Advanced PCFG Models

Sara Stymne

Syntactic Parsing
2018-02-06

Slides partly from Joakim Nivre

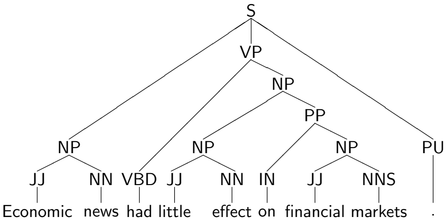
1. Problems with Treebank PCFGs
2. Parent Annotation
3. Lexicalization
4. Latent Variables
5. Other Parsing Frameworks
6. Some Hints for Assignments

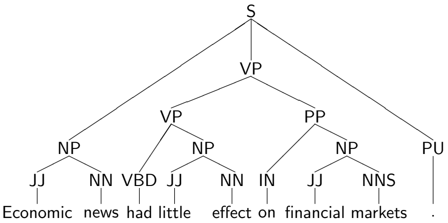
Lack of Sensitivity to Structural Context

Tree Context	NP PP	DT NN	PRP
Anywhere	11%	9%	6%
NP under S	9%	9%	21%
NP under VP	23%	7%	4%

Lack of Sensitivity to Lexical Information

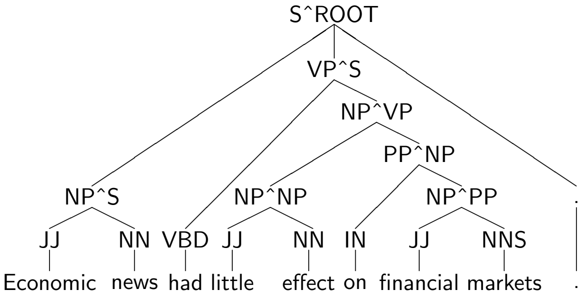
S	→	NP VP PU	1.00
VP	→	VP PP	0.33
VP	→	VBD NP	0.67
NP	→	NP PP	0.14
NP	→	JJ NN	0.57
NP	→	JJ NNS	0.29
PP	→	IN NP	1.00
PU	→	.	1.00
JJ	→	Economic	0.33
JJ	→	little	0.33
JJ	→	financial	0.33
NN	→	news	0.50
NN	→	effect	0.50
NNS	→	markets	1.00
VBD	→	had	1.00
IN	→	on	1.00





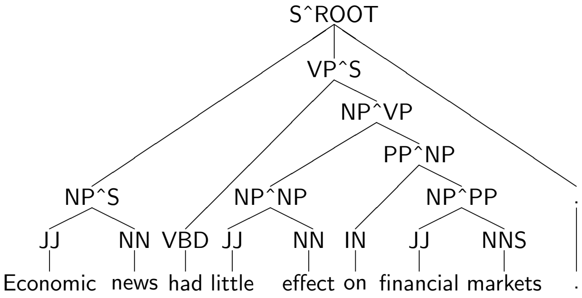
Parent Annotation

Replace nonterminal A with A^*B* when A is child of B.



Parent Annotation

Replace nonterminal A with A^*B* when A is child of B.



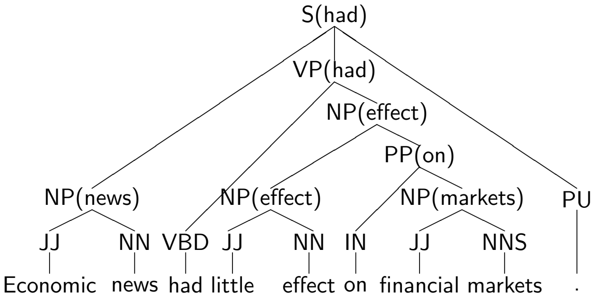
Described in the first seminar article

Lexicalization

Nonterminals: $N_{\text{lex}} = \{A(a) \mid A \in N, a \in \Sigma\}$

Rules: $A(a) \rightarrow \ldots B(a) \ldots$

$A(a) \rightarrow a$



Smoothing of the Lexicalized PCFG

$$
\begin{aligned}
q &= Q(\mathsf{A}(\mathsf{a}) \rightarrow \mathsf{B}(\mathsf{b})\ \mathsf{C}(\mathsf{a})) \\
&= P(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C}, \mathsf{b} \,|\, \mathsf{A}, \mathsf{a}) \\
&= P(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C} \,|\, \mathsf{A}, \mathsf{a}) \cdot P(\mathsf{b} \,|\, \mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C}, \mathsf{a})
\end{aligned}
$$

$$
\begin{aligned}
q_1 &= P(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C} \,|\, \mathsf{A}, \mathsf{a}) \\
&\approx \lambda \frac{\textsc{count}(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C}, \mathsf{a})}{\textsc{count}(\mathsf{A}, \mathsf{a})} + (1-\lambda) \frac{\textsc{count}(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C})}{\textsc{count}(\mathsf{A})}
\end{aligned}
$$

$$
\begin{aligned}
q_2 &= P(\mathsf{b} \,|\, \mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C}, \mathsf{a}) \\
&\approx \lambda \frac{\textsc{count}(\mathsf{b}, \mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C}, \mathsf{a})}{\textsc{count}(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C}, \mathsf{a})} + (1-\lambda) \frac{\textsc{count}(\mathsf{b}, \mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C})}{\textsc{count}(\mathsf{A} \rightarrow_2 \mathsf{B}\ \mathsf{C})}
\end{aligned}
$$

Non-lexicalized CKY Parsing

PARSE(G, x)
for j from 1 to n do
  for all $A : A \rightarrow x_j \in R$
    $\mathcal{C}[j-1, j, A] := Q(A \rightarrow x_j)$
for j from 2 to n do
  for i from $j-2$ downto 0 do
    for k from $i+1$ to $j-1$ do
      for all $A \rightarrow BC \in R$ and $\mathcal{C}[i,k,B] > 0$ and $\mathcal{C}[k,j,C] > 0$
        if $(\mathcal{C}[i,j,A] < Q(A \rightarrow B\ C) \cdot \mathcal{C}[i,k,B] \cdot \mathcal{C}[k,j,C])$ then
          $\mathcal{C}[i,j,A] := Q(A \rightarrow B\ C) \cdot \mathcal{C}[i,k,B] \cdot \mathcal{C}[k,j,C]$
          $\mathcal{B}[i,j,A] := (k, B, C)$
return BUILD-TREE($\mathcal{B}[0, n, S]$)

Lexicalized CKY Parsing

PARSE(G, x)
for j from 1 to n do
  for all $A : A(x_j) \rightarrow x_j \in R$
    $\mathcal{C}[j-1, j, j, A] := Q(A(x_j) \rightarrow x_j)$
for j from 2 to n do
  for i from $j-2$ downto 0 do
    for k from $i+1$ to $j-1$ do
      for h from $i+1$ to k do
        for m from $k+1$ to j do
          for all $A : A(x_h) \rightarrow B(x_h)C(x_m) \in R$ and $\mathcal{C}[i, k, h, B] > 0$ and $\mathcal{C}[k, j, m, C] > 0$
            if $(\mathcal{C}[i, j, h, A] < Q(A(x_h) \rightarrow B(x_h)C(x_m)) \cdot \mathcal{C}[i, k, h, B] \cdot \mathcal{C}[k, j, m, C])$ then
              $\mathcal{C}[i, j, h, A] := Q(A(x_h) \rightarrow B(x_h)C(x_m)) \cdot \mathcal{C}[i, k, h, B] \cdot \mathcal{C}[k, j, m, C]$
              $\mathcal{B}[i, j, h, A] := (k, B, h, C, m)$
      for h from $k+1$ to j do
        for m from $i+1$ to k do
          for all $A : A(x_h) \rightarrow B(x_m)C(x_h) \in R$ and $\mathcal{C}[i, k, m, B] > 0$ and $\mathcal{C}[k, j, h, C] > 0$
            if $(\mathcal{C}[i, j, m, A] < Q(A(x_h) \rightarrow B(x_m)C(x_h)) \cdot \mathcal{C}[i, k, m, B] \cdot \mathcal{C}[k, j, h, C])$ then
              $\mathcal{C}[i, j, h, A] := Q(A(x_h) \rightarrow B(x_m)C(x_h)) \cdot \mathcal{C}[i, k, m, B] \cdot \mathcal{C}[k, j, h, C]$
              $\mathcal{B}[i, j, h, A] := (k, B, m, C, h)$
return $\max_h \mathcal{C}[0, n, h, S]$, BUILD-TREE($\mathcal{B}[0, n, \text{argmax}_h\, \mathcal{C}[0, n, h, S], S]$)

Complexity

- Two extra loops in the algorithm, for the head of left and right trees
- Complexity is thus $O(n^5)$ instead of $O(n^3)$
- Too slow for many practical applications
- Pruning techniques often used
  - Means that we do not necessarily find the best tree, even given our model

Latent Variables

- Extract treebank PCFG
- Repeat k times:
  1. Split every nonterminal A into A_1 and A_2 (and duplicate rules)
  2. Train a new PCFG with the split nonterminals using EM
  3. Merge back splits that do not increase likelihood

Some Famous Parsers

	Par	Lex	Mark	Lat
Collins	+	+	+	−
Charniak	+	+	+	−
Stanford	+	−	+	−
Berkeley	+	−	+	+

Other Parsing Frameworks

- Shift-reduce parsing (transition-based)
  - Does not need a chart
  - Greedy
  - Linear time complexity
- Neural networks in parsing
  - Can reduce independence assumptions
  - Typically gives better results
  - Example: Recurrent neural network grammars (RNNG)

CNF conversion 1

Probably easiest to solve by a recursive function

XXX represent either a list or string

```
List contains two strings
e.g.:  ["IN", "as"]
    Do nothing

List contains two items, string and list
e.g. : ["NP" ["PRP", XXX]]
    Contract the two grammar symbols, and remove one list
    Apply cnf-method to the resulting tree
      cnf(["NP+PRP", XXX])

List contains three symbols, string, list, list
e.g. ["NP", ["DT", XXX], ["NNS", XXX]]
    Keep as it is, and apply cnf-method to the two lists
       cnf(["DT", XXX]), cnf(["NNS", XXX])
```

CNF conversion 2

Probably easiest to solve by a recursive function
XXX represent either a list or string

```
List contains more than three symbols, string, list, list, list, ...
e.g. ["S", ["NP", XXX], ["VP", XXX], [".", XXX]]
    Keep first two items, create an extra list with new label to which
      you give a "new" label. Apply cnf to the resulting tree

      cnf(["S", ["NP", XXX],
           ["new-name",  ["VP", XXX], [".", XXX]]])

   # think about the naming and markovization!


List contains something else:
    Something has gone wrong!
```

Backtrace

```
Assume that backpointers are lists:\\
(lh, rh1, rh2, min, mid, max)  if binary rule\\
(pos, word) if preterminal rule \\
```

```
BACKTRACE(bp, bpchart)
   if not defined(bp) then
       return NONE
       # should not happen!
   else if length(bp) == 2 then
       return makeList(pos, word)
   else   # length(bp) == 6
       return makeList(lh, BACKTRACE(bpchart(min, mid, rh1),bpchart)
                        BACKTRACE(bpchart(mid, max, rh2),bpchart))
```