Dependency Parsing and Brown Clustering

Steven Abney

University of Michigan

2015 Mar 27

Dependency parsing

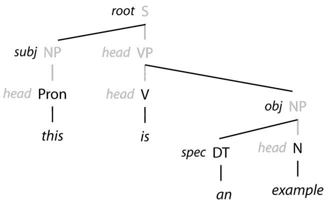
Problem Definition

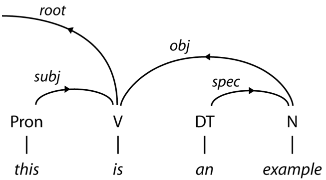
- Learning a dependency parser in a new language
  - Variant of **grammatical inference**
  - We know how to do supervised learning from a treebank
- Treebanks
  - I know of TBs for 43 languages:

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

  - 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

	Input		**L Output**	**U Output**
[1]	this	Pron	2 subj	2
[2]	is	V	0 root	0
[3]	an	DT	4 det	4
[4]	example	N	2 obj	2

- 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):

		T	N
Root	*is*	*an*	*example*
(is)	*this*		

$\xRightarrow{\text{LD, Pop}}$

	T	N
Root	*is*	*example*
(is)	*this*	*an*

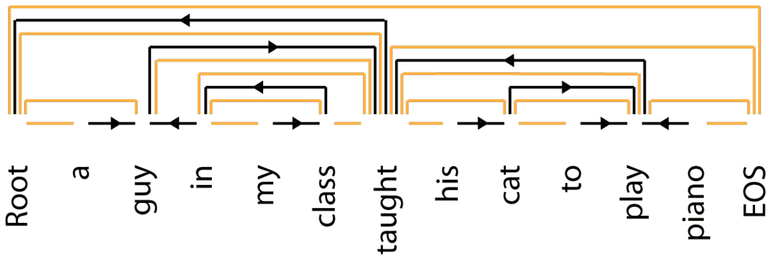
Oracle = Classifier

- Supervised training

Instance (feature vector)	**Label** (next op)
buffer 0 form = example buffer 0 lemma = example buffer 0 cpos = N buffer 0 fpos = NN buffer 0 morph = sg buffer 1 form = None buffer 1 fpos = None buffer 2 fpos = None buffer 3 fpos = None buffer 0 lc role = None stack 0 form = an stack 0 lemma = a stack 0 cpos = D stack 0 fpos = DT stack 0 morph = None stack 0 role = None stack 1 fpos = VBZ	LD

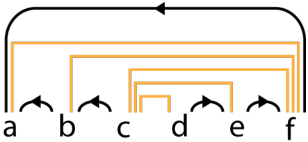
- Features are mostly features of words: form, lemma, cpos, fpos, morph

Eisner chart-parsing for dependencies (modified)



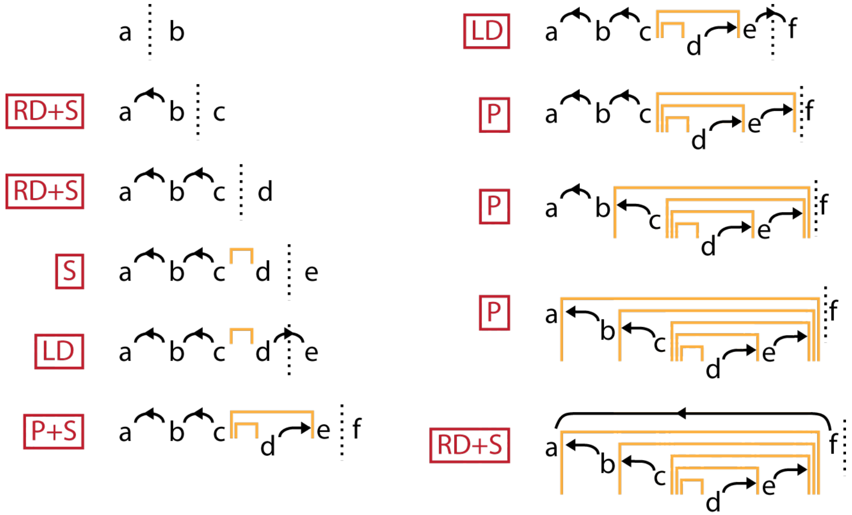
- **Edges** and **voids**
  - Voids “hide” completed material
- Bottom-up binary and unary combinations
  - Void + edge = right-spreading void
  - Edge + void = left-spreading void
  - Unary: void → edge
- We can use CKY algorithm (n^3); naive dep. chart parsing is n^5

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



McDonald et al 2005: probabilistic version

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

$$S = \sum_k w_k \underbrace{\sum_i \overbrace{f_k(g_i, d_i)}^{0,1}}_{c_k} = \mathbf{w} \cdot \mathbf{c}$$

  - 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 $\mathbf{w} = \mathbf{0}$
  - 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

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta[\mathbf{c}(G) - \mathbf{c}(T_i)]$$

- Perceptron has fixed step size η, MIRA has adaptive step size
- Averaging makes it more robust:

$$\mathbf{w} \leftarrow \frac{1}{N}(\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(N)})$$

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**

$$\begin{array}{ll} \text{EOS} & \to 0 \\ \textit{an} & \to 1 \\ \textit{example} & \to 2 \\ \textit{is} & \to 3 \\ \textit{this} & \to 1 \end{array}$$

$$\begin{array}{ccccc} 1 & 3 & 1 & 2 & 0 \\ \textit{this} & \textit{is} & \textit{an} & \textit{example} & \text{EOS} \end{array}$$

$$p(\textit{text}|\textit{model}) = p(1|0)\,p(\textit{this}|1) \;\times\; p(3|1)\,p(\textit{is}|3) \;\times\; \ldots$$

- Choose model to maximize **likelihood** $L = p(\textit{text}|\textit{model})$

Simplifying the likelihood function

$$L = \underbrace{p(1|0)\,p(\mathit{this}|1)}_{\substack{\alpha=0\\ \beta=1\\ x=\mathit{this}}} \times \underbrace{p(3|1)\,p(\mathit{is}|3)}_{\substack{\alpha=1\\ \beta=3\\ x=\mathit{is}}} \times \ldots$$

- Group factors by α, β, x

$$L = \prod_{\alpha,\beta,x} [\, p(\beta|\alpha)\, p(x|\beta)\,]^{\mathrm{ct}(\alpha,\beta,x)}$$

Simplifying the likelihood function

- Taking the log makes it more tractable

$$\ell = \sum_{\alpha,\beta,x} \mathrm{ct}(\alpha,\beta,x)[\log p(\beta|\alpha) + \log p(x|\beta)]$$

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

- 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 α.

$$\begin{aligned} \text{if } \beta = \alpha: \quad & p(\beta, x) = p(x) \\ \text{if } \beta \neq \alpha: \quad & p(\beta, x) = 0 \end{aligned}$$

- So:

$$\sum_{\beta} p(\beta, x) \log p(\beta, x) \;=\; p(x) \log p(x)$$

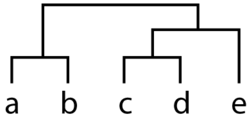
- And:

$$\ell/N \;=\; \underbrace{\sum_{\alpha,\beta} p(\alpha, \beta) \log \frac{p(\alpha, \beta)}{p(\alpha)\, p(\beta)}}_{I(A;B)} + \underbrace{\sum_{x} p(x) \log p(x)}_{H(X)}$$

- 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 α, β. Compute the resulting value of $I(A;B)$.
- Choose the pair that gives the maximum new $I(A;B)$.
- Produces a **hierarchical clustering**



- But how to do it efficiently?

Maximize graph weight = sum of edge weights

- Score = mutual information:

$$I = \sum_{\alpha,\beta} \underbrace{p(\alpha,\beta) \log \frac{p(\alpha,\beta)}{p_1(\alpha)\, p_2(\beta)}}_{q(\alpha,\beta)}$$

- 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

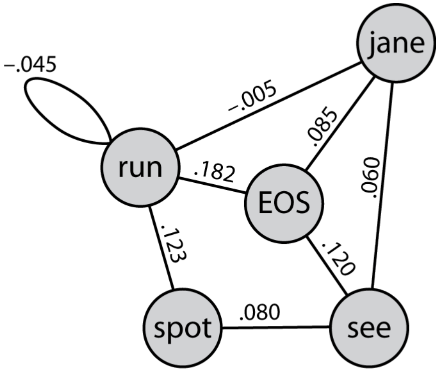
$$Q(\alpha, \beta) = \begin{cases} q(\alpha, \beta) + q(\beta, \alpha) & \text{if } \alpha \neq \beta \\ q(\alpha, \alpha) & \text{if } \alpha = \beta \end{cases}$$

- 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

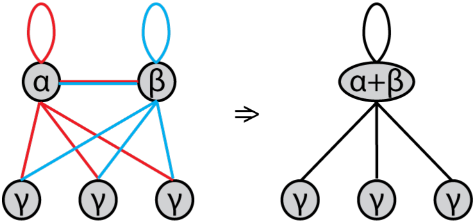


$$I = 0.602$$

Algorithm

- The algorithm:
  - Build graph
  - For each pair of nodes (α, β), 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 $S(\alpha, \beta) = \sum_{\nu} Q(\nu, \alpha + \beta)$

$$\Delta = -s(\alpha) - s(\beta) + \underbrace{Q(\alpha, \beta)}_{\text{dbl-counted}} + S(\alpha, \beta)$$

$Q(\nu, \alpha + \beta)$

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

$$
\begin{aligned}
Q(\nu, \alpha + \beta) &= q(\nu, \alpha + \beta) + q(\alpha + \beta, \nu) \\
q(\nu, \alpha + \beta) &= p(\nu, \alpha + \beta) \log \frac{p(\nu, \alpha + \beta)}{p_1(\nu)\, p_2(\alpha + \beta)} \\
p(\nu, \alpha + \beta) &= p(\nu, \alpha) + p(\nu, \beta)
\end{aligned}
$$

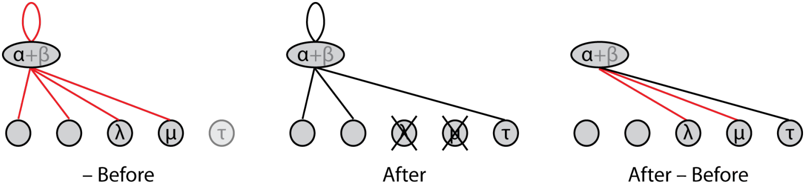
Loss

- $\Delta \leq 0$. 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 $S(\alpha, \beta)$?

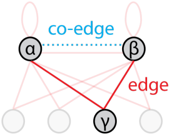
Δs and ΔS



$$
\begin{aligned}
\Delta s(\alpha) &= Q(\tau,\alpha) - Q(\lambda,\alpha) - Q(\mu,\alpha) \\
\Delta S(\alpha,\beta) &= Q(\tau,\alpha+\beta) - Q(\lambda,\alpha+\beta) - Q(\mu,\alpha+\beta)
\end{aligned}
$$

Algorithm, final form

- Create graph
  - Compute $Q(\alpha, \beta)$ for edges, $s(\alpha)$ for nodes
  - **Co-edge** (α, β) iff α and β 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:

$$\begin{aligned} s(\alpha) &= s(\alpha) - Q(\lambda, \alpha) - Q(\mu, \alpha) \\ S(\alpha, \beta) &= S(\alpha, \beta) - Q(\lambda, \alpha + \beta) - Q(\mu, \alpha + \beta) \end{aligned}$$

  - Delete nodes λ and μ, add node τ. Compute $Q(\nu, \tau)$ and $s(\tau)$.
  - Post-update:

$$\begin{aligned} s(\alpha) &= s(\alpha) + Q(\tau, \alpha) \\ S(\alpha, \beta) &= S(\alpha, \beta) + Q(\tau, \alpha + \beta) \end{aligned}$$

Attribute-value clustering

Returning to tree scoring in dependency parsing

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

$$
\begin{aligned}
S &= \sum_k w_k \sum_i f_k(g_i, d_i) \\
&= \sum_i \mathbf{w} \cdot \mathbf{v}_i
\end{aligned}
$$

- 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.

    role = subj
    g-lemma = bark
    d-lemma = dog
    $\Rightarrow$ subj(bark,dog)

  - Which we view as

    attribute: subj(bark,__)
    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 α, β, x, y:

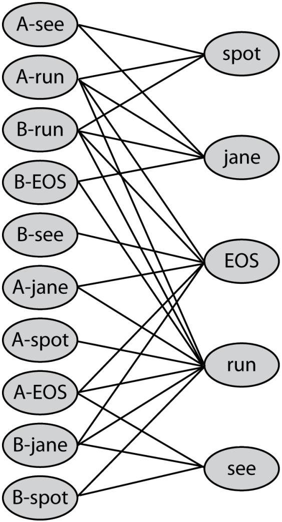
$$\begin{aligned}
\ell/N &= \sum_{\alpha,\beta,x,y} p(\alpha,\beta,x,y) \log \left[\, p(\alpha,\beta)\, p(x|\alpha)\, p(y|\beta) \,\right] \\
&= \sum_{\alpha,\beta,x,y} p(\alpha,\beta,x,y) \left[\log \frac{p(\alpha,\beta)}{p(\alpha)\, p(\beta)} + \log p(x,\alpha) + \log p(y,\beta)\right] \\
&= \underbrace{\sum_{\alpha,\beta} p(\alpha,\beta) \log \frac{p(\alpha,\beta)}{p(\alpha)\, p(\beta)}}_{I(A;B)} + \underbrace{\sum_{x} p(x) \log p(x)}_{-H(X)} + \underbrace{\sum_{y} p(y) \log p(y)}_{-H(Y)}
\end{aligned}$$

- Same bottom line: seek classes that maximize $I(A; B)$
  - Once we have the graph, the algorithm is the same

Bigraph

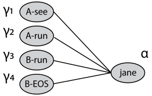
see spot run EOS
run spot run EOS
run run EOS
see jane EOS
jane run EOS
run jane EOS

- 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 α.



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

$$Q(\alpha,\gamma) = q(\gamma,\alpha)$$

- Hence:

$$\begin{aligned} s(\alpha) &= \sum_\gamma p(\gamma,\alpha) \log \frac{p(\gamma,\alpha)}{p(\gamma)\, p(\alpha)} \\ &= \sum_\gamma p(\gamma,\alpha) \log \frac{p_\alpha(\gamma)}{p(\gamma)} \end{aligned}$$

Context distributions

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

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

- Hence:

$$\begin{aligned} s(\alpha+\beta) &= \sum_{\gamma} p(\gamma, \alpha+\beta) \log \frac{p(\gamma, \alpha+\beta)}{p(\gamma)\, p(\alpha+\beta)} \\ &= \sum_{\gamma} p(\gamma, \alpha) \log \frac{p_{\alpha+\beta}(\gamma)}{p(\gamma)} + \sum_{\gamma} p(\gamma, \beta) \log \frac{p_{\alpha+\beta}(\gamma)}{p(\gamma)} \end{aligned}$$

Loss

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

$$L(\alpha,\beta) = s(\alpha) + s(\beta) - \overbrace{Q(\alpha,\beta)}^{=0} - s(\alpha+\beta)$$

- Using our previous results:

$$\begin{array}{rr|lcl}
 & s(\alpha)+s(\beta) & \sum_g p(\gamma,\alpha)\log\frac{p_\alpha(\gamma)}{p(\gamma)} & + & \sum_g p(\gamma,\beta)\log\frac{p_\beta(\gamma)}{p(\gamma)} \\
- & s(\alpha+\beta) & \sum_g p(\gamma,\alpha)\log\frac{p_{\alpha+\beta}(\gamma)}{p(\gamma)} & + & \sum_g p(\gamma,\beta)\log\frac{p_{\alpha+\beta}(\gamma)}{p(\gamma)} \\
\hline
= & L(\alpha,\beta) & \sum_g p(\gamma,\alpha)\log\frac{p_\alpha(\gamma)}{p_{\alpha+\beta}(\gamma)} & + & \sum_g p(\gamma,\beta)\log\frac{p_\beta(\gamma)}{p_{\alpha+\beta}(\gamma)}
\end{array}$$

Punch line

- 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_α from p_β
- Minimizing loss = merging the pair of clusters whose context distributions are most similar