Dependency Parsing and Brown Clustering

Steven Abney

University of Michigan

2015 Mar 27
Dependency parsing
## Problem Definition

- Learning a dependency parser in a new language
  - Variant of **grammatical inference**
  - We know how to do supervised learning from a treebank

### Treebanks

- I know of TBs for 43 languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Language</th>
<th>Language</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>English, Middle</td>
<td>Hungarian</td>
<td>Romanian</td>
</tr>
<tr>
<td>Armenian, Ancient</td>
<td>English, Old</td>
<td>Icelandic</td>
<td>Russian</td>
</tr>
<tr>
<td>Basque</td>
<td>Estonian</td>
<td>Indonesian</td>
<td>Slavonic, Old Church</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>Finnish</td>
<td>Italian</td>
<td>Slovene</td>
</tr>
<tr>
<td>Catalan</td>
<td>French</td>
<td>Japanese</td>
<td>Spanish</td>
</tr>
<tr>
<td>Chinese</td>
<td>German</td>
<td>Karuk</td>
<td>Swedish</td>
</tr>
<tr>
<td>Czech</td>
<td>Gothic</td>
<td>Korean</td>
<td>Thai</td>
</tr>
<tr>
<td>Danish</td>
<td>Greek</td>
<td>Latin</td>
<td>Turkish</td>
</tr>
<tr>
<td>Dutch</td>
<td>Greek, Ancient</td>
<td>Polish</td>
<td>Ugaritic</td>
</tr>
<tr>
<td>English</td>
<td>Hebrew</td>
<td>Portuguese</td>
<td>Vietnamese</td>
</tr>
<tr>
<td>English, Early Modern</td>
<td>Hindi-Urdu</td>
<td>Portuguese, Medieval</td>
<td></td>
</tr>
</tbody>
</table>

- But there are 6800 languages (Ethnologue)
- Increasing interest from e.g. Google, DoD

### Transfer:

$L_1$ treebank $\rightarrow$ parser $\rightarrow$ $L_2$
Dependency Trees

govr: 2 0 4 2
role: subj root det obj
pos: Pron V DT N
w: this is an example
id: 1 2 3 4
### Dependency-parsing task

<table>
<thead>
<tr>
<th>Input</th>
<th>L Output</th>
<th>U Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. this</td>
<td>Pron</td>
<td>2 subj</td>
</tr>
<tr>
<td>2. is</td>
<td>V</td>
<td>0 root</td>
</tr>
<tr>
<td>3. an</td>
<td>DT</td>
<td>4 det</td>
</tr>
<tr>
<td>4. example</td>
<td>N</td>
<td>2 obj</td>
</tr>
</tbody>
</table>

- **Attachment Score = proportion correct**
- **LAS, UAS**
“Transition-based” dependency parsing (Nivre)

- Dependency parsers: transition-based, chart
- “Arc-eager” operations (my variant):
  - **LD**: $T$ is left dependent of $N$. Next must be Pop.
  - **RD**: $N$ is right dependent of $T$. Next must be Shift.
  - **Pop**: remove $T$ from stack.
  - **Shift**: move $N$ from buffer to stack.

- Transition $=$ (configuration $\Rightarrow$ configuration):

<table>
<thead>
<tr>
<th></th>
<th>$T$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td><em>is</em></td>
<td><em>an</em></td>
</tr>
<tr>
<td></td>
<td><em>is</em></td>
<td><em>this</em></td>
</tr>
<tr>
<td></td>
<td><em>this</em></td>
<td></td>
</tr>
</tbody>
</table>

$\overset{\text{LD, Pop}}{\Rightarrow}$

<table>
<thead>
<tr>
<th></th>
<th>$T$</th>
<th>$N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root</td>
<td><em>is</em></td>
<td><em>example</em></td>
</tr>
<tr>
<td></td>
<td><em>is</em></td>
<td><em>this</em></td>
</tr>
<tr>
<td></td>
<td><em>this</em></td>
<td><em>an</em></td>
</tr>
</tbody>
</table>

- Root (is) is this example
Oracle = Classifier

- Supervised training

<table>
<thead>
<tr>
<th>Instance (feature vector)</th>
<th>Label (next op)</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer 0 form = example</td>
<td>LD</td>
</tr>
<tr>
<td>buffer 0 lemma = example</td>
<td></td>
</tr>
<tr>
<td>buffer 0 cpos = N</td>
<td></td>
</tr>
<tr>
<td>buffer 0 fpos = NN</td>
<td></td>
</tr>
<tr>
<td>buffer 0 morph = sg</td>
<td></td>
</tr>
<tr>
<td>buffer 1 form = None</td>
<td></td>
</tr>
<tr>
<td>buffer 1 fpos = None</td>
<td></td>
</tr>
<tr>
<td>buffer 2 fpos = None</td>
<td></td>
</tr>
<tr>
<td>buffer 3 fpos = None</td>
<td></td>
</tr>
<tr>
<td>buffer 0 lc role = None</td>
<td></td>
</tr>
<tr>
<td>stack 0 form = an</td>
<td></td>
</tr>
<tr>
<td>stack 0 lemma = a</td>
<td></td>
</tr>
<tr>
<td>stack 0 cpos = D</td>
<td></td>
</tr>
<tr>
<td>stack 0 fpos = DT</td>
<td></td>
</tr>
<tr>
<td>stack 0 morph = None</td>
<td></td>
</tr>
<tr>
<td>stack 0 role = None</td>
<td></td>
</tr>
<tr>
<td>stack 1 fpos = VBZ</td>
<td></td>
</tr>
</tbody>
</table>

- Features are mostly features of words: form, lemma, cpos, fpos, morph
Eisner chart-parsing for dependencies (modified)

- **Edges and voids**
  - Voids “hide” completed material
- Bottom-up binary and unary combinations
  - Void + edge = right-spreading void
  - Edge + void = left-spreading void
  - Unary: void → edge
- We can use CKY algorithm ($n^3$); naive dep. chart parsing is $n^5$
**General pattern**

- One terminal void for each covering edge
  - Spread rightward first (left dependents)
  - Then spread leftward (right dependents)
  - Create covering edge
Edges & voids correspond to stack actions

\[ \text{a} \rightarrow \text{b} \]
\[ \text{RD+S} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \]
\[ \text{RD+S} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \]
\[ \text{S} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \]
\[ \text{LD} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
\[ \text{P} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
\[ \text{P} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
\[ \text{P} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
\[ \text{P} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
\[ \text{P} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
\[ \text{RD+S} \quad \text{a} \rightarrow \text{b} \rightarrow \text{c} \rightarrow \text{d} \rightarrow \text{e} \rightarrow \text{f} \]
McDonald et al 2005: probabilistic version

- Similar features to Nivre, but Eisner chart parsing
- Tree score:

\[ S = \sum_{k} w_k \sum_{i} f_k(g_i, d_i) = w \cdot c \]

- Features \( f_k(g, d) \) = features of \( g, d \), and words around them
- Positive-weighted features = good tree, negative = bad
- Probability = \( \exp(S) \)
- Learning = determining weights \( w_k \)
Supervised learning

- Classic approach is EM; compute-expensive but weak performance
- Alternative: error-driven update (perceptron, MIRA)
  - Initial weight vector \( w = 0 \)
  - Parse a sentence; get \( k \) best parses \( T_i \). Gold parse = \( G \).
  - For each \( T_i \neq G \): if \( S(T_i) \geq S(G) \) then
    \[
    w^{(t+1)} \leftarrow w^{(t)} + \eta [c(G) - c(T_i)]
    \]
- Perceptron has fixed step size \( \eta \), MIRA has adaptive step size
- Averaging makes it more robust:
  \[
  w \leftarrow \frac{1}{N}(w^{(1)}, \ldots, w^{(N)})
  \]
Brown clustering
Word clustering for dependency parsing

- Sparse data problem
  - Feature values are often words, lemmas
  - Most words are rare: many words in test never seen in training
  - Back off to groups of words: clusters

```
buffer 0 form = another
buffer 0 lemma = another
buffer 0 cpos = D
buffer 0 fpos = DQ
buffer 0 morph = sg
buffer 1 form = example
buffer 1 fpos = NN
buffer 2 fpos = None
buffer 3 fpos = None
buffer 0 lc role = None
stack 0 form = is
stack 0 lemma = be
stack 0 cpos = V
stack 0 fpos = VBZ
stack 0 morph = 3s
stack 0 role = root
stack 1 fpos = PP
```
Brown clustering
HMMs with classes

Model is assignment of words to classes
Each word belongs to unique class:

Classes are not hidden

p(text|model) = p(1|0) p(this|1) × p(3|1) p(is|3) × ...

Choose model to maximize likelihood $L = p(text|model)$
Simplifying the likelihood function

\[ L = p(1|0) p(this|1) \times p(3|1) p(is|3) \times \ldots \]

\[ \alpha = 0 \quad \beta = 1 \quad x = this \]

\[ \alpha = 1 \quad \beta = 3 \quad x = is \]

- Group factors by \( \alpha, \beta, x \)

\[ L = \prod_{\alpha, \beta, x} [p(\beta|\alpha) p(x|\beta)]^{ct(\alpha, \beta, x)} \]
Simplifying the likelihood function

Taking the log makes it more tractable

\[ \ell = \sum_{\alpha, \beta, x} \text{ct}(\alpha, \beta, x) \left[ \log p(\beta | \alpha) + \log p(x | \beta) \right] \]

\[ \ell / N = \sum_{\alpha, \beta, x} p(\alpha, \beta, x) \left[ \log \frac{p(\alpha, \beta)}{p(\alpha)} + \log \frac{p(\beta, x)}{p(\beta)} \right] \]

Move \( p(\beta) \) and distribute

\[ = \sum_{\alpha, \beta} p(\alpha, \beta) \log \frac{p(\alpha, \beta)}{p(\alpha) p(\beta)} + \sum_{\beta, x} p(\beta, x) \log p(\beta, x) \]
Simplifying the likelihood function

- Class is unique given word. Suppose \( x \)’s class is \( \alpha \).

\[
\begin{align*}
\text{if } \beta = \alpha: & \quad p(\beta, x) = p(x) \\
\text{if } \beta \neq \alpha: & \quad p(\beta, x) = 0
\end{align*}
\]

- So:

\[
\sum_{\beta} p(\beta, x) \log p(\beta, x) = p(x) \log p(x)
\]

- And:

\[
\ell / N = \sum_{\alpha, \beta} p(\alpha, \beta) \log \frac{p(\alpha, \beta)}{p(\alpha) p(\beta)} + \sum_{x} p(x) \log p(x)
\]

- Choose classes to maximize \( I(A; B) \)
How do we maximize $I(A; B)$?

- Start off with every word in its own cluster
- Consider merging two clusters $\alpha, \beta$. Compute the resulting value of $I(A; B)$.
- Choose the pair that gives the maximum new $I(A; B)$.
- Produces a **hierarchical clustering**

![Diagram](attachment:image.png)

- But how to do it efficiently?
Maximize graph weight = sum of edge weights

- Score = mutual information:

\[
I = \sum_{\alpha, \beta} \frac{p(\alpha, \beta) \log \left( \frac{p(\alpha, \beta)}{p_1(\alpha) p_2(\beta)} \right)}{q(\alpha, \beta)}
\]

- Think of it as a graph
  - Nodes are clusters
  - Edges connect clusters that co-occur: \( p(\alpha, \beta) > 0 \)
  - **Edge weight** is \( q(\alpha, \beta) \)
  - These are directed edges
Undirected graph

- Combine pairs of directed edges to make one undirected edge

\[ Q(\alpha, \beta) = \begin{cases} 
q(\alpha, \beta) + q(\beta, \alpha) & \text{if } \alpha \neq \beta \\
q(\alpha, \alpha) & \text{if } \alpha = \beta 
\end{cases} \]

- Now:

\[ I = \sum_{\alpha \leq \beta} Q(\alpha, \beta) \]
An example

see spot run EOS
run spot run EOS
run run EOS
see jane EOS
jane run EOS
run jane EOS

I = 0.602
Algorithm

- The algorithm:
  - Build graph
  - For each pair of nodes \((\alpha, \beta)\), compute the cost (loss) of merging \(\alpha + \beta\)
  - Choose the minimum-cost pair and merge them
  - Update \(p, Q\), etc. and repeat

- Loss \(L(\alpha, \beta)\)
  - Merging \(\alpha + \beta\) cannot increase \(I\). \(L(\alpha, \beta) \geq 0\), small is good.
  - What is the effect of doing a merger?
  - How do we update loss matrix for other pairs, without recomputing from scratch?
Loss

Node weight

\[ s(\alpha) = \sum_{\nu} Q(\nu, \alpha) \]

Merged-node weight

\[ S(\alpha, \beta) = \sum_{\nu} Q(\nu, \alpha + \beta) \]

\[ \Delta = -s(\alpha) - s(\beta) + Q(\alpha, \beta) + S(\alpha, \beta) \]

(double-counted)
\( Q(\nu, \alpha + \beta) \)

- \( Q(\nu, \alpha + \beta) \) can be computed without actually creating a node:

\[
\begin{align*}
Q(\nu, \alpha + \beta) &= q(\nu, \alpha + \beta) + q(\alpha + \beta, \nu) \\
q(\nu, \alpha + \beta) &= p(\nu, \alpha + \beta) \log \frac{p(\nu, \alpha + \beta)}{p_1(\nu) p_2(\alpha + \beta)} \\
p(\nu, \alpha + \beta) &= p(\nu, \alpha) + p(\nu, \beta)
\end{align*}
\]
\[ \Delta \leq 0. \text{ Loss } = -\Delta: \]

\[ L(\alpha, \beta) = s(\alpha) + s(\beta) - Q(\alpha, \beta) - S(\alpha, \beta) \]

- Maintain array \( s \) and matrix \( S \), compute \( L \) on the fly.
- Updating
  - Suppose we merge \( \lambda + \mu \Rightarrow \tau \)
  - No effect on \( Q(\alpha, \beta) \)
  - What is the effect on \( s(\alpha) \) and \( S(\alpha, \beta) \)?
\[ \Delta s(\alpha) = Q(\tau, \alpha) - Q(\lambda, \alpha) - Q(\mu, \alpha) \]

\[ \Delta S(\alpha, \beta) = Q(\tau, \alpha + \beta) - Q(\lambda, \alpha + \beta) - Q(\mu, \alpha + \beta) \]
Algorithm, final form

- Create graph
  - Compute $Q(\alpha, \beta)$ for edges, $s(\alpha)$ for nodes
  - **Co-edge** $(\alpha, \beta)$ iff $\alpha$ and $\beta$ share a neighbor
  - Compute $S(\alpha, \beta)$ for each co-edge

- Main loop
  - Among co-edges, maximize $s(\lambda) + s(\mu) - Q(\lambda, \mu) - S(\lambda, \mu)$
  - Pre-update:
    \[
    s(\alpha) = s(\alpha) - Q(\lambda, \alpha) - Q(\mu, \alpha) \\
    S(\alpha, \beta) = S(\alpha, \beta) - Q(\lambda, \alpha + \beta) - Q(\mu, \alpha + \beta)
    \]
  - Delete nodes $\lambda$ and $\mu$, add node $\tau$. Compute $Q(\nu, \tau)$ and $s(\tau)$.
  - Post-update:
    \[
    s(\alpha) = s(\alpha) + Q(\tau, \alpha) \\
    S(\alpha, \beta) = S(\alpha, \beta) + Q(\tau, \alpha + \beta)
    \]
Attribute-value clustering
Returning to tree scoring in dependency parsing

- **Tree score**
  - Edge candidates \((g_i, d_i)\). Tree = subset \(\forall\) word has 1 govr
  - Edge has set of features: \(\{k \mid f_k(g_i, d_i) = 1\}\).
  - \(v_i\) is a bit vector whose \(k\)-th bit is \(f_k(g_i, d_i)\).
  - Tree score:

\[
S = \sum_k w_k \sum_i f_k(g_i, d_i) = \sum_i w \cdot v_i
\]

- **Candidate-edge feature set:**
  
  order:dep-govr
d-form:dog
d-lemma:dog
d-cpos:N
g-form:barks
g-lemma:bark
g-cpos:V
  :
**Attribute-value clustering**

- Usual approach: use plain text to get clusters
- Alternative
  - Build clusters that are specific to parsing
  - Let’s include higher-order features, e.g.
    
    \[
    \text{role = subj} \quad \text{g-lemma = bark} \quad \text{d-lemma = dog} \quad \Rightarrow \quad \text{subj(bark,dog)}
    \]

  - Which we view as
    
    \[
    \text{attribute: subj(bark,\_)}
    \]
    
    \[
    \text{value: dog}
    \]

- Goal: **simultaneous clustering of attributes and values**
- Generally applicable to instances represented as sets of AV pairs
Different generative model

- Generating a single data point:

\[
p(x, y) = p(\alpha, \beta) p(x|\alpha) p(y|\beta)
\]

- Log likelihood: group by \(\alpha, \beta, x, y\):

\[
\ell/N = \sum_{\alpha,\beta,x,y} p(\alpha, \beta, x, y) \log [ p(\alpha, \beta) p(x|\alpha) p(y|\beta) ]
\]

\[
= \sum_{\alpha,\beta,x,y} p(\alpha, \beta, x, y) \left[ \log \frac{p(\alpha, \beta)}{p(\alpha) p(\beta)} + \log p(x, \alpha) + \log p(y, \beta) \right]
\]

\[
= \sum_{\alpha,\beta} p(\alpha, \beta) \log \frac{p(\alpha, \beta)}{p(\alpha) p(\beta)} + \sum_x p(x) \log p(x) + \sum_y p(y) \log p(y)
\]

\[
\underbrace{I(A;B)}_{\text{Same bottom line: seek classes that maximize } I(A;B)} + \underbrace{-H(X)}_{\text{Once we have the graph, the algorithm is the same}} + \underbrace{-H(Y)}_{\text{Once we have the graph, the algorithm is the same}}
\]
Bigraph

see spot run EOS
run spot run EOS
run run EOS
see jane EOS
jane run EOS
run jane EOS

- No loops
- No edges between atts or between values
Context distributions

- Define **context distribution** \( p_\alpha(\gamma) = \frac{p(\gamma, \alpha)}{p(\alpha)} \)

- Distribution over contexts of \( \alpha \).

- Since \( p(\alpha, \gamma) = 0 \) in the bigraph case, we have:

\[
Q(\alpha, \gamma) = q(\gamma, \alpha)
\]

- Hence:

\[
s(\alpha) = \sum_{\gamma} p(\gamma, \alpha) \log \frac{p(\gamma, \alpha)}{p(\gamma) p(\alpha)}
= \sum_{\gamma} p(\gamma, \alpha) \log \frac{p_\alpha(\gamma)}{p(\gamma)}
\]
Context distributions

- Can also be defined for \( s(\alpha + \beta) \):

\[
p_{\alpha+\beta}(\gamma) = \frac{p(\gamma, \alpha + \beta)}{p(\alpha + \beta)}
\]

- Hence:

\[
s(\alpha + \beta) = \sum_{\gamma} p(\gamma, \alpha + \beta) \log \frac{p(\gamma, \alpha + \beta)}{p(\gamma) p(\alpha + \beta)}
\]

\[
= \sum_{\gamma} p(\gamma, \alpha) \log \frac{p_{\alpha+\beta}(\gamma)}{p(\gamma)} + \sum_{\gamma} p(\gamma, \beta) \log \frac{p_{\alpha+\beta}(\gamma)}{p(\gamma)}
\]
**Loss**

- Since the graph is now a bigraph, $L$ simplifies (slightly):

\[
L(\alpha, \beta) = s(\alpha) + s(\beta) - \underbrace{Q(\alpha, \beta) - s(\alpha + \beta)}_{= 0}
\]

- Using our previous results:

\[
\begin{align*}
\sum_g p(\gamma, \alpha) \log \frac{p(\alpha)(\gamma)}{p(\gamma)} &+ \sum_g p(\gamma, \beta) \log \frac{p(\beta)(\gamma)}{p(\gamma)} \\
- \sum_g p(\gamma, \alpha) \log \frac{p(\alpha + \beta)(\gamma)}{p(\gamma)} &+ \sum_g p(\gamma, \beta) \log \frac{p(\alpha + \beta)(\gamma)}{p(\gamma)} \\
= &\quad L(\alpha, \beta) \quad \sum_g p(\gamma, \alpha) \log \frac{p(\alpha)(\gamma)}{p(\alpha + \beta)(\gamma)} + \sum_g p(\gamma, \beta) \log \frac{p(\beta)(\gamma)}{p(\alpha + \beta)(\gamma)}
\end{align*}
\]
Finally:

\[
\frac{L(\alpha, \beta)}{p(\alpha + \beta)} = \frac{p(\alpha)}{p(\alpha + \beta)} D(p_\alpha \| p_{\alpha + \beta}) + \frac{p(\beta)}{p(\alpha + \beta)} D(p_\beta \| p_{\alpha + \beta})
\]

- This is the Jensen-Shannon divergence of \( p_\alpha \) from \( p_\beta \)
- Minimizing loss = merging the pair of clusters whose context distributions are most similar