Dependency Parsing

Allan Jie

February 20, 2016

Slides: `http://www.statnlp.org/dp.html`

Table of Contents

Dependency

Dependency focus on the relationship between words and further solve the ambiguity.

For example, dependency for a sentence: *I saw a girl with a telescope*.

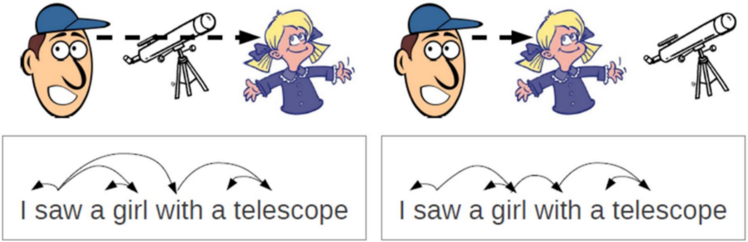


Figure: Dependency for resolving ambiguity

We can see from Fig. 1 that same sentence may have different meanings which lead to different dependency structures in two boxes.

Labeled/Unlabeled Dependency

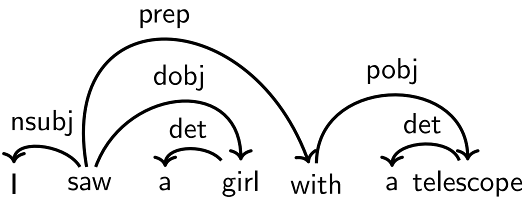
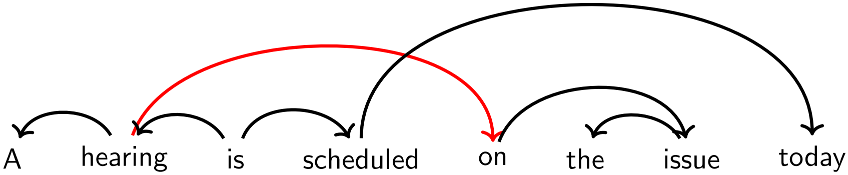


Table: Dependency Relations

Label	Description	Label	Description
nsubj	nominal subject	tmod	temporal modifier
csubj	clausal subject	appos	appositional modifier
dobj	direct object	det	determiner
iobj	indirect object	prep	prepositional modifier
pobj	object of preposition		

Projective/Non-projective Dependency

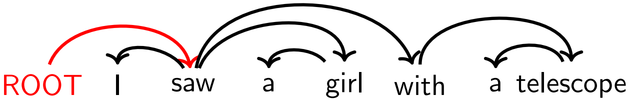
Informally, “**projective**” dependency structure means the tree does not contain any crossing arcs as shown in previous example. We focus on projective dependency parsing.
A non-projective dependency tree contains crossing arcs.

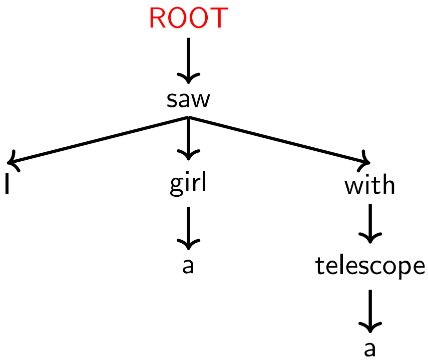


Example from “Dependency Parsing” by Kubler, Nivre, and McDonald, 2009

All the examples above are dependency with flatten structure. We can view it from a tree perspective. The dependency tree structure satisfy the following constraints: single-headed, connected and acylic.

In practice, we make a ROOT word as the final head of the dependency tree.




ROOT
saw
I
girl
with
a
telescope
a

Problem Description: Dependency Parsing Task

How to find the dependency structure (tree)?

Given an input sentence $\mathbf{x}$, the corresponding dependency tree $\mathbf{y}^*$ is:

$$\mathbf{y}^* = \arg\max_{y \in GEN(\mathbf{x})} \text{score}(x, y),$$

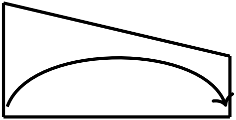
where $GEN(\mathbf{x})$ is set of all dependency structures for $\mathbf{x}$.

Table of Contents

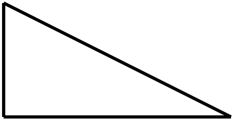
Eisner's Dynamic Programming Algorithm

There are many algorithms for dependency parsing but this introduction focus on Eisner's Algorithm since we will use it later.

There are two basic structures in dynamic programming derivation:

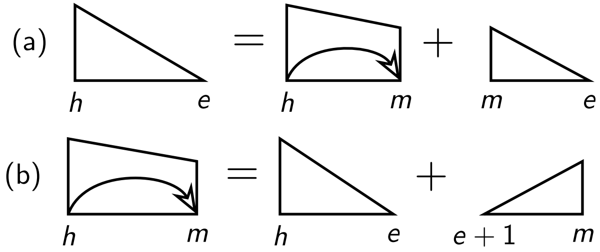


Incomplete Subtree (Span)



Complete Subtree (Span)

Dynamic Programming Derivation

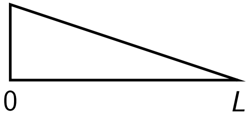


Where h, e and m are the indices of the sentence. The arc from x_h to x_m id the dependency between them. Due to space limit, the symmetric part of right-headed spans' derivation is elided.

The complexity of our parsing algorithm is $\mathcal{O}(n^3)$, which is same as the first-order parsing algorithm.

Dynamic Programming Derivation

Finally, we can construct the complete span of the sentence from index 0 to its length using a bottom-up approach.



Algorithm 1 Eisner's Dynamic Programming Algorithm

Initialize the table: $C[s][s][d][c] = 0; \quad \forall s \in \{0, ..., n\}, d \in \{\leftarrow, \rightarrow\}, c \in \{0, 1\}$
for k in 1 to n **do**
  for s in 0 to n **do**
    $t = s + k$
    if $t > n$ **then**
      break
    end if
    //build the incomplete spans
    $C[s][t][\leftarrow][0] = \max_{s \leq u < t}(C[s][u][\rightarrow][1] + C[u+1][t][\leftarrow][1] + s(t, s))$
    $C[s][t][\rightarrow][0] = \max_{s \leq u < t}(C[s][u][\rightarrow][1] + C[u+1][t][\leftarrow][1] + s(s, t))$
    //build the complete spans
    $C[s][t][\leftarrow][1] = \max_{s \leq u < t}(C[s][u][\leftarrow][1] + C[u][t][\leftarrow][0])$
    $C[s][t][\rightarrow][1] = \max_{s \leq u < t}(C[s][u][\rightarrow][0] + C[u][t][\rightarrow][1])$
  end for
end for
return $C[0][n][\rightarrow][1]$ as the best parse.

Table of Contents

Tree-CRF Model

With Eisner's dynamic programming derivation, the score for all the possible structures can be efficiently calculated by inside-outside algorithm. We used a discriminative tree-CRF model to learn the weights, which is formulated as the following log-linear equation:

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp(\sum_k \lambda_k f_k(\mathbf{x}, \mathbf{y}))}{\sum_y \exp(\sum_k \lambda_k f_k(\mathbf{x}, \mathbf{y}))},$$

where y here is the dependency structure and x is the sentence. $f_k(x, y)$ is the k^{th} feature function for sample pair (x, y) and λ_k is the corresponding weight for the feature function.
The dynamic programming derivation is used for gradient calculation.

Features

Features are adopted from "Online Large-Margin Training of Dependency Parsers" by McDonald et al.

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

Table 1: Features used by system. p-word: word of parent node in dependency tree. c-word: word of child node. p-pos: POS of parent node. c-pos: POS of child node. p-pos+1: POS to the right of parent in sentence. p-pos-1: POS to the left of parent. c-pos+1: POS to the right of child. c-pos-1: POS to the left of child. b-pos: POS of a word in between parent and child nodes.