Beyond PCFGs

L645
Fall 2009
Topic #1: Lexicalized parsing

Lexical information seems to be crucial to parser success, so we’re going to look at work done in Collins (1997)

- *Three Generative, Lexicalised Models for Statistical Parsing*
- The task is to generate trees as in the Wall Street Journal part of the Penn Treebank
- The models are:
  - Lexicalized
  - Dependency-based
Dependencies

How do we incorporate dependency information from a treebank which doesn’t explicitly have dependencies?

(1) John Smith, the president of IBM, announced his resignation yesterday.

Here are some of the dependencies, along with their representation ... note that base NPs are treated as a unit

- \textit{Dependent} \rightarrow \textit{Head} \ldots \text{Hcat\_Mother\_DCat}

- [John Smith] \rightarrow \text{announced} \ldots \text{VP\_S\_NP}

- [the president] \rightarrow [John Smith] \ldots \text{NP\_NP\_NP}

- [his resignation] \rightarrow \text{announced} \ldots \text{VBD\_VP\_NP}

- yesterday \rightarrow \text{announced} \ldots \text{VBD\_VP\_NP}
Dependency probabilities

Each dependency has a particular probability, e.g., \( P(VBD\_VP\_NP) \)

The probability of a rule is made from the probabilities of the components

- \( P(VP \rightarrow VBD\_NP\_NP) \) composed of \( P(VBD\_VP\_NP) \) for first NP and \( P(VBD\_VP\_NP) \) for second NP

- Note that we do not treat adjuncts different from complements

- Note also that this depends upon our being able to locate the head of a rule (VBD) \( \rightarrow \) Collins uses head-finding rules/heuristics
Collins (1996) roughly used the following formula (which I’ve abbreviated):

\[ P(D|S) = \prod_{j=1}^{n} P(d(w_j, h, R)) \]

where:

- \( D \) = the overall dependency parse
- \( S \) = the sentence
- \( d(w, h, R) \) = a dependency between word \( w \), its headword \( h \), and the name of the relationship \( R \) (e.g., VBD_VP_NP) between them

But dependents need to be reworked to allow for lexical information, i.e., lexical dependencies
yesterday attaches to announced, even though not all NPs would be in this structure (cf. his resignation/NN papers/NNS)
Propagating Head words

S(announced)

NP(Smith)

NP(Smith)

NP(Smith)

NP(Smith)

NP(president)

NP(president)

NP(president)

NP(president)

NP(president)

VP(announced)

VP(announced)

VP(announced)

VP(announced)

VP(announced)

NP(resignation)

NP(resignation)

NP(resignation)

NP(resignation)

NP(resignation)

NP(yesterday)

NP(yesterday)

NP(yesterday)

NP(yesterday)

NP(yesterday)

S(announced)

John

Smith

the

president

IBM

resignation

yesterday
Lexicalized rules

- $S(\text{announced}) \rightarrow \text{NP}(\text{Smith}) \ \text{VP}(\text{announced})$
- $\text{NP}(\text{Smith}) \rightarrow \text{NP}(\text{Smith}) \ \text{NP}(\text{president})$
- $\text{NP}(\text{Smith}) \rightarrow \text{NNP}(\text{John}) \ \text{NNP}(\text{Smith})$
- $\text{NP}(\text{president}) \rightarrow \text{NP}(\text{president}) \ \text{PP}(\text{of})$
- $\text{PP}(\text{of}) \rightarrow \text{IN}(\text{of}) \ \text{NP}(\text{IBM})$
- ...

But how do we incorporate these probabilities? Don't they seem rather specific (i.e., data sparseness could be a problem)? ...
Generative models

Collins (1997) differs from Collins (1996) in that it is a generative model, not a conditional model.

- Generative model: maximize the sentence-tree pairing $P(S, T)$

- Conditional (or parsing) model: maximize the tree given the sentence: $P(T|S)$

But maximizing one is equivalent to maximizing the other.
Model 1

So, now we have lexical dependencies, but how do we get reasonable probability estimates for $P(RHS|LHS)$

That is, we have way too many possible rules after adding lexical information

- First, we note that the parent $LHS$ of a headword $h$ is composed of left and right sisters, in addition to the headword’s label $H$

  \[(3) \quad LHS \rightarrow L_n...L_1 \ H \ R_1...R_m\]

- We can break down the probability of $LHS$ into generating the left sisters, the head category, and the right sisters
The three steps for generating RHS

Collins’ model 1 is top-down, so we know the LHS and the head word

1. Generate the head constituent label, with probability $P(H|LHS, h)$

2. (Using H probability from step 1), generate the modifiers to the right, with probability $\prod_i P(R_i|LHS, h, H)$

3. (Using H probability from step 1), generate the modifiers to the left, with probability $\prod_i P(L_i|LHS, h, H)$

NB: Both right and left modifiers can generate the modifier STOP, which indicates that that branch is done

Given these 3 probabilities, we can calculate $P(RHS|LHS)$
Currently, the right sisters use the probability $P(R_i|LHS, h, H)$ (and likewise for the left sisters)

- This means that each sister is dependent *only* on the headword and its information, i.e., not dependent on any of its sisters
- But generating a third NP, for instance, should be much less likely than generating one NP to the right
- In principle, you could condition on any previous modifier information (because the model is depth-first and thus fills in all of a modifiers information)
- So, Collins incorporates a distance measure, which is used as a condition:

$P(R_i|LHS, h, H, distance(i - 1))$
The distance measure

The distance measure is a vector calculated based on the following information about the string between the headword and the current sister:

1. Is the string of zero length? (allows a preference for right-branching structures to be learned)
2. Does the string contain a verb?
3. Does the string contain 0, 1, 2, or > 2 commas?

Why these features? Because they are sensible and they worked well in a previous system.
Model 2

For the second model, Collins:

- distinguishes adjuncts from complements
  - Mark complement categories with a C (e.g. NP-C) whether a subject or an object
  - Identify them in the training data (Penn Treebank) using rules similar to head-finding rules

- adds subcategorization information

The two tasks are related, because to know that something is a complement we have to know that it was selected for
Motivating subcategorization

Remember in model 1 how we assumed that each child of a parent node was independently generated (which is why the distance feature was introduced)

This leads to some bad choices, even with complements marked:

• \[ NP_C \text{ Dreyfus} \ [NP_C \text{ the best fund}] \ [VP \text{ was low}] \]

• \[ NP_C \ [NP \text{ Dreyfus} \ [NP \text{ the best fund}] \ [VP \text{ was low}] \]

It turns out that \( P(NP-C(Dreyfus)|S,VP,was) \ast P(NP-C(fund)|S,VP,was) \) is “unreasonably high”, giving the wrong parse
Adding subcategorization information

1. Choose head constituent $H$ with probability $P(H|LHS, h)$

2. Choose left and right subcat frames $LC$ and $RC$ with probabilities:
   - $P(LC|LHS, H, h)$
   - $P(RC|LHS, H, h)$

3. Generate left and right modifiers with probabilities:
   - $P(L_i|LHS, H, h, LC)$
   - $P(R_i|LHS, H, h, RC)$
   - Distance measure also used, but we ignore it here

The important thing to note is that subcategorization information is added to the surrounding, conditioning context
Model 2—a few more details

What happens in step 3 above is that complements are checked off the subcategorization frame as they are found

- When the subcat frame is non-empty, the probability of stopping \( P(STOP) \) is 0
- When the subcat frame is empty, the probability of generating a complement is 0
  - An adjunct can still be generated, however

In other words, “all and only the required complements will be generated”
Model 3

The third model Collins uses we won’t discuss in detail

• Traces (i.e., *wh*-movement) are given a better treatment

• Pass gap requirements into the left and right subcat frames

• Keep track of if a trace is more likely to be generated to the left or right of a head word
## Results for lexicalized dependency parsing

For sentences with less than 40 words:

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>Crossing Brackets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 1</td>
<td>87.4%</td>
<td>88.1%</td>
<td>0.96</td>
</tr>
<tr>
<td>Model 2</td>
<td>88.1%</td>
<td>88.6%</td>
<td>0.91</td>
</tr>
<tr>
<td>Model 3</td>
<td>88.1%</td>
<td>88.6%</td>
<td>0.91</td>
</tr>
</tbody>
</table>
Topic #2: Parser Reranking

Charniak and Johnson (2005):

- Coarse to fine n-best parsing and MaxEnt discriminative reranking
Motivation

50-best parses returned by a parser often include better parses than 1-best

- Reranker can select the best parse

- relying on only a small number of parses per sentence
  - Thus, the features used for reranking can be arbitrary functions of the parse tree
Overview

• For each string $s$, parsing algorithm returns $n$—highest probability parses

$$\mathcal{Y}(s) = \{y_1(s), ..., y_n(s)\}$$

– and returns the probability ($p(y)$) of each parse $y$

• Feature extractor consists of functions $f = (f_1, ..., f_m)$, where each $f_j$ maps parse to a real number

$$f(y) = (f_1(y), ..., f_m(y))$$
Overview (2)

• Parse score $v_\theta$ is a linear function of feature values

$$v_\theta = \theta \cdot f(y) = \sum_{j=1}^{m} \theta_j f_j(y)$$

  – each feature $f_j$ is associated with a weight $\theta_j$

• Take the best reranked parse

$$\hat{y}(s) = \arg \max_{y \in \mathcal{Y}(s)} v_\theta(y)$$
Generating \( n \)-best lists

Discriminative reranker: requires \( n \)-best lists

- Problem: dynamic programming throws away some of the best parses
- One solution: use a beam search

Solution using dynamic programming:

- Viterbi: for optimal parse, the decision at each choice point must be optimal
  - Second-best parse: all but one of the parsing decisions are optimal
- \( n^{th} \) best parse: can have at most \( n \) suboptimal decisions
  - all but one are in the \( 2^{nd}-(n-1)^{th} \)-best parses

Idea: find best parse, then \( 2^{nd} \)-best, then \( 3^{rd} \)-best, etc.
Generating $n$-best lists (2)

Straightforward solution: Store $n$ best parses at each edge

- Issue: space
  - Normally require $O(m^2)$ states ($m =$ length of sentence)
  - Now require $O(nm^2)$
Coarse-to-fine parsing

But: Charniak parser is a coarse-to-fine parser:

- First produces crude parse based upon a less complex version of the complete grammar
  - uses only standard CFG features (parent & neighbor labels)
  - can be efficiently parsed with dynamic programming
  - outputs a packed parse

- Then prune the edges in the forest, whose probability $p(n|s)$ is below a certain threshold
Probabilities for pruning

\[ p(n_{j,k}^i|s) = \frac{\alpha(n_{j,k}^i)\beta(n_{j,k}^i)}{p(s)} \]

- \( n_{j,k}^i \) = constituent of type \( i \) spanning from \( j \) to \( k \)
- \( \alpha(n_{j,k}^i) \) = outside probability of constituent
- \( \beta(n_{j,k}^i) \) = inside probability of constituent
Ranking unpruned edges

Use a fine-grained probabilistic model, with non-local contextual information:

- lexical head of one’s parents & its POS
- parent’s & grandparent’s category labels

These are refinements (splits) of the coarse-grained model

Fine-grained pass keeps $n$-best possibilities

- 50-best possibilities has an oracle $f$-score of 96.8% (vs. 89.7% base $f$-score)
Features for reranking (general ideas)

Feature vector \( f(y) \)

- First feature \( f_1(y) \) is the log probability of the best parse

- Other features are integer-valued:
  - Each value associated with a particular configuration
  - \( f_j(y) = \text{number of times that the configuration is realized} \)
    * e.g., \( f_{\text{eat,pizza}}(y) = \text{number of times a phrase in } y \text{ headed by } \text{eat} \text{ has a complement headed by } \text{pizza} \)

In total, there are 1,148,697 features used (occurring at least 5 times in training)
More on features

Feature schema are used: abstract scheme from which specific features are derived

- parameterized in various ways
- e.g., **Heads** schema parameterized by type of heads
  - e.g., functional head of NP = determiner
  - e.g., lexical head of NP = noun

Ignored all features which did not vary on the parses of at least $t$ sentences ($t$ was set to 5)
Features for reranking (specific examples)

**RightBranch** (2): e.g., number of nonterminal nodes on the path from root to rightmost non-punctuation preterminal

**Heavy** (1049): nodes classified by category, binned length, at end of sentence, followed by punctuation

**Neighbours** (38,245): nodes classified by category, binned length, POS of preterminals to node’s left/right

**Heads** (208,599): tuples of head-head dependencies
Feature weighting & Classification

Find feature weights $\hat{\theta}$ via a MaxEnt estimator

(5) $\hat{\theta} = \arg \min_\theta L_D(\theta) + R(\theta)$

- $L_D(\theta) =$ loss function
- $R(\theta) =$ regularization penalty term

For MaxEnt modeling:

(6) $v_\theta(y) = \theta \cdot f(y) = \sum \theta_j f_j(y)$

(7) $P_\theta(y|\mathcal{Y}) = \frac{\exp v_\theta(y)}{\sum_{y' \in \mathcal{Y}} \exp v_\theta(y')}$
Results

- With Charniak parser: \(f\)-score of .9102
- With Collins parser: \(f\)-score of .9037
Topic #3: Parser Adaptation

McClosky et al. (2006):

• *Reranking and Self-Training for Parser Adaptation*
Motivation

Parser performance degrades when moving to a new domain

• Treebanks for training are expensive & difficult to produce

• Parsers are too sensitive to training domain

*Parser adaptation*: “leverage[s] existing labeled data from one domain [to] create a parser capable of parsing a different domain”

• Status: Charniak parser has 89.7% on in-domain WSJ but 82.9% on Brown data
Approaches

- **Parse re-ranking**
  - Start with standard generative parser & use it to get $n$-best results
  - Using more fine-grained features can make parser more broadly applicable

- **Self-training**
  - Parse unlabeled data & add it to training corpus
  - Two things help: more data & greater relevancy of data
Adapt WSJ to Brown data

- Add previously unparsed NANC (North American News Corpus) to the training data (24 million sentences)
- This data is the same domain as WSJ—i.e., adapting without changing training domains

Parse NANC with WSJ-trained parser/reranker

- Best parses are combined with WSJ to train a new parser
Adapting self-training

Testing on the Brown development data, baseline WSJ parser has $f$-score of 83.9%

- Reranker increases it to 85.8%
- Reranker+self-training increases to 87.7%

Brown training data results in 87.4% accuracy
Incorporating in-domain data

What about if we have some in-domain data?

• Adding some Brown data can increase $f$-measure to 88.1%

• NB: too much NANC data, however, lowers accuracy, as it begins to outweigh the more accurate Brown data
Topic #4: Unlexicalized Parsing

Klein and Manning (2003)

- Accurate Unlexicalized Parsing
Motivation for unlexicalized parsing

Why explore unlexicalized parsing?

- Context-free assumptions of PCFGs are too strong
  - Adding parent annotation helps (Johnson 1998)
  - Bilexical dependencies may not help as much as thought (Gildea 2001): extremely sparse

- Subcategorization-type information seems to be complementary to lexicalization
Goals

Goals:

• set a better lower-bound for unlexicalized parsers

• determine where lexicalized probabilities are needed and available

• affirm the value of linguistic analysis for feature discovery
Vertical Markovization

PCFG grammar is imperfect:

- “Category symbols are too coarse to adequately render the expansions independent of context”
  - Subject NPs are 8.7 times more likely than object NPs to expand as a pronoun

This can be accounted for by parent annotation:

- NP^S \approx \text{subject}
- NP^\text{VP} \approx \text{object}
Vertical Markovization (2)

Can generalize this to the adding the past $v$ ancestors to the node

- “only the past $v$ vertical ancestors matter to the current expansion”

Results show that more vertical annotation tends to help
Horizontal Markovization

Just as one can have vertical context, there is also horizontal context to consider, i.e., within the RHS of a rule

Problem: many rules are rare and many have never been observed

• Leads to weird parses

Solution: markovize the rules, out from the head child

• only the past $h$ horizontal ancestors matter to the current expansion (nb: head always matters)
Example

Consider \( v = 1, h = 1 \)

Treebank rule: \( VP \rightarrow VBZ\ NP\ PP \)

- \( <VP:[VBZ]> \rightarrow VBZ \)
- \( <VP:[VBZ]...NP> \rightarrow <VP:[VBZ]> NP \)
- \( <VP:[VBZ]...PP> \rightarrow <VP:[VBZ]...NP> PP \)
- \( VP \rightarrow <VP:[VBZ]...PP> \)

Default treebank rules: \( v = 1, h = \infty \)
External & Internal Annotation

- Parent annotation = external annotation: indicates properties about the external environment

- Head lexicalization = internal annotation: marks a distinctive aspect of internal contents of a node

Both pieces of information are useful
Unary-Internal

e.g., S can expand as only VP

- Only occurs when S is the complement of a verb and has an empty, controlled subject

- Want to restrict unary rules to such contexts

UNARY-INTERNAL marks any nonterminal node with one child (-U)

- e.g., the incorrect VP^S → VBD NP^NP S^VP-U has a low probability
Unary-DT

Likewise, determiner tags are unary only for demonstrative determiners (*this*, *that*), not regular ones (*a*, *the*)

- **UNARY-DT** captures this

- **UNARY-RB** is also useful (cf. *as well* vs. *also*)
Tag Splitting

Parent annotate POS tags, too! (TAG-PA)

- “when a tag somewhat regularly occurs in a non-canonical position, its distribution is usually distinct”
  - e.g., RB^ADVP → *also* vs. RB^VP → *not* vs. RB^NP → *only*

- PTB tagset conflates grammatical distinctions which could help parsing
  - e.g., IN marks subordinating conjunctions, complementizers, prepositions
  - defined SPLIT-IN, SPLIT-AUX, & SPLIT-CC models (NB: closed classes)
Head annotation

Information about the head tag can provide useful indication about constituent’s behavior

- **POSS-NP**: possessive NPs behave differently: only NPs which are found in NP $\rightarrow$ NP $\alpha$

- **SPLIT-VP**: VP refers to all types of VPs
  - annotate all VPs with their head tag (e.g., VB), merging all finite forms to VBF
  - this captures finite/non-finite distinctions
Distance

Remaining errors: attachment level & conjunction scope

• high & low attachments often have same probabilities

Goal: capture a preference for right-branching

• Marking non-recursive base NPs helps indicate PP attachment (for 2 PPs following an NP)

• Base NP = dominating only a preterminal
Topic #4b: Automatic discovery of unlexicalized rules

Petrov et al. (2006) learn “hidden” annotations of trees automatically

• Split & merge nodes: “allocates subsymbols adaptively where they are most effective, like a linguist would”

• Use the EM algorithm

We’ll discuss this in class ...
References


