7) Parsing Accuracy

Paula Buttery

Dept of Computer Science & Technology, University of Cambridge
We have looked at:

- grammars (PCFG, dependency, CCG)
- parsing algorithms (dynamic, deterministic, heuristic)
- parse scoring models (Bayesian, log-linear, cost-functions)
- methods for selecting n-best parses (beams, agendas)

But what do we need to do to make the parses as accurate as possible...?
Reminder: PCFGs have some shortcomings

When we looked at PCFGs we noted two sources of inaccuracy:

- **The independence** assumption: unable to model structural dependency across the tree as a whole
  - The choice of how a non-terminal expands depend on the location in the parse tree.
  - In English, subject NPs are more likely to be pronouns (≈ 90%), and objects NPs are more likely to be non-pronominal (≈ 60%)

- Lack of **lexical specificity**: unable to model the structural behaviour specific to a lexical item
  - If the PCFG reflect thats *VP*-attachment of *PPs* are more common then we will always get some people like beer in cold glasses wrong
  - Lack of subcategorisation
  - Non-sensical co-ordination can be probable...
Reminder: PCFGs have some shortcomings

Lack of **lexical specificity**: these co-ordinated trees have the same probability...

From Jurafsky and Martin version 3, following Collins
Reminder: PCFGs have some shortcomings

Lack of **lexical specificity**: these co-ordinated trees have the same probability...

From Jurafsky and Martin version 3, following Collins

Today will we look as how to get around these issues.
Relax independence by splitting non-terminals

For the pronoun issue, intuition is that we need more NP rules:

Instead of $NP \rightarrow PRP$ we need two rules:

$NP_{subject} \rightarrow PRP$

$NP_{object} \rightarrow PRP$

How can we implement this without a semantic treebank? by annotating non-terminals with their parent nodes.

One way to implement this intuition of splits is to do parent annotation (John-son, 1998), in which we annotate each node with its parent in the parse tree. Thus, an $NP$ node that is the subject of the sentence and hence has parent $S$ would be annotated $NP^{S}$, while a direct object $NP$ whose parent is $VP$ would be annotated $NP^{VP}$.

Figure 12.8 shows an example of a tree produced by a grammar that parent-annotates the phrasal non-terminals (like $NP$ and $VP$). All the non-terminal nodes (except the pre-terminal part-of-speech nodes) in parse (b) have been annotated with the identity of their parent.

In addition to splitting these phrasal nodes, we can also improve a PCFG by splitting the pre-terminal part-of-speech nodes (Klein and Manning, 2003b). For example, different kinds of adverbs (RB) tend to occur in different syntactic positions: the most common adverbs with ADVP parents are also and now, with VP parents isn't and not, and with NP parents only and just. Thus, adding tags like $RB^{ADVP}$, $RB^{VP}$, and $RB^{NP}$ can be useful in improving PCFG modeling.

Similarly, the Penn Treebank tag IN can mark a wide variety of parts-of-speech, including subordinating conjunctions (while, as, if), complementizers (that, for), and prepositions (of, in, from). Some of these differences can be captured by parent annotation (subordinating conjunctions occur under S, prepositions under PP), while others require specifically splitting the pre-terminal nodes.

Figure 12.9 shows an example from Klein and Manning (2003b) in which even a parent-annotated grammar incorrectly parses works as a noun in to see if advertising works. Splitting pre-terminals to allow if to prefer a sentential complement results in the correct verbal parse.

To deal with cases in which parent annotation is insufficient, we can also hand-write rules that specify a particular node split based on other features of the tree. For example, to distinguish between complementizer IN and subordinating conjunction IN, both of which can have the same parent, we could write rules conditioned on other aspects of the tree such as the lexical identity (the lexeme that is likely to be a complementizer, as a subordinating conjunction).

Node-splitting is not without problems; it increases the size of the grammar and hence reduces the amount of training data available for each grammar rule, leading to overfitting. Thus, it is important to split to just the correct level of granularity for a particular training set. While early models employed hand-written rules to try to find an optimal number of non-terminals (Klein and Manning, 2003b), modern models...
Relax independence by splitting non-terminals

For the pronoun issue, intuition is that we need more NP rules: instead of $NP \rightarrow PRP$ we need two rules:

- $NP_{subject} \rightarrow PRP$
- $NP_{object} \rightarrow PRP$

How can we implement this without a semantic treebank?
Relax independence by splitting non-terminals

For the pronoun issue, intuition is that we need more NP rules: instead of $NP \rightarrow PRP$ we need two rules:

- $NP_{subject} \rightarrow PRP$
- $NP_{object} \rightarrow PRP$

How can we implement this without a semantic treebank? by annotating non-terminals with their parent nodes

From Jurafsky and Martin version 3
Other examples of parent annotation:

- e.g. differentiating between **adverbs** by annotating pre-terminals (called **tag splitting**), with their parents: most common adverbs directly under ADVP are *also* and *now*; under VP are *n’t* and *not*; under NP are *only* and *just* ...

- e.g. **subordinating conjunctions**, *while*, *as*, *if*, occur under *S*

Where parent annotation can’t help we could split on other features (i.e. hand write rules for specific feature scenarios)

See https://nlp.stanford.edu/manning/papers/unlexicalized-parsing.pdf for some discussion
A **trade-off** between splitting and training

- Splitting non-terminals increases the grammar size
- Increased grammar size means less data per rule instance for MLE
- **Split and merge** techniques automatically search for the optimal splits by **maximising the likelihood** of the training set (e.g. Petrov et al. 2006)
non-terminal splitting example...
Lexicalised-PCFGs include **lexical** info in the grammar

**Collins** and **Charniak** parsers use lexicalised-PCFGs

- Lexicalisation can include both the **head** word token and its part-of-speech

```
TOP
  S(dumped,VBD)
  NP(workers,NNS)
    NNS(workers,NNS)
    workers
    dumped
  VP(dumped,VBD)
    VBD(dumped,VBD)
    NP(sacks,NNS)
      NNS(sacks,NNS)
      sacks
    PP(into,P)
      P(into,P)
      into
      DT(a,DT)
      a
      NN(bin,NN)
    NP(bin,NN)
      NN(bin,NN)
      bin
```

From Jurafsky and Martin version 3
Lexicalised-PCFGs include lexical info in the grammar

Creating the treebank:
- For each rule, one of the RHS daughters is the head
- The head information for the LHS of the rule is the same as the RHS head

Estimating probabilities:
- Pre-terminal rules always have a probability of 1
- All other rule probabilities need to be calculated ...
  ... but the data available per rule is now very sparse
Lexicalised-PCFGs include lexical info in the grammar

Creating the treebank:
- For each rule, one of the RHS daughters is the head.
- The head information for the LHS of the rule is the same as the RHS head.

Estimating probabilities:
- Pre-terminal rules always have a probability of 1.
- All other rule probabilities need to be calculated...
  ... but the data available per rule is now very sparse.
Collins handles sparsity by generating the RHS of rules

- RHS of every rule consists of a **head** plus all the non-terminals to the head’s **left** and all the non-terminals to the head’s **right**

  \[ LHS \rightarrow L_m \ldots L_1 H R_1 \ldots R_n \]

- To use a rule we:
  - first **generate the head**,  
  - then all the **left dependents** from the head outwards  
  - and finally all the **right dependents** from the head outwards

- We imagine a **STOP** non-terminal at the edges of the rule

  \[ LHS \rightarrow \text{STOP} L_m \ldots L_1 H R_1 \ldots R_n \text{STOP} \]
Rule probability is the **product** of all generated pieces

Remember that for PCFGs: \( P(A \rightarrow B) = P(B|A) \)

For lexicalised-PCFGs rule probability is the **product** of its pieces:

* General rule form: \( A \rightarrow STOP L_m \ldots L_1 H R_1 \ldots R_n STOP \)
  - The probability of the head \( H \) with associated word \( h_w \) and tag \( h_t \) given the parent, \( A \) is:
    \[
P(H(w_h, t_h)) = P(H(h_w, h_t)|A, h_w, h_t)
    \]
  - The probability of modifiers to the left of the head is:
    \[
    \prod_{i=1}^{m+1} P(L_i(lw_i, lt_i)|A, H, h_w, h_t)
    \]
  - The probability of modifiers to the right of the head is:
    \[
    \prod_{i=1}^{n+1} P(R_i(rw_i, rt_i)|A, H, h_w, h_t)
    \]

where \( L_{m+1} = STOP \) and \( R_{n+1} = STOP \)
lexicalised-PCFG rule probability estimation...
Collins models have **other conditional features**

- Collins 1 includes a **distance metric** in the conditional probabilities
- Collins 2 includes conditioning on **subcategorisation** and **argument/adjunct**

- In training Collin’s interpolates three models:
  - fully lexicalised (conditioning on the head word and tag),
  - just the head tag
  - unlexicalized

https://www.aclweb.org/anthology/J03-4003.pdf
We can now understand better Charniak’s **coarse-to-fine** parsing strategy:

1. produce a parse forest using simple version of the grammar
   i.e. find possible parses using coarse-grained non-terminals, e.g. \( VP \)
2. refine most promising of coarse-grained parses using complex grammar
   i.e with feature-based, lexicalised non-terminals, e.g. \( VP[\text{buys}/\text{VBZ}] \)

- **Coarse-grained step** can be **efficiently parsed** using e.g. CKY
- But the simple grammar **ignores contextual features** so best parse might not be accurate
- **Output a pruned packed parse** forest for the parses generated by the simple grammar (using a beam threshold)
- **Evaluate remaining parses with complex grammar** (i.e. each coarse-grained state is split into several fine-grained states)