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 Politècnica de Catalunya
{carreras,lluism,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 $\mathcal{K} = \{\text{LOC}, \text{PER}, \text{ORG}, \text{MISC}\}$.

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 $ne_1 = (s_1, e_1)_{k_1}$ and $ne_2 = (s_2, e_2)_{k_2}$ overlap, denoted as $ne_1 \sim ne_2$ iff $e_1 \geq s_2 \wedge e_2 \geq s_1$. 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 $y \in \mathcal{Y}$:

$$\text{NEch} : \mathcal{X} \rightarrow \mathcal{Y}$$

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 $\text{h}_\text{B}(w)$ which decides whether a word w *begins* a NE phrase or not, and a function $\text{h}_\text{I}(w)$ which decides whether a word is *inside* a NE phrase or not. Furthermore, we define the predicate BI*, which tests whether a certain phrase is formed by a starting *begin* word and subsequent *inside* words. Formally, $\mathrm{BI}^*((s,e)_k) = (\mathrm{h_B}(s) \ \wedge \ \forall i : s < i \leq e : \mathrm{h_I}(i))$. 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 $\mathcal{Y}_{\mathrm{BI}^*} = \{y \in \mathcal{Y} \mid \forall ne \in y \ \mathrm{BI}^*(ne)\}$.

The second layer selects the best coherent set of NE phrases by applying learning at phrase level. We assume a number of scoring functions, which given a NE 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 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:

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

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 $\mathcal{Y}_{\mathrm{BI}^*}$. 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 $\mathrm{h_B}$ and $\mathrm{h_I}$ and the functions score_k, for $k \in \mathcal{K}$.

Each function is implemented using a perceptron[1] and a representation function.

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

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

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

The functions $\mathrm{h_B}$ and $\mathrm{h_I}$ predict whether a word *begins* or is *inside* a NE phrase, respectively. Each one consists of a perceptron weight vector, $\bar{w}_{\mathrm{B}}$ and $\bar{w}_{\mathrm{I}}$, and a shared representation function Φ_{w}, explained in section 4. Each function is computed as $h_l = \bar{w}_l \cdot \Phi_{\mathrm{w}}(x)$, for $l \in \{\mathrm{B}, \mathrm{I}\}$, 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 NE phrase of category k. For each function there is a vector $\bar{w}_k$, and a shared representation function Φ_{p}, also explained in section 4. The score is given by the expression $\mathrm{score}_k(s,e) = \bar{w}_k \cdot \Phi_{\mathrm{p}}(s,e)$.

3.1 Learning Algorithm

We propose a mistake-driven online learning algorithm for training the parameter vectors $\bar{w}$ of each perceptron all in one go. The algorithm starts with all vectors initialized to $\bar{0}$, 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: $W = \{\bar{w}_{\mathrm{B}}, \bar{w}_{\mathrm{I}}\} \cup \{\bar{w}_k | k \in \mathcal{K}\}$.
- Initialize: $\forall \bar{w} \in W \ \bar{w} = \bar{0}$;
- for $t = 1 \ldots T$, for $i = 1 \ldots m$:
  1. $\hat{y} = \mathrm{NEch}_W(x^i)$
  2. $\mathrm{learning\_feedback}(W, x^i, y^i, \hat{y})$
- 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 $\mathrm{goldB}(i)$ and $\mathrm{goldI}(i)$ be respectively the perfect indicator functions for the *begin* and *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 *begin* word, if misclassified:
     if $(\bar{w}_{\mathrm{B}} \cdot \Phi_{\mathrm{w}}(x_s) \leq 0)$ then
     $\bar{w}_{\mathrm{B}} = \bar{w}_{\mathrm{B}} + \Phi_{\mathrm{w}}(x_s)$

2. Update misclassified *inside* words:
   $\forall i : s < i \leq e$: such that $(\bar{w}_{\mathrm{I}} \cdot \Phi_{\mathrm{w}}(x_i) \leq 0)$
   $\bar{w}_{\mathrm{I}} = \bar{w}_{\mathrm{I}} + \Phi_{\mathrm{w}}(x_i)$
3. Update score function, if it has been applied:
   if $(\bar{w}_{\mathrm{B}} \cdot \Phi_{\mathrm{w}}(x_s) > 0 \ \wedge$
   $\forall i : s < i \leq e : \bar{w}_{\mathrm{I}} \cdot \Phi_{\mathrm{w}}(x_i) > 0)$ then
   $\bar{w}_k = \bar{w}_k + \Phi_{\mathrm{p}}(s, e)$

- Over-predicted phrases: $\forall (s, e)_k \in \hat{y} \setminus y^*$:

1. Update score function:
   $\bar{w}_k = \bar{w}_k - \Phi_{\mathrm{p}}(s, e)$
2. Update *begin* word, if misclassified :
   if $(\mathrm{goldB}(s) = 0)$ then
   $\bar{w}_{\mathrm{B}} = \bar{w}_{\mathrm{B}} - \Phi_{\mathrm{w}}(x_s)$
3. Update misclassified *inside* words :
   $\forall i : s < i \leq e$: such that $(\mathrm{goldI}(i) = 0)$
   $\bar{w}_{\mathrm{I}} = \bar{w}_{\mathrm{I}} - \Phi_{\mathrm{w}}(x_i)$

This feedback models the interaction between the two layers of the recognition process. The *Begin-Inside* identification filters out phrase candidates for the scoring layer. Thus, misclassifying words of a correct phrase blocks the generation of the candidate and produces a missed phrase. Therefore, we move the *begin* or *end* prediction vectors toward the misclassified words of a missed phrase. When an incorrect phrase is predicted, we move away the prediction vectors of the *begin* and *inside* words, provided that they are not in the beginning or inside a phrase in the gold chunking. Note that we deliberately do not care about false positives *begin* or *inside* 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.

3.2 Voted Perceptron and Kernelization

Although the analysis above concerns the perceptron algorithm, we use a modified version, the *voted perceptron* algorithm, introduced in (Freund and Schapire, 1999).

The key point of the voted version is that, while training, it stores information in order to make better predictions on test data. Specifically, all the prediction vectors $\bar{w}^j$ 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:

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

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 Φ_{w} and Φ_{p}, which respectively map a word or a phrase and their local context into a feature vector in $\Re^n$, particularly, $\{0, 1\}^n$. First, we define a set of predicates which are computed on words and return one or more values:

- **Form**(w), **PoS**(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* (`f`), *capitalized* (`C`), *lowercased* (`l`), *punctuation mark* (`.`), *quote* (`'`) or *other* (`x`). For instance, the word type pattern for the phrase "John Smith payed 3 euros" would be `CClxl`.

For the function $\Phi_{\mathrm{w}}(x_i)$ we compute the predicates in a *window* of words around x_i, that is, words x_{i+l} with $l \in [-L_{\mathrm{w}}, +L_{\mathrm{w}}]$. 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 $\Phi_{\mathrm{p}}(s, e)$ we represent the context of the phrase by evaluating a $[-L_{\mathrm{p}}, 0]$ window of predicates at the s word and a separate $[0, +L_{\mathrm{p}}]$ window at the e word. At the s window, we also codify 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} and L_{p}) to 3 (we tested 2 and 3) , and we did not considered 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.

English devel.	Precision	Recall	$F_{\beta=1}$
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	$F_{\beta=1}$
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	$F_{\beta=1}$
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	$F_{\beta=1}$
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.

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.

[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.

References

X. Carreras, L. Màrquez, 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 *Proceedings of the EMNLP'02*.

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*.