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
let PRON us VPv take NP DATEP Tuesday , the fifteenth.

SENTENCE_ID BAC002_E

[ANIM + CASE ACC NUM PL PERS 1 PRED PRO PRON-FORM WE PRON-TYPE PERS 9]

PASSIVE-
PRED LET &lt;2,10&gt;9
STMT-TYPE IMPERATIVE

SUBJ [PERS 2 PRED PRO 2 PRON-TYPE NULL]

TNS-ASP [MOOD IMPERATIVE]

[ANIM- NUM SG PRED fifteen SPEC SPEC-FORM THE SPEC-TYPE DEF]

OBJ [CASE ACC GEND NEUT NTYPE GRAIN COUNT PROPER DATE]

APP NUM SG PERS 3 PRED TUESDAY

XCOMP OBJ [CASE ACC GEND NEUT NTYPE NUMBER ORD TIME DATE]

NUM SG PERS 9 PRED TAKE &lt;9,13&gt; SUBJ 10
Corpus-derived grammars

- Grammar is extracted automatically from a large linguistically annotated corpus
  - Focuses on frequently occurring 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

ROOT

NP-SBJ

NNP BELL INDUSTRIES Inc. NNP increased

VP

PP-DIR

PP-DIR

NP its quarterly

PRP$ to

TO

NP

IN

NP

CD 10 cents

NNS

from

CD NNS DT a

NP-ADV

share

NN NNS seven cents
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

- Probabilistic context-free grammars (PCFGs) associate a *rule probability* $p(r)$ with each rule ⇒ features are *local trees*

- Probability of a tree $y$ is $P(y) = \prod_{r \in y} p(r) = \prod_{r} p(r)^{f_r(y)}$ where $f_r(y)$ is the number of times $r$ appears in $y$

- Probability of a string $x$ is $P(x) = \sum_{y \in \mathcal{Y}(x)} P(y)$
Lexicalized PCFGs

- **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

\[
P \left( \begin{array}{c} S \\ NP \\ rice \\ VP \\ grows \end{array} \right) = \frac{2}{3}
\]

<table>
<thead>
<tr>
<th>Rule</th>
<th>Count</th>
<th>Rel Freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NP → rice</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>NP → corn</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>VP → grows</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
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 $\hat{p} = \arg\max_p \prod_{i=1}^{n} P_p(y_i)$.
- 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

\[
P\left( \begin{array}{c} \text{NP} \\ \text{rice} \end{array} \begin{array}{c} \text{VP} \\ \text{grows} \end{array} \right) = \frac{4}{9}
\]

<table>
<thead>
<tr>
<th>rule</th>
<th>count</th>
<th>rel freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NP → rice</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>NP → bananas</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>VP → grows</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>VP → grow</td>
<td>1</td>
<td>1/3</td>
</tr>
</tbody>
</table>

\[
P\left( \begin{array}{c} \text{NP} \\ \text{bananas} \\ \text{VP} \\ \text{grow} \end{array} \right) = \frac{1}{9}
\]

\[Z = \frac{5}{9}\]
Renormalization

\[
P \left( \begin{array}{c}
  S \\
  \:\: NP \\
  \:\: rice
  \\
  \:\: VP \\
  \:\: grows
\end{array} \right) = \frac{4}{9} \quad \frac{4}{5}
\]

\[
P \left( \begin{array}{c}
  S \\
  \:\: NP \\
  \:\: rice \\
  \:\: VP \\
  \:\: grows
\end{array} \right) = \frac{1}{9} \quad \frac{1}{5}
\]

\[Z = \frac{5}{9}\]

<table>
<thead>
<tr>
<th>rule</th>
<th>count</th>
<th>rel freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>NP → rice</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>NP → bananas</td>
<td>1</td>
<td>1/3</td>
</tr>
<tr>
<td>VP → grows</td>
<td>2</td>
<td>2/3</td>
</tr>
<tr>
<td>VP → grow</td>
<td>1</td>
<td>1/3</td>
</tr>
</tbody>
</table>
Other values do better!

\[
\begin{align*}
\text{rule} & \quad \text{count} & \text{rel freq} \\
S \rightarrow \text{NP VP} & \quad 3 & 1 \\
\text{NP} \rightarrow \text{rice} & \quad 2 & 2/3 \\
\text{NP} \rightarrow \text{bananas} & \quad 1 & 1/3 \\
\text{VP} \rightarrow \text{grows} & \quad 2 & 1/2 \\
\text{VP} \rightarrow \text{grow} & \quad 1 & 1/2 \\
(\text{Abney 1997}) & & \\
\end{align*}
\]

\[
P \left( \begin{array}{c}
\text{S} \\
\text{NP} \\
\text{VP} \\
\text{rice} \\
grows \\
\end{array} \right) = \frac{2}{6} \quad \frac{2}{3}
\]

\[
P \left( \begin{array}{c}
\text{S} \\
\text{NP} \\
\text{VP} \\
\text{bananas} \\
grow \\
\end{array} \right) = \frac{1}{6} \quad \frac{1}{3}
\]

\[
Z = \frac{3}{6}
\]
## Make dependencies local – GPSG-style

<table>
<thead>
<tr>
<th>rule</th>
<th>count</th>
<th>rel freq</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S \rightarrow \text{NP} +\text{singular}$ $\text{VP} +\text{singular}$</td>
<td>2</td>
<td>$\frac{2}{3}$</td>
<td>$\left( \begin{array}{c} S \ \text{NP} +\text{singular} \ \text{VP} +\text{singular} \ \text{rice} \ \text{grows} \end{array} \right)$ $= \frac{2}{3}$</td>
</tr>
<tr>
<td>$S \rightarrow \text{NP} +\text{plural}$ $\text{VP} +\text{plural}$</td>
<td>1</td>
<td>$\frac{1}{3}$</td>
<td>$\left( \begin{array}{c} S \ \text{NP} +\text{plural} \ \text{VP} +\text{plural} \ \text{bananas} \ \text{grow} \end{array} \right)$ $= \frac{1}{3}$</td>
</tr>
<tr>
<td>$\text{NP} +\text{singular}$ $\rightarrow$ $\text{rice}$</td>
<td>2</td>
<td>$1$</td>
<td>$\left( \begin{array}{c} S \ \text{NP} +\text{singular} \ \text{VP} +\text{singular} \ \text{rice} \ \text{grows} \end{array} \right)$</td>
</tr>
<tr>
<td>$\text{NP} +\text{plural}$ $\rightarrow$ $\text{bananas}$</td>
<td>1</td>
<td>$1$</td>
<td>$\left( \begin{array}{c} S \ \text{NP} +\text{plural} \ \text{VP} +\text{plural} \ \text{bananas} \ \text{grow} \end{array} \right)$</td>
</tr>
<tr>
<td>$\text{VP} +\text{singular}$ $\rightarrow$ $\text{grows}$</td>
<td>2</td>
<td>$1$</td>
<td>$\left( \begin{array}{c} S \ \text{NP} +\text{singular} \ \text{VP} +\text{singular} \ \text{rice} \ \text{grows} \end{array} \right)$</td>
</tr>
<tr>
<td>$\text{VP} +\text{plural}$ $\rightarrow$ $\text{grow}$</td>
<td>1</td>
<td>$1$</td>
<td>$\left( \begin{array}{c} S \ \text{NP} +\text{plural} \ \text{VP} +\text{plural} \ \text{bananas} \ \text{grow} \end{array} \right)$</td>
</tr>
</tbody>
</table>
Maximum entropy or log linear models

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

\[
S_w(y) = \sum_{j=1}^{m} w_j f_j(y) \\
V_w(y) = \exp S_w(y) \\
Z_w = \sum_{y \in \mathcal{Y}} V_w(y) \\
P_w(y) = \frac{V_w(y)}{Z_w} = \frac{1}{Z_w} \exp \sum_{j=1}^{m} w_j f_j(y) \\
\log P_\lambda(y) = \sum_{j=1}^{m} w_j f_j(y) - \log Z_w
\]
PCFGs are log-linear models

\[ \mathcal{Y} = \text{set of all trees generated by grammar } G \]

\[ f_j(y) = \text{number of times the } j\text{th rule is used in } y \in \mathcal{Y} \]

\[ p(r_j) = \text{probability of } j\text{th rule in } G \]

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

\[
\begin{align*}
    f \begin{pmatrix}
        S \\
        \begin{array}{c}
            \text{NP} \\
            \text{VP}
        \end{array}
    \end{pmatrix}
        &= 
    \begin{array}{c}
        1 \\
        1 \\
        0 \\
        1 \\
        0
    \end{array} \\
    &\text{S } \rightarrow \text{NP VP } \text{NP } \rightarrow \text{rice } \text{NP } \rightarrow \text{bananas } \text{VP } \rightarrow \text{grows } \text{VP } \rightarrow \text{grow}
\end{align*}
\]

\[
    P_{w}(y) = \prod_{j=1}^{m} p(r_j)^{f_j(y)} = \prod_{j=1}^{m} (\exp w_j)^{f_j(y)} = \exp(\sum_{j=1}^{m} w_j f_j(\omega))
\]

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

$$\hat{w} = \arg\max_w L_D(w), \text{ where}$$

$$\log L_D(w) = \sum_{i=1}^{n} \log P_w(y_i) = \sum_{i=1}^{n} (S_w(y_i) - \log Z_w)$$

• 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 $L_D(w) = P_w(x_1, y_1) \cdots P_w(x_n, y_n)$
- The PCFG MLE is a *generative model* that models the distribution of strings $P(x)$ as well as trees given strings $P(y|x)$
- The conditional distribution $P(y|x)$ is important for parsing, but the marginal distribution $P(x)$ is not
- Generative models “waste” some of their parameters to model the marginal distribution $P(x)$
- Optimize *conditional likelihood* $L'_D(w) = P_w(y_1|x_1) \cdots P_w(y_n|x_n)$
Generative vs discriminative training

When the PCFG independence assumptions are violated, the MLE may not accurately model $P(y|x)$. 

- $\frac{95}{100} \times x \quad 2\times a \quad \frac{1}{100} \times a \ b$ 
  
- Rule count rel freq rel freq 
  - $A \rightarrow x \quad 95 \quad \frac{95}{100} \quad \frac{69}{100}$ 
  - $A \rightarrow A \ b \quad 2 \quad \frac{2}{100} \quad \frac{1}{10}$ 
  - $A \rightarrow a \quad 2 \quad \frac{2}{100} \quad \frac{2}{10}$ 
  - $A \rightarrow a \ b \quad 1 \quad \frac{1}{100} \quad \frac{1}{100}$
Linguistic example of discriminative training

100 ×

VP
V
run

2 ×

VP
V
see
NP
people
PP
with
NP
with
PP
telescopes
telescopes

1 ×

VP
V
see
NP
people
PP
with
NP
with
PP
telescopes
telescopes

... × 2/105 × ...

... × 2/7 × ...

... × 1/7 × ...

... × 1/7 × ...

<table>
<thead>
<tr>
<th>Rule</th>
<th>count</th>
<th>rel freq</th>
<th>rel freq</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>
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)

$$
\mathcal{Y}(x_i) = \{y : X(y) = X(y_i)\}
$$

$$
\hat{w} = \arg\max_{\lambda} PL_D(w)
$$

$$
PL_D(w) = \prod_{i=1}^{n} P_{\lambda}(y_i|x_i)
$$

$$
P_w(y|x) = \frac{V_w(y)}{Z_w(x)}
$$

$$
V_w(y) = \exp\sum_j w_j f_j(y)
$$

$$
Z_w(x) = \sum_{y'\in \mathcal{Y}(x)} V_w(y')
$$
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 $P(x)$ and $P(y|x)$
- Conditional ML estimates a conditional distribution
  - learns $P(y|x)$ but not $P(x)$
  - conditional distribution is what you need for parsing
  - cognitively more plausible?
- Conditional estimation requires labelled training data: no obvious EM extension
Conditional estimation

<table>
<thead>
<tr>
<th></th>
<th>Correct parse’s features</th>
<th>All other parses’ features</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>

- 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

<table>
<thead>
<tr>
<th></th>
<th>Correct parse’s features</th>
<th>All other parses’ features</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence 1</td>
<td>[1, 3, 2]</td>
<td>[2, 2, 2] [3, 1, 2] [2, 6, 2]</td>
</tr>
<tr>
<td>sentence 2</td>
<td>[7, 2, 5]</td>
<td>[2, 5, 5]</td>
</tr>
<tr>
<td>sentence 3</td>
<td>[2, 4, 4]</td>
<td>[1, 1, 4] [7, 2, 4]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- *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 ⇒ irrelevant
Pseudo-maximal features $\Rightarrow$ unbounded $\hat{w}_j$

<table>
<thead>
<tr>
<th></th>
<th>Correct parse’s features</th>
<th>All other parses’ features</th>
</tr>
</thead>
<tbody>
<tr>
<td>sentence 1</td>
<td>$[1, 3, 2]$</td>
<td>$[2, 3, 4]$ $[3, 1, 1]$ $[2, 1, 1]$</td>
</tr>
<tr>
<td>sentence 2</td>
<td>$[2, 7, 4]$</td>
<td>$[3, 7, 2]$</td>
</tr>
<tr>
<td>sentence 3</td>
<td>$[2, 4, 4]$</td>
<td>$[1, 1, 1]$ $[1, 2, 4]$</td>
</tr>
</tbody>
</table>

- A **pseudo-maximal feature** always reaches its maximum value within a parse on the correct parse

- If $f_j$ is pseudo-maximal, $\hat{w}_j \to \infty$ (hard constraint)

- If $f_j$ is pseudo-minimal, $\hat{w}_j \to -\infty$ (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 $\hat{w}$ is finite and small
- In these experiments, the regularizer is a polynomial penalty term

$$\hat{w} = \arg\max_{w} \log PL_D(w) - c \sum_{j=1}^{m} |w_j|^p$$

($p = 2$ 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

⇒ 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
  
  \[
  \text{precision} \quad P_{\bar{y}}(y) = \frac{|E(y)|}{|E(y) \cap E(\bar{y})|}
  \]
  \[
  \text{recall} \quad R_{\bar{y}}(y) = \frac{|E(\bar{y})|}{|E(y) \cap E(\bar{y})|}
  \]
  \[
  \text{f score} \quad F_{\bar{y}}(y) = \frac{2}{\frac{1}{P_{\bar{y}}(y)} + \frac{1}{R_{\bar{y}}(y)}}
  \]

Edges

- (0, NP, 2)
- (2, VP, 3)
- (0, S, 3)

ROOT

- S
  - NP
    - DT
    - N
    - VP
      - VB
        - the
        - dog
        - barks
## Conditional and Joint ML PCFG evaluation

<table>
<thead>
<tr>
<th>Metric</th>
<th>MLE</th>
<th>MCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>log likelihood of training data</td>
<td>13857</td>
<td>13896</td>
</tr>
<tr>
<td>log <em>conditional</em> likelihood of training data</td>
<td>1833</td>
<td>1769</td>
</tr>
<tr>
<td>log <em>marginal</em> probability of training strings</td>
<td>12025</td>
<td>12127</td>
</tr>
<tr>
<td>Labelled precision of test data</td>
<td>0.815</td>
<td>0.817</td>
</tr>
<tr>
<td>Labelled recall of test data</td>
<td>0.789</td>
<td>0.794</td>
</tr>
</tbody>
</table>

- 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

$$\hat{y} = \arg\max_{y \in \mathcal{Y}(x)} S(x, y)$$

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 \(Y_c(x_i)\) of parse trees
- Best parse \(\tilde{y}_i = \arg\max_{y \in Y_c(x_i)} F_{y_i}(y)\), where \(F_{y_i}(y)\) measures parse accuracy
- \(w\) is chosen to maximize the regularized log pseudo-likelihood 
  \[ \sum_i \log P_w(\tilde{y}_i | x_i) + R(w) \]
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 $|\mathcal{V}(x)| = 36.1$
- 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; ≈ 4% error reduction

⇒ *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 tendency 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
• 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.
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.
The CoPar feature indicates the depth to which adjacent conjuncts are parallel.

There are 9 CoPar features.
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) 6 words
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: \(f\text{-score} = 0.904\) (20% error reduction)
Which kinds of features are best?

<table>
<thead>
<tr>
<th></th>
<th># of features</th>
<th>f-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Treebank trees</td>
<td>375,646</td>
<td>0.901</td>
</tr>
<tr>
<td>Correct parses</td>
<td>271,267</td>
<td>0.902</td>
</tr>
<tr>
<td>Incorrect parses</td>
<td>876,339</td>
<td>0.903</td>
</tr>
<tr>
<td>Correct &amp; incorrect parses</td>
<td>883,936</td>
<td>0.903</td>
</tr>
</tbody>
</table>

- 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

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

- Generative and discriminative parsers both identify the likely parse $y$ of a string $x$, i.e., estimate $P(y|x)$
- *Generative parsers also define language models*, estimate $P(x)$
- *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 discriminative 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 similar 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 $\beta_k$ with feature bin $k$
- Optimize feature weights $\alpha = (\alpha_1, \ldots, \alpha_m)$ on main training data $M$
- Optimize regularizer constants $\beta$ on held-out data $H$

$$L_D(\alpha) = \prod_{i=1}^{n} P_{\alpha}(y_i|x_i), \text{ where } D = ((y_1, x_1), \ldots, (y_n, x_n))$$

$$\hat{\alpha}(\beta) = \arg \max_{\alpha} \log L_M(\alpha) - \sum_{j=1}^{m} \beta_{b(j)} \alpha_j^2$$

$$\hat{\beta} = \arg \max_{\beta} \log L_H(\hat{\alpha}(\beta))$$
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

$$\hat{p} = \underset{p}{\text{argmax}} \ L_D(p), \text{ where}$$

$$L_D(p) = \prod_{i=1}^{n} P_p(x_i) = \underset{p}{\text{argmax}} \ \prod_{i=1}^{n} \sum_{y \in \mathcal{Y}(x_i)} P(y)$$
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

\[
\hat{p} = \arg\max_p L_D(p), \text{ where } \\
L_D(p) = \prod_{i=1}^{n} P_p(x_i|x_i)
\]