Learning a Perceptron-Based Named Entity Chunker via Online Recognition Feedback

Xavier Carreras and Lluís Màrquez and Lluís Padró
TALP Research Center
Departament de Llenguatges i Sistemes Informàtics
Universitat Politécnica de Catalunya
{carreras, lluis, padro}@lsi.upc.es

1 Introduction

We present a novel approach for the problem of Named Entity Recognition and Classification (NERC), in the context of the CoNLL-2003 Shared Task.

Our work is framed into the learning and inference paradigm for recognizing structures in Natural Language (Punyakanok and Roth, 2001; Carreras et al., 2002). We make use of several learned functions which, applied at local contexts, discriminatively select optimal partial structures. On the top of this local recognition, an inference layer explores the partial structures and builds the optimal global structure for the problem.

For the NERC problem, the structures to be recognized are the named entity phrases (NE) of a sentence. First, we apply learning at word level to identify NE candidates by means of a Begin-Inside classification. Then, we make use of functions learned at phrase level —one for each NE category— to discriminate among competing NEs.

We propose a simple online learning algorithm for training all the involved functions together. Each function is modeled as a voted perceptron (Freund and Schapire, 1999). The learning strategy works online at sentence level. When visiting a sentence, the functions being learned are first used to recognize the NE phrases, and then updated according to the correctness of their solution. We analyze the dependencies among the involved perceptrons and a global solution in order to design a global update rule based on the recognition of named-entities, which reflects to each individual perceptron its committed errors from a global perspective.

The learning approach presented here is closely related to—and inspired by—some recent works in the area of NLP and Machine Learning. Collins (2002) adapted the perceptron learning algorithm to tagging tasks, via sentence-based global feedback. Crammer and Singer (2003) presented an online topic-ranking algorithm involving several perceptrons and ranking-based update rules for training them.

2 Named-Entity Phrase Chunking

In this section we describe our NERC approach as a phrase chunking problem. First we formalize the problem of NERC, then we propose a NE-Chunker.

2.1 Problem 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 the set of NE categories, which in the CoNLL-2003 setting is \( \mathcal{K} = \{ \text{LOC, PER, ORG, MISC} \} \).

A NE phrase, denoted as \((s,e)_k\), is a phrase spanning from word \( x_s \) to word \( x_e \), having \( s \leq e \), with category \( k \in \mathcal{K} \). Let \( \mathcal{NE} \) be the set of all potential NE phrases, expressed as \( \mathcal{NE} = \{(s,e)_k \mid 0 \leq s \leq e, k \in \mathcal{K} \} \).

We say that two different NE phrases \( ne_1 = (s_1,e_1)_k \) and \( ne_2 = (s_2,e_2)_k \) overlap, denoted as \( ne_1 \sim ne_2 \) iff \( e_1 \geq s_2 \land e_2 \geq s_1 \). A solution for the NERC problem is a set \( y \) formed by NE phrases that do not overlap, also known as a chunking. We define the set \( \mathcal{Y} \) as the set of all possible chunkings. Formally, it can be expressed as:

\[
\mathcal{Y} = \{ y \subseteq \mathcal{NE} \mid \forall ne_1, ne_2 \in y \ ne_1 \not\sim ne_2 \}
\]

The goal of the NE extraction problem is to identify the correct solution \( y \in \mathcal{Y} \) for a given sentence \( x \).

2.2 NE-Chunker

The NE-Chunker is a function which given a sentence \( x \in \mathcal{X} \) identifies the set of NE phrases \( y \in \mathcal{Y} \):

\[
\text{NECh} : \mathcal{X} \to \mathcal{Y}
\]

The NE-Chunker recognizes NE phrases in two layers of processing. In the first layer, a set of NE candidates for a sentence is identified, out of all the potential phrases in \( \mathcal{NE} \). To do so, we apply learning at word level in order to perform a Begin-Inside classification. That is, we assume a function \( \text{BI}_W(w) \) which decides whether a word \( w \) begins a NE phrase or not, and a function \( \text{BI}^*_W(w) \) which decides whether a word is inside a NE phrase or not. Furthermore, we define the predicate \( \text{BI}^+ \), which tests whether a certain phrase is formed by
a starting \textit{begin} word and subsequent \textit{inside} words. Formally, $BI^*(s, e) = (l_B(s) \land \forall i: s < i \leq e : l_1(i))$.

The recognition will only consider solutions formed by phases in $\mathcal{NE}$ which satisfy the $BI^*$ predicate. Thus, this layer is used to filter out candidates from $\mathcal{NE}$ and consequently reduce the size of the solution space $\mathcal{Y}$. Formally, the solution space that is explored can be expressed as $\mathcal{Y}_{BI^*} = \{ y \in \mathcal{Y} \mid \forall ne \in y \ BI^*(ne) \}$.

The second layer selects the best coherent set of NE phrases by applying learning at phrase level. We assume a function $\text{score}_k$ which produces a positive score if the phrase is likely to belong to category $k$, and a negative score otherwise.

Given this, the NE-Chunker is a function which searches a NE chunking for a sentence $x$ according to the following optimality criterion:

$$\text{NEch}(x) = \arg \max_{y \in \mathcal{Y}_{BI^*}} \sum_{(s,e) \in y} \text{score}_k(s, e)$$

That is, among the considered chunkings of the sentence, the optimal one is defined to be the one whose NE phrases maximize the summation of phrase scores. Practically, there is no need to explicitly enumerate each possible chunking in $\mathcal{Y}_{BI^*}$. Instead, by using dynamic programming the optimal chunking can be found in quadratic time over the sentence length, performing a Viterby-style exploration from left to right (Punyakanok and Roth, 2001).

Summarizing, the NE-Chunker recognizes the set of NE phrases of a sentence as follows: First, NE candidates are identified in linear time, applying a linear number of decisions. Then, the optimal coherent set of NE phrases is selected in quadratic time, applying a quadratic number of decisions.

## 3 Learning via Recognition Feedback

We now present an online learning strategy for training the learning components of the NE-Chunker, namely the functions $l_B$ and $l_1$ and the functions $\text{score}_k$, for $k \in K$.

Each function is implemented using a perceptron\footnote{Actually, we use a variant of the model called the \textit{voted perceptron}, explained below.} and a representation function.

A perceptron is a linear discriminant function $h_{\hat{w}} : \mathbb{R}^n \rightarrow \mathbb{R}$ parametrized by a weight vector $\hat{w}$ in $\mathbb{R}^n$. Given an instance $\bar{x} \in \mathbb{R}^n$, a perceptron outputs as prediction the inner product between vectors $\bar{x}$ and $\hat{w}$, $h_{\hat{w}}(x) = \hat{w} \cdot \bar{x}$.

The representation function $\Phi : \mathcal{X} \rightarrow \mathbb{R}^n$ codifies an instance $x$ belonging to some space $\mathcal{X}$ into a vector in $\mathbb{R}^n$ with which the perceptron can operate.

The functions $l_B$ and $l_1$ predict whether a word \textit{begins} or is \textit{inside} a NE phrase, respectively. Each one consists of a perceptron weight vector, $\bar{w}_B$ and $\bar{w}_I$, and a shared representation function $\Phi_w$, explained in section 4. Each function is computed as $l_i = \bar{w}_i \cdot \Phi_w(x)$, for $l \in \{B, I\}$, and the sign is taken as the binary classification.

The functions $\text{score}_k$, for $k \in K$, compute a score for a phrase $(s, e)$ being a NE phrase of category $k$. For each function there is a vector $w_k$, and a shared representation function $\Phi_p$, also explained in section 4. The score is given by the expression $\text{score}_k(s, e) = w_k \cdot \Phi_p(s, e)$.

### 3.1 Learning Algorithm

We propose a mistake-driven online learning algorithm for training the parameter vectors $\hat{w}$ of each perceptron all in one go. 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 chunking as specified above using the current vectors. If the predicted chunking is not perfect the vectors which are responsible of the incorrect predictions are updated additively.

The sentence-based learning algorithm is as follows:

- **Input:** $\{(x^1, y^1), \ldots, (x^m, y^m)\}$.
- **Define:** $W = \{ \bar{w}_B, \bar{w}_I \} \cup \{ \bar{w}_k | k \in K \}$.
- **Initialize:** $\forall \bar{w} \in W$ $\bar{w} = 0$;
- for $t = 1 \ldots T$, for $i = 1 \ldots m$:
  1. $\hat{y} = \text{NEch}_W(x^i)$
  2. learning feedback($W, x^i, y^i, \hat{y}$)
- **Output:** the vectors in $W$.

We now describe the learning feedback. Let $y^*$ be the gold set of NE phrases for a sentence $x$, and $\hat{y}$ the set predicted by the NE-Chunker. Let $\text{goldB}(i)$ and $\text{goldI}(i)$ be respectively the perfect indicator functions for the \textit{begin} and \textit{inside} classifications, that is, they return 1 if word $x_i$ begins or is inside some phrase in $y^*$ and 0 otherwise. We differentiate three kinds of phrases in order to give feedback to the functions being learned:

- 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 \textit{begin} word, if misclassified:
     if $(\bar{w}_B \cdot \Phi_w(x_s)) \leq 0$ then $\bar{w}_B = \bar{w}_B + \Phi_w(x_s)$
layers of the recognition process. The missed phrase. Therefore, we move the blocks the generation of the candidate and produces a layer. Thus, misclassifying words of a correct phrase be expressed as the sum of instances \( h_{\bar{w}} \cdot \Phi_w(x_i) \). The final hypothesis is an averaged vote over the decisions the vector \( \bar{w}_k \) generated after every mistake are stored, together with a weight \( c^j \), which corresponds to the number of decisions the vector \( \bar{w}^j \) survives until the next mistake. Let \( J \) be the number of vector that a perceptron accumulates. The final hypothesis is an averaged vote over the predictions of each vector, computed with the expression

\[
h_{\bar{w}}(\bar{x}) = \sum_{j=1}^{J} c^j (\bar{w}^j \cdot \bar{x})
\]

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

\[
h_{\bar{w}}(\bar{x}) = \sum_{j=1}^{J} c^j \sum_{i=1}^{J} s_{x,i} K(\bar{x}^i, \bar{x})
\]

In this paper we work with polynomial kernels \( K(x, x') = (x \cdot x' + 1)^d \), where \( d \) is the degree of the kernel.

### 4 Feature-Vector Representation

In this section we describe the representation functions \( \Phi_w \) and \( \Phi_p \), which respectively map a word or a phrase and their local context into a feature vector in \( \mathbb{R}^n \), particularly, \( \{0, 1\}^n \). First, we define a set of predicates which are computed on words and return one or more values:

- **Form(\( w \))**: The form and PoS of word \( w \).
- **Orthographic(\( w \))**: 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\(^2\)), or pattern (URL).
- **Affixes(\( w \))**: The prefixes and suffixes of the word \( w \) (up to 4 characters).
- **Word Type Patterns(\( w_s \ldots w_e \))**: Type pattern of consecutive words \( w_s \ldots w_e \). The type of a word is either functional (\( \xi \)), capitalized (\( \zeta \)), lowercased (\( \lambda \)), punctuation mark (\( \zeta \)), quote (\( ' \)) or other (\( \kappa \)). For instance, the word type pattern for the phrase “John Smith payed 3 euros” would be \( \text{CClxl} \).

For the function \( \Phi_w(x_i) \) we compute the predicates in a window of words around \( x_i \), that is, words \( x_{i+l} \) with \( l \in [-L_w, +L_w] \). Each predicate label, together with each relative position \( l \) and each returned value forms a final binary indicator feature. The word type patterns are evaluated in all sequences within the window which include the central word \( i \).

For the function \( \Phi_p(s, e) \) we represent the context of the phrase by evaluating a \([-L_p, 0] \) window of predicates at the \( s \) word and a separate \([0, +L_p] \) window at the \( e \) word. At the \( s \) window, we also codefify the named entities already recognized at the left context, capturing their category and relative position. Furthermore, we represent the \( (s, e) \) phrase by evaluating the predicates without capturing the relative position in the features. In particular,

\(^2\)Functional words are determiners and prepositions which typically appear inside NEs.
for the words within \((s, e)\) we evaluate the form, affixes and type patterns of sizes 2, 3 and 4. We also evaluate the complete concatenated form of the phrase and the word type pattern spanning the whole phrase. Finally, we make use of a gazetteer to capture possible NE categories of the whole NE form and each single word within it.

5 Experiments and Results

A list of functional words was automatically extracted from each language training set, selecting those lower-cased words within NEs appearing 3 times or more. For each language, we also constructed a gazetteer with the NEs in the training set. When training, only a random 40% of the entries was considered.

We performed parameter tuning on the English language. Concerning the features, we set the window sizes \((L_w \text{ and } L_p)\) to 3 (we tested 2 and 3), and we did not consider features occurring less than 5 times in the data. When moving to German, we found better to work with lemmas instead of word forms.

Concerning the learning algorithm, we evaluated kernel degrees from 1 to 5. Degrees 2 and 3 performed somewhat better than others, and we chose degree 2. We then ran the algorithm through the English training set for up to five epochs, and through the German training set for up to 3 epochs. 3 On both languages, the performance was still slightly increasing while visiting more training sentences. Unfortunately, we were not able to run the algorithm until performance was stable. Table 1 summarizes the obtained results on all sets. Clearly, the NERC task on English is much easier than on German. Figures indicate that the moderate performance on German is mainly caused by the low recall, specially for ORG and MISC entities. It is interesting to note that while in English the performance is much better on the development set, in German we achieve better results on the test set. This seems to indicate that the difference in performance between development and test sets is due to irregularities in the NEs that appear in each set, rather than overfitting problems of our learning strategy.

The general performance of phrase recognition system we present is fairly good, and we think it is competitive with state-of-the-art named entity extraction systems.

### Acknowledgments

This research has been 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 grant by the Catalan Government Research Department.

|                | English devel. | Precision | Recall | F\(_{β=1}\) |
|----------------|----------------|-----------|--------|------------|
| LOC            | 90.77%         | 93.63%    | 92.18  |
| MISC           | 91.98%         | 80.80%    | 86.03  |
| ORG            | 86.02%         | 83.52%    | 84.75  |
| PER            | 91.37%         | 90.77%    | 91.07  |
| Overall        | 90.06%         | 88.47%    | 89.26  |

|                | English test   | Precision | Recall | F\(_{β=1}\) |
|----------------|----------------|-----------|--------|------------|
| LOC            | 86.66%         | 89.15%    | 87.88  |
| MISC           | 84.90%         | 72.08%    | 77.97  |
| ORG            | 82.73%         | 77.60%    | 80.09  |
| PER            | 88.25%         | 86.39%    | 87.31  |
| Overall        | 85.81%         | 82.84%    | 84.30  |

|                | German devel.  | Precision | Recall | F\(_{β=1}\) |
|----------------|----------------|-----------|--------|------------|
| LOC            | 75.21%         | 67.32%    | 71.05  |
| MISC           | 76.90%         | 42.18%    | 54.48  |
| ORG            | 76.80%         | 47.22%    | 58.48  |
| PER            | 76.87%         | 60.96%    | 67.99  |
| Overall        | 76.36%         | 55.06%    | 63.98  |

|                | German test    | Precision | Recall | F\(_{β=1}\) |
|----------------|----------------|-----------|--------|------------|
| LOC            | 72.89%         | 65.22%    | 68.84  |
| MISC           | 67.14%         | 42.09%    | 51.74  |
| ORG            | 77.67%         | 42.30%    | 54.77  |
| PER            | 87.23%         | 70.88%    | 78.21  |
| Overall        | 77.83%         | 58.02%    | 66.48  |

Table 1: Results obtained for the development and the test data sets for the English and German languages.

### References

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

M. Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments Perceptron Algorithms. In *EMNLP’02*, 3:1025–1058.

K. Crammer and Y. Singer. 2003. A Family of Additive Online Algorithms for Category Ranking. *Journal of Machine Learning Research*, 3:1025–1058.

Y. Freund and R. E. Schapire. 1999. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 37(3):277–296.

V. Punyakanok and D. Roth. 2001. The Use of Classifiers in Sequential Inference. In *Proceedings of the NIPS-13*.

\(^{3}\)Implemented in PERL and run on a Pentium IV (Linux, 2.5GHz, 512Mb) it took about 120 hours for English and 70 hours for German.