The Tanl Named Entity Recognizer at Evalita 2009

Giuseppe Attardi[1], Stefano Dei Rossi[1], Felice Dell'Orletta[2], and Eva Maria Vecchi[2]

[1] Dipartimento di Informatica, Università di Pisa, largo B. Pontecorvo 3, I-56127 Pisa, Italy
[2] ILC-CNR, via G. Moruzzi 1, I-56124 Pisa, Italy
{attardi, deirossi}@di.unipi.it, {felice.dellorletta,evamaria.vecchi}@ilc.cnr.it

Abstract. We describe the tagger present in the Tanl toolkit, which is a flexible and customizable tool for use in various tagging tasks, including POS tagging and SuperSense tagging. The tagger uses a variety of features, both local and global, which can be specified in a configuration file. The tagger is based on a Maximum Entropy classifier and uses dynamic programming to select accurate sequences of tags. We applied it to the NER tagging task in Evalita 2009, customizing the set of features to use and generating a set of dictionaries from the training corpus, that also provide additional features. The final accuracy is further improved by applying simple symbolic rules.

Keywords: natural language processing, named entity tagger, chunker.

1 Architecture

The Tanl [1] tagger is a generic, customizable text chunker, which can be applied to tasks such as POS tagging, Super Sense tagging and Named Entity recognition. The chunker uses a Maximum Entropy classifier for learning how to chunk texts. Maximum Entropy is a more efficient technique than SVM, and by complementing it with dynamic programming it can achieve similar levels of accuracy.

The tagger has an option to transforms the IOB annotations into a more refined set of tags: the B tag is replaced by U when an entity consisting of a single token; the last I tag of an entity of more than one token is replaced by E. Experiments have shown that for NER the refinement is effective, helping the classifier to better separate the data.

The tagger scans the input from left to right and extracts features, representing the current state, that are fed to a Maximum Entropy classifier to learn a model for tagging. Feature extraction is accomplished by an object of class `FeatureExtractor` that can be specialized for the purposes of different chinking tasks. During tagging, the same feature extractor is applied and the classifier computes a probability distribution for the tags to assign to the current token.

Since the Maximum Entropy classifiers assigns tags to each token independently, it may produce inadmissible sequences of tags. Hence a dynamic programming technique is applied to select correct sequences. A probability is assigned to a sequence of tags t_1, t_2, …, t_n for sentence s, based on the probability of transition between two consecutive tags $P(t_{i+1} \mid t_i)$, and the probability of a tag $P(t_i \mid s)$, obtained from the probability distribution computed by Maximum Entropy:

$$P(t_1, t_2, ..., t_n) = \prod_{i=1}^{n} [P(t_i|s)P(\ )]t_i|t_{i-1})$$

In principle the algorithm should compute the sequence with maximum probability. We use instead a dynamic programming solution which operates on a window of size $w = 5$, long enough for most NEs.

For each position n, we compute the best probability $PB(t_n)$ considering the n-grams of length $k < w$ preceding t_n:

$$PB(t_n) = \max_k PB(t_{n-k-1}) \ldots PB(t_{n-1})$$

A baseline is computed first, assuming that the k-gram is made all of 'O' tags:

$$PB_O(t_n) = \max_k PB(t_{n-k-1}) P(t_{n-k} = \text{O}) \ldots P(t_{n-1} = \text{O})$$

Similarly for each class C we compute:

$$PB_C(t_n) = \max_k PB(t_{n-k-1}) P(t_{n-k} = \text{C}) \ldots P(t_{n-1} = \text{C})$$

and finally

$$PB(t_n) = \max(PB_O(t_n), \max_C PB_C(t_n)$$

2 Feature Extractor

The modular architecture of the chunker involves the use of an abstract class called `FeatureExtractor` for extracting features during training and analysis. The class `NerFeatureExtractor` is a specialization of the abstract class designed for Named Entity tagging. It extracts a basic set of features from the current and surrounding tokens. More specific features, to extract for a given task or for a given language, can be specified through a configuration file.

2.1 Feature Specification

There are two mechanisms to specify the additional features to extract: as attributes of the tokens or as token features expressed by a regular expression.

An example of an attribute feature is the following:

```
POSTAG -1 0
```

which requests to use as features the POS tag of the previous (`-1`) and current (`0`) token.

Token features can be expressed with regular expressions, for instance, in:

```
MorphFeature    FORM    ^\p{Lu}         -1 +1
```

```
MorphFeature    FORM    ^\p{Lu}*$       0
```

The first line indicates to use as features the property of starting with an uppercase letter (Unicode property `Lu`) for the previous (`-1`) and next token (`+1`); the second line indicates the feature representing that the current token consists of all upper case letters.

2.2 Dictionaries

Dictionaries are used to group tokens with specific properties. They associate an entity type to tokens. For NER, several dictionaries were created automatically by preprocessing the training data, according to the following criteria:

Dictionary. Consists in all words annotated as entities that appear more than 5 times in the training corpus.
Prefix. Three letter prefixes of entity words whose frequency is > 9 and whose $\chi 2 > 3.84$.
Suffix. Similarly for suffixes.
LastWords. Words occurring as last in a complex entity more than 9 times and whose $\chi 2 > 3.84$.
FirstWords. Similarly for words appearing as first.
LowerIn. Lowercase words occurring inside an entity.
Bigrams. All bigrams that precede an entity and occur more than 5 times, whose probability is > 0.5 and also > the probability of its first word.
Frequent Words. Words that occur more than 5 times in the training corpus.
Designators. Words that precede an entity.

2.3 Standard Features

The standard `NerFeatureExtractor` extracts two types of features: local and global. Local features represent properties of tokens close to the current token.

Global features are properties valid at the document level. For instance, if a word in a document had been previously annotated with a certain tag, then it is likely that other occurrences of the same word should be tagged similarly. Global features represent these properties. They are particularly useful in cases where the word context is ambiguous but the word appeared previously in a simpler context.

2.4 Morphological Features

Local features are extracted from the analysis of the current word and the context in which the word appears. There are two kinds of local features, those extracted from the current word and those extracted from its surrounding words.

2.4.1 Features of Current Word

The following features of the current word are extracted by the `NerFeatureExtractor`:

first word of sentence and capitalized; first word of sentence and not capitalized; two parts joined by a hyphen.

2.4.2 Dictionary Features

The following dictionary features are extracted:

3-letter suffix present in the suffix dictionary; similarly for 3-letter prefix; presence in dictionary LastWords; presence in dictionary FirstWords; not present in the Frequent Words dictionary; lowercase word present in dictionary LowerIn.

2.4.3 Features from Surrounding Words

The following features of the surrounding words are extracted:

both previous, current and following words are capitalized; both current and following words are capitalized; both current and previous words are capitalized; word is in a sequence within quotes; the two previous words are in the bigrams dictionary for a certain type.

2.5 Global Features

The `NerFeatureExtractor` adds a global feature, whenever a previous occurrence of the current word:

was preceded by a word designator; was preceded by a bigram in the bigram dictionary; was present in the dictionary FirstWords; was present in the dictionary of last words; was in capitalized without being at the start of a sentence; was an acronym.

3 Experiments

A remarkable aspect of our NER is that it can do without POS features: the morphological features it computes are sufficient to categorize tokens according to their function in a sentence.

The following features were specified in the configuration file for the NER task:

the previous word is capitalized; the following word is capitalized; the current word is in upper case; the current word is in mixed case; the current word is a single uppercase character; the current word is a uppercase character and a dot; the current word contains digits; the current word is two digits; the current word is four digits; the current word is made of digits and ‘/’; the current word contains $; the current word contains %; the current word contains ‘; the current word is made of digits and dots.

The NER using features similar to these had been tested on the CoNLL 2003 corpus and test set, achieving state of the art scores: 97.85% accuracy, 90.75% precision, 87.97% recall, 89.34 FB1.

3.1 Post Processing Rules

An error analysis of the NER output on a development set, obtained from 10% of the training corpus, revealed many mistakes, particularly for Location entities. To correct the most obvious cases, we introduced a couple of post processing rules. A common case of errors is tagging as Person a street name which includes a proper name, e.g. "via Vittorio Veneto". The tagger annotates "Vittorio Veneto" as a Person. The post processing Rule 1 corrects these mistakes.

Let's call *NP constituent* a word whose POS tag is either an adjective, number, noun, or foreign word. The following post processing rules are applied to the output of the tagger:

Rule 1 If a token is not annotated as an entity, its POS is noun and it is associated to Location in the dictionary FirstWords, the following token is capitalized and annotated as B-LOC, or a capitalized NP constituent or a number followed by a capitalized NP constituent, then annotate the current token as B-LOC, and the following token as I-LOC.

Rule 2 If a token is not annotated as an entity, its POS is number and the next token is annotated as Location, then tag the current token as Location as well.

Applying these rules, the accuracy for Locations improved by over 5 in FB1 and consequently also the overall accuracy of the tagger improved.

4 Results

The official results are quite poor due to a wrong submission. We report in the following Table unofficial results, before and after the application of the above post processing rules.

	Accuracy	*Precision*	*Recall*	*FB1*
Before	96.99%	78.41%	68.22%	72.96
PostProc	97.02%	78.57%	68.45%	73.16

5 Conclusions

The NER tagger that we developed has some interesting features: it does not use POS tagging information nor external resources. In this configuration the NER tagger achieved state of the art accuracy on the official English benchmark from CoNLL 2003.

When applied to the Evalita 2009 data sets, with minor configuration changes for taking into account differences in word order between English and Italian, the

accuracy dropped significantly, despite the fact that both benchmarks have approximately the same size and the tagger extracts a similar number of training features.

A SuperSense Tagger [3] based on the same tagger also achieved excellent accuracy. Further investigation is required to explain such unexpected drop in performance in the Italian NER task.

Acknowledgments. This work has been supported in part by a grant from Fondazione Cassa di Risparmio di Pisa.

References

1. Attardi, G., et al.: Tanl (Text Analytics and Natural Language Processing): Analisi di Testi per il Semantic Web e il Question Answering, http://medialab.di.unipi.it/wiki/SemaWiki
2. Chieu, H.L., Ng, H.T.: Named Entity Recognition with a Maximum Entropy Approach. In: Proceedings of CoNLL-2003, pp. 160--163. Edmonton, Canada (2003)
3. Dei Rossi, S.: SuperSense Tagging. Master Thesis, Dipartimento di Informatica, Università di Pisa (2009)