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

\[\Rightarrow\] 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

```
S
|--- NP
|   |--- PRP
|   |   He
|   |--- VBD
|   |   raised
|   |--- DT
|   |   the
|   |--- NN
|   |   price
```

- 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
  \[ \hat{t} = \operatorname{argmax}_{t \in \mathcal{T}_c(s)} w \cdot f(t) \]
is predicted correct
**Linear ranking example**

\[ w = (-1, 2, 1) \]

<table>
<thead>
<tr>
<th>Candidate parse tree ( t )</th>
<th>features ( f(t) )</th>
<th>parse score ( w \cdot f(t) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>(1, 3, 2)</td>
<td>7</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>(2, 2, 1)</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- 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 \( T_c(s) \) for each sentence \( s \)
- Feature functions \( f \) apply to each tree \( t \in T_c(s) \), producing *feature values* \( f(t) = (f_1(t), \ldots, f_m(t)) \)
- Return *highest scoring tree*

\[
\hat{t}(s) = \arg\max_t w \cdot f(t) = \arg\max_t \sum_{j=1}^{m} w_j f_j(t)
\]
Many models define the best candidate \( \hat{t} \) in terms of a linear combination of feature values \( \mathbf{w} \cdot \mathbf{f}(t) \):

- Exponential, Log-linear, Gibbs models, MaxEnt

\[
P(t) = \frac{1}{Z} \exp \mathbf{w} \cdot \mathbf{f}(t)
\]

\[
Z = \sum_{t \in T} \exp \mathbf{w} \cdot \mathbf{f}(t) \quad \text{(partition function)}
\]

\[
\log P(t) = \mathbf{w} \cdot \mathbf{f}(t) - \log Z
\]

- Perceptron algorithm (including averaged version)
- Support Vector Machines
- Boosted decision stubs
PCFGs are exponential models

\[ f_j(t) = \text{number of times the } j\text{th rule is used in } t \]
\[ w_j = \log p_j, \text{ where } p_j \text{ is probability of } j\text{th rule} \]

\[
\begin{align*}
S & \rightarrow NP \quad VP \\
NP & \rightarrow \text{rice} \\
VP & \rightarrow \text{grows}
\end{align*}
\]

\[ P_{PCFG}(t) = \prod_j p_j^{f_j(t)} = \prod_j \exp(w_j)^{f_j(t)} = \prod_j \exp w_j f_j(t) \]
\[ = \exp \sum_j w_j f_j(t) = \exp w \cdot f(t) \]

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
  \[ w \cdot (f(t_1) - c(s)) - w \cdot (f(t_2) - c(s)) = w \cdot f(t_1) - w \cdot f(t_2) \]
  - features that don’t vary on $T_c(s)$ are useless
  - subtract most frequently occurring value $c_j(s)$ for each feature $f_j$ in sentence $s$ \(\Rightarrow\) sparser feature vectors
## Getting the feature weights

<table>
<thead>
<tr>
<th>s</th>
<th>f(t*(s))</th>
<th>{f(t) : t ∈ ℳc(s), t ≠ t*(s)}</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence 1</td>
<td>(1, 3, 2)</td>
<td>(2, 2, 3) (3, 1, 5) (2, 6, 3)</td>
</tr>
<tr>
<td>sentence 2</td>
<td>(7, 2, 1)</td>
<td>(2, 5, 5)</td>
</tr>
<tr>
<td>sentence 3</td>
<td>(2, 4, 2)</td>
<td>(1, 1, 7) (7, 2, 1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- n-best parser produces trees ℳc(s) for each sentence s
- Treebank gives *correct tree* t*(s) ∈ ℳc(s) for sentence s
- Feature functions f apply to each tree t ∈ ℳc(s), producing *feature values* \(f(t) = (f_1(t), \ldots, f_m(t))\)
- Machine learning algorithm selects *feature weights* w to prefer t*(s) (e.g., so \(w \cdot f(t^*(s))\) is greater than \(w \cdot f(t')\) for other \(t' ∈ ℳc(s)\))
Conditional ML estimation of $w$

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

$$P_w(t|s) = \frac{1}{Z_w(s)} \exp w \cdot f(t) \quad \text{exponential model}$$

$$Z_w(s) = \sum_{t' \in \mathcal{T}_c(s)} \exp w \cdot f(t')$$

$$D = ((s_1, t_1^*), \ldots, (s_n, t_n^*)) \quad \text{treebank training data}$$

$$L_D(w) = \prod_{i=1}^{n} P_w(t_i^*|s_i) \quad \text{conditional likelihood of } D$$

$$\hat{w} = \arg\max_w L_D(w)$$
(Joint) MLE for exponential models is hard

\[ D = (t_1^*, \ldots, t_n^*) \]

\[ L_D(w) = \prod_{i=1}^{n} P_w(t_i^*) \]

\[ \hat{w} = \arg\max_w L_D(w) \]

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

- Joint MLE selects \( w \) to make \( t_i^* \) 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 \( \hat{w} \) 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) \( t^* \) 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)

\[
D = ((t_1^*, s_1), \ldots, (t_n^*, s_n))
\]

\[
L_D(w) = \prod_{i=1}^{n} P_w(t_i^*|s_i)
\]

\[
\hat{w} = \arg \max_w L_D(w)
\]

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

$$ P(t, s) = P(t|s)P(s) $$

- 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

100 × run

2 × see people with telescopes

1 × see people with telescopes

<table>
<thead>
<tr>
<th>Rule</th>
<th>count</th>
<th>rel freq</th>
<th>better vals</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP → V</td>
<td>100</td>
<td>100/105</td>
<td>4/7</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>3</td>
<td>3/105</td>
<td>1/7</td>
</tr>
<tr>
<td>VP → VP PP</td>
<td>2</td>
<td>2/105</td>
<td>2/7</td>
</tr>
<tr>
<td>NP → N</td>
<td>6</td>
<td>6/7</td>
<td>6/7</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>1</td>
<td>1/7</td>
<td>1/7</td>
</tr>
</tbody>
</table>
Regularization

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

$$
\hat{w} = \arg\min_{w} Q(w) + R(w)
$$

$$
Q(w) = -\log L_D(w) \quad \text{(objective function)}
$$

$$
R(w) = c \sum_{j} |w_j|^p \quad \text{(regularizer)}
$$

$$
L_D(w) = \prod_{i} P_w(t_i^*|s_i)
$$

- $p = 2$ 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 $t^*(s)$, choose parse $t^+(s)$ in $\mathcal{T}_c(s)$ closest to $t^*(s)$
- Maximize conditional likelihood of $(t_1^+, \ldots, t_n^+)$

- Closest parse $t_i^+ = \arg\max_{t \in \mathcal{T}(s_i)} F_{t_i^*}(t)$
  - $F_{t^*}(t)$ is *f-score* of $t$ relative to $t^*$
- $w$ 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 $T_c^+(t_i^*)$ equally close to the correct parse $t_i^*$: 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

$$L_D(w) = \prod_i P(T_c(t_i^*)|T_c(s_i))$$
Likelihood of multiple best parses

- Treebank $\mathcal{D} = ((t^*_1, s_1), \ldots , (t^*_n, s_n))$
- $n$-best candidates $\mathcal{T}_c(s_i)$ of sentence $s_i$
- $\mathcal{T}_c^+(t^*_i) = \text{trees in } \mathcal{T}_c(s_i) \text{ with max } f\text{-score}$
- $\mathcal{w}$ chosen to maximize the regularized log conditional likelihood of $\mathcal{T}_c^+(t^*_i)$

\[
L_D(\mathcal{w}) = \prod_i P_w(\mathcal{T}_c^+(t^*_i) | \mathcal{T}_c(s_i)) = \prod_i \frac{\sum_{t \in \mathcal{T}_c^+(t^*)} \exp \mathcal{w} \cdot f(t)}{\sum_{t \in \mathcal{T}_c(s_i)} \exp \mathcal{w} \cdot f(t)}
\]

- $\partial \log L / \partial w_j$ is a difference in expectations over $\mathcal{T}_c^+(t^*)$ 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)
There are at least two sensible notions of head (c.f., Grimshaw)

- **Functional heads**: determiners of NPs, auxiliary 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!

```
DT A record NN date NN has RB n’t VBN been VBN set
```
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*.
\textbf{n-gram rule features generalize rules}

- Collects \textit{adjacent constituents} in a local tree
- Also includes \textit{relationship to head} (e.g., adjacent? left or right?)
- Parameterized by \textit{ancestor-annotation}, \textit{lexicalization} and \textit{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*. 
The RightBranch feature’s value is the number of nodes on the right-most branch (ignoring punctuation) (c.f., Charniak 00)

- Reflects the tendency toward right branching in English
- Only 2 different features, but very useful in final model!
• 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.
The CoLenPar feature indicates the difference in length in adjacent conjuncts and whether this pair contains the last conjunct.

4 words

CoLenPar feature: (2,true)  6 words
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
A Neighbours feature indicates the node’s category, its binned length and j left and k right POS tags for $j, k \leq 1$.
A tree $n$-gram feature is a tree fragment that connects 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
- $T_c(s)$ 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 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!
- 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 $T_c^+(t^*)$ 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
• 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

<table>
<thead>
<tr>
<th>Estimator</th>
<th># features</th>
<th>sec 20-21</th>
<th>sec 24</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponential model, ( p = 2 )</td>
<td>670,688</td>
<td>0.9085</td>
<td>0.9037</td>
</tr>
<tr>
<td>exponential model, ( p = 1 )</td>
<td>14,549</td>
<td>0.9078</td>
<td>0.9024 (( p = 0.137 ))</td>
</tr>
<tr>
<td>averaged perceptron</td>
<td>523,374</td>
<td>0.9068</td>
<td>0.9028 (( p = 0.528 ))</td>
</tr>
<tr>
<td>expected f-score</td>
<td>670,688</td>
<td>0.9084</td>
<td>0.9029 (( p = 0.313 ))</td>
</tr>
</tbody>
</table>

- Because the exponential model with \( p = 2 \) is usually the first model I test a new feature on, the features may be biased to work well with it.
Multiple runs of *averaged perceptron* on data in random order

- Exponential model $p = 2$ adjusting regularizer weight $c$
• 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 differentiated 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
Soviet leaders said they would support their Kabul clients by all means necessary -- and did.
Kia is the most aggressive of the Korean Big Three in offering financing.
Two years ago, the district decided to limit the bikes to fire roads in its 65,000 hilly acres.
The company also pleased analysts by announcing four new store openings planned for fiscal 1990, ending next August.
But funds generally are better prepared this time around.
The U.S. said it would fully support the resistance: and did n't.
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 = (0.896860 0.908797)
Features from correct/incorrect parses only

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

<table>
<thead>
<tr>
<th>Source</th>
<th># features</th>
<th>20-21 f-score</th>
<th>24 f-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>All parses</td>
<td>670,688</td>
<td>0.9085</td>
<td>0.9037</td>
</tr>
<tr>
<td>Correct parses</td>
<td>173,409</td>
<td>0.9087</td>
<td>0.9043</td>
</tr>
<tr>
<td>Incorrect parses</td>
<td>670,544</td>
<td>0.9085</td>
<td>0.9036</td>
</tr>
</tbody>
</table>