Rule Based *vs.* Statistical Chunking of `CoNLL` Data Sets

Igor Boehm

Franckstrasse 7c
4020 Linz Austria
`igor@bytelabs.org`

Abstract

One of the most common operations in language processing are segmentation and labelling [7]. Chunking is a popular representative of a segmentation process which aims to segment tagged tokens into meaningful structures.

This paper compares two chunking approaches, namely an approach based on regular expression rules developed by a human, and a machine based chunking approach based on a N-gram statistical tagger. Experimental results show that the performance of the machine based chunker is very similar to the results obtained by the regular expression chunker.

Another interesting fact is that it was considerably harder to define regular expressions which capture noun phrases than verb phrases. Obviously this difficulty was caused by the fact that the structure of noun phrases may be very complex.

1. Introduction

Chunking is the process of annotating tagged tokens with structures in a non-hierarchical and non-recursive way. Since chunkers do not try to analyse complete sentences, but only try to build "chunks" of words, the rule system of chunkers is relatively simple, robust, and efficient.

The aim of this paper is to develop and compare two chunking approaches, namely a rule based method using a `NLTK` lite (Natural Language Toolkit) chunker which is based on regular expressions, and a statistical method using the part-of-speech tagger `TnT` (Trigrams'n'Tags).

Since according to [1] shallow parsing can be treated as POS tagging, it is possible to use the TnT tagger which is based on trigram statistics to identify chunks based on frequencies, yielding results comparable with other, more elaborate approaches.

Chunking, or also referred to as shallow parsing or chunk parsing, has been a relatively active field in the last few years. For example the `CoNLL-2000` (Conference on Computational Natural Language Learning) in Lisbon, Portugal, focused on text chunking, comparing the performance of various chunking approaches.

[2] is another research which looks at statistical recognition of noun phrases with a chunk tagger, showing that a part-of-speech tagger can be (re)used for the task of partial parsing.

Partial parsing has also been utilized in [3] in order to ease identification of clauses, where a clause is a sequence of words in a sentence that contains a subject and a predicate.

Probably one of the most important areas in computational linguistics where shallow parsing is used, would be the process of information extraction. One reason for this is that many information extraction algorithms generally do not extract all the possible information in a text; they simply extract enough to fill out some sort of template of required data [5].

2. Approach

As mentioned in the previous section, a shallow parser based on regular expressions and a N-gram tagger, represented by the TnT tagger are used in order to chunk the data set into appropriate structures.

The main difference between these two approaches is that it is necessary to develop the regular expression parser by defining efficient regular expression rules, whereas the TnT tagger can be easily trained on gold standard chunked data by determining trigram frequencies of chunks. Thus using the TnT tagger for chunking is a very straight forward task.

The `NLTK` lite chunk parser operates on corpora which has been tagged with the appropriate part-of-speech tags and uses regular expressions over sequences of part-of-speech tags in order to identify the extent of text to be chunked.

The quality of the NLTK lite regular expression chunker highly depends on the quality of the regular expressions whereas the quality of the TnT tagger solely depends on the size and quality of the gold standard data.

We are mainly interested in noun and verb phrase chunking. A noun phrase is a phrase whose head is a noun or a pronoun, optionally accompanied by a set of modifiers. These modifiers can be:

- **Determiners:** Articles (*the*, *a*), demonstratives (*this*, *that*), numerals, possessives (*my*, *their*), and quantifiers (*some*, *many*)
- **Adjectives:** the *green* highlands
- **Complements:** in the form of adpositional phrases or relative clauses

A verb phrase consists of a verb, often one or two complements, and any number of adjuncts.

2.1. Training `NLTK` on Chunk Data

Developing regular expression rules for the `NLTK` lite chunker can be conducted without deep knowledge of the grammatical properties of the language. This assumption is based on the results presented in [6] which basically states that "language novices", with almost no training were able to come fairly close to the performance of a machine-learned annotator, learning a small number of powerful rules in a short amount of time on a small training set.

Thus in this experiment the rules have been developed based solely on the information provided by the gold standard data. The following iteration of steps has been repeatedly applied to the training data up until a point where the addition of new rules did not significantly improve the performance of the chunker:

- Start with an empty ruleset.
    1. `Step:` Define or refine a rule.
    2. `Step:` Execute chunker on training data.
    3. `Step:` Compare result with previous run.
- Repeat `Steps` 1 through 3 until the performance does not improve significantly .

This approach works reasonably well, although there is a little drawback which results from the size of the training set. The training data consists of 211.727 phrases which means that the steps required in the process of testing the performance of a new rule, namely chunking and evaluating training data, takes quite a long time.

Thus the application and evaluation of regular expression rules has been conducted on a subset of the training data consisting of 1.000 phrases, because [6] clearly shows that it will not significantly decrease the performance of the resulting regular expression based chunker if it is trained on a small training set.

2.2. Training `TnT` on Chunk Data

Since we are labeling each word with a particular chunk label, we may treat chunking in the same way as tagging and apply a statistical tagger like TnT for this task. The application of TnT consists of two steps [4], namely parameter generation and tagging.

The first step creates model parameters from a training corpus which are applied to new text during step two when the actual tagging is performed.

Tools for creating the necessary model parameters from a training corpus are already implemented in TnT, thus the development cycle of creating a chunker based on TnT is quite short and straight forward.

3. Data Set

The training and test data which is available for this task consists of partitions of the Wall Street Journal (WSJ) corpus. The Wall Street Journal Newspaper is an influential international daily newspaper published in New York, primarily covering U.S. and international business and financial news and issues.

Sections 15-18 of WSJ are used as training data and section 20 as test data. Table 4 displays the amount of tokens present in each data set used for this experiment.

3.1. Important Data Set Aspects

Both the training and test data has already been tagged with the appropriate POS and IOB (Inside Outside Begin) tags. The IOB notation encodes the name of the chunk type, for example I-NP for noun phrase words and I-VP for verb phrase words, and it also encodes if the chunk is a B-chunk for the first word of the chunk, and I-chunk for each other word in the chunk. Every word which is labeled with an "O" is outside of any chunk.

What makes the WSJ corpus interesting is the amount of special and punctuation characters which is used. Special characters like "\$", "λ", "#" etc. are treated just like other part of speech tags, which means that they can appear at the beginning or within a chunk. Punctuation characters are always treated as being outside of chunks and labeled with an "O".

3.2. Data Set Preprocessing

3.2.1. NLTK Backoff Tagger Preprocessing Steps

Because of good support of the `CoNLL` data set in `NLTK` lite, it was not necessary at all to preprocess the data prior to chunking or evaluation.

3.2.2. TnT Tagger Preprocessing Steps

The file format for untagged text files for TnT requires that each token of the text occupies its own line, delimited by a linefeed character. The format of tagged files is similar to that of untagged files. It is extended by adding an additional column per line where the columns are separated by any number of white space characters. Thus the first column represents the tokens and the second column represents the according tag.

Since we want to train TnT to tag words with either NP or VP IOB tags, it was necessary to remove the POS tags from the original `CoNLL` training and test data to produce a gold standard digestible for TnT.

In order to be able to evaluate the chunking results produced by TnT, a `Perl` script provided by `CoNLL` has been used. This script expects its input to contain lines with the following four symbols:

1. The current word.
2. Its part-of-speech tag.
3. The chunk tag according to the corpus.
4. The predicted chunk tag.

Because the result of the TnT tagger does not comply with this format, it had to be merged with the golden standard in order to include the correct information for the evaluation process.

4. Results

4.1. Evaluation Approach

The performance of our chunkers is evaluated in terms of *precision*, *recall* and *F-measure*.

- **Precision:** The percentage of correct guessed chunks:

$$P = \frac{|reference \cap test|}{test}$$

- **Recall:** The percentage of correct chunks were guessed:

$$R = \frac{|reference \cap test|}{reference}$$

- **F-Measure:** The harmonic mean of precision and recall:

$$F_{\alpha=0.5} = \frac{1}{(\alpha/P + (1-\alpha)/R)}$$

- **F-Rate:** According to `CoNNL` evaluator:

$$F = \frac{2 * P * R}{R + P}$$

With these measures we are basically trying to evaluate how well the chunked text matches the golden standard.

	Precision	Recall	F-Measure
NP	79.3%	80.1%	79.7%
VP	76.5%	84.4%	80.3%

Table 1: *NLTK: NP and VP Chunker Performance.*

	Precision	Recall	F-Measure
NP	79.59%	82.35%	80.95%
VP	78.36%	76.76%	77.55%

Table 2: *TnT: NP and VP Chunker Performance.*

4.2. Results in Detail

Running the `NLTK` lite chunker on unchunked corpora and then comparing the output with the golden standard yields the results displayed in table 1.

Table 2 shows the results achieved by the TnT tagger when it was trained on NP chunks, and then on VP chunks. Since the process of training the TnT tagger was very straight forward, a combined NP and VP chunker has also been tested and table 3 yields the results of that experiment.

	Precision	Recall	F-Measure
NP	79.89%	82.87%	81.36%
VP	76.79%	78.97%	77.86%
Overall	79.05%	81.81%	80.41%

Table 3: *TnT: Combined NP/VP Chunker Performance.*

Table 4 displays the total amount of tokens in each data set used for this experiment.

These results clearly show that the performance of the `NLTK` lite chunker is comparable to the results produced by TnT. The `NLTK` lite chunker even outperforms the TnT VP chunker with regards to *recall* and *F-measure*.

5. Conclusions and Future Work

These results back up our initial assumptions that the statistical chunker is comparable with more elaborate approaches in terms of performance as stated in [1]. In our case the more elaborate approach is represented by a set of hand crafted regular expressions which are matched against sequences of POS tags.

We have also backed up the results of [6] by developing regular expression rules based only on their impact on values like recall, precision and F-measure, without looking at the underlying grammatical properties of chunks.

Another interesting aspect is the amount of time and effort necessary to produce a reasonable chunker. Training TnT for the task of chunking was much easier and straight forward than having to develop regular expres-

`CoNLL` Data Set	Amount of Tokens
Training data	211727
Test data	47377

Table 4: *Tokens used in Experiment.*

sion rules.

Also the process of developing and refining regular expression rules for noun phrases was much more time consuming than for verb phrases. Obviously this difficulty was caused by the fact that some noun phrases can have a fairly complex structure, making it hard to capture with regular expressions.

This evaluation could be improved by putting more focus on the underlying grammatical properties of the language in the process of generating regular expression rules for the `NLTK` chunker, in order to investigate if the performance of such rules can significantly outperform a machine based learning approach.

6. Source Code

The *Python* code implemented for this paper can be found in:

```
$/home/s0565052/icl
```

The results of the TnT tagger can be found in:

```
$/home/s0565052/icl/src/tntEVALUATION
```

7. References

[1] Miles, O. “Shallow Parsing as Part-of-Speech Tagging”, Proceedings of CoNLL-2000 and LLL-2000, Lisbon, Portugal, 145–147, 2000.

[2] Wojciech, S. and Thorsten, B. “Chunk Tagger - Statistical Recognition of Noun Phrases”, ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing, Saarbrcken, 1998.

[3] X. Carreras and L. arquez and V. Punyakanok and D. Roth “Learning and Inference for Clause Identification”, Learning and Inference for Clause Identification. In Proceedings of the 14th ECML, Helsinki, Finland, 2002.

[4] Brants, T., “TnT - A Statistical Part-of-Speech Tagger”, In Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000, April 29 – May 3, 2000 Seattle, WA.

[5] Jurafsky, D., Martin, H. M., “Speech and Language Processing”, Prentice-Hall, 2003.

[6] Eric, B., Grace, N., “Man vs. Machine: A Case Study in Base Noun Phrase Learning”, Proceedings of ACL’99, University of Maryland, MD, USA, 1999.

[7] Ewan, Klein, Steven, Bird, Edward, Loper, “NLTK Lite Tutorials: Chunk Parsing”, `http://nltk.sourceforge.net/lite/doc/en/chunk.html`, 16-11-2005.