Discriminative approaches to Statistical Parsing

Mark Johnson

Brown University

University of Tokyo, 2004

Joint work with Eugene Charniak (Brown) and Michael Collins (MIT)

Supported by NSF grants LIS 9720368 and IIS0095940

Talk outline

- A typology of approaches to parsing
- Applications of parsers
- Representations and features of statistical parsers
- Estimation (training) of statistical parsers
  - maximum likelihood (generative) estimation
  - maximum *conditional* likelihood (discriminative) estimation
- Experiments with a discriminatively trained reranking parser
- Advantages and disadvantages of generative and discriminative training
- Conclusions and future work

Grammars and parsing

- A *(formal) language* is a set of strings
  - For most practical purposes, human languages are infinite sets of strings
  - In general we are interested in the *mapping from surface form to meaning*
- A grammar is a finite description of a language
  - Usually assigns each string in a language a *description* (e.g., parse tree, semantic representation)
- Parsing is the process of characterizing (recovering) the descriptions of a string
- Most grammars of human languages are either *manually constructed* or *extracted automatically from an annotated corpus*
  - Linguistic expertise is necessary for both!

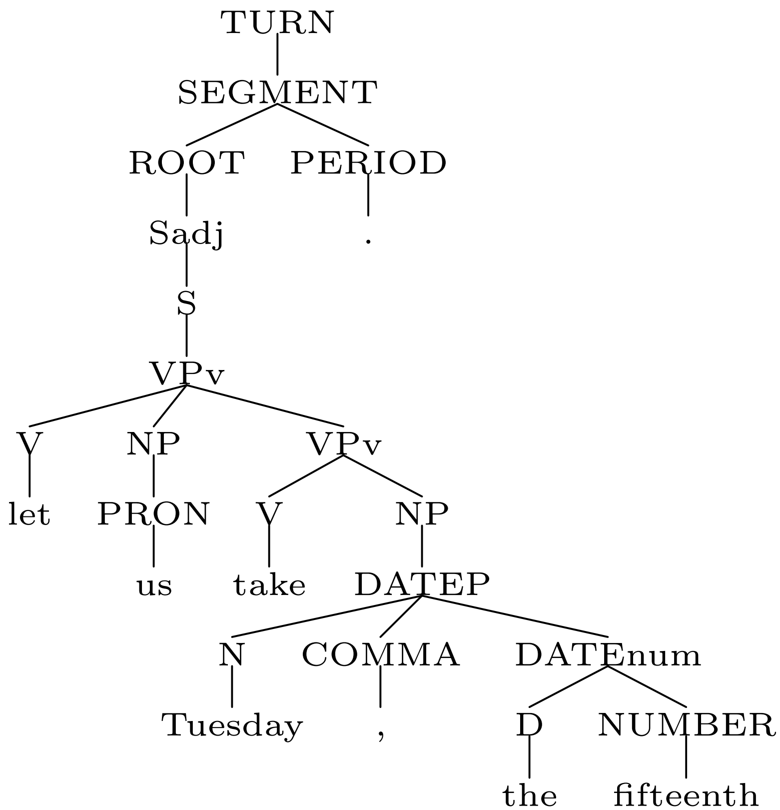
Manually constructed grammars

Examples: Lexical-functional grammar (LFG), Head-driven phrase-structure grammar (HPSG), Tree-adjoining grammar (TAG)

- Linguistically inspired
  - Deals with linguistically interesting phenomena
  - Ignores boring (or difficult!) but frequent constructions
  - Often explicitly models the form-meaning mapping
- Each theory usually has its own kind of representation

  ⇒ Difficult to compare different approaches
- *Constructing broad-coverage grammars is hard and unrewarding*
- Probability distributions can be defined over their representations
- Often involve *long-distance constraints*

  ⇒ Computationally expensive and difficult

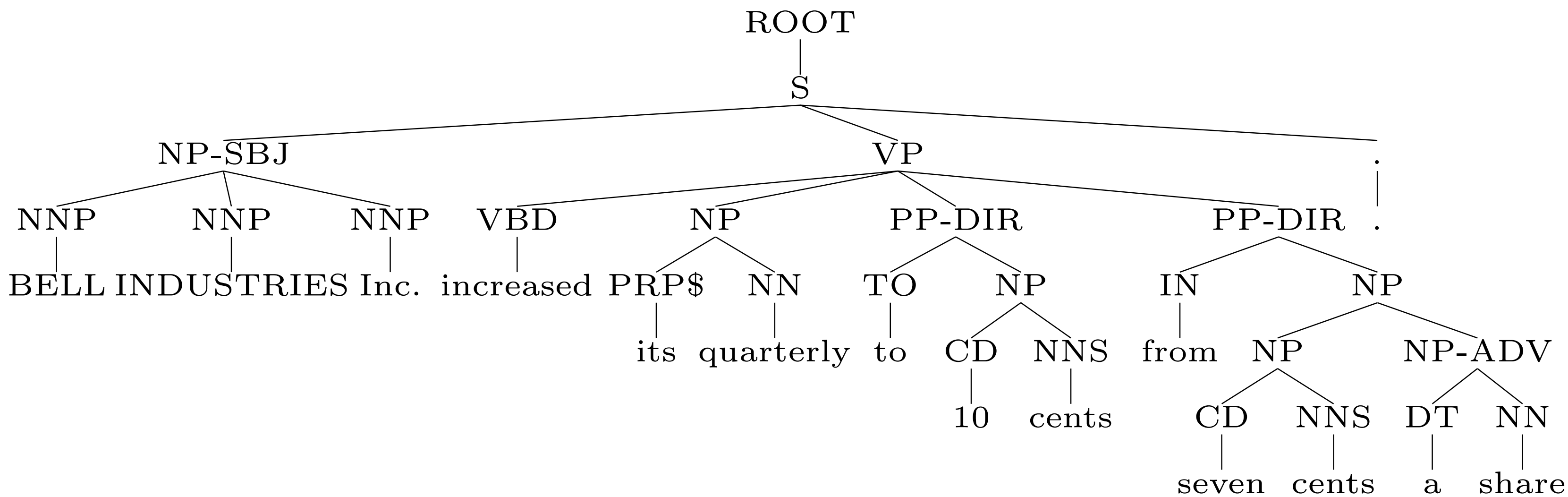



```
[ SENTENCE_ID  BAC002_E
  { [ OBJ     9[ ANIM +
                 CASE ACC
                 NUM PL
                 PERS 1
                 PRED PRO
                 PRON-FORM WE
                 PRON-TYPE PERS ]
      PASSIVE −
      PRED  LET⟨2,10⟩9
      STMT-TYPE IMPERATIVE
      SUBJ    2[ PERS 2
                 PRED PRO
                 PRON-TYPE NULL ]
      TNS-ASP [ MOOD IMPERATIVE ]
      XCOMP  10[ OBJ  13[ ANIM −
                          APP  [ NTYPE [ NUMBER ORD
                                         TIME DATE ]
                                 NUM SG
                                 PRED fifteen
                                 SPEC  [ SPEC-FORM THE
                                         SPEC-TYPE DEF ] ]
                          CASE ACC
                          GEND NEUT
                          NTYPE [ GRAIN COUNT
                                  PROPER DATE
                                  TIME DAY ]
                          NUM SG
                          PERS 3
                          PRED  TUESDAY ]
                 PASSIVE −
                 PRED  TAKE⟨9,13⟩
                 SUBJ [9] ] ] } ]
```

Corpus-derived grammars

- Grammar is extracted automatically from a large linguistically annotated corpus
  - Focuses on frequently occuring constructions
  - Only models phenomena that can be (easily) annotated
  - Typically ignores semantics and most of the rich details of linguistic theories
- Different models extracted from the same corpus can usually be compared
- *Constructing corpora is hard, unrewarding work*
- *Generative models* usually only involve local constraints
  - Dynamic programming possible, but usually involves heuristic search

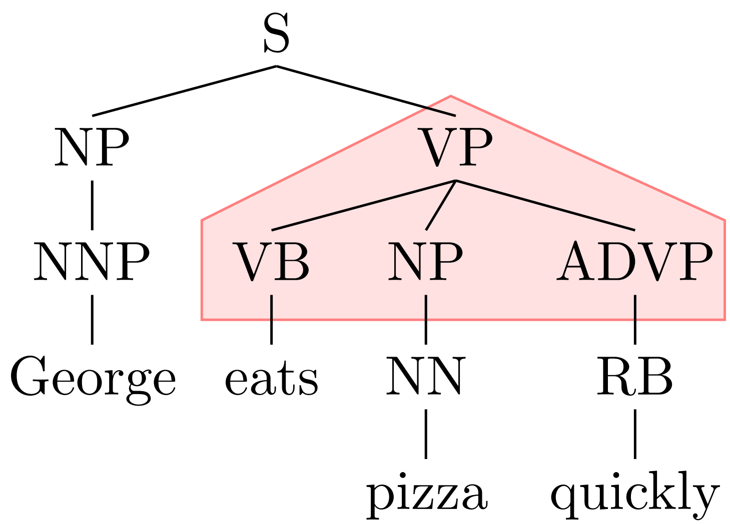
Sample Penn treebank tree



Applications of (statistical) parsers

1. Applications that use syntactic *parse trees*
   - information extraction
   - (short answer) question answering
   - summarization
   - machine translation
2. Applications that use the *probability distribution* over strings or trees (parser-based language models)
   - speech recognition and related applications
   - machine translation

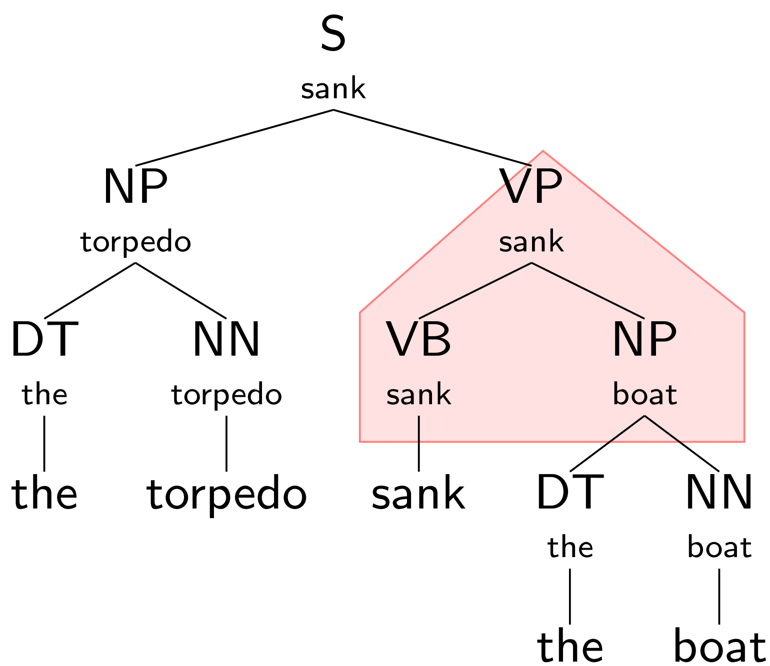
PCFG representations and features



$$0.14\text{: VP} \rightarrow \text{VB NP ADVP}$$

- Probabilistic context-free grammars (PCFGs) associate a *rule probability* $p(r)$ with each rule $\Rightarrow$ features are *local trees*
- Probability of a tree y is $\mathrm{P}(y) = \prod_{r \in y} p(r) = \prod_r p(r)^{f_r(y)}$ where $f_r(y)$ is the number of times r appears in y
- Probability of a string x is $\mathrm{P}(x) = \sum_{y \in \mathcal{Y}(x)} \mathrm{P}(y)$

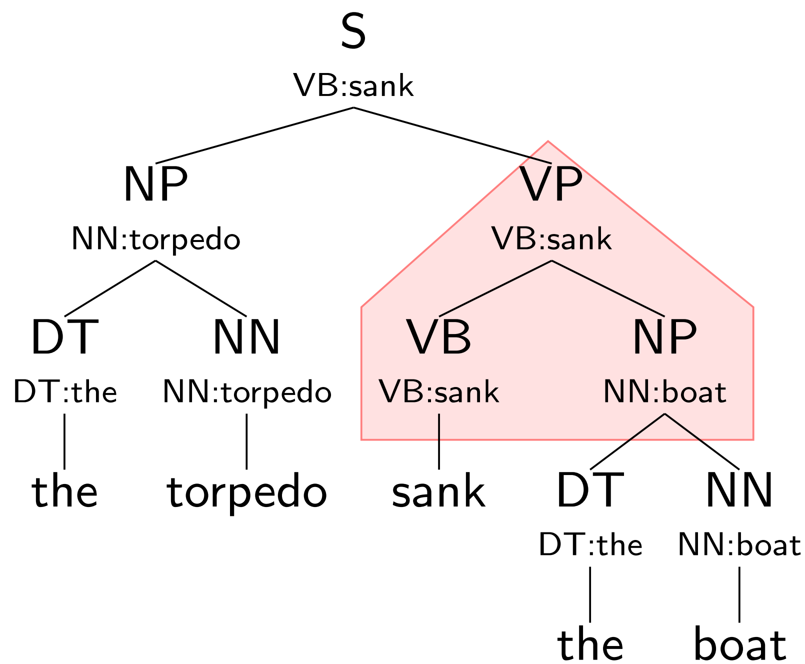
Lexicalized PCFGs



$$0.02: \ \underset{\text{sank}}{\text{VP}} \rightarrow \underset{\text{sank}}{\text{VB}} \ \underset{\text{boat}}{\text{NP}}$$

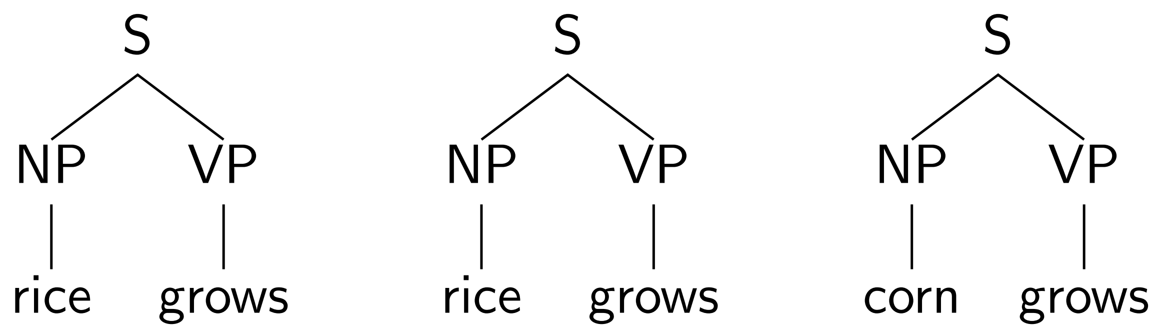
- *Head annotation* captures *subcategorization* and *head-to-head dependencies*
- Sparse data is a serious problem: smoothing is essential!

Modern (generative) statistical parsers

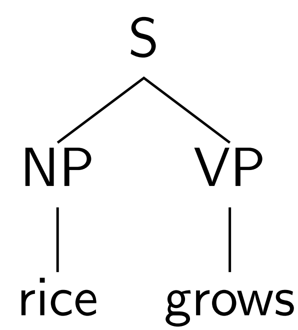


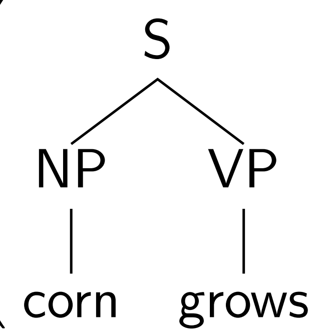
- Generates a tree via a very large number of small steps (generates NP, then NN, then boat)
- Each step in this branching process conditions on a large number of (already generated) variables
- *Sparse data is the major problem: smoothing is essential!*

Estimating PCFGs from visible data

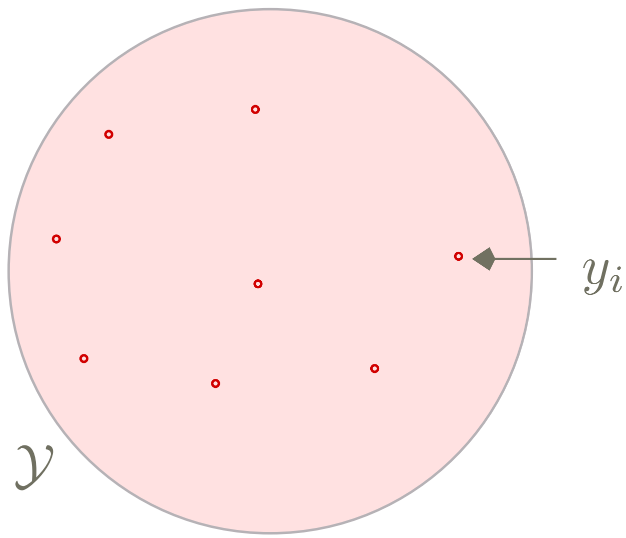


Rule	Count	Rel Freq
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	$2/3$
NP $\rightarrow$ corn	1	$1/3$
VP $\rightarrow$ grows	3	1

P () $= 2/3$

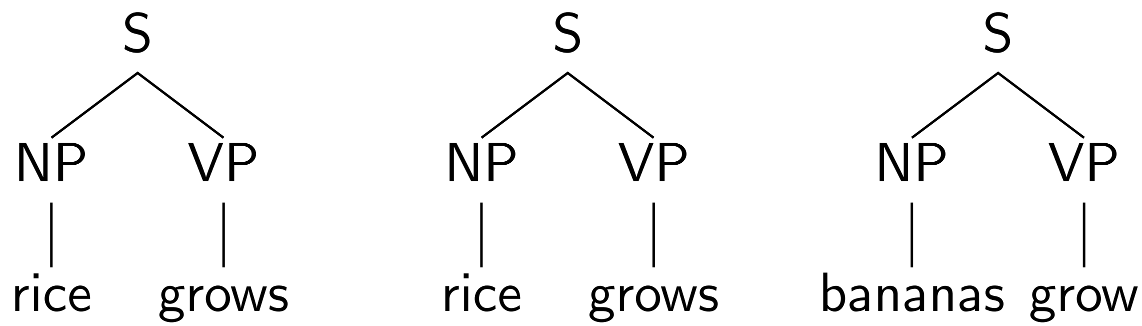
P () $= 1/3$

Why is the PCFG MLE so easy to compute?

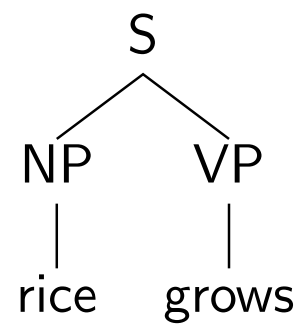


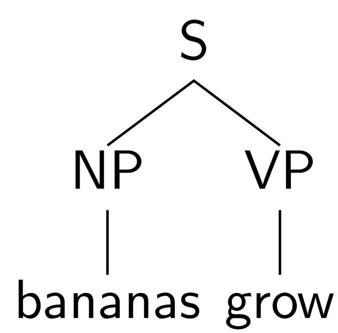
- Visible training data $D = (y_1, \ldots, y_n)$, where y_i is a parse tree
- The MLE is $\hat{p} = \operatorname{argmax}_p \prod_{i=1}^n \mathrm{P}_p(y_i)$
- It is easy to compute because PCFGs are always normalized, i.e., $Z = \sum_{y \in \mathcal{Y}} \prod_r p(r)^{f_r(y)} = 1$,
  where $\mathcal{Y}$ is the set of all trees generated by the grammar

Non-local constraints and the PCFG MLE



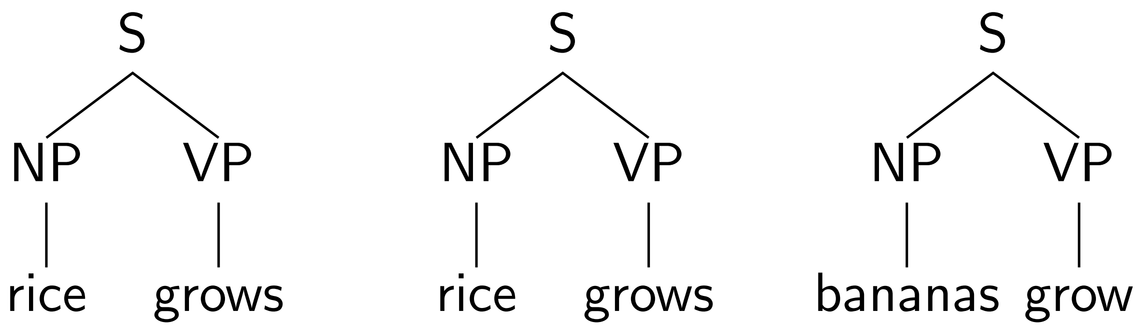
rule	count	rel freq
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	$2/3$
NP $\rightarrow$ bananas	1	$1/3$
VP $\rightarrow$ grows	2	$2/3$
VP $\rightarrow$ grow	1	$1/3$

P () $= 4/9$

P () $= 1/9$

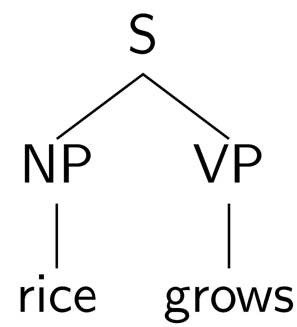
$Z = 5/9$

Renormalization



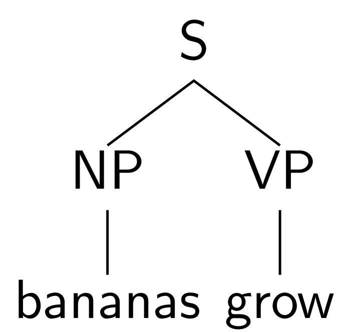
rule	count	rel freq
S $\rightarrow$ NP VP	3	1
NP $\rightarrow$ rice	2	$2/3$
NP $\rightarrow$ bananas	1	$1/3$
VP $\rightarrow$ grows	2	$2/3$
VP $\rightarrow$ grow	1	$1/3$

P(



) $=$ $4/9$ $4/5$

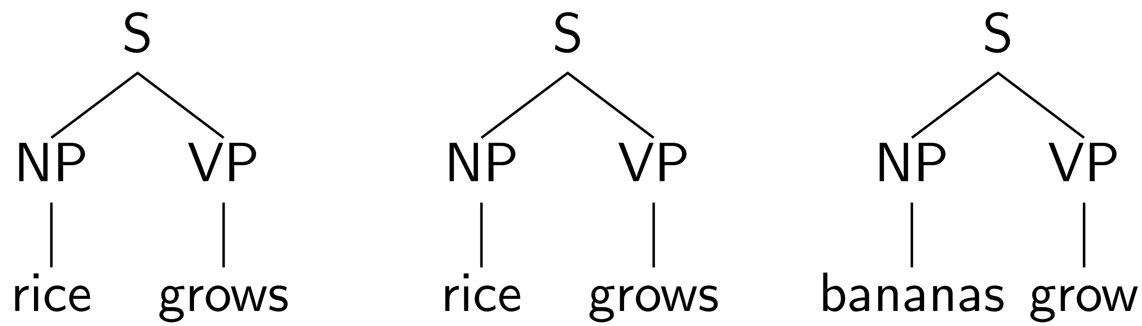
P(



) $=$ $1/9$ $1/5$

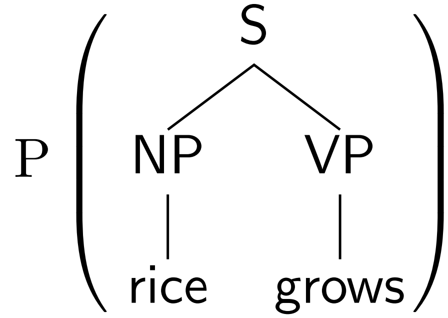
$Z = 5/9$

Other values do better!

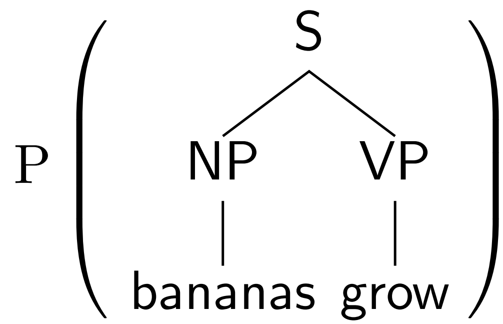


rule	count	rel freq
S → NP VP	3	1
NP → rice	2	2/3
NP → bananas	1	1/3
VP → grows	2	1/2
VP → grow	1	1/2

(Abney 1997)



$= \quad 2/6 \quad 2/3$



$= \quad 1/6 \quad 1/3$

$Z = 3/6$

Make dependencies local – GPSG-style

rule	count	rel freq
$\text{S} \rightarrow \underset{+\text{singular}}{\text{NP}}\ \underset{+\text{singular}}{\text{VP}}$	2	2/3
$\text{S} \rightarrow \underset{+\text{plural}}{\text{NP}}\ \underset{+\text{plural}}{\text{VP}}$	1	1/3
$\underset{+\text{singular}}{\text{NP}} \rightarrow \text{rice}$	2	1
$\underset{+\text{plural}}{\text{NP}} \rightarrow \text{bananas}$	1	1
$\underset{+\text{singular}}{\text{VP}} \rightarrow \text{grows}$	2	1
$\underset{+\text{plural}}{\text{VP}} \rightarrow \text{grow}$	1	1

$$P\left(\begin{array}{c} \text{S} \\ \begin{array}{cc} \underset{+\text{singular}}{\text{NP}} & \underset{+\text{singular}}{\text{VP}} \\ | & | \\ \text{rice} & \text{grows} \end{array} \end{array}\right) = 2/3$$

$$P\left(\begin{array}{c} \text{S} \\ \begin{array}{cc} \underset{+\text{plural}}{\text{NP}} & \underset{+\text{plural}}{\text{VP}} \\ | & | \\ \text{bananas} & \text{grow} \end{array} \end{array}\right) = 1/3$$

Maximum entropy or log linear models

- $\mathcal{Y}$ = set of syntactic structures (not necessarily trees)
- $f_j(y)$ = number of occurences of jth feature in $y \in \mathcal{Y}$
  (these features need not be conventional linguistic features)
- w_j are "feature weight" parameters

$$\begin{aligned}
S_w(y) &= \sum_{j=1}^{m} w_j f_j(y) \\
V_w(y) &= \exp S_w(y) \\
Z_w &= \sum_{y \in \mathcal{Y}} V_w(y) \\
\mathrm{P}_w(y) &= \frac{V_w(y)}{Z_w} = \frac{1}{Z_w} \exp \sum_{j=1}^{m} w_j f_j(y) \\
\log \mathrm{P}_\lambda(y) &= \sum_{j=1}^{m} w_j f_j(y) - \log Z_w
\end{aligned}$$

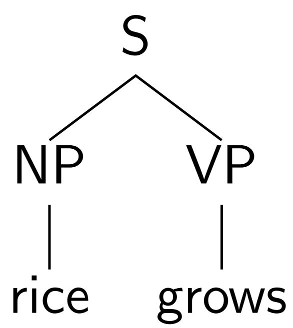
PCFGs are log-linear models

$\mathcal{Y}$ = set of all trees generated by grammar G

$f_j(y)$ = number of times the jth rule is used in $y \in \mathcal{Y}$

$p(r_j)$ = probability of jth rule in G

Choose $w_j = \log p(r_j)$, so $p(r_j) = \exp w_j$

$f\Bigg($ 

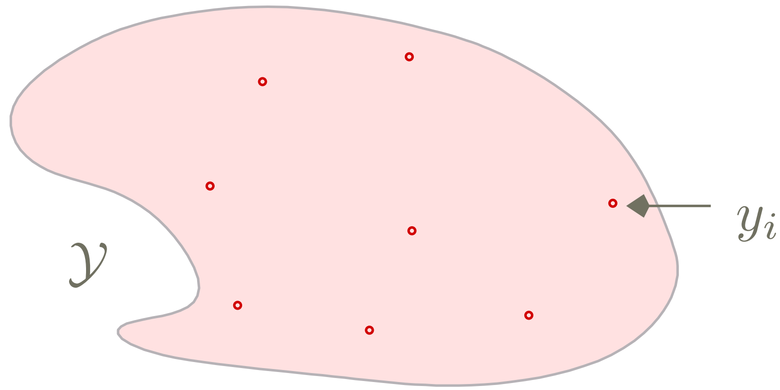
$\Bigg) = [\underbrace{1}_{\text{S}\rightarrow\text{NP VP}}, \underbrace{1}_{\text{NP}\rightarrow\text{rice}}, \underbrace{0}_{\text{NP}\rightarrow\text{bananas}}, \underbrace{1}_{\text{VP}\rightarrow\text{grows}}, \underbrace{0}_{\text{VP}\rightarrow\text{grow}}]$

$$\mathrm{P}_w(y) = \prod_{j=1}^{m} p(r_j)^{f_j(y)} = \prod_{j=1}^{m} (\exp w_j)^{f_j(y)} = \exp(\sum_{j=1}^{m} w_j f_j(\omega))$$

So *a PCFG is just a log linear model* with $Z = 1$.

Max likelihood estimation of log linear models

Visible training data $D = (y_1, \ldots, y_n)$, where $y_i \in \mathcal{Y}$ is a tree



$$
\begin{aligned}
\hat{w} &= \operatorname*{argmax}_{w} L_D(w), \text{where} \\
\log L_D(w) &= \sum_{i=1}^{n} \log \mathrm{P}_w(y_i) = \sum_{i=1}^{n} (S_w(y_i) - \log Z_w)
\end{aligned}
$$

- In general no closed form solution $\Rightarrow$ optimize $\log L_D(w)$ numerically.
- Calculating Z_w involves *summing over all parses of all strings*

$\Rightarrow$ *computationally intractible* (Abney suggests Monte Carlo)

Summary so far

All dependencies are local or context-free:

- if features have *"context free" branching structure* (i.e., rules) then *partition function Z can be calculated analytically*

$\Rightarrow$ MLE has a simple analytic form (relative frequency)

Structures exhibit non-local constraints:

- with *non-local constraints, MLE is in general not relative frequency*
- Usually no analytic form for Z

$\Rightarrow$ no analytic solution for the MLE

$\Rightarrow$ *no reason to only use local tree rule features*
(i.e., the $f_j(y)$ can be any easily computable function of y)

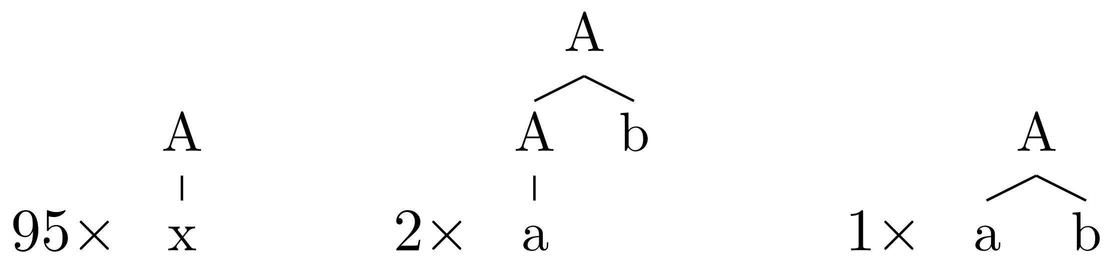
- If it is necessary to enumerate $\mathcal{Y}$ to calculate Z then *MLE is infeasible*

Conditional Likelihood and Discriminative training

Given training data $D = ((x_1, y_1), \ldots, (x_n, y_n))$ of strings x_i and corresponding parse y_i:

- The PCFG MLE optimizes $L_D(w) = \mathrm{P}_w(x_1, y_1) \ldots \mathrm{P}_w(x_n, y_n)$
- The PCFG MLE is a *generative model* that models the *distribution of strings* $\mathrm{P}(x)$ as well as *trees given strings* $\mathrm{P}(y|x)$
- The conditional distribution $\mathrm{P}(y|x)$ is important for parsing, but the marginal distribution $\mathrm{P}(x)$ is not
- Generative models "waste" some of their parameters to model the marginal distribution $\mathrm{P}(x)$
- Optimize *conditional likelihood* $L'_D(w) = \mathrm{P}_w(y_1|x_1) \ldots \mathrm{P}_w(y_n|x_n)$

Generative vs discriminative training

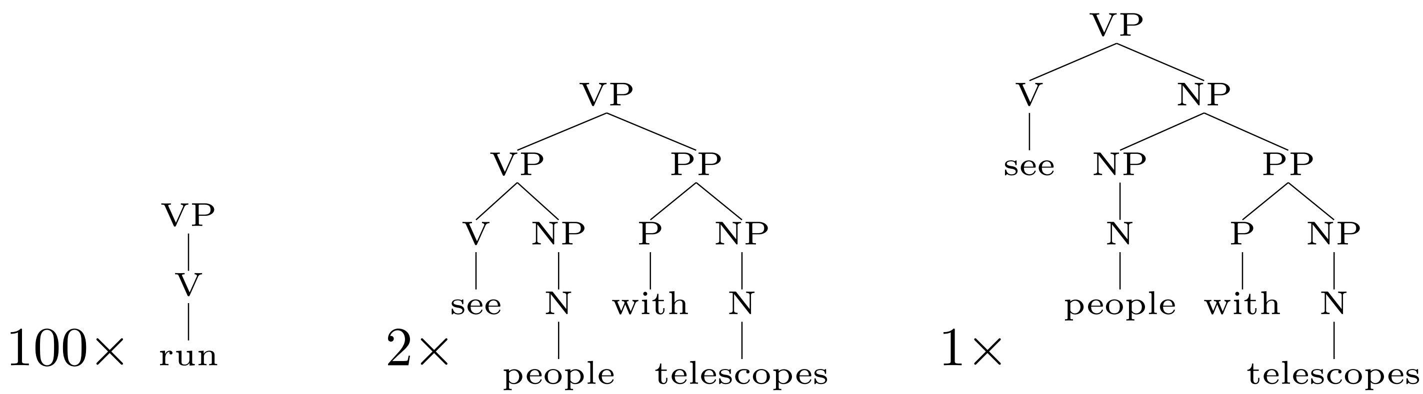


95/100   2/100×2/100   1/100

Rule	**count**	**rel freq**	**rel freq**
A → x	95	95/100	69/100
A → A b	2	2/100	1/10
A → a	2	2/100	2/10
A → a b	1	1/100	1/100

- When the PCFG independence assumptions are violated, the MLE may not accurately model $\mathrm{P}(y|x)$

Linguistic example of discriminative training



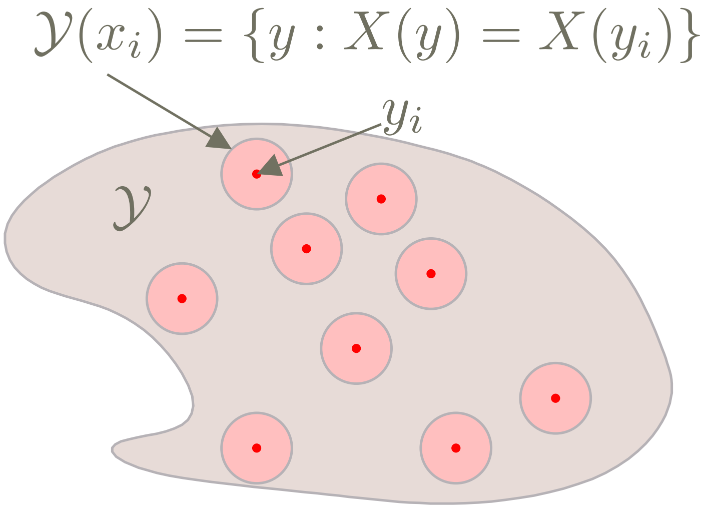
$\ldots \times 2/105 \times \ldots$ $\qquad$ $\ldots \times 1/7 \times \ldots$

$\ldots \times 2/7 \times \ldots$ $\qquad$ $\ldots \times 1/7 \times \ldots$

Rule	**count**	**rel freq**	**rel freq**
VP $\rightarrow$ V	100	100/105	4/7
VP $\rightarrow$ V NP	3	3/105	1/7
VP $\rightarrow$ VP PP	2	2/105	**2/7**
NP $\rightarrow$ N	6	6/7	6/7
NP $\rightarrow$ NP PP	1	1/7	**1/7**

Conditional estimation for log linear models

The *pseudo-likelihood* of w is the *conditional probability* of the *hidden part* (syntactic structure) w given its *visible part* (yield or terminal string) $x = X(y)$ (Besag 1974)



$$\begin{aligned} \widehat{w} &= \underset{\lambda}{\operatorname{argmax}} \, \mathrm{PL}_D(w) \\ \mathrm{PL}_D(w) &= \prod_{i=1}^{n} \mathrm{P}_\lambda(y_i|x_i) \\ \mathrm{P}_w(y|x) &= \frac{V_w(y)}{Z_w(x)} \end{aligned}$$

$$V_w(y) = \exp \sum_j w_j f_j(y) \qquad Z_w(x) = \sum_{y' \in \mathcal{Y}(x)} V_w(y')$$

Conditional ML estimation

- The pseudo-partition function $Z_w(x)$ is *much easier to compute* than the partition function Z_w
  - Z_w requires a sum over $\mathcal{Y}$
  - $Z_w(x)$ requires a sum over $\mathcal{Y}(x)$ (parses of x)
- *Maximum likelihood* estimates full joint distribution
  - learns $\mathrm{P}(x)$ and $\mathrm{P}(y|x)$
- *Conditional ML* estimates a conditional distribution
  - learns $\mathrm{P}(y|x)$ but not $\mathrm{P}(x)$
  - conditional distribution is what you need for parsing
  - cognitively more plausible?
- Conditional estimation requires labelled training data: no obvious EM extension

Conditional estimation

	Correct parse's features	**All other parses' features**
sentence 1	$[1, 3, 2]$	$[2, 2, 3]$ $[3, 1, 5]$ $[2, 6, 3]$
sentence 2	$[7, 2, 1]$	$[2, 5, 5]$
sentence 3	$[2, 4, 2]$	$[1, 1, 7]$ $[7, 2, 1]$
...	...	...

- Training data is *fully observed* (i.e., parsed data)
- Choose w to maximize (log) likelihood of *correct* parses relative to other parses
- Distribution of *sentences* is ignored
- *Nothing is learnt from unambiguous examples*
- Other kinds of discriminative learners can also train from this data

Pseudo-constant features are uninformative

	Correct parse's features	**All other parses' features**
sentence 1	$[1, 3, \mathit{2}]$	$[2, 2, \mathit{2}]$ $[3, 1, \mathit{2}]$ $[2, 6, \mathit{2}]$
sentence 2	$[7, 2, \mathit{5}]$	$[2, 5, \mathit{5}]$
sentence 3	$[2, 4, \mathit{4}]$	$[1, 1, \mathit{4}]$ $[7, 2, \mathit{4}]$
...	...	...

- *Pseudo-constant features* are identical within every set of parses
- They contribute the same constant factor to each parses' likelihood
- They do not distinguish parses of any sentence $\Rightarrow$ irrelevant

Pseudo-maximal features ⇒ unbounded $\widehat{w_j}$

	Correct parse's features	**All other parses' features**
sentence 1	$[1, \mathit{3}, 2]$	$[2, \mathit{3}, 4]$ $[3, \mathit{1}, 1]$ $[2, \mathit{1}, 1]$
sentence 2	$[2, \mathit{7}, 4]$	$[3, \mathit{7}, 2]$
sentence 3	$[2, \mathit{4}, 4]$	$[1, \mathit{1}, 1]$ $[1, \mathit{2}, 4]$

- A *pseudo-maximal feature* always reaches its maximum value within a parse on the correct parse
- If f_j is pseudo-maximal, $\widehat{w_j} \to \infty$ (hard constraint)
- If f_j is pseudo-minimal, $\widehat{w_j} \to -\infty$ (hard constraint)

Regularization

- f_j is pseudo-maximal over training data $\not\Rightarrow$ f_j is pseudo-maximal over all $\mathcal{Y}$ (sparse data)
- With many more features than data, log-linear models can over-fit
- Regularization: add *bias* term to ensure $\widehat{w}$ is finite and small
- In these experiments, the regularizer is a polynomial penalty term

$$\widehat{w} \;=\; \operatorname*{argmax}_{w} \log \mathrm{PL}_D(w) - c \sum_{j=1}^{m} |w_j|^p$$

($p = 2$ gives a Gaussian prior).

Conditional estimation of PCFGs

- MCLE involves maximizing a complex non-linear function
  - conjugate gradient (iterative optimization)
  - each iteration involves summing over all parses of each training sentence

$\Rightarrow$ Use the small ATIS treebank corpus
  - Trained on 1088 sentences of ATIS1 corpus
  - Tested on 294 sentences of ATIS2 corpus

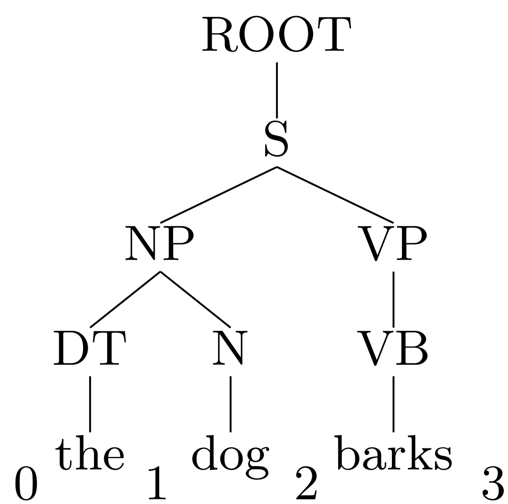
- MCLE estimator initialized with MLE probabilities
- (Added in 2003: I think there may be better ways to do the conditional estimation)

Parser evaluation

- A node's *edge* is its label and beginning and ending *string positions*
- $E(y)$ is the set of edges associated with a tree y (same with forests)
- If y is a parse tree and $\bar{y}$ is the correct tree, then

precision	$P_{\bar{y}}(y) = \lvert E(y)\rvert / \lvert E(y) \cap E(\bar{y})\rvert$
recall	$R_{\bar{y}}(y) = \lvert E(\bar{y})\rvert / \lvert E(y) \cap E(\bar{y})\rvert$
f score	$F_{\bar{y}}(y) = 2/(P_{\bar{y}}(y)^{-1} + R_{\bar{y}}(y)^{-1})$

	Edges	
(0	NP	2)
(2	VP	3)
(0	S	3)



Conditional and Joint ML PCFG evaluation

	MLE	MCLE
– log likelihood of training data	13857	13896
– log *conditional* likelihood of training data	1833	1769
– log *marginal* probability of training strings	12025	12127
Labelled precision of test data	0.815	0.817
Labelled recall of test data	0.789	0.794

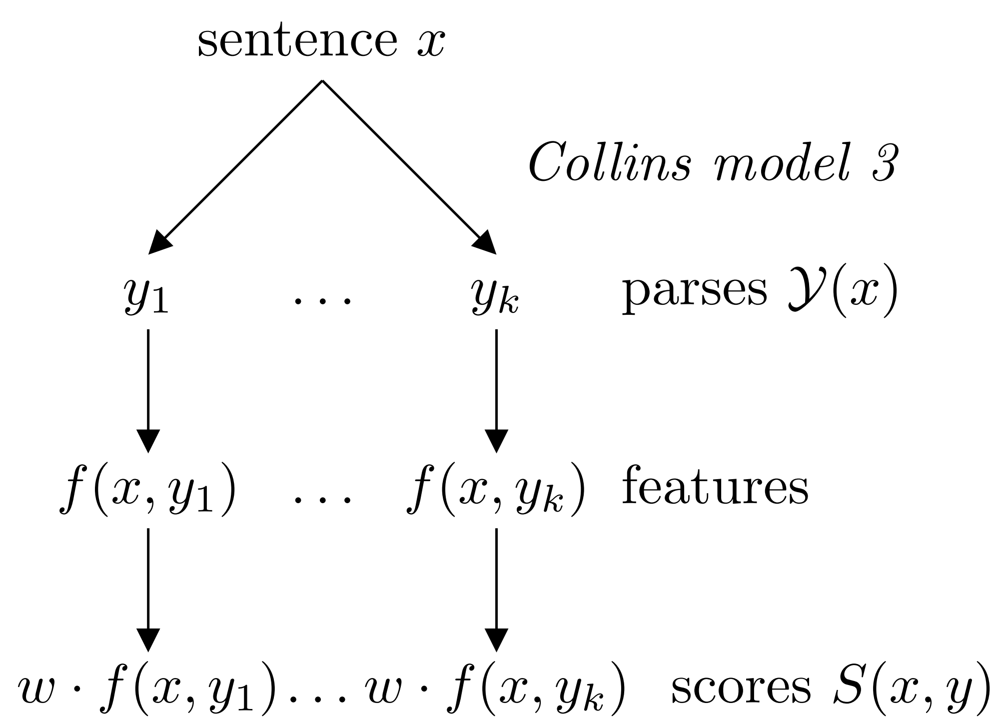
- Precision/recall difference *not significant* ($p \approx 0.1$)

Experiments in Discriminative Parsing

- Collins Model 3 parser produces a *set of candidate parses* $\mathcal{Y}(x)$ for each sentence x
- The discriminative parser has a weight w_j for each feature f_j
- The score for each parse is $S(x, y) = w \cdot f(x, y)$
- The highest scoring parse

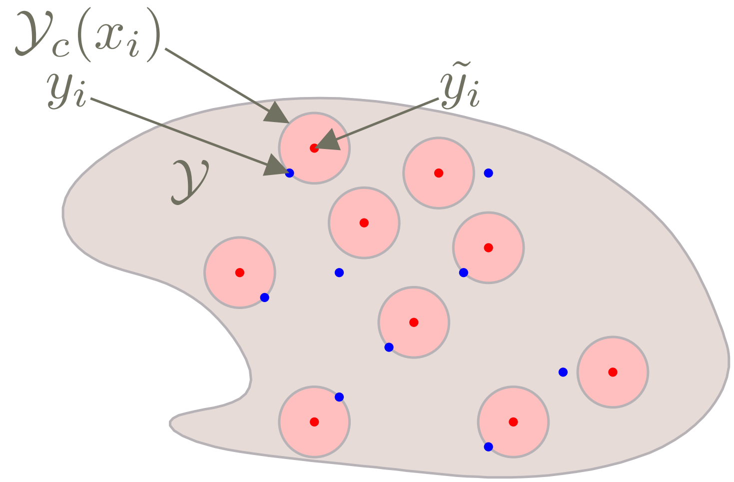
$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} S(x, y)$$

is predicted correct



Training the discriminative parser

- Training data $((x_1, y_1), \ldots, (x_n, y_n))$
- Each string x_i is parsed using Collins parser, producing a set $\mathcal{Y}_c(x_i)$ of parse trees
- Best parse $\tilde{y}_i = \mathrm{argmax}_{y \in \mathcal{Y}_c(x_i)} F_{y_i}(y)$, where $F_{y'}(y)$ measures parse accuracy
- w is chosen to maximize the regularized log pseudo-likelihood $\sum_i \log \mathrm{P}_w(\tilde{y}_i | x_i) + R(w)$



Baseline and oracle results

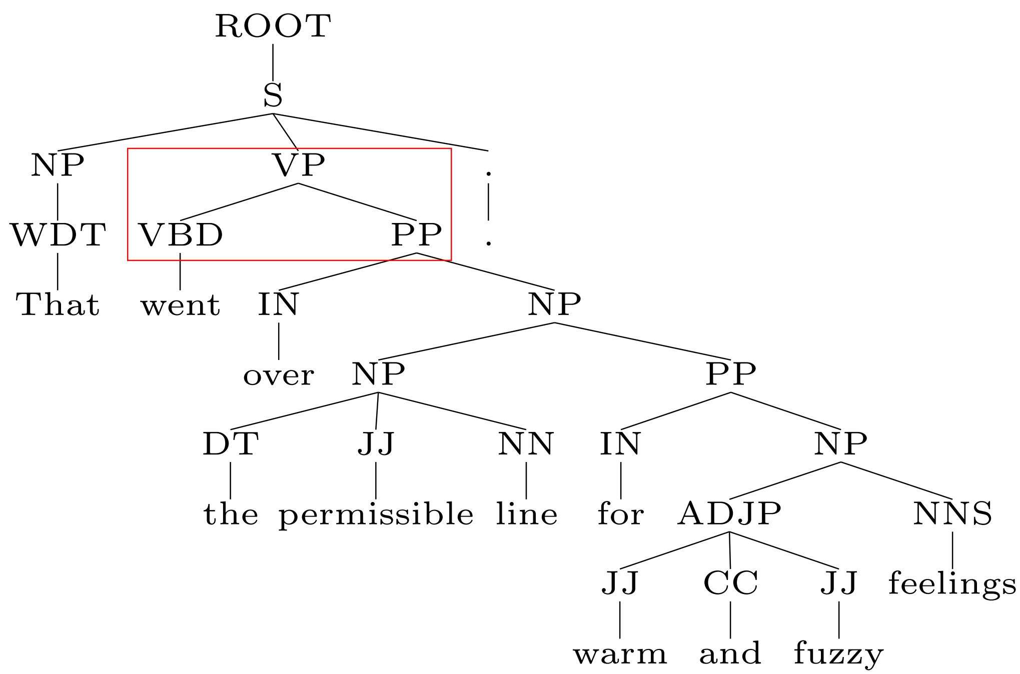
- Training corpus: 36,112 Penn treebank trees, development corpus 3,720 trees from sections 2-21
- Collins Model 2 parser failed to produce a parse on 115 sentences
- Average $|\mathcal{Y}(x)| = 36.1$
- Model 2 f-score $= 0.882$ (picking parse with highest Model 2 probability)
- Oracle (maximum possible) f-score $= 0.953$
  (i.e., evaluate f-score of closest parses $\tilde{y}_i$)

$\Rightarrow$ Oracle (maximum possible) error reduction 0.601

Expt 1: Only “old” features

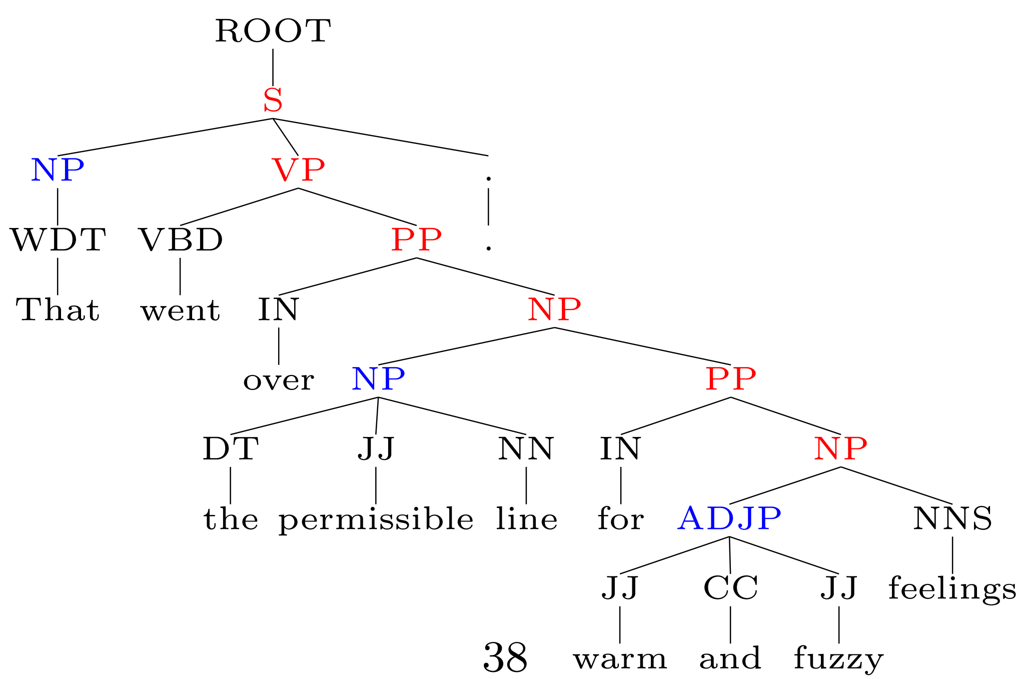
- Features: (1) *log Model 2 probability*, (9717) local tree features
- Model 2 already conditions on local trees!
- Feature selection: features must vary on 5 or more sentences
- Results: f-score $= 0.886$; $\approx 4\%$ error reduction

$\Rightarrow$ *discriminative training alone can improve accuracy*



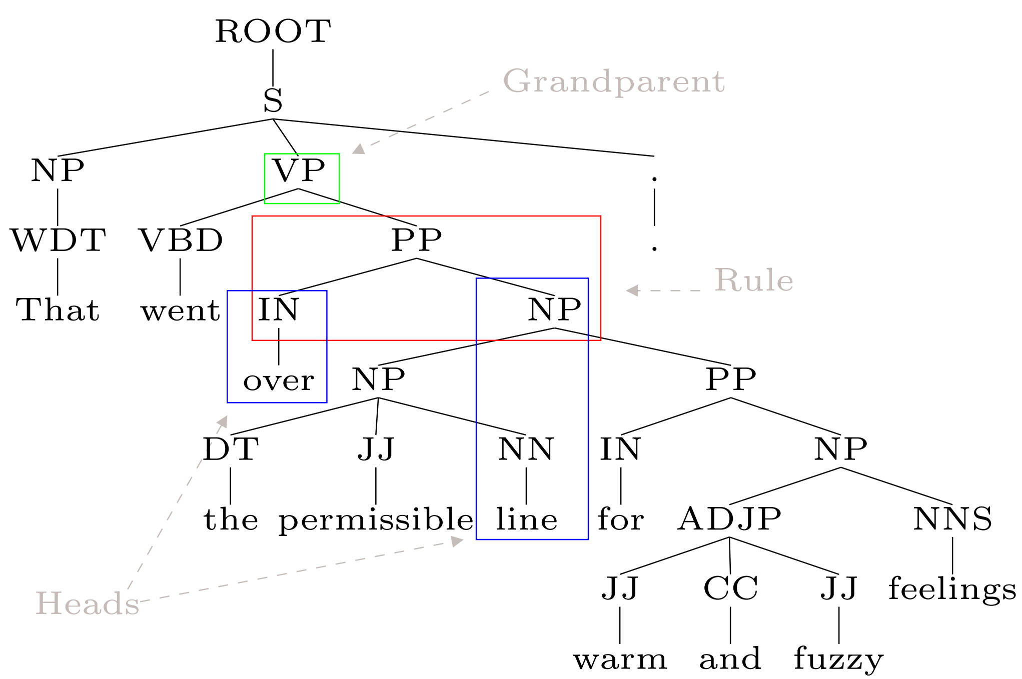
Expt 2: Rightmost branch bias

- The RightBranch feature's value is the number of nodes on the right-most branch (ignoring punctuation)
- Reflects the tendancy toward right branching
- LogProb + RightBranch: f-score = 0.884 (probably significant)
- LogProb + RightBranch + Rule: f-score = 0.889



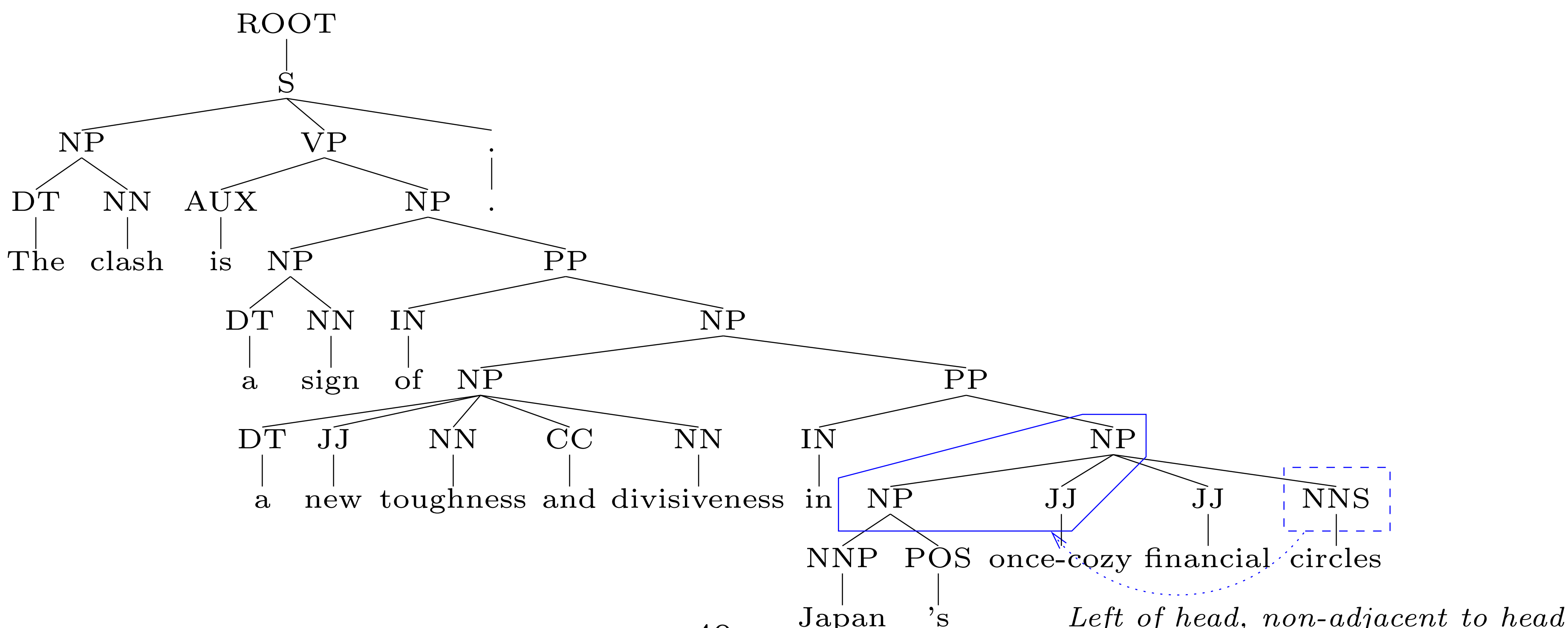
Lexicalized and parent-annotated rules

- *Lexicalization* associates each constituent with its head
- *Parent annotation* provides a little "vertical context"
- With all combinations, there are 158,890 rule features



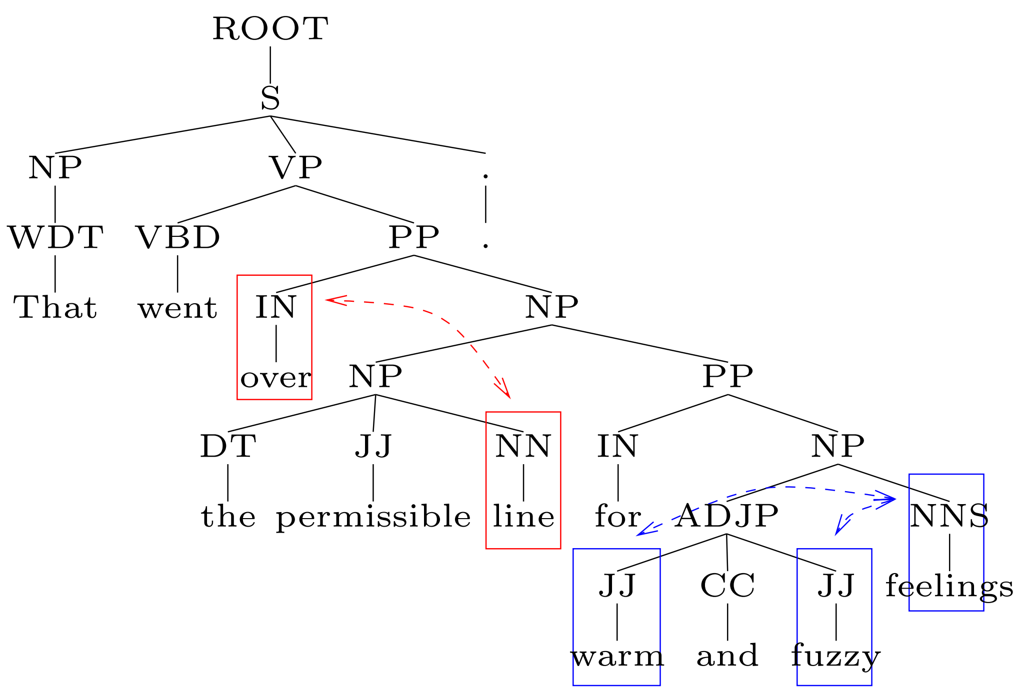
n-gram rule features generalize rules

- Collects adjacent constituents in a local tree
- Also includes relationship to head
- Constituents can be ancestor-annotated and lexicalized
- 5,143 unlexicalized rule bigram features, 43,480 lexicalized rule bigram features



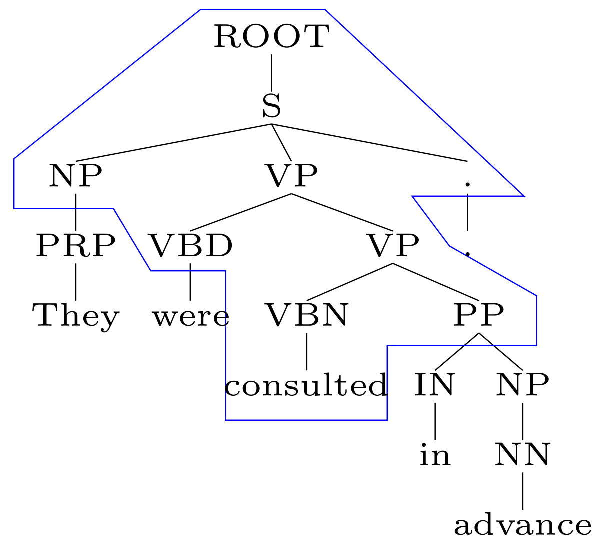
Head to head dependencies

- Head-to-head dependencies track the function-argument dependencies in a tree
- Co-ordination leads to phrases with multiple heads or functors
- With all combinations, there are 121,885 head-to-head features



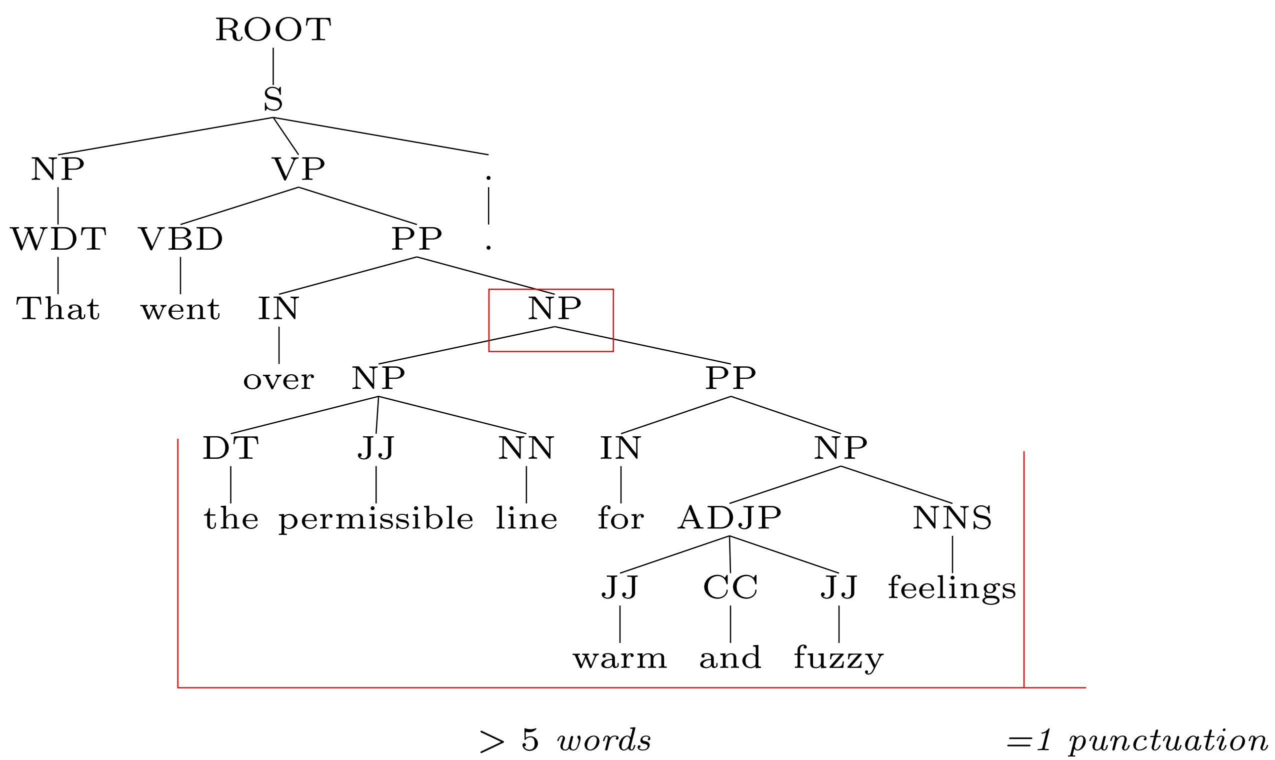
Head trees record all dependencies

- Head trees consist of a (lexical) head, all of its projections and (optionally) all of the siblings of these nodes
- These correspond roughly to TAG elementary trees



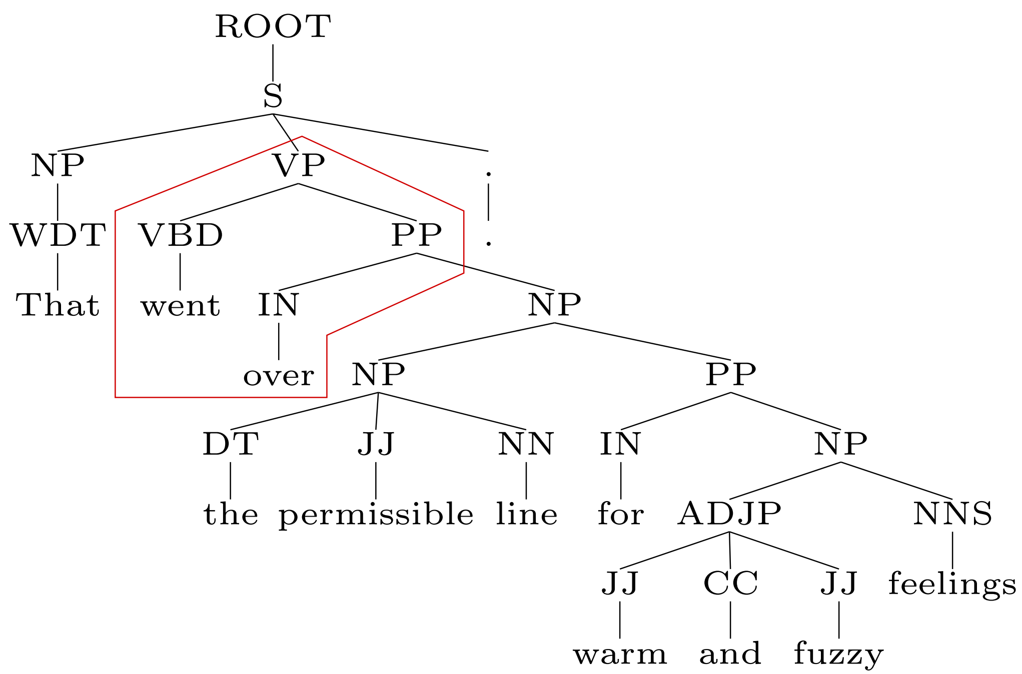
Constituent Heavyness and location

- Heavyness measures the constituent's category, its (binned) size and (binned) closeness to the end of the sentence
- There are 984 Heavyness features



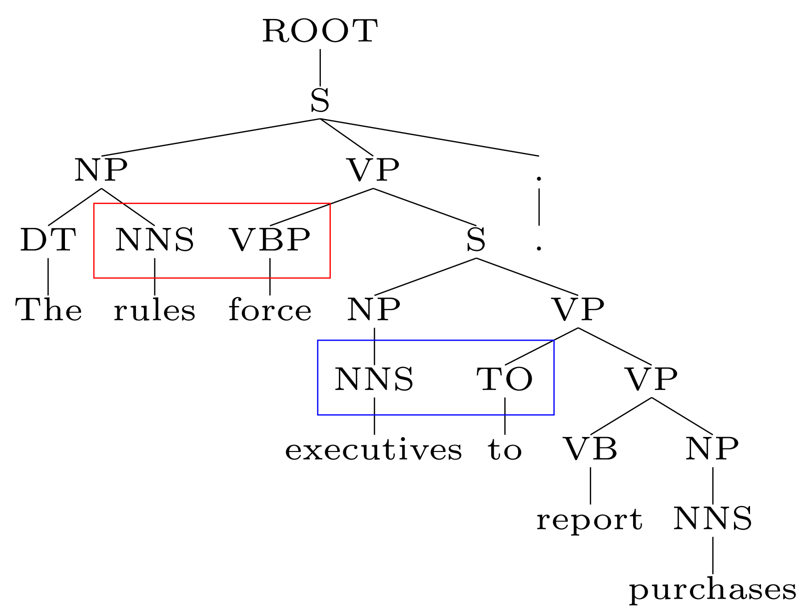
Tree n-gram

- A tree n-gram are tree fragments that connect sequences of adjacent n words
- There are 62,487 tree n-gram features



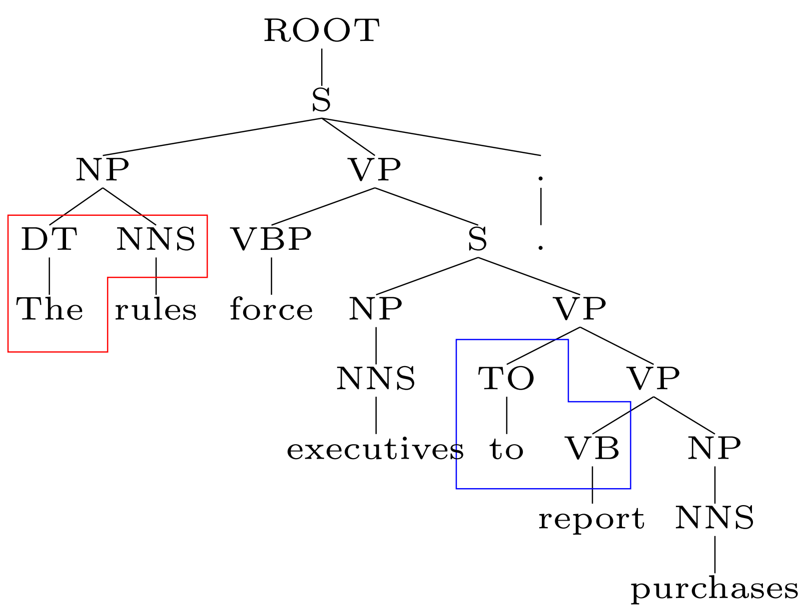
Subject-Verb Agreement

- The SubjVerbAgr features are the POS of the subject NP's lexical head and the VP's functional head
- There are 200 SubjVerbAgr features



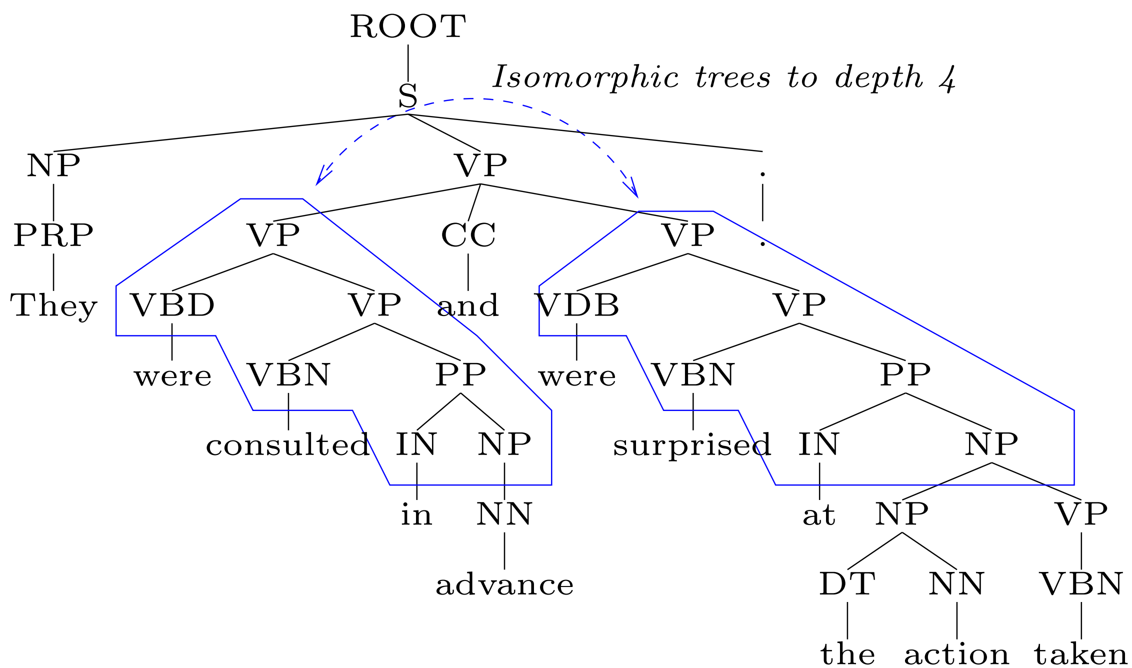
Functional-lexical head dependencies

- The SynSemHeads features collect pairs of functional and lexical heads of phrases (Grimshaw)
- This captures number agreement in NPs and aspects of other head-to-head dependencies
- There are 1,606 SynSemHeads features



Coordination parallelism (1)

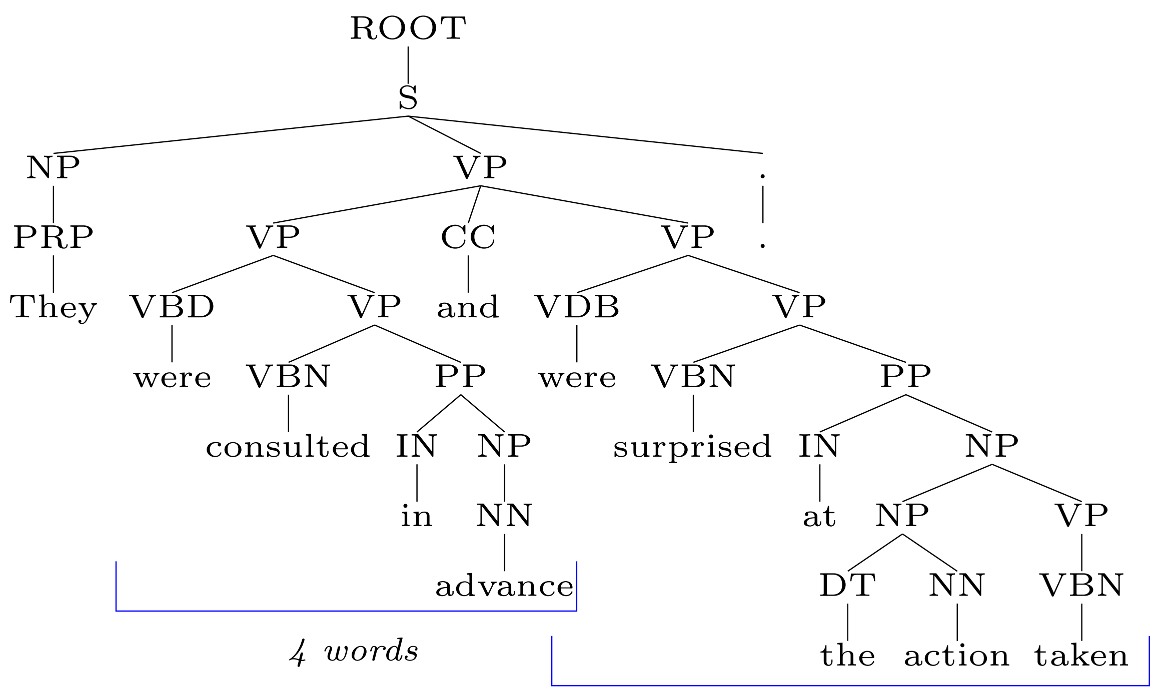
- The CoPar feature indicates the depth to which adjacent conjuncts are parallel
- There are 9 CoPar features



Coordination parallelism (2)

- The CoLenPar feature indicates the difference in length in adjacent conjuncts and whether this pair contains the last conjunct.

- There are 22 CoLenPar features



CoLenPar feature: (2,true)

Regularization

- General form of regularizer: $c \sum_j |w_j|^p$
- $p = 1$ leads to sparse weight vectors. (Kazama and Tsujii, 2003)
  - If $|\partial L / \partial w_j| < c$ then $w_j = 0$
- Experiment on small feature set:
  - 164,273 features
  - $c = 2, p = 2$, f-score $= 0.898$
  - $c = 4, p = 1$, f-score $= 0.896$, only 5,441 non-zero features!
  - Earlier experiments suggested that optimal performance is obtained with $p \approx 1.5$

Experimental results with all features

- Features must vary on parses of at least 5 sentences in training data
- In this experiment, 692,708 features
- regularization term: $4 \sum_j |w_j|^2$
- dev set results: *f-score* $= 0.904$ (20% error reduction)

Which kinds of features are best?

	# of features	f-score
Treebank trees	375,646	0.901
Correct parses	271,267	0.902
Incorrect parses	876,339	0.903
Correct & incorrect parses	883,936	0.903

- Features from incorrect parses characterize failure modes of Collins parser
- There are far more ways to be wrong than to be right!

Evaluating feature classes

Δ **f-score**	$\Delta - logL$	**# w**	**av w[j]**	**sd w[j]**	**zeroed class**
-0.0187508	1814.32	1	0.629557	inf	LogProb
-0.00185951	145.987	2	-0.477453	1.59274e-05	RightBranch
5.50245e-05	9.44562	9717	0.000637244	0.0024974	Rule:0:0:0:0:0:0:0:0
-0.00106989	0.896624	48723	0.000629753	0.00235112	Rule:1:0:0:0:0:0:0:0
-0.000611704	2.77256	68035	0.000639053	0.00255555	NGramTree:3:2:1:0
-0.000270621	1.66255	21543	0.000944576	0.0028058	Heads:2:0:1:1
-0.00031439	5.4608	10187	0.000908379	0.00225115	Word:2
-0.00241466	61.5452	984	-0.00115477	0.0119984	Heavy
-0.00153331	47.0448	7450	0.000453298	0.00513622	Neighbours:1:1
0.000127092	11.0722	9	0.145198	0.0562	CoPar
-0.00018458	5.28722	22	0.0155067	0.0313398	CoLenPar
-9.55417e-05	1.30432	200	-0.00147174	0.00723214	SubjVerbAgr

Summary

- Generative and discriminative parsers both identify the likely parse y of a string x, i.e., estimate $\mathrm{P}(y|x)$
- *Generative parsers also define language models*, estimate $\mathrm{P}(x)$
- *Discriminative estimation doesn't require feature independence*
  - suitable for grammar formalisms without CF branching structure
- *Parsing is equally complex* for generative and discriminative parsers
  - *depends on features used*
  - reranking uses one parser to narrow the search space for another
- *Estimation* is *computationally inexpensive for generative parsers*, but *expensive for discriminative parsers*
- Because a discrimative parser can use the generative model's probability estimate as a feature, *discriminative parsers almost never do worse* than the generative model, and often do substantially better.

Discriminative learning in other settings

- Speech recognition
  - Take x to be the acoustic signal, $\mathcal{Y}(x)$ all strings in recognizer lattice for x
  - Training data: $D = ((y_1, x_1), \ldots, (y_n, x_n))$, where y_i is correct transcript for x_i
  - Features could be n-grams, log parser prob, cache features
- Machine translation
  - Take x to be input language string, $\mathcal{Y}(x)$ a set of target language strings (e.g., generated by an IBM-style model)
  - Training data: $D = ((y_1, x_1), \ldots, (y_n, x_n))$, where y_i is correct translation of x_i
  - Features could be n-grams of target language strings, word and phrase correspondences, ...

Conclusion and directions for future work

- Discriminatively trained parsing models can perform better than standard generative parsing models
- *Features can be arbitrary functions of parse trees*
  - Difficult to tell which features are most useful
  - Are there techniques to systematically evaluate and explore possible features?
- Generative parser language models can be applied to a variety of applications. Are there similiar generic discriminative parsers?
- Efficient computational procedures for search and estimation
  - *Dynamic programming*
  - *Approximation methods* (variational methods, best-first or beam search)

Regularizer tuning in Max Ent models

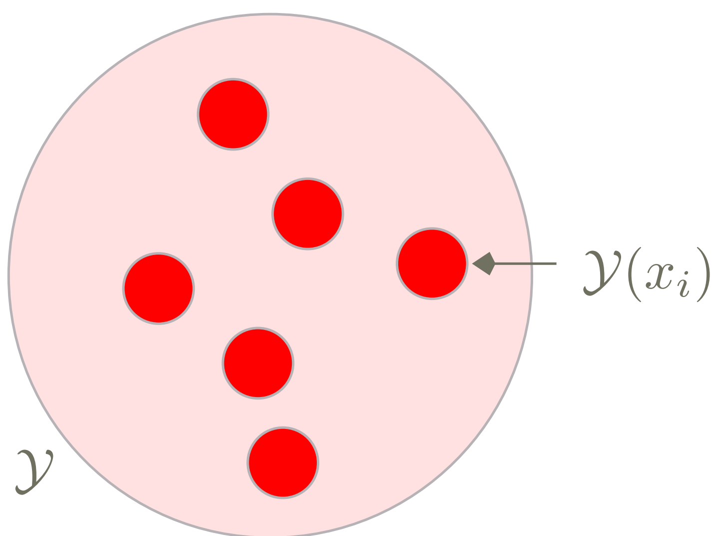
- Associate each feature f_j with bin $b(j)$
- Associate regularizer constant β_k with feature bin k
- Optimize feature weights $\alpha = (\alpha_1, \ldots, \alpha_m)$ on main training data M
- Optimize regularizer constants β on held-out data H

$$
\begin{aligned}
L_D(\alpha) &= \prod_{i=1}^{n} \mathrm{P}_\alpha(y_i|x_i), \ \text{where } D = ((y_1, x_1), \ldots, (y_n, x_n)) \\
\hat{\alpha}(\beta) &= \operatorname*{argmax}_\alpha \log L_M(\alpha) - \sum_{j=1}^{m} \beta_{b(j)} \alpha_j^2 \\
\hat{\beta} &= \operatorname*{argmax}_\beta \log L_H(\hat{\alpha}(\beta))
\end{aligned}
$$

Expectation maximization for PCFGs

- Hidden training data: $D = (x_1, \ldots, x_n)$, where x_i is a string
- The Inside-Outside algorithm is an Expectation-Maximization algorithm for PCFGs

$$
\begin{aligned}
\hat{p} &= \operatorname*{argmax}_p L_D(p), \text{ where} \\
L_D(p) &= \prod_{i=1}^{n} \mathrm{P}_p(x_i) = \operatorname*{argmax}_p \prod_{i=1}^{n} \sum_{y \in \mathcal{Y}(x_i)} \mathrm{P}(y)
\end{aligned}
$$



Why there is no conditional ML EM

- Conditional ML conditions on the string x
- Hidden training data: $D = (x_1, \ldots, x_n)$, where x_i is a string
- The likelihood is the probability of predicting the string x_i given the string x_i, a *constant function*

$$\begin{aligned} \hat{p} &= \operatorname*{argmax}_p L_D(p), \text{ where} \\ L_D(p) &= \prod_{i=1}^{n} \mathrm{P}_p(x_i | x_i) \end{aligned}$$