) and conditional random fields (Lafferty et al., 2001) optimize the likelihood of segmentations of output symbol sequences through variations of Viterbi search. A non-stochastic, non-generative method for output sequence optimization is presented by Argamon et al. (1999), who propose a memory-based sequence learner that finds alternative chunking analyses of a sequence, and produces one best-guess analysis by a tiling algorithm that finds an optimal joining of the alternative analyses.

In this paper we introduce a symbolic machine-learning method that can be likened to the approaches of the latter type of output sequence optimizers, but which does not perform a search in a space of possible analyses. The approach is to have a point-wise symbolic machine-learning classifier predict series of overlapping n-grams (in the current study, trigrams) of class symbols, and have a simple voting mechanism decide on the final output sequence based on the overlapping predicted trigrams. We show that the approach has similar positive effects when applied to a memory-based classifier and a maximum-entropy classifier, while yielding mixed effects with a sparse-winnow classifier. We then proceed to compare the trigram prediction method to a feedback-loop method and a stacking method applied using the memory-based classifier. The three methods attain comparable error reductions. Finally, we combine the trigram-prediction method with each of the two other methods. We show that the combination of the trigram-prediction method and the feedback-loop method does not improve performance due to the label bias problem. In contrast, the combination of the trigram-prediction method and the stacking method leads to the overall best results, indicating that the latter two methods solve complementary aspects of the near-sightedness problem.

The structure of the paper is as follows. First,

we introduce the three chunking sequence segmentation tasks studied in this paper and explain the automatic algorithmic model selection method for the three machine-learning classifiers used in our study, in Section 2. The subsequent three sections report on empirical results for the different methods proposed for correcting the near-sightedness of classifiers: the new class-trigrams method, a feedback-loop approach in combination with single classes and class trigrams, and two types of stacking in combination with single classes and class trigrams. Section 6 sums up and discusses the main results of the comparison.

2 Data and methodology

The three data sets we used for this study represent a varied set of sentence-level chunking tasks of both syntactic and semantic nature: English base phrase chunking (henceforth CHUNK), English named-entity recognition (NER), and disfluency chunking in transcribed spoken Dutch utterances (DISFL).

CHUNK is the task of splitting sentences into non-overlapping syntactic phrases or constituents. The used data set, extracted from the WSJ Penn Treebank, contains 211,727 training examples and 47,377 test instances. The examples represent seven-word windows of words and their respective (predicted) part-of-speech tags, and each example is labeled with a class using the IOB type of segmentation coding as introduced by Ramshaw and Marcus (1995), marking whether the middle word is inside (I), outside (O), or at the beginning (B) of a chunk. Words occuring less than ten times in the training material are attenuated (converted into a more general string that retains some of the word's surface form). Generalization performance is measured by the F-score on correctly identified and labeled constituents in test data, using the evaluation method originally used in the "shared task" subevent of the CoNLL-2000 conference (Tjong Kim Sang and Buchholz, 2000) in which this particular training and test set were used. An example sentence with base phrases marked and labeled is the following: [*He*]$_{NP}$ [*reckons*]$_{VP}$ [*the current account deficit*]$_{NP}$ [*will narrow*]$_{VP}$ [*to*]$_{PP}$ [*only $ 1.8 billion*]$_{NP}$ [*in*]$_{PP}$ [*September*]$_{NP}$.

NER, named-entity recognition, is to recognize and type named entities in text. We employ the English NER shared task data set used in the CoNLL-2003 conference, again using the same evaluation method as originally used in the shared task (Tjong Kim Sang and De Meulder, 2003). This data set discriminates four name types: persons, organizations, locations, and a rest category of "miscellany names". The data set is a collection of newswire articles from the Reuters Corpus, RCV1[1]. The given training set contains 203,621 examples; as test set we use the "testb" evaluation set which contains 46,435 examples. Examples represent seven-word windows of unattenuated words with their respective predicted part-of-speech tags. No other task-specific features such as capitalization identifiers or seed list features were used. Class labels use the IOB segmentation coding coupled with the four possible name type labels. Analogous to the CHUNK task, generalization performance is measured by the F-score on correctly identified and labeled named entities in test data. An example sentence with the named entities segmented and typed is the following: [*U.N.*]$_{organization}$ *official* [*Ekeus*]$_{person}$ *heads for* [*Baghdad*]$_{location}$.

DISFL, disfluency chunking, is the task of recognizing subsequences of words in spoken utterances such as fragmented words, laughter, self-corrections, stammering, repetitions, abandoned constituents, hesitations, and filled pauses, that are not part of the syntactic core of the spoken utterance. We use data introduced by Lendvai et al. (2003), who extracted the data from a part of the Spoken Dutch Corpus of spontaneous speech[2] that is both transcribed and syntactically annotated. All words and multi-word subsequences judged not to be part of the syntactic tree are defined as disfluent chunks. We used a single 90% – 10% split of the data, producing a training set of 303,385 examples and a test set of 37,160 examples. Each example represents a window of nine words (attenuated below an occurrence threshold of 100) and 22 binary features representing various string overlaps (to encode possible repetitions); for details, cf. (Lendvai

[1] Reuters Corpus, Volume 1, English language, 1996-08-20 to 1997-08-19.

[2] CGN, Spoken Dutch Corpus, version 1.0, `http://lands.let.kun.nl/cgn/ehome.htm`.

et al., 2003). Generalization performance is measured by the F-score on correctly identified disfluent chunks in test data. An example of a chunked Spoken Dutch Corpus sentence is the following ("uh" is a filled pause; without the disfluencies, the sentence means "I have followed this process with a certain amount of scepticism for about a year"): [*ik uh*] *ik heb met de nodige scepsis* [*uh*] *deze gang van zaken* [*zo'n*] *zo'n jaar aangekeken*.

We perform our experiments on the three tasks using three machine-learning algorithms: the memory-based learning or k-nearest neighbor algorithm as implemented in the TiMBL software package (version 5.1) (Daelemans et al., 2004), henceforth referred to as MBL; maximum-entropy classification (Guiasu and Shenitzer, 1985) as implemented in the maxent software package (version 20040930) by Zhang Le[3], henceforth MAXENT; and a sparse-winnow network (Littlestone, 1988) as implemented in the SNoW software package (version 3.0.5) by Carlson et al. (1999), henceforth WINNOW. All three algorithms have algorithmic parameters that bias their performance; to allow for a fair comparison we optimized each algorithm on each task using wrapped progressive sampling (Van den Bosch, 2004) (WPS), a heuristic automatic procedure that, on the basis of validation experiments internal to the training material, searches among algorithmic parameter combinations for a combination likely to yield optimal generalization performance on unseen data. We used wrapped progressive sampling in all experiments.

3 Predicting class trigrams

There is no intrinsic bound to what is packed into a class label associated to a windowed example. For example, complex class labels can span over trigrams of singular class labels. A classifier that learns to produce trigrams of class labels will at least produce syntactically valid trigrams from the training material, which might partly solve some near-sightedness problems of the single-class classifier. Although simple and appealing, the lurking disadvantage of the trigram idea is that the number of class labels increases explosively when moving from single class labels to wider trigrams. The CHUNK data, for example, has 22 classes ("IOB" codes associated with chunk types); in the same training set, 846 different trigrams of these 22 classes and the start/end context symbol occur. The eight original classes of NER combine to 138 occurring trigrams. DISFL only has two classes, but 18 trigram classes.

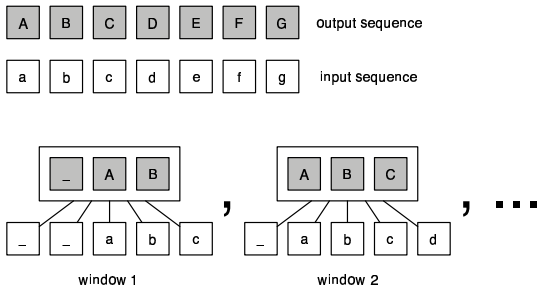

Figure 2: Windowing process with trigrams of class symbols. Sequences of input symbols and output symbols are converted into windows of fixed-width input symbols each associated with, in this example, trigrams of output symbols.

Figure 2 illustrates the procedure by which windows are created with, as an example, class trigrams. Each windowed instance maps to a class label that incorporates three atomic class labels, namely the focus class label that was the original unigram label, plus its immediate left and right neighboring class labels.

While creating instances this way is trivial, it is not entirely trivial how the output of overlapping class trigrams recombines into a normal string of class sequences. When the example illustrated in Figure 2 is followed, each single class label in the output sequence is effectively predicted three times; first, as the right label of a trigram, next as the middle label, and finally as the left label. Although it would be possible to avoid overlaps and classify only every three words, there is an interesting property of overlapping class label n-grams: it is possible to vote over them. To pursue our example of trigram classes, the following voting procedure can be followed to decide about the resulting unigram class label sequence:

1. When all three votes are unanimous, their common class label is returned;
2. When two out of three votes are for the same

[3]Maximum Entropy Modeling Toolkit for Python and C++, `http://homepages.inf.ed.ac.uk/s0450736/maxent_toolkit.html`.

	MBL			MAXENT			WINNOW		
Task	Baseline	Trigram	red.	Baseline	Trigram	red.	Baseline	Trigram	red.
CHUNK	91.9	**92.7**	10	90.3	**91.9**	17	**89.5**	88.3	-11
NER	77.2	**80.2**	17	47.5	**74.5**	51	68.9	**70.1**	4
DISFL	77.9	**81.7**	17	75.3	**80.7**	22	**70.5**	65.3	-17

Table 1: Comparison of generalization performances of three machine-learning algorithms in terms of F-score on the three test sets without and with class trigrams. Each third column displays the error reduction in F-score by the class trigrams method over the other method. The best performances per task are printed in bold.

class label, this class label is returned;

3. When all three votes disagree (i.e., when majority voting ties), the class label is returned of which the classifier is most confident.

Classifier confidence, needed for the third tie-breaking rule, can be heuristically estimated by taking the distance of the nearest neighbor in MBL, the estimated probability value of the most likely class produced by the MAXENT classifier, or the activation level of the most active unit of the WINNOW network.

Clearly this scheme is one out of many possible schemes, using variants of voting as well as variants of n (and having multiple classifiers with different n, so that some back-off procedure could be followed). For now we use this procedure with trigrams as an example. To measure its effect we apply it to the sequence tasks CHUNK, NER, and DISFL. The results of this experiment, where in each case WPS was used to find optimal algorithmic parameters of all three algorithms, are listed in Table 1. We find rather positive effects of the trigram method both with MBL and MAXENT; we observe relative error reductions in the F-score on chunking ranging between 10% and a remarkable 51% error reduction, with MAXENT on the NER task. With WINNOW, we observe decreases in performance on CHUNK and DISFL, and a minor error reduction of 4% on NER.

4 The feedback-loop method versus class trigrams

An alternative method for providing a classifier access to its previous decisions is a feedback-loop approach, which extends the windowing approach by feeding previous decisions of the classifier as features into the current input of the classifier. This approach was proposed in the context of memory-based learning for part-of-speech tagging as MBT (Daelemans et al., 1996). The number of decisions fed back into the input can be varied. In the experiments described here, the feedback loop iteratively updates a memory of the three most recent predictions.

Task	Baseline	Feedback	Trigrams	Feed+Tri
CHUNK	91.9	**93.0**	92.7	89.8
NER	77.2	78.1	**80.2**	77.5
DISFL	77.9	78.6	**81.7**	79.1

Table 2: Comparison of generalization performances in terms of F-score of MBL on the three test sets, with and without a feedback loop, and the error reduction attained by the feedback-loop method, the F-score of the trigram-class method, and the F-score of the combination of the two methods.

The feedback-loop approach can be combined both with single class and class trigram output. In the latter case, the full trigram class labels are copied to the input, retaining at any time the three most recently predicted labels in the input. Table 2 shows the results for both options on the three chunking tasks. The feedback-loop method outperforms the trigram-class method on CHUNK, but not on the other two tasks. It does consistently outperform the baseline single-class classifier. Interestingly, the combination of the two methods performs worse than the baseline classifier on CHUNK, and also performs worse than the trigram-class method on the other two tasks.

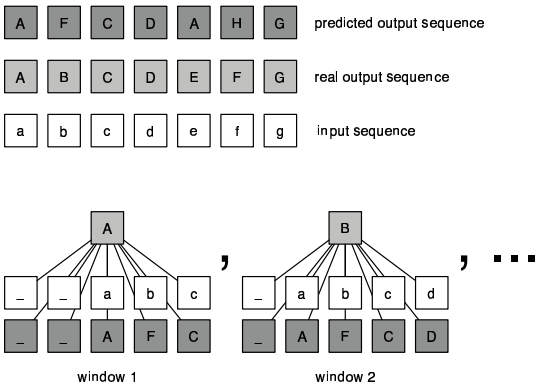

Figure 3: The windowing process after a first-stage classifier has produced a predicted output sequence. Sequences of input symbols, predicted output symbols, and real output symbols are converted into windows of fixed-width input symbols and predicted output symbols, each associated with one output symbol.

5 Stacking versus class trigrams

Stacking, a term popularized by Wolpert (1992) in an artificial neural network context, 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 (here, we copy a window of seven predictions to the input, centered around the middle position), and providing these enriched examples as training material to a second-stage classifier. Figure 3 illustrates the procedure. Given the (possibly erroneous) output of a first classifier on an input sequence, a certain window of class symbols from that predicted sequence is copied to the input, to act as predictive features for the real class label.

To generate the output of a first-stage classifier, two options are open. We name these options **perfect** and **adaptive**. They differ in the way they create training material for the second-stage classifier:

Perfect – the training material is created straight from the training material of the first-stage classifier, by windowing over the real class sequences. In doing so, the class label of each window is excluded from the input window, since it is always the same as the class to be predicted. In training, this focus feature would receive an unrealistically high weight, especially considering that in testing this feature would contain errors. To assign a very high weight to a feature that may contain an erroneous value does not seem a good idea in view of the label bias problem.

Task	Baseline	Perfect stacking	Adaptive stacking
CHUNK	91.9	92.0	**92.6**
NER	77.2	78.3	**78.9**
DISFL	77.9	80.5	**81.6**

Table 3: Comparison of generalization performances in terms of F-score of MBL on the three test sets, without stacking, and with perfect and adaptive stacking.

Adaptive – the training material is created indirectly by running an internal 10-fold cross-validation experiment on the first-stage training set, concatenating the predicted output class labels on all of the ten test partitions, and converting this output to class windows. In contrast with the perfect variant, we do include the focus class feature in the copied class label window. The adaptive approach can in principle learn from recurring classification errors in the input, and predict the correct class in case an error re-occurs.

Table 3 lists the comparative results on the CHUNK, NER, and DISFL tasks introduced earlier. They show that both types of stacking improve performance on the three tasks, and that the adaptive stacking variant produces higher relative gains than the perfect variant; in terms of error reduction in F-score as compared to the baseline single-class classifier, the gains are 9% for CHUNK, 7% for NER, and 17% for DISFL. There appears to be more useful information in training data derived from cross-validated output with errors, than in training data with error-free material.

Stacking and class trigrams can be combined. One possible straightforward combination is that of a first-stage classifier that predicts trigrams, and a second-stage stacked classifier that also predicts trigrams (we use the adaptive variant, since it produced the best results), while including a centered seven-positions-wide window of first-stage trigram class labels in the input. Table 4 compares the results

Task	Adaptive stacking	Trigram	Combination
CHUNK	92.6	92.8	**93.1**
NER	78.9	80.2	**80.6**
DISFL	81.6	81.7	**81.9**

Table 4: Comparison of generalization performances in terms of F-score by MBL on the three test sets, with adaptive stacking, trigram classes, and the combination of the two.

of adaptive stacking and trigram classes with those of the combination of the two. As can be seen, the combination produces even better results than both the stacking and the trigram-class methods individually, on all three tasks. Compared to the baseline single-class classifier, the error reductions are 15% for CHUNK, 15% for NER, and 18% for DISFL.

As an additional analysis, we inspected the predictions made by the trigram-class method and its combinations with the stacking and the feedback-loop methods on the CHUNK task to obtain a better view on the amount of disagreements between the trigrams. We found that with the trigram-class method, in 6.3% of all votes some disagreement among the overlapping trigrams occurs. A slightly higher percentage of disagreements, 7.1%, is observed with the combination of the trigram-class and the stacking method. Interestingly, in the combination of the trigram-class and feedback-loop methods, only 0.1% of all trigram votes are not unanimous. This clearly illustrates that in the latter combination the resulting sequence of trigrams is internally very consistent – also in its errors.

6 Conclusion

Classifiers trained on chunking tasks that make isolated. near-sighted decisions on output symbols and that do not optimize the resulting output sequences afterwards or internally through a feedback loop, tend to produce weak models for sequence processing tasks. To combat this weakness, we have proposed a new method that uses a single symbolic machine-learning classifier predicting trigrams of classes, using a simple voting mechanism to reduce the sequence of predicted overlapping trigrams to a sequence of single output symbols. Compared to their near-sighted counterparts, error reductions are attained of 10 to 51% with MBL and MAXENT on three chunking tasks. We found weaker results with a WINNOW classifier, suggesting that the latter is more sensitive to the division of the class space in more classes, likely due to the relatively sparser co-occurrences between feature values and class labels on which WINNOW network connection weights are based.

We have contrasted the trigram-class method against a feedback-loop method (MBT) and a stacking method, all using a memory-based classifier (but the methods generalize to any machine-learning classifier). With the feedback-loop method, modest error reductions of 3%, 4%, and 17% are measured; stacking attains comparable improvements of 7%, 9%, and 17% error reductions in the chunking F-score. We then combined the trigram-class method with the two other methods. The combination with the feedback-loop system led to relatively low performance results. A closer analysis indicated that the two methods appear to render each other ineffective: by feeding back predicted trigrams in the input, the classifier is very much geared towards predicting a next trigram that will be in accordance with the two partly overlapping trigrams in the input, as suggested by overwhelming evidence in this direction in training material – this problem is also known as the label bias problem (Lafferty et al., 2001). (The fact that maximum-entropy markov models also suffer from this problem prompted Lafferty *et al.* to propose conditional random fields.)

We also observed that the positive effects of the trigram-class and stacking variants do not mute each other when combined. The overall highest error reductions are attained with the combination: 15% for CHUNK, 15% for NER, and 18% for DISFL. The combination of the two methods solve more errors than the individual methods do. Apparently, they both introduce complementary disagreements in overlapping trigrams, which the simple voting mechanism can convert to more correct predictions than the two methods do individually.

Further research should focus on a deep quantitative and qualitative analysis of the different errors the different methods correct when compared to the baseline single-class classifier, as well as the errors they may introduce. Alternatives to the

IOB-style encoding should also be incorporated in these experiments (Tjong Kim Sang, 2000). Additionally, a broader comparison with point-wise predictors (Kashima and Tsuboi, 2004) as well as Viterbi-based probabilistic models (McCallum et al., 2000; Lafferty et al., 2001; Sha and Pereira, 2003) in large-scale comparative studies is warranted. Also, the scope of the study may be broadened to all sequential language processing tasks, including tasks in which no segmentation takes place (e.g. part-of-speech tagging), and tasks at the morpho-phonological level (e.g. grapheme-phoneme conversion and morphological analysis).

Acknowledgements

The authors wish to thank Sander Canisius for discussions and suggestions. The work of the first author is funded by NWO, the Netherlands Organisation for Scientific Research; the second author's work is partially funded by the EU BioMinT project.

References

S. Argamon, I. Dagan, and Y. Krymolowski. 1999. A memory-based approach to learning shallow natural language patterns. *Journal of Experimental and Theoretical Artificial Intelligence*, 10:1–22.

A. J. Carlson, C. M. Cumby, J. L. Rosen, and D. Roth. 1999. Snow user guide. Technical Report UIUCDCS-R-99-2101, Cognitive Computation Group, Computer Science Department, University of Illinois, Urbana, Illinois.

X. Carreras, L. Màrques, and L. Padró. 2002. Named entity extraction using AdaBoost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei, Taiwan.

W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. 1996. MBT: A memory-based part of speech tagger generator. In E. Ejerhed and I. Dagan, editors, *Proceedings of WVLC*, pages 14–27. ACL SIGDAT.

W. Daelemans, J. Zavrel, K. van der Sloot, and A. van den Bosch. 2004. TiMBL: Tilburg memory based learner, version 5.1.0, reference guide. Technical Report ILK 04-02, ILK Research Group, Tilburg University.

R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. 2003. Named entity recognition through classifier combination. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada.

Y. Freund and R. E. Schapire. 1996. Experiments with a new boosting algorithm. In L. Saitta, editor, *Proceedings of ICML-96*, pages 148–156, San Francisco, CA. Morgan Kaufmann.

S. Guiasu and A. Shenitzer. 1985. The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1).

H. Kashima and Y. Tsuboi. 2004. Kernel-based discriminative learning algorithms for labeling sequences, trees and graphs. In *Proceedings of ICML-2004*, Banff, Canada.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, Williamstown, MA.

P. Lendvai, A. van den Bosch, and E. Krahmer. 2003. Memory-based disfluency chunking. In *Proceedings of DISS'03)*, Gothenburg, Sweden, pages 63–66.

N. Littlestone. 1988. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318.

A. McCallum, D. Freitag, and F. Pereira. 2000. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of ICML-00*, Stanford, CA.

V. Punyakanok and D. Roth. 2001. The use of classifiers in sequential inference. In *NIPS-13; The 2000 Conference on Advances in Neural Information Processing Systems*, pages 995–1001. The MIT Press.

L.A. Ramshaw and M.P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of WVLC-95*, Cambridge, MA, pages 82–94.

A. Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of EMNLP, May 17-18, 1996, University of Pennsylvania*.

F. Sha and F. Pereira. 2003. Shallow parsing with Conditional Random Fields. In *Proceedings of HLT-NAACL 2003*, Edmonton, Canada.

W. Skut and T. Brants. 1998. Chunk tagger: statistical recognition of noun phrases. In *ESSLLI-1998 Workshop on Automated Acquisition of Syntax and Parsing*.

E. Tjong Kim Sang and S. Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132.

E. Tjong Kim Sang and F. De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

E. Tjong Kim Sang. 2000. Noun phrase recognition by system combination. In *Proceedings of ANLP-NAACL 2000*, pages 50–55. Seattle, Washington, USA. Morgan Kaufman Publishers.

A. van den Bosch. 2004. Wrapped progressive sampling search for optimizing learning algorithm parameters. In R. Verbrugge, N. Taatgen, and L. Schomaker, editors, *Proceedings of the 16th Belgian-Dutch AI Conference*, pages 219–226, Groningen, The Netherlands.

J. Veenstra. 1998. Fast NP chunking using memory-based learning techniques. In *Proceedings of BENELEARN'98*, pages 71–78, Wageningen, The Netherlands.

D. H. Wolpert. 1992. Stacked Generalization. *Neural Networks*, 5:241–259.