Boosting for Efficient Model Selection for Syntactic Parsing

Rachel Bawden[◇*] **Benoît Crabbé**[†‡]
[◇] LIMSI, CNRS, Univ. Paris-Sud, Universite Paris-Saclay
[†] Alpage, Univ. Paris-Diderot (Sorbonne Paris Cité) & Inria
[‡] Institut Universitaire de France
rachel.bawden@limsi.fr
benoit.crabbe@linguist.univ-paris-diderot.fr

Abstract

We present an efficient model selection method using boosting for transition-based constituency parsing. It is designed for exploring a high-dimensional search space, defined by a large set of feature templates, as for example is typically the case when parsing morphologically rich languages. Our method removes the need to manually define heuristic constraints, which are often imposed in current state-of-the-art selection methods. Our experiments for French show that the method is more efficient and is also capable of producing compact, state-of-the-art models.

1 Introduction

Model selection in feature-based parsing is crucial because features define a parsing model's capacity to predict syntactic structure. Choosing an optimal model is a trade-off between generalisation performance, compactness and parsing speed. Although too rarely mentioned, to this we should also add the speed of the selection method, which can determine how much of the search space can actually be explored. Parsing of languages other than English, and in particular morphologically rich languages, spurred on by initiatives such as the SPMRL (Statistical Parsing of Morphologically Rich Languages) shared tasks (Seddah et al., 2014), has received a heightened interest in recent years. For such languages, it is natural to want to exploit morphologically rich data to improve parsing performance. However the effect of this is an explosion in the number of possible models due to a huge number of potential features.

In this paper we introduce an efficient, language-independent model selection method for transition-based constituency parsing. It is designed for model selection when faced with a large number of possible feature templates, which is typically the case for morphologically rich languages, for which we want to exploit morphological information. The method we propose uses multi-class boosting (Zhu et al., 2006) for iterative selection in constant time, using virtually no *a priori* constraints on the search space. We do however introduce a pre-ranking step before selection in order to guide the selection process. We provide experiments on the adaptation of boosting for model selection in the parsing of high-dimensional data, using the transition-based lexicalised constituency parser presented in (Crabbé, 2015) and illustrating the feasibility of the method for our working language, French. Our results show that it is possible to produce high-performing, compact models much more efficiently than naive methods.

The structure of the paper is as follows. We begin by describing the transition-based parser used throughout the paper (Section 2). In Section 3 we review related work, both in model selection for parsing (Section 3.1) and on the boosting algorithm used (Section 3.2) in our proposal. In Section 4 we present our adaptation of the method for parsing and in Section 5 our experiments and results.

2 Discriminative constituency parsing

We base our experiments on the multilingual discriminative constituency parser described in (Crabbé, 2014; Crabbé, 2015) and inspired by transition-based parsing algorithms (Zhu et al., 2013). The parser

[*] This work was carried out while the first author was a Master's student at Alpage (Univ. Paris-Diderot & Inria).

Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, pages 1–11, Osaka, Japan, December 11-17 2016.

is an accurate and efficient transition-based lexicalised parser, capable of easily integrating lexical and morphological features. Following standard practice in transition-based parsing, the key structure for parsing is the *configuration* $C = \langle j, \mathbf{S} \rangle$ where j is the index of the parser in the queue of input tokens and $\mathbf{S}$ is a stack of partially constructed tree structures. A derivation $C_{0 \Rightarrow \tau}$ of length τ is represented by a sequence of configurations $C_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{\tau-1}}{\Rightarrow} C_\tau$ where each $a_i \in A$ is the parser *action* used to move from C_i to C_{i+1}. We use the same set A of actions as described in (Crabbé, 2014). Derivations are scored by a function of the form:

$$W(C_{0 \Rightarrow \tau}) = \sum_{i=0}^{\tau-1} \mathbf{w} \cdot \mathbf{\Phi}(a_i, C_i)$$

where $\mathbf{w} \in \mathbb{R}^D$ is a weight vector and where $\mathbf{\Phi}(a, C) \in \{0, 1\}^D$ denotes a function encoding a boolean-valued feature vector from a pair (a, C).

Let GEN_τ be the set of derivations of length τ. The best derivation in this set is defined as:

$$\hat{C}_{0 \Rightarrow \tau} = \underset{C_{0 \Rightarrow \tau} \in \text{GEN}_\tau}{\text{argmax}} \; W(C_{0 \Rightarrow \tau})$$

Weights for individual features are learnt using an averaged multi-class perceptron (Collins, 2002). They can either be optimised globally (over sequences of derivations) or locally (for each individual action). The first strategy is known to give better results for perceptron-based parsing (Zhang and Nivre, 2012).

2.1 Feature functions

The feature vector $\mathbf{\Phi}(a, C)$ is the result of a sequence of boolean feature functions $\phi_1(a, C) \ldots \phi_D(a, C)$, which have access to the action, to the top elements of the stack in C and to the beginning of the queue from index j. Each function is defined as per the following pattern:

$$\phi_l(a, C) = \begin{cases} 1 & \textbf{if} \quad \text{attr}_0 = a \\ & \textbf{and} \quad \text{attr}_1 = v_1 \\ & (\textbf{and} \quad \text{attr}_2 = v_2)? \\ & (\textbf{and} \quad \text{attr}_3 = v_3)? \\ 0 & \textbf{otherwise} \end{cases}$$

in which attr_0 is valued by the action $a \in A$ and the attributes $\text{attr}_{1 \leq i \leq 3}$ are extracted from configuration C. In practice, feature functions have access to the top three elements of the stack (see Figure 1) and the first four elements of the queue. They can address non-terminal categories, word-forms and morphological features (such as the part-of-speech (PoS), the gender, the number, the mood etc.) from the heads in the stack and from the words in the queue. Their values v_i are extracted from the configuration C, as illustrated in the following example:

$$\phi_l(a, C) = \begin{cases} 1 & \textbf{if} \quad \text{attr}_0 = a \\ & \textbf{and} \quad q_0.\text{word} = \text{“pot”} \\ & \textbf{and} \quad s_2.\text{cat} = \text{VP} \\ 0 & \textbf{otherwise} \end{cases}$$

As shown in the pattern above, a function can contain up to three attribute-value pairs, excluding the action. We refer to each pair as a condition, and refer to features as being uni-, bi- or tri-conditional depending on the number of conditions they contain.

2.2 Feature templates

It is common practice to use feature templates when defining hand-crafted models rather than to specify individual features. Feature templates are abstract feature representations in which only the attributes are specified, such that features with the same attribute types are grouped into sets. In the case of templates, which can also have up to three conditions, a condition refers simply to the attribute. For example, the bi-conditional template ‘q_0(word) & s_1(t, h, tag)’ represents all bi-conditional feature functions related to the word-form of the first queue element and to the PoS tag of the lexical head associated with the top of the second stack element.

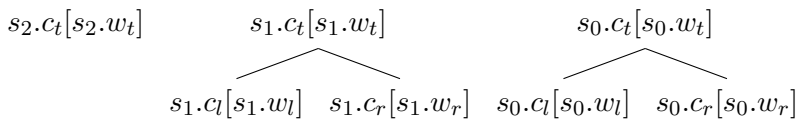

Figure 1: Stack elements for instantiation in feature functions. c_t, c_l and c_r represent non-terminal categories at three positions (top (t), left (l) and right (r)). Lexical information about the head is also available, indicated in squared brackets as $s_i.w_{\{t,l,r\}}$.

Since a feature vector is usually sparse, using templates is an advantage for computational reasons; the values of the feature functions can be dynamically and efficiently bound during parsing. They also enable a more compact and readable representation of models, making models easier to define manually.

When working with parsers that rely on templates, which is the case of most current implementations of existing feature-based parsers (Nilsson and Nugues, 2010; Ballesteros and Nivre, 2014; Crabbé, 2014, etc.), the template seems to be an acceptable level of granularity for specifying models. A selection process at the template-level rather than the feature-level has the advantage of being compatible with these parsers and also reduces the selection time by reducing the combinatorics of the selection process. We therefore focus our work on template selection.

3 Model selection for parsing

3.1 Previous work

For feature-based models, there are two main strategies for model selection. *Filter methods* remove features based on a static analysis, whereas *wrapper methods* iteratively refit the model by forward or backward selection. In the parsing literature, the wrapper method is the most prevalent. While some backward wrappers exist, provided that the template set is small (Attardi et al., 2007), most work focuses on forward wrappers, with a variety of constraints to reduce the search space and thus the time required. Nilsson and Nugues (2010) constrain the search space by imposing an order on stack and queue elements, under the assumption that more local elements are more useful than more distant ones. Ballesteros and Nivre (2014) and Ballesteros and Bohnet (2014) use a combination of forward and backward methods and fix heavy rule-based constraints on the order of templates selected. He et al. (2013) also implement template selection for discriminative parsing. Although applied to graph-based parsing, their work shares a likeness with our own, by their use of a pre-ranking wrapper to order templates prior to selection.

The reason for introducing such constraints is that wrapper methods are computationally intensive and can be known to take weeks to select a model, even with constraints and fully optimised, multi-processed implementations. Take for example the case of the forward wrapper. It starts with an empty model ($M \leftarrow \emptyset$). At each iteration, the template t from the pool of potential templates P that results in the highest overall accuracy gain is added to the model ($M \leftarrow M \cup t$). The process stops when the model's loss ceases to decrease. The most time-consuming part of the process is the training of the possible models, in order to select the template t that results in the highest accuracy gain. In principle, this requires fitting $|P|$ models at each iteration, and the size of the models requiring training grows at each iteration. Given that fitting a single parsing model can take hours (see Section 5.4), it is impractical to perform selection for parsing based on iteratively refitting a series of large and ever-growing models.

3.2 Model selection via boosting

We propose to overcome the limitations of the naive forward wrapper by using sequential additive fitting based on boosting (Freund and Schapire, 1999). Starting with an empty model ($M \leftarrow \emptyset$), additive fitting consists of evaluating the addition of a new template t by fitting t on its own before adding it ($M \leftarrow M \cup t$), without modifying the already fitted content M. At each iteration, as with the naive method, $|P|$ models need to be fit, but they are small and of constant size. As we will show in the

Algorithm 1 SAMME (Zhu et al., 2006)

▷ Data = $\{(x_1, y_1), \ldots, (x_N, y_N)\}$

1: Initialisation of data weights

$$w_i^{(1)} = \frac{1}{N}, \text{ i = 1, 2,..., N}$$

2: **for** iteration t=1 to T **do**

(i) Fit each weak learner $h_j(x)$ in the pool P to data using weights $w_i^{(t)}$

(ii) Calculate the weighted error of each weak learner $h_j(x)$

$$err_{h_j} = \frac{\sum_{i=1}^{N} w_i^{(t)} \mathbb{I}(y_i \neq h_j(x_i))}{\sum_{i=1}^{N} w_i^{(t)}}$$

(iii) Select the $h_j(x)$ with the lowest weighted error err_{h_j} provided that $err_{h_j} > \left(1 - \frac{1}{|Y|}\right)$.

Call the learner $g^{(t)}$ and its weighted error $err^{(t)}$

(iv) Calculate $\alpha^{(t)}$, where Y is the set of classes

$$\alpha^{(t)} = \log\left(\frac{1 - err^{(t)}}{err^{(t)}}\right) + \log\left(|Y| - 1\right)$$

(v) Update data weights using $\alpha^{(t)}$

$$w_i^{(t+1)} = w_i^{(t)} \cdot \exp\left(\alpha^{(t)} \mathbb{I}\left(y_i \neq g^{(t)}(x_i)\right)\right), \text{ i = 1,2,...,N}$$

(vi) Normalise weights such that $\sum_{i=1}^{N} w_i = 1$

end for

3: Prediction is the argmax of a weighted prediction of models $g^{(t)}$, t = 1, 2,..., T

$$f(\text{x}) = \text{argmax}_y \sum_{t=1}^{T} \alpha^{(t)} \mathbb{I}\left(g^{(t)}(x) = y\right)$$

remainder of the paper, this allows for a huge reduction in selection time, meaning that heavy constraints are not needed to reduce the search space.

We use the multi-class AdaBoost variant SAMME (Stagewise Additive Modelling using a Multi-class Exponential loss function) as described in (Zhu et al., 2006) and adapted here in Algorithm 1. The algorithm is designed for predictive modelling and provides the means of combining a set of *weak learners*[1] to produce a strong learner, by additively and iteratively selecting the best weak learner $g^{(t)}$ and calculating its coefficient $\alpha^{(t)}$ (its importance in the final model) until no more weak learners are available. An additive fit is achieved by encoding the exponential loss of already selected learners in a weight distribution over data examples, which is updated at each iteration. The algorithm comes with a theoretical guarantee that as long as the selected learner has a weighted accuracy above chance, the model's boosted accuracy will not decrease.

In the case of parsing, weak learners can be seen as weak parsers, trained each on a very small set of templates. It can be seen as iterative forward selection in that the selection of a weak parser constitutes the selection of the templates on which it is trained. In this paper, we use the term *weak parser* as an alias for the set of templates on which it is trained. Boosting has previously been used for feature selection for automatic face-detection in the domain of imagery (Viola and Jones, 2004) and for a variety of classification tasks by Das (2001). However to our knowledge, boosting methods have not yet been used in the context of template selection for parsing.

4 Adapting boosting for template selection for parsing

Although the algorithm has a theoretical guarantee, certain aspects must be reviewed to adapt it to template selection for parsing. Here we shall review four of these aspects, which prove essential, both in terms of providing a correctly functioning implementation of boosted selection and in terms of the time required for selection: (i) local training during selection, (ii) template grouping for weak parsers, (iii) a user-defined stopping criterion, and (iv) pre-ranking of weak parsers to reduce the pool size at each iteration.

[1] A weak learner (or *weak classifier*, *weak hypothesis*) is a classifier that performs better than random classification.

4.1 Local training of weak parsers

Ensemble methods such as boosting are notoriously problematic when it comes to structured prediction problems such as parsing that require globally optimised training (Cortes et al., 2014). Following Wang et al. (2007), who achieve good results on English and Chinese by boosting locally optimised parsers, we decide to train weak parsers locally during selection. However, unlike Wang et al. (2007), we perform boosting for the unique aim of selecting templates, rather than using the model fitted during boosting. In order to subsequently use the selected templates for parsing, we take the resulting template set, once selection is complete, and fit a globally optimised model. We make the assumption that locally boosted weak parsers provide a good approximation of a template set that can be used to produce a high-performing global model.[2]

4.2 Template grouping for weak parsers

In our approach, the smallest manipulable units for selecting parse models are templates. Instead of considering that each weak parser is trained on a single template, we choose to group templates in order to train larger and stronger weak parsers. The pool of weak parsers is therefore represented by the different template groups available for selection.

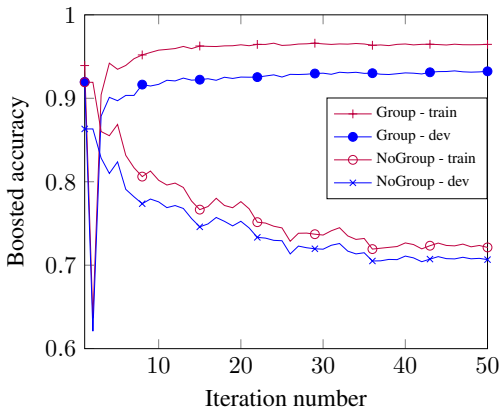

Figure 2: Boosted accuracy with and without grouping

Basis for grouping We use the conditions contained by templates (as defined in Section 2.2) as the basis for grouping. We group templates that share conditions, such that each group contains a single tri-conditional template as well as all the templates with a combination of the three conditions. Each group therefore contains seven templates. Note that uni- and bi-conditional templates necessarily belong to more than one group.

Why group? Grouping templates has several advantages: it allows for cases of co-prediction, it can accelerate model selection by reducing the number of weak parsers trained, and it strengthens the weak parsers. Although in theory handling individual templates would allow for a finer-grained selection, in practice, boosted accuracy does not increase as expected. It has previously been noted by Mukherjee and Schapire (2011) that the guarantee appears to be too weak when the learners perform only just above random. This can be seen in Figure 2, which shows the boosted accuracy for training and development sets with and without grouping. The scores of the resulting models are also significantly lower for individual rather than for grouped templates, and selection is more likely to terminate prematurely due to a lack of sufficiently strong weak parsers, effectively stunting the final model size.

4.3 Stopping criterion

The boosting algorithm has a last-resort stopping criterion, when there are no more weak parsers left. However this is not ideal for a parsing model; time efficiency at parse time is important and, depending on the number of weak parsers available, a model selected with this stopping criterion could be huge, and therefore slow at test time. We observe that the F-score of the retrained model continues to increase (albeit very gradually as the model size increases) as more and more templates are added (tested up to 200 templates), despite the fact that the weighted error during boosting appears to stabilise. We propose a practical stopping criterion, by which the maximum model size (the maximum number of templates) is defined in advance by the user, serving as a second stopping criterion for selection.

4.4 Limiting the pool size by pre-ranking

The size of the weak parser pool is a potential source of problems in terms of efficiency, given the possibility of a very large number of weak parsers. We therefore limit the pool size, by selecting the

[2] Although boosted accuracy is not in strictly perfect correlation with the accuracy of a retrained global model.

parsers from a pool of size k at each iteration, according to a certain criterion. The aim of model selection being to optimise the final performance of the parsing model, we introduce a pre-ranking step, in which all possible weak parsers are globally trained and ranked according to their accuracy. Although this is an approximation of the search space, pre-ranking should be able to guide the search. Weak parsers are not replaced in the pool once selected, which means that the pool is not limited to the original k best-ranked learners; each time a learner is selected, a new one becomes available.[3]

5 Experiments

The aim of our experiments is to investigate the properties of the boosting algorithm in terms of selection time and performance to test the feasibility of the selection method for high-dimensional search spaces. Our working language is French, for which we use a corpus with rich morphological annotations.[4]

5.1 Data

We conduct our experiments for French, using the French TreeBank (FTB) (Abeillé et al., 2003), from the 2014 SPMRL shared task (Seddah et al., 2014), but with automatic predictions of morphological tags[5] by MarMoT (Mueller et al., 2013), trained with ten-fold jackknifing. Templates can refer to the following morphological attributes: word-form, PoS tag, gender, number, mood and multi-word expression tags, as well as the syntactic categories of stack elements. They can access the top two stack elements and the next three queue elements, making a total of 36,050 possible templates and 34,220 possible weak parsers.[6] The corpus is divided into three sets: train, dev and test. Training data is used for pre-ranking and selection, and development data is used for tuning and comparing parameter values. Results on the test set are reported in Section 5.5. For pre-ranking, we use variants of the training set, which vary depending on the maximum length of the sentences they contain. We refer to these variants as FTB-10, FTB-20, FTB-50, where FTB-X contains all sentences from the original with a maximum sentence length of X. To avoid any confusion, we refer to the full set as FTB-all. Data characteristics can be found in Table 1.

Set	#Sents.	#Tokens	Avg. Len.	#(a, C)
FTB-20-train	4,903	63,239	12.90	184,811
FTB-50-train	13,006	330,711	25.43	979,127
FTB-all-train	14,759	443,113	30.02	1,314,580
FTB-all-dev	1235	38,820	31.43	115,225
FTB-all-test	2541	75,216	29.60	223,107

Table 1: Basic data characteristics

5.2 Setup

As described in Section 4, our weak parsers use template groups selected without replacement. During the pre-ranking step, different FTB-X-train variants are evaluated. Weak parsers are trained globally using a single perceptron iteration, a beam size of 4 and early updates (Collins and Roark, 2004). During selection, we also use a single perceptron iteration, but training is done locally. We evaluate the performance of selected models by retraining them globally using FTB-all data, with 30 perceptron iterations, a beam of size 8 and early updates. All experiments were run on an Intel(R) Xeon(R) CPU E5645 @ 2.40GHz. Training is multi-processed but all times reported are approximated for a single processor to enable a rough comparison.

[3] Replacement of learners is common in boosting, especially if they are decision stumps. However we find that with replacement, selection is slower and larger models are unobtainable.

[4] Although not amongst the most morphologically rich languages, the French data contains sufficiently rich morphological annotations to result in a huge number of possible templates, enabling us to test our selection method.

[5] In a realistic scenario where taggers are applied to raw texts before parsing, better parsing results are obtained when training is done on predicted rather than gold tags.

[6] The huge number of templates is due to the large number of morphological attributes available.

5.3 Parameter choices

We investigate the effect on performance and efficiency of selection of varying two parameters: (i) the data used to pre-rank the weak parsers and (ii) the size k of the pool during selection. We also compare against a non-boosted version in which weak learners are simply added in pre-ranking order, which is equivalent to greedy forward selection.

Data	# Sents.	Pre-ranking Time (hours)
FTB-20-train	4,903	63
FTB-50-train	13,006	496
FTB-all-train	14,759	743

Table 2: Pre-ranking times using different data types.

Selection method	Time (mins./iter.)
Pre-rank order	0
Boost, $k = 50$	18.53
Boost, $k = 100$	33.67

Table 3: Selection time per iteration.

Pre-ranking is the most time-consuming step in the process, and the speed is largely determined by training data size (see Table 2). The number of weak parsers is identical for all FTB data, yet there is a stark difference in pre-rank times; FTB-20-train is more than ten times faster than FTB-all-train. Average selection times per iteration, (see Table 3),[7] are directly correlated with the size of the pool k, but are fast compared to the pre-ranking step.

In Figure 3 we study the effect on model performance of varying pre-ranking data and the value of k for two types of model: a small model with a maximum size of 36 templates (for efficient parsing) and a larger model with a maximum size of 120 templates (for higher accuracy but slower parsing).[8]

For the 36-template model, all boosting runs outperform the selection method by pre-ranking only. The effect is greatest when the smaller datasets are used. The crossing pattern seen in the graph for $k = 50$ and $k = 100$ indicates that when the smaller dataset is used for pre-ranking, the larger value for k (of 100) is a better choice, and conversely, when the full dataset is used, the smaller value (of 50) is better. It is therefore possible to achieve almost comparable F-scores as when using full data for pre-ranking and a pool size of 50 by using a smaller dataset (and drastically reducing the time needed to pre-rank) and a larger, less constraining pool size of 100.

F-scores for the larger, 120-template models are higher, but for all methods we observed that the F-score starts to converge at such a large model size. Again, scores are superior when FTB-all-train data is used, but this time, the best model is reached using the simple pre-ranking order of template groups, as long as FTB-all-train is used to pre-rank. For these larger models, it appears that there is less need for the selection process, since a sufficiently large number of reasonably strong templates is all that is required. The disadvantage with this method is that all data is needed for pre-ranking, which, as we have shown above, is more time-consuming. However, as with the smaller models, boosting enables a high-performing model to be selected much faster, by using smaller pre-ranking data.

5.4 Topline model: Naive forward wrapper

We also implemented a naive forward wrapper (see Section 3.1), which we refer to as FWRAP. This method serves as a topline for parsing performance, as the method is a more accurate exploration of the model space, and, as it is very time-consuming, a baseline for selection time. As with the boosting method, templates are added by groups. We change the setup slightly to make the method feasible, using a smaller and less annotated dataset to reduce the number of weak parsers and the time taken to train them. Selection is performed using a variant of FTB-20-train, with the selection criterion being an increased accuracy on FTB-20-dev. We forbid templates from addressing morphological values other than the word-form, the PoS tag and a smoothed version of the word-form,[9] although we allow access to the third stack element and the fourth queue element. Importantly, the total number of weak parsers is far fewer than with the FTB data used above (9,880 vs. 34,220), which means that the initial setup is more

[7] We take the median selection time rather than the mean due to variation due to sporadic differences in machine usage.

[8] For all boosting methods for each parameter combination, an average is calculated over 5 runs to account for random variation due to the use of weighted resampling for training.

[9] Where words of fewer than two occurrences are replaced by a symbol $\$unknown\$$.

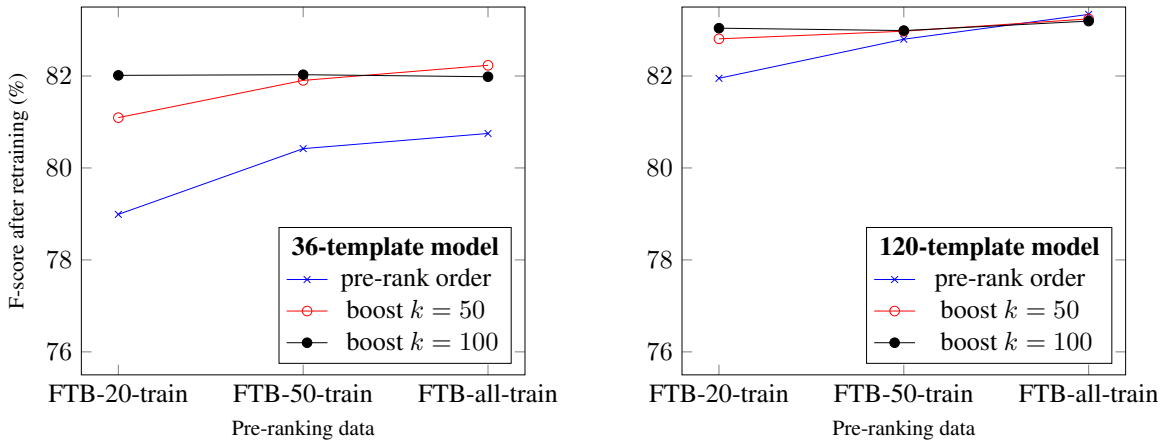

Figure 3: Average F-scores (over 5 separate runs) of selected models with a maximum size of 36 templates (left) and 120 templates (right) for different FTB pre-ranking data and different pool sizes (k).

favourable for FWRAP. Models are trained globally with a beam size of 4 and max-violation updates (Huang et al., 2012) for added speed. Since model sizes vary, the maximum number of perceptron iterations is 35, but we use a test for convergence, enabling training to stop early for smaller models.

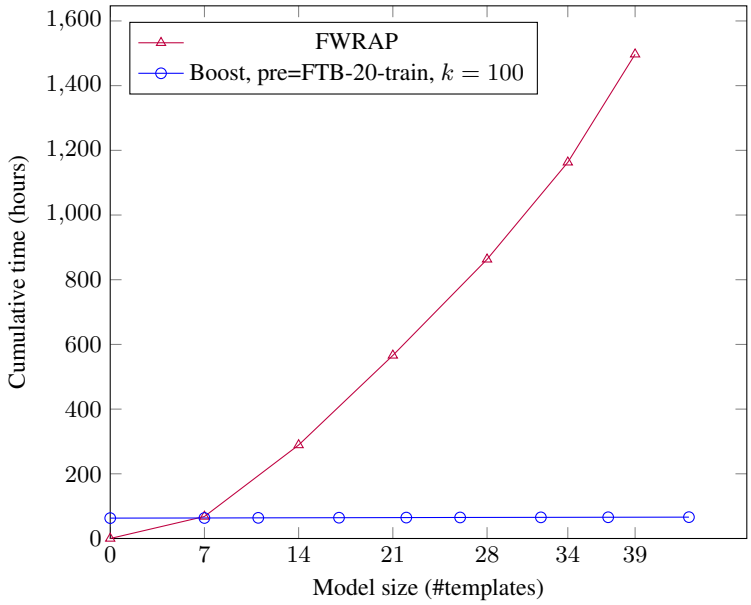

Figure 4: Comparison of cumulative selection time (including pre-ranking) in hours (approximated for a single processor) for FWRAP and a boosted method.

Figure 4 shows selection times for FWRAP and the boosting method (FTB-20-train, $k = 100$). Training an ever-growing model during selection for FWRAP results in an increasing selection time per iteration, whereas the boosting method's selection time is constant and considerably faster overall.

5.5 Results

In Table 4, we provide results for the final selected models on both the development and test sets, evaluated using `evalb`. We provide comparative scores of the Berkeley parser (Petrov et al., 2006), of the current highest performing single parser on French SPMRL data (Durrett and Klein, 2015), of the manually chosen model in (Crabbé, 2014) and of the model selected by FWRAP (Section 5.4). As before, we give our results for two model sizes, using the stopping criterion mentioned in Section 4.3 (a maximum size of 36 templates and of 120 templates).

Model	#tpls.	F-score (%) on FTB-all dev (P)	dev (NoP)	test (P)	test (NoP)	Select. time (hrs)	Parse speed (#Toks./Sec.)
Berkeley (Petrov et al., 2006)	-	79.74	-	80.38	80.73	-	-
Neural CRF (Durrett and Klein, 2015)	-	80.65	-	81.25	-	-	-
Manual (Crabbé, 2014)	68	-	81.79	-	81.43	-	-
FWRAP (Section 5.4)	34	81.80	83.53	**81.52**	83.43	1,163	3,646
FTB-all-train/Pre-rank order	35	79.13	80.75	78.50	80.36	743	3,607
FTB-all-train/$k = 50$	36	**80.73**	**82.42**	79.91	81.69	745	3,478
FTB-20-train/$k = 100$	36	80.58	82.11	**80.34**	**82.23**	**66**	3,426
FTB-all-train/Pre-rank order	117	81.63	83.34	81.14	82.97	743	1,347
FTB-all-train/$k = 100$	119	**81.84**	**83.39**	**81.26**	**83.08**	762	1,328
FTB-20-train/$k = 100$	119	81.59	83.29	80.98	82.71	**80**	1,295

Table 4: Comparison of models. Results are given for evaluation with punctuation (P) and without punctuation (NoP). Selection times include pre-ranking. All times are approximated for a single processor.

The best-performing model is selected using FWRAP, with a higher F-score than the current state-of-the-art single parser (Durrett and Klein, 2015).[10] However the selection time of 1,163 hours means that the method is not very tractable. Our final boosting-based results show the same general pattern as obtained on the development set in the previous section, showing that the approach is robust to a change in dataset, and also to a scaling-up of model training to full optimisation. As in Section 5, the best compromise is the boosted model with pre-ranking on FTB-20-train and $k = 100$. It results in one of the highest scores for the compact model, and the overall selection time is greatly reduced. Importantly, it also outperforms the state-of-the-art manual model (Crabbé, 2014), whilst being almost twice as compact. The results show that for larger models, selection via boosting is less useful, with comparable results between boosted and the "pre-rank only" model. If a large model is needed, simply sequentially adding individually trained weak parsers based on their accuracies can produce a high-performing model.

6 Conclusion

We have successfully performed efficient model selection by using stepwise additive fitting. Experiments on high-dimensional data for French show that compact, state-of-the-art parse models can be achieved, and confirm our hypothesis that locally trained parsers can provide a good approximation for globally trained models. Unlike current template selection methods for parsing (Nilsson and Nugues, 2010; Ballesteros and Nivre, 2014), we use no hand-written heuristic constraints to limit the search space, instead opting for pre-ranking of weak parsers to guide the search. Although pre-ranking is relatively time-consuming, the times are very low compared to standard selection methods. We provide a realistic selection scenario, which involves using only a portion of the training data for pre-ranking, capable of selecting a large model (120 templates) in a couple of hours (when multi-processed), as well as compact models that outperform state-of-the-art manually defined models. A release of all source code is available online at `https://bitbucket.org/rbawden/hyparse-boost-feature-selection`.

In future works, we will extend the approach to a variety of other languages, in particular morphologically rich languages (e.g. Arabic, Hungarian or Korean) and extend the possible templates used to take into account further features. Another interesting perspective would be to study the combination of our approach with constraining heuristics, although language-specific and manual rules would be required.

References

Anne Abeillé, Lionel Clément, and François Toussenel. 2003. Building a Treebank for French. In *Treebanks*, chapter 10, pages 165–188. Kluwer, Dordrecht.

[10]Note that using parser ensembles and reranking may still lead to higher scores at the expense of parse time efficiency (Björkelund et al., 2014).

Giuseppe Attardi, Felice Dell'Orletta, Maria Simi, Atanas Chanev, and Massimiliano Ciaramita. 2007. Multilingual Dependency Parsing and Domain Adaptation using DeSR. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP–CoNLL '07)*, pages 1112–1118, Prague, Czech Republic.

Miguel Ballesteros and Bernd Bohnet. 2014. Automatic Feature Selection for Agenda-Based Dependency Parsing. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING '14)*, pages 794–805, Dublin, Ireland.

Miguel Ballesteros and Joakim Nivre. 2014. MaltOptimizer: Fast and effective parser optimization. *Natural Language Engineering*, pages 1–27.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Müller, Wolfgang Seeker, and Zsolt Szántó. 2014. The IMS-Wrocław-Szeged-CIS Entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax Meet Unlabeled Data. In *Proceedings of the 1st Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages (SPMRL–SANCL '14)*, pages 97–102, Dublin, Ireland.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL '04)*, pages 111–118, Barcelona, Spain.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '02)*, Philadelphia, Pennsylvania, USA.

Corinna Cortes, Mehryar Mohri, and Umar Syed. 2014. Deep Boosting. In *Proceedings of the 31st International Conference on Machine Learning (ICML '14)*, volume 32, pages 1179–1187, Beijing, China.

Benoît Crabbé. 2014. An LR-inspired generalized lexicalized phrase structure parser. In *Proceedings of the 25th International Conference on Computational Linguistics (COLING '14)*, pages 541–552, Dublin, Ireland.

Benoît Crabbé. 2015. Multilingual discriminative lexicalized phrase structure parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP '15)*, pages 1847–1856, Lisbon, Portugal.

Sanmay Das. 2001. Filters, Wrappers and a Boosting-Based Hybrid for Feature Selection. *Proceedings of the 18th International Conference on Machine Learning (ICML '01)*, pages 74–81.

Greg Durrett and Dan Klein. 2015. Neural CRF Parsing. In *Proceedings of the Association for Computational Linguistics (ACL '15)*, Beijing, China.

Yoav Freund and Robert E. Schapire. 1999. A Brief Introduction to Boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI '99)*, volume 2, pages 1401–1406, Stockholm, Sweden.

He He, Hal Daumé III, and Jason Eisner. 2013. Dynamic Feature Selection for Dependency Parsing. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP '13)*, pages 1455–1464, Seattle, Washington, USA.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL '12)*, pages 142–151, Montreal, Canada.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient Higher-Order CRFs for Morphological Tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP '13)*, pages 322–332, Seattle, Washington, USA.

Indraneel Mukherjee and Robert E. Schapire. 2011. A Theory of Multiclass Boosting. *Journal of Machine Learning Research*, 14(1):437–497.

Peter Nilsson and Pierre Nugues. 2010. Automatic Discovery of Feature Sets for Dependency Parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING '10)*, pages 824–832, Beijing, China.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning Accurate, Compact, and Interpretable Tree Annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (COLING–ACL '06)*, pages 433–440, Sydney, Australia.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages (SPMRL–SANCL '14)*, pages 103–109, Dublin, Ireland.

Paul Viola and Michael J. Jones. 2004. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154.

Qin Iris Wang, Dekang Lin, and Dale Schuurmans. 2007. Simple training of dependency parsers via structured boosting. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '07)*, pages 1756–1762, Hyderabad, India.

Yue Zhang and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING '12)*, pages 1391–1400, Mumbai, India.

Ji Zhu, Ann Arbor, and Trevor Hastie. 2006. Multi-class AdaBoost. Technical report, Stanford University.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL '13)*, pages 434–443, Sofia, Bulgaria.