Boosting Trees for Clause Splitting

Xavier Carreras and **Lluís Màrquez**
TALP Research Center
LSI Department
Universitat Politècnica de Catalunya (UPC)
Jordi Girona Salgado 1–3. E-08034 Barcelona. Catalonia.
{carreras,lluism} @lsi.upc.es

1 Introduction

We present a system for the CoNLL-2001 shared task: the clause splitting problem. Our approach consists in decomposing the clause splitting problem into a combination of binary "simple" decisions, which we solve with the AdaBoost learning algorithm. The whole problem is decomposed in two levels, with two chained decisions per level. The first level corresponds to parts 1 and 2 presented in the introductory document for the task. The second level corresponds to the part 3, which we decompose in two decisions and a combination procedure.

2 System Architecture

AdaBoost (Freund and Schapire, 1997) is a general method for obtaining a highly accurate classification rule by combining many *weak* classifiers, each of which may be only moderately accurate. In designing our system, a generalized version of the AdaBoost algorithm has been used —AdaBoost.MH, (Schapire and Singer, 1999), which works with very simple domain partitioning weak hypotheses (decision stumps) with confidence rated predictions.

This particular boosting algorithm is able to work efficiently in very high dimensional feature spaces, and has been applied, with significant success, to a number of NLP disambiguation tasks, such as: POS tagging, PP–attachment disambiguation, text categorization, and word sense disambiguation.

In our particular setting, weak rules are extended to arbitrarily deep decision trees following the suggestion of (Schapire and Singer, 1999) and the definition presented in (Carreras and Màrquez, 2001). These more complex weak rules allow the algorithm to work in a higher dimensional feature space that contains conjunctions of simple features, and this fact has turned out to be crucial for improving results in the present domain.

As a general guideline for all the parts, our approach consists in giving the learning algorithm a very large number of binary simple features, assuming that the relevant features will be identified for constructing the classifiers. In section 2.1 the type of features used in the system are described. The remaining sections explain how the specific parts of the problem are solved.

2.1 Type of features

In the system, 4 types of features are used, which are the following. In the notation, indexes relative to the sequence of words are marked with the subscript w and indexes relative to the sequence of chunks are marked with the subscript c.

- **Word Window.** A word window of context size n centered in position i_w is the sequence of words in the sentence placed at $i_w + j_w$ positions, with $j_w \in [-n, +n]$. For each word in the window, its part-of-speech (POS) together with its relative position j_w forms a feature; for words whose POS is `CC`, `DT`, `EX`, `IN`, `MD`, `PDT`, `V*` or `W*` the word form with its relative position j_w is also a feature.
- **Chunk window.** A chunk window of context size n centered in position i_c is the sequence of chunks in the sentence placed at $i_c + j_c$ positions, with $j_c \in [-n, +n]$. The chunk window features are the tags of the chunks in the window together with the relative position j_c.
- **Sentence patterns.** A pattern represents the structure of the sentence which is rel-

evant for distinguishing clauses. The following elements are considered: a) Punctuation marks (`''`, ``` `` ```, `(`, `)`, `,`, `.`, `:`) and coordinate conjunctions (`CC`); b) The word "`that`", together with its POS; c) Relative pronouns, represented as `W`; and d) `VP` chunks. Given two word positions i_w and j_w, the corresponding feature will be the pattern representing the sentence from the position i_w to the position j_w.

- **Sentence features.** These features count the number of occurrences of relevant elements of the sentence. Specifically, we consider the chunks with tag `VP`, `WP` or `WP$`, the words whose POS is a punctuation mark and the word `that`. When available, S-points and E-points of the clause structure are also considered. Given a word in the i_w-th position, four features are generated for each element: two indicating the count of the element to the left and to the right of the i_w-th word, and two indicating the existence or not of the element to the left and to the right-hand side.

2.2 Parts 1 and 2

The purpose of these parts is the identification of the clause start positions (S-points) for the first part and clause end positions (E-points) for the second part. For each part, a classifier is trained to decide whether a given word matches the target of the part. Only words at the chunk boundaries are considered candidates: words at the beginning of chunks for S-points, and words at the end of chunks for E-points. The rest are always labeled as not S-points or not E-points, respectively.[1]

For a word candidate w the features considered are *word* and *chunk windows* of size 3, two *sentence patterns* —from the beginning to w, and from w to the end— and the *sentence features*, which include the left-hand side already identified S-points and E-points, respectively for each part. When these parts are included as subtasks of the part 3, *sentence features* for part 2 also include the S-points of the whole sentence.

[1] In the available data, a small proportion of words which are not chunk boundaries are labeled as S-points or E-points, but they correspond to annotation errors.

2.3 Part 3

The objective of this part is to build clauses, given the S-points and E-points of the sentence identified in the previous parts. Since this part is too complex to be handled directly, we have divided it into three chained subtasks: 1) Decide how many open brackets correspond to each S-point. 2) For each open bracket and each E-point to the right, i.e. a clause candidate, predict a confidence value for being a clause. 3) Match each open bracket with an E-point producing a consistent and confident clause splitting for the whole sentence.

2.3.1 Open Brackets

In this subtask the number of open brackets for each S-point must be predicted. Since our classification method only outputs binary decisions, our approach consists in predicting whether in a given S-point an additional bracket has to be opened or not. We assume that each S-point contains at least 1 open bracket. In practice, this subtask consists of two classifiers: from 1 to 2, and from 2 to 3 open brackets. Higher number of brackets are not frequent enough in the data sets to be learned.

The features used for this task, given an S-point in the i-th position, are the following: 1) *Word* and *chunk windows* of size 3. 2) Two *sentence patterns*, from the beginning to i, and from i to the end. 3) For each E-point to the right, a *sentence pattern* from i to the E-point, representing the possible clause starting at i. 4) *Sentence features*, including counts for S-points and E-points.

2.3.2 Clause candidates

Each open bracket identified in the last subtask paired with each E-point to the right forms a clause candidate. The purpose of this subtask is to predict a confidence value for the candidate to be a clause in the sentence. A classifier learned for classifying candidates as clauses or not will output such confidence values.

For each clause candidate starting at i and ending at j, the features are the following: 1) *Word* and *chunk windows* of size 3, centered in i. 2) One *sentence pattern* from i to j, representing the clause candidate. 3) *Sentence features*, counting the relevant elements separately from the beginning to i, from i to j and from j to the end.

2.3.3 Clause resolver

Given all clause candidates with its confidence value, a clause split must be output for the whole sentence, with the constraint that the splitting must be consistent, according to the definition of a clause. Given a list of clause candidates, our strategy consists in applying repeatedly the following procedure until the list is empty:

- Extract the clause candidate c_{max} with highest confidence value, and output c_{max} as a clause of the whole sentence.
- Remove from the list all clause candidate which are not consistent with c_{max}: a) candidates linked to the same c_{max} open bracket. b) candidates which contain the c_{max} open bracket or the c_{max} E-point, but not both.

3 Experiments and Results

The system parameters (i.e., number and depth of weak decision trees for the classifiers and window sizes for the features) are tuned using a validation set consisting of a 15% of the training sentences, having learned with the remaining 85%. All window sizes have been tuned to 3. The best performing depth for all tasks has been 3. For tasks 1, 2 and 3.2 the best number of weak rules is 2000; for the task 3.1, 1,500 weak rules are needed for deciding among 1 or 3 brackets, and only 100 weak rules for deciding among 2 and 3.

Table 1 shows for each task the number of examples, the number of positive examples and the number of generated features.

Part	Examples	Positive	Features
1	138,069	22,950	43,356
2	138,069	17,150	42,853
3.1: 1 or 2	23,075	1,519	36,240
3.1: 2 or 3	1,600	81	5,361
3.2	39,209	16,294	14,040

Table 1: Dimensions of the decision problems

Table 2 shows the results obtained on the development set and test set. The scores are lower on the test set. For parts 1 and 2, the F_1 scores are about 2 points lower, but for the part 3 the performance drops up to 6 points (this decrease is specially noticeable in the recall rate). We attribute that to the fact that part 3 strongly depends on the results of the first two parts.

development	precision	recall	$F_{\beta=1}$
part 1	95.77%	92.08%	93.89%
part 2	91.27%	89.00%	90.12%
part 3	87.18%	82.48%	84.77%

test	precision	recall	$F_{\beta=1}$
part 1	93.96%	89.59%	91.72%
part 2	90.04%	88.41%	89.22%
part 3	84.82%	73.28%	78.63%

Table 2: Final results for the shared task

4 Some Comments

The presented system follows a greedy approach for solving the whole problem. At a first stage, all S-points become one or more open brackets. At the combination stage clause candidates are selected greedily according to its confidence value. Despite these simplifying criterions, we think that the whole system performs quite well.

We are currently working on a system architecture which tries to overcome these limitations, by simulating a bottom-up parsing approach by means of solving simple binary decisions.

Acknowledgments

We would like to thank Montserrat Civit and Victoria Arranz for their invaluable help in the design of linguistically sensible features.

This research has been partially funded by the European Commission (NAMIC – IST-1999-12392) and the Spanish Research Department (Hermes – TIC2000-0335-C03-02). Xavier Carreras holds a grant by the Department of Universities, Research and Information Society of the Catalan Government.

References

X. Carreras and L. Màrquez. 2001. Boosting Algorithms for Anti–Spam E–mail Filtering. Submitted to Recent Advances in Natural Language Processing RANLP'01.

Y. Freund and R. E. Schapire. 1997. A Decision–Theoretic Generalization of On–line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

R. E. Schapire and Y. Singer. 1999. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, 37(3):297–336.