Filtering-Ranking Perceptron Learning for Partial Parsing

A. Xavier Carreras (carreras@lsi.upc.es)
TALP, LSI, Technical University of Catalonia (UPC)
Dept. LSI – Campus Nord UPC. 08034 Barcelona. Spain.

B. Lluís Màrquez (lluism@lsi.upc.es)
TALP, LSI, Technical University of Catalonia (UPC)

C. Jorge Castro (castro@lsi.upc.es)
LSI, Technical University of Catalonia (UPC)

Abstract. This work introduces a general phrase recognition system based on perceptrons, and a global online learning algorithm to train them together. The method applies to complex domains in which some structure has to be recognized. This global problem is broken down into two layers of local subproblems: a filtering layer, which reduces the search space by identifying plausible phrase candidates; and a ranking layer, which builds the optimal phrase structure by discriminating among competing phrases. A recognition-based feedback rule is presented which reflects to each local function its committed errors from a global point of view, and allows to train them together online as perceptrons. As a result, the learned functions automatically behave as filters and rankers, rather than binary classifiers, which we argue to be better for this type of problems. Extensive experimentation on partial parsing tasks gives state-of-the-art results and evinces the advantages of the global training method over optimizing each function locally and independently.

Keywords: Online Learning, Perceptrons, Natural Language Processing, Partial Parsing, Sequential Data, Structure Recognition

1. Introduction

Over the past few years, many machine learning methods have been successfully applied to Natural Language Processing tasks in which phrases of some type have to be recognized. Generally, given an input sentence —as a sequence of words— the task is to predict a bracketing for the sentence representing a structure of phrases, either sequential or hierarchical. For instance, syntactic analysis of Natural Language provides several problems of this type, such as partial parsing tasks or full parsing. Although there exists an important body of work on probabilistic approaches for tagging and segmenting sequential data (from generative models, such as Hidden Markov Models, to conditional exponential models, such as Maximum Entropy Markov Models or Conditional Random Fields), in this paper we focus on the paradigm that makes use of discriminative learning to train the components of the phrase recognition model.

The usual approach in the discriminative framework consists of decomposing the global phrase recognition problem into a number of local learnable subproblems. Then, a decoder algorithm infers the global solution from the outcomes of the local subproblems. The basic task in partial parsing, known as Chunking (Abney, 1991; Tjong Kim Sang & Buchholz, 2000), consists of finding non-recursive base phrases (or *chunks*) organized in sequence. A typical approach is to perform a tagging. In this case, local subproblems include learning whether a word *opens*, *closes*, or is *inside* a phrase of some type (noun phrase, verb phrase, etc.), and the inference process consists of sequentially computing the optimal tag sequence which encodes the phrases, by means of dynamic programming (Punyakanok & Roth, 2001) or beam search (Kudo & Matsumoto, 2001). When hierarchical structure has to be recognized, additional local decisions are required to determine the embedding of phrases, resulting in a more complex inference process which recursively builds the global solution. Such type of systems can be found for deeper levels of partial parsing (Tjong Kim Sang & Déjean, 2001; Carreras, Màrquez, Punyakanok, & Roth, 2002; Kudo & Matsumoto, 2002), or in full parsing (Magerman, 1996; Ratnaparkhi, 1999; Haruno, Shirai, Ooyama, & Aizawa, 1999). In general, a learning system for these tasks makes use of several learned functions which interact in some way to determine the final structure.

A usual methodology for solving the local subproblems is to use a discriminative learning algorithm to learn a classifier for each local decision. Each individual classifier is trained separately from the others, maximizing some local measure such as the accuracy of the local decision. However, when performing the phrase recognition task, the classifiers are used together and dependently, in the sense that one classifier predictions' may affect the prediction of another. Indeed, the global performance of a system is measured in terms of precision and recall of the recognized phrases, which, although related, is not the local classification accuracy measure for which the local classifiers are usually trained.

Recent works on the area provide alternative learning strategies in which the learning process is guided from a global point of view. Collins (2000) recasts the parsing task into a ranking problem. First, a permissive baseline parsing model is used to generate a list of candidate parses, which are then ranked by a global model so that the best solution is ranked the highest. The global model is trained discriminatively for that purpose, and takes into account representations of complete parse trees. Later, Collins (2002) shows that if there exists a decoding algorithm for the global model (such as Viterbi for a Markov tagger), then its parameters can be trained in the online setting with Perceptron. The overall model is directly a global ranking function, avoiding the use of a baseline model, and is trained online by receiving feedback from the output of the decoder. Similar in nature, Crammer and Singer present online algorithms for multiclass classification (2003b) and category ranking (2003a), in which several perceptrons receive feedback from the global solution they produce over a training instance. In all cases, there is a notion of global margin: given a global discriminatory function predicting scores for solutions, the margin of an instance is defined as the difference in scores between the correct and the top-ranked solutions. Thus, rather than simply making margins positive, there have been also attempts to maximize the global margins in a training sample, such as Hidden Markov Support Vector Machines (Altun, Tsochantaridis, & Hofmann, 2003) or Max-Margin Markov Networks (Taskar, Guestrin, & Koller, 2003).

Regarding the pure probabilistic methods there has been also interest on training globally avoiding to some extent the locality of conditional models. In this way, Conditional Random Fields (CRF) provide techniques to estimate a single exponential model for the joint probability of the entire sequence of labels given the sequence of observations (Lafferty, McCallum, & Pereira, 2001). More recently, Dynamic CRFs represent a generalization of linear–chain CRFs, which are able to represent several subtasks jointly in a single graphical model and to capture interactions between them (Sutton, Rohanimanesh, & McCallum, 2004).

In this paper we introduce a learning architecture based on filters and rankers for the general task of recognizing phrases in a sentence. The strategy for recognizing phrases is adopted from (Carreras et al., 2002), and can be sketched as follows. Given a sentence, learning is first applied at word level to identify phrase candidates of the solution. Then, learning is applied at phrase level to score phrase candidates and discriminate among competing ones. This second layer allows to deal with complete candidate phrases and partially constructed phrase structures. Thus, an advantage of working at this level is that rich and informed feature representations can be used to exploit structural properties of the examples, possibly through the use of kernel functions. However, a disadvantage of working with high level constructs is that the number of candidates to explore increases (e.g., there is a quadratic number of phrases with respect to the number of words in a sequence) and the search space may become prohibitively expensive to explore. For this reason, the word–level layer of our architecture plays a crucial role by filtering out non plausible phrase candidates and thus reducing the search space in which the high–level layer operates. The decoding strategy can be seen as an inference process which, guided by the learned functions, explores only a plausible set of coherent solutions to find the best scored one.

As a main contribution, we propose the FR-Perceptron learning algorithm to train the decisions in the system as linear functions, all in one go. The learning strategy follows Collins (2002) in two main aspects. First, it works online at sentence level: when visiting a sentence, the perceptrons are first used to recognize the set of phrases in it, and then updated according to the correctness of the solution. Second, the type of update is conservative since it operates only on the mistakes of the global solution. The algorithm presented here extends Collins' in that not only it trains a score function for ranking potential output labelings, but also trains the filtering component which provides candidates to the ranker. The extension is in terms of the feedback rule for the perceptron, which reflects to each individual function its committed errors when recognizing a set of phrases. As a result, the learned functions are automatically approximated to behave as word filters and phrase rankers, and thus, become adapted to the recognition strategy.

This paper extends and gives a comprehensive empirical study of two preliminary works (Carreras & Màrquez, 2003a, 2003b). Regarding the analysis of the algorithm a convergence proof is presented. Regarding the empirical study, we provide extensive experimentation on two relevant problems of the Partial Parsing domain, namely base Chunking and Clause Identification. Moreover, we incorporate, in the experimental architecture, the results of Freund and Schapire (1999) on voted perceptrons to produce robust predictions and allow the use of kernel functions. The performance achieved by the presented learning architecture is comparable to the state–of–the–art systems on base chunking and substantially better in the recognition of clauses. Besides, the experiments presented help understanding the behavior of the different components of the architecture and give evidence about its advantages compared to other standard alternative learning strategies.

The rest of the article is organized as follows. Section 2 formalizes the problem of recognizing phrases and describes the strategy we follow for the recognition. Section 3 introduces the perceptron learning algorithm for the task and provides convergence analysis. Section 4 describes two particular partial parsing problems and a feature specification to solve them with learning techniques. Extensive experimentation on these problems is presented in section 5. Finally, section 6 summarizes and discusses some lines for further research.

2. Phrase Recognition

2.1. Formalization

Let x be a sentence consisting of a sequence of n words $[x_1, x_2, \ldots, x_n]$, belonging to the sentence space $\mathcal{X}$. Let $\mathcal{K}$ be a predefined set of phrase categories. For example, in the syntactic parsing task $\mathcal{K}$ may include, among others, noun phrases, verb phrases, prepositional phrases, and clauses. A *phrase*, denoted as $(s,e)_k$, is the sequence of consecutive words spanning from word x_s to word x_e, having $s \leqslant e$, with category $k \in \mathcal{K}$.

Let $p_1 = (s_1, e_1)_{k_1}$ and $p_2 = (s_2, e_2)_{k_2}$ be two different phrases. We define that p_1 and p_2 *overlap* iff $s_1 < s_2 \leqslant e_1 < e_2$ or $s_2 < s_1 \leqslant e_2 < e_1$, and we note it as $p_1 \sim p_2$. Also, we define that p_1 is *embedded* in p_2 iff $s_2 \leqslant s_1 \leqslant e_1 \leqslant e_2$, and we note it as $p_1 \prec p_2$.

Let $\mathcal{P}$ be the set of all possible phrases, expressed as $\mathcal{P} = \{(s,e)_k \mid 1 \leqslant s \leqslant e,\ k \in \mathcal{K}\}$. In a phrase recognition problem, a *solution* for an input sentence x is a finite set y of phrases which is *coherent* with respect to some *constraints*. We consider two types of constraints: overlapping and embedding. For the problem of recognizing sequentially organized phrases, often referred to as *chunking*, phrases are not allowed to overlap or embed. Thus, the solution space can be formally expressed as $\mathcal{Y} = \{y \subseteq \mathcal{P} \mid \forall p_1, p_2 \in y \;\; p_1 \not\sim p_2 \wedge p_1 \not\prec p_2\}$. More generally, for the problem of recognizing phrases hierarchically organized, a solution is a set of phrases which do not overlap but may be embedded. Formally, the solution space is $\mathcal{Y} = \{y \subseteq \mathcal{P} \mid \forall p_1, p_2 \in y \;\; p_1 \not\sim p_2\}$. Note that, for simplicity and readability, we have defined the $\mathcal{P}$ and $\mathcal{Y}$ sets independently from concrete word sequences. However, whenever we are referring to the $\mathcal{P}$ and $\mathcal{Y}$ sets in the context of a concrete input sequence $[x_i]_{i=1}^n$ we implicitly restrict to phrases $(s,e)_k$ which span actual word sequences in x (i.e., $1 \leqslant s \leqslant e \leqslant n$), and to solutions formed with these phrases.

The learning problem for phrase recognition is as follows. Given a training set $S = \{(x^1, y^1), \ldots, (x^m, y^m)\}$, where x^i are sentences in $\mathcal{X}$ and y^i are solutions in $\mathcal{Y}$, the goal is to learn a function $\mathrm{R} : \mathcal{X} \rightarrow \mathcal{Y}$ which correctly recognizes phrases on unseen sentences. In order to evaluate a phrase recognition system, the standard measures for recognition tasks are used: *precision* (p) —the proportion of recognized phrases that are correct—, *recall* (r) —the proportion of solution phrases that are correctly recognized— and their harmonic mean $\mathrm{F}_{\beta=1}$, which is taken as the standard performance measure to compare systems. Let $|\cdot|$ be the number of elements in a set, the computation of the evaluation measures in a test set $\{(x^i, y^i)\}_1^l$ can be expressed as follows:

$$p = \frac{\sum_{i=1}^{l} |y^i \cap \mathrm{R}(x^i)|}{\sum_{i=1}^{l} |\mathrm{R}(x^i)|} \qquad r = \frac{\sum_{i=1}^{l} |y^i \cap \mathrm{R}(x^i)|}{\sum_{i=1}^{l} |y^i|} \qquad \mathrm{F}_{\beta=1} = \frac{2\,p\,r}{p+r}$$

2.2. Recognizing Phrases – The Model

The model for recognizing phrases is described here as a function $\mathrm{R} : \mathcal{X} \rightarrow \mathcal{Y}$ which, given a sentence x, identifies the set of phrases y of x. We assume two components within this function, both being learning components of the recognizer. First, we assume a filtering function F which, given a sentence x, identifies a set of candidate phrases, not necessarily coherent, for the sentence, $\mathrm{F}(x) \subseteq \mathcal{P}$. Second, we assume a *score* function which, given a phrase, produces a real-valued prediction indicating the plausibility of the phrase.

The Phrase Recognizer is a function which searches a coherent phrase set for a sentence x according to the following optimality criterion:

$$\mathrm{R}(x) = \arg \max_{y \subseteq \mathrm{F}(x) \;|\; y \in \mathcal{Y}} \sum_{(s,e)_k \in y} \mathrm{score}((s,e)_k, x, y) \tag{1}$$

That is, among all the coherent subsets of candidate phrases, the optimal solution is defined as the one whose phrases maximize the summation of phrase scores.

The function F is only used to reduce the search space of the R function. Note that the R function constructs the optimal phrase set by evaluating scores of phrase candidates, and, that there is a quadratic number of possible phrases with respect to the length of the sentence (that is, the size of $\mathcal{P}$ set). Thus, considering straightforwardly all phrases in $\mathcal{P}$ would result in a very expensive exploration. The function F is intended to filter out phrase candidates from $\mathcal{P}$ by applying decisions at word level. A simple setting for this function is a *start-end* classification for each phrase type: each word of the sentence is considered as *k-start* —if it is likely to start a type-k phrase— and as *k-end* —if it is likely to end a type-k phrase. Each *k-start* word x_s with each *k-end* word x_e, having $s \leqslant e$, form the phrase candidate $(s,e)_k$. Assuming *start* and *end* binary classification functions, $\mathrm{h}_{\mathrm{S}}^k$ and $\mathrm{h}_{\mathrm{E}}^k$, for each type $k \in \mathcal{K}$, the filtering function is expressed as:

$$\mathrm{F}(x) = \{\, (s,e)_k \in \mathcal{P} \mid \mathrm{h}_{\mathrm{S}}^k(x_s, x) = +1 \,\wedge\, \mathrm{h}_{\mathrm{E}}^k(x_e, x) = +1 \,\}$$

Alternatives to this setting for F may be to consider a single pair of *start-end* classifiers, independent of the phrase types, or to perform a different tagging for identifying phrases, such as a *begin-inside* classification. In general, each classifier of the F function will be applied to each word in the sentence, and deciding the best strategy for identifying phrase candidates will depend on the sparseness of phrases in a sentence, the length of phrases and the number of categories.

Once the phrase candidates are identified, the optimal coherent phrase set is selected according to (1). Due to its nature, there is no need to explicitly enumerate each possible coherent phrase set, which would result in an exponential exploration. Instead, by guiding the exploration through the problem constraints and using dynamic programming the optimal coherent phrase set can be found in polynomial time with respect to the sentence length. For chunking problems, the solution can be found in quadratic time by performing a Viterbi-style exploration from left to right (Punyakanok & Roth, 2001). When embedding of phrases is allowed, a cubic-time bottom-up exploration is required (Carreras et al., 2002). As noted above, in either cases there will be the additional cost of applying a quadratic number of decisions for scoring phrases.

3. Online Learning via Recognition Feedback

In this section we describe an online learning strategy for training the learning components of the Phrase Recognizer, namely the *start-end* classifiers in F and the *score* function. In doing so, the following learning challenges must be faced. First, the learning algorithm should optimize the sentence-level F_1 measure, rather than accuracies on the local decisions. Second, several functions must be learned taking into account their interactions –recall that the *start-end* functions define the input space to *score* function, and that the *score* function is applied recursively when dealing with hierarchical structure. Also, there is a quadratic number of phrase candidates over the sentence length, which makes the exploration of the entire space computationally prohibitive in datasets of real size. So, the algorithm should be efficient at training the *score* function in terms of the number of phrase examples considered.

Each learning component is implemented as a linear separator on a feature space defined by a representation function. A linear separator is a function $\mathrm{h}_{\mathbf{w}} : \mathbb{R}^d \to \mathbb{R}$ parametrized by a vector $\mathbf{w}$ in $\mathbb{R}^d$, which, given an instance $\mathbf{x} \in \mathbb{R}^d$, outputs as prediction the inner product between $\mathbf{w}$ and $\mathbf{x}$: $\mathrm{h}_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$. The representation function $\phi : \mathcal{X} \to \mathbb{R}^d$ takes an instance x belonging to some space $\mathcal{X}$ and outputs a vector in $\mathbb{R}^d$ with which the linear separator operates.

The function F is composed of two classifiers per phrase type, $\mathrm{h}_{\mathrm{S}}^k$ and $\mathrm{h}_{\mathrm{E}}^k$, each of which predicts whether a word x_i starts (S) or ends (E) a phrase of type k, respectively. Thus, the F function is formed by a prediction vector for each classifier, noted as $\mathbf{w}_{\mathrm{S}}^k$ or $\mathbf{w}_{\mathrm{E}}^k$, and a unique shared representation function ϕ_{w} which maps a word in context into a feature vector. A classifier prediction on a word x_i is computed as $\mathrm{h}_{\mathrm{S}}^k(x_i, x) = \mathbf{w}_{\mathrm{S}}^k \cdot \phi_{\mathrm{w}}(x_i, x)$, and similarly for the $\mathrm{h}_{\mathrm{E}}^k$, and the sign is taken as the binary decision.

The *score* function computes a real-valued score for a phrase candidate $(s, e)_k$. We implement this function with a prediction vector $\mathbf{w}^k$ for each type $k \in \mathcal{K}$, and also a shared representation function ϕ_{p} which maps a phrase into a feature vector. The score prediction is then given by the expression: $\mathrm{score}((s, e)_k, x, y) = \mathbf{w}^k \cdot \phi_{\mathrm{p}}((s, e)_k, x, y)$.

It is worth noting that the representation function ϕ_{p} takes the global solution y as an argument. When the task is performed, the function R actually builds the solution y with dynamic programming, making use of the function ϕ_{p} to score phrases. In practice, thus, ϕ_{p} has only access to a partial built of the solution y. In particular, when embedding of phrases is not allowed, while visiting a phrase $(s, e)_k$ the solution y will contain only the phrases to the left of the x_s word (due to the left-to-right exploration). When embedding is allowed, the solution y will contain only the phrases included within the $[x_s, \ldots, x_e]$ fragment (due to the bottom-up exploration).

3.1. The FR-Perceptron Learning Algorithm

We introduce a mistake-driven online learning algorithm for training the parameter vectors of the Phrase Recognizer all together. We name the algorithm FR-Perceptron since it is a Perceptron-based learning algorithm that approximates the prediction vectors in F as *Filters* of words, and the score vectors as *Rankers* of phrases. The algorithm, presented in Figure 1, starts with all vectors initialized to $\mathbf{0}$, and then runs repeatedly in a number of epochs T through all the sentences in the training set. Given a sentence, it predicts its optimal phrase solution as specified in (1) using the current vectors. As in the traditional Perceptron algorithm, if the predicted phrase set is not perfect the vectors responsible of the incorrect prediction are updated additively within the *recognition_learning_feedback* function.

The recognition-based feedback function (presented in the second part of Figure 1) has been derived by analyzing the dependencies between each function and a global solution, and naturally fits the phrase recognition setting. To do so, the rule tracks the interaction between the two layers of the recognition process. The *start-end* layer filters out phrase candidates for the scoring layer. Thus, misclassifying the boundary words of a correct phrase blocks the generation of the candidate and produces a missed phrase. In this case, we move the *start* or *end* prediction vectors toward the misclassified boundary words of a missed phrase. When an incorrect phrase is predicted, we move away the prediction vectors from the *start* or *end* words, provided that they are not boundary words of a phrase in the correct solution. Note that we deliberately do not care about false positives on the filtering layer, i.e., *start* or *end* words which do not finally over-produce a phrase, since these local errors do not hurt the global performance of the recognizer. This fact is crucial at explaining the filtering behavior of the *Start-End* layer.

Regarding the *Score* layer, each category prediction vector is moved toward missed phrases and moved away from over-predicted phrases. It is important to note that the feedback rule operates only on the basis of the global predicted solution $\hat{y}$, avoiding to update the *score* function on each of its local predictions on phrase candidates. This fact has the effect of approximating the behavior of the *score* function as a *ranker* over the set of candidate phrases, assigning higher predictions to the phrases in the solution than to the other incorrect competing phrases. As a consequence, this simple feedback rule tends to approximate the desired behavior of the global R function, that is, to make the summation of the scores of the correct phrase set maximal with respect to other phrase set candidates.

3.2. Discussion on the FR-Perceptron Algorithm

The update strategy of the FR-Perceptron is *ultraconservative* in the way defined by Crammer and Singer for multiclass classification (2003b) or category ranking problems (2003a). In these works, a set of prototype vectors (one for each class or category) are trained online with algorithms based on Perceptron. The notion of ultraconservative online

algorithm FR-Perceptron
  input: a training set $S = \{(x^i, y^i)\}_{i=1}^m$
  define: $W = \{\mathbf{w}_\mathrm{S}^k, \mathbf{w}_\mathrm{E}^k, \mathbf{w}^k \mid k \in \mathcal{K}\}$
  initialize: $\forall \mathbf{w} \in W \; \mathbf{w} := \mathbf{0}$
  for $t = 1 \ldots T$
    for $i = 1 \ldots m$
      $\hat{y} := \mathrm{R}_W(x^i)$
      recognition_learning_feedback$(W, x^i, y^i, \hat{y})$
  output: the vectors in W
end-algorithm

The **recognition_learning_feedback**$(W, x, y^*, \hat{y})$ function is defined by cases as follows. In the notation, let goldS(x_i, k) and goldE(x_i, k) be, respectively, the perfect indicator functions for *start* and *end* boundaries of phrases of type k. That is, they return 1 if word x_i starts/ends some k-phrase in y^* and -1 otherwise. Also, for simplicity, we do not write the $x, \hat{y}$ parameters in the representation functions.

1. Phrases correctly identified: $\forall (s,e)_k \in \; y^* \cap \hat{y}$:
   - Do nothing, since they are correct.
2. Missed phrases: $\forall (s,e)_k \in \; y^* \setminus \hat{y}$:
   - Update misclassified boundary words (type-A updates):
     if $(\mathbf{w}_\mathrm{S}^k \cdot \phi_\mathrm{w}(x_s) \leqslant 0)$ then $\mathbf{w}_\mathrm{S}^k := \mathbf{w}_\mathrm{S}^k + \phi_\mathrm{w}(x_s)$
     if $(\mathbf{w}_\mathrm{E}^k \cdot \phi_\mathrm{w}(x_e) \leqslant 0)$ then $\mathbf{w}_\mathrm{E}^k := \mathbf{w}_\mathrm{E}^k + \phi_\mathrm{w}(x_e)$
   - Update score function, if applied:
     if $(\mathbf{w}_\mathrm{S}^k \cdot \phi_\mathrm{w}(x_s) > 0 \;\wedge \mathbf{w}_\mathrm{E}^k \cdot \phi_\mathrm{w}(x_e) > 0)$ then $\mathbf{w}^k := \mathbf{w}^k + \phi_\mathrm{p}((s,e)_k)$
3. Over-predicted phrases: $\forall (s,e)_k \in \; \hat{y} \setminus y^*$:
   - Update score function: $\mathbf{w}^k := \mathbf{w}^k - \phi_\mathrm{p}((s,e)_k)$
   - Update words misclassified as S or E (type-B updates):
     if (goldS$(x_s, k) = -1$) then $\mathbf{w}_\mathrm{S}^k := \mathbf{w}_\mathrm{S}^k - \phi_\mathrm{w}(x_s)$
     if (goldE$(x_e, k) = -1$) then $\mathbf{w}_\mathrm{E}^k := \mathbf{w}_\mathrm{E}^k - \phi_\mathrm{w}(x_e)$

Note 1. All updates are performed at the same time, that is, the conditions of the feedback rule are evaluated with the initial set of vectors.
Note 2. The updates of the *start-end* vectors are performed only once per word, although an incorrect prediction on a word may generate several incorrect phrases.

Figure 1. Pseudocode for the FR-Perceptron algorithm

algorithms stands for update rules which modify only the prototypes corresponding to mistakes in the global solution of an example. For instance, in the multiclass setting a certain prototype is demoted only if its prediction score is higher than the score of the true-label's prototype. Also related, in (Har-Peled, Roth, & Zimak, 2002) a generalized *constraint classification* setting is proposed, which allows to model, in such a conservative way, multiclass and multilabel classification problems, and ranking problems which can be expressed with order relation pairs.

Directly related to FR-Perceptron are the global linear models for parsing and tagging by Collins (2002, 2004), which apply to solution spaces of exponential size. In this family of models, a sentence-solution pair is represented in a feature vector, and a weight vector in the same dimensionality produces a prediction score for that pair. Then, the learning problem consists of estimating the weight vector so that the correct solution is ranked the highest. In (Collins, 2002), a perceptron algorithm is presented for tagging problems, working with representations for which there exists a decoding algorithm that, given the weight vector, picks the top-ranked solution for a sentence in polynomial time (using dynamic programming). Basically, the online learning algorithm runs the decoder on a given example and then updates the weight vector conservatively, looking only at the mistakes found in the top-ranked global solution. FR-Perceptron can be seen as an instance of this basic algorithm, and, in fact, the score functions are trained as in (Collins, 2002) (see proof below in subsection 3.3). However, FR-Perceptron extends Collins' algorithm by training also a filtering component which, working at word level (linear with the sentence length), discards solutions and makes the decoding process more efficient in terms of the number of candidates considered at the phrase level (quadratic with the sentence length).

As in all conservative algorithms mentioned, the type of update of FR-Perceptron on each function has a global effect in the sense that interactions of functions are captured and they become dependent. The learning approach is driven so as to optimize the global accuracy of the recognizer, rather than local accuracies of each individual function. Also, by learning online from the output of the structure decoder, the algorithm naturally selects the most informative instances out of the quadratic number of possible phrase candidates. As we show experimentally in section 5, the algorithm effectively models the functions as word filters and phrase rankers, which contributes positively on optimizing the F_1 measure on precision-recall. We also provide empirical evidence in favor of the FR-Perceptron with respect to learning strategies which train the components independently.

3.3. Convergence Proof

We show in this section a convergence result for the FR-Perceptron algorithm. The result we present depends on some restrictive hypotheses and has to be seen only as a first step in the task of achieving a deep and more useful analysis. See the discussion in Section 6.

The convergence proof we give is based on the proof by Novikoff (1962) for the basic perceptron algorithm as much as on the proof presented by Collins (2002) for the perceptron-based sequence tagging algorithm. Assuming *separability* for the Phrase Recognition function and linear separability for each of the *start* and *end* classifiers, the number of errors committed by the learning algorithm can be upper bounded. The proof is presented in two steps, which are described in the two following subsections.

To simplify the notation in the following analysis, we deliberately eliminate the k indexes from the formulae, that is, we assume one single type of phrases. The proof below can be easily extended to the general case with several types of phrases making two straightforward changes. First, the *score* vectors $\mathbf{w}^k$ would be concatenated in a single scoring vector $\mathbf{w} \in \mathbb{R}^{d\times|\mathcal{K}|}$. Second, we should assume linear separability for each of the *start-end* classification functions associated to k-phrases.

3.3.1. *Scoring Layer without Filtering*

In this setting, we assume no filtering components in the learning architecture and we require, for each example (x, y), a search space $\hat{\mathcal{Y}}(x) \subseteq \mathcal{Y}$ containing the solution y. Thus, the Phrase Recognition function can be written as:

$$\mathrm{R}(x) = \arg\max_{y\in\hat{\mathcal{Y}}(x)} \sum_{p\in y} \mathrm{score}(p, x, y)$$

We can rewrite the R function in terms of a single dot product, by considering a *global* representation function Φ_{p}, which corresponds to the summation of the ϕ_{p} representations of all the phrases in y:

$$\begin{aligned}
\mathrm{R}(x) &= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \sum_{p\in y} \mathrm{score}(p, x, y) \\
&= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \sum_{p\in y} \mathbf{w}\cdot\phi_{\mathrm{p}}(p, x, y) \\
&= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \; \mathbf{w}\cdot\sum_{p\in y} \phi_{\mathrm{p}}(p, x, y) \\
&= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \; \mathbf{w}\cdot\Phi_{\mathrm{p}}(x, y)
\end{aligned}$$

Given this notation and by using the assumption that the solutions are in the search spaces, the updating rule for the scoring layer of the FR-Perceptron algorithm can be rewritten globally as follows:

$$\mathbf{w} = \mathbf{w} + \Phi_{\mathrm{p}}(x, y) - \Phi_{\mathrm{p}}(x, \hat{y})$$

In this form, the FR-Perceptron algorithm (without filtering) is the same than the algorithm analyzed in (Collins, 2002), which proves its convergence. We reproduce here his theorem, in a more general version which allows an initial vector $\mathbf{w}_0$ different from zero. Later we will use this fact.

DEFINITION 1. *A training set $S = \{(x^i, y^i)\}_{i=1}^m$ is separable with margin $\delta > 0$ if there exists some vector $\mathbf{w}^*$ with $||\mathbf{w}^*|| = 1$ such that:*

$$(\forall i : 1 \leqslant i \leqslant m)(\forall z : z \in \hat{\mathcal{Y}}(x^i) \wedge \ z \neq y^i)$$
$$(\mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, y^i) \geqslant \mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, z) + \delta).$$

THEOREM 1. *For any training set $S = \{(x^i, y^i)\}_{i=1}^m$ which is separable with margin $\delta > 0$, the* FR-Perceptron *algorithm with no filtering components makes a number of mistakes bounded by*

$$\frac{R^2}{\delta^2} + \frac{2\|\mathbf{w}_0\|}{\delta},$$

where $\mathbf{w}_0$ refers to the initial value of $\mathbf{w}$ and R is a constant such that $(\forall i : 1 \leqslant i \leqslant m)\,(\forall z : z \in \hat{\mathcal{Y}}(x^i)) \ \ (||\Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, z)|| \leqslant R)$.

Proof. We essentially follow the main steps in the Collins' proof for a slightly weaker version of Theorem 1 that assumes that the initial vector $\mathbf{w}_0$ is the zero vector.

We enumerate the updates made by the FR-Perceptron algorithm by using an index j and starting from $j = 1$. We have to show that j is upper bounded by $R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$.

First, we show a lower bound on the norm of the weight vector at update number j. Let $\mathbf{w}_{j-1}$ be the weight vector just before the j'th mistake is made. Suppose that this mistake is made at example (x^i, y^i) and let $\hat{y}$ be the output proposed at this example. So, $\hat{y} = \arg\max_{y \in \hat{\mathcal{Y}}(x^i)} \mathbf{w}_{j-1} \cdot \Phi_{\mathrm{p}}(x^i, y)$. It follows from the algorithm update rule that $\mathbf{w}_j = \mathbf{w}_{j-1} + \Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, \hat{y})$. We take inner products of both sides with the vector $\mathbf{w}^*$ given by Definition 1:

$$\mathbf{w}^* \cdot \mathbf{w}_j = \mathbf{w}^* \cdot \mathbf{w}_{j-1} + \mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, y^i) - \mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, \hat{y}) \geqslant \mathbf{w}^* \cdot \mathbf{w}_{j-1} + \delta,$$

where the inequality follows because the training set is separable with margin δ. Hence, reasoning by induction, $\mathbf{w}^* \cdot \mathbf{w}_j \geqslant \mathbf{w}^* \cdot \mathbf{w}_0 + j\delta$. Now, by applying twice the Schwarz inequality ($|\mathbf{u} \cdot \mathbf{v}| \leqslant \|\mathbf{u}\|\|\mathbf{v}\|$) and using the fact that $\mathbf{w}^*$ has norm 1 it follows that $\|\mathbf{w}_j\| \geqslant \mathbf{w}^* \cdot \mathbf{w}_0 + j\delta \geqslant j\delta - \|\mathbf{w}_0\|$.

An upper bound for $\|\mathbf{w}_j\|^2$ can be also derived:

$$\begin{aligned}\|\mathbf{w}_j\|^2 &= \|\mathbf{w}_{j-1}\|^2 + \|\Phi_\mathrm{p}(x^i,y^i) - \Phi_\mathrm{p}(x^i,\hat{y})\|^2 \\ &\quad +2\mathbf{w}_{j-1}\cdot(\Phi_\mathrm{p}(x^i,y^i) - \Phi_\mathrm{p}(x^i,\hat{y})) \\ &\leqslant \|\mathbf{w}_{j-1}\|^2 + R^2,\end{aligned}$$

where the last inequality follows because, by assumption, $\|\Phi_\mathrm{p}(x^i,y^i) - \Phi_\mathrm{p}(x^i,\hat{y})\|^2 \leqslant R^2$, and $\mathbf{w}_{j-1}\cdot(\Phi_\mathrm{p}(x^i,y^i) - \Phi_\mathrm{p}(x^i,\hat{y})) \leqslant 0$ because $y^i \in \hat{\mathcal{Y}}(x^i)$ (by the hypothesis in this section) and $\hat{y}$ is the highest scoring candidate for x^i under $\mathbf{w}_{j-1}$ in $\hat{\mathcal{Y}}(x^i)$. Reasoning again by induction we have $\|\mathbf{w}_j\|^2 \leqslant \|\mathbf{w}_0\|^2 + jR^2$.

Finally, note that if $j \geqslant \|\mathbf{w}_0\|/\delta$ (otherwise, the bound we want to show is straightforward) then $j\delta - \|\mathbf{w}_0\|$ is non-negative and we can combine the bounds on the norm of $\mathbf{w}_j$ we have just obtained:

$$(j\delta - \|\mathbf{w}_0\|)^2 \leqslant \|\mathbf{w}_j\|^2 \leqslant \|\mathbf{w}_0\|^2 + jR^2.$$

From this expression it is immediate to see that $j \leqslant R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$. □

3.3.2. *Filtering and Scoring*

When including the filtering component, the recognition model is written as:

$$\mathrm{R}(x) = \arg\max_{y\in\mathcal{Y}_{SE}(x)} \mathbf{w}\cdot\Phi_\mathrm{p}(x,y)\,,$$

where $\mathcal{Y}_{SE}(x) = \{y \in \mathcal{Y}\,|\,(\forall(s,e)\in y)(h_S(x_s,x) > 0 \,\wedge\, h_E(x_e,x) > 0)\}$.

DEFINITION 2. *(Start-End separability) A training set* $\{(x^i,y^i)\}_{i=1}^m$ *is Start-End Separable with margin* $\gamma > 0$ *if there exist vectors* $\mathbf{w}_\mathrm{S}{}^*$ *and* $\mathbf{w}_\mathrm{E}{}^*$ *with* $||\mathbf{w}_\mathrm{S}^*|| = 1$, $||\mathbf{w}_\mathrm{E}^*|| = 1$ *such that:*

$$\begin{aligned}&(\forall i: 1 \leqslant i \leqslant m)(\forall j: 1 \leqslant j \leqslant n_i) \\ &\quad(\mathrm{goldS}(x_j^i)(\mathbf{w}_\mathrm{S}^*\cdot\phi_\mathrm{w}(x_j^i,x)) \geqslant \gamma \,\wedge\, \mathrm{goldE}(x_j^i)(\mathbf{w}_\mathrm{E}^*\cdot\phi_\mathrm{w}(x_j^i,x)) \geqslant \gamma).\end{aligned}$$

In order to show the convergence of the FR-Perceptron algorithm we assume the separability of the sample in the sense of Definition 1, considering the whole solution space, $\hat{\mathcal{Y}} = \mathcal{Y}$. In addition, we assume also the Start-End separability of Definition 2. We will denote by SE-LF a learning feedback stage of FR-Perceptron where updates affect to the *start-end* vectors $\{\mathbf{w}_\mathrm{S}, \mathbf{w}_\mathrm{E}\}$ and maybe the *score* vector $\mathbf{w}$, and by SCORE-LF those stages where updates only changes the *score* vector. Our goal is to bound the number of SE-LF and SCORE-LF stages, and, therefore, the number of training errors. We start by bounding the number of SE-LF stages that the algorithm FR-Perceptron makes when running on a Start-End separable sample.

LEMMA 1. *Let* $R_{SE} = \max_{1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n_i} \|\phi_{\mathrm{w}}(x_j^i, x^i)\|$ *of a training set* $\{(x^i, y^i)\}_{i=1}^m$ *satisfying Definition 2 with margin* γ. *Then the number of* SE-LF *stages of* FR-Perceptron *is at most*

$$\frac{2R_{SE}^2}{\gamma^2}.$$

Proof. It follows immediately from Novikoff's proof (Novikoff, 1962) for the standard perceptron algorithm. □

Now, to conclude the convergence of the algorithm FR-Perceptron we must demonstrate that after a SE-LF stage there is room only for a finite number of consecutive SCORE-LF stages before a point where, no more changes of any kind are necessary or a new SE-LF stage is required. We consider two cases. First, we assume that in the last SE-LF stage the updated values of $\mathbf{w}_{\mathrm{S}}$ and $\mathbf{w}_{\mathrm{E}}$ are so that for some $1 \leqslant i \leqslant m$ the solution y^i does not belong to $\mathcal{Y}_{SE}(x^i)$. In this case, it is clear that if $\{\mathbf{w}_{\mathrm{S}}, \mathbf{w}_{\mathrm{E}}\}$ have not been yet modified when FR-Perceptron visits the example (x^i, y^i), a change on $\{\mathbf{w}_{\mathrm{S}}, \mathbf{w}_{\mathrm{E}}\}$ of type A (see Figure 1) will be made when the algorithm visits such example. So, we have the following lemma.

LEMMA 2. *Assume that for some* i, $1 \leqslant i \leqslant m$, *the search space* $\mathcal{Y}_{SE}(x^i)$ *does not include the solution* y^i. *At this point at most* $m-1$ *consecutive* SCORE-LF *stages are possible.*

Second, we consider the case where the last SE-LF stage has updated values for $\{\mathbf{w}_{\mathrm{S}}, \mathbf{w}_{\mathrm{E}}\}$ such that for all i, $1 \leqslant i \leqslant m$ it is $y^i \in \mathcal{Y}_{SE}(x^i)$. Let $\mathbf{w}_0$ be the value of $\mathbf{w}$ when this SE-LF stage is completed. Here we are in the hypothesis of Theorem 1 and therefore, after at most $R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$ consecutive SCORE-LF stages the algorithm reaches to a point where no more learning feedbacks are necessary or a new SE-LF stage appears (generated by a type-B change, see Figure 1).

LEMMA 3. *Assume that for all* i, $1 \leqslant i \leqslant m$, *the search space* $\mathcal{Y}_{SE}(x^i)$ *contains the solution* y^i. *At this point at most* $R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$ *consecutive* SCORE-LF *stages are possible.*

Summarizing, from Lemmas 1, 2 and 3 we conclude the next theorem that shows the convergence of FR-Perceptron.

THEOREM 2. *For any training set* $S = \{(x^i, y^i)\}_{i=1}^m$ *separable with margin* $\delta > 0$ *and Start-End separable with margin* $\gamma > 0$, *it holds:*

1. *The number of learning feedbacks that affect to the start-end vectors* (SE-LF *stages) is bounded by* $2R_{SE}^2/\gamma^2$.

2. *After a learning feedback stage l that has affected the start-end vectors* (SE-LF *stage*) *there are at most*

$$\max\left(m-1, \frac{R^2}{\delta^2}+\frac{2\|\mathbf{w}_0^l\|}{\delta}\right)$$

consecutive learning feedbacks that only affects to the score *vectors* (SCORE-LF *stages*). *Here,* $\mathbf{w}_0^l$ *is the vector* $\mathbf{w}$ *when the updates corresponding to stage l have been just made.*

3. *As a consequence of the previous two points, the* FR-Perceptron *algorithm makes a finite number of errors on the training set. When no more errors occur the total number of errors committed by the algorithm is bounded by*

$$\frac{2R_{SE}^2}{\gamma^2}\left(1+\max\left(m-1, \frac{R^2}{\delta^2}+\frac{2MAX}{\delta}\right)\right),$$

where MAX is the maximum of the values $\|\mathbf{w}_0^l\|$ at any SE-LF *stage.*

3.4. Voted Perceptron, Dual Formulation and Kernels

Although the analysis above concerns the perceptron algorithm, in the experimental part we use a modified version, the *voted perceptron* algorithm, introduced by Freund and Schapire (1999). The key point of the voted version is that, while training, it stores information in order to make more robust predictions on test data. Specifically, all the prediction vectors $\mathbf{w}^j$ generated after every mistake are stored, together with a weight c^j that is set during training, while the vector survives until the next mistake. In particular, when a vector is generated its weight is set to one. Then, for each phrase correctly identified each of the three responsible vectors (namely *start*, *end* and *score*) is incremented by one its weight. Let J be the total number of vectors that a perceptron accumulates. The final hypothesis is an averaged vote over the predictions of each vector, computed with the expression:

$$h_{\mathbf{w}}(\mathbf{x})=\sum_{j=1}^{J} c^j\,(\mathbf{w}^j\cdot\mathbf{x})$$

This expression corresponds to the *averaged* prediction method. The authors also propose a *voted* prediction method which takes the sign of each $\mathbf{w}^j\cdot\mathbf{x}$, instead of the real value. In the experimental section we show that both methods substantially outperform the unweighted prediction expression of the perceptron (denoted *last*, since it only makes use of the last $\mathbf{w}^J$ vector).

Moreover, we work with the dual formulation of the model, which allows the use of kernel functions, and thus learning non-linear separators. It is also shown in (Freund & Schapire, 1999) that a vector $\mathbf{w}$ can be expressed as the sum of instances $\mathbf{x}^j$ that were added ($s^j = +1$) or subtracted ($s^j = -1$) in order to create it, as $\mathbf{w} = \sum_{j=1}^{J} s^j \mathbf{x}^j$. Given a kernel function $K(x, x')$, the final expression of a dual voted perceptron becomes:

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=1}^{J} c^j \sum_{l=1}^{j} s^l K(\mathbf{x}^l, \mathbf{x})$$

Taking advantage of the recursion in the expression, computing a prediction requires J kernel computations. In this paper we work only with polynomial kernels $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$, where d is the *degree* of the kernel.

4. The Problems: Chunking and Clause Identification

Partial Parsing belongs to the area of syntactic analysis of Natural Language, and deals with tasks in which the goal is to detect partial syntactic structure of text. Particularly, in *Base Chunking*, also known as *Shallow Parsing*, the task consists of identifying a sequence of non-recursive phrases, taking into account the syntactic categories. Each phrase, or *chunk*, groups related words according to their syntactic function. Types of chunks include, among others, noun phrases (NP), verb phrases (VP) and prepositional phrases (PP). In the example below, the chunks are marked between brackets together with their type:

(The space shuttle Atlantis)$_{NP}$ (blasted)$_{VP}$ (into)$_{PP}$ (orbit)$_{NP}$ (from)$_{PP}$ (Cape Canaveral)$_{NP}$ and (its crew)$_{NP}$ (launched)$_{VP}$ (the Galileo space probe)$_{NP}$ (on)$_{PP}$ (a flight)$_{NP}$ (to)$_{PP}$ (the planet Jupiter)$_{NP}$.

A higher level of partial parsing is the problem of *Clause Identification*, in which the syntactic clauses of the sentence have to be recognized. Clauses can be roughly defined as sequences of words with a verb and a subject, possibly implicit. Clauses form a hierarchical structure which constitutes the skeleton of the full syntactic tree of a sentence. In the example below, clauses are enclosed within brackets:

Table I. Information on the datasets

Data Set	WSJ sections	#sentences	#words	#phrases
Chunking				
Train	15–18	8,936	211,727	106,978
Devel.	21	1,671	40,039	20,159
Test	20	2,012	47,377	23,852
Clause Identification				
Train	15–18	8,936	211,727	24,841
Devel.	20	2,012	47,377	5,418
Test	21	1,671	40,039	4,856

```
( ( The space shuttle Atlantis blasted into orbit from
Cape Canaveral ) and ( its crew launched the Galileo
space probe on a flight to the planet Jupiter ) . )
```

The CoNLL conference (Computational Natural Language Learning), through the organization of shared tasks, provides benchmarks on Natural Language problems in which many learning-based systems can be compared. The 2000 and 2001 editions dealt respectively with the tasks of Chunking (Tjong Kim Sang & Buchholz, 2000) and Clause Identification (Tjong Kim Sang & Déjean, 2001), and provided data. [1] In this work, we followed the CoNLL settings for both problems. The Chunking problem consists of recognizing the set of chunks of a sentence on the basis of words and part-of-speech (PoS) tags. There are eleven different types of chunks. The data consists of a training set and a test set extracted from the WSJ sections of the Penn Treebank. We also constructed a development set to perform tuning of our system. [2]

The Clause Identification problem consists of recognizing the set of clauses on the basis of words, PoS tags, and chunks. The data contains a training set, a development set and a test set extracted also from the WSJ corpus. In the CoNLL-2001 setting the problem was simplified by removing the clause–type labels. As a consequence, there is only one category of phrases to be considered.

Table I summarizes details of both data sets.

[1] Data sets and more details can be found at the CoNLL-2000 and CoNLL-2001 websites: `http://cnts.uia.ac.be/conllYYYY`, where `YYYY` is 2000 or 2001 .

[2] The development set for Chunking corresponds to the WSJ section 21. The chunks have been extracted with the `chunklink.pl` script available at the CoNLL-2000 website. The PoS tags have been extracted from the test set of the CoNLL-2001 task on Clause Identification.

4.1. Representation Functions

In this section we describe the representation functions ϕ_{w} and ϕ_{p} used in both partial parsing problems, which respectively map a word or a phrase and their local contexts into a feature vector in $\mathbb{R}^d$.

Basically, the functions represent the instances by capturing a number of relevant features for the task. Such features define a vector space, usually huge, in which each feature corresponds to one dimension, being in most of the cases binary-valued. Essentially, we reproduce similar feature spaces than other relevant works for these tasks, which reported state-of-the-art results. For Chunking, we consider (Kudo & Matsumoto, 2001; Collins, 2002; Sha & Pereira, 2003), and for Clause Identification (Carreras et al., 2002).

First, we define a set of primitive functions which apply to words or sequences of words, and will be used to define the representations:

- **Word**(x_i): The form of word x_i.
- **PoS**(x_i): The part-of-speech tag of word x_i.
- **ChunkTag**(x_i): The chunk tag of word x_i.
- P-**Gram**($[x_s, \ldots, x_e]$): The conjunction of the outputs of the primitive P on words $[x_s, \ldots, x_e]$. We work with Word-Grams and PoS-Grams.
- **Counts**($[x_s, \ldots, x_e]$): The number of occurrences of a predefined set of relevant elements which appear in the sequence $[x_s, \ldots, x_e]$. In particular, the relevant elements we consider are: a) relative pronouns (e.g., `that`); b) punctuation marks (`. , ; :`); c) quotes; d) verb phrase chunks; and e) relative phrase chunks.

4.1.1. *Representing Words*

For the function $\phi_{\mathrm{w}}(x_i, x)$, we compute some primitive functions in a *window* of words around x_i, that is, words x_{i+l} with $l \in [-L_{\mathrm{w}}, +L_{\mathrm{w}}]$. Each primitive label, together with each relative position l and each returned value forms a final binary indicator feature. For example, a binary feature may be "the word at relative position -1 is `that`". Specifically, for Chunking we compute:

- Word and PoS primitives.
- Word-Grams on all possible sequences within the window which include the central word.
- PoS-Grams on all possible sequences within the window which include the central word.

And for Clause Identification we compute:

- Word and PoS primitives.

- ChunkTag primitives.
- Left Start-End predictions: binary flags indicating whether the words in $[-L_{\mathrm{w}}, -1]$ have been predicted as *start* and/or *end* words of a clause.
- Separate counts in the $[x_1, \ldots, x_{i-1}]$ and $[x_i, \ldots, x_n]$ fragments.

4.1.2. *Representing Phrases*

For the function $\phi_{\mathrm{p}}((s,e)_k, x, y)$, we capture features of the context of the phrase and of the phrase itself. In both problems, for the context we evaluate a $[-L_{\mathrm{p}}, 0]$ window of primitives at the s-th word and a separate $[0, +L_{\mathrm{p}}]$ window at the e-th word. These windows include Words and PoS primitives. In Chunking, we also evaluate features representing the recognized chunks in y to the left of the s-th word. We consider up to 3 chunks, and codify the relative position of each one.

As for the (s,e) phrase itself, we evaluate primitives within the fragment $[x_s, \ldots, x_e]$ without capturing the relative position. Specifically, in Chunking we capture:

- The length of the phrase (real-valued feature).
- Word and PoS primitives.
- PoS-Grams on all subsequences of $[x_s, \ldots, x_e]$ of up to size 3, and also the complete PoS-Gram on the whole sequence.

For Clause Identification, we consider:

- Internal Pattern: Concatenation of the relevant elements in the sentence fragment $[x_s, \ldots, x_e]$. The following elements are considered: a) Punctuation marks; b) Coordinate conjunctions; c) The word "`that`"; d) Relative pronouns; e) Verb phrase chunks; and f) The top clauses within the $[x_s, \ldots, x_e]$ fragment, available in y. In the pattern, a clause reduces all the elements it contains, resulting in an atomic element which hides the structure inside it.
- Counts in $[x_s, \ldots, x_e]$. Here we count also the number of internal clauses within (s,e), and the relevant elements within such clauses do not count for these features.

Experimenting on the respective development sets, for Chunking both L_{w} and L_{p} were set to 2, whereas for Clause Identification were set to 3.

5. Experimental Evaluation

In this section we provide extensive experimentation of the proposed learning method on the two presented partial parsing problems, namely Clause Identification and base Chunking . Due to the hierarchical nature of Clause Identification, more attention will be payed to this problem (subsection 5.1). However, the results on syntactic Chunking (subsection 5.2) allows us to generalize the good behavior of the FR-Perceptron algorithm for cases in which there are many types of phrases to learn.

5.1. On Clause Identification

In this part, we are interested in testing the effectiveness of the learning strategy for the recursive phrase recognition model.

We started by training a Clause Identification model with the FR-Perceptron. Since in this problem there is only one type of phrases to be recognized, namely the clauses, the model is composed by three functions: the *start* filter, the *end* filter and the *score* ranker.

To avoid some computation in the *Start-End* layer, we did not consider clause boundary words which would overlap with the chunks provided in the input. Regarding features extracted from phrases during training, and in particular the function $\phi_{\mathrm{p}}(p, x, y)$, while a batch environment permits only to extract features from the gold solution y^*, the online setting allows to work with the predicted solution $\hat{y}$. In preliminary experimentation, we found slightly better results when working with features extracted from the predicted structure. All results presented in this experimentation were obtained using polynomial kernels of degree two. Initial tests revealed a very poor performance for the linear case and no significant improvements for degrees greater than two. The development of specific kernels for exploiting the data structures present in the problems addressed is a question that deserves further attention but, by now, is out of the scope of this work.

We trained the model for up to 50 epochs on the training data, and evaluated it on the development set with the three different prediction methods sketched in section 3.4: *last*, *averaged* and *voted*. Figure 2 shows the performance of the model (F_1 measure) on the training data (left plot) and on the development data (right plot). Clearly, the learning curve on the training set shows that the learning strategy effectively optimizes the global F_1 measure on the task through the learning epochs. However, the performance gets stable around 96, indicating that the training set is not separable under the current choice of features and kernel. Looking at the performance on the development set, the

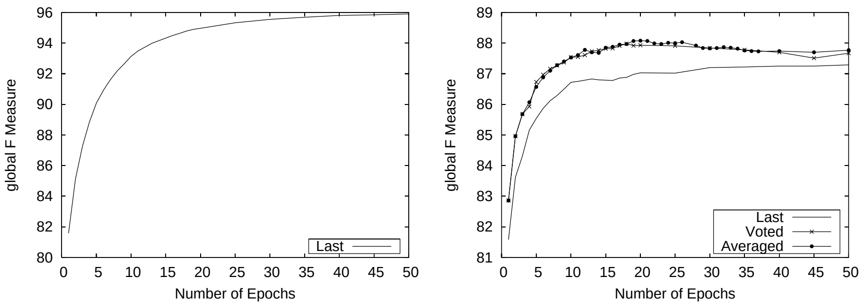

Figure 2. Learning curve on Clause Identification. Left plot: training set in *last* prediction mode. Right plot: development set in *last*, *averaged* and *voted* modes

model shows a good generalization curve, with the best results over 88% and with no significant overfitting. Although at epoch 50 the FR-Perceptron has not converged, the generalization performance seems to be stable from epoch 20, showing only minor decreases in further epochs. Looking at the three prediction methods, both the *averaged* and the *voted* methods perform substantially better than the default *last* method. We chose the *averaged* version as our running prediction method for the remaining experiments, as it achieves slightly better results.

All the extra experiments we performed in order to better understand the behavior of the FR-Perceptron algorithm are presented in the following subsections.

5.1.1. *Recognition Feedback vs. Classification Feedback*

In the FR-Perceptron setting, all the functions are trained together online via recognition feedback. In this experiment we compare the FR-Perceptron to alternative learning settings in which training is based on a binary classification penalty. That is, each function is understood as a binary classifier: the *start-end* functions decide whether a word is or not a boundary word, and the *score* function decides if a phrase belongs to a solution or not. In this context, each function is modeled at the local level, so each prediction the function has made is subject to correction. In contrast, the FR-Perceptron operates at the global level in a conservative way, meaning that only the predictions which have effect on the global solutions are subject to correction. In the classification context, we consider two settings for generating training samples: batch and online. The details are as follows:

– *Batch learning with classification feedback.* This corresponds to the usual approach of training each function separately making use of a batch learning algorithm for binary classification. We selected two algorithms which work in the same hypothesis space (kernel-based linear functions): the batch version of the Voted Perceptron (CB-VP), and "soft-margin" Support Vector Machines (SVM)[3]. For training, we generated three data sets from training sentences, one for each function. For the *start-end* sets, we considered an example for each word in the data, except those breaking chunks. To train the *score* classifier, it is not feasible to generate one example for each possible phrase candidate in the data, since this generation would produce 1,377,843 examples with a 98.2% proportion of negatives. A first direct approach is to generate only phrase candidates formed with all pairs of correct phrase boundaries, which greatly reduces the number of negative examples. However, the resulting *score* classifiers are trained in a context in which perfect identification of phrase boundaries is assumed, which is not the real situation when running on the top of learned *start-end* functions. Table II shows the overall performance of the system running with CB-VP (averaging predictions) and SVM classifiers trained batch. As it could be expected, SVM perform better than CB-VP. But, clearly, the performance is much lower than with FR-Perceptron, suggesting that the *score* functions in this setting are not robust enough. To overcome this limitation, we generated new training samples for the *score* function, now considering the actual behavior of the learned *start-end* functions. The approach is similar to the one in (Carreras et al., 2002): first the learned *start-end* functions are applied to the words in the training data; then, phrase candidates are generated considering pairs of boundary words whose prediction is above a certain threshold θ. Besides, the positive phrases are always generated. Thus, this procedure only adds the negative phrase candidates which pass the boundary threshold. We performed several trials only with SVM, first selecting θ and then the C regularization parameter of the SVM. This threshold was only used for generation of training examples, when testing the filters were used as usual. Table II summarizes the results. Note that, by considering predicted boundaries in training, the precision of the system easily improves 4 points. However, as the amount of negative instances substantially increases, the recall goes down. The best performance for batch models was found using a threshold of -1.0, with SVM. We will use this model to compare against other learning settings.

[3] We used the SVM^{light} package available at `http://svmlight.joachims.org`.

Table II. Performance on the Clause Identification development set when training the functions of the model separately as classifiers. In the first two models, the examples for the *score* function are generated using the gold *start-end* words (goldSE). For the models below, phrase examples are generated with the learned *start-end* functions, taking boundaries with a prediction higher than a threshold θ. In any case, the number of positive training examples is 24,841, whereas the number of negative examples is shown in the third column.

Algorithm	Generation	#Neg.	Precision	Recall	F_1
CB-VP	goldSE	26,374	83.84	80.55	82.16
SVM	goldSE	26,374	84.31	82.83	83.57
SVM	$\theta = 0$	28,165	88.14	82.85	85.41
SVM	$\theta = -0.3$	28,747	88.34	82.76	85.46
SVM	$\theta = -0.5$	29,145	88.46	82.61	85.43
SVM	$\theta = -0.7$	29,610	88.54	82.48	85.41
SVM	$\theta = -0.9$	30,432	88.91	82.58	85.63
SVM	$\theta = -1.0$	59,498	91.12	81.27	**85.91**
SVM	$\theta = -1.1$	97,101	91.49	80.80	85.82
SVM	$\theta = -1.2$	120,856	91.33	80.51	85.58
SVM	$\theta = -1.5$	240,463	92.31	78.01	84.56

– *Online learning with classification feedback (CO-VP).* In this setting all functions are trained together online with the Voted Perceptron algorithm, visiting one sentence at a time, and providing binary classification feedback to each function for each prediction. Given a sentence, the *start-end* functions are first applied to each word and, according to their positive decisions, phrase examples are generated for the *score* function. In this way, the input of the *score* function is dynamically adapted to the *start-end* behavior. Thus, this learning setting is similar to the FR-Perceptron in that functions are learned together and the input to the *score* function is naturally ruled, but differs from it in that the update is done on each local mistake, whereas in the FR-Perceptron the update depends only on the global prediction after the inference. We trained the online classification model for up to 25 epochs, and tested it with averaged predictions.

Figure 3 shows the learning curve in terms of the F_1 measure of the two online models, together with a straight line corresponding to the performance of the best SVM batch model. Clearly, the performance of the FR-Perceptron is better than those of the classification–based

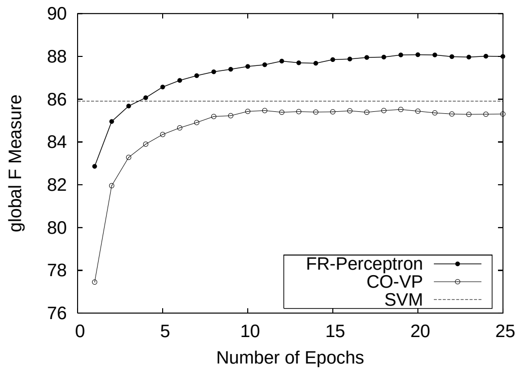

Figure 3. Performance on the Clause Identification development set for different learning settings: the FR-Perceptron model, and classification-based models trained with VP in the online setting (CO-VP) and with SVM in the batch setting.

models. At any epoch, the curve is more than 2 points higher than the one of the online classification-based model (CO-VP). This fact gives empirical evidence in favor of the recognition–based feedback for learning in phrase recognition problems. Below, we provide more results showing why the recognition feedback guides the learning strategy much better than the classification feedback.

Looking at classification-based models, SVM performs slightly better than CO-VP. However, the SVM model has been obtained after tuning the generation of negative phrases, taking into account the learned *start-end* functions. It is worth to recall that when training straightforwardly the *score* function with correct boundaries (see Table II), the SVM model performs more than two points worse than the CO-VP model. In contrast, the online model automatically rules the interaction between both layers.

As a summary, table III shows the performance of each learning strategy on the development and the test sets. Parameters have been optimized in the development set. The performance is significantly lower on the test set, which, rather than overfitting of parameter tuning, should be attributed to more difficulty on that portion of the data —a similar drop is exhibited by most systems running on this data (Tjong Kim Sang & Déjean, 2001). It is interesting to see that the better performance of FR-Perceptron comes from the substantially higher recall figures. This fact relates to the behavior of the filtering layer, which will be explored in detail in the next subsections.

As a complementary information, Figure 4 shows the size of each learned function in terms of the number of different vectors which compose its dual form.

Table III. Results on the Clause Identification development and test sets, for different learning settings: the FR-Perceptron model and the classification-based models trained batch with SVM and online with VP (CO-VP). The number of epochs used for testing (T) has been optimized on the development set, as well as parameters of the SVM.

		Development			Test		
Model	T	Prec.	Recall	F_1	Prec.	Recall	F_1
SVM	-	**91.12**	81.27	85.91	89.18	77.92	83.17
CO-VP	19	91.06	80.62	85.52	**89.25**	77.62	83.03
FR-Perceptron	20	90.56	**85.73**	**88.08**	88.17	**82.10**	**85.03**

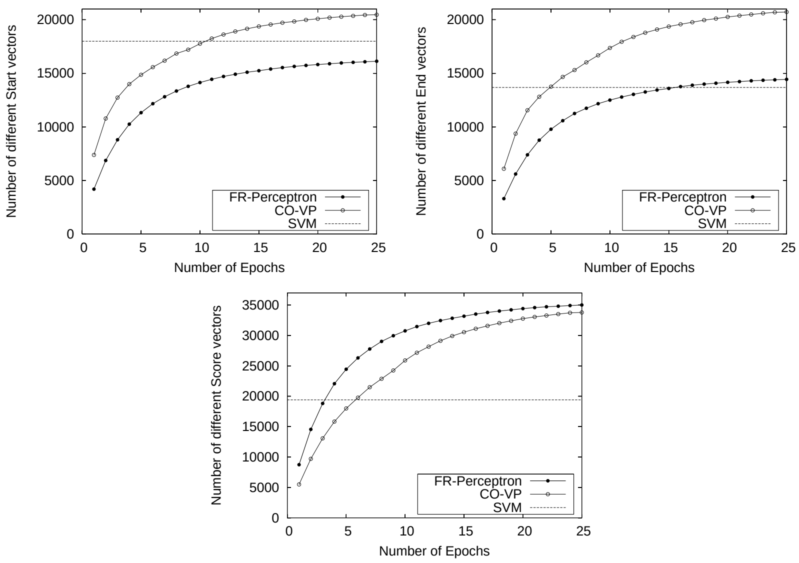

Figure 4. Number of different vectors accumulated in the *start* (top left), *end* (top right), and *score* (bottom) functions for each learning model.

5.1.2. *Behavior of the Start-End Functions*

To get an idea of how the learning strategy of FR-Perceptron works, it is interesting to look at the evolution of the performance of the *Start-End* filtering layer. Figure 5 plots the precision/recall curves of the *start-end* decisions, for each learning strategy considered. On both decisions, the FR-Perceptron starts with high levels of recall and low levels of precision and, along the learning epochs, substantially improves the precision with minor decreases in recall. In contrast, the classification models depart from a high precision and low recall values, and throughout the learning process increase the recall while maintaining

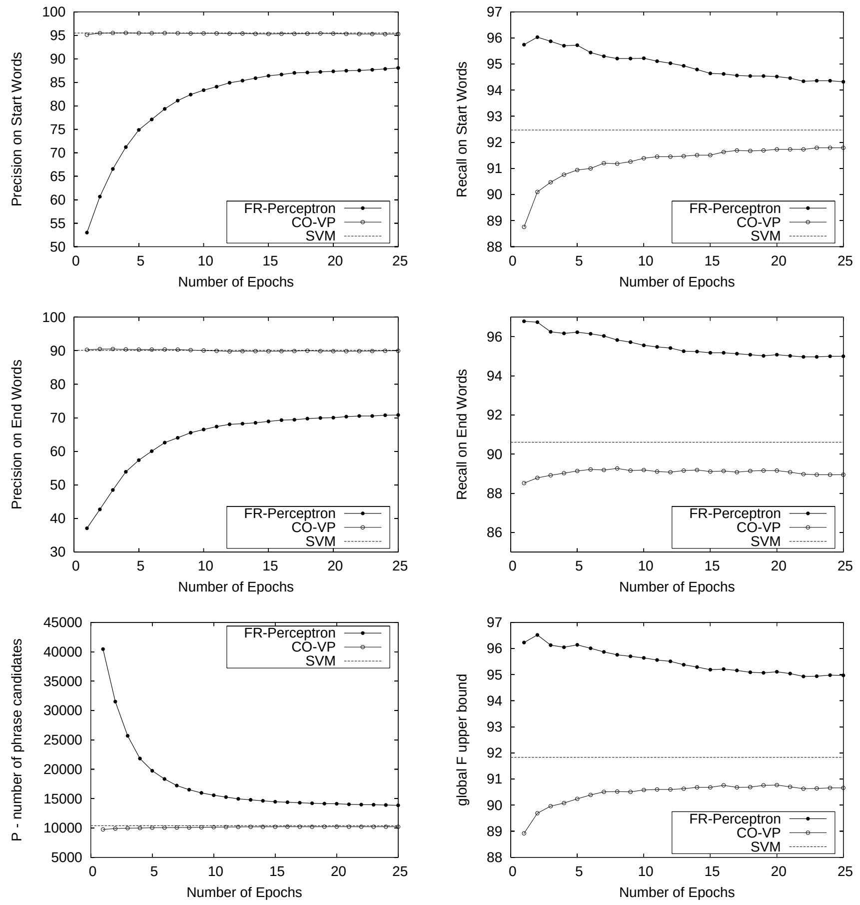

Figure 5. Start-End behavior, for the three learning strategies, on the Clause Identification development set. The first two rows show precision/recall curves on the identification of Start and End words, respectively. The third row shows the behavior of the *Start-End* filtering layer from a global point of view: The left-hand side plots the size of the set of candidate phrases (inversely related to *start/end* precision); The right-hand side plots the maximum achievable global F_1 measure, assuming perfect *score* functions (related to *start/end* recall).

the precision. As it has been argued, the optimal *start-end* functions should behave as filters which do not block any phrase of the solution. Thus, they should maintain very high recall values on phrase boundary words, and try to maximize the precision. The plots evince that this behavior is automatically obtained via the global recognition feedback.

The penalty on the global F_1 measure caused by errors in the *Start-End* layer is also shown in Figure 5 (bottom row, right plot). The curve shows, for each epoch, the maximum achievable global F_1 measure given the phrases proposed by the *Start-End* layer, that is, it is assumed a perfect *score* function given the learned *start-end* functions. Additionally, the filtering precision of the *Start-End* layer is also shown in Figure 5 (bottom row, left plot), in terms of the number of phrase candidates it produces. For the latter measure, the total number of possible phrase candidates in the development set is 300,511. The FR-Perceptron exhibits the expected behavior for a filter: while it maintains a high recall on identifying correct phrases (above 95%), it substantially reduces the number of phrase candidates throughout the learning process, and, thus, the search space for the scoring layer. As a consequence, the input space of the *score* functions is progressively simplified. Also, since the number of explored phrase instances is reduced at each epoch, the recognition process becomes more efficient. Unfortunately, the size of each function in terms of the number of vectors combined also grows (recall Figure 4) and, in practice, the overall model is each time slower.

Far from the behavior of the FR-Perceptron, the models trained via classification feedback do not adapt the filter behavior to maximize the global performance and, although they aggressively reduce the search space, provide only a moderate upper bound on the global F_1.

5.1.3. *Behavior of the Score Function*

A complementary question on this task concerns the learnability of the *score* functions. In this experiment we trained *score* functions in special settings of the filtering layer.

In the first setting, we assumed a perfect classification of starting and ending words. In this situation, we trained a *score* function via recognition feedback, and another via classification feedback, visiting in both cases the training sentences online. The top-left plot of Figure 6 shows the learning curves on the development set. The plot also presents the performance of the SVM scorer on the top of perfect filters, which has been trained on the same space. The three approaches obtain a similar performance when they are stable, being the recognition-based scorer slightly better. It is also noticeable that the recognition feedback achieves high performance levels much faster than the classification feedback. In all cases, the performance is very high (F_1 over 97.5), indicating that the score function can be effectively learned and that the filtering component is the real bottleneck of the system.

In the second setting, we considered a model with no filtering layer. Therefore, it explored all possible phrases in a sentence (except those breaking chunks). This fact makes the model computationally very expensive, since at each training epoch it has to visit 1,377,843 phrase candidates. In contrast, the filtering component in the regular model produces a filtered set ranging from 140,000 instances at the first epoch

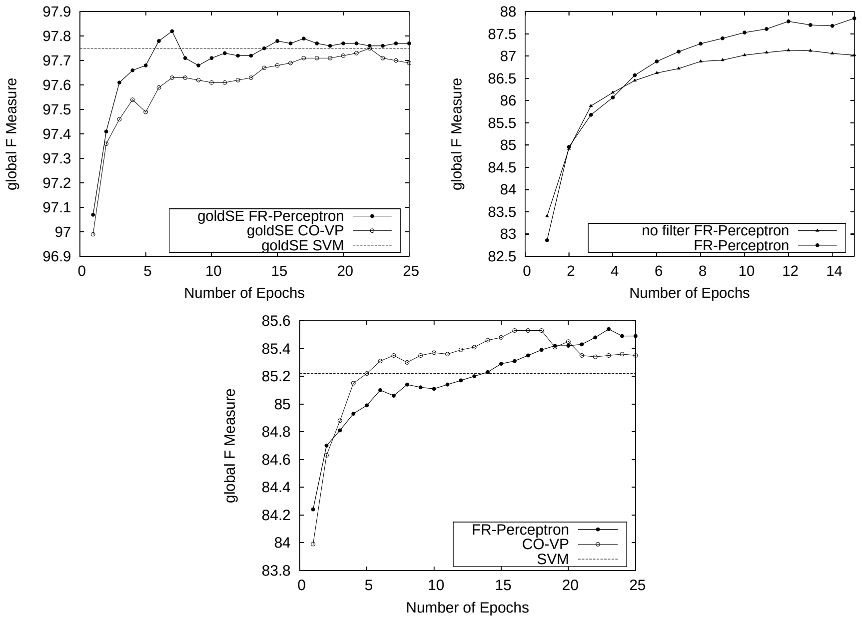

Figure 6. Performance on the Clause Identification development set for special settings of the filters. Top-left plot: using perfect *start-end* filters, showing the performance obtained by training the *score* function as a classifier or as a ranker (classification vs. recognition feedback). Top-right plot: effect of using a *start-end* filter or not. Bottom plot: using the learned *start-end* filters of the FR-Perceptron, and learning new *score* functions from scratch.

to 65,000 instances when stable (see also bottom left plot of Figure 5 for the reduction in the development set). The top-right plot of Figure 6 shows the performance of this model, together with that of the regular FR-Perceptron. The results in terms of global F_1 measure are very competitive, outperforming the results of the classification-based models of the previous experiment. This fact indicates that the *score* function can be straightforwardly learned, at the cost of working in a computationally very expensive search space. However, the combination of filtering functions with the ranker still provides the best results, and makes the overall model computationally feasible.

Finally, we considered the *start-end* filters learned with the FR-Perceptron in the experiment described in the previous section. These filters define phrase candidates both in the training data and the test data. With the training candidates, we trained *score* functions from scratch, using the three running learning settings: (a) online via recognition feedback, corresponding to the FR-Perceptron with fixed *start-end* functions, (b) online with VP, giving binary classification feedback to

Table IV. Comparison on Clause Identification, with the top-performing systems published so-far on the test set.

Reference	Technique	Precision	Recall	F_1
FR-Perceptron	F&R VP	88.17	82.10	85.03
(Carreras et al., 2002)	AdaBoost class.	90.18	78.11	83.71
(Carreras & Màrquez, 2001)	AdaBoost class.	84.82	78.85	81.73
(Molina & Pla, 2001)	HMM	70.85	70.51	70.68

each prediction; and, (c) batch, with the SVM binary classification algorithm. Note that in this setting, where *start-end* functions are fixed, the only advantage of training a classification model in the online setting is that the current prediction can be used to extract structural features, which turns out to provide better results. The bottom plot of Figure 6 depicts the learning curves for these models. The online models reach similar performances in this case: the classification-based model is faster at learning in terms of epochs, but the recogniton-based model end with slightly better figures. The SVM batch model behaves slightly worse. Note also that the FR-Perceptron achieves a performance more than 2 points lower than in the regular model which trains both components together. This result suggests that capturing interactions between the two layers while learning leads to a better generalization performance.

5.1.4. *Comparing to Related Works*

Table IV gives results of the top-performing published methods in this problem. Our new approach clearly outperforms a previous result (Carreras et al., 2002), which consisted of a robust combination of AdaBoost classifiers working in the same phrase recognition model. The scoring classifiers in that system were trained taking into account the type of errors produced in the filtering layer, as we did with the SVM models. The introduced online learning strategy achieves better results automatically. The other two works on the table correspond to the best systems at competition time.

5.2. On Syntactic Chunking

In the Chunking problem we were specially interested in testing the effectiveness of the FR-Perceptron when training filters and rankers for many types of phrases. Also, in this problem phrases can not be embedded. We trained a model, FRP-Chunker, for the 11 types of chunks in the data, with tree functions per chunk type.

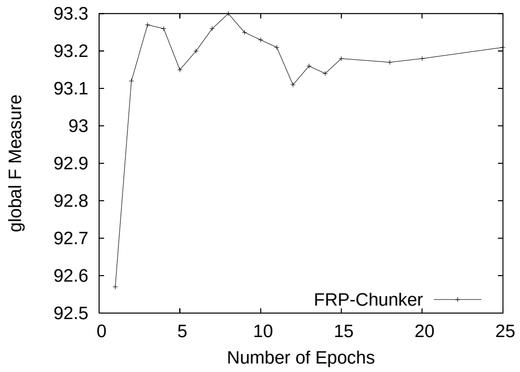

Figure 7. Results on the Chunking development set for the eleven type FRP-Chunker.

Figure 7 plots the evolution of performance (F_1 measure) on the development set for the FRP-Chunker. The results become stable in few epochs at very competitive performance figures. The best performance in terms of F_1 was found at 4 learning epochs. Table V (first 3 columns) presents the results, with figures detailed per chunk type. The overall results obtained are very competitive with respect to the state–of–the–art systems for the task (see below the comparative subsection 5.2.2).

5.2.1. *Training and Combining many Mono-category FRP-Chunkers*
In the experimentation presented so-far we have seen the advantages of training globally the filters and rankers of a phrase recognizer. A complementary question to experiment with when dealing with many types of phrases is whether training globally a multi-category chunker is advantageous over training several mono-category chunkers and then combining them. In this experiment we trained eleven independent FRP-Chunkers, one for each chunk type in the data. Each chunker, thus, is formed by two *start-end* filters and a *score* function for the associated chunk type. For each chunker, we trained up to 15 epochs and selected its best performing epoch on the development set. Then, we put together straightforwardly the eleven mono-category chunkers to form a single multi-category chunker.

The results for this model are shown in the last two blocks of Table V. The second block provides the results of each individual chunker applied separately to the test. The last block shows the results obtained by the combined chunker. As it can be seen, the results of the combined chunker tend to be marginally lower than the individual performances. Note that the prediction scores in both cases are the same. Thus, the difference in performance is produced by the inference when re-

Table V. Performance in Chunking per phrase types, on the test set. The first triple of columns (Chunker) corresponds to the eleven type FRP-Chunker working with 4 epochs. The second triple of columns presents the results obtained by individual Mono FRP-Chunkers, each on their specific type. The third triple of columns corresponds to the combination of the 11 Mono FRP-Chunkers.

	Chunker			Mono-Chunkers			Combined Mono-C.		
	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1
overall	94.17	93.26	93.71	-	-	-	93.39	92.60	93.00
ADJP	83.24	68.04	74.87	84.05	67.35	74.78	85.55	66.21	74.65
ADVP	83.72	78.41	80.98	86.33	78.06	81.99	86.73	77.71	81.97
CONJP	50.00	44.44	47.06	50.00	44.44	47.06	57.14	44.44	50.00
INTJ	100.00	50.00	66.67	00.00	00.00	00.00	0.00	0.00	0.00
LST	0.00	0.00	0.00	00.00	00.00	00.00	0.00	0.00	0.00
NP	94.48	94.33	94.41	94.83	94.13	94.48	92.75	93.13	92.94
PP	96.58	97.88	97.22	97.15	97.76	97.45	97.09	97.71	97.40
PRT	78.57	62.26	69.47	74.53	74.53	74.53	83.33	70.75	76.53
SBAR	91.76	79.07	84.94	86.35	82.80	84.54	90.59	80.93	85.49
UCP	0.00	0.00	0.00	00.00	00.00	00.00	0.00	0.00	0.00
VP	94.07	93.32	93.70	94.04	93.88	93.96	93.48	93.26	93.37

solving overlapping constraints. Compared to the trained 11-category FRP-Chunker, the combined model achieves a moderately lower performance. This result seems to indicate that training dependently the chunkers adapts slightly better each prediction function to fit into the inference layer. However, since there is little gain in performance, it might seem more practical to learn a separate recognizer for each phrase category. First, concentrating on a single category makes the training procedure much easier, since the learning effort (in terms of the number of epochs, complexity of the kernel, etc.) can be adapted to that category according to the complexity of its phrases. Second, from an engineering point of view, it is more desirable to have separate chunkers, and to be able to combine them at will depending on the requirements of a particular task.

5.2.2. *Comparison to Related Works*

Table VI provides the results of the top-performing methods published so-far on the test set of the Chunking task. As it can be seen, our system is situated among the top systems for the problem, which achieve all similar performances. The methods with better results than our system perform the task as a tagging, solved with multiclass learning techniques. Kudo and Matsumoto (2001) performed several taggings which were later combined. Each tagging is produced with SVM classifiers performing a pairwise multiclass prediction. They report a performance of 93.85 with an individual tagging and 93.91 by combining many taggings in a majority voting scheme. Their system makes use of several hundreds of SVM classifiers applied to each word, whereas we only need 22 perceptrons for filtering words and 11 perceptrons for scoring phrases. In contrast, their feature space is simpler than ours, since in the score functions we exploit rich features on phrases. The best performing work on the data set is by Zhang et al. (2002), which apply regularized Winnow. They report a base performance of 93.57, and an improved performance of 94.17 by making use of enhanced grammatical information (noted with +), which was unavailable to us. The last three rows of the table correspond to the best systems at competition time (CoNLL-2000). Table VII shows comparative results on recognizing individual noun phrases (NP). Apart from the referenced systems which apply to the complete chunking task, there have been recently many systems specifically trained for this category. It has to be noted that in this table there are systems working with the full set of chunk types, and others working specifically with NP phrases. Thus, training conditions are not exactly the same, since the latter only make use of NP phrase annotations. This remark is noted in the second column of the table. Our specific NP-chunker, achieving 94.48, slightly outperforms all of them, whereas our multi-chunker, at 94.41, obtains competitive performance. Sha and Pereira (2003) obtained 94.38 with Conditional Random Fields. They also report 94.09 for the Markovian tagging architecture globally trained with perceptron of Collins (2002). The latter work reports an F_1 at 93.53 on the NP chunking data originally defined by Ramshaw and Marcus (1995), which is not identical to the CoNLL-2000 data but fairly comparable. On the same data, Punyakanok and Roth (2004) report F_1 at 94.15 with a conditional Markovian model which makes use of probabilistic SNoW classifiers.

6. Conclusions

We have presented a global learning algorithm, FR-Perceptron, for the general problem of recognizing structures of phrases, in which, typically, several different learning functions are required to recognize the structure. The FR-Perceptron algorithm works online getting feedback from a global point of view, and trains all the functions together, so that interactions between functions when performing the task can be captured in the learning process. In particular, our algorithm learns two layers of decision functions: a filtering layer, which reduces the

Table VI. Comparison on the Chunking task, with the top-performing systems published so-far on the test set.

Reference	Technique	Precision	Recall	F_1
(Zhang et al., 2002)	Winnow (+)	94.28	94.07	94.17
(Kudo & Matsumoto, 2001)	SVM voting	93.89	93.92	93.91
(Kudo & Matsumoto, 2001)	SVM single	93.95	93.75	93.85
FRP-Chunker	F&R VP	94.17	93.26	93.71
(Zhang et al., 2002)	Winnow	93.54	93.60	93.57
(Kudo & Matsumoto, 2000)	SVM	93.45	93.51	93.48
(van Halteren, 2000)	MBL&WPD voting	93.13	93.51	93.32
(Tjong Kim Sang, 2000)	MBL voting	94.04	91.00	92.50

Table VII. Comparison on Noun Phrase Chunking, with the top-performing systems published so-far on the test set. Second column indicates the scope of the chunker, namely whether the system works specifically for that chunk (NP), or for the full set of chunks (all).

Reference	S	Technique	Prec.	Rec.	F_1
FRP-Chunker	NP	F&R VP	94.83	94.13	94.48
FRP-Chunker	all	F&R VP	94.48	94.33	94.41
(Kudo & Matsumoto, 2001)	all	SVM voting	94.47	94.32	94.39
(Zhang et al., 2002)	all	Winnow (+)	94.39	94.37	94.38
(Sha & Pereira, 2003)	NP	CRF	*unav.*	*unav.*	94.38
(Kudo & Matsumoto, 2001)	all	SVM single	94.54	94.09	94.32
(Sha & Pereira, 2003)	NP	MM-VP	*unav.*	*unav.*	94.09
(Zhang et al., 2002)	all	Winnow	93.80	93.99	93.89

solution space to a set of plausible candidates, and a ranking layer, which explores the candidates to select the optimal ones.

By conducting extensive experimental evaluation on two partial parsing tasks, we conclude that the global online learning via recognition feedback clearly outperforms alternative training strategies based on the traditional binary classification feedback. To explain this fact, we have shown in detail how the behavior of the filtering and the ranking layers during training is effectively adapted by the algorithm so as to optimize the global performance F_1 measure. Our technique has been shown to be competitive when comparing to other published works on the tasks. For Clause Identification, we have substantially improved the best result on the task, and for Chunking we have obtained a performance comparable to the top systems of the state-of-the-art.

Regarding the learning algorithm, we have provided analysis of its convergence. The convergence proof provided in Section 3.3 gives insight into the learning algorithm we propose but it has two drawbacks. First, it relies on very restrictive assumptions (two separability hypotheses). Second, the upper bound claimed by Theorem 2 does not illustrate the goodness of FR-Perceptron over the classical approach that considers two independent training steps. Note that assuming both separability hypothesis, this bound is worse than the bound one gets for independent training. One of the goals of future work will be to improve the convergence proof. We would like to find a convergence result that does not assume full start–end separability. Ideally, it had to show that the algorithm works well even if the start–end classifiers make one–side errors. In addition, the new proof had to provide a tighter bound on the number of errors. Here, one hopes to show a bound at least as good as the bound for the classical approach when training on samples having both separability hypotheses.

In the experimental work, we would like first to incorporate a kernel function which better exploits the structures we are dealing with. As an alternative of working in implicit feature spaces via kernel functions, we would like to work also with explicitly propositionalized rich feature spaces, as the ones defined in (Cumby & Roth, 2003). This approach would open the avenue for using the FR-Perceptron architecture with other online algorithms not suitable for kernels, which might report better results. We have in mind to replace Perceptron by some variants of the Winnow algorithm (Littlestone, 1988), an online mistake–driven algorithm with multiplicative updates.

Finally, we would like to move to other complex Natural Language Processing tasks which involve intermediate subtasks, so as to apply global learning strategies. Examples in the syntactic area may be to perform PoS tagging and chunking at the same time, or to scale up to the full parsing problem.

Acknowledgements

The authors would like to thank Dan Roth, Michael Collins, Thomas G. Dietterich and the two anonymous reviewers for their helpful comments relating this work. This work was supported by: the IST Programme of the European Community, under the PASCAL Network of Excellence (IST-2002-506778) and the Meaning project (IST-2001-34460); and the Spanish Research Department, under the Aliado project (TIC2002-

04447-C02). Xavier Carreras was supported by a pre-doctoral grant from the Catalan Research Department. This publication only reflects the authors' views.

References

Abney, S. P. (1991). Parsing by chunks. In Berwick, R. C., Abney, S. P., & Tenny, C. (Eds.), *Principle-based parsing: Computation and Psycholinguistics*, pp. 257–278. Kluwer, Dordrecht.

Altun, Y., Tsochantaridis, I., & Hofmann, T. (2003). Hidden Markov Support Vector Machines. In *Proceedings of the 20th International Conference on Machine Learning, ICML-03* Washington DC, USA.

Carreras, X., & Màrquez, L. (2001). Boosting Trees for Clause Splitting. In *Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001* Tolouse, France.

Carreras, X., & Màrquez, L. (2003a). Online Learning via Global Feedback for Phrase Recognition. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems, NIPS-03* Vancouver, Canada.

Carreras, X., & Màrquez, L. (2003b). Phrase Recognition by Filtering and Ranking with Perceptrons. In *Proceedings of the 4th Conference on Recent Advances on Natural Language Processing, RANLP-03* Borovets, Bulgaria.

Carreras, X., Màrquez, L., Punyakanok, V., & Roth, D. (2002). Learning and Inference for Clause Identification. In *Proceedings of the 14th European Conference on Machine Learning, ECML-02* Helsinki, Finland.

Collins, M. (2000). Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning, ICML-00* Stanford, CA USA.

Collins, M. (2002). Discriminative training methods for hidden markov models: Theory and experiments perceptron algorithms. In *Proceedings of the SIGDAT Conference on Empirical Methods in Natural Language Processing, EMNLP-02.*

Collins, M. (2004). Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Harry Bunt, J. C., & Satta, G. (Eds.), *New Developments in Parsing Technology*, chap. 2. Kluwer.

Crammer, K., & Singer, Y. (2003a). A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, *3*, 1025–1058.

Crammer, K., & Singer, Y. (2003b). Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, *3*, 951–991.

Cumby, C., & Roth, D. (2003). On Kernel Methods for Relational Learning. In *Proceedings of the 20th International Conference on Machine Learning, ICML-2003* Washington DC, USA.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, *37*(3), 277–296.

Har-Peled, S., Roth, D., & Zimak, D. (2002). Constraint Classification for Multiclass Classification and Ranking. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems, NIPS-02* Vancouver, Canada.

Haruno, M., Shirai, S., Ooyama, Y., & Aizawa, H. (1999). Using decision trees to construct a practical parser. *Machine Learning*, *34*(1-3).

Kudo, T., & Matsumoto, Y. (2001). Chunking with Support Vector Machines . In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference, NAACL-01* Pittsburgh, PA, USA.

Kudo, T., & Matsumoto, Y. (2002). Japanese Dependency Analyisis using Cascaded Chunking . In *Proceedings of the 6th Conference on Natural Language Learning, CoNLL-2002* Taipei, Taiwan.

Kudo, T., & Matsumoto, Y. (2000). Use of Support Vector Learning for Chunk Identification. In *Proceedings of CoNLL-2000 and LLL-2000* Lisbon, Portugal.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditonal random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning, ICML-01.*

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

Magerman, D. M. (1996). Learning Grammatical Structure Using Statistical Decision–Trees. In *Proceedings of the 3rd International Colloquium on Grammatical Inference, ICGI*, pp. 1–21. Springer-Verlag Lecture Notes Series in Artificial Intelligence 1147.

Molina, A., & Pla, F. (2001). Clause Detection using HMM. In *Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001* Tolouse, France.

Novikoff, A. B. J. (1962). On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, Vol. XII, pp. 615–622.

Punyakanok, V., & Roth, D. (2001). The use of classifiers in sequential inference. In *Proceedings of the 15th Annual Conference on Neural Information Processing Systems, NIPS-01.*

Punyakanok, V., & Roth, D. (2004). Inference with classifiers: The phrase identification problem. *Journal of Machine Learning Research*, *to appear.*

Ramshaw, L. A., & Marcus, M. P. (1995). Text chunking using transformation-based learning. In *Proceedings of the Third Annual Workshop on Very Large Corpora.*

Ratnaparkhi, A. (1999). Learning to Parse Natural Language with Maximum-Entropy Models. *Machine Learning*, *34*(1), 151–175.

Sha, F., & Pereira, F. (2003). Shallow parsing with conditional random fields. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL.*

Sutton, C., Rohanimanesh, K., & McCallum, A. (2004). Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. In *Proceedings of the 21st International Conference on Machine Learning, ICML-04* Alberta, Canada.

Taskar, B., Guestrin, C., & Koller, D. (2003). Max-Margin Markov Networks. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems, NIPS-03* Vancouver, Canada.

Tjong Kim Sang, E. F., & Buchholz, S. (2000). Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 4th Conference on Natural Language Learning, CoNLL-2000.*

Tjong Kim Sang, E. (2000). Text Chunking by System Combination. In *Proceedings of CoNLL-2000 and LLL-2000* Lisbon, Portugal.

Tjong Kim Sang, E. F., & Déjean, H. (2001). Introduction to the CoNLL-2001 shared task: Clause identification. In *Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001.*

van Halteren, H. (2000). Chunking with WPDV Models. In *Proceedings of CoNLL-2000 and LLL-2000* Lisbon, Portugal.

Zhang, T., Damereau, F., & Johnson, D. (2002). Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, *2*, 615–637.