In: *Proceedings of CoNLL-2000 and LLL-2000*, pages 151–153, Lisbon, Portugal, 2000.

Text Chunking by System Combination

Erik F. Tjong Kim Sang
CNTS – Language Technology Group
University of Antwerp
erikt@uia.ua.ac.be

1 Introduction

We will apply a system-internal combination of memory-based learning classifiers to the CoNLL-2000 shared task: finding base chunks. Apart from testing different combination methods, we will also examine if dividing the chunking process in a boundary recognition phase and a type identification phase would aid performance.

2 Approach

Tjong Kim Sang (2000) describes how a system-internal combination of memory-based learners can be used for base noun phrase (baseNP) recognition. The idea is to generate different chunking models by using different chunk representations. Chunks can be represented with bracket structures but alternatively one can use a tagging representation which classifies words as being inside a chunk (I), outside a chunk (O) or at a chunk boundary (B) (Ramshaw and Marcus, 1995). There are four variants of this representation. The B tags can be used for the first word of chunks that immediately follow another chunk (the IOB1 representation) or they can be used for every chunk-initial word (IOB2). Alternatively an E tag can be used for labeling the final word of a chunk immediately preceding another chunk (IOE1) or it can be used for every chunk-final word (IOE2). Bracket structures can also be represented as tagging structures by using two streams of tags which define whether words start a chunk or not (O) or whether words are at the end of a chunk or not (C). We need both for encoding the phrase structure and hence we will treat the two tag streams as a single representation (O+C). A combination of baseNP classifiers that use the five representation performs better than any of the included systems (Tjong Kim Sang, 2000). We will apply such a classifier combination to the CoNLL-2000 shared task.

The individual classifiers will use the memory-based learning algorithm IB1-IG (Daelemans et al., 1999) for determining the most probable tag for each word. In memory-based learning the training data is stored and a new item is classified by the most frequent classification among training items which are closest to this new item. Data items are represented as sets of feature-value pairs. Features receive weights which are based on the amount of information they provide for classifying the training data (Daelemans et al., 1999).

We will evaluate nine different methods for combining the output of our five chunkers (Van Halteren et al., 1998). Five are so-called voting methods. They assign weights to the output of the individual systems and use these weights to determine the most probable output tag. Since the classifiers generate different output formats, all classifier output has been converted to the O and the C representations. The most simple voting method assigns uniform weights and picks the tag that occurs most often (Majority). A more advanced method is to use as a weight the accuracy of the classifier on some held-out part of the training data, the tuning data (TotPrecision). One can also use the precision obtained by a classifier for a specific output value as a weight (TagPrecision). Alternatively, we use as a weight a combination of the precision score for the output tag in combination with the recall score for competing tags (Precision-Recall). The most advanced voting method examines output values of pairs of classifiers and assigns weights to tags based on how often they appear with this pair in the tuning data (TagPair, Van Halteren et al., (1998)).

Apart from these voting methods we have also applied two memory-based learners to the output of the five chunkers: IB1-IG and IGTREE, a decision tree variant of IB1-IG (Daelemans et al., 1999). This approach is called classifier stacking. Like with the voting algorithms, we have tested these meta-classifiers with the output of the first classification stage. Unlike the voting algorithms, the classifiers do not require a uniform input. Therefore we have tested if their performance can be improved by supplying them with information about the input of the first classification stage. For this purpose we have used the part-of-speech tag of the current word as compressed representation of the first stage input (Van Halteren et al., 1998).

The combination methods will generate a list of open brackets and a list of close brackets. We have converted these to phrases by only using brackets which could be matched with the closest matching candidate and ignoring the others. For example, in the structure $[_{NP}$ a $[_{NP}$ b $]_{NP}$ $[_{VP}$ c $]_{PP}$ d $]_{VP}$, we would accept $[_{NP}$ b $]_{NP}$ as a noun phrase and ignore all other brackets since they cannot be matched with their closest candidate for a pair, either because of type inconsistencies or because there was some other bracket in between them.

We will examine three processing strategies in order to test our hypothesis that chunking performance can be increased by making a distinction between finding chunk boundaries and identifying chunk types. The first is the single-pass method. Here each individual classifier attempts to find the correct chunk tag for each word in one step. A variant of this is the double-pass method. It processes the data twice: first it searches for chunks boundaries and then it attempts to identify the types of the chunks found. The third processing method is the n-pass method. It contains as many passes as there are different chunk types. In each pass, it attempts to find chunks of a single type. In case a word is classified as belonging to more than one chunk type, preference will be given to the chunk type that occurs most often in the training data. We expect the n-pass method to outperform the other two methods. However, we are not sure if the performance difference will be large enough to compensate for the extra computation that is required for this processing method.

3 Results

In order to find out which of the three processing methods and which of the nine combination methods performs best, we have applied them to the training data of the CoNLL-2000 shared task (Tjong Kim Sang and Buchholz, 2000) in a 10-fold cross-validation experiment (Weiss and Kulikowski, 1991). For the single-pass method, we trained IB1-IG classifiers to produce the most likely output tags for the five data representations. In the input of the classifiers a word was represented as itself, its part-of-speech tag and a context of four left and four right word/part-of-speech tag pairs. For the four IO representations we used a second phase with a limited input context (3) but with additionally the two previous and the two next chunk tags predicted by the first phase. The classifier output was converted to the O representation (open brackets) and the C representation (close brackets) and the results were combined with the nine combination methods. In the double-pass method finding the most likely tag for each word was split in finding chunk boundaries and assigning types to the chunks. The n-pass method divided this process into eleven passes each of which recognized one chunk type.

For each processing strategy, all combination results were better than those obtained with the five individual classifiers. The differences between combination results within each processing strategy were small and between the three strategies the best results were not far apart: the best $F_{\beta=1}$ rates were 92.40 (single-pass), 92.35 (double-pass) and 92.75 (n-pass).

Since the three processing methods reach a similar performances, we can choose any of them for our remaining experiments. The n-pass method performed best but it has the disadvantage of needing as many passes as there are chunk types. This will require a lot of computation. The single-pass method was second-best but in order to obtain good results with this method, we would need to use a stacked classifier because those performed better ($F_{\beta=1}$=92.40) than the voting methods ($F_{\beta=1}$=91.98). This stacked classifier requires preprocessed combinator training data which can be obtained by processing the original training data with 10-fold cross-validation. Again this will require a lot of work for new data sets.

We have chosen for the double-pass method because in this processing strategy it is possible to obtain good results with majority voting. The advantage of using majority voting is that it does not require extra preprocessed combinator training data so by using it we avoid the extra computation required for generating this data. We have applied the double-pass method with majority voting to the CoNLL-2000 test data while using the complete training data. The results can be found in table 1. The recognition method performs well for the most frequently occurring chunk types (NP, VP and PP) and worse for the other seven (the test data did not contain UCP chunks). The recognition rate for NP chunks ($F_{\beta=1}$=93.23) is close to the result for a related standard baseNP data set obtained by Tjong Kim Sang (2000) (93.26). Our method outperforms the results mentioned in Buchholz et al. (1999) in four of the five cases (ADJP, NP, PP and VP); only for ADVP chunks it performs slightly worse. This is surprising given that Buchholz et al. (1999) used 956696 tokens of training data and we have used only 211727 (78% less).

test data	precision	recall	$F_{\beta=1}$
ADJP	85.25%	59.36%	69.99
ADVP	85.03%	71.48%	77.67
CONJP	42.86%	33.33%	37.50
INTJ	100.00%	50.00%	66.67
LST	0.00%	0.00%	0.00
NP	94.14%	92.34%	93.23
PP	96.45%	96.59%	96.52
PRT	79.49%	58.49%	67.39
SBAR	89.81%	72.52%	80.25
VP	93.97%	91.35%	92.64
all	94.04%	91.00%	92.50

Table 1: The results per chunk type of processing the test data with the double-pass method and majority voting. Our method outperforms most chunk type results mention in Buchholz et al. (1999) (F_{ADJP}=66.7, F_{ADVP}=77.9 F_{NP}=92.3, F_{PP}=96.8, F_{NP}=91.8) despite the fact that we have used about 80% less training data.

4 Concluding Remarks

We have evaluated three methods for recognizing non-recursive non-overlapping text chunks of arbitrary syntactical categories. In each method a memory-based learner was trained to recognize chunks represented in five different ways. We have examined nine different methods for combining the five results. A 10-fold cross-validation experiment on the training data of the CoNLL-2000 shared task revealed that (1) the combined results were better than the individual results, (2) the combination methods perform equally well and (3) the best performances of the three processing methods were similar. We have selected the double-pass method with majority voting for processing the CoNLL-2000 shared task data. This method outperformed an earlier text chunking study for most chunk types, despite the fact that it used about 80% less training data.

References

Sabine Buchholz, Jorn Veenstra, and Walter Daelemans. 1999. Cascaded grammatical relation assignment. In *Proceedings of EMNLP/VLC-99*. Association for Computational Linguistics.

Walter Daelemans, Jakub Zavrel, Ko van der Sloot, and Antal van den Bosch. 1999. *TiMBL: Tilburg Memory Based Learner, version 2.0, Reference Guide*. ILK Technical Report 99-01. http://ilk.kub.nl/.

Lance A. Ramshaw and Mitchell P. Marcus. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*. Association for Computational Linguistics.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the CoNLL-2000*. Association for Computational Linguistics.

Erik F. Tjong Kim Sang. 2000. Noun phrase recognition by system combination. In *Proceedings of the ANLP-NAACL 2000*. Seattle, Washington, USA. Morgan Kaufman Publishers.

Hans van Halteren, Jakub Zavrel, and Walter Daelemans. 1998. Improving data driven wordclass tagging by system combination. In *Proceedings of COLING-ACL '98*. Association for Computational Linguistics.

Sholom M. Weiss and Casimir A. Kulikowski. 1991. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems*. Morgan Kaufmann.