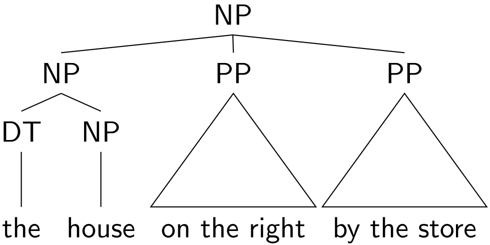
CS 4650/7650
Modern statistical parsers

Jacob Eisenstein

October 16, 2014

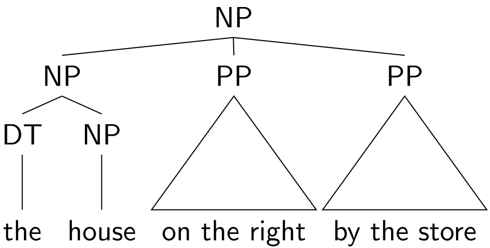
Subsumption

The PTB standard for multiple PPs is "two-level attachment"



Subsumption

The PTB standard for multiple PPs is “two-level attachment”

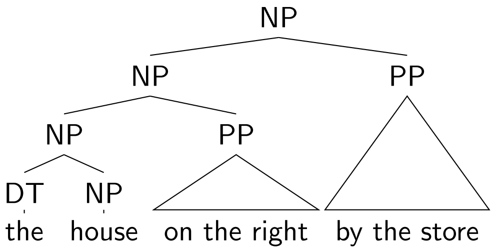


- However, multiple PPs are rare, and we have observed probabilities:

$$P(\mathrm{NP} \rightarrow \mathrm{NP\ PP}) = 0.112$$
$$P(\mathrm{NP} \rightarrow \mathrm{NP\ PP\ PP}) = 0.006$$

Subsumption

The PTB standard for multiple PPs is “two-level attachment”

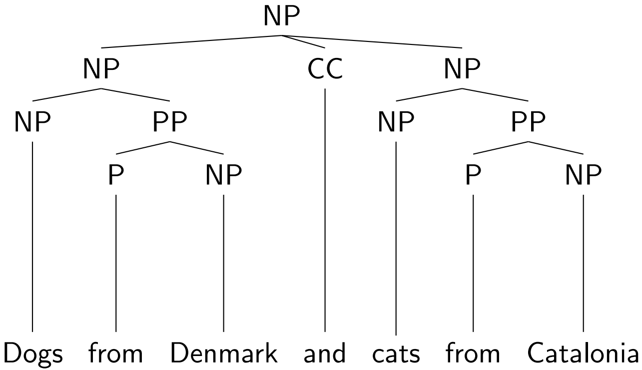


- However, multiple PPs are rare, and we have observed probabilities:

$$P(\mathrm{NP} \to \mathrm{NP\ PP}) = 0.112$$
$$P(\mathrm{NP} \to \mathrm{NP\ PP\ PP}) = 0.006$$
$$P(\mathrm{NP} \to \mathrm{NP\ PP})^2 = 0.013$$

- The PCFG parser will choose the “adjunction representation,” even though it is never annotated this way.
- Johnson (1998) showed the 9% of all productions are **subsumed**.

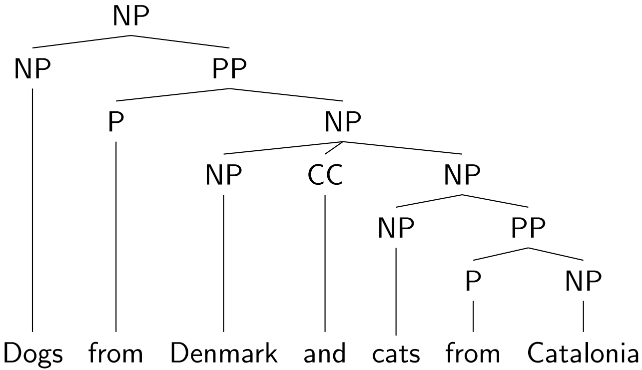
Equivalent parses under PCFGs



NP $\rightarrow$ NP CC NP	1
NP $\rightarrow$ NP PP	2
PP $\rightarrow$ P NP	2

All parses get the same probability, regardless of the training data.

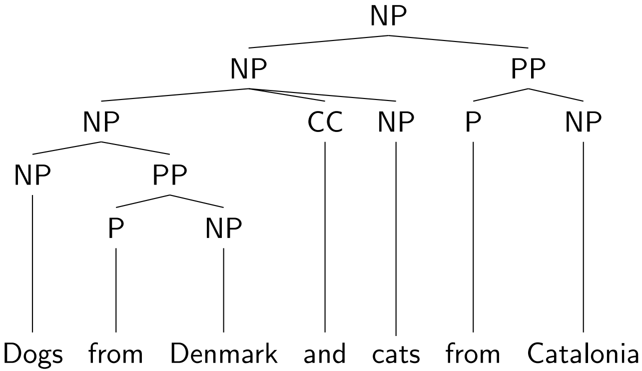
Equivalent parses under PCFGs



NP $\rightarrow$NP CC NP	1
NP $\rightarrow$NP PP	2
PP $\rightarrow$P NP	2

All parses get the same probability, regardless of the training data.

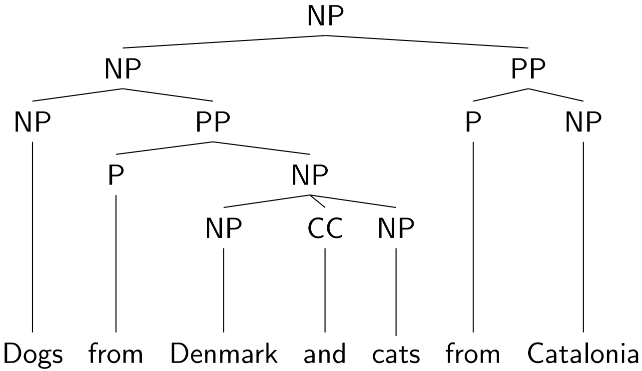
Equivalent parses under PCFGs



NP $\rightarrow$ NP CC NP	1
NP $\rightarrow$ NP PP	2
PP $\rightarrow$ P NP	2

All parses get the same probability, regardless of the training data.

Equivalent parses under PCFGs



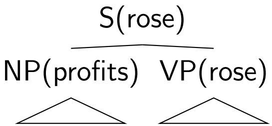
NP $\rightarrow$ NP CC NP	1
NP $\rightarrow$ NP PP	2
PP $\rightarrow$ P NP	2

All parses get the same probability, regardless of the training data.

The Charniak parser

The Charniak (1997) parser gives a relatively straightforward way to lexicalize PCFGs.

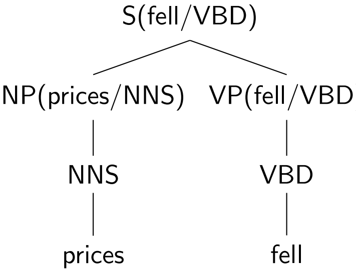
- Compute the head probability:
  $P(s_i | t_i, s_{p(i)}, t_{p(i)})$.
  - s_i is the head of constituent i
  - t_i is the syntactic category
  - $p(i)$ is the parent of node i
- Compute the rule probability:
  $P(r_i | t_i, s_i, t_{p(i)})$.
- Score each production by the product of the rule probability and the head probabilities.
- Apply standard CKY bottom-up parsing.



Head probabilities

The head probabilities capture “bilexical” phenomena, like the PP attachment (*President of Mexico*) example.

- $P(prices|NNS) = .013$
- $P(prices|NNS, NP) = .013$
- $P(prices|NNS, NP, S) = .025$
- $P(prices|NNS, NP, S, VBD) = .052$
- $P(prices|NNS, NP, S, VBD, fell) = .146$



Lexically conditioned rule probabilities

The rule probabilities capture phenomena like verb complement frames.

Local Tree	*come*	*take*	*think*	*want*
VP $\rightarrow$ V	9.5%	2.6%	4.6%	5.7%
VP $\rightarrow$ V NP	1.1%	32.1%	0.2%	13.9%
VP $\rightarrow$ V PP	34.5%	3.1%	7.1%	0.3%
VP $\rightarrow$ V SBAR	6.6%	0.3%	73.0%	0.2%
VP $\rightarrow$ V S	2.2%	1.3%	4.8%	70.8%
VP $\rightarrow$ V NP S	0.1%	5.7%	0.0%	0.3%
VP $\rightarrow$ V PRT NP	0.3%	5.8%	0.0%	0.0%
VP $\rightarrow$ V PRT PP	6.1%	1.5%	0.2%	0.0%

Data sparseness

- The Penn Treebank is still the main dataset for syntactic analysis of English.
- Yet 1M words is not nearly enough data to accurately estimate lexicalized models.
  - 965K constituents
  - 66 examples of WHADJP
  - only 6 of these aren't *how much* or *how many*
- Smoothing is absolutely critical for lexicalized parsers.

Smoothing the Charniak Parser

Head probability:

$$\begin{aligned}\hat{P}(s_i|t_i, s_{p(i)}, t_{p(i)}) =& \lambda_1 P_{mle}(s_i|t_i, s_{p(i)}, t_{p(i)}) \\ &+ \lambda_2 P_{mle}(s_i|t_i, \text{cluster}(s_{p(i)}), t_{p(i)}) \\ &+ \lambda_3 P_{mle}(s_i|t_i, t_{p(i)}) \\ &+ \lambda_4 P_{mle}(s_i|t_i)\end{aligned}$$

Smoothing the Charniak Parser

Head probability:

$$\begin{aligned}\hat{P}(s_i|t_i, s_{p(i)}, t_{p(i)}) = &\lambda_1 P_{mle}(s_i|t_i, s_{p(i)}, t_{p(i)}) \\ &+ \lambda_2 P_{mle}(s_i|t_i, \text{cluster}(s_{p(i)}), t_{p(i)}) \\ &+ \lambda_3 P_{mle}(s_i|t_i, t_{p(i)}) \\ &+ \lambda_4 P_{mle}(s_i|t_i)\end{aligned}$$

For example:

	$P(profit\|NP, rose, S)$	$P(corp.\|JJ, profit, NP)$
$P(s_i\|t_i, s_{p(i)}, t_{p(i)})$	0	.245
$P(s_i\|t_i, \text{cluster}(s_{p(i)}), t_{p(i)})$	.0035	.015
$P(s_i\|t_i, t_{p(i)})$	.00063	.0053
$P(s_i\|t_i)$	.00056	.0042

Smoothing the Charniak Parser

Head probability:

$$\begin{aligned}\hat{P}(s_i|t_i, s_{p(i)}, t_{p(i)}) =& \lambda_1 P_{mle}(s_i|t_i, s_{p(i)}, t_{p(i)}) \\ &+ \lambda_2 P_{mle}(s_i|t_i, \text{cluster}(s_{p(i)}), t_{p(i)}) \\ &+ \lambda_3 P_{mle}(s_i|t_i, t_{p(i)}) \\ &+ \lambda_4 P_{mle}(s_i|t_i)\end{aligned}$$

For example:

	$P(profit \mid NP, rose, S)$	$P(corp. \mid JJ, profit, NP)$
$P(s_i \mid t_i, s_{p(i)}, t_{p(i)})$	0	.245
$P(s_i \mid t_i, \text{cluster}(s_{p(i)}), t_{p(i)})$	.0035	.015
$P(s_i \mid t_i, t_{p(i)})$	.00063	.0053
$P(s_i \mid t_i)$	.00056	.0042

We have to tune $\lambda_1 \ldots \lambda_4$, and an equivalent set of parameters for the rule probabilities.

Smoothing the Charniak Parser

- The Charniak parser suffers from acute sparsity problems because it estimates the probability of entire rules.
- Another extreme would be to generate the children independently from each other.
  e.g., $P(S \rightarrow NP\ VP) \approx P_L(S \rightarrow NP)P_R(S \rightarrow VP)$
- The Collins (1999) and Charniak (2000) go for a compromise, conditioning on the parent and the head child.

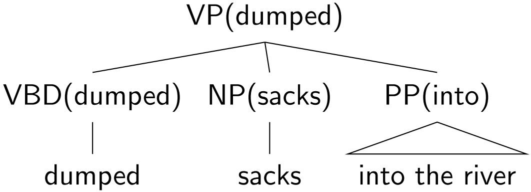
The Collins Parser

- The Charniak parser focuses on lexical relationships between children and parents.
- The Collins (1999) parser focuses on relationships between adjacent children of the same parent. It decomposes each rule as,

$$X \rightarrow L_i L_{i-1} \dots L_1 H R_1 \dots R_{j-1} R_j$$

- Each L and R is a child constituent of X, and they are generated from the head H outwards.
- The outermost elements of L and R are special $\bullet$ symbols.

Example



To model this rule, we would compute:

$$P(VP(dumped, VBD) \rightarrow \bullet\ VBD(dumped, VBD)\ NP(sacks, NNS)\ PP(into, P)\ \bullet)$$

Example

- Here's the generative process:

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(sacks, NNS)|VP(dumped, VBD), VBD(dumped, VBD))$

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(sacks, NNS)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(into, PP)|VP(dumped, VBD), VBD(dumped, VBD))$

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(sacks, NNS)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(into, PP)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(sacks, NNS)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(into, PP)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
- The rule probability is the product of these generative probabilities.

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(sacks, NNS)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(into, PP)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
- The rule probability is the product of these generative probabilities.
- **Horizontal Markovization**: we condition only on the head

Example

- Here's the generative process:
  - Generate the head:
    $P(H|LHS) = P(VBD(dumped, VBD)|VP(dumped, VBD))$
  - Generate the left dependent:
    $P_L(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(sacks, NNS)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(NP(into, PP)|VP(dumped, VBD), VBD(dumped, VBD))$
  - Generate the right dependent:
    $P_R(\bullet|VP(dumped, VBD), VBD(dumped, VBD))$
- The rule probability is the product of these generative probabilities.
- **Horizontal Markovization**: we condition only on the head
- Collins parser also conditions on a "distance" of each constituent from the head.

Smoothing the Collins Parser

- Estimation is eased by factoring the rule probabilities, but smoothing is still needed.

$$\begin{aligned}\hat{P}(R_i(rw_i, rt_i)|p(i), hw, ht) = &\lambda_1 P_{mle}(R_i(rw_i, rt_i)|p(i), hw, ht) \\ &+ \lambda_2 P_{mle}(R_i(rw_i, rt_i)|p(i), ht) \\ &+ \lambda_3 P_{mle}(R_i(rw_i, rt_i)|p(i))\end{aligned}$$

- We set λ using Witten-Bell smoothing.
- Is it worth modeling bilexical dependencies?

The importance of bilexical dependencies

Back-off level	Number of accesses	Percentage
0	3,257,309	1.49
1	24,294,084	11.0
2	191,527,387	87.4
Total	219,078,780	100.0

- In general, bilexical probabilites are rarely available...

The importance of bilexical dependencies

Back-off level	Number of accesses	Percentage
0	3,257,309	1.49
1	24,294,084	11.0
2	191,527,387	87.4
Total	219,078,780	100.0

- In general, bilexical probabilites are rarely available...
- ...but they are active in 29% of the rules in **top-scoring** parses.

The importance of bilexical dependencies

Back-off level	Number of accesses	Percentage
0	3,257,309	1.49
1	24,294,084	11.0
2	191,527,387	87.4
Total	219,078,780	100.0

- In general, bilexical probabilites are rarely available...
- ...but they are active in 29% of the rules in **top-scoring** parses.
- Still, they don't seem to play a big role in accuracy (Bikel 2004).

The complexity of lexicalized parsing

- Straightforward lexicalized parsing is $\mathcal{O}(N^5 G)$, where
  - N is the length of the sentence
  - G is the state space, equal to g^3 (cubic in the number of original non-terminals, because we condition on the head and the parent), times V^3 (cubic in the vocabulary size, for the same reason)
- Exhaustive search is totally infeasible; Collins and Charniak both use beam search to eliminate unpromising nodes from the chart.
- Eisner and Satta (2000, etc) give ways to parse more restricted classes of bilexical grammars in $O(N^4)$ or $O(N^3)$

Summary of lexicalized parsing

- Lexicalized parsing resulted in substantial accuracy gains from our original PCFG:

Vanilla PCFG	72%
Parent-annotations	80%
Charniak (1997)	86%
Collins (1999)	87%

Summary of lexicalized parsing

- Lexicalized parsing resulted in substantial accuracy gains from our original PCFG:

Vanilla PCFG	72%
Parent-annotations	80%
Charniak (1997)	86%
Collins (1999)	87%

- But the explosion in the size of the grammar required elaborate smoothing techniques and made parsing slow.
- Treebank syntactic categories are too coarse,
  but lexicalized categories may be too fine.
  Is there a middle ground?

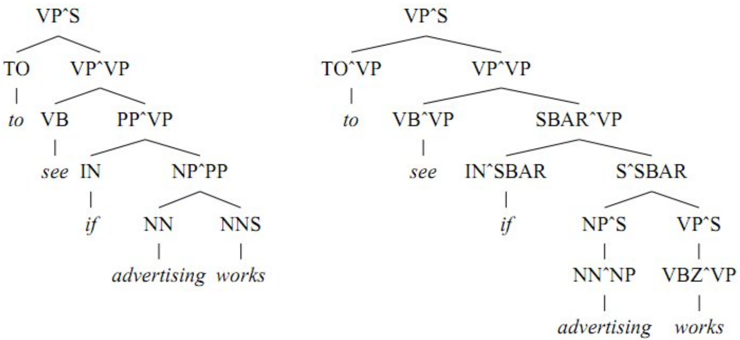
Accurate unlexicalized parsing (Klein and Manning 2003)

- Key idea is that the right level of linguistic detail is somewhere between treebank categories and individual words.
- For example, *on*/PP behaves differently from *of*/PP, but *cat*/N and *dog*/N do not.
- Approach: horizontal and vertical markovization, plus a series of linguistically-motivated splits to the Treebank categories.

Markovization

Vertical Order		Horizontal Markov Order				
		$h = 0$	$h = 1$	$h \leq 2$	$h = 2$	$h = \infty$
$v = 1$	No annotation	71.27 (854)	72.5 (3119)	73.46 (3863)	72.96 (6207)	72.62 (9657)
$v \leq 2$	Sel. Parents	74.75 (2285)	77.42 (6564)	77.77 (7619)	77.50 (11398)	76.91 (14247)
$v = 2$	All Parents	74.68 (2984)	77.42 (7312)	77.81 (8367)	77.50 (12132)	76.81 (14666)
$v \leq 3$	Sel. GParents	76.50 (4943)	78.59 (12374)	79.07 (13627)	78.97 (19545)	78.54 (20123)
$v = 3$	All GParents	76.74 (7797)	79.18 (15740)	79.74 (16994)	79.07 (22886)	78.72 (22002)

Example



Annotating the IN tag with its parent causes it to prefer SBAR complements, resolving this error.

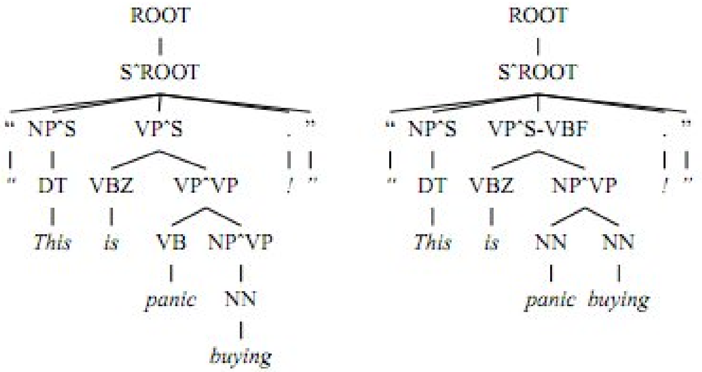
State-splitting

	Cumulative			Indiv.
Annotation	Size	F_1	Δ F_1	Δ F_1
Baseline ($v \leq 2, h \leq 2$)	7619	77.77	–	–
UNARY-INTERNAL	8065	78.32	0.55	0.55
UNARY-DT	8066	78.48	0.71	0.17
UNARY-RB	8069	78.86	1.09	0.43
TAG-PA	8520	80.62	2.85	2.52
SPLIT-IN	8541	81.19	3.42	2.12
SPLIT-AUX	9034	81.66	3.89	0.57
SPLIT-CC	9190	81.69	3.92	0.12
SPLIT-%	9255	81.81	4.04	0.15
TMP-NP	9594	82.25	4.48	1.07
GAPPED-S	9741	82.28	4.51	0.17
POSS-NP	9820	83.06	5.29	0.28
SPLIT-VP	10499	85.72	7.95	1.36
BASE-NP	11660	86.04	8.27	0.73
DOMINATES-V	14097	86.91	9.14	1.42
RIGHT-REC-NP	15276	87.04	9.27	1.94

Examples:

- BASE-NP: non-recursive NPs
- SPLIT-CC: distintinguish *and* and *but* from other CCs

Example



The original parse assigned a VP complement to a finite verb (is). Splitting the VP tag into finite and infinitival categories resolves this error.

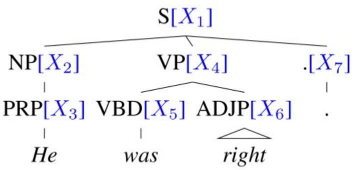
Automatic state-splitting

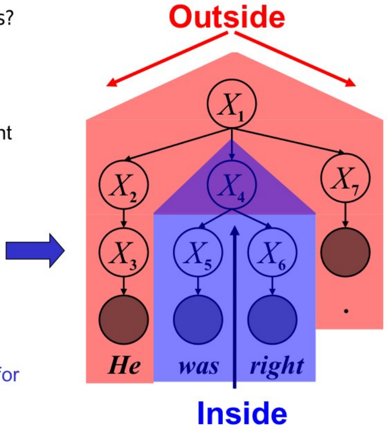
- The Klein and Manning unlexicalized parser requires substantial engineering.
- It would be a lot of work to apply this to a new language.
- Can we split the Treebank syntactic categories automatically?

State splitting through hidden variables (Petrov and Klein, 2007)

Can you automatically find good symbols?

- Brackets are known
- Base categories are known
- Induce subcategories
- Clever split/merge category refinement



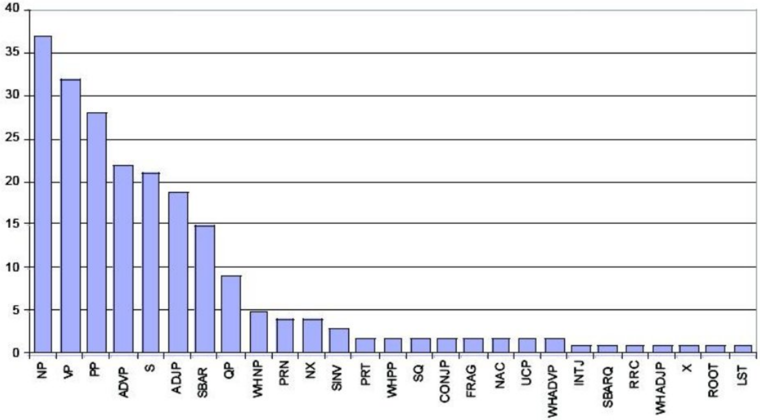


EM algorithm, like Forward-Backward for HMMs, but constrained by tree.

State splitting through hidden variables

- We'll talk more about latent variables next week.
- For now, think of it as structured clustering:
  - Assign a random subcategory to each node.
  - Learn a PCFG.
  - Apply the PCFG to relabel the nodes
    - subject to constraints of original annotations: VP3 can be relabeled as VP7, but not as an NP
  - Repeat

Number of phrasal subcategories



Examples

- Proper Nouns (NNP):

NNP-14	Oct.	Nov.	Sept.
NNP-12	John	Robert	James
NNP-2	J.	E.	L.
NNP-1	Bush	Noriega	Peters
NNP-15	New	San	Wall
NNP-3	York	Francisco	Street

- Personal pronouns (PRP):

PRP-0	It	He	I
PRP-1	it	he	they
PRP-2	it	them	him

Accuracy

Vanilla PCFG	72%
Parent-annotations	80%
Lexicalized (Charniak 1997)	86%
Lexicalized (Collins 1999)	87%
Lexicalized (Charniak 2000)	90.1%
State-splitting (Petrov and Klein 2007)	90.6%

Discriminative parsing

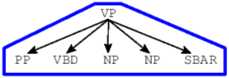
- Generative parsers assume observations are conditionally independent given the label.
- This prohibits redundant features like morphology and word clusters
- This made a big difference in sequence labeling (25% error reduction).
- Can it help in parsing?

Reranking

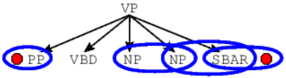
- Key idea: generate an N-best list of parses, learn a *ranking* function to score them (Collins, 2002)
- Advantage: can include arbitrary features.
- Can be as simple as perceptron
  - **Learning**
    $$w_n \leftarrow w_{n-1} + \eta(f(t,s) - f(\hat{t},s)) \quad (1)$$
    where $f(t,s)$ are the features of the correct parse and $f(\hat{t},s)$ are the features of the best-scoring parse.
  - **Decoding**: produce K parses from the generative model, return $\arg\max_k \mathbf{w}^\top f(t_k, s)$

Features (Collins and Koo 2003)

Rules These include all context-free rules in the tree, for example VP -> PP VBD NP NP SBAR.

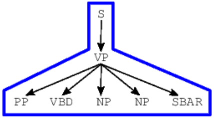


Bigrams These are adjacent pairs of non-terminals to the left and right of the head. As shown, the example rule would contribute the bigrams (Right,VP,NP,NP), (Right,VP,NP,SBAR), (Right,VP,SBAR,STOP), to the right of the head, and (Left,VP,PP,STOP) to the left of the head.

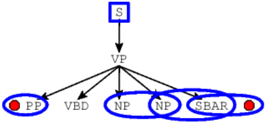


Features (Collins and Koo 2003)

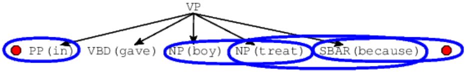
Grandparent Rules Same as **Rules**, but also including the non-terminal above the rule.



Grandparent Bigrams Same as **Bigrams**, but also including the non-terminal above the bigrams.

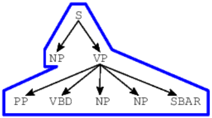


Lexical Bigrams Same as **Bigrams**, but with the lexical heads of the two non-terminals also included.

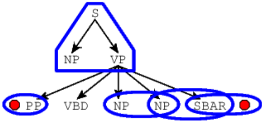


Features (Collins and Koo 2003)

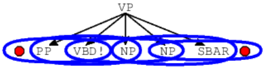
Two-level Rules Same as **Rules**, but also including the entire rule above the rule.



Two-level Bigrams Same as **Bigrams**, but also including the entire rule above the rule.

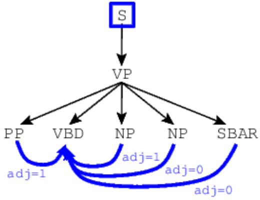


Trigrams All trigrams within the rule. The example rule would contribute the trigrams (VP,STOP,PP,VBD!), (VP,PP,VBD!,NP), (VP, VBD!,NP,NP), (VP,NP,NP,SBAR) and (VP,NP, SBAR,STOP) (! is used to mark the head of the rule).

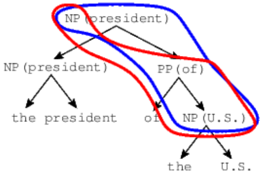


Features (Collins and Koo 2003)

Head-Modifiers All head-modifier pairs, with the grandparent non-terminal also included. An `adj` flag is also included, which is 1 if the modifier is adjacent to the head, 0 otherwise. As an example, say the non-terminal dominating the example rule is `S`. The example rule would contribute `(Left,S,VP,VBD,PP,adj=1)`, `(Right,S,VP,VBD,NP,adj=1)`, `(Right,S,VP,VBD,NP,adj=0)`, and `(Right,S,VP,VBD,SBAR,adj=0)`.



PPs Lexical trigrams involving the heads of arguments of prepositional phrases. The example shown at right would contribute the trigram `(NP,NP,PP,NP,president,of,U.S.)`, in addition to the relation `(NP,NP,PP,NP,of,U.S.)` which ignores the headword of the constituent being modified by the PP. The three non-terminals (for example `NP, NP, PP`) identify the parent of the entire phrase, the non-terminal of the head of the phrase, and the non-terminal label for the PP.



Accuracy

Vanilla PCFG	72%
Parent-annotations	80%
Lexicalized (Charniak 1997)	86%
Lexicalized (Collins 1999)	87%
Lexicalized (Charniak 2000)	90.1%
State-splitting (Petrov and Klein 2007)	90.6%
Reranking (Charniak and Johnson 2005)	91.0%

Globally-normalized conditional models for parsing

The CFG-CRF model (Finkel et al., 2008):

$$P(t|s;\theta) = \frac{1}{Z_s} \prod_{r \in t} \phi(r|s;\theta)$$

$$Z_s = \sum_{t' \in \tau(s)} \prod_{r \in t'} \phi(r|s;\theta)$$

- Each parse t is a collection of productions $\{r\}$.
- Each production has a non-negative potential $\phi(r|s;\theta) = e^{\boldsymbol{\theta}^\top \mathbf{f}(r,s)}$.
- The unnormalized score for a parse is the product of its potentials.
- The *partition function* Z_s is the sum of the scores for all possible parses for a sentence s.

Decoding

- If the features are local in t, we can use CKY to decode.
- Features need not be local in s.
  (just like in discriminative sequence models)
- Just like CKY, but you multiply potentials ϕ rather than probabilities.
- We need only the unnormalized score $\prod_{r \in t} \phi(r|s;\theta)$.

Features

- standard PCFG stuff, with and without parent annotation (no need for complicated smoothing!)
- lexicon features over words and tags
  (including prev word and next word, and unknown word classes)
- bigrams and trigrams of *word classes* under each subtree

Learning

Just like logistic regression:

$$\mathcal{L} = \left[\sum_{(t,s)\in\mathcal{D}} \left(\sum_{r\in t} \sum_{i} \theta_i f_i(r,s) \right) - Z_s \right] + \sum_i \frac{\theta_i^2}{2\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \left[\sum_{(t,s)\in\mathcal{D}} \left(\sum_{r\in t} \sum_{i} f_i(r,s) \right) - E_\theta[f_i|s] \right] + \frac{\theta_i}{\sigma^2}$$

- compute Z_s using the inside algorithm
- compute $E_\theta[f_i|s]$ using inside-outside

Learning

Just like logistic regression:

$$\mathcal{L} = \left[\sum_{(t,s)\in\mathcal{D}} \left(\sum_{r\in t} \sum_{i} \theta_i f_i(r,s) \right) - Z_s \right] + \sum_i \frac{\theta_i^2}{2\sigma^2}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_i} = \left[\sum_{(t,s)\in\mathcal{D}} \left(\sum_{r\in t} \sum_{i} f_i(r,s) \right) - E_\theta[f_i|s] \right] + \frac{\theta_i}{\sigma^2}$$

- compute Z_s using the inside algorithm
- compute $E_\theta[f_i|s]$ using inside-outside
- But unfortunately, $\mathcal{O}(N^3 G)$ is still too slow.

Tricks

Chart prefiltering:

- Run a non-probabilistic CFG to prune away productions which do not lead to any valid parse.
- This saves time during inside-outside.

Tricks

Chart prefiltering:

- Run a non-probabilistic CFG to prune away productions which do not lead to any valid parse.
- This saves time during inside-outside.

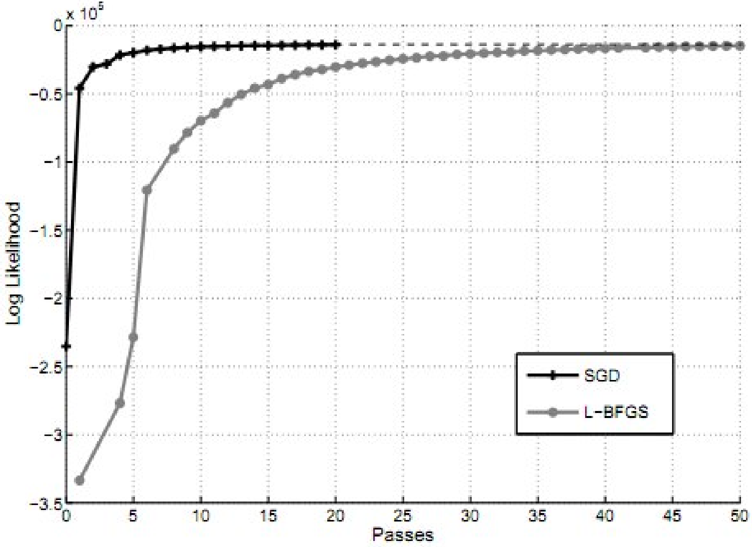
Parallelization via stochastic gradient descent:

- Let $\hat{\mathcal{L}}(\mathcal{D}_b^{(i)}; \boldsymbol{\theta})$ equal the likelihood computed from a "minibatch" of b examples.
- Then we can approximate the gradient, $\nabla \mathcal{L}(\mathcal{D}; \boldsymbol{\theta}) \approx \sum_i \nabla \mathcal{L}(\mathcal{D}_b^{(i)}; \boldsymbol{\theta})$.
- We can then parallelize the minibatches, and make stochastic gradient updates,

$$\theta_{k+1} = \theta_k - \eta_k \nabla \mathcal{L}(\mathcal{D}_b^{(i)}; \boldsymbol{\theta}),$$

where η_k is the learning rate after the k^{th} update.

Stochastic gradient



Results

Vanilla PCFG	72%
Parent-annotations	80%
Lexicalized (Charniak 1997)	86%
Lexicalized (Collins 1999)	87%
State-splitting (Petrov and Klein 2007)	90.6%
Reranking (Charniak and Johnson 2005)	91.0%
CFG-CRF (Finkel et al 2008)	89.0

Results

Vanilla PCFG	72%
Parent-annotations	80%
Lexicalized (Charniak 1997)	86%
Lexicalized (Collins 1999)	87%
State-splitting (Petrov and Klein 2007)	90.6%
Reranking (Charniak and Johnson 2005)	91.0%
CFG-CRF (Finkel et al 2008)	89.0

- better than basic generative parsers, worse than reranking
- The key advantage of this approach is the simplicity:
  no need for complicated smoothing or backoff, just throw redundant information at the learner and let it sort things out
- Room for better features?

Results

Vanilla PCFG	72%
Parent-annotations	80%
Lexicalized (Charniak 1997)	86%
Lexicalized (Collins 1999)	87%
State-splitting (Petrov and Klein 2007)	90.6%
Reranking (Charniak and Johnson 2005)	91.0%
CFG-CRF (Finkel et al 2008)	89.0

- better than basic generative parsers, worse than reranking
- The key advantage of this approach is the simplicity: no need for complicated smoothing or backoff, just throw redundant information at the learner and let it sort things out
- Room for better features?
- ... or better models? An alternative CRF based on tree-adjoining grammar, scored 91.1 (Carreras et al, 2008)

Recap

- A big part of parsing research has been figuring out the right level of description:
  - Unlexicalized PCFGs on treebank categories are too coarse.
  - Lexicalized parsers start with very rich probability models, and then apply smoothing for tractability.
  - State-splitting approaches start with the treebank categories and split as needed.
- Reranking approaches can learn rich feature representations, but can only apply them to a limited set of possible parses.
- Globally-normalized conditional parsers learn feature-based models and apply them to decode over the entire space of possible parses.