Chapter 5

Application to shallow parsing

The goal of this chapter is to show that even complex recursive NLP tasks such as parsing (assigning syntactic structure to sentences using a grammar, a lexicon and a search algorithm) can be redefined as a set of cascaded classification problems with separate classifiers for tagging, chunk boundary detection, chunk labeling, relation finding, etc. In such an approach, input vectors represent a focus item and its surrounding context, and output classes represent either a label of the focus (e.g., part of speech tag, constituent label, type of grammatical relation) or a segmentation label (e.g., start or end of a constituent). In this chapter, we show how a shallow parser can be constructed as a cascade of MBLP-classifiers and introduce software that can be used for the development of memory-based taggers and chunkers.

Although in principle full parsing could be achieved in this modular, classification-based way (see section 5.5), this approach is more suited for shallow parsing. Partial or shallow parsing, as opposed to full parsing, recovers only a limited amount of syntactic information from natural language sentences. Especially in applications such as information retrieval, question answering, and information extraction, where large volumes of, often ungrammatical, text have to be analyzed in an efficient and robust way, shallow parsing is useful. For these applications a complete syntactic analysis may provide too much or too little information. For example, in text mining applications such as information extraction, summarization, ontology extraction from text and question answering we are more interested in finding concepts (e.g., simple NPs and VPs) and grammatical relations between their heads (e.g., who did what to whom, when, where, why and how) than in elaborate configurational syntactic
analyses. Shallow parsing is also useful for reducing the search space of full parsers.

Abney (1991) was the first to argue for the relevance of shallow parsing, both from the point of view of psycholinguistic evidence and from the point of view of practical applications. His own approach used hand-crafted cascaded finite-state transducers to construct a shallow parse. Typical modules within a shallow parser architecture include the following:

1. Part-of-speech (POS) tagging. Given a word and its context, decide what the correct morpho-syntactic class of that word is (noun, verb, etc.). POS tagging is a well-understood problem in NLP (Van Halteren, 1999).

2. Chunking. Given the words and their morpho-syntactic class, decide which words can be grouped as chunks (noun phrases, verb phrases, prepositional phrases, complete clauses, etc.) and determine their heads.

3. Relation finding. Given the NP chunks in a sentence, decide which relations their heads have with the main verb (subject, object, location, etc.).

The concept of shallow parsing has no clearly defined meaning however, and is used sometimes in a very limited sense, referring only to tagging and chunking, and sometimes in a broader sense, referring also to tasks such as prepositional phrase assignment (see section 1.2) and named-entity recognition. It can best be interpreted as a family of related tasks attempting to recover some syntactic-semantic information in a robust and deterministic way at the expense of ignoring detailed configurational syntactic information. In this chapter, we restrict its meaning to the three tasks above and demonstrate an MBLP approach to them. The approach is evaluated on the WSJ treebank corpus (Marcus et al., 1993). We also introduce new software, MBTG and MBT, two wrappers around TiMBL that are useful for tagging and chunking.

5.1 Part-of-speech tagging

Part-of-speech (POS) tagging is a process in which a morpho-syntactic class is assigned to each word in a text on the basis of the word’s formal and lexical properties and of the context in which it occurs. It is a first
5.1. PART-OF-SPEECH TAGGING

level of abstraction in text analysis, often used as a preprocessing module in many language technology applications such as parsing, information retrieval, spelling error correction, speech synthesis, and text mining. Just as it is a reliable heuristic in morpho-phonology (see chapter 4) to assume that a spelling symbol will have the same pronunciation or morphological structure decision in similar contexts, the main idea in a memory-based approach to POS tagging is that an ambiguous word will have the same POS tag in similar contexts (Daelemans et al., 1996).

5.1.1 Memory-based tagger architecture

The construction of a POS tagger for a specific corpus is achieved in the following way. Given an annotated corpus, three data structures are automatically extracted: a lexicon, an instance base for known words (words occurring in the lexicon), and an instance base for unknown words. The lexicon associates words with their ambiguous tag, henceforth referred to as ambitag: a symbol representing all the POS tags a word can have according to the corpus. E.g., for a word like executive which occurs in the WSJ corpus both as an adjective (JJ) and as a noun (NN), the corresponding lexical ambitag is NN-JJ (the word occurs more frequently as NN than as JJ, hence the order). A word like current also occurs as both JJ and NN, but less as NN, and will therefore get the ambitag JJ-NN.

During tagging, each word in the text to be tagged is looked up in the lexicon. If it is found, its lexical representation is retrieved and its context in the sentence is determined, and the resulting pattern is disambiguated using extrapolation from nearest neighbors in the known words instance base. When a word is not found in the lexicon, its lexical representation is computed on the basis of its form, its context is determined, and the resulting pattern is disambiguated using extrapolation from nearest neighbors in the unknown words instance base. In each case, the output is a best guess of the POS tag for the word in its current context.

The instances are represented by a variety of features of the focus word to be tagged and word forms in its immediate context. The reason for separating known and unknown words is the following: for known words the ambitag of the focus word turns out to be the most important feature. However, for unknown words we do not know the ambitag, and therefore we are restricted to context and word form features to construct the unknown word’s instance representation. Below we will use the following notation for the features. Since we go from left to right, we can assume that the words to the left of the word to be tagged have been
disambiguated already. These tags are denoted by $D$, the position of the (ambitag) of the focus word is given by $F$, and the not yet disambiguated words to the right are denoted by their ambitag $A$. In both known and unknown word tagging an important source of information is the inclusion of previous tagger decisions as features for current tagger decisions with the $D$ features in both known and unknown word tagging. These features allow the approach to escape from the local windowing limitations. Other solutions to sequence learning problems are introduced in chapter 7.

Features referring to particular word forms are denoted as $W$ or $W$ for the word corresponding to the focus position. As the presence of features with thousands of words as feature values would make the tagging considerably slower and low frequency word values would not be likely to match anyway, only the most frequent words (e.g., the 100 most frequent words) are kept as values, and the others are substituted by the symbol ‘HAPAX’ annotated with some additional information. E.g., HAPAX-N means that the word contains numeric symbols, HAPAX-C means the word is capitalized, HAPAX-H that it is hyphenated, and HAPAX-0 means no special attributes are associated with the word. A word such as B-52 would then get the value HAPAX-CHN. This process is known as attenuation (Eisner, 1996).

Returning to the level of features rather than values, especially for the unknown word instances there are a number of features referring to the parts of the word form: its suffix letters ‘$S$’, prefix letters ‘$P$’, a capitalization feature ‘$C$’, the presence of a hyphen ‘$H$’, and the presence of numerals ‘$N$’. These features provide a kind of “poor man’s morphology” that may be useful to guess the POS tag of an unknown word.

Tables 5.1 and 5.2 display example instances from the known words and the unknown words instance bases (on WSJ material) respectively. For the selection of instances for the unknown words case base, only words are selected that occur with relatively low frequency, as these words will have characteristics more similar to unknown words than frequent words.

5.1.2 Results

In previous work on an MBLP approach to tagging (Daelemans & Van den Bosch, 1996; Zavrel & Daelemans, 1997; Van Halteren et al., 2001) for different corpora and different languages, the approach consistently outperforms the well-known transformation-based learning approach (Brill, 1994) and some trigram-based approaches, but often achieves slightly worse results than maximum-entropy approaches (Ratnaparkhi, 1996) and
### Instance representation

<table>
<thead>
<tr>
<th>Word</th>
<th>D</th>
<th>W</th>
<th>D</th>
<th>W</th>
<th>F</th>
<th>W</th>
<th>A</th>
<th>W</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumers</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>NNS</td>
<td>HAPAX-C</td>
<td>MD</td>
<td>HAPAX-0</td>
<td>NNS</td>
</tr>
<tr>
<td>may</td>
<td>--</td>
<td>--</td>
<td>NNS</td>
<td>HAPAX-C</td>
<td>MD</td>
<td>HAPAX-0</td>
<td>VBP-VB</td>
<td>HAPAX-0</td>
<td>MD</td>
</tr>
<tr>
<td>want</td>
<td>NNS</td>
<td>HAPAX-C</td>
<td>MD</td>
<td>HAPAX-0</td>
<td>VBP-VB</td>
<td>HAPAX-0</td>
<td>TO</td>
<td>TO</td>
<td>VB</td>
</tr>
<tr>
<td>to</td>
<td>MD</td>
<td>HAPAX-0</td>
<td>VB</td>
<td>HAPAX-0</td>
<td>TO</td>
<td>to</td>
<td>NN-VB-VBP</td>
<td>HAPAX-0</td>
<td>TO</td>
</tr>
<tr>
<td>move</td>
<td>VB</td>
<td>HAPAX-0</td>
<td>TO</td>
<td>to</td>
<td>NN-VB-VBP</td>
<td>HAPAX-0</td>
<td>PRPS</td>
<td>their</td>
<td>VB</td>
</tr>
<tr>
<td>their</td>
<td>TO</td>
<td>to</td>
<td>VB</td>
<td>HAPAX-0</td>
<td>PRPS</td>
<td>their</td>
<td>NNS</td>
<td>HAPAX-0</td>
<td>PRPS</td>
</tr>
<tr>
<td>telephones</td>
<td>VB</td>
<td>HAPAX-0</td>
<td>PRPS</td>
<td>their</td>
<td>NNS</td>
<td>HAPAX-0</td>
<td>DT</td>
<td>a</td>
<td>NNS</td>
</tr>
<tr>
<td>a</td>
<td>PRPS</td>
<td>their</td>
<td>NNS</td>
<td>HAPAX-0</td>
<td>DT</td>
<td>a</td>
<td>JJ-RB</td>
<td>HAPAX-0</td>
<td>DT</td>
</tr>
<tr>
<td>little</td>
<td>NNS</td>
<td>HAPAX-0</td>
<td>DT</td>
<td>a</td>
<td>JJ-RB</td>
<td>HAPAX-0</td>
<td>RBR</td>
<td>HAPAX-0</td>
<td>RB</td>
</tr>
<tr>
<td>closer</td>
<td>DT</td>
<td>a</td>
<td>RB</td>
<td>HAPAX-0</td>
<td>RBR</td>
<td>HAPAX-0</td>
<td>TO</td>
<td>to</td>
<td>RBR</td>
</tr>
<tr>
<td>to</td>
<td>RB</td>
<td>HAPAX-0</td>
<td>RBR</td>
<td>HAPAX-0</td>
<td>TO</td>
<td>to</td>
<td>DT</td>
<td>the</td>
<td>TO</td>
</tr>
<tr>
<td>the</td>
<td>RBR</td>
<td>HAPAX-0</td>
<td>TO</td>
<td>to</td>
<td>DT</td>
<td>the</td>
<td>NN-NNP</td>
<td>HAPAX-C</td>
<td>DT</td>
</tr>
<tr>
<td>TV</td>
<td>TO</td>
<td>to</td>
<td>DT</td>
<td>the</td>
<td>NN-NNP</td>
<td>HAPAX-C</td>
<td>VBN-VB-HAPAX-0</td>
<td>NN</td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>DT</td>
<td>the</td>
<td>NN</td>
<td>HAPAX-C</td>
<td>VBN-VB-HAPAX-0</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>NN</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>

Table 5.1: Example of instances of the POS tagging task (known words instances). Instances represent fixed-sized snapshots of a focus (an ambitag), surrounded by a left and right context (of disambiguated tags on the left, and ambiguous tags on the right, and highly-frequent words or else attenuated symbols (HAPAX) to the left and to the right).

An HMM-approach with powerful smoothing such as TnT (Brants, 2000). The memory-based approach yields 96.4% accuracy on the WSJ corpus and 97% on the LOB corpus (Zavrel & Daelemans, 1999). Comparison is difficult because of the different data sets, features, and representations used in different learning approaches. Given the minimal language engineering involved in MBLP (making a tagger on the basis of a new annotated corpus is a matter of hours) and the computational efficiency of the method both in training and testing (in the order of thousands of words per second), this state-of-the-art performance is remarkable. In contrast to explicitly probabilistic methods, there is no need in an MBLP
Table 5.2: Example of instances of the POS tagging task (unknown words instance base). Instances represent ‘morphological’ information about the focus word (first letter and the three last letters), surrounded by a left and right context (of one disambiguated tag to the left, one ambiguous tag to the right, and the corresponding attenuated words).

approach for an additional smoothing component for sparse data, as this is already embodied in the similarity-based extrapolation itself (Zavrel & Daelemans, 1997). The use of the weighted similarity metric allows for an easy integration of different information sources (e.g., context tags, words, morphology, spelling, etc.) with no clear a-priori back-off ordering. Moreover, the fact that only one parameter is needed per feature (i.e., its information-theoretic weight) makes MBLP more robust to overfitting than approaches that use very large numbers of parameters. The downside of this robustness is that the feature-weighting capabilities are quite rough: each feature is weighted in isolation, so that no specific weights are assigned to interesting feature combinations, and the weight estimate of conjunctions of redundant features tends to be too large, and there is also no separate weight for specific values of a feature.

5.1.3 Memory-based tagging with MBT and MBTG

MBTG and MBT are two programs built around TIMBL that allow you to construct a tagger on the basis of a tagged corpus (MBTG) and use this tagger to tag new text (MBT). Although in principle it would be possible to use TIMBL directly, the software provides an easy solution to building the different instance bases with relevant word and context features, including preceding tagger decisions. As an example dataset we derived POS information from the CoNLL shared task data (Tjong Kim Sang & Buchholz, 2000), which is a part of the WSJ corpus. See also chapter 6 for a description of these data.
The input file containing the material for generating a tagger must consist of two whitespace-separated columns. The first column contains a word or punctuation mark, as well as its POS tag in the corresponding position of the second column. A line may also contain only the symbol `<utt>` to mark the end of a sentence. The following is example input.

```plaintext
He PRP
reckons VBZ
the DT
current JJ
account NN
deficit NN
will MD
narrow VB
to TO
only RB
# #
1.8 CD
billion CD
in IN
September NNP
```

In generating the tagger, information has to be provided to the tagger generator about the context and the form of the words to be tagged. This is done by the parameters `-p` (feature pattern for known words), and `-P` (feature pattern for unknown words). Patterns are built up as combinations of the symbols introduced earlier:

**For -p and -P**

- d Predicted left context (tag)
- a Right context (ambitag)
- w Left or right context (word)
- c The focus contains capitalized characters
- h The focus word contains a hyphen
- n The focus word contains numerical characters
- p Character at the start of the word
- s Character at the end of the word

**For -p only (known words)**

- f Focus (ambitag for known words)
- W Focus (word)
For -P only (unknown words)

F Focus (position of the unknown word)
The symbols d, a, w, p, and s can occur more than once to indicate the scope of the context. Symbols to the left of the focus symbols indicate left context, and symbols to the right of the focus symbols indicate right context.

For example, for known words, the following are a few possible patterns:

dfa focus ambitag with one disambiguated tag on the left and one ambitag to the right
ddfa focus ambitag with two disambiguated tags to the left and one ambitag to the right
ddfWa as previous, plus the focus word (note that w can be declared only immediately after f)
dwdfWaw as previous, plus for each context tag the corresponding word (two left, one right)

For unknown words:

dFa one disambiguated tag to the left and one ambitag to the right
psdFa as previous, plus the first and last letter of the unknown word to be tagged
psssdFa as previous, plus the three last letters of the word to be tagged
psssdwFaw as previous, plus the left and right neighboring words

In addition to constructors for these commonly used features, the MBTG software also allows you to add your own additional features to the instances created using the option -E in combination with adding extra columns to the input file, where each column corresponds to an additional feature associated with the word at that position when used as a focus. An example command line using a file T.train with the POS tagging training data of the CoNLL shared data for tagger generation and corresponding output is the following ("%" is the command line prompt):
% Mbtg -p dwdwfWaw -P dwFawpsschm -T T.train

Memory Based Tagger Generator Version 2.0
(c) ILK and CNTS 1998 - 2004.
Induction of Linguistic Knowledge Research Group, Tilburg University
Centre for Dutch Language and Speech, University of Antwerp

Based on Timbl version 5.1.0 (Release)

Constructing a tagger from: T.train
Creating lexicon: T.train.lex of 19122 entries.
Creating ambitag lexicon: T.train.lex.ambi.05
Creating ambitag translation table: T.train.ambi.05
Creating list of most frequent words: T.train.top100

Create known words case base
Timbl options: ' -a IGTREE +vS -H'
Algorithm = IGTREE
Processing data from the file
T.train.......................................................
ready: 211727 words processed.
Creating case base: T.train.known.dwdwfWaw
Deleted intermediate file: T.train.known.inst.dwdwfWaw

Create unknown words case base
Timbl options: ' -a IB1 +vS -H'
Algorithm = IB1
Processing data from the file
T.train.......................................................
ready: 211727 words processed.
Creating case base: T.train.unknown.pssschndwFaw
Deleted intermediate file: T.train.unknown.inst.pssschndwFaw

Created settings file ‘T.train.settings’

Ready:
Time used: 50
Words/sec: 4234

The output shows which version of TiMBL was used and reports on the generation of a number of data files that will be used by the tagger MBT. These data include the following:

- A frequency-sorted lexicon `T.train.lex` containing for each word the different tags it was assigned, along with their frequency in the training corpus.
- A lexicon associating with each word an ambitag, derived from the previous lexicon file, and a translation table for associating the generated ambitag letter codes with a more understandable representation. Limited frequency-based smoothing is implemented in this approach: whenever a word–tag combination occurs less than a given percentage (5% by default) of the word’s total frequency, it is not included in the ambitag. The parameter `-% <percentage>` modifies this threshold.
• A list with the (by default) 100 most frequent words in the corpus. Only words in this list will be used when the symbols \( w, W \) are used in the \(-p, -P\) patterns. The number of most frequent words can be modified with the parameter \( -M < \text{number} > \). All words not in the most-frequent-words list are transformed into the special HAPAX-symbols discussed earlier.

• An instance base for known words. The process consists of two steps. First, instances are created using the specified information sources for known words (as indicated in \(-p\)), then the case base is generated from that (which may imply a significant storage reduction, depending on the TiMBL options used, in this case IGTREE). Finally, the intermediate file with instances is deleted — this can be overruled with the option \(-X\).

• An instance base for unknown words. It is parallel to the procedure for known words, but it uses information sources specified in the \(-P\) pattern, and uses as default TiMBL settings the IB1-IG algorithm (which uses the overlap metric with gain ratio feature weighting).

• The tagger generation process ends with some information about the real time needed to construct the tagger (total time used and number of words per second), and with the construction of a settings file, which will be used by the MBT executable to use the tagger on new data.

The settings for our training data are the following:

\[
\text{e <utt>}
\text{1 T.train.lex.ambi.05}
\text{k T.train.known.ddfa}
\text{u T.train.unknown.dFapsss}
\text{r T.train.ambi.05}
\text{p ddfa}
\text{P dFapsss}
\text{O N: -a IGTREE +vS U: -a IB1 +vS}
\text{L T.train.top100}
\]

Given that Mbtg was used to generate data files and a settings file defining a memory-based tagger, MBT can be used to tag text. For example, continuing our example:
Calling MBT with the settings file of the trained tagger starts the memory-based tagger by reading the data files and a test input file (in this case in the same format as the training data), and sends the tagged input to standard output computing accuracy statistics by comparing the predicted tags to the gold standard ones provided in the test file. The tagger can also read untagged text from input or from a text file. The text should then be tokenized (i.e., punctuation marks should be separated from the words). In the output, word and predicted tag are separated by a single slash (known word) or a double slash (unknown word).

More parameters are available to modify the behavior of the MBTG and MBT executables and to use the software in client-server mode, for those we refer to the reference guide accompanying the software (Daelemans et al., 2003).
5.2 Constituent chunking

As soon as sentences have been disambiguated at the word level concerning their morpho-syntactic category, a next step in shallow parsing will group words into phrases and assign a label to these phrases. If we restrict chunking to finding non-overlapping and non-recursive base chunks, the task can be defined as a classification task by generalizing the approach of Ramshaw and Marcus (1995), who proposed to convert NP-chunking to tagging each word with I for a word inside an NP, O for outside an NP, and B for the start of an NP that is preceded by another NP. The decision on these so-called IOB tags for a word can be made by looking at the POS tag and the identity of the focus word and its local context. For the more general task of chunking other non-recursive phrases, we simply extended the tag set with IOB tags for each type of phrase. To illustrate this encoding with the extended IOB tag set, we can represent the following tagged and chunked sentence:

\[ \text{But} / CC \ [ \text{NP the} / DT \text{dollar} / NN \text{NP}] \ [ \text{ADVP later} / RB \text{ADVP}] \ [ \text{VP rebounded} / VBD \text{VP}] \ , / \ [ \text{VP finishing} / VBG \text{VP}] \ [ \text{ADJP slightly} / RB \text{ADV P}] \ [ \text{Prep against} / IN \text{Prep}] \ [ \text{NP the} / DT \text{yen} / NNS \text{NP}] \ [ \text{ADJP although} / IN \text{ADJP}] \ [ \text{ADJP slightly} / RB \text{lower} / JJR \text{ADJP}] \ [ \text{Prep against} / IN \text{Prep}] \ [ \text{NP the} / DT \text{mark} / NN \text{NP}] . / \]

This representation can then be used to generate instances using a moving window approach exactly the same way as is done in POS tagging.

5.2.1 Results

Table 5.3 (from Buchholz et al., 1999) shows the accuracy of this memory-based chunking approach when training and testing on Wall Street Journal material. We report on precision, recall, and F-scores (with $\beta = 1$). In this case, the features for the MBLP-classifier are the word form and the POS tag as provided by the tagger of the two words to the left, the focus word, and one word to the right (Veenstra, 1998; Tjong Kim Sang & Veenstra, 1999). Adverbial functions are included here as chunking results as well: this classifier assigns adverbial functions such as locative or temporal to the chunks.
### 5.2. CONSTITUENT CHUNKING

<table>
<thead>
<tr>
<th>Type</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPchunks</td>
<td>92.5</td>
<td>92.2</td>
<td>92.3</td>
</tr>
<tr>
<td>VPchunks</td>
<td>91.9</td>
<td>91.7</td>
<td>91.8</td>
</tr>
<tr>
<td>ADJPchunks</td>
<td>68.4</td>
<td>65.0</td>
<td>66.7</td>
</tr>
<tr>
<td>ADVPchunks</td>
<td>78.0</td>
<td>77.9</td>
<td>77.9</td>
</tr>
<tr>
<td>PPchunks</td>
<td>91.9</td>
<td>92.2</td>
<td>92.0</td>
</tr>
<tr>
<td>ADVFUNCs</td>
<td>78.0</td>
<td>69.5</td>
<td>73.5</td>
</tr>
</tbody>
</table>

Table 5.3: Results of chunking (in %): segmentation and labeling experiments. Reproduced from (Buchholz et al., 1999).

#### 5.2.2 Using MBT and MBTG for chunking

Although we could easily use TIMBL for chunking by combining words and output of the POS tagger to construct the features of the instances, as was done in the work reported earlier in this chapter, an alternative approach is to use the convenience of MBTG and MBT to construct a combined tagger-chunker. This can be achieved by concatenating the POS-tag and the IOB-tag associated with each word in the sentence, and using MBTG to construct a tagger for these combined tags. The following is an example of the type of input file needed for this.

```
He PRP/B-NP
reckons VBZ/B-VP
the DT/B-NP
current JJ/I-NP
account NN/I-NP
deficit NN/I-NP
will MD/B-VP
narrow VB/I-VP
to TO/B-PP
only RB/B-NP
# #/I-NP
1.8 CD/I-NP
billion CD/I-NP
in IN/B-PP
September NNP/B-NP
. ./O
<utt>
```
As POS tagging and chunking are very much related, it makes sense to combine these two learning tasks into one. Interestingly, although the learning task becomes more complex (more classes have to be learned that describe a more complex output space) and data sparseness therefore increases, the results on tagging and chunking separately do not degrade. The following shows the output and the results of the tagger generation and tagging phases using this combined approach (without any specific optimization for this task).

```
% Mbtg -T TC.train -p dwdwfWaw -P psschndwFaw
... Created settings file 'TC.train.settings'

Ready:
  Time used: 55
  Words/sec: 3849

% Mbt -s TC.train.settings -T TC.test

Memory Based Tagger Version 2.0
(c) ILK and CNTS 1998 - 2004.
Induction of Linguistic Knowledge Research Group, Tilburg University
Centre for Dutch Language and Speech, University of Antwerp

Based on Timbl version 5.1.0 (release)

Reading the ambitags from: TC.train.ambi.05...ready, (1468 tags).
  Reading frequent words list from: TC.train.top100...ready, (100 words).
Reading case-base for known words from:
  TC.train.known.dwdwfWaw... ready.
Reading case-base for unknown words from:
  TC.train.unknown.psschndwFaw... ready.
Sentence delimiter set to '<utt>'
  Beam size = 1
  Known Tree, Algorithm = IGTREE
  Unknown Tree, Algorithm = IB1

Processing data from the file TC.test: ..................

Rockwell // NNP/B-NP NNP/B-NP
International / NNP/I-NP NNP/I-NP
Corp. / NNP/I-NP NNP/I-NP
's / POS/B-NP POS/B-NP
Tulsa / NNP/I-NP NNP/I-NP
unit / NN/I-NP NN/I-NP
said / VBD/B-VP VBD/B-VP
it / PRP/B-NP PRP/B-NP
signed / VBD/B-VP VBD/B-VP

Done:
  47377 words processed.
  Known words: 39226 correct from 44075 (88.9983 %)
  Unknown words: 2443 correct from 3302 (73.9855 %)
  Total : 41669 correct from 47377 (87.952 %)
  Time used: 49
  Words/sec: 966
```
5.3. RELATION FINDING

Tagging accuracy drops only slightly to 94.5% (from 94.6%) compared to the tagger trained uniquely on POS tagging information. The precision, recall and F-score for chunking over all constituent types are 83.5%, 87.2%, and 85.3%, respectively. In the following chapters we will return to this sequence learning task with approaches providing better results.

5.3 Relation finding

After POS tagging, phrase chunking and labeling, the last step of shallow parsing consists of resolving the attachment between labeled phrases. Work on an MBLP approach to relation finding evolved from work on complement-adjunct distinction (Buchholz, 1998) to subject and object detection (Daelemans et al., 1999), and, finally, relation finding for all relations annotated in the WSJ corpus (Buchholz et al., 1999; Buchholz, 2002).

In this approach, relation finding is done by using a classifier to assign a grammatical relation (GR) between pairs of words in a sentence. One of these words is always a verb, since this yields the most important GRs. The other word (the focus) is the head of a phrase that can be assigned a grammatical relation (e.g., a noun as head of an NP). The class to be predicted is the grammatical relation holding between this phrase and the verb.

5.3.1 Relation finder architecture

An instance for such a pair of words is constructed by extracting a set of feature values from the sentence. The instance contains information about the verb and the focus: a feature for the word form and a feature for the POS of both. It also has similar features for the local context of the focus. Experiments on the training data suggest an optimal context width of two words to the left and one to the right, as was the case for chunking. In addition to the lexical and the local context information, superficial information about clause structure was included as well: the distance from the verb to the focus, counted in numbers of words. A negative distance means that the focus is to the left of the verb. Other features contain the number of other verbs between the verb and the focus, and the number of intervening commas. These features were chosen by manual “feature
Table 5.4: The first five instances for the example sentence. Features 1–3 are the features for distance and intervening VPs and commas. Features 4 and 5 show the verb and its POS. Features 6–8, 9–11 and 17–19 describe the context words/chunks, features 12–16 the focus chunk. Empty contexts are indicated by the “-” for all features. Some words are abbreviated.

engineering” (Buchholz, 2002). Table 5.4 shows some of the instances corresponding to the following sentence (POS tags after the slash, chunks denoted with square and curly brackets, and adverbial functions after the dash). All this information is provided by tagging and chunking:


5.3.2 Results

Table 5.5 shows the results of the experiments. In the first row, only POS tag features are used. Other rows show the results when adding several types of chunk information as extra features. The more structure is added, the better the results: precision increases from 60.7% to 74.8%, recall from 41.3% to 67.9% — in spite of the fact that the added information is not always correct, because it was predicted by other modules of the shallow parser. With “perfect” information from these modules, a precision of 86.3% and recall of 80.8% would be attainable.

In Buchholz (2002), the MBLP approach to GR finding described here was further investigated and optimized both for accuracy and for efficiency.
5.4. CONCLUSION

Table 5.5: Results (in %) of grammatical relation assignment with increasing levels of structure in the test data added by earlier modules in the cascade. Columns show the number of features in the instances, the average distance between the verb and the focus element, precision, recall and F-score (with $\beta = 1$) over all relations, and F-score over four selected relations. Reproduced from (Buchholz et al., 1999).

<table>
<thead>
<tr>
<th>Structure in input</th>
<th>Feat.</th>
<th>$\Delta$</th>
<th>Prec.</th>
<th>Recall</th>
<th>All F</th>
<th>Subj F</th>
<th>Obj F</th>
<th>Loc F</th>
<th>Temp F</th>
</tr>
</thead>
<tbody>
<tr>
<td>words and POS only</td>
<td>13</td>
<td>6.1</td>
<td>60.7</td>
<td>41.3</td>
<td>49.1</td>
<td>52.8</td>
<td>49.4</td>
<td>34.0</td>
<td>38.4</td>
</tr>
<tr>
<td>+VP chunks</td>
<td>17</td>
<td>6.6</td>
<td>63.8</td>
<td>47.9</td>
<td>54.7</td>
<td>62.9</td>
<td>51.5</td>
<td>39.0</td>
<td>42.8</td>
</tr>
<tr>
<td>+NP chunks</td>
<td>17</td>
<td>4.2</td>
<td>65.9</td>
<td>55.7</td>
<td>60.4</td>
<td>64.1</td>
<td>75.6</td>
<td>37.9</td>
<td>42.1</td>
</tr>
<tr>
<td>+VP chunks</td>
<td>17</td>
<td>4.5</td>
<td>72.1</td>
<td>62.9</td>
<td>67.2</td>
<td>78.6</td>
<td>75.6</td>
<td>40.8</td>
<td>46.8</td>
</tr>
<tr>
<td>+ADVP/ADJP chunks</td>
<td>17</td>
<td>4.4</td>
<td>72.1</td>
<td>63.0</td>
<td>67.3</td>
<td>78.8</td>
<td>75.8</td>
<td>40.4</td>
<td>46.5</td>
</tr>
<tr>
<td>+Prep chunks</td>
<td>17</td>
<td>4.4</td>
<td>72.5</td>
<td>64.3</td>
<td>68.2</td>
<td>81.2</td>
<td>75.7</td>
<td>40.4</td>
<td>47.1</td>
</tr>
<tr>
<td>+PP chunks</td>
<td>18</td>
<td>3.6</td>
<td>73.6</td>
<td>65.6</td>
<td>69.3</td>
<td>81.6</td>
<td>80.3</td>
<td>40.6</td>
<td>48.3</td>
</tr>
<tr>
<td>+ADVFUNCs</td>
<td>19</td>
<td>3.6</td>
<td>74.8</td>
<td>67.9</td>
<td>71.2</td>
<td>81.8</td>
<td>81.0</td>
<td>46.9</td>
<td>63.3</td>
</tr>
</tbody>
</table>

by careful feature engineering, system design adaptation, and algorithm parameter optimization, increasing precision and recall to 80% and 86.5%, respectively. Finally, in a surprising learning curve result, Van den Bosch and Buchholz (2002) show that when sufficient training data is available, words only can be used successfully as features to predict constituent structure and grammatical relations, obviating the need for POS tags.

5.4 Conclusion

From the point of view of text mining applications, robust shallow parsing seems at present to yield more useful results than deep parsing, by providing for each sentence what the main constituents and the grammatical relations between them are. In this chapter, we showed that an MBLP approach to shallow parsing is feasible by dividing the problem into a number of subproblems (tagging, chunking, and relation finding), each of which can be handled by a memory-based classifier.

This combination of memory-based classifiers, especially when provided in a server-client set-up, can be extended with domain-specific tokenizers and named-entity recognizers to provide a flexible shallow understanding architecture for use in text mining applications. In the current version of our memory-based shallow parser, the first paragraph
of this book receives the following analysis:

[NP-SBJ-1 This/DT book/NN NP-SBJ-1] [VP-1 presents/VBZ VP-1] [NP-OBJ-1 an/DT simple/JJ and/CC efficient/JJ approach/NN NP-OBJ-1] [PP to/TO PP] [VP-2 solving/VBG VP-2] [NP-OBJ-2 Natural/NNP Language/NNP Processing/NNP problems/NNS NP-OBJ-2].

This chapter finishes our selection of illustrations of the MBLP approach to NLP tasks, started in the previous chapter. Our main goal was to provide a few salient examples showing how to make NLP problems fit the memory-based approach. In the next chapter, we return to a machine learning perspective and discuss the eager-lazy learning dimension. We show empirically that highest accuracy can be achieved in a lazy learning approach like MBLP.

5.5 Further reading

A lot of work since Ramshaw and Marcus (1995) has focused on machine learning approaches to shallow parsing. A good place to start is the papers and references in the special issue of the Journal of Machine Learning Research on this topic (Hammerton et al., 2002). In the context of the CoNLL shared tasks, training and test data for chunking and clause boundary detection is available, and many machine learning results on these data can be accessed through the SIGNLL web site.

The memory-based shallow parsing approach described in this chapter has been used successfully in question answering (Buchholz & Daelemans, 2001), and is currently being adapted and applied in projects on information extraction from biomedical text, automatic subtitling by summarization (Daelemans et al., 2004a), ontology extraction from text (Reinberger...)

---

SIGNLL is the Association for Computational Linguistics’ special interest group on machine learning of language, http://www.aclweb.org/signll/
et al., 2004), spoken language parsing (Canisius & Van den Bosch, 2004), and other applications. An area of current research is to integrate an MBLP module for PP-attachment as an additional component in the shallow parser. Earlier work has shown that this task in isolation is indeed feasible in a memory-based approach (Zavrel & Daelemans, 1997; Van Herwijnen et al., 2004; Kokkinakis, 2000) and could be integrated into the current architecture in a way similar to the integration of grammatical relation finding.

An alternative approach to memory-based chunking and relation finding has been proposed in Argamon et al. (1999) under the name of memory-based sequence learning. The method is based on a search among possible bracketings of sentences, keeping all training data available. The approach can be seen as a linear algorithmic simplification of the DOP memory-based approach to full parsing discussed in chapter 2.

Moving from shallow parsing to full parsing by extending the memory-based chunking approach iteratively to approximate full parsing has not been entirely successful yet (see for example Tjong Kim Sang, 2002 for an empirical investigation). In contrast, the OCTOPUS parser for Chinese (Streiter, 2001b; Streiter, 2001a) uses complete parse trees as memory instances, and retrieves nearest neighbors by matching sequences of keywords, where processes of alignment, nearest neighbor adaptation, and chunking cooperate in providing a parse tree for the input sentences. Another approach to memory-based learning with complete parse trees as “classes” is TüSBL (Kübler, 2004). In this system conventional tagging and chunking are used to provide features for a new similarity metric on a dynamically computed set of features working on complete parse trees as examples.

An alternative way to full parsing defined as a classification-based approach is the framework of shift-reduce parsing, where the next parser decision is predicted given the current state of the parse and the local context as instances. The different steps of the derivations of a parse are used as training instances. A memory-based approach was explored in Veenstra and Daelemans (2000) and recently developed in the context of Swedish and English dependency parsing with the MALT parser (Nivre et al., 2004; Nivre & Scholz, 2004). In combination with alternative parsing methods, memory-based learning has also been found useful for tasks such as the enrichment of parser output (Jijkoun & de Rijke, 2004).