CHUNKING: SHALLOW PARSING

Stephen.Pulman@comlab.ox.ac.uk

- Parsing vs. chunking
- Using HMMs for chunking
- An alternative approach: transformation based learning
- Another alternative: maximum entropy modelling

References

Many papers, including Brill's, Ratnaparkhi's, and Ramshaw and Marcus (below), can be found online at:

`http://aclweb.org/anthology-new`

This contains downloadable papers from all the major computational linguistics conferences, and from the main journal, Computational Linguistics.

There is a very full list of references, links to training and test data, and to online demos at:

`http://staff.science.uva.nl/~erikt/research/np-chunking.html`

Allen, Natural Language Understanding, Chapter 7; Jurafsky and Martin, Section 8.6. (Edition 1), Chapter 5 (Edition 2)

Antonio Molina and Ferran Pla, 2002, ‘Shallow Parsing using Specialized HMMs’ Journal of Machine Learning Research, Vol 2 pp 595-613
`http://www.jmlr.org/papers/v2/molina02a.html`

Brill, E. 1992 A simple rule-based part of speech tagger, in Proceedings of the 3rd ACL Conference on Applied Natural Language Processing, Trento Italy, 152-155.

Ramshaw L.A. and Marcus M.P. ”Transformation-based Learning”. Proceeding of 3th ACL Workshop on Very Large Corpora at ACL’95. 1995.

Adwait Ratnaparkhi. 1997. A simple introduction to maximum entropy models for natural language processing. Technical Report 97-08, Institute for Research in Cognitive Science, University of Pennsylvania.

Adwait Ratnaparkhi, 1996, A Maximum Entropy Model for Part-of-Speech Tagging, Proceedings of the Conference on Empirical Methods in Natural Language Processing”, ACL, 133-12.

Christopher D. Manning and Hinrich Schütze, 1999, Foundations of Statistical Natural Language Processing, MIT Press

A Penn Treebank tree:

```
( (S
    (NP
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP (NNP Nov.) (CD 29) )))
    (. .) ))
```

Later, we will discuss parsing techniques that are capable of assigning full trees like this to sentences. But for now, we concentrate on a shallower level of analysis, known as 'chunking'. We'll concentrate on two kinds of 'chunk':

1. 'base NPs': non-recursive NPs i.e. not containing any other NPs.

2. 'verb groups': components of a Verb Phrase, excluding any obligatory complements or optional modifiers ('adjuncts').

Why not Verb Phrases? Because in order to do that accurately, we would need information about the required complements of each verb, in order to know that a sequence like 'on a horse' should be part of the verb phrase in:

He relied on the horse.

but not in:

He arrived on the horse.

Note that:

*He relied. (felt to be incomplete)

He arrived. (OK)

Given a complete parse tree like that shown earlier, we can recover the chunks as below:

```
[Pierre Vinken]/NP, [61 years]/NP old, [will join]/VG
[the board]/NP as [a nonexecutive director]/NP [Nov. 29]/NP.
```

Note that there is no nesting or overlapping of chunks. We can transform this to an equivalent 'chunk tag' representation:

```
Pierre/B-NP Vinken/I-NP, 61/B-NP years/I-NP old/O, will/B-VG
join/I-VG the/B-NP board/I-NP as/O ...
```

A tag of the form B-X means 'beginning of an X', I-X means 'inside an X', and O means 'outside any chunk'. It is easy to see how to reconstruct the actual chunks from this notation.

Now we can regard chunking as an extension of tagging, and use HMMs to do it automatically. We train an HMM from a POS and chunk-tagged corpus. The simplest way to do this is to treat the POS tags as if they were words, and the chunk tags as if they were POS tags. So instead of estimating output probabilities like P(the | DT) and transition probabilities like P(NN | DT), we have output probabilities of the form: P(DT | B-NP) and transition probabilities of the form P(I-NP | B-NP).

Now in order to do chunking we first do ordinary POS tagging of the sentence, and then we tag the POS tags with chunk tags. Then we recover the chunks from these tags.

In fact, it turns out that performance can be improved if we enrich the chunk tags by combining them with POS tags and selected words (e.g. all the closed class words). From the training corpus we produce a derived corpus where the tags are:

Word	POS tag	Original chunk tag	Chunk+POS	Chunk+POS+Word
You	PRP	B-NP	PRP-B-NP	you-PRP-B-NP
will	MD	B-VG	MD-B-VP	will-MD-B-VG
start	VB	I-VG	VB-I-VG	VB-I-VG
the	DT	B-NP	DT-B-NP	the-DT-B-NP
car	NN	I-NP	NN-I-NP	NN-I-NP

We can vary the words we include by experimenting to minimise error.

MEASURING ACCURACY

For some tasks, like POS tagging, we simply count the number of errors. A typical HMM tagger will achieve about 95% accuracy, so about 1 in 20 POS tags will be wrongly assigned. (The unigram baseline is about 80%). Note that this is for a 'trigram' tagger, where the transition probabilities are of the form P(C$_n$ | C$_{n-1}$, C$_{n-2}$), i.e. we condition on the previous two tags instead of just one.

ASIDE: Strictly speaking, trigrams violate the Markov assumption that previous states don't affect what happens in the current state. A well known way of overcoming this is to make our POS tags more complex, to encode earlier states implicitly:

```
<start>/START the/DT man/NN arrived/VBD <end>/END
<start>/START the/START-DT man/DT-NN arrived/NN-VBD <end>/VBD-END
```

Now we can just proceed in training and tagging just as for the bigram case, with a simple unpacking of POS labels when we have finished.

RECALL AND PRECISION

We also measure accuracy by RECALL and PRECISION. For example, for the NP chunking task we would have:

$$\text{Recall} = \frac{\text{Number of correct NPs found}}{\text{Number of NPs in test corpus}}$$

$$\text{Precision} = \frac{\text{Number of correct NPs found}}{\text{Number of NPs found}}$$

Recall and precision are often combined in the 'F' measure:

$$F = \frac{2\text{*Recall*Precision}}{\text{Recall}+\text{Precision}}$$

ACCURACY OF HMM CHUNKING

As reported by Molina and Pla (2002):

Chunk tags	precision	recall	F
Basic	84.31	84.35	84.33
with POS tags	89.58	89.55	89.57
with POS+words	91.96	92.41	92.19

Trained on sections 15-18 of WSJ Penn Treebank, tested on 20. Results improve by about 1.5% if trained on 0-19.

Transformation Based Tagging

There are various non-statistical methods for tagging which rely on machine learning techniques of one form or another. Brill's method is simple and popular:

1. assign each word its most likely part of speech independent of context (P(Tag|Word) - learn this from a corpus).
2. apply an ordered set of 'patch' rules to change incorrect tags to correct ones (rules also learned from a corpus).

Patch rules are of the following form:

Change tag A to tag B:
-when {first/second} {preceding/following} word is W
-when one of {1/2/3} {preceding/following} words is tagged Z
-when {current/preceding} word {is/is not} capitalised
etc

We learn these rules by creating a set of templates like this, instantiating them with all the particular tag/word combinations seen in the data, and retaining those instantiated rules that help.

Transformation Based Tagging

Start with a null set of patch rules.

A. Compare the current corpus (first time in from applying step 1 on previous slide) with the original and count errors $\langle$wrong tag, right tag, count$\rangle$. Rank them by frequency. If there are no errors, or a threshold is reached, stop.

B. Create a patch rule by filling in each of the templates. This will typically give a very large number of possible patches.

C. Starting with the most frequent errors, compute the reduction in error for each patch rule, i.e. (number of errors corrected minus number of new errors introduced).

D. Add the patch rule with the highest error reduction, retag the corpus and go to A. Note this imposes an ordering on patch rules: they are applied in the order they are added.

Example patches:

Replace POS with VBZ if the preceding word is tagged PRP
He/PRP 's/POS leaving/VBG... $\rightarrow$
        He/PRP 's/VBZ leaving/VBG...

Replace VBP with VB if one of the 3 preceding words is tagged MD
He/PRP might/MD leave/VBP $\rightarrow$
        He/PRP might/MD leave/VB

Replace VBP with VB if the preceding word is tagged TO
He/PRP wants/VBZ to/TO leave/VBP... $\rightarrow$
        He/PRP wants/VBZ to/TO leave/VB....

Applying the final unigram+patches system to unseen text gives an error rate of about 4.5%, which is comparable to an HMM tagger.

Transformation based NP chunking

NB treatment of possessives, and slightly different ‘chunk tags’ in this work:

Even [Mao Tse-tung] [’s China] began in [1949] with [a partnership] between [the communists] and [a number] of [smaller, non-communist parties].

> I = inside some NP; O = outside
> B = leftmost member of NP immediately following another NP

Even/O Mao/I Tse-tung/I ’s/B China/I began/O in/O 1949/I with/O a/I partnership/ between/O the/I communists/I and/O a/I number/I of/O smaller/I non-communist/I parties/I.

Can reconstitute brackets easily:

W1/I W2/O $\rightarrow$ W1] W2
W1/I W2/B $\rightarrow$ W1][W2
W1/O W2/I $\rightarrow$ W1 [W2
W1/O W2/B $\rightarrow$ W1/O W2/I (correct an error)

TBL chunking

Baseline: start with tagged text. Assign each tag its most likely chunk tag (unigram). Using Brill's procedure, learn templates based on these patterns:

Word Pattern	Meaning	Chunk Tag Pattern	Meaning
W_0	current word	T_0	current tag
W_{-1}	previous word	T_{-1}, T_0	current and previous
W_1	next	T_0, T_1	current and next
W_{-1},W_0	current and previous	T_{-2},T_{-1}	two previous
W_0,W_1	current and next	T_1,T_2	two next
W_{-1},W_1	previous and next		
W_{-2},W_{-1}	two previous		
W_1,W_2	two next		
$W_{-1,-2,-3}$	one of three previous		
$W_{1,2,3}$	one of three next		

100 rule templates:
(10 Word Patterns * 5 chunk tag patterns) + (10 POS Patterns * 5 chunk tag patterns)

Rules learned:

JJ/I w/0	→	JJ/O w/O
w1/I w2/I DT/I	→	w1/I w2/I DT/B
w/O DT/I w2/O	→	w/O DT/I w2/I
...		
w/O about/O	→	w/O about/I (‘including [about three N]’)

Test set = 50K words

Training	Recall	Precision
Baseline	81.9	78.2
50K	90.4	89.8
100K	91.8	91.3
200K	92.3	91.8

Maximum Entropy Tagging

In HMM tagging we only use two features of the context: the word itself and the tag of the previous word. It would be nice to use a richer set of features, e.g.

if previous tag is VB, previous word is ‘be’, current word ends in ‘y’, then tag of current word = JJ, etc., as well as combinations of them. We can do this in a general way by using ‘indicator’ features of the form $f_i = 1$ if previous word is ‘cat’, current word ends in ‘ed’ and ‘dog appears within 5 words to the right of the current word, and $f_i = 0$ otherwise. These features will always be of the form:

f(item, label) = 1 if conditions are met, 0 otherwise.

But how do we use such features? One way might be to gather probability estimates for each of them independently from the training data, and then combine them in some way. But while that is possible, it is not optimal, for some of the features may not be independent of each other.

For example, it is common to have as features suffixes and prefixes of words of different lengths:

e.g. leaving = suffixes = g, ng, ing. Clearly these are not independent features. So we need a way of getting a probability distribution which takes account of any dependencies in the training data, but which makes no other assumptions at all. This is known as a ‘maximum entropy’ distribution, because it is the one that has the most uncertainty (fewest assumptions, least commitment) about things for which there is no direct evidence in the data.

Maximum Entropy Distributions

Imagine we have just one (binary valued) feature, f1, and two data items, a and b. So we want to estimate the probability distribution p(x,y) where $x \in \{a,b\}$ and $y \in \{0,1\}$. Let's assume that all we know from the data is that $p(f1(a)=0) + p(f1(b)=0) = 0.6$. We can represent this as:

p(x,y)	0	1	
a	?	?	
b	?	?	
total	0.6		1.0

An unwarranted estimate:

p(x,y)	0	1	
a	0.4	0.3	
b	0.2	0.1	
total	0.6	0.4	1.0

An estimate that maximises the entropy:

p(x,y)	0	1	
a	0.3	0.2	
b	0.3	0.2	
total	0.6	0.4	1.0

Training a Max Ent model

Training such models can be computationally expensive. The most popular algorithm is known as 'generalised iterative scaling': you begin by assigning each feature a weight and a probability (from the training data), and then interatively adjusting the weights and probabilities under the constraint that the resulting expectation for each feature is the same as it was in the training data. When the algorithm converges (i.e. no further important changes occur in the weights) you are guaranteed to have a probability distribution that is faithful to what you saw in training and which distributes the rest of the probability mass in such a way as to give the maximum entropy distribution.

Ratnaparkhi's useful introduction is at:

http://citeseer.ist.psu.edu/128751.html

Running a Max Ent Tagger

The Max Ent distribution allows you now to calculate the most likely tag for each word given the current context. To calculate the most likely sequence of tags for a sentence:

```
Find the N (usually N = 5) most likely tags for the first word.

for words 2 - end of sentence
  For each of N previous tags
     calculate  tags for current word
  Choose N best sequences from among these
```

Ratnaparkhi's tagger gets 96-97% accuracy on an unseen portion of the Treebank.

You can use exactly the same technique for chunking, using a set of features like those in TBL. This gets over 90% accuracy.