Implementation

December 30, 2013

A dependency parse comprises of a (head word, modifier word) pair. The modifier is dependent on the head word. Given a sentence, there are several such parses possible. The crux of implementation is building a hypergraph using all these possible parses.

1 Hypergraph

A hypergraph is a pair H = (V, E), where V is the set of vertices, E is the set of hyperedges. A weighted hypergraph is associated with the set of weights W. Each hyperedge e ϵ E of a weighted graph is a triple e = (T(e), h(e), f(e)), where h(e) ϵ V is its head vertex and T(e) ϵ V^* is an ordered list of tail vertices. f(e) is a weight function from W|T(e)| to W.

The words in a sentence combine in different ways to form sub parses. The subparses form the vertices of the hypergraph. One or more of the sub parses that combine to form another sub parse form the tail nodes of the edge. The resultant subparse is the head node of the edge. The weights of the edges are determined by an algorithm presented later. The best parse can be computed as the maximum weighted path encompassing all the words leading to the root. The probability sub parse SP formed at vertex V_{SP}, for a given sentence is given by: P(V_{SP}) = $\sum_{\forall e \epsilon H(e)=V_{SP}}$ f(e)

The probability of this subparse for the entire grammar is obtained by summing the P(V_{SP}) for all the sentences in the grammar.

2 Eisner's parsing algorithm

Eisner's parsing algorithm is an efficient way to come up with all the possible parses. The computational complexity of Regular CKY parsing for lexicalized grammar is $\boldsymbol{O(n^5)}$ while that for Eisner's algorithm is $\boldsymbol{O(n^3)}$. Here, we discuss about a modified version of the Eisner's algorithm.

The 3 basic compenents of this algorithm are: incomplete constituents, complete constituents, complete stopped constituent. The constituent has 3 attributes namely index of the head word, index of the modifier word and the direction. A complete constituent with head word h can can choose to take another word as its dependent with a certain probability called the continue

probability. The head word of a complete constituent stops taking further arguments in that direction with a probability called the stop probability, leading to the formation of a complete stopped constituent. A word h takes another word m as its argument only m stopped taking arguments on both sides. Thus a complete constituent with head word h combines with a complete stopped constituent m to create an incomplete constituent. An incomplete constituent with head word h and modifier m then combines with a complete stopped constituent whose head word is m to form a complete constituent. This continues until two complete stopped constituents are formed in either directions together spanning the entire sentence. A complete stopped constituent is a type of complete constituent. Every word by default is a complete constituent.

The psuedocode of the algorithm is given in Figure 1. The dynamic programming table C[s][t][d][c] stores the sum of probabilities of the subtrees that can be formed from s to t in the direction d. c indicates the type of the constituent. if c = 0, it is an incomplete constituent. c = 1, it is a complete constituent. c = 2 indicates that it is a complete stop constituent.

Algorithm 1 Eisner's parsing algorithm

Initialization:
for s = 0 to n **do**
Form complete constituent of size 1
C[s][s][→][1] = C[s][s][←][1] = 0.0 Form complete constituent stop of size 1
C[s][s][→][2] = C[s][s][←][2] = 0.0
end for
for k = 1 to n + 1 **do**
for s = 0 to n **do**
t=s+k
if t > n then break
First: create incomplete constituent
$C[s][t][\leftarrow][0] = \sum_{s \leq r < t} (C[s][r][\rightarrow][2] + C[r + 1][t][\leftarrow][1] + S(t, s))$
$C[s][t][\rightarrow][0] = \sum_{s \leq r < t} (C[s][r][\rightarrow][1] + C[r + 1][t][\leftarrow][2] + S(s, t))$
Second: create complete constituent
$C[s][t][\leftarrow][1] = \sum_{s \leq r < t} (C[s][r][\leftarrow][2] + C[r][t][\leftarrow][0])$
$C[s][t][\rightarrow][1] = \sum_{s < r \leq t} (C[s][r][\rightarrow][0] + C[r][t][\rightarrow][2])$
Second: create complete constituent stop
$C[s][t][\leftarrow][2] = \sum_{s \leq r < t} (C[s][r][\leftarrow][1])$
$C[s][t][\rightarrow][2] = \sum_{s < r \leq t} (C[s][r][\rightarrow][1])$
end for
end for
Return C[0][n][→][1] as the highest score for any parse

3 Algorithm

The motive of the parser is to obtain the parameters of the grammar which are

- The probability that a head word takes a modifier (depProb[h, m, direction])
- The probability that a head word continues to take further arguments (contProb[h, direction, adj])
- The probability that a head word stops taking further arguments (stopProb[h, direction, adj]).

An open source library called PyDecode is used to build a hypergraph, with all the 3 constituents of the Eisner's parsing algorithm as its nodes. An edge has head word, modifier word, direction, adjacency and state values stored in it. The direction indicates the direction in which the edge is being formed. The adjacency indicates if the modifier word is the first child of the head word, in which case it is "adj", or not, in which case it is "non-adj". If an edge does not have a modifier, the modifier word is "—".The edges of the hypergraph are assigned weights according to algorithm given in Figure 2:

Lets assume incomplete constituent as i.c and complete constiuent stop as c.s
if edge.headNode ϵ i.c **then**
    return depProb[edge.headWord, edge.modifierWord, edge.dir] * contProb[edge.headWord, edge.dir, edge.isAdj]
else if edge.headNode ϵ c.s **then**
    return stopProb[edge.headWord, edge.dir, edge.isAdj]
else
    return 1
end if

The insideOutside algorithm is run on the entire hypergraph. The inside probability of the root of the hypergraph gives the total probability of the sentence Z. The marginals of the nodes and the edges of the hypergraph are computed using the PyDecode library. The marginals = marginals / Z gives the counts for the EM algorithm.

The em algorithm is run 20 times for all the sentences. The hypergraph is built for each sentence. The stop and dep probabilities are incremented according to the algorithm in Figure 3:

4 EM

- sentence; $w_1 \dots w_n$
- vocabulary; $\mathcal{W}$

```
Lets assume incomplete constituent as i.c and complete constiuent stop as c.s
for edge in hypergraph.edges do
    headWord, modWord, direct, adj, state = edge.label.split()
    if edge.headNode ϵ i.c then
        depCounts[headWord, modWord, direct, adj] += marginals[edge.label]
    end if
    if edge.headNode ϵ c.s then
        stopCounts[headWord, direct, adj] += marginals[edge.label]
    end if
end for
```

- modifier; $m \in \{1 \dots n\}$
- head; $h \in \{0 \dots n\}$
- direction; $\mathcal{D} = \{\text{L}, \text{R}\}$
- let the type of incomplete constituent be Trap, complete constituent be Tri and complete stop constituent be TriStop
- edge; $\mathcal{E} = \{E.headWord \epsilon Trap, E.headWord \epsilon Tri, E.headWord \epsilon TriStop\}$ The label of an edge comprises of headWord h, modifier m, direction dir, CONT, ADJ as mentioned in the earlier section
- marginals $p(edge)$

The probabilities

- $p(\text{CONT}|w, dir, \text{ADJ})$; $\text{CONT} \in \{0, 1\}$, $w \in \mathcal{W}$, $\text{ADJ} \in \{0, 1\}$, $dir \in \{0, 1\}$
- $stopCounts(w, dir, \text{ADJ})$
- $p(m|h, dir, \text{ADJ})$; $\text{CONT} \in \{0, 1\}$, $h, m \in \mathcal{W}$, $\text{ADJ} \in \{0, 1\}$
- $depCount(h, m, dir)$
- $depCount(h, m, dir, \text{ADJ})$

Estimation Step
Fill in the c charts.

$$stopCounts(h, dir, \text{ADJ} = 1) \leftarrow \sum p(\text{E(h, dir, ADJ} = 1, \text{CONT} = 0))$$

$$stopCounts(h, dir, \text{ADJ} = 0) \leftarrow \sum p(\text{E(h, dir, ADJ} = 0, \text{CONT} = 0))$$

$$depCounts(h, m, dir, \text{ADJ}) \leftarrow \sum p(\text{E(h, m!} = --, \text{dir, ADJ, CONT} = 1))$$

$$depCounts(h, m, dir) \leftarrow \sum_{ADJ=\{0,1\}} p(\mathrm{E(h, m! = --, dir, CONT = 1)})$$

Maximization Step

$$p(\mathrm{CONT = 0}|h, dir, \mathrm{ADJ}) \leftarrow \frac{stopCounts(h, dir, \mathrm{ADJ})}{\sum_{m \in \mathcal{W}} depCounts(h, m, dir, ADJ) + stopCounts(h, dir, \mathrm{ADJ})}$$

$$p(m|h, dir) \leftarrow \frac{depCounts(h, m, dir)}{\sum_{m \in \mathcal{W}} depCounts(h, m, dir)}$$