Features of Statistical Parsers

Mark Johnson
Brown Laboratory for Linguistic Information Processing

CoNLL 2005

Features of Statistical Parsers

Confessions of a bottom-feeder:
Dredging in the Statistical Muck

Mark Johnson
Brown Laboratory for Linguistic Information Processing

CoNLL 2005

Features of Statistical Parsers

Confessions of a bottom-feeder:
Dredging in the Statistical Muck

Mark Johnson
Brown Laboratory for Linguistic Information Processing

CoNLL 2005

With much help from *Eugene Charniak*, *Michael Collins* and *Matt Lease*

Outline

- Goal: find features for identifying good parses
- Why is this difficult with generative statistical models?
- Reranking framework
- Conditional versus joint estimation
- Features for parse ranking
- Estimation procedures
- Experimental set-up
- Feature selection and evaluation

Features for accurate parsing

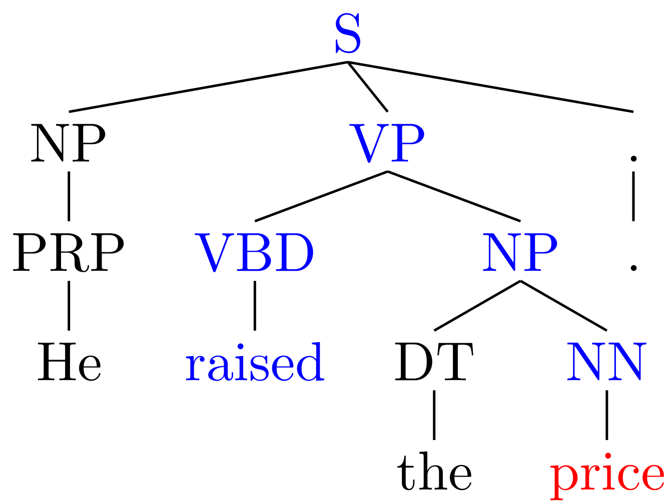
- *Accurate parsing requires good features*

⇒ need a *flexible* method for evaluating a *wide range* of features

- *parse ranking framework* is current best method for doing this
  - \+ works with virtually any kind of representation
  - \+ features can encode virtually any kind of information (syntactic, lexical semantics, prosody, etc.)
  - \+ can exploit the currently best-available parsers
  - − efficient algorithms are hard(-er) to design and implement
  - − fishing expedition

Why not a generative statistical parser?

- Statistical parsers (Charniak, Collins) generate parses node by node
- Each step is conditioned on the structure already generated



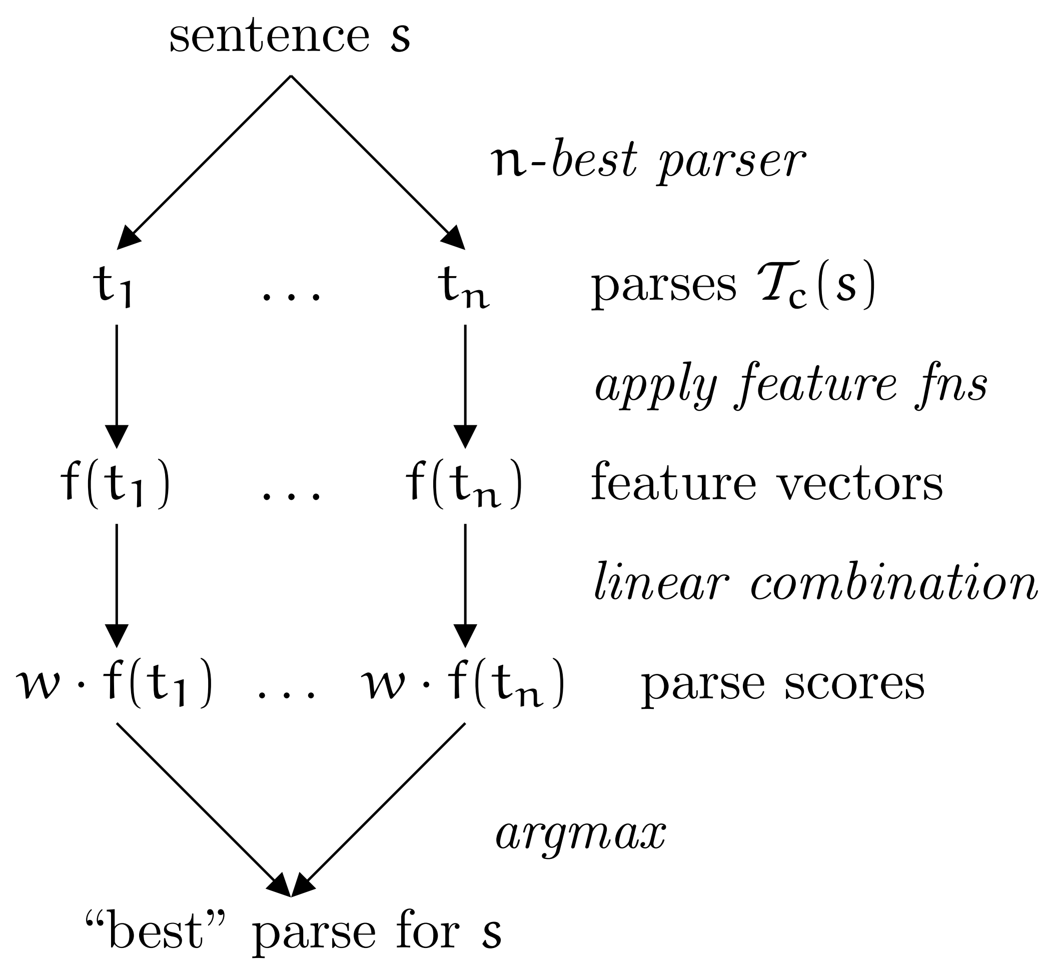
- Encoding dependencies is as difficult as designing a feature-passing grammar (GPSG)
- *Smoothing interacts in mysterious ways with these encodings*
- Conditional estimation should produce better parsers *with our current lousy models*

Linear ranking framework

- Generate n *candidate parses* $\mathcal{T}_c(s)$ for each sentence s
- Map each parse $t \in \mathcal{T}_c(s)$ to a real-valued *feature vector* $f(t) = (f_1(t), \ldots, f_m(t))$
- Each feature f_j is associated with a *weight* w_j
- The *highest scoring* parse

$$\hat{t} = \underset{t \in \mathcal{T}_c(s)}{\operatorname{argmax}} \, w \cdot f(t)$$

is predicted correct



Linear ranking example

$$w = (-1, 2, 1)$$

Candidate parse tree t	features $f(t)$	parse score $w \cdot f(t)$
t_1	$(1, 3, 2)$	7
t_2	$(2, 2, 1)$	3
$\ldots$	$\ldots$	$\ldots$

- Parser designer specifies *feature functions* $f = (f_1, \ldots, f_m)$
- *Feature weights* $w = (w_1, \ldots, w_m)$ specify each feature's "importance"
- n-best parser produces trees $\mathcal{T}_c(s)$ for each sentence s
- Feature functions f apply to each tree $t \in \mathcal{T}_c(s)$, producing *feature values* $f(t) = (f_1(t), \ldots, f_m(t))$
- Return *highest scoring tree*

$$\hat{t}(s) = \underset{t}{\operatorname{argmax}}\, w \cdot f(t) = \underset{t}{\operatorname{argmax}} \sum_{j=1}^{m} w_j f_j(t)$$

Linear ranking, statistics and machine learning

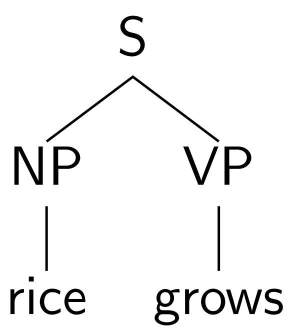
- Many models define the best candidate $\hat{t}$ in terms of *a linear combination of feature values* $w \cdot f(t)$
  - Exponential, Log-linear, Gibbs models, MaxEnt

$$\begin{aligned} P(t) &= \frac{1}{Z} \exp w \cdot f(t) \\ Z &= \sum_{t \in \mathcal{T}} \exp w \cdot f(t) \qquad \text{(partition function)} \\ \log P(t) &= w \cdot f(t) - \log Z \end{aligned}$$

  - Perceptron algorithm (including averaged version)
  - Support Vector Machines
  - Boosted decision stubs

PCFGs are exponential models

$$\begin{aligned} f_j(t) &= \text{number of times the } j\text{th rule is used in } t \\ w_j &= \log p_j, \text{ where } p_j \text{ is probability of } j\text{th rule} \end{aligned}$$



$$f\left(\text{tree} \right) = [\underbrace{1}_{\text{S}\to\text{NP VP}}, \underbrace{1}_{\text{NP}\to\text{rice}}, \underbrace{0}_{\text{NP}\to\text{bananas}}, \underbrace{1}_{\text{VP}\to\text{grows}}, \underbrace{0}_{\text{VP}\to\text{grow}}]$$

$$\begin{aligned} P_{\mathrm{PCFG}}(t) &= \prod_j p_j^{f_j(t)} = \prod_j \exp(w_j)^{f_j(t)} = \prod_j \exp w_j f_j(t) \\ &= \exp \sum_j w_j f_j(t) = \exp w \cdot f(t) \end{aligned}$$

So *a PCFG is just a special kind of exponential model* with $Z = 1$.

Features in linear ranking models

- Features can be *any real-valued function* of parse t and sentence s
  - *counts* of number of times a particular structure appears in t
  - *log probabilities* from other models
    - $\log P_c(t)$ is our most useful feature!
    - *generalizes reference distributions* of MaxEnt models
- Subtracting a constant $c(s)$ from a feature's value doesn't affect *difference between parse scores* in a linear model

$$w \cdot (f(t_1) - c(s)) - w \cdot (f(t_2) - c(s)) \quad = \quad w \cdot f(t_1) - w \cdot f(t_2)$$

  - features that don't vary on $\mathcal{T}_c(s)$ are useless
  - subtract most frequently occuring value $c_j(s)$ for each feature f_j in sentence $s \Rightarrow$ sparser feature vectors

Getting the feature weights

s	$f(t^\star(s))$	$\{f(t) : t \in \mathcal{T}_c(s), t \neq t^\star(s)\}$
sentence 1	$(1, 3, 2)$	$(2, 2, 3)$ $(3, 1, 5)$ $(2, 6, 3)$
sentence 2	$(7, 2, 1)$	$(2, 5, 5)$
sentence 3	$(2, 4, 2)$	$(1, 1, 7)$ $(7, 2, 1)$
...	...	...

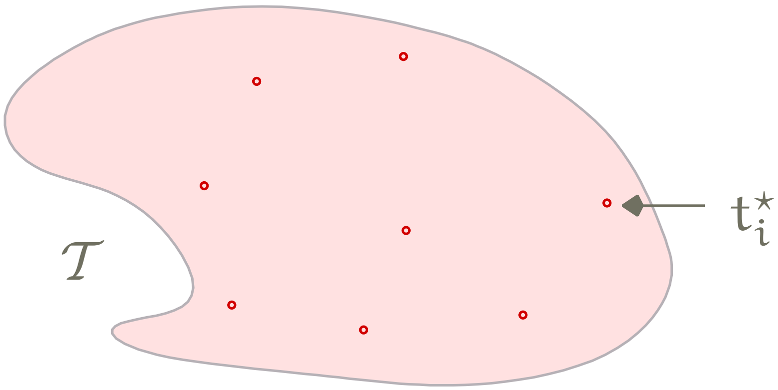
- n-best parser produces trees $\mathcal{T}_c(s)$ for each sentence s
- Treebank gives *correct tree* $t^\star(s) \in \mathcal{T}_c(s)$ for sentence s
- Feature functions f apply to each tree $t \in \mathcal{T}_c(s)$, producing *feature values* $f(t) = (f_1(t), \ldots, f_m(t))$
- Machine learning algorithm selects *feature weights* w to prefer $t^\star(s)$ (e.g., so $w \cdot f(t^\star(s))$ is greater than $w \cdot f(t')$ for other $t' \in \mathcal{T}_c(s)$)

Conditional ML estimation of w

- Conditional ML estimation selects w to make $t^\star(s)$ as likely as possible *compared to the trees in* $\mathcal{T}_c(s)$
- Same as conditional MaxEnt estimation

$$
\begin{aligned}
P_w(t|s) &= \frac{1}{Z_w(s)} \exp w \cdot f(t) && \textit{exponential model} \\
Z_w(s) &= \sum_{t' \in \mathcal{T}_c(s)} \exp w \cdot f(t') \\
D &= ((s_1, t_1^\star), \ldots, (s_n, t_n^\star)) && \textit{treebank training data} \\
L_D(w) &= \prod_{i=1}^{n} P_w(t_i^\star | s_i) && \textit{conditional likelihood of } D \\
\widehat{w} &= \operatorname*{argmax}_w L_D(w)
\end{aligned}
$$

(Joint) MLE for exponential models is hard



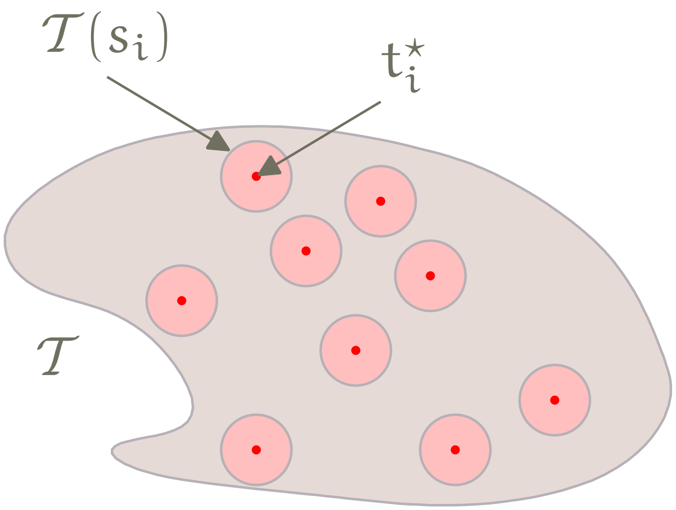
$$
\begin{aligned}
D &= (t_1^\star, \ldots, t_n^\star) \\
L_D(w) &= \prod_{i=1}^{n} P_w(t_i^\star) \\
\widehat{w} &= \operatorname*{argmax}_{w} L_D(w)
\end{aligned}
$$

$$P_w(t) = \frac{1}{Z_w} \exp w \cdot f(t), \quad Z_w = \sum_{t' \in \mathcal{T}} \exp w \cdot f(t')$$

- Joint MLE selects w to make $t_i^\star$ as likely as possible
- $\mathcal{T}$ is set of *all possible parses for all possible strings*
- $\mathcal{T}$ is infinite $\Rightarrow$ cannot be enumerated $\Rightarrow$ Z_w cannot be calculated
- For a PCFG, Z_w and hence $\widehat{w}$ are easy to calculate, but ...
- in general $\partial L_D / \partial w_j$ and Z_w are *intractable analytically and numerically*
- Abney (1997) suggests a Monte-Carlo calculation method

Conditional MLE is easier

- The *conditional likelihood* of w is the *conditional probability* of the *hidden part* of the data (syntactic structure) $t^\star$ given its *visible part* (yield or terminal string) s
- The conditional likelihood can be numerically optimized because $\mathcal{T}_c(s)$ can be enumerated (by a parser)



$$
\begin{aligned}
D &= ((t_1^\star, s_1) \ldots, (t_n^\star, s_n)) \\
L_D(w) &= \prod_{i=1}^{n} P_w(t_i^\star | s_i) \\
\widehat{w} &= \operatorname*{argmax}_{w} L_D(w)
\end{aligned}
$$

$$
P(t|s) = \frac{1}{Z_w(s)} \exp w \cdot f(t), \qquad Z_w(s) = \sum_{t' \in \mathcal{T}_c(s)} \exp w \cdot f(t')
$$

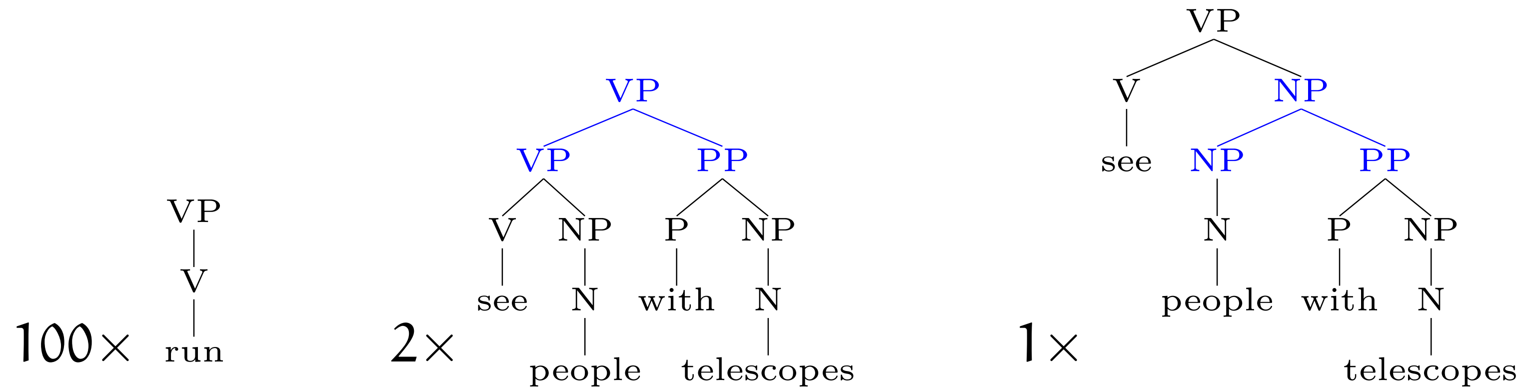
Conditional vs joint estimation

- Joint MLE maximizes probability of training trees *and strings*
  - Generative statistical parsers usually use joint MLE
  - Joint MLE is simple to compute (relative frequency)
- Conditional MLE maximizes probability of trees given strings
  - Conditional estimation uses less information from the data
  - learns nothing from distribution of strings
  - ignores unambiguous sentences (!)

$$\mathrm{P}(\mathrm{t}, s) \; = \; \mathrm{P}(\mathrm{t}|s)\mathrm{P}(s)$$

- *Joint MLE should be better (lower variance) if your model correctly predicts the distribution of parses and strings*
  - *Any good probabilistic models of semantics and discourse?*

Conditional vs joint MLE for PCFGs



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

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

Rule	**count**	**rel freq**	**better vals**
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**

Regularization

- Overlearning $\Rightarrow$ add regularization R that penalizes "complex" models
- Useful with a wide range of objective functions

$$
\begin{aligned}
\hat{w} &= \underset{w}{\operatorname{argmin}}\, Q(w) + R(w) \\
Q(w) &= -\log L_D(w) \qquad \text{(objective function)} \\
R(w) &= c \sum_j |w_j|^p \qquad \text{(regularizer)} \\
L_D(w) &= \prod_i P_w(t_i^\star | s_i)
\end{aligned}
$$

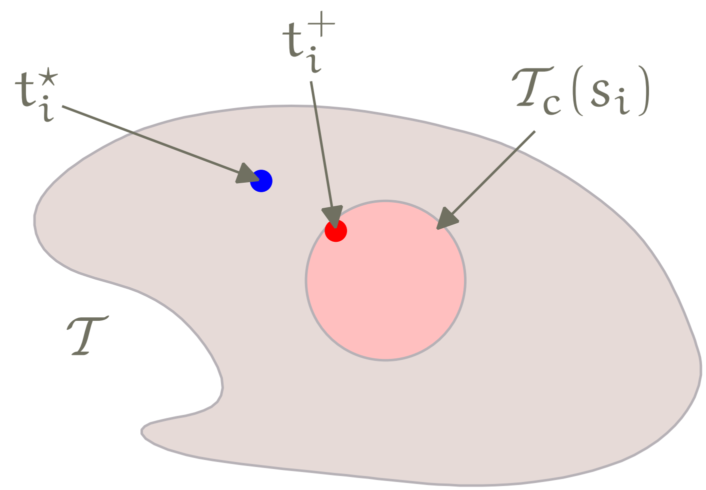
- $p = 2$ known as the Gaussian prior
- $p = 1$ known as the Laplacian or exponential prior
  - sparse solutions
  - requires special care in optimization (Kazama and Tsujii, 2003)

If candidate parses don't include correct parse

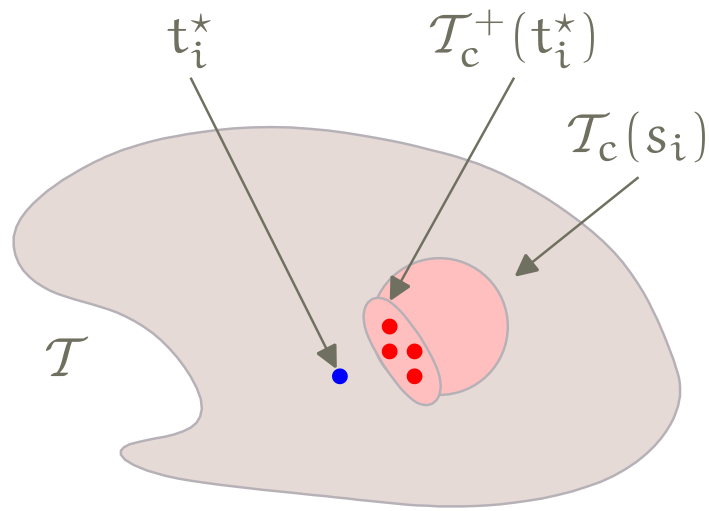
- If $\mathcal{T}_c(s)$ doesn't include $t^\star(s)$, choose parse $t^+(s)$ in $\mathcal{T}_c(s)$ closest to $t^\star(s)$
- Maximize conditional likelihood of $(t_1^+, \ldots, t_n^+)$

- Closest parse $t_i^+ = \mathrm{argmax}_{t \in \mathcal{T}(s_i)} F_{t_i^\star}(t)$
  - $F_{t^\star}(t)$ is *f-score* of t relative to $t^\star$
- w chosen to maximize the regularized log conditional likelihood of t_i^+

$$L_D(w) = \prod_i P_w(t_i^+ | s_i)$$



Multiple closest parses

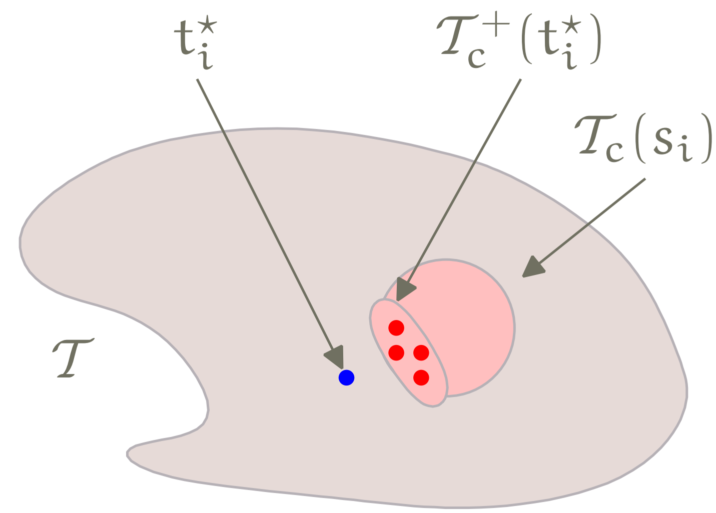


- There can be more than one candidate parses $\mathcal{T}_c^+(t_i^\star)$ equally close to the correct parse $t_i^\star$: which one(s) should we declare to be the best parse?
- Picking a parse at random does not work as well as ...
- picking the parse with the highest Charniak parse probability, but ...
- *maximizing probability of all close parses* (EM-like scheme in Riezler '02) works best of all

$$L_D(w) \;=\; \prod_i P(\mathcal{T}_c(t_i^\star)|\mathcal{T}_c(s_i))$$

Likelihood of multiple best parses

- Treebank $D = ((t_1^\star, s_1), \ldots, (t_n^\star, s_n))$
- n-best candidates $\mathcal{T}_c(s_i)$ of sentence s_i
- $\mathcal{T}_c^+(t_i^\star) =$ trees in $\mathcal{T}_c(s_i)$ with max f-score
- w chosen to maximize the regularized log conditional likelihood of $\mathcal{T}_c^+(t_i^\star)$



$$
\begin{aligned}
L_D(w) &= \prod_i P_w(T_c^+(t_i^\star) | \mathcal{T}_c(s_i)) \\
&= \prod_i \frac{\sum_{t \in \mathcal{T}_c^+(t^\star)} \exp w \cdot f(t)}{\sum_{t \in \mathcal{T}_c(s_i)} \exp w \cdot f(t)}
\end{aligned}
$$

- $\partial \log L / \partial w_j$ is a *difference in expectations* over $\mathcal{T}_c^+(t^\star)$ and $\mathcal{T}_c(s_i)$

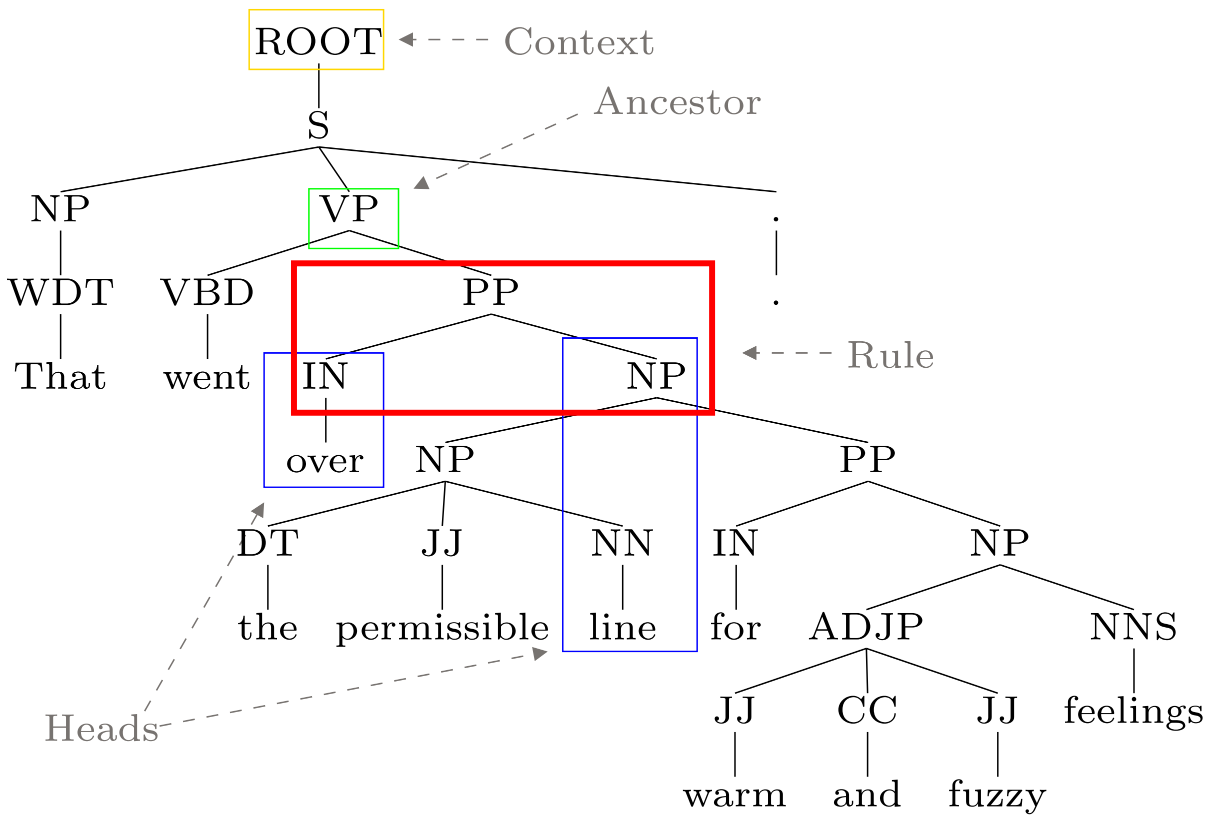
Features for ranking parses

- Features can be any real-valued function of parse trees
- In these experiments the features come in two kinds:
  - The logarithm of the tree's probability estimated by the Charniak parser
  - The number of times a particular configuration appears in the parse

Which ones improve parsing accuracy the most?

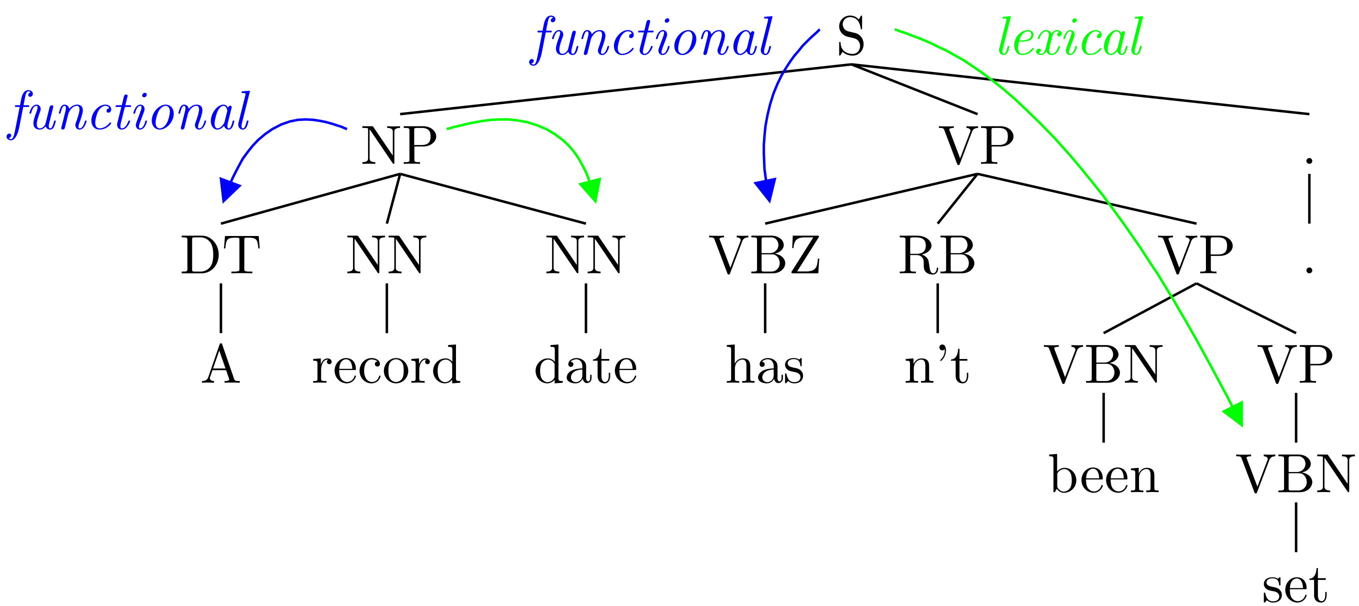
Lexicalized and parent-annotated rules

- *Lexicalization* associates each constituent with its head
- *Ancestor annotation* provides a little "vertical context"
- *Context annotation* indicates constructions that only occur in main clause (c.f., Emonds)



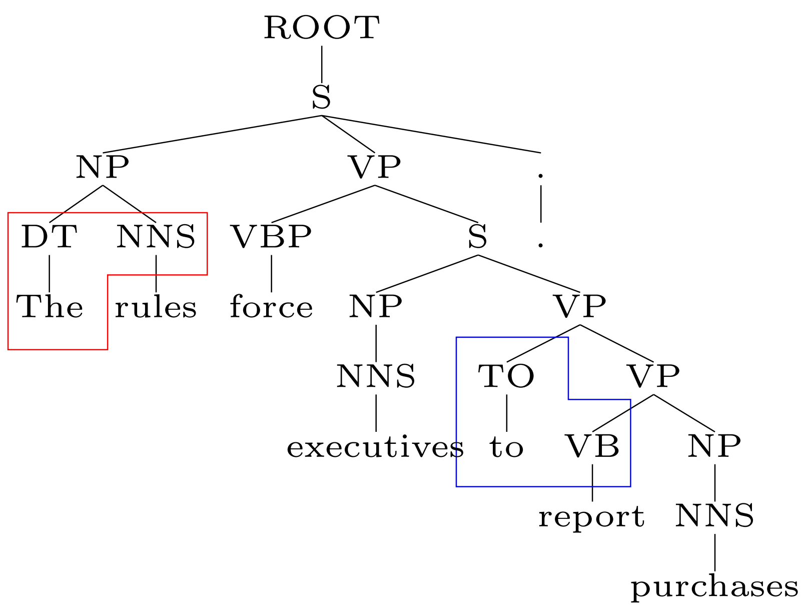
Functional and lexical heads

- There are at least two sensible notions of head (c.f., Grimshaw)
  - *Functional heads:* determiners of NPs, auxilary verbs of VPs, etc.
  - *Lexical heads:* rightmost Ns of NPs, main verbs in VPs, etc.
- In a Maxent model, it is easy to use both!



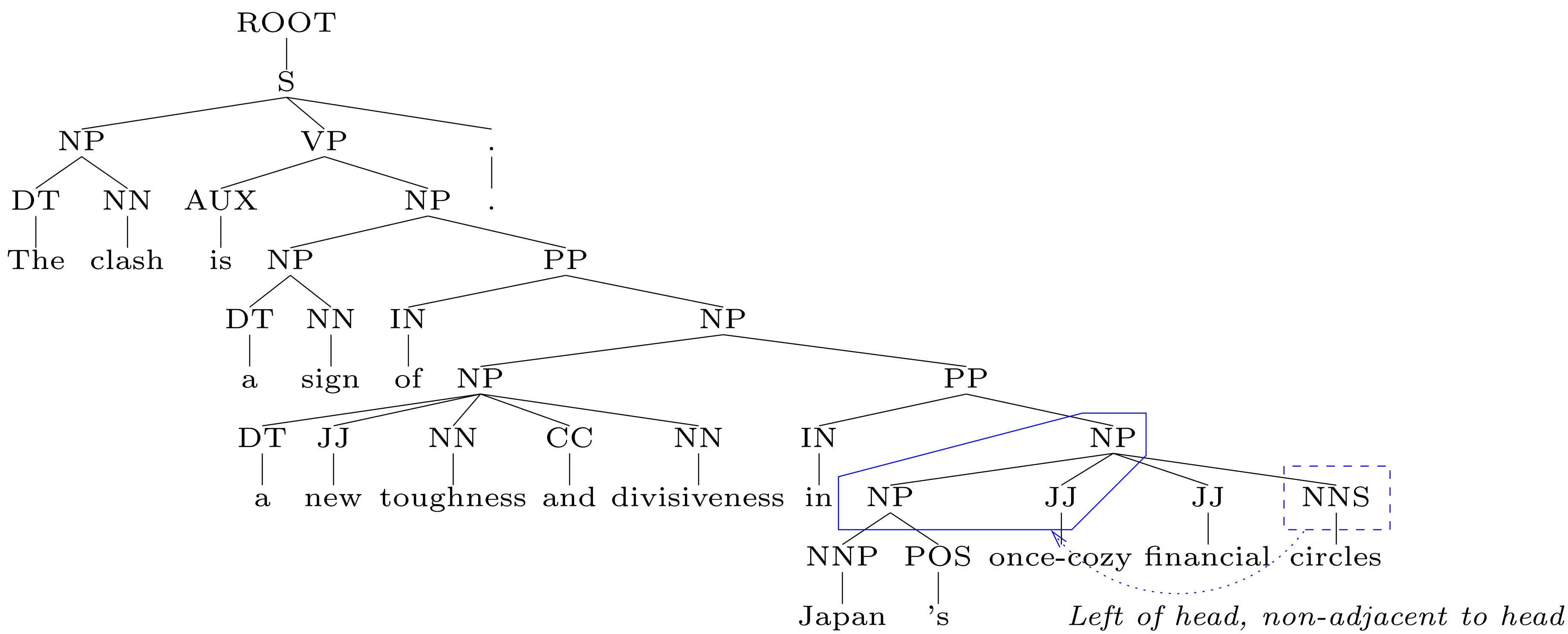
Functional-lexical head dependencies

- The SynSemHeads features collect pairs of functional and lexical heads of phrases
- This captures *number agreement in NPs* and aspects of other head-to-head dependencies
- Parameterized by *lexicalization*



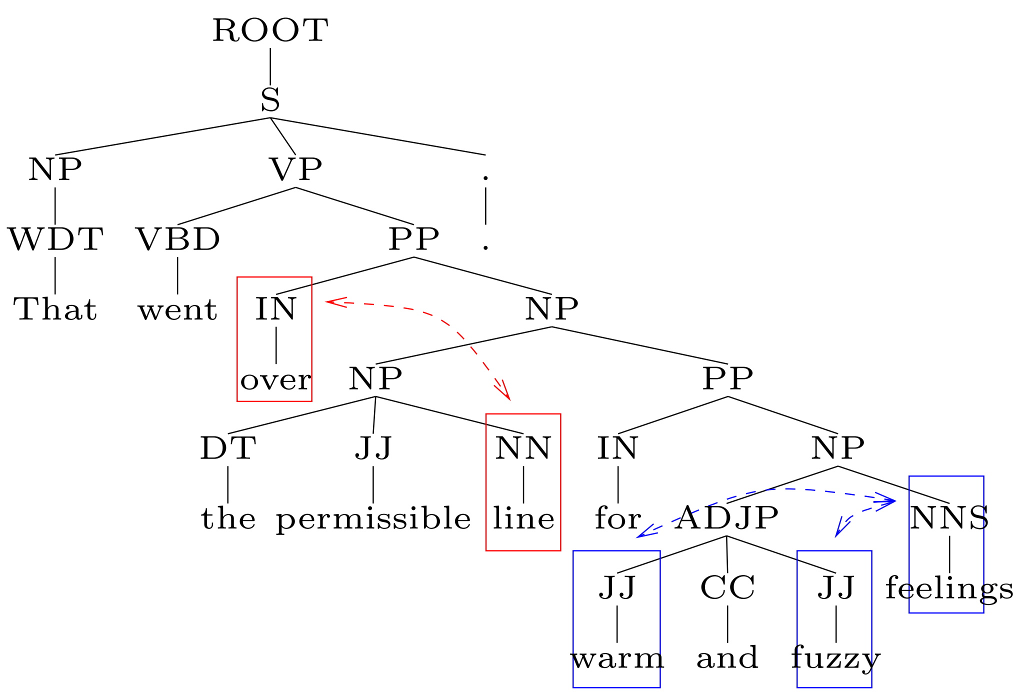
n-gram rule features generalize rules

- Collects *adjacent constituents* in a local tree
- Also includes *relationship to head* (e.g., adjacent? left or right?)
- Parameterized by *ancestor-annotation*, *lexicalization* and *head-type*



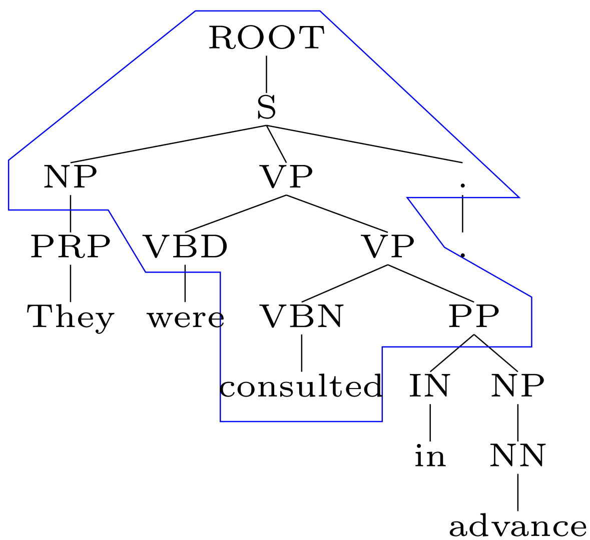
Head to head dependencies

- Head-to-head dependencies track the function-argument dependencies in a tree
- Co-ordination leads to phrases with multiple heads or functors
- Parameterized by *head type*, *number of governors* and *lexicalization*



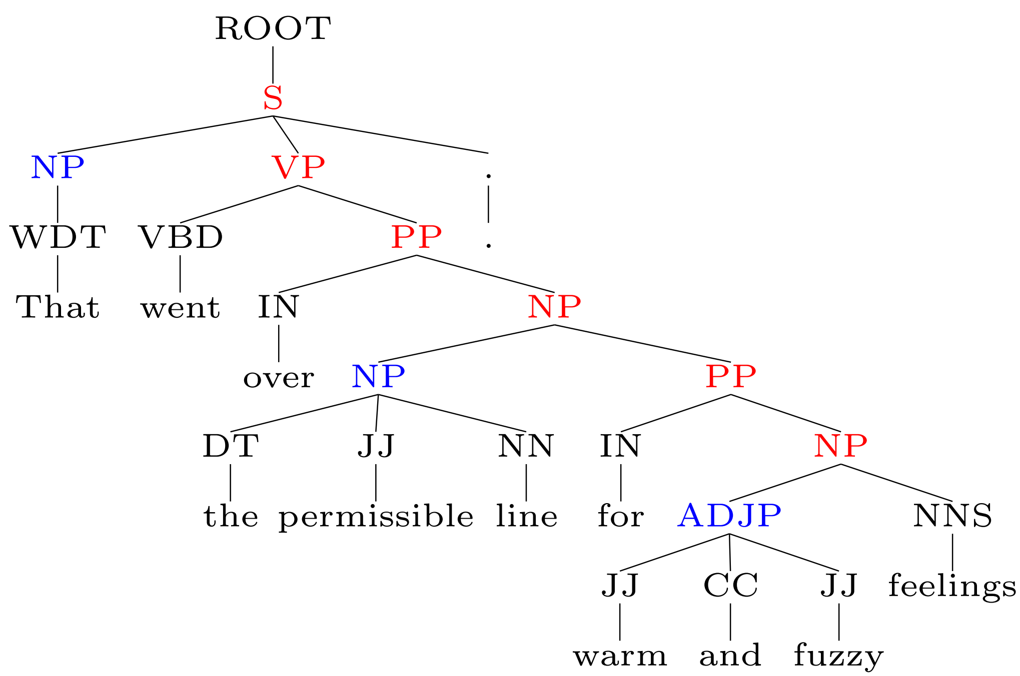
Head trees record all dependencies

- Head trees consist of a (lexical) head, all of its projections and (optionally) all of the siblings of these nodes
- correspond roughly to TAG elementary trees
- parameterized by *head type*, *number of sister nodes* and *lexicalization*



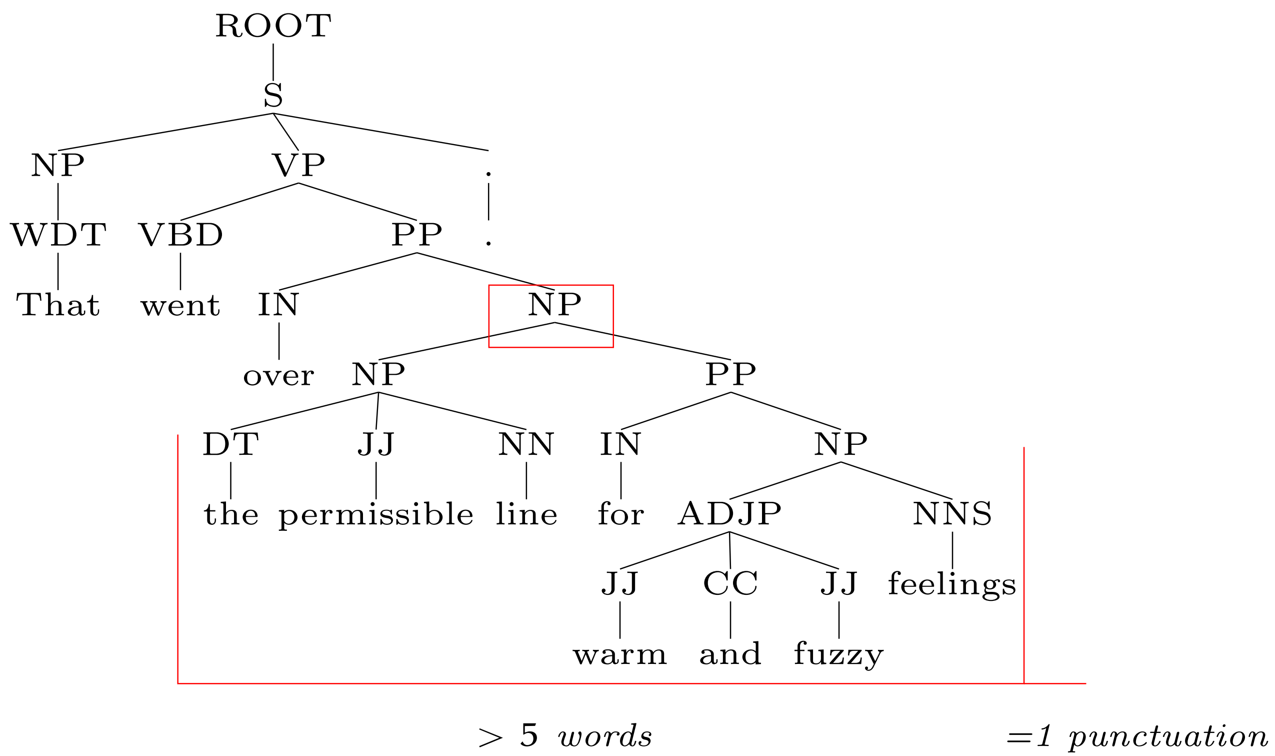
Rightmost branch bias

- The RightBranch feature's value is the number of nodes on the right-most branch (ignoring punctuation) (c.f., Charniak 00)
- Reflects the tendancy toward right branching in English
- Only 2 different features, but very useful in final model!



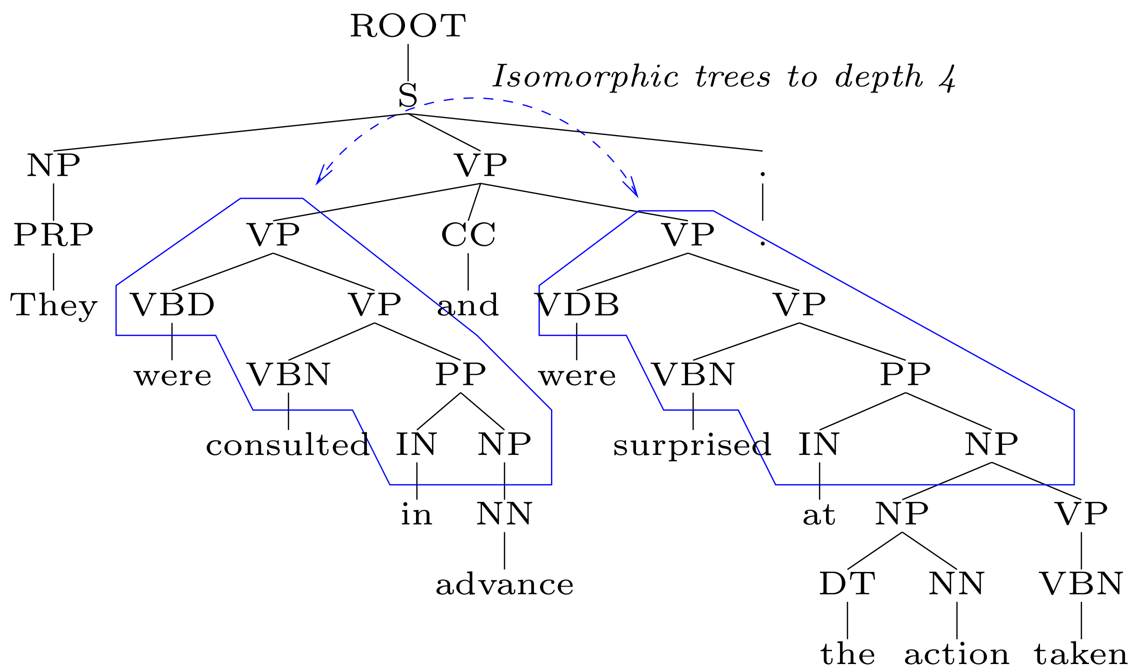
Constituent Heavyness and location

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



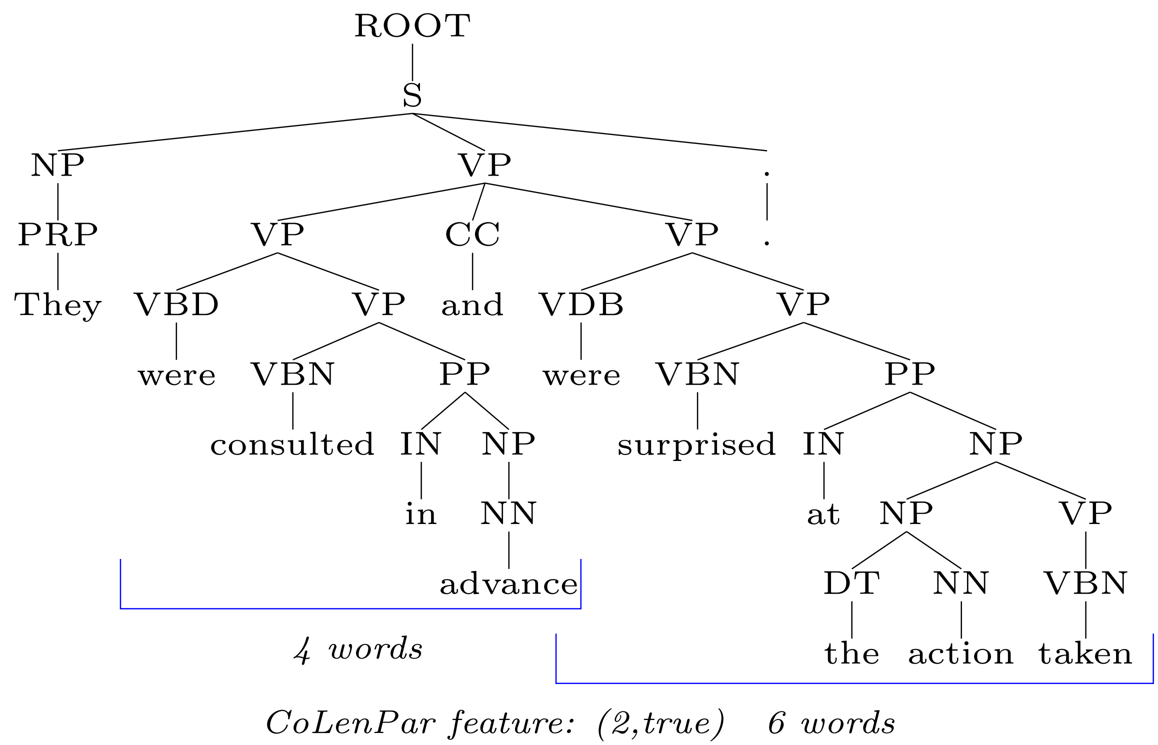
Coordination parallelism (1)

- A CoPar feature indicates the depth to which adjacent conjuncts are parallel



Coordination parallelism (2)

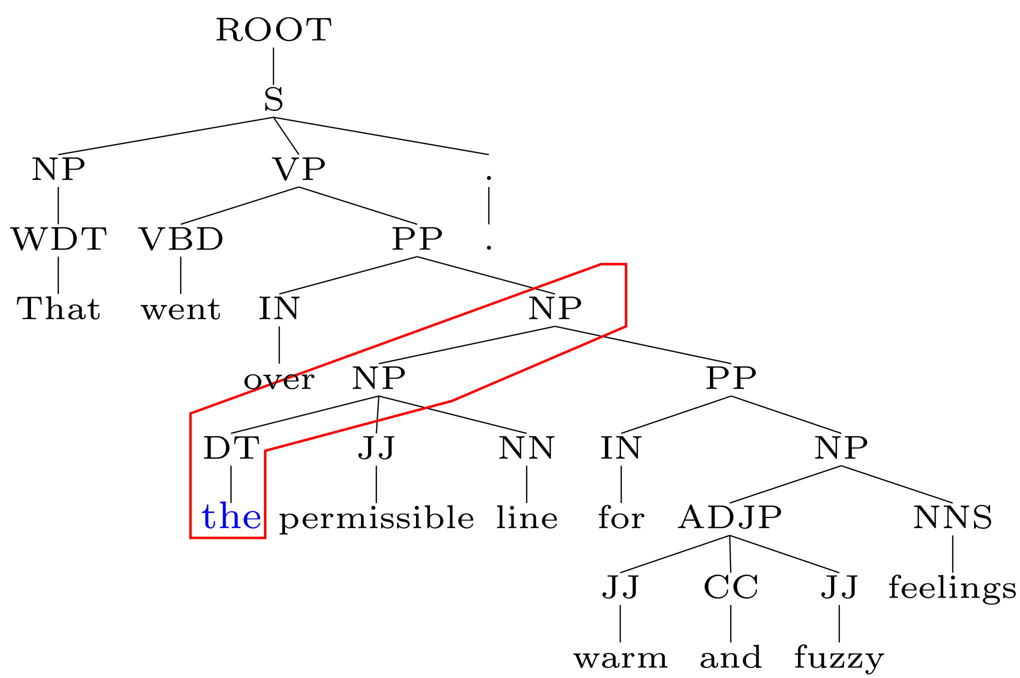
- The CoLenPar feature indicates the difference in length in adjacent conjuncts and whether this pair contains the last conjunct.



Word

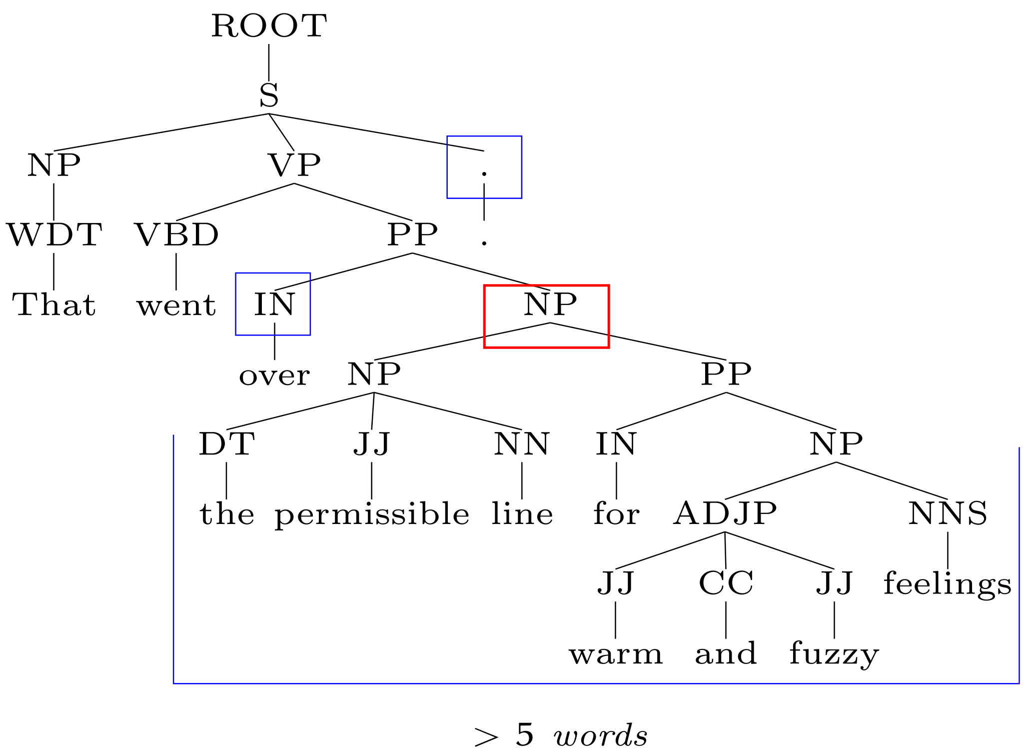
- A Word feature is a word plus n of its parents (c.f., Klein and Manning's non-lexicalized PCFG)

- A WProj feature is a word plus all of its (maximal projection) parents, up to its governor's maximal projection



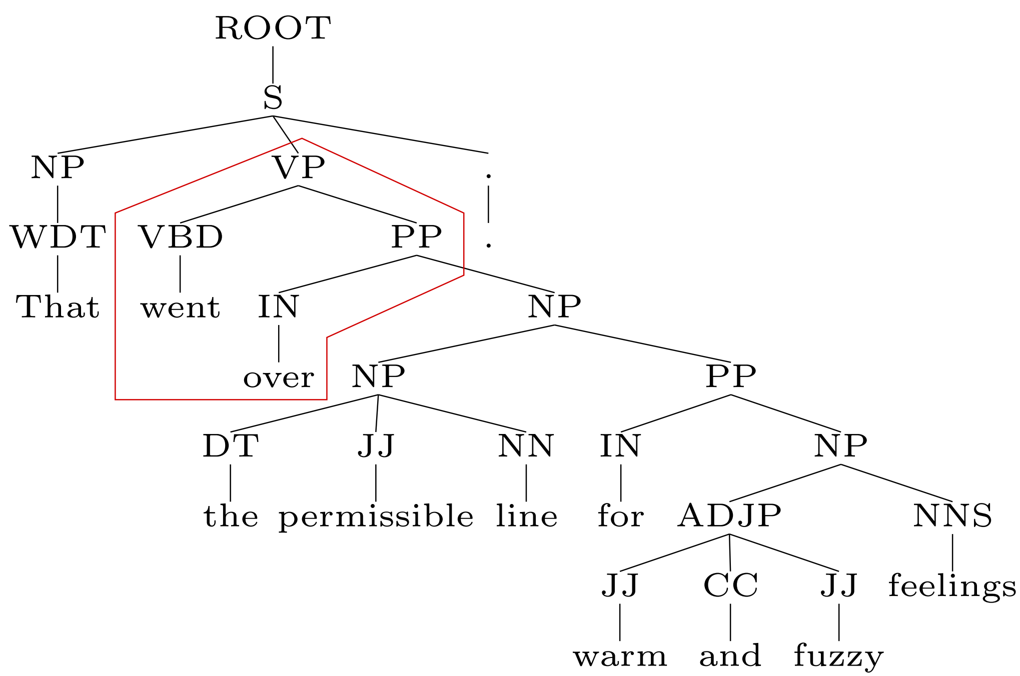
Neighbours

- A Neighbours feature indicates the node's category, its binned length and j left and k right POS tags for $\mathsf{j}, \mathsf{k} \leq 1$



Tree n-gram

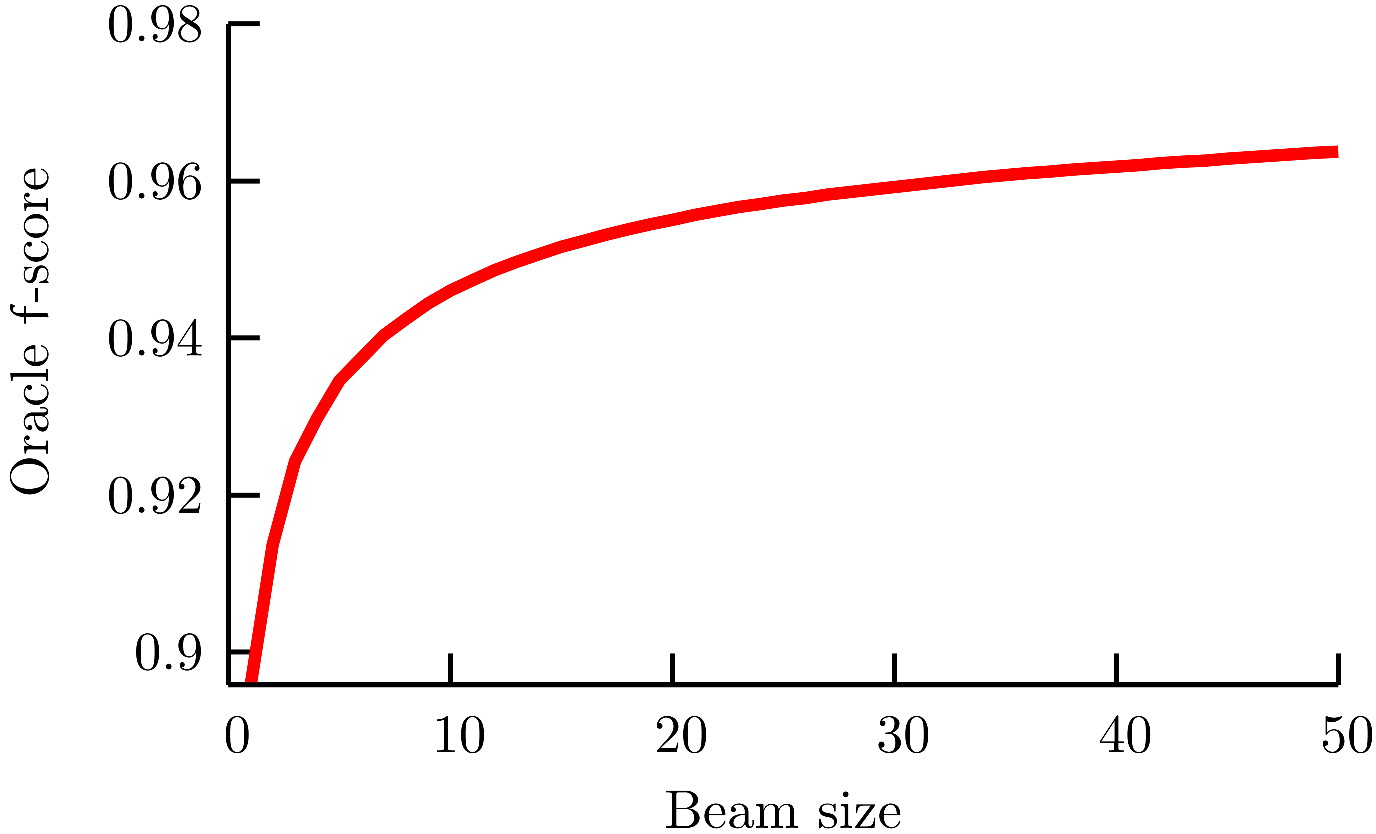
- A tree n-gram feature is a tree fragment that connect sequences of adjacent n words, for $n = 2, 3, 4$ (c.f. Bod's DOP models)
- lexicalized and non-lexicalized variants



Experimental setup

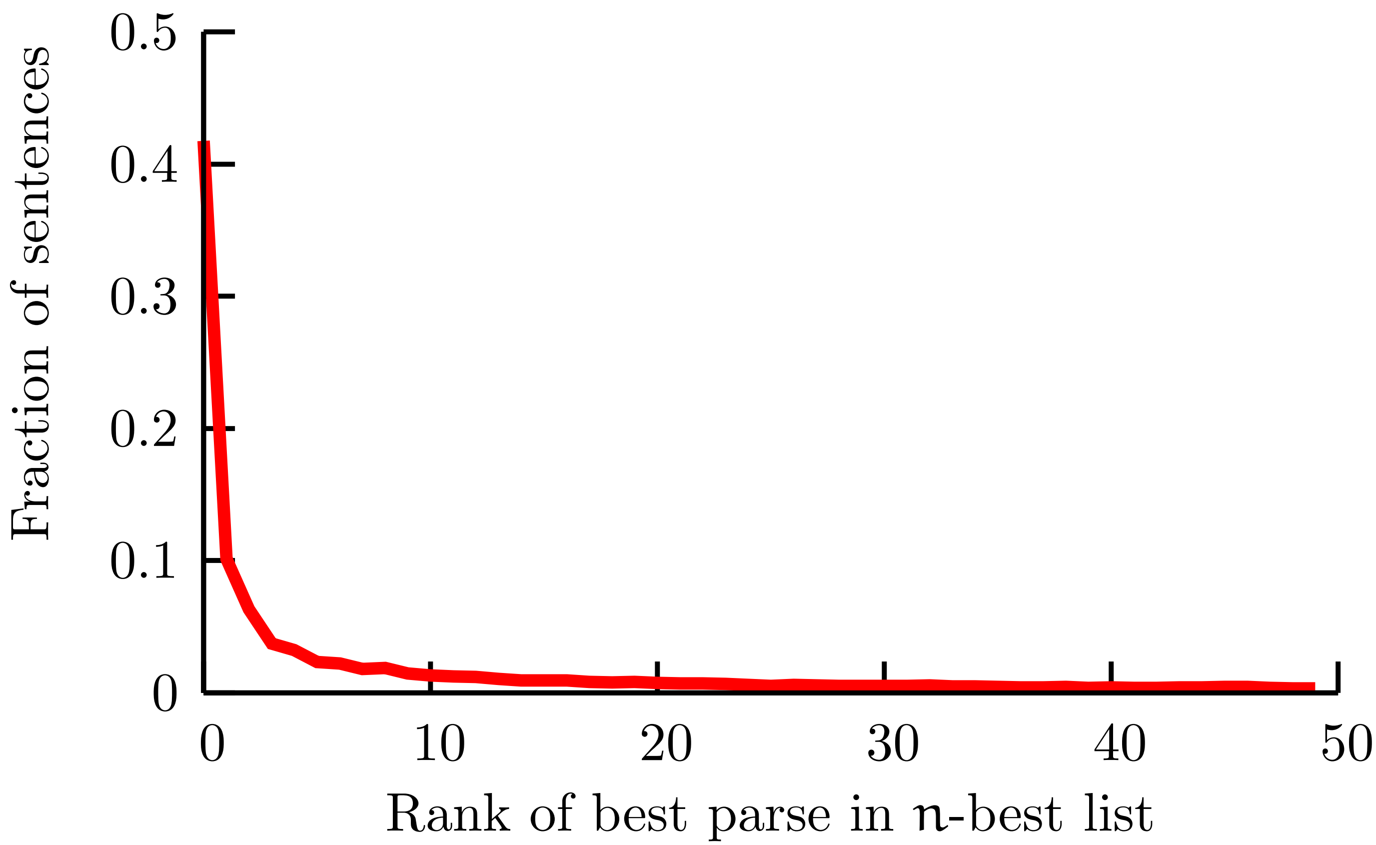
- Feature tuning experiments done using Collins' split: sections 2-19 as train, 20-21 as dev and 24 as test
- $\mathcal{T}_c(s)$ computed using Charniak 50-best parser
- Features which vary on less than 5 sentences pruned
- Optimization performed using LMVM optimizer from Petsc/TAO optimization package
- Regularizer constant c adjusted to maximize f-score on dev

f-score vs. n-best beam size



- F-score of Charniak's most probable parse = 0.896
- Oracle f-score of Charniak's 50-best parses = 0.965 (66% redn)
- oracle f-score continues to rise at wide beam widths
- no guarantee that reranker performance improves with beam width!

Rank of best parse

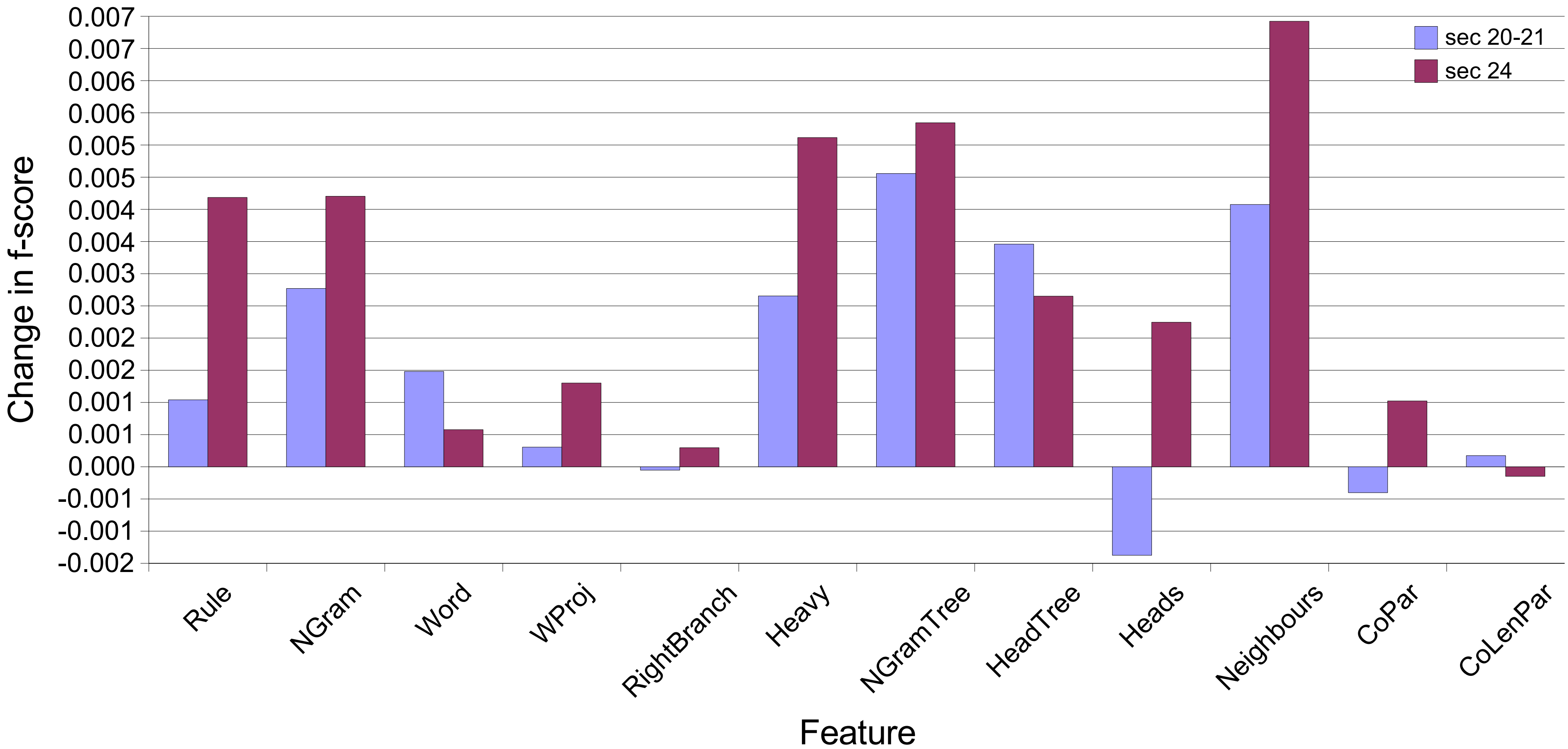


- Charniak parser's most likely parse is the best parse 41% of the time
- Reranker picks Charniak parser's most likely parse 58% of the time

Evaluating features

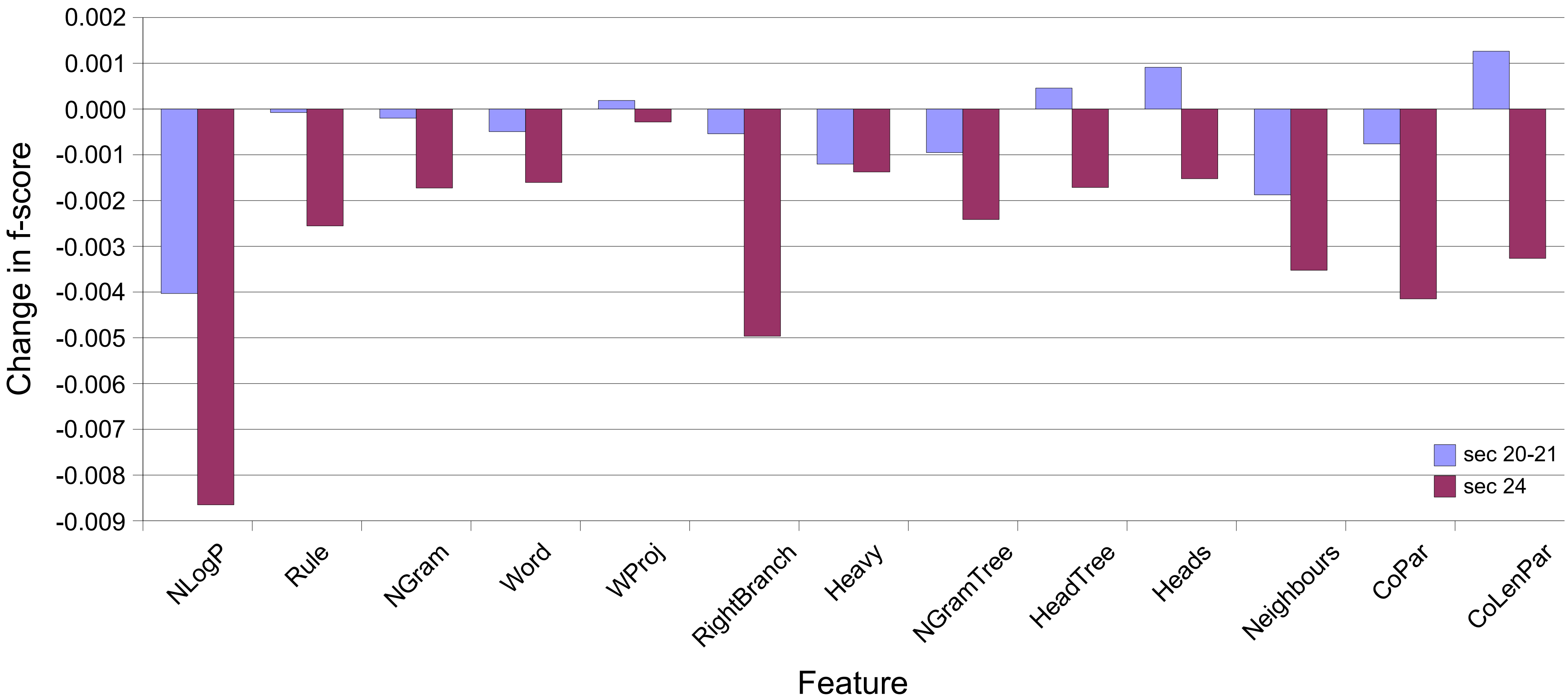
- The feature weights are not that indicative of how important a feature is
- The MaxEnt ranker with regularizer tuning takes approx 1 day to train
- The *averaged perceptron* algorithm takes approximately 2 minutes
  - used in experiments comparing different sets of features
  - all closest parses $\mathcal{T}_{\mathsf{c}}^{+}(\mathsf{t}^{\star})$ count as "correct"
  - Used to compare models with the following features:
    NLogP Rule NGram Word WProj RightBranch Heavy NGramTree HeadTree Heads Neighbours CoPar CoLenPar

Adding one feature class



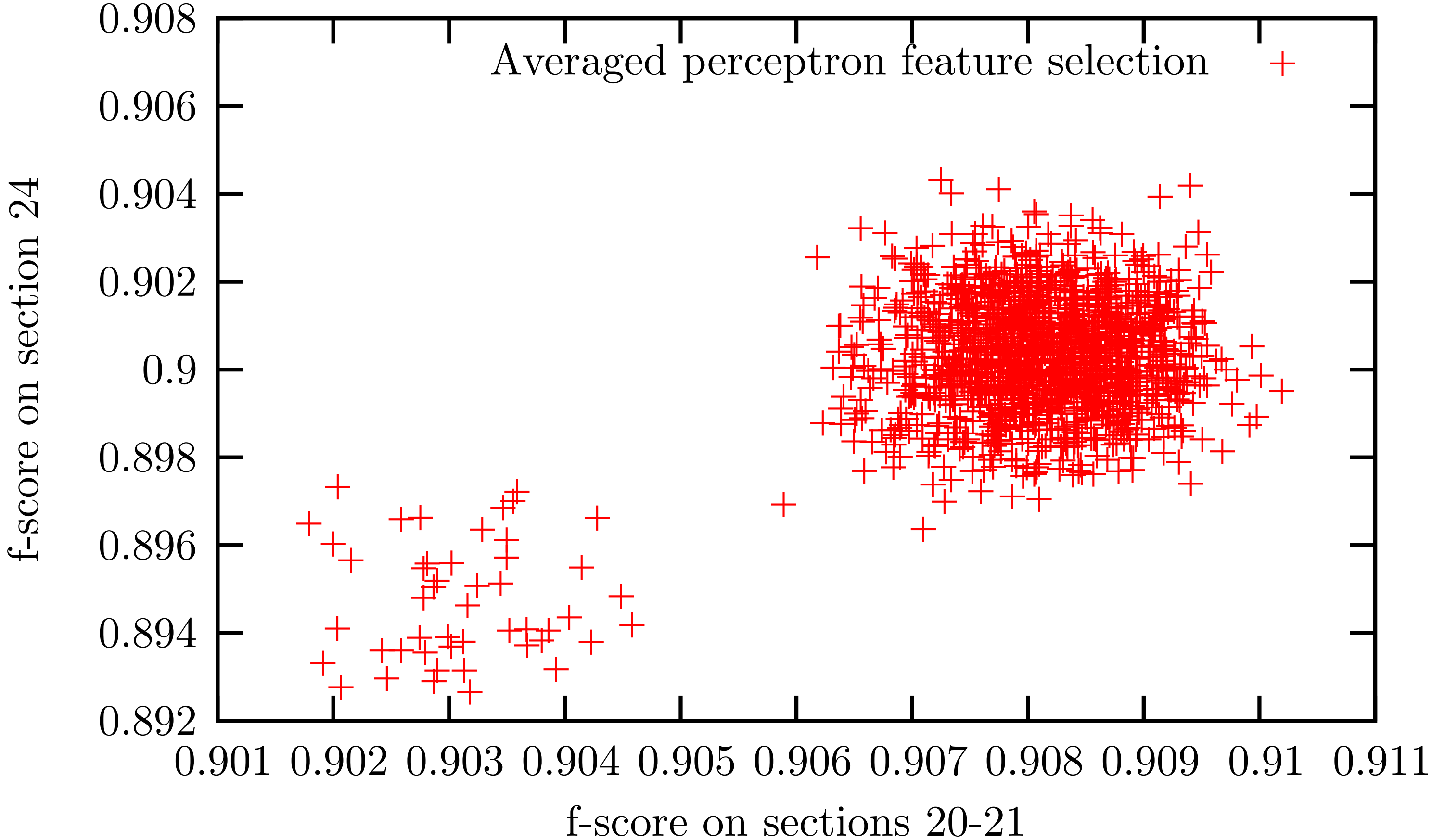
- Averaged perceptron baseline with only base parser log prob feature
  - section 20–21 f-score = 0.894913
  - section 24 f-score = 0.889901

Subtracting one feature class



- Averaged perceptron baseline with all features
  - section 20–21 f-score = 0.906806
  - section 24 f-score = 0.902782

Feature selection is hard



- Greedy feature selection using *averaged perceptron* optimizing f-score on sec 20–21
- All models also evaluated on section 24

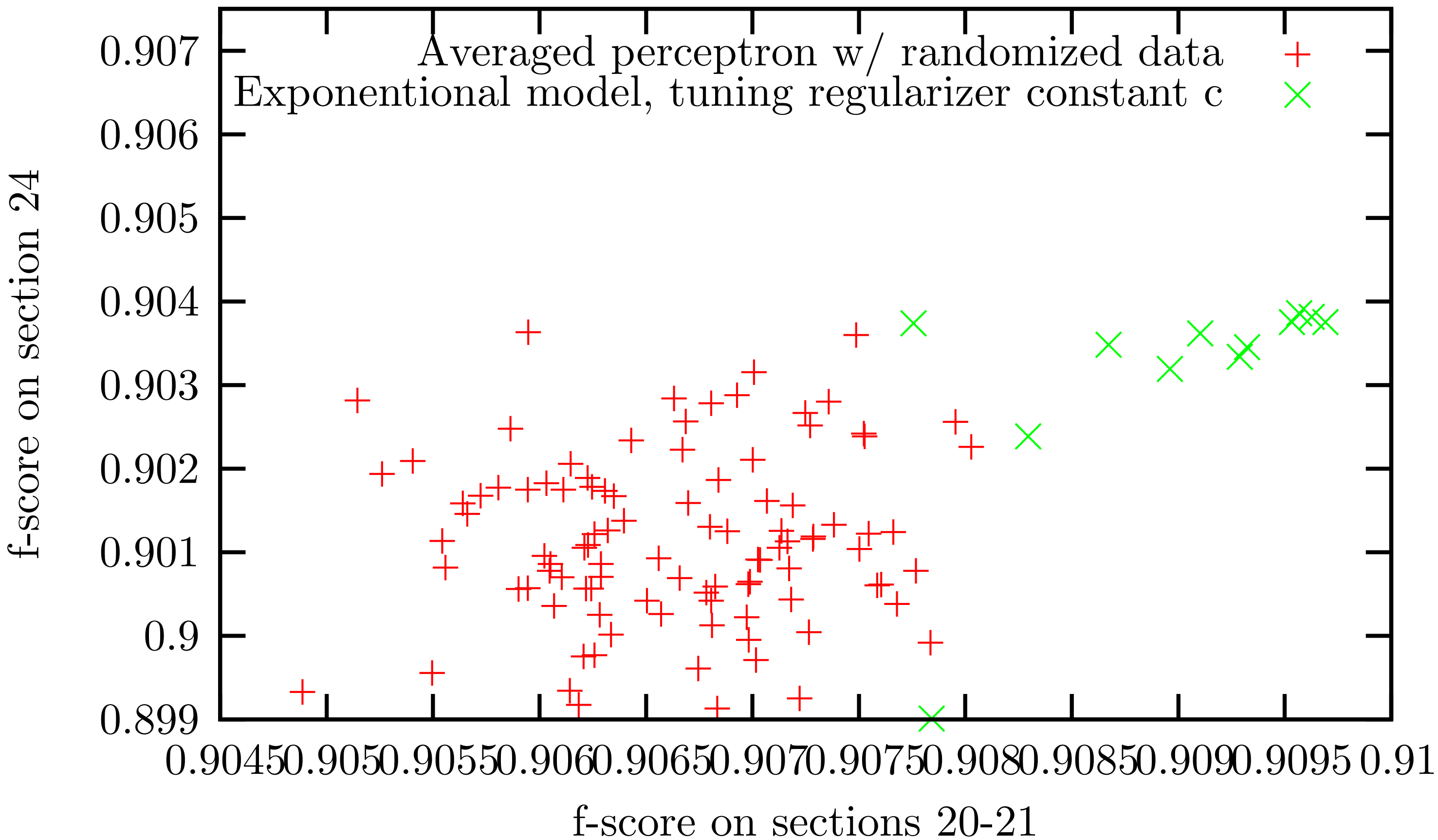
Comparing estimators

- Training on sections 2–19, regularizer tuned on 20–21, evaluate on 24

Estimator	# features	sec 20-21	sec 24
exponential model, $\mathsf{p} = 2$	670,688	0.9085	0.9037
exponential model, $\mathsf{p} = 1$	14,549	0.9078	0.9024 ($p = 0.137$)
averaged perceptron	523,374	0.9068	0.9028 ($p = 0.528$)
expected f-score	670,688	0.9084	0.9029 ($p = 0.313$)

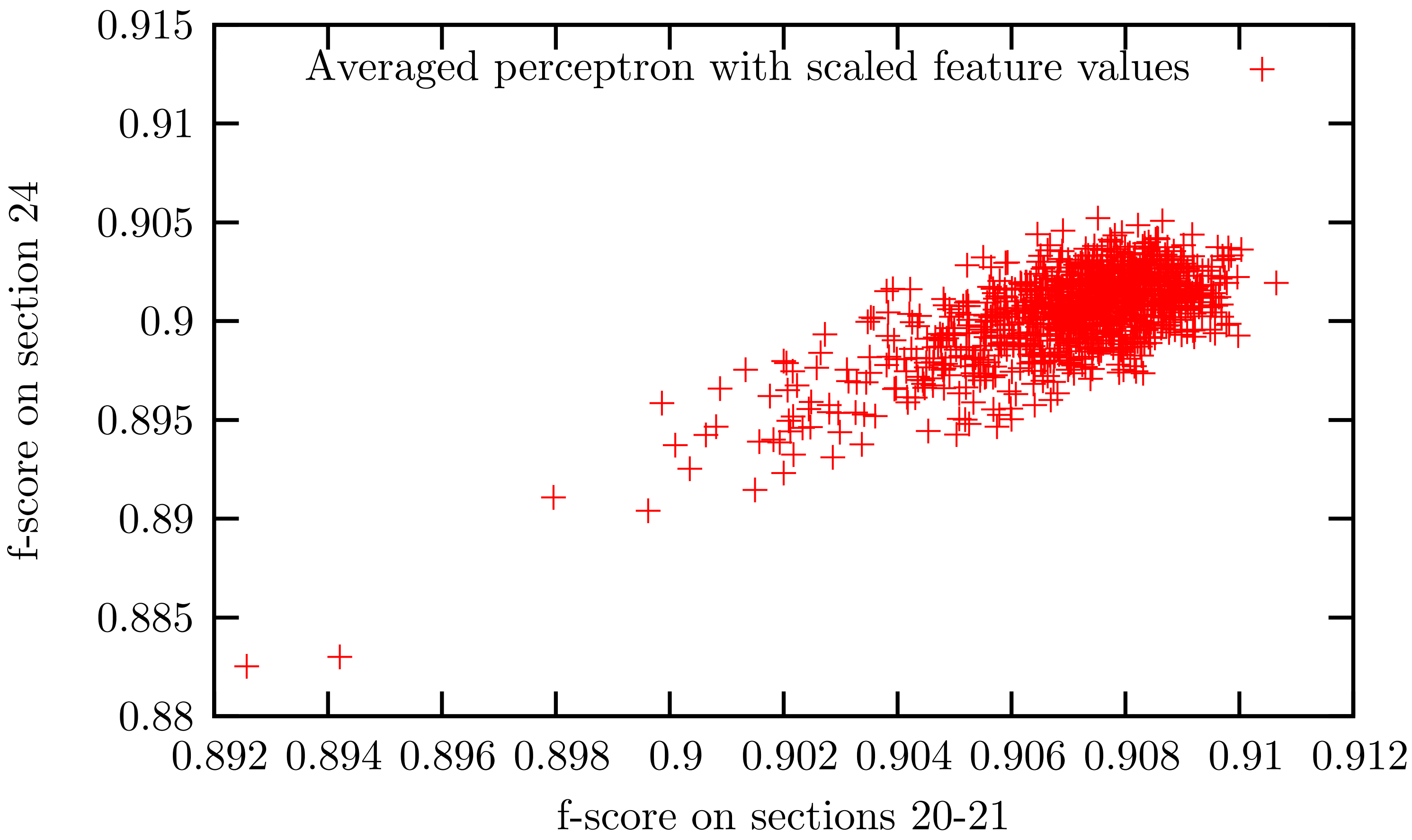
- *Because the exponential model with* $\mathsf{p} = 2$ *is usually the first model I test a new feature on, the features may be biased to work well with it.*

Averaged perceptron vs Exponentional model



- Multiple runs of *averaged perceptron* on data in random order
- Exponentional model $p = 2$ adjusting regularizer weight c

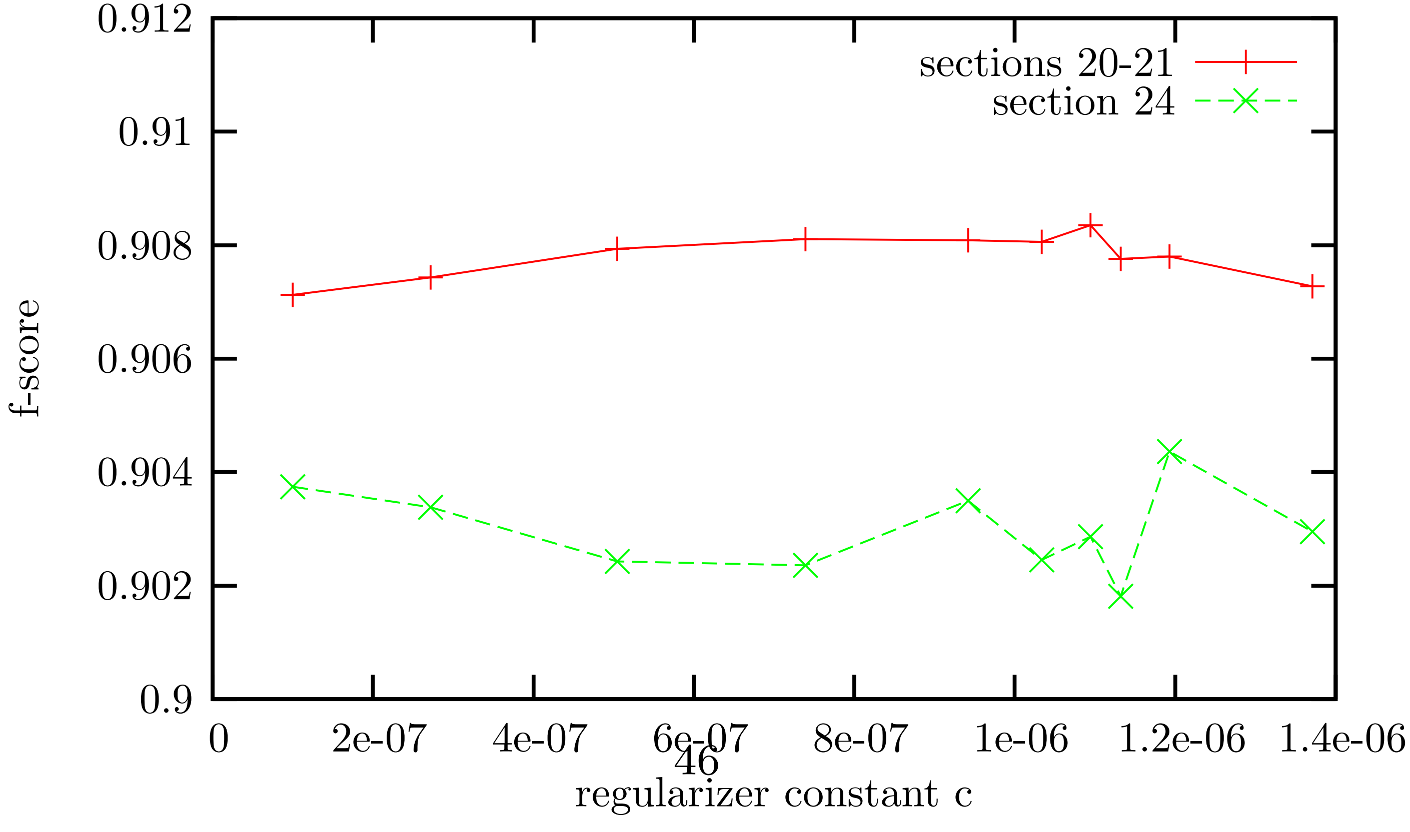
Class scaling with averaged perceptron



- Every feature class is associated with its own scaling factor
- Scaling factors adjusted to maximize av perceptron f-score on sec 20-21
- (Different features to other experiments)

Expected f-score

- The *expected f-score* is computed by calculating the *expected number of nodes* and the *expected number of correct nodes* of the parse trees in the corpus under the exponential model
- This should take *the size of the sentence* into account during training
- The expected f-score can be calculated and *differentiatiated wrt to* w



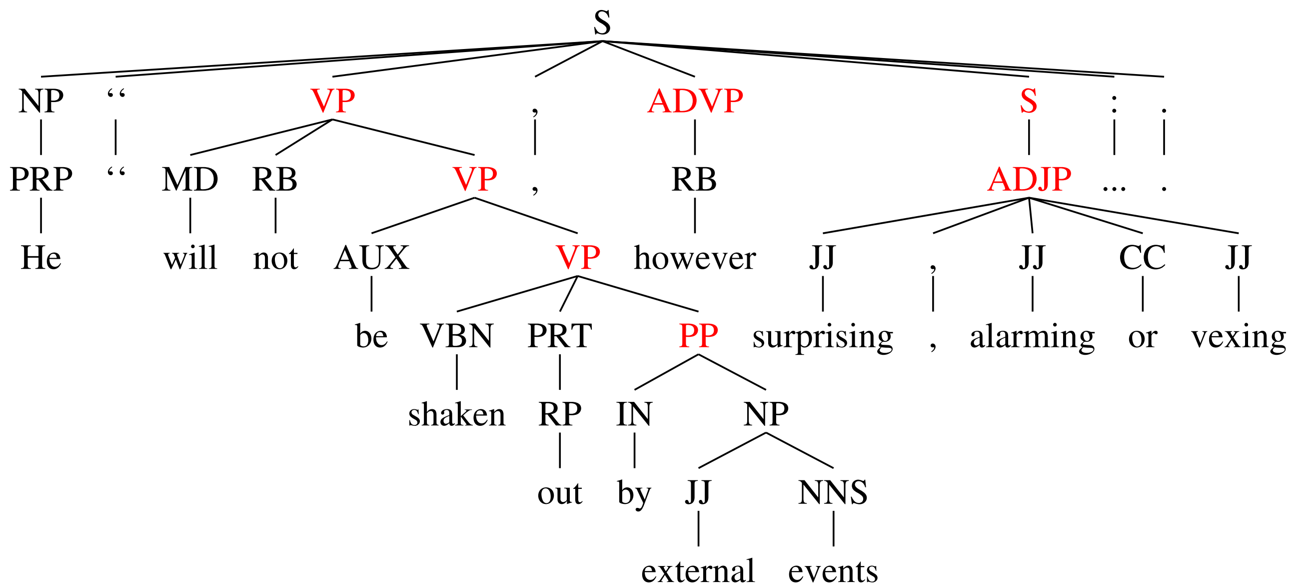
Results on all training data

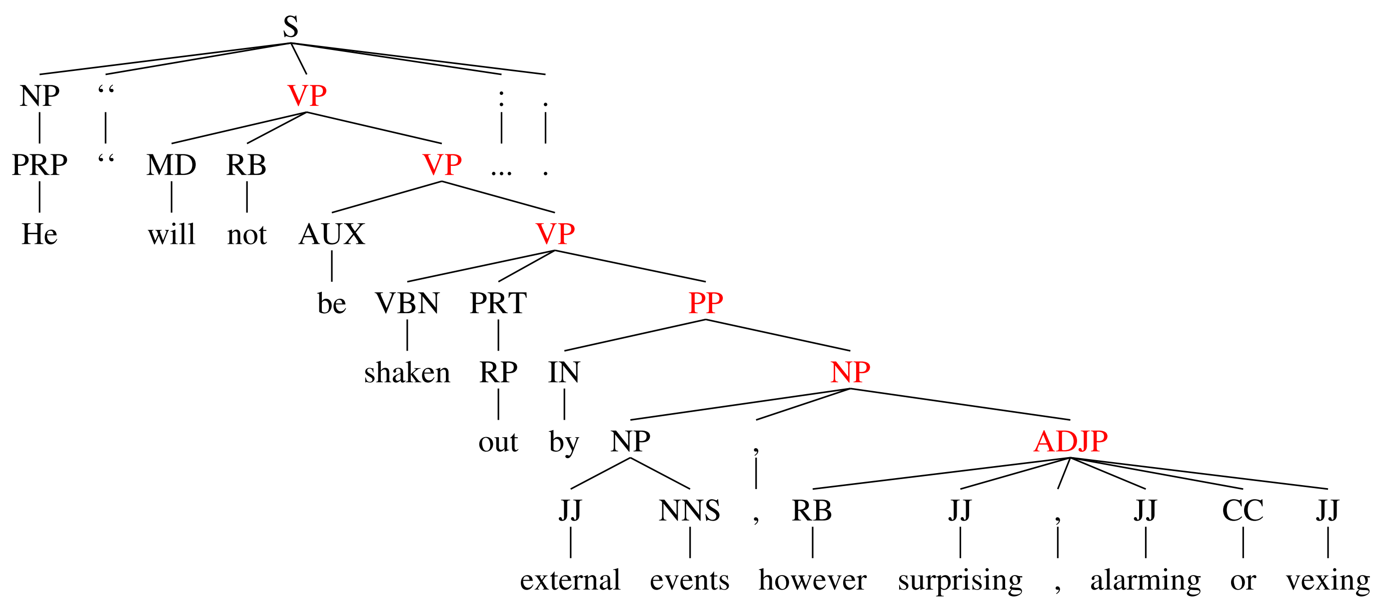
- Features must vary on parses of at least 5 sentences in training data
- In this experiment, 730,134 features
- Exponential model trained on sections 2-21
- Gaussian regularization $p = 2$, constant selected to optimize f-score on section 24
- On section 23: recall = 90.78, precision = 91.51, f-score = 91.15
- Will be available on the web this week

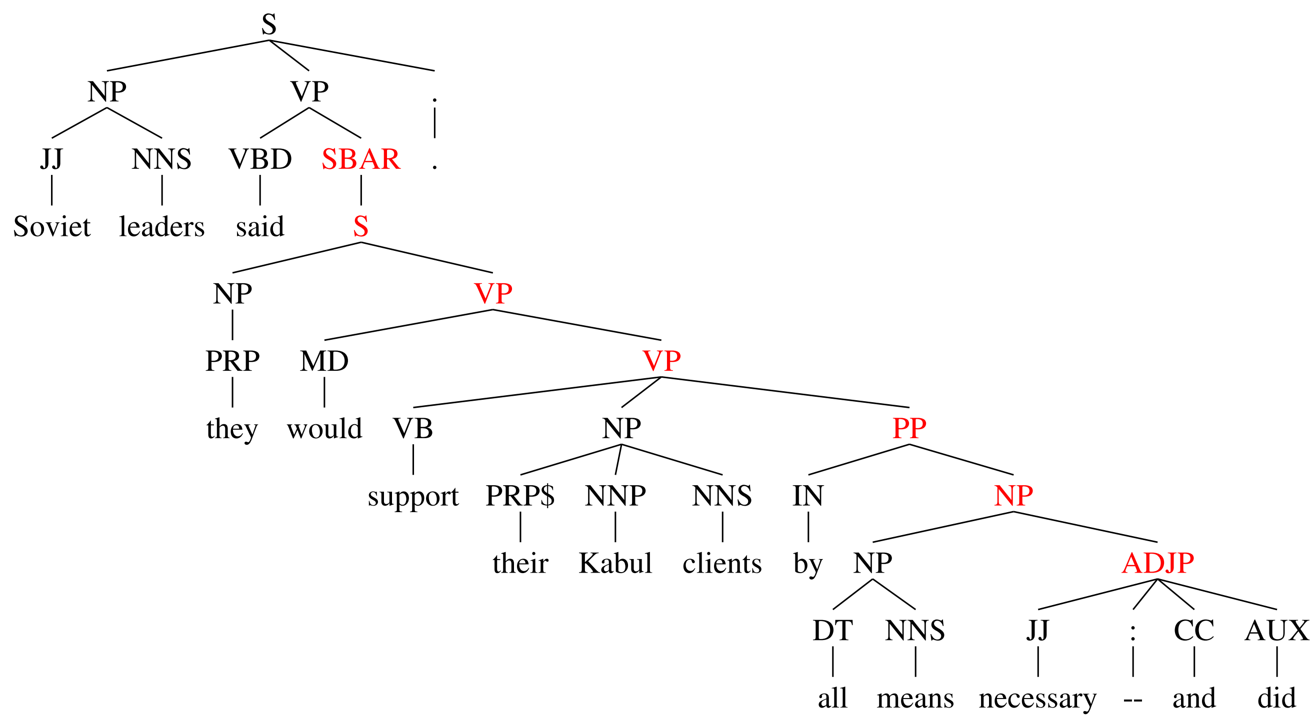
Conclusion and future work

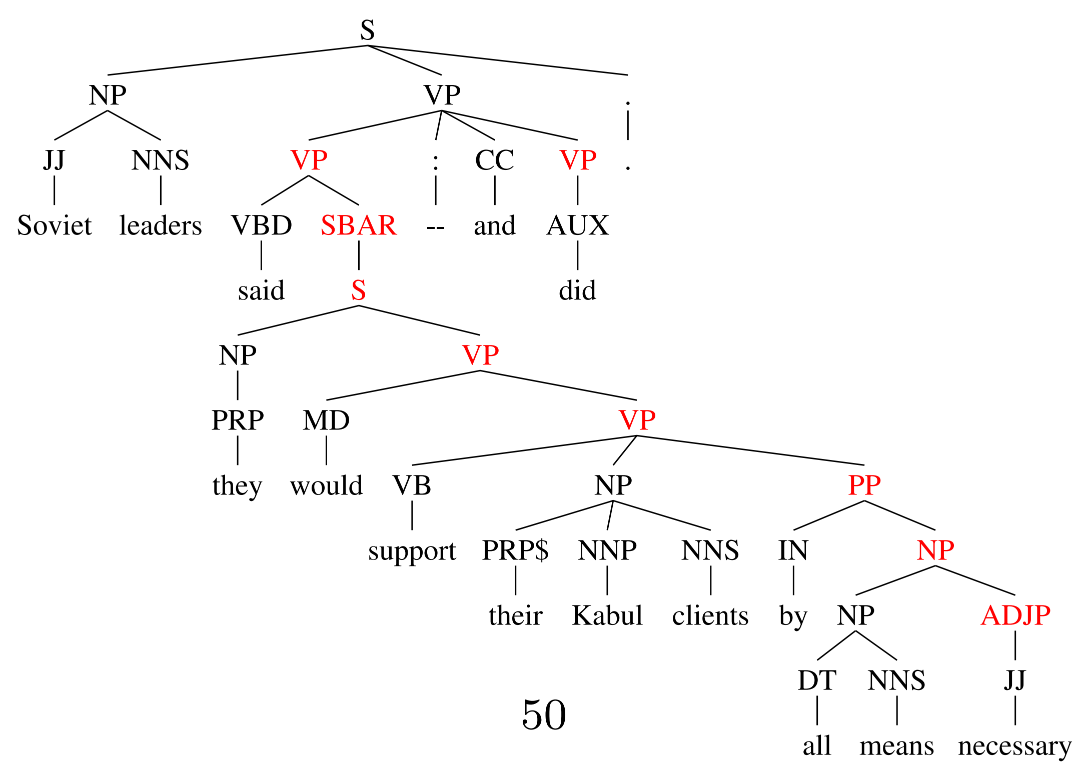
- Good features and a good machine learning algorithm can produce a state-of-the-art parser
- Good candidate trees are a big help!
- The parse ranking framework lets us explore lots of different kinds of features
  - *what a pity it's not clear which ones are important*
- Future work
  - different kinds of information (prosody, morphology, word classes)
  - richer representations (empty nodes, predicate-argument structures)
  - build discriminatively-estimated features back into Charniak parser

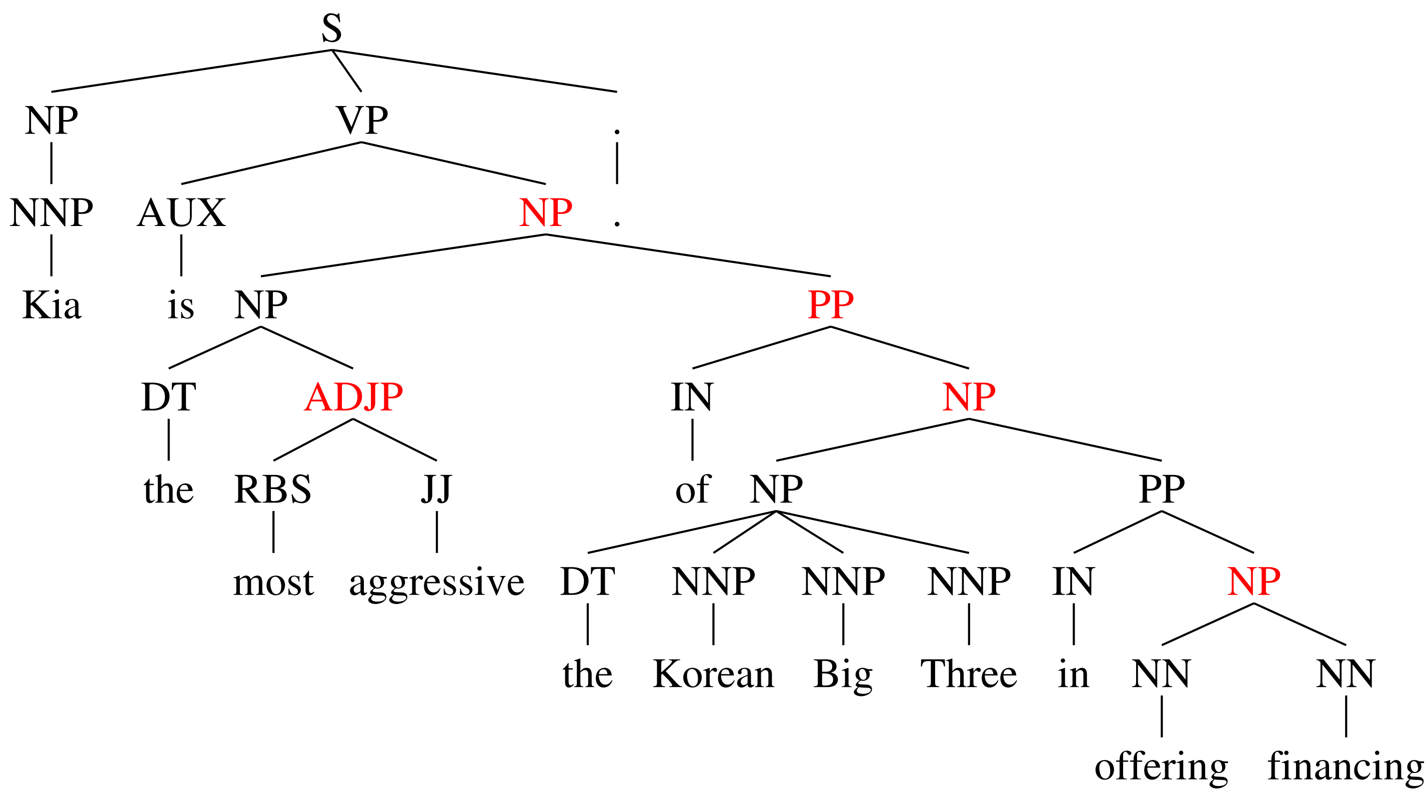
Sample parser errors

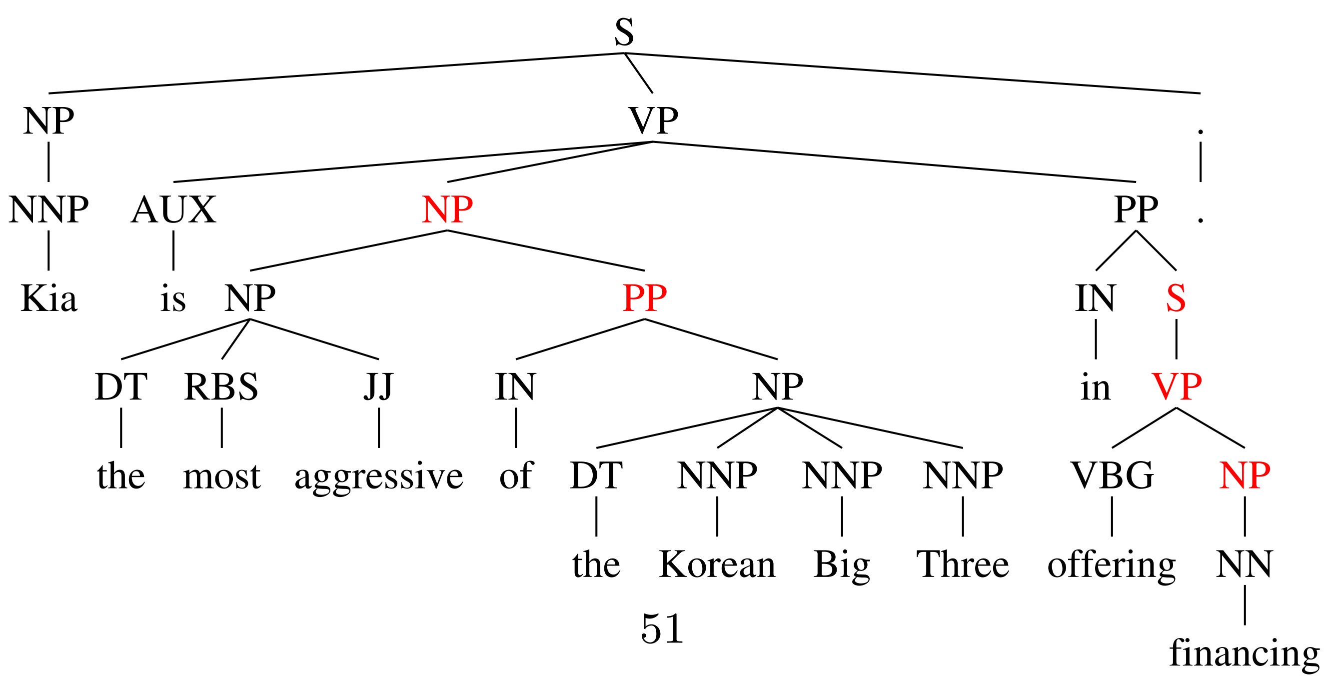


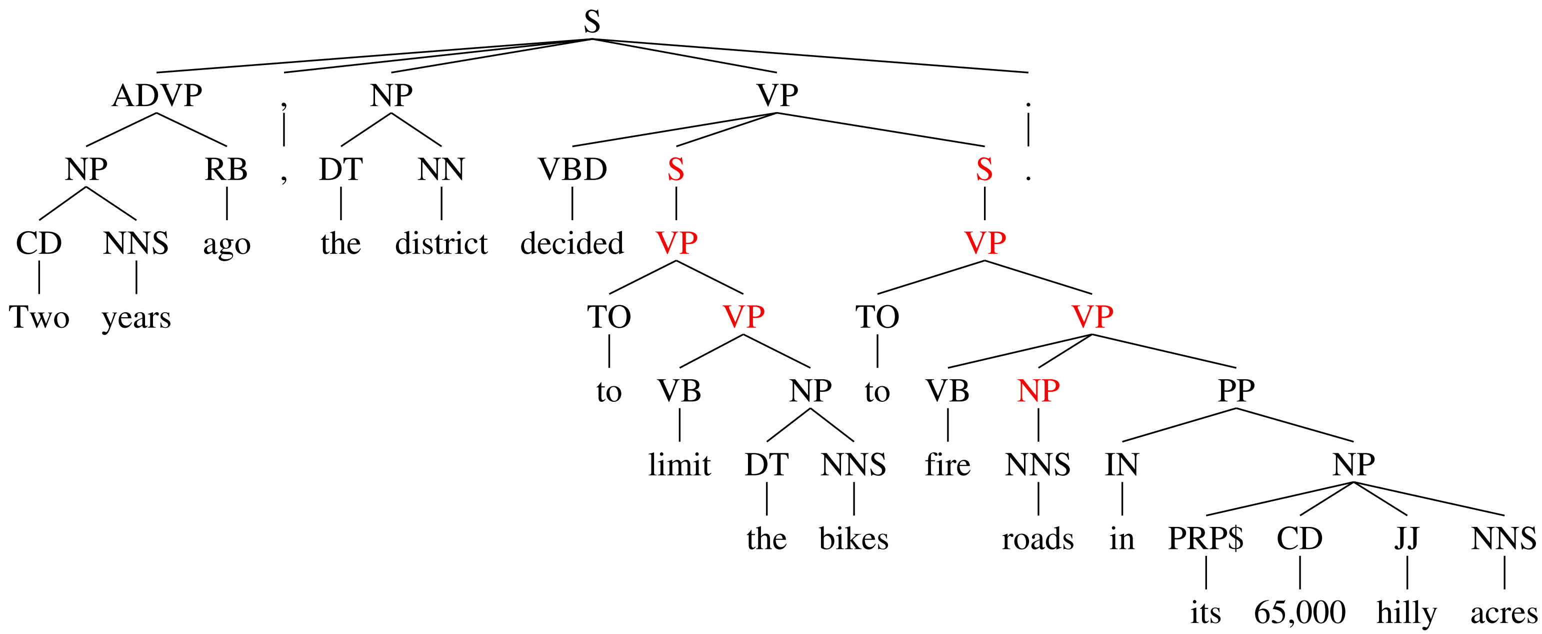


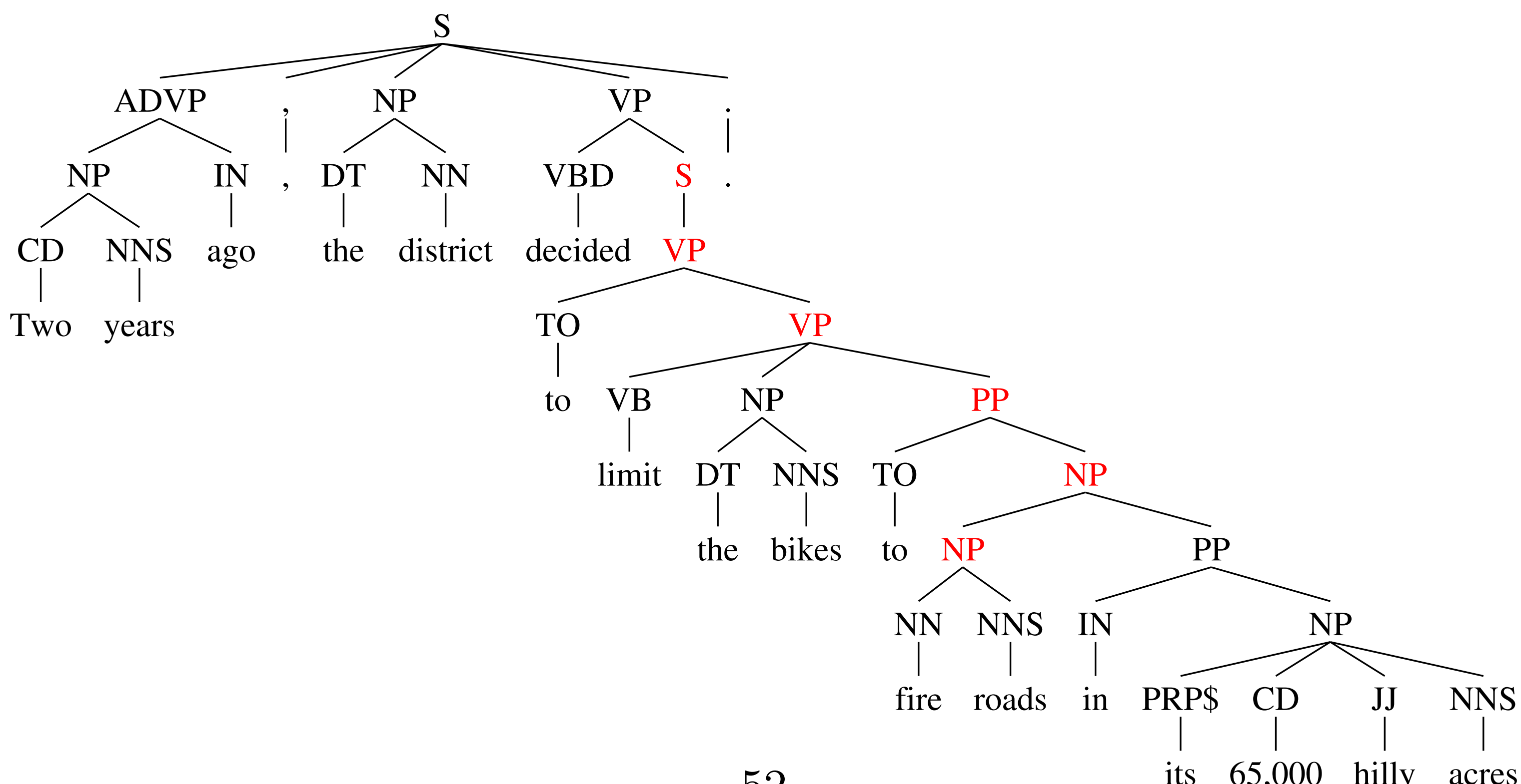

S
NP VP .
JJ NNS VBD SBAR .
Soviet leaders said S
NP VP
PRP MD VP
they would VB NP PP
support PRP$ NNP NNS IN NP
their Kabul clients by NP ADJP
DT NNS JJ : CC AUX
all means necessary -- and did

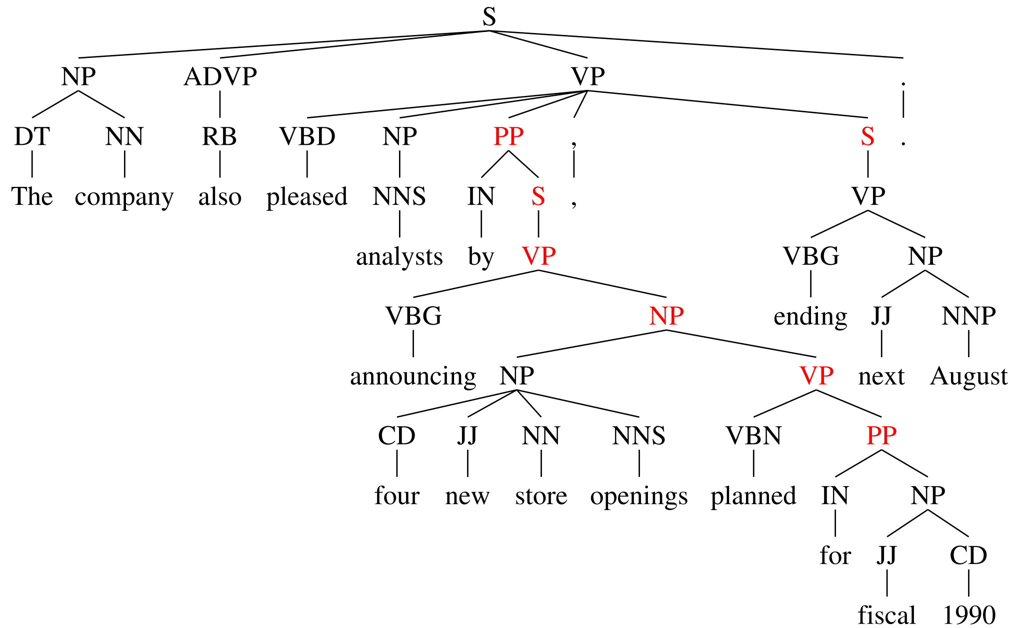

S
NP VP .
JJ NNS VP : CC VP .
Soviet leaders VBD SBAR -- and AUX
said S did
NP VP
PRP MD VP
they would VB NP PP
support PRP$ NNP NNS IN NP
their Kabul clients by NP ADJP
DT NNS JJ
all means necessary

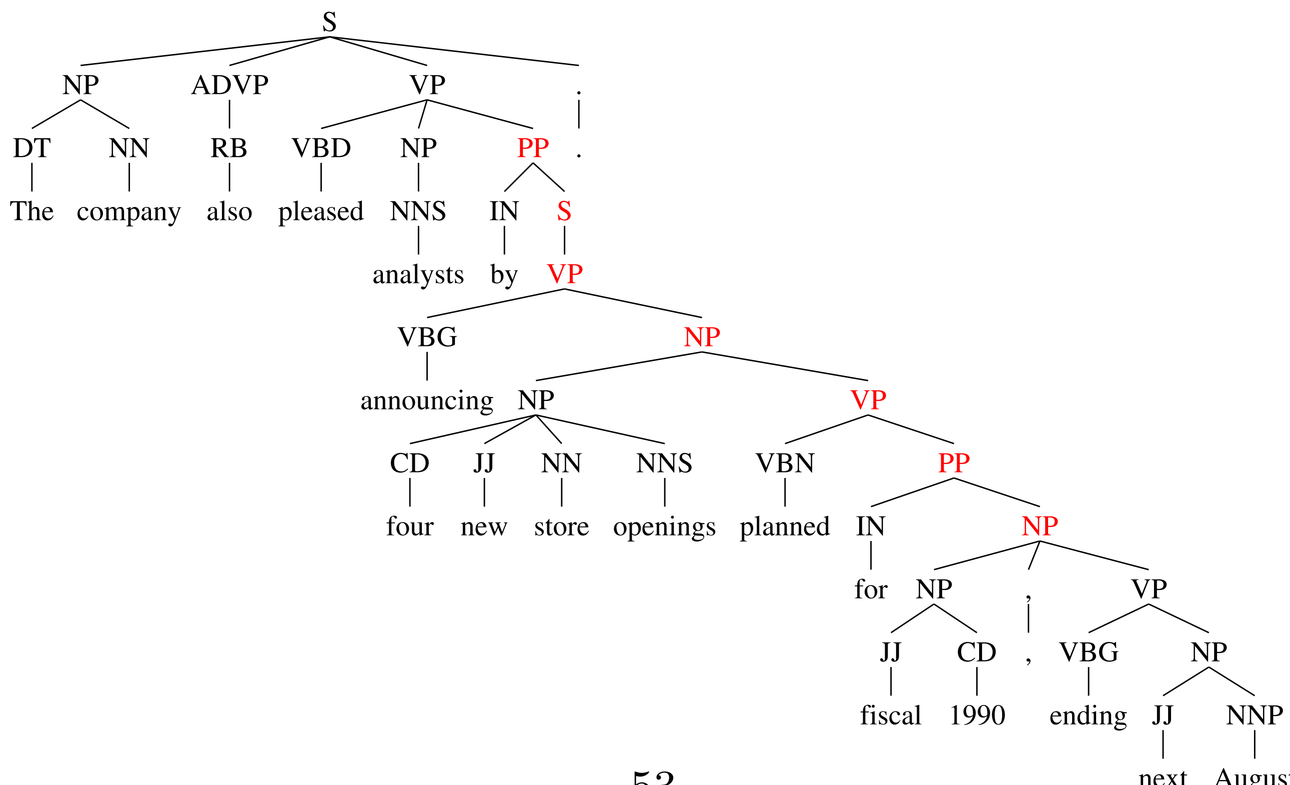

S
NP VP .
NNP AUX NP .
Kia is NP PP
DT ADJP IN NP
the RBS JJ of NP PP
most aggressive DT NNP NNP NNP IN NP
the Korean Big Three in NN NN
offering financing

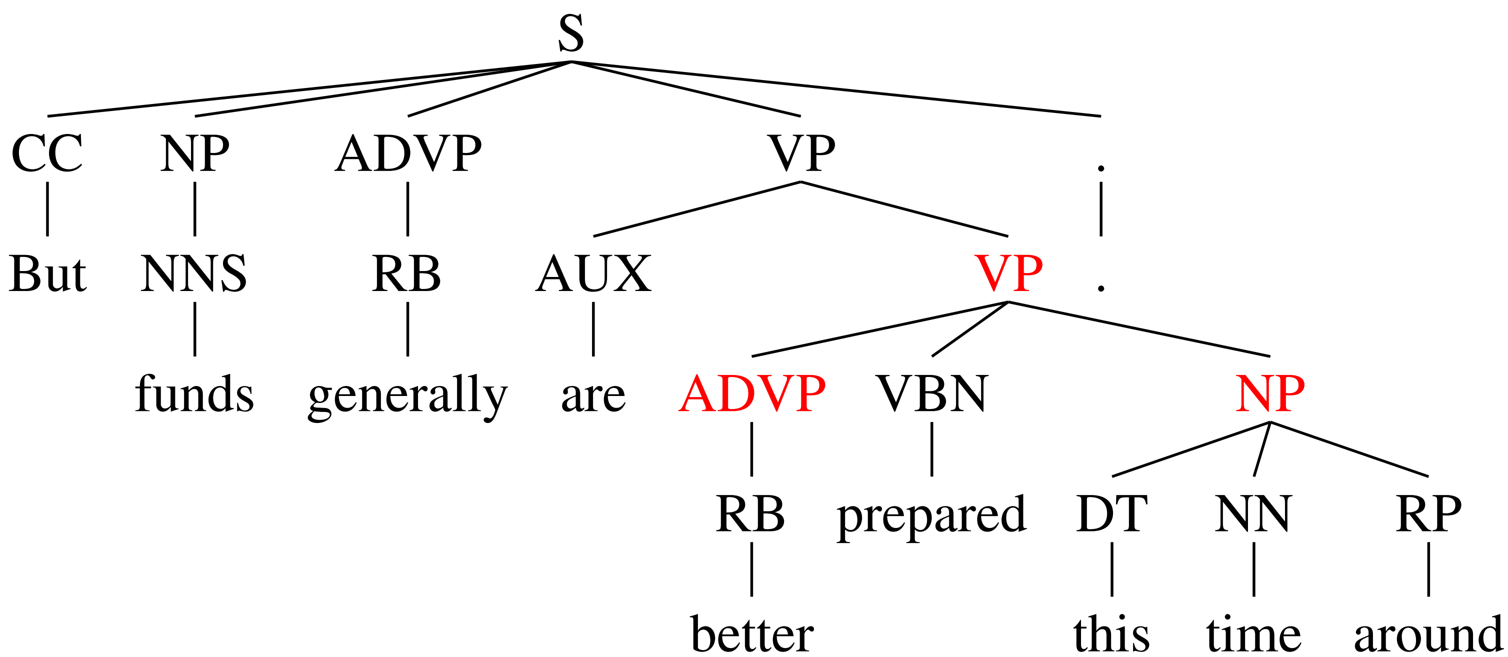

S
NP VP .
NNP AUX NP PP .
Kia is NP PP IN S
DT RBS JJ IN NP in VP
the most aggressive of DT NNP NNP NNP VBG NP
the Korean Big Three offering NN
financing

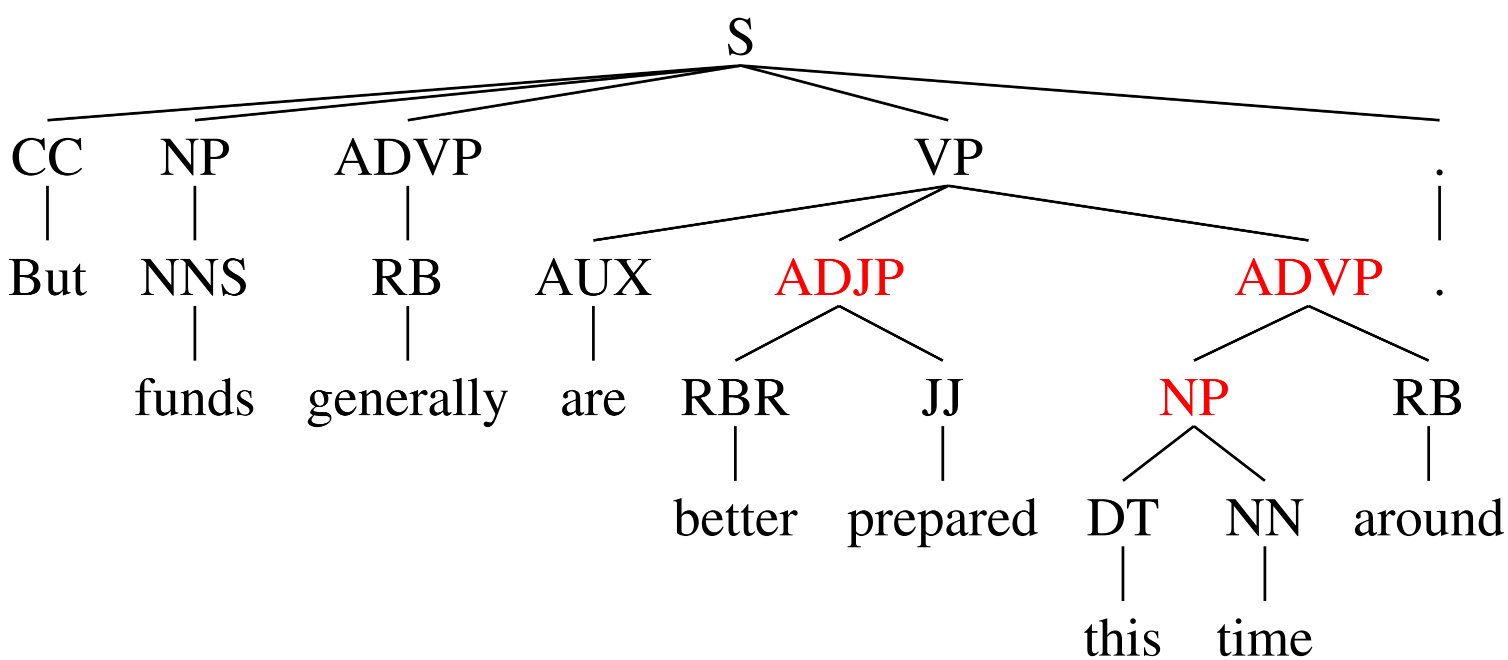

S
ADVP , NP VP .
NP RB , DT NN VBD S S .
CD NNS ago the district decided VP VP
Two years TO VP TO VP
to VB NP to VB NP PP
limit DT NNS fire NNS IN NP
the bikes roads in PRP$ CD JJ NNS
its 65,000 hilly acres

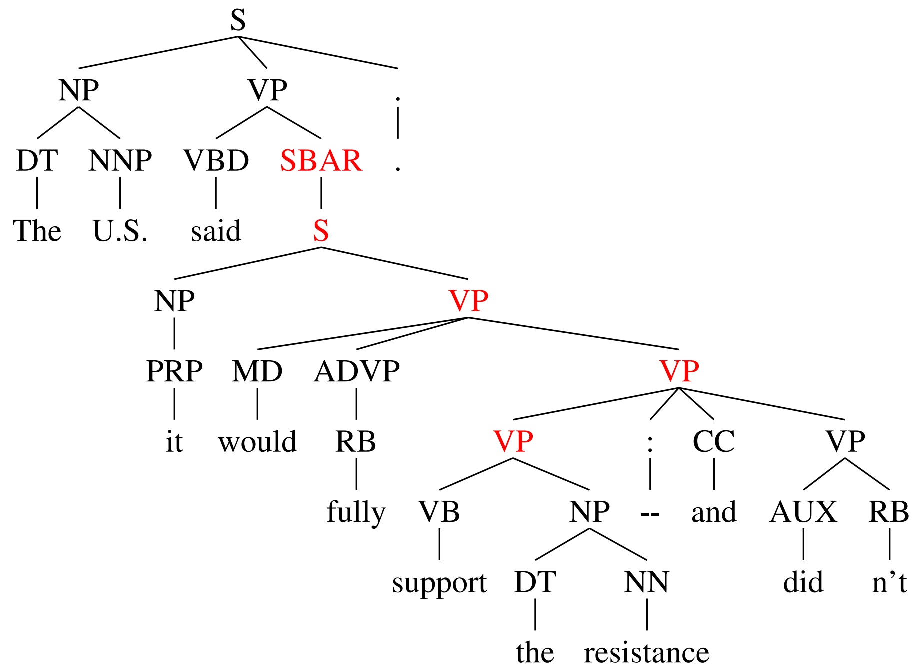

S
ADVP , NP VP .
NP IN , DT NN VBD S .
CD NNS ago the district decided VP
Two years TO VP
to VB NP PP
limit DT NNS TO NP
the bikes to NP PP
NN NNS IN NP
fire roads in PRP$ CD JJ NNS
its 65,000 hilly acres

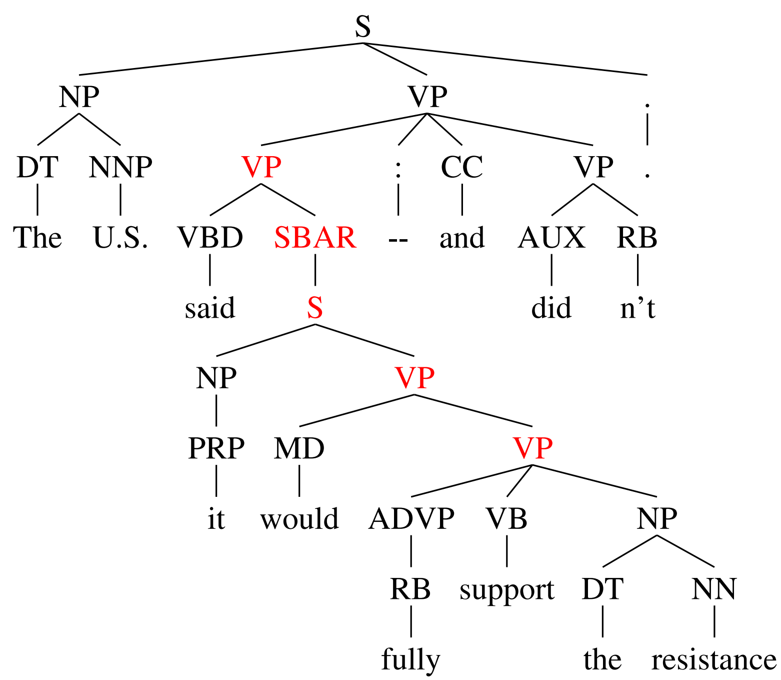

S
NP ADVP VP .
DT NN RB VBD NP PP , S .
The company also pleased NNS IN S , VP
analysts by VP VBG NP
VBG NP ending JJ NNP
announcing NP VP next August
CD JJ NN NNS VBN PP
four new store openings planned IN NP
for JJ CD
fiscal 1990


S
NP ADVP VP .
DT NN RB VBD NP PP .
The company also pleased NNS IN S
analysts by VP
VBG NP
announcing NP VP
CD JJ NN NNS VBN PP
four new store openings planned IN NP
for NP , VP
JJ CD , VBG NP
fiscal 1990 ending JJ NNP
next August


S
CC NP ADVP VP .
But NNS RB AUX VP .
funds generally are ADVP VBN NP
RB prepared DT NN RP
better this time around


S
CC NP ADVP VP .
But NNS RB AUX ADJP ADVP .
funds generally are RBR JJ NP RB
better prepared DT NN around
this time


S
NP VP .
DT NNP VBD SBAR .
The U.S. said S
NP VP
PRP MD ADVP VP
it would RB VP : CC VP
fully VB NP -- and AUX RB
support DT NN did n't
the resistance


S
NP VP .
DT NNP VP : CC VP .
The U.S. VBD SBAR -- and AUX RB
said S did n't
NP VP
PRP MD VP
it would ADVP VB NP
RB support DT NN
fully the resistance

Significance testing (av. perceptron)

comparing exponential model $p = 2$ with averaged perceptron

nsentences = 1345 in test corpus.

model 1 nfeatures = 670688, corpus f-score = 0.9037

model 2 nfeatures = 670688, corpus f-score = 0.902782

permutation test significance of corpus f-score difference = 0.58234

model 1 better on 214 = 15.9108% sentences

model 2 better on 170 = 12.6394% sentences

models 1 and 2 tied on 961 = 71% sentences

binomial 2-sided significance of sentence-by-sentence comparison = 0.0280806

bootstrap 95% confidence interval for model 1 f-scores = (0.897672 0.9096)

bootstrap 95% confidence interval for model 2 f-scores = (0.896832 0.908697)

Significance testing (p=1)

comparing exponential models $p = 2$ **with** $p = 1$

nsentences = 1345 in test corpus.

model 1 nfeatures = 670688, corpus f-score = 0.9037

model 2 nfeatures = 670688, corpus f-score = 0.902357

permutation test significance of corpus f-score difference = 0.22695

model 1 better on 121 = 8.99628% sentences

model 2 better on 98 = 7.28625% sentences

models 1 and 2 tied on 1126 = 83% sentences

binomial 2-sided significance of sentence-by-sentence comparison = 0.136934

bootstrap 95% confidence interval for model 1 f-scores = (0.897672 0.9096)

bootstrap 95% confidence interval for model 2 f-scores = (0.896315 0.908321)

Significance testing (expected f-score)

comparing exponential model $p = 2$ with expected f-score

nsentences = 1345 in test corpus.

model 1 nfeatures = 670688, corpus f-score = 0.9037

model 2 nfeatures = 670688, corpus f-score = 0.902865

permutation test significance of corpus f-score difference = 0.59533

model 1 better on 169 = 12.5651% sentences

model 2 better on 150 = 11.1524% sentences

models 1 and 2 tied on 1026 = 76% sentences

binomial 2-sided significance of sentence-by-sentence comparison = 0.313546

bootstrap 95% confidence interval for model 1 f-scores = (0.897672 0.9096)

bootstrap 95% confidence interval for model 2 f-scores = (0.89686 0.908797)

Features from correct/incorrect parses only

- Features that varied on less than 5 sentences were pruned
- Exponential model, $p = 2$

Source	**# features**	**20-21 f-score**	**24 f-score**
All parses	670,688	0.9085	0.9037
Correct parses	173,409	0.9087	0.9043
Incorrect parses	670,544	0.9085	0.9036