Natural Language Processing

Lecture 19.1—3/17/2015

Martha Palmer

Solution 1: Rule Rewriting

- The grammar rewriting approach attempts to capture local tree information by rewriting the grammar so that the rules capture the regularities we want.
  - By splitting and merging the non-terminals in the grammar.
    - Example: split NPs into different classes…
  - Remember, we rewrote the grammar rules for CKY, and we rewrote the IOB tags.

Example: NPs

- Our CFG rules for NPs don't condition on where in a tree the rule is applied
- But we know that not all the rules occur with equal frequency in all contexts.
  - Consider *NP*s that involve pronouns vs. those that don't.

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

Other Examples

- There are lots of other examples like this in any treebank
  - Many at the part of speech level
  - Recall that many decisions made in annotation efforts are directed towards improving annotator agreement, not towards doing the right thing.
    - Often this involves conflating distinct classes into a larger class
      - TO, IN, Det, etc.

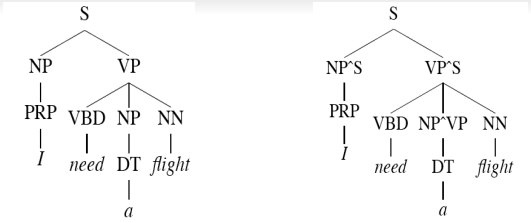
Rule Rewriting

- Three approaches
  - Use linguistic knowledge to directly rewrite rules by hand
    - NP_Obj and the NP_Subj approach
  - Automatically rewrite the rules using context to capture some of what we want
    - Ie. Incorporate context into a context-free approach
  - Search through the space of rewrites for the grammar that maximizes the probability of the training set

Local Context Approach

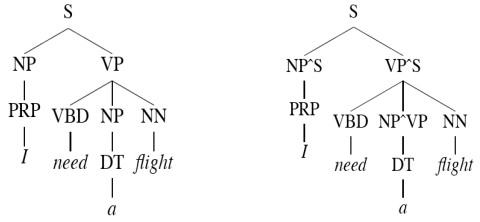
- Condition the rules based on their parent nodes
  - This splitting based on tree-context captures some of the linguistic intuitions

Parent Annotation



- Now we have non-terminals NP^S and NP^VP that should capture the subject/object and pronoun/full NP cases. That is...
  - The rules are now
    - NP^S -> PRP
    - NP^VP -> DT
    - VP^S -> NP^VP

Parent Annotation



- Recall what's going on here. We're in effect rewriting the treebank, thus rewriting the grammar.
- And changing the probabilities since they're being derived from different counts…
  - And if we're splitting what's happening to the counts?

Auto Rewriting

- If this is such a good idea we may as well apply a learning approach to it.
- Start with a grammar (perhaps a treebank grammar)
- Search through the space of splits/merges for the grammar that in some sense maximizes parsing performance on the training/development set.

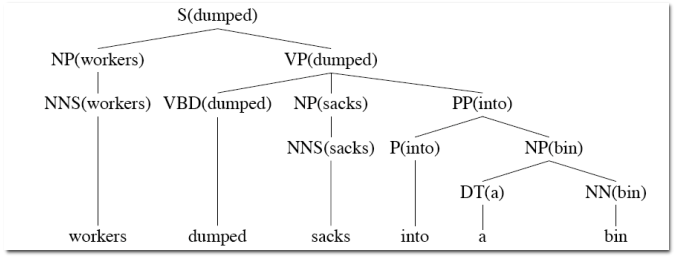
Auto Rewriting

- Basic idea…
  - Split every non-terminal into two new non-terminals across the entire grammar (X becomes X1 and X2).
  - Duplicate all the rules of the grammar that use X, dividing the probability mass of the original rule almost equally.
  - Run EM to readjust the rule probabilities
  - Perform a merge step to back off the splits that look like they don't really do any good.

Solution 2: Lexicalized Grammars

- Lexicalize the grammars with heads
- Compute the rule probabilities on these lexicalized rules
- Run Prob CKY as before

Dumped Example



How?

- We used to have
  - VP -> V NP PP P(rule|VP)
    - That's the count of this rule divided by the number of VPs in a treebank
- Now we have fully lexicalized rules...
  - VP(dumped)-> V(dumped) NP(sacks)PP(into)

P(r|VP ^ dumped is the verb ^ sacks is the head of the NP ^ into is the head of the PP)

To get the counts for that..

Declare Independence

- When stuck, exploit independence and collect the statistics you can…
- There are a larger number of ways to do this...
- Let's consider one generative story: given a rule we'll
  1. Generate the head
  2. Generate the stuff to the left of the head
  3. Generate the stuff to the right of the head

Example

- So the probability of a lexicalized rule such as
  - VP(dumped) → V(dumped)NP(sacks)PP(into)
- Is the product of the probability of
  - *"dumped"* as the head
  - With nothing to its left
  - "*sacks*" as the head of the first right-side thing
  - "*into*" as the head of the next right-side element
  - And nothing after that

Example

- That is, the rule probability for

$$P(VP(dumped,VBD) \rightarrow VBD(dumped, VBD)\ NP(sacks,NNS)\ PP(into,P))$$

is estimated as

$$\begin{aligned} P_H(VBD|VP, dumped) &\times P_L(STOP|VP,VBD,dumped) \\ &\times P_R(NP(sacks,NNS)|VP,VBD,dumped) \\ &\times P_R(PP(into,P)|VP,VBD,dumped) \\ &\times P_R(STOP|VP,VBD,dumped) \end{aligned}$$

Framework

- That's just one simple model
  - Collins Model 1
- You can imagine a gazzillion other assumptions that might lead to better models
- You just have to make sure that you can get the counts you need
- And that it can be used/exploited efficiently during decoding

Features

- C for Case, Subjective/Objective
  - *She visited her.*
- P for Person agreement, (1st, 2nd, 3rd)
  - *I like him, You like him, He likes him,*
- N for Number agreement, Subject/Verb
  - *He likes him, They like him.*
- G for Gender agreement, Subject/Verb
  - English, reflexive pronouns *He washed himself.*
  - Romance languages, det/noun
- T for Tense,
  - auxiliaries, sentential complements, etc.
  - ** will finished* is bad