Topics in Parsing: Context and Markovization; Dependency Parsing

COMP-599

Oct 17, 2016

Outline

Review

Incorporating context

- Markovization
- Learning the context

Dependency parsing

- Eisner's algorithm

Review of CYK

Describe the general process of the CYK algorithm

- Is it top-down or bottom-up? What does this mean?
- What is the chart used for? What are the entries in the cells?
- What did those arrows that we drew mean?

Vanilla PCFGs

Estimate of rule probabilities:

- MLE estimates:

$$\Pr(\alpha \rightarrow \beta) = \frac{\#(\alpha \rightarrow \beta)}{\#\alpha}$$

- e.g., Pr(S -> NP VP) = #(S -> NP VP) / #(S)
    - Recall: these distributions are normalized by LHS symbol

Even with smoothing, doesn’t work very well:

- Not enough context
- Rules are too sparse

Subject vs Object NPs

NPs in subject and object positions are not identically distributed:

- Obvious cases – pronouns (*I* vs *me*)
  - But both appear as NP -> PRP -> *I/me*
- Less obvious: certain classes of nouns are more likely to appear in subject than object position, and vice versa.
  - For example, subjects tend to be **animate** (usually, humans, animals, other moving objects)

Many other cases of obvious dependencies between distant parts of the syntactic tree.

Sparsity

Consider subcategorization of verbs, with modifiers

• *ate*	VP -> VBD
• *ate quickly*	VP -> VBD AdvP
• *ate with a fork*	VP -> VBD PP
• *ate a sandwich*	VP -> VBD NP
• *ate a sandwich quickly*	VP -> VBD NP AdvP
• *ate a sandwich with a fork*	VP -> VBD NP PP
• *quickly ate a sandwich with a fork*	VP -> AdvP VBD NP PP

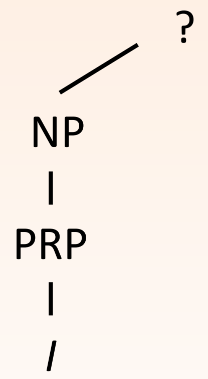
We should be able to factorize the probabilities:

- of having an adverbial modifier, of having a PP modifier, etc.

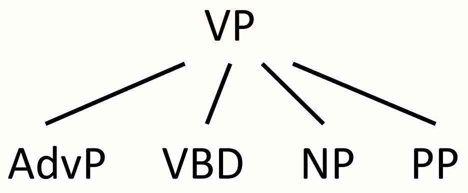
Wrong Independence Assumptions

Vanilla PCFGs make independence assumptions that are too strong AND too weak.

Too strong: *vertically*, up and down the syntax tree



Too weak: *horizontally*, across the RHS of a production



Adding Context

Add more context vertically to the PCFG

- Annotate with the parent category

Before: NP -> PRP, NP -> Det NN, etc.

Now:

Subjects:

NP^S -> PRP, NP^S -> Det NN, etc.

Objects:

NP^VP -> PRP, NP^VP -> Det NN, etc.

Learn the probabilities of the rules separately (though they may influence each other through interpolation/smoothing)

Example

Let’s help Pierre Vinken find his ancestors.

```
( (S
    (NP
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP (NNP Nov.) (CD 29) )))
    (. .) ))
```

Note that the tree here is given in bracket parse format, rather than drawn out as a graph.

Removing Context

Conversely, we break down the RHS of the rule when estimating its probability.

Before:	Pr(VP -> START AdvP VBD NP PP END) as a unit
Now:	Pr(VP -> START AdvP) *
	Pr(VP -> AdvP VBD) *
	Pr(VP -> VBD NP) *
	Pr(VP -> NP PP) *
	Pr(VP -> PP END)

- In other words, we're making the same N-gram assumption as in language modelling, only over non-terminal categories rather than words.
- Learn probability of factors separately

Example

Let's help Pierre Vinken find his children.

```
( (S
    (NP
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
        (PP (IN as)
          (NP (DT a) (JJ nonexecutive) (NN director) ))
        (NP (NNP Nov.) (CD 29) )))
    (. .) ))
```

Markovization

Vertical markovization: adding ancestors as context

- Zeroth order – vanilla PCFGs
- First order – the scheme we just described
- Can go further:
  - e.g., Second order: NP^VP^S -> …

Horizontal markovization: breaking RHS into parts

- Infinite order – vanilla PCFGs
- First order – the scheme we just described
- Can choose any other order, do interpolation, etc.

Evaluating Parsers

How well does this work in practice?

First need a measure of the performance of a parser!

Usually measure at the level of *constituents*

```
(VP (VB join) (NP (DT the) (NN board)))
```

- Constituents here are the VP, and the NP
    - We shouldn’t really count the leaf nodes (VB, DT, NN), as these are the POS tags, but common measures often do!
- Two things to consider:
    - **Gold standard** – the correct parse
    - **System prediction** – the output of our parser

Recall

Of the constituents in the **gold standard**, what percentage of them were correctly recovered?

e.g., Gold standard:

[A [B C [D E]] [F G]]

System prediction:

[A B [C [D E]] [F G]]

Precision

Of the constituents in the **system prediction**, what percentage of them are actually correct?

e.g., Gold standard:

[A [B C [D E]] [F G]]

System prediction:

[A B [C [D E]] [F G]]

Game the Measure

How can we get near-100% precision or near-100% recall without doing any real work?

Recall?

Precision?

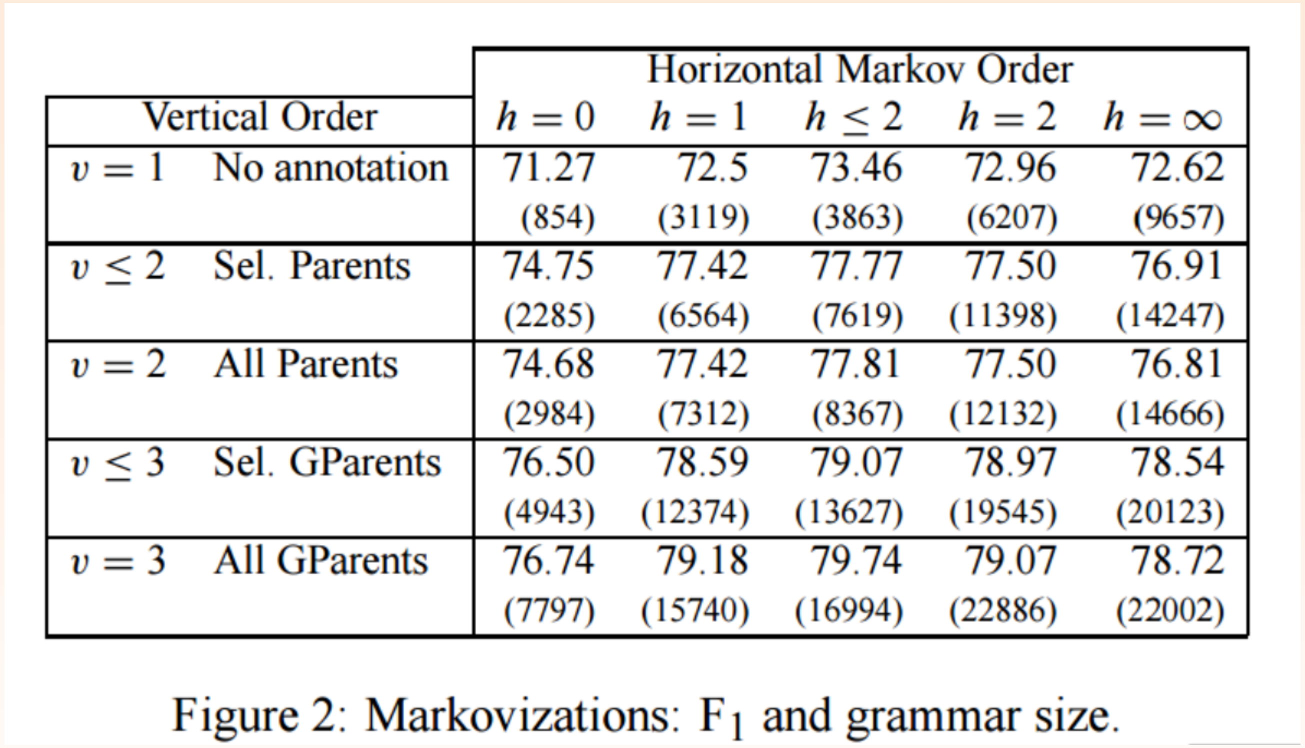
F1-measure

Take a harmonic mean between Recall and Precision:

$$F_1 = \frac{2 * P * R}{P + R}$$

- Can only do well on F1 if system does well on both recall and precision
- F1 suffers if P and R are highly imbalanced

Effect of Category Splitting



Vertical Order		Horizontal Markov Order				
		$h = 0$	$h = 1$	$h \leq 2$	$h = 2$	$h = \infty$
$v = 1$	No annotation	71.27 (854)	72.5 (3119)	73.46 (3863)	72.96 (6207)	72.62 (9657)
$v \leq 2$	Sel. Parents	74.75 (2285)	77.42 (6564)	77.77 (7619)	77.50 (11398)	76.91 (14247)
$v = 2$	All Parents	74.68 (2984)	77.42 (7312)	77.81 (8367)	77.50 (12132)	76.81 (14666)
$v \leq 3$	Sel. GParents	76.50 (4943)	78.59 (12374)	79.07 (13627)	78.97 (19545)	78.54 (20123)
$v = 3$	All GParents	76.74 (7797)	79.18 (15740)	79.74 (16994)	79.07 (22886)	78.72 (22002)

Figure 2: Markovizations: F_1 and grammar size.

WSJ results by Klein and Manning (2003)

- With additional heuristics from linguistic insights, they got up to 87.04 F1

Can We Learn These Distinctions?

Above: human linguistic insights to make splits

- NP split into NP^S and NP^VP for subjects and objects

We are in AI! Let's automate this too!

Petrov and Klein (2006)

Basic idea:

- Introduce a latent variable associated with each nonterminal
- NP becomes NP-1, NP-2, NP-3, … NP-*k*
- Adaptively increase and decrease *k* for each non-terminal category to maximize training corpus likelihood
- Called the **split-merge** algorithm

Annotated Rules

Rules are now in the following form:

$$A_x \rightarrow B_y C_z$$

e.g., $S_3 \rightarrow NP_1\ VP_4$

or $DT_2 \rightarrow the$

And we need to learn the probabilities of each of these rules

Split-Merge Algorithm

Overall algorithm:

- Start off with the original, initial grammar, deriving probability estimates in the usual way
- Do for a n iterations:
    - **Split** the grammar by duplicating each non-terminal symbol
    - Get latent annotations over the training corpus in order to update the probabilities of the rules
    - **Merge** by merging together some of the subsymbols back into one subsymbol

Splitting

Suppose we currently have *n* states in the grammar for a non-terminal A. After splitting, we'll have 2*n* states.

Split A_x into $A_{x'}$ and $A_{x''}$:

Case 1: A_x on LHS. i.e., $P(A_x \rightarrow\ B_y\ C_z)$:

Copy probabilities

- Set $P\left(A_{x'} \rightarrow\ B_y\ C_z\right)$ to $P(A_x \rightarrow\ B_y\ C_z)$
- Set $P\left(A_{x''} \rightarrow\ B_y\ C_z\right)$ to $P(A_x \rightarrow\ B_y\ C_z)$

Case 2: A_x on RHS. i.e., $P(D_r \rightarrow A_x\ E_s)$:

Halve the probabilities

- Set $P(D_r \rightarrow A_{x'}\ E_s)$ to $P(D_r \rightarrow A_x\ E_s)/\ 2$
- Set $P(D_r \rightarrow A_{x''}\ E_s)$ to $P(D_r \rightarrow A_x\ E_s)/\ 2$

Randomness

To make the two new states different from each other, we'll also add a little bit of randomness to the probabilities.

e.g., Copy 0.46 to be 0.452 and 0.464

Halve 0.5 to 0.2501 and 0.2449

Just make sure to renormalize the distributions properly.

We'll see why this is important.

Learning New Grammar

Now that we have latent variables, we can't use simple MLE or MAP estimates for the rule probabilities.

- If we did have the latent variable annotations, we could do this, but we don't.
- What algorithm did we discuss before that solved this same problem?

Expectation Maximization Again

Use a version of EM to predict the label annotations in the trees in the training set

Starting with the probabilistic grammar after splitting:

- "Guess" the labels of the trees (E-step)
- Improve the grammar based on the guesses (M-step)

Without randomness:

Everything would be symmetric

The two subsymbols would be equally likely in all cases in E-step.

Since everything is tied, estimates never improve in M-step!

Merging

We have *n* subsymbols, and want to merge two of them together (result, *n-1* subsumbols).

- Try merging each pair – see how much training corpus likelihood suffers.
- Merge the pair with the lowest loss
- How to calculate training corpus likelihood loss?

See paper for more details

Results

After six split-merge-smooth cycles, P/R results improve to 89.8/89.6.

This is despite not have any manual linguistic annotations or complex feature extraction!

Interesting hierarchies can also be observed over the course of training:

Hierarchical Learning

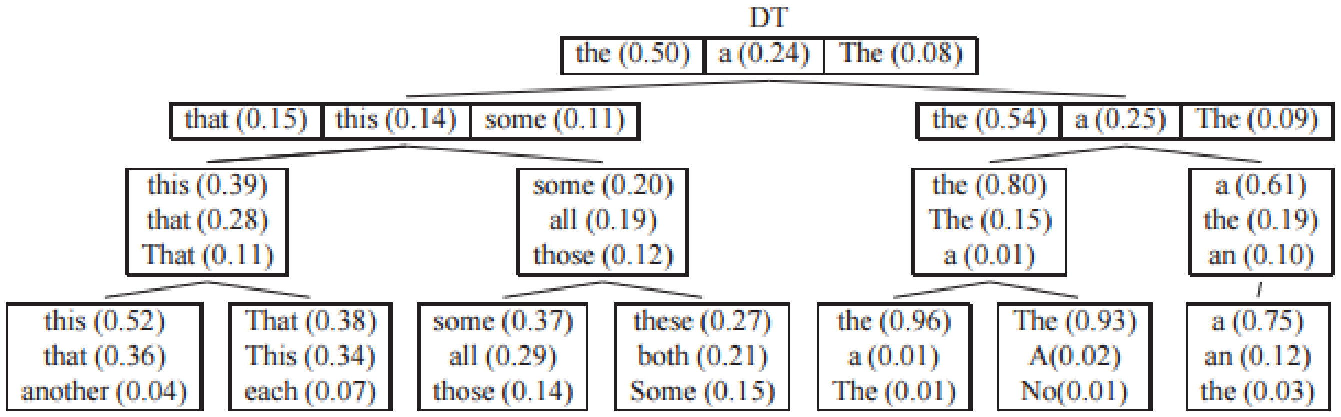

DT
the (0.50) a (0.24) The (0.08)
that (0.15) this (0.14) some (0.11)
the (0.54) a (0.25) The (0.09)
this (0.39) that (0.28) That (0.11)
some (0.20) all (0.19) those (0.12)
the (0.80) The (0.15) a (0.01)
a (0.61) the (0.19) an (0.10)
this (0.52) that (0.36) another (0.04)
That (0.38) This (0.34) each (0.07)
some (0.37) all (0.29) those (0.14)
these (0.27) both (0.21) Some (0.15)
the (0.96) a (0.01) The (0.01)
The (0.93) A(0.02) No(0.01)
a (0.75) an (0.12) the (0.03)

Figure 2: Evolution of the DT tag during hierarchical splitting and merging. Shown are the top three words for each subcategory and their respective probability.

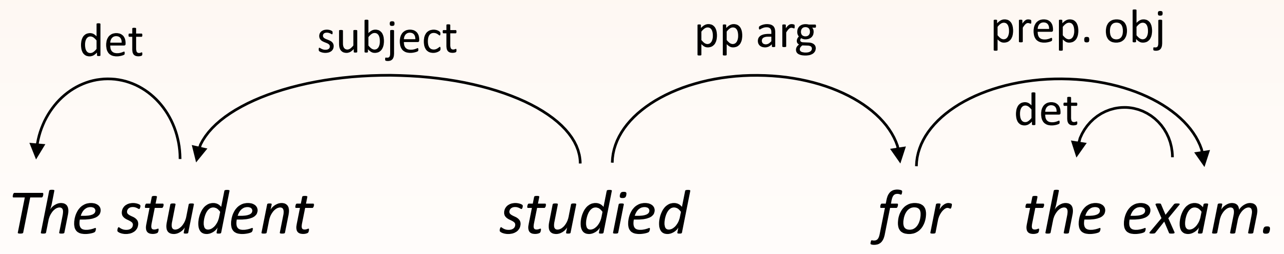
Lessons Learned

We have seen a method that seems to partially automate the job of a linguist!

Results in improved parsing performance

EM can be applied in many settings with latent variables, such as with tree structures.

What About Dependency Parsing?



Eisner's Algorithm (1997)

A $O(N^3)$ dynamic programming algorithm for dependency parsing:

Find

$$\underset{t \in \tau(\text{sent})}{\operatorname{argmax}} S(t)$$

Assumptions

Trees are projective (no crossing dependencies)

Score of tree is equal to the sum of the score of each of edge in the tree

$$\text{S}(t) = \sum_{(a,b) \in t} S(a, b)$$

Learning S(a, b)

$S(a, b)$ represents the “goodness” of an edge headed by word a with dependent word b.

How do we learn this?

- Define some heuristics (Eisner, 1997)
- e.g., learn how often these words are in a dependency relation in a training corpus
- As part of parsing (McDonald et al., ACL 2005)

General Idea

Just as in CKY parsing, start by parsing subspans, then combine them in order to form larger spans.

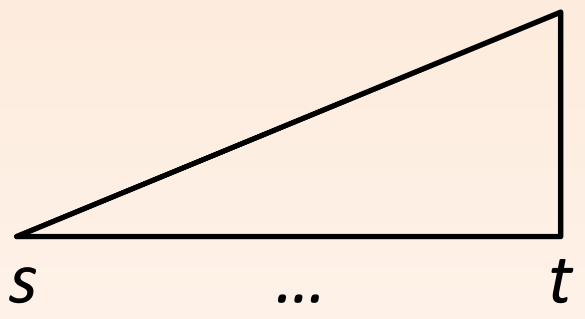
```
for k : 1 … n
  for s : 1 … n
    # find substructures in [s, s + k]
```

Distinguish **left** and **right** subtrees

Distinguish **complete** and **incomplete** subtrees

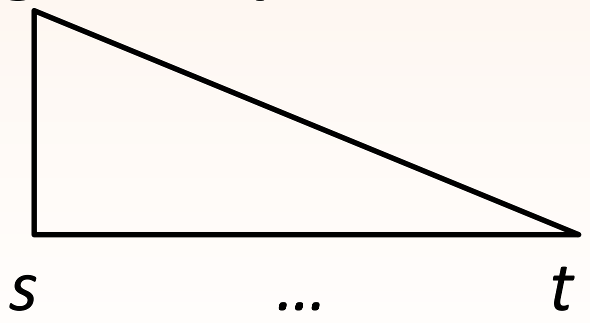
Complete Subtrees

Left complete subtree



- headed by t
- words [s, t) have no more dependents
- t may have more dependents

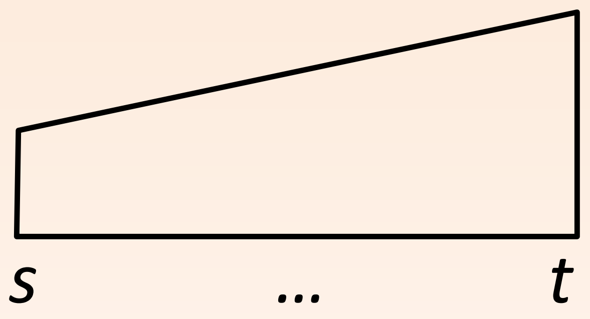
Right complete subtree



- headed by s
- words (s, t] have no more dependents
- s may have more dependents

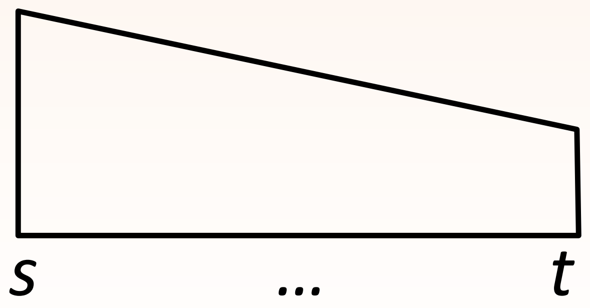
Incomplete Subtrees

Left incomplete subtree



- headed by t
- words [s, t) may have more dependents
- edge (t, s) exists

Right incomplete subtree



- headed by s
- words (s, t] may have more dependents
- edge (s, t) exists

Defining the Chart

Let $C[s][t][x][y]$ be the score of the best possible subtree such that:

- it spans words [s, t]
- $x \in \{\leftarrow, \rightarrow\}$ (left or right subtree)
- $y \in \{0, 1\}$ (incomplete or complete)

Recurrence Relations

$$C[s][t][\leftarrow][0] = \max_{s \le r < t} C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + S(t,s)$$

$$C[s][t][\rightarrow][0] = \max_{s \le r < t} C[s][r][\rightarrow][1] + C[r+1][t][\leftarrow][1] + S(s,t)$$

$$C[s][t][\leftarrow][1] = \max_{s \le r < t} C[s][r][\leftarrow][1] + C[r][t][\leftarrow][0]$$

$$C[s][t][\rightarrow][1] = \max_{s < r \le t} C[s][r][\rightarrow][0] + C[r][t][\rightarrow][1]$$

Notes

Base case of $C[s][s][*][*] = 0$

Notice that in the recurrence, chart cells only depend on cells with a shorter span.

Thus, the recurrence (and the algorithm) works.

Let's do an example, graphically, to understand the algorithm in more detail.