Phrase Recognition by Filtering and Ranking with Perceptrons

Xavier Carreras and **Lluís Màrquez**
TALP Reseach Center
Technical University of Catalonia (UPC)
Campus Nord UPC, E-08034 Barcelona
{carreras,lluism}@lsi.upc.es

Abstract

We present a phrase recognition system based on perceptrons, and an online learning algorithm to train them together. The recognition strategy applies learning in two layers, first at word level, to filter words and form phrase candidates, second at phrase level, to rank phrases and select the optimal ones. We provide a global feedback rule which reflects the dependencies among perceptrons and allows to train them together online. Experimentation on Partial Parsing problems and Named Entity Extraction gives state-of-the-art results on the CoNLL public datasets. We also provide empirical evidence that training the functions together is clearly better than training them separately, as in the conventional approach.

1 Introduction

Over the past few years, many machine learning methods have been successfully applied to tasks in which phrases of some type have to be recognized. In Natural Language, relevant problems of this type include Named Entity Recognition and Classification (NERC), (Tjong Kim Sang & De Meulder 03), or, in the syntactic level, partial parsing tasks (Tjong Kim Sang & Buchholz 00; Tjong Kim Sang & Déjean 01), or even full parsing (Ratnaparkhi 99).

The general approach consists of decomposing the global phrase recognition problem into a number of local learnable subproblems, and infer the global solution from the outcomes of the local subproblems. For chunking problems –as shallow parsing or named entity extraction– the approach is typically to perform a tagging. Here, local subproblems include learning whether a word *opens*, *closes*, or is *inside* a phrase of some type (noun phrase, verb phrase, ...), and the inference process consists of sequentially computing the optimal tag sequence which encodes the phrases, by means of dynamic programming (Punyakanok & Roth 01). When hierarchical structure has to be recognized, additional local decisions are required to determine the embedding of phrases, which results in a more complex inference process which recursively builds the global solution (Ratnaparkhi 99; Carreras *et al.* 02; Kudo & Matsumoto 02).

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, and 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 directly the local accuracy measure for which the local classifiers are trained.

Recent works on the area provide alternative learning strategies in which the learning process is guided from a global point of view. In the online setting, (Collins 02) presents a variant of the perceptron for tagging, in which the feedback for the perceptron is given from the output of the Viterbi decoding algorithm. Also, (Crammer & Singer 03) present a topic-ranking algorithm, in which several perceptrons receive feedback from the ranking they produce over a training instance. Alternatively, works on sequential learning provide probabilistic conditional models (Lafferty *et al.* 01) or derivations of margin-based algorithms for the sequential setting (Altun *et al.* 03).

In this paper we present a global learning strategy for the general task of recognizing phrases. We adopt our general phrase recognition model presented in (Carreras *et al.* 02). 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. The inference for the global solution consists of an exploration of all coherent solutions to find the best scored one. By working also at phrase level, the model allows to design features over partial structures of the solution, richer than the usual word-level features.

As a main contribution, we propose a recognition-based feedback rule which allows to learn the de-

cisions in the system as perceptrons, all in one go. The learning strategy works online at sentence level. When visiting a sentence, the perceptrons are first used to recognize the set of phrases, and then updated according to the correctness of their solution. The recognition feedback reflects to each individual perceptron its committed errors from a global point of view. As a result, the resulting learned functions behave as filters and rankers, rather than default classifiers, which we argue to be better for these tasks.

In the experimentation on three relevant problems, namely Chunking, Clause Identification and Named Entity Extraction, our learning architecture exhibits state-of-the-art performance, while being conceptually simple and flexible.

2 The Phrase Recognition Model

In this section we first formalize the problem and then present the phrase recognition system.

2.1 Formalization

Let x be a sentence belonging to the sentence space $\mathcal{X}$, formed by n words x_i, with i ranging from 0 to $n-1$. Let $\mathcal{K}$ be a predefined set of phrase categories. In syntactic parsing, $\mathcal{K}$ may include noun phrases, verb phrases, and clauses, among others. A *phrase*, denoted as $(s,e)_k$, is the sequence of consecutive words spanning from word x_s to word x_e, having $s \leq e$, with category $k \in \mathcal{K}$.

Let $ph_1 = (s_1,e_1)_{k_1}$ and $ph_2 = (s_2,e_2)_{k_2}$ be two different phrases. We define that ph_1 and ph_2 *overlap* iff $s_1 < s_2 \leq e_1 < e_2$ or $s_2 < s_1 \leq e_2 < e_1$, and we note it as $ph_1 \sim ph_2$. Furthermore, we define that ph_1 is *embedded* in ph_2 iff $s_2 \leq s_1 \leq e_1 \leq e_2$, and we note it as $ph_1 \prec ph_2$.

Let $\mathcal{P}$ be the set of all possible phrases, formally expressed as $\mathcal{P} = \{(s,e)_k \mid 0 \leq s \leq e, k \in \mathcal{K}\}$. A *solution* for a phrase recognition problem is a 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, which we refer to as *chunking*, we do not allow phrases to overlap or embed. Thus, the solution space can be formally expressed as $\mathcal{Y} = \{y \subseteq \mathcal{P} \mid \forall\, ph_1, ph_2 \in y \;\; ph_1 \not\sim ph_2 \wedge ph_1 \not\prec ph_2\}$.

More generally, for the problem of recognizing phrases organized hierarchically, a solution is a set of phrases which do not overlap but may be embedded. Formally, the solution space is $\mathcal{Y} = \{y \subseteq \mathcal{P} \mid \forall\, ph_1, ph_2 \in y \;\; ph_1 \not\sim ph_2\}$.

In order to evaluate a phrase recognition system we use the standard measures for recognition tasks: *precision*, *recall* and their harmonic mean $F_{\beta=1}$.

2.2 Recognizing Phrases

The Phrase Recognizer is a function which, given a sentence x, identifies the set of phrases y of x: $\mathrm{PhRec} : \mathcal{X} \rightarrow \mathcal{Y}$.

We assume two components within this function, both will be learning components of the recognizer. First, we assume a function PhCI which, given a sentence x, identifies a set of candidate phrases, not necessarily coherent, for the sentence, $\mathrm{PhCI}(x) \subseteq \mathcal{P}$.

Second, we assume a number of scoring functions, which, given a phrase, produce a real-valued score indicating the plausibility of the phrase. In particular, for each category $k \in \mathcal{K}$ we assume a function score_k which, given a phrase, produces a positive score if the phrase is likely to belong to category k, and a negative score otherwise.

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

$$\mathrm{PhRec}(x) = \arg \max_{y \subseteq \mathrm{PhCI}(x) \mid y \in \mathcal{Y}} \sum_{(s,e)_k \in y} \mathrm{score}_k(s,e) \quad (1)$$

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

The function PhCI is only used to reduce the search space of the PhRec function. Note that the PhRec function constructs the optimal phrase set by evaluating scores of phrase candidates, and, regarding the length of the sentence, there is a quadratic number of possible phrases, that is, the set $\mathcal{P}$. Thus, considering straightforwardly all phrases in $\mathcal{P}$ would result in a very expensive exploration. The function PhCI 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: each word of the sentence is tested as *start* —if it is likely to start phrases— and as *end* —if it is likely to end phrases. Each start word s with each end word e, having $s \leq e$, forms the phrase candidates $(s,e)_k, k \in \mathcal{K}$. Alternatives to this setting may be to consider a pair of *start-end* classifiers for each category in $\mathcal{K}$, or to perform a *Begin-Inside* classification, either projected to categories or not. In general, each classifier will be applied to each word in the sentence, and deciding the best strategy for identifying phrase candidates will de-

pend 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 the nature of it, 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 over 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 01). When embedding of phrases is allowed, a cubic-time bottom-up exploration is required (Carreras *et al.* 02). As noted above, in either cases there will be the additional cost of applying a quadratic number of decisions for scoring phrases.

Summarizing, the phrase recognition system is performed in two layers: the identification layer, which filters out phrase candidates in linear time, and the scoring layer, which selects the optimal coherent phrase set in quadratic or cubic time.

3 Learning via Recognition Feedback

In this section we describe an online learning strategy for training the learning components of the Phrase Recognizer, namely the classifiers in PhCI and the functions score_k, $k \in \mathcal{K}$. Each function is implemented using a perceptron[1] and a representation function.

A perceptron is a function $\mathrm{h}_{\mathbf{w}} : \mathbb{R}^n \rightarrow \mathbb{R}$ parametrized by a vector $\mathbf{w}$ in $\mathbb{R}^n$, which given an instance $\mathbf{x} \in \mathbb{R}^n$ outputs as prediction the inner product between the $\mathbf{x}$ and $\mathbf{w}$ vectors, $\mathrm{h}_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$. To remind the reader, the traditional training algorithm for binary classification works as follows. It starts with the vector $\mathbf{w}$ initialized to zeros, $\mathbf{w} = 0$. Given a new example $(\mathbf{x}, y)$, it predicts the label of $\mathbf{x}$ as $\hat{y} = sign(\mathbf{w} \cdot \mathbf{x})$. The learning strategy is mistake-driven: if the predicted output $\hat{y}$ differs from y, then the prediction vector is updated by orienting it toward $\mathbf{x}$, $\mathbf{w} = \mathbf{w} + y\mathbf{x}$; if the prediction is correct, $\mathbf{w}$ is left unchanged.

The representation function $\Phi : \mathcal{X} \rightarrow \mathbb{R}^n$ takes an instance x belonging to some space $\mathcal{X}$ and outputs a vector in $\mathbb{R}^n$ with which the perceptron operates. We will discuss a setting for this function in section 5.

Let us fix the PhCI function to a *start-end* classification. The function consists of two classifiers, h_{S} and h_{E}, each of which predicts whether a word x starts or ends a phrase, respectively. The function PhCI is formed by two perceptron vectors, $\mathbf{w}_{\mathrm{S}}$ and $\mathbf{w}_{\mathrm{E}}$, and a shared representation function Φ_{w}. A classifier prediction is computed as $\mathrm{h}_{\mathrm{S}}(x) = \mathbf{w}_{\mathrm{S}} \cdot \Phi_{\mathrm{w}}(x)$, and similarly for h_{E}, and the sign is taken as the binary classification.

The functions score_k, for $k \in \mathcal{K}$, compute a score for a phrase (s, e) being a phrase of category k. For each function there is a vector w_k, and there is a shared function Φ_{p}. The score is given by the expression $\mathrm{score}_k(s, e) = \mathbf{w}_k \cdot \Phi_{\mathrm{p}}(s, e)$.

3.1 Learning Algorithm

The learning problem consists in setting the parameter vectors $\mathbf{w}$ of each perceptron. We propose a mistake-driven online learning algorithm for training the parameter vectors all together.

The algorithm starts with all vectors initialized to 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. If the predicted phrase set is not perfect the vectors responsible of the incorrect prediction are updated additively. The sentence-based learning algorithm is as follows:

- Input: $\{(x^1, y^1), \ldots, (x^m, y^m)\}$, x^i are sentences, y^i are solutions in $\mathcal{Y}$
- Define: $W = \{\mathbf{w}_{\mathrm{S}}, \mathbf{w}_{\mathrm{E}}\} \cup \{w_k | k \in \mathcal{K}\}$.
- Initialize: $\forall \mathbf{w} \in W$ $\mathbf{w} = 0$;
- for $t = 1 \ldots T$, for $i = 1 \ldots m$:
  1. $\hat{y} = \mathrm{PhRec}_W(x^i)$
  2. $\mathrm{learning\_feedback}(W, x^i, y^i, \hat{y})$
- Output: the vectors in W.

The learning feedback is specified in Figure 1. We stem from the traditional Perceptron update rule for binary classification described above. By analyzing the dependencies between each perceptron and a solution, we derive a feedback rule which naturally fits the phrase recognition setting. The feedback models 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. Therefore, we move the *start* or *end* prediction vectors toward the misclassified boundary words of a missed

[1] Actually, we use a variant of the model called the *voted perceptron*, explained below in section 3.3.

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

Figure 1: The Recognition Feedback. y^* is the gold set of phrases for a sentence x, and $\hat{y}$ is the set predicted by the PhRec function. $\mathrm{goldS}(i)$ and $\mathrm{goldE}(i)$ are respectively the perfect indicator functions for *start* and *end* classifications, i.e. they return 1 if word x_i starts/ends some phrase in y^* and 0 otherwise.

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 gold solution. Note that we deliberately do not care about false positives *start* or *end* words which do not finally over-produce a phrase.

Regarding the scoring layer, each category prediction vector is moved toward missed phrases and moved away from over-predicted phrases.

Intrinsically, this simple feedback rule approximates the desired behavior of the global PhRec 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 The Classification Setting Alternative

The usual alternative to the described learning algorithm is to train each function of the system separately. To do so, each function is modeled as a classifier. The *start-end* classifiers, working at word level, decide whether a particular word starts and/or ends or not some phrase in the correct solution. The score functions, working at phrase level, decide whether a particular phrase candidate is in the solution or not.

A main shortcoming of the classification modeling is that the individual classification loss functions do not directly reflect the global goal of the phrase recognition problem, measured in terms of precision and recall. This fact constitutes a major motivation for learning all the functions together.

Second, it is not clear how to generate training examples for training the score functions. Note that these functions operate at phrase level, which is quadratic over the length of a sentence. Thus, considering a training instance for each possible phrase in a collection of thousands of sentences generates millions of examples, most of which are negatives, and makes usual learning techniques no feasible at all. A practical issue is to generate phrase examples only for the correct *start* and *end* words, but then there are no guarantees on scoring phrase candidates generated by false positive boundary words. By training all functions together, in an online fashion, we are able to control the interaction of both layers in a natural way. On section 6.3 we give empirical evidence that the learning approach we propose works better than the classification setting.

3.3 Voted Perceptron and Kernelization

Although the analysis above concerns the perceptron algorithm, we use a modified version, the *voted perceptron* algorithm, presented and detailed in (Freund & Schapire 99). The key point of the voted version is that, while training, it stores information in order to make more robust predictions on test data. In particular, each perceptron vector generated after every mistake during training is stored, together with a weight for the vector. In our architecture, this weight corresponds to the number of positive decisions correctly predicted by the vector. Then, when testing, the prediction is an *averaged* vote over the predictions of each vector.

As a secondary issue, in the same work the dual formulation of the perceptron is presented, which allows the use of *kernel functions*. These functions allow to efficiently work in richer feature spaces in which the learning problem may be easier to solve. In this paper we work with polynomial kernels $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$, where d is the *degree* of the kernel. In binary spaces, this kernel has the effect of working with all the conjunctions of the initial features of up to size d (Cristianini & Shawe-Taylor 00).

4 Phrase Recognition in Natural Language

In this section we briefly describe problems in the Natural Language domain in which phrases of some type have to be recognized.

Chunking In this problem, also known as Shallow Parsing, the base syntactic phrases, or chunks, of a

sentence have to be recognized. The chunks in a sentence can not overlap and are non-recursive, that is, they can not be embedded. We followed the setting of the CoNLL-2000 shared task (Tjong Kim Sang & Buchholz 00), which provides data for English[2]. The problem consists of recognizing the chunks of a sentence on the basis of words and part-of-speech tags (PoS). There are 11 types of chunks (noun phrases, verb phrases, prepositional phrases, ...). The data consists of sections of the Penn WSJ treebank, namely a training set (sections 15–18, 8,936 sentences), and a test set (section 20, 2,012 sentences) to perform evaluation and compare with other systems. In order to perform tuning of our system, we generated development data by dividing the training data into a training set (sections 15–17) and a validation set (section 18).

Clause Identification The goal of is the problem is to recognize the clauses of a sentence. A clause can be roughly defined as a phrase with a subject, possibly implicit, and a predicate. Clauses in a sentence form a hierarchical structure which constitutes the skeleton of the full syntactic tree. Thus, embedding of clauses is allowed. We followed the setting of the CoNLL-2001 shared task[3] (Tjong Kim Sang & Déjean 01) on Clause Identification for English. The problem consists of recognizing the set of clauses on the basis of words, part-of-speech tags, and chunks. There is only one category of phrases to be considered, namely the clauses. As in chunking, the data consists of sections of the Penn WSJ treebank, namely a training set (sections 15–18, 8,936 sentences), a development set (section 20, 2,012 sentences) for tuning the system, and a test set (section 21, 1,671 sentences) to evaluate and compare with other systems.

Named Entity Recognition and Classification (NERC) The goal of this problem is to recognize Named-Entity (NE) phrases in a sentence and categorize them according to some predefined set of categories. In this problem, embedding of phrases is not allowed. We followed the setting of the CoNLL-2003 shared task[4], which concerned the problem for the English language, considering four NE types, $\mathcal{K} = \{\text{LOCATION}, \text{ORGANIZATION}, \text{PERSON}, \text{OTHERS}\}$. The data provided is a part of the Reuters corpus, including a training set (14,987 sentences), a development set (3,466), and a test set (3,684 sentences) to perform evaluation.

[2] `http://cnts.uia.ac.be/conll2000/chunking`
[3] `http://cnts.uia.ac.be/conll2001/clauses`
[4] `http://cnts.uia.ac.be/conll2003/ner`

5 Feature-Vector Representation

In this section we describe the representation functions Φ_{w} and Φ_{p}, which respectively map a word or a phrase and their local context into a feature vector in $\mathbb{R}^n$, which in practice is a binary space. First we define a set of primitive functions which apply to words or sequences of words:

- **Word(w)**: The form of word w.
- **PoS(w)**: The part-of-speech tag of word w.
- **ChunkTag(w)**: The chunk tag of word w.
- **OrthoFlags(w)**: Binary flags of word w with regard to how is it capitalized (*initial-caps*, *all-caps*), the kind of characters that form the word (*contains-digits*, *all-digits*, *alphanumeric*, *roman-number*), the presence of punctuation marks (*contains-dots*, *contains-hyphen*, *acronym*), single character patterns (*lonely-initial*, *punctuation-mark*, *single-char*), or the membership of the word to a predefined class (*functional-word*[5]), or pattern (*URL*).
- **OrthoTag($w_s \dots w_e$)**: A tag with regard to orthographic features, which is either *capitalized* (`C`), *lowercased* (`l`), *functional* (`f`), *punctuation mark* (`.`), *quote* (`'`) or *other* (`x`).
- **Affixes(w)**: The prefixes and suffixes of the word w (up to 4 characters).
- **P-Gram($w_s \dots w_e$)**: The conjunction of the outputs of the primitive P on words $w_s \dots w_e$. We work with Word-Grams, PoS-Grams and OrthoTag-Grams . For instance, the OrthoTag-Gram for "John Smith payed 3 euros" is `CClxl`.

Representing Words For the function $\Phi_{\text{w}}(x_i)$, we compute primitives in a *window* of words around x_i, that is, words x_{i+l} with $l \in [-L_{\text{w}}, +L_{\text{w}}]$. Each primitive label, together with each relative position l and each returned value forms a final binary indicator feature. Specifically, we compute:

- Word and PoS primitives.
- PoS-Grams on all sequences within the window which include the central word.
- For Clause Id., also ChunkTag primitives (given in the input).
- For NERC, also OrthoFlags, Affixes, and OrthoTag-Grams on all sequences which include the central word.
- Left Start-Ends: flags indicating whether the words in $[-L_{\text{w}}, -1]$ have been predicted as *start* and/or *end* words of a k-phrase, $k \in \mathcal{K}$.

[5] Functional words are determiners, prepositions and conjunctions.

Representing Phrases For the function $\Phi_{\mathrm{p}}(s,e)$, we capture the context of the phrase and the phrase itself. For the context, we evaluate:

- A $[-L_{\mathrm{p}},0]$ window of primitives at the s word and a separate $[0,+L_{\mathrm{p}}]$ window at the e word. These windows include Words and PoS; on Clause Id., also ChunkTags; on NERC, also OrthoFlags and OrthoTag-Grams features.
- Left Phrases (only for Chunking and NERC): Representation of the elements to the left of the s word. An element is either an already recognized phrase (we reduce its words into a single element and represent it by the phrase type) or a word outside a phrase (represented by its PoS). We consider up to 3 elements, and codify the relative position of each one, and also all conjunctions.

As for the (s,e) phrase itself, we evaluate primitives from s to e without capturing the relative position. Specifically we capture:

- The length of the phrase (real-valued feature).
- Word and PoS primitives.
- PoS-Grams on all subsequences of $w_s..w_e$ of up to size 3, and also the complete PoS-Gram on the whole sequence.
- Internal Pattern (only for Clause Id.): Concatenation of the relevant elements in the sentence fragment $w_s..w_e$. The following elements are considered: a) Punctuation marks and coordinate conjunctions; b) The word "`that`"; c) Relative pronouns; d) Verb phrase chunks; and e) The top clauses within the s to e fragment, already recognized through the bottom up search (a clause in a pattern reduces all the elements within it into an atomic element).
- For NERC, also: a) the Word-Gram of the whole phrase; b) OrthoTag-Grams on subsequences of sizes 2, 3 and 4; c) the affixes of each word.

Experimenting on the respective development sets, for Chunking we set both L_{w} and L_{p} to 2, whereas for the two other problems we set them to 3.

6 Experimentation

We experimented with our phrase recognition system on the three problems presented above. As stated, each system is composed of Start-End functions and Scoring functions.

6.1 Learning Details

For Chunking and Clause Identification we set a pair of *start-end* functions for each type of phrase, whereas for NERC we set only two *start-end* functions which were shared among all NE types.

Regarding the feature space, we filtered out features occurring less than 3 times in training. As for the polynomial kernel, we fixed the degree of all functions to 2, since initial experiments on the development sets showed poor performance for the linear case (specially on Clause Id.) and no significant improvements for higher degrees. Also, concerning features which represent the solution being recognized (Left Start-Ends, Left Phrases, Internal Pattern), we found better to encode the outputs of the functions being learned, rather than the correct values of the solution.

In Clause Identification, to avoid some computation in the *start-end* layer, we did not consider clause boundary words which would break the chunking provided in the input, that is, we do not consider clauses which would overlap with chunks.

On each problem, we ran the learning algorithm which trains all the functions together for 15 epochs on the training data, and evaluated the performance on the development sets to select the optimal point in terms of F_1. As a general behavior, on each problem the global performance substantially increased during the first 5 epochs, and then became somewhat stable, with minor improvements (below, Figure 2 plots the learning curve on Clause Identification).

6.2 Results and Comparison to Related Works

Table 1 shows the obtained performance on each problem. The results are fairly good in all cases.

On Chunking, we obtained a very good performance of 93.74 which would situate us in first position at competition time (Tjong Kim Sang & Buchholz 00). The two best published works on this data perform the task as a tagging, solved with multiclass learning techniques. (Kudo & Matsumoto 01) performed several taggings with SVM classifiers, which were later combined. They report a performance of 93.85 with an individual tagging and 93.91 by combining many taggings. 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 we exploit rich features on phrases. The best work on the data is (Zhang *et al.* 02), which apply regularized Winnow. They report a base performance of 93.57,

		development			test		
	T	precision	recall	$F_{\beta=1}$	precision	recall	$F_{\beta=1}$
Chunking	10	-	-	-	94.19%	93.29%	93.74
Clause Id.	11	89.80%	84.05%	86.83	87.99%	81.01%	84.36
NERC	12	89.59%	88.17%	88.87	83.93%	83.43%	83.68

Table 1: Results of the three problems on the development and test sets. The T column shows the optimal number of epochs, tuned on the development sets.

and an improved performance of 94.17 by making use of external grammatical information.

Table 2 shows the performance of our chunker on each individual phrase type. Looking at the recognition of Noun Phrases (NP), our system, achieving 94.41, slightly outperforms recent systems trained specifically for this chunk. (Kudo & Matsumoto 01) obtained 94.29 with combination of SVMs. (Sha & Pereira 03) obtained 94.38 with Conditional Random Fields. They also report 94.09 for the voted perceptron tagging architecture presented in (Collins 02).

On Clause Identification, our system obtains a similar performance than the best system published so-far (Carreras *et al.* 02), which obtained (p=92.53%; r=82.48%; f1=87.22) on the development and (p=90.18%; r=78.11%; f1=83.71) on the test. Our performance is lower on the development, but much better on the test. That system made use of the same phrase recognition model, and the decisions were learned by AdaBoost classifiers. Also, the scoring function was a robust combination of several classifiers. We provide some more discussion on this comparison in the following experiment.

On NERC, we obtained a performance of 83.68 on the test set, which is competitive but far from the top systems of the competition (Tjong Kim Sang & De Meulder 03), which achieved above 88 in F1. It seems that the feature engineering and use of external resources allows to substantially improve the results on this problem. Our system at competition time (Carreras *et al.* 03) was based on a similar learning architecture, the difference being that at word level we performed *Begin-Inside* decisions instead of *Start-End* decisions. There we achieved a slightly better performance of (p=85.81%; r=82.84%; f1=84.30).

	precision	recall	$F_{\beta=1}$
ADJP	83.38%	67.58%	74.65
ADVP	83.54%	77.94%	80.65
CONJP	60.00%	33.33%	42.86
INTJ	100.00%	100.00%	100.00
LST	0.00%	0.00%	0.00
NP	94.47%	94.36%	94.41
PP	96.55%	97.86%	97.20
PRT	80.72%	63.21%	70.90
SBAR	91.18%	79.25%	84.80
VP	94.22%	93.52%	93.87
all	94.19%	93.29%	93.74

Table 2: Recognition results on Chunking per types.

6.3 On Feedback: Recognition *vs* Classification

In this experiment we were interested in comparing the learning strategy via recognition feedback against training the functions separately as classifiers, as discussed in section 3.2. We did the comparison on Clause Identification. In both cases, as note above, the model was composed by *start-end* functions, which identify clause candidates, and a score function.

In order to learn the functions separately as classifiers, we generated three data sets from the training data, one for each function. For the *start-end* sets, we considered all words in the data, except those breaking chunks. For the scoring layer, we generated all phrase candidates formed with all pairs of correct phrase boundaries. We then ran the voted perceptron algorithm on each set for up to 15 epochs. We also trained SVM[6] classifiers adjusting the soft margin C parameter on the development set.

Figure 2 shows the performance curve on the development set in terms of the F_1 measure with respect to the number of epochs during training. Clearly, the behaviour of the Rec-VP is much better than the others: the recognition model gets stable around 86.5, whereas the classification models achieve around 79 for VP, and 81 for SVM. Given this evidence, the learning architecture we present seems to be better than the usual strategies based on separate classifiers.

As noted above, in our previous work (Carreras *et al.* 02) we achieved comparable performance (though our new results are better on the test) with the same

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

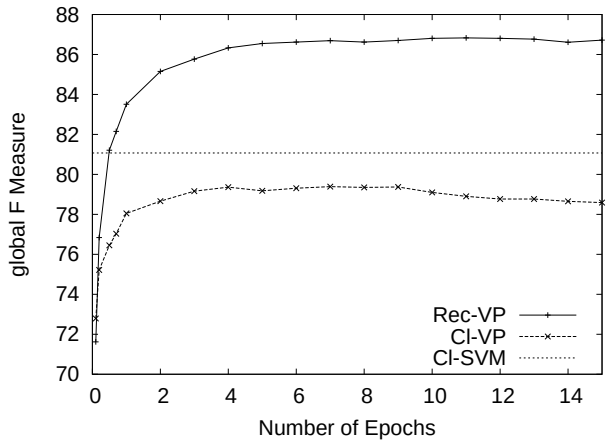

Figure 2: Evolution of the performance on Clause Identification with respect to the visited training sentences. Rec-VP is the voted perceptron model trained via recognition feedback. Cl-VP and Cl-SVM are respectively the voted perceptron and SVM models trained batch via classification feedback.

recognition model, but the functions implemented as AdaBoost classifiers. The learning strategy there was training first classifiers for the *start-end* layer, and then generate training examples for the score function, taking into account the *start-end* behaviour. This required a tuning procedure to select the amount of incorrect boundaries introduced to produce negative clause examples. Also, the score function was a robust combination of several classifiers. In any case, our new learning strategy is much simpler, flexible and produces state-of-the-art results.

7 Conclusions

We have presented a novel learning architecture for general phrase structure recognition. The method makes use of several decision functions operating in two layers: at word level, to identify phrase candidates, and at phrase level, to score the optimal ones. Doing so, we are able to incorporate rich features which represent partial structures of the solution.

The main contribution of the work is to propose a very simple online learning algorithm for training, at the same time, all the involved functions in the form of voted perceptrons.

We have empirically proved the generality and feasibility of the approach by applying it to different phrase recognition problems on Named Entity recognition and partial parsing, in which we improve the state-of-the-art. The experimentation evidences that exploiting the interaction between learned functions during learning results in a better global behaviour.

Acknowledgments

Research partially funded by the European Commission (Meaning, IST-2001-34460) and the Spanish Research Dept. (Hermes, TIC2000-0335-C03-02; Petra - TIC2000-1735-C02-02). Xavier Carreras holds a predoctoral grant by the Catalan Research Dept.

References

(Altun *et al.* 03) Y. Altun, T. Hofmann, and M. Johnson. Loss Functions and Optimization Methods for Discriminative Learning of Label Sequences . In *Proc. of the EMNLP*, 2003.

(Carreras *et al.* 02) X. Carreras, L. Màrquez, V. Punyakanok, and D. Roth. Learning and Inference for Clause Identification. In *Proc. of the 14th ECML*, 2002.

(Carreras *et al.* 03) X. Carreras, L. Màrquez, and L. Padró. Learning a Perceptron-Based Named Entity Chunker via Online Recognition Feedback. In *Proc. of CoNLL-2003*, 2003.

(Collins 02) M. Collins. Discriminative training methods for hidden markov models: Theory and experiments perceptron algorithms. In *Proc. of the EMNLP'02*, 2002.

(Crammer & Singer 03) K. Crammer and Y. Singer. A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 2003.

(Cristianini & Shawe-Taylor 00) N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

(Freund & Schapire 99) Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 1999.

(Kudo & Matsumoto 01) T. Kudo and Y. Matsumoto. Chunking with Support Vector Machines . In *Proc. of NAACL 2001*, 2001.

(Kudo & Matsumoto 02) T. Kudo and Y. Matsumoto. Japanese Dependency Analyisis using Cascaded Chunking . In *Proc. of CoNLL-2002*, 2002.

(Lafferty *et al.* 01) J. Lafferty, A. McCallum, and F. Pereira. Conditonal random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML'01*, 2001.

(Punyakanok & Roth 01) V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *Proc. of NIPS*, 2001.

(Ratnaparkhi 99) A. Ratnaparkhi. Learning to Parse Natural Language with Maximum-Entropy Models. *Machine Learning*, 1999.

(Sha & Pereira 03) F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proc. of HLT-NAACL*, 2003.

(Tjong Kim Sang & Buchholz 00) E. F. Tjong Kim Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proc. of CoNLL-2000 and LLL-2000*, 2000.

(Tjong Kim Sang & De Meulder 03) Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL-2003*. Edmonton, Canada, 2003.

(Tjong Kim Sang & Déjean 01) Erik F. Tjong Kim Sang and Hervé Déjean. Introduction to the CoNLL-2001 shared task: Clause identification. In *Proceedings of CoNLL-2001*, 2001.

(Zhang *et al.* 02) T. Zhang, F. Damereau, and D. Johnson. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2002.