Natural Language Processing

Lecture 17—10/22/2015
Jim Martin

Today

- Wrap up Statistical Parsing (Chapter 14)

Simple Probability Model

- A derivation (tree) consists of the collection of grammar rules that are in the tree
  - The probability of a tree is the product of the probabilities of the rules in the derivation.

$$P(T,S) = \prod_{node \in T} P(rule(n))$$

Rule Probabilities

- So... What's the probability of a rule?
- Start at the top...
  - A tree should have an *S* at the top. So given that we know we need an *S*, we can ask about the probability of each particular *S* rule in the grammar.
    - That is P(particular S rule | S is what I need)
- So in general we need

$$P(\alpha \rightarrow \beta \mid \alpha)$$

For each rule in the grammar

Training the Model

- We can get the estimates we need from an annotated database (i.e., a treebank)

$$P(\alpha \rightarrow \beta|\alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_\gamma \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

  - For example, to get the probability for a particular *VP* rule, just count all the times the rule is used and divide by the number of *VP*s overall.

Question

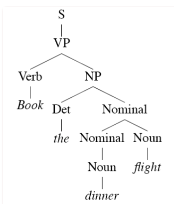
- Given this approach…

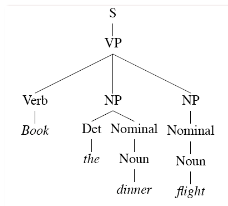
$$P(\alpha \rightarrow \beta|\alpha) = \frac{\text{Count}(\alpha \rightarrow \beta)}{\sum_\gamma \text{Count}(\alpha \rightarrow \gamma)} = \frac{\text{Count}(\alpha \rightarrow \beta)}{\text{Count}(\alpha)}$$

- Do we have to worry about smoothing to deal with zero counts?

Example

- Consider...
  - *Book the dinner flight*





Examples

- These trees consist of the following rules.

Rules			P	Rules			P
S	→	VP	.05	S	→	VP	.05
VP	→	Verb NP	.20	VP	→	Verb NP NP	.10
NP	→	Det Nominal	.20	NP	→	Det Nominal	.20
Nominal	→	Nominal Noun	.20	NP	→	Nominal	.15
Nominal	→	Noun	.75	Nominal	→	Noun	.75
				Nominal	→	Noun	.75
Verb	→	book	.30	Verb	→	book	.30
Det	→	the	.60	Det	→	the	.60
Noun	→	dinner	.10	Noun	→	dinner	.10
Noun	→	flights	.40	Noun	→	flights	.40

$$P(T_{left}) = .05 * .20 * .20 * .20 * .75 * .30 * .60 * .10 * .40 = \mathbf{2.2 \times 10^{-6}}$$
$$P(T_{right}) = .05 * .10 * .20 * .15 * .75 * .75 * .30 * .60 * .10 * .40 = \mathbf{6.1 \times 10^{-7}}$$

Probabilistic CKY

```
function PROBABILISTIC-CKY(words,grammar) returns most probable parse
                                              and its probability
  for j ← from 1 to LENGTH(words) do
     for all { A | A → words[j] ∈ grammar}
        table[j−1, j, A] ← P(A → words[j])
     for i ← from j−2 downto 0 do
        for k ← i+1 to j−1 do
             for all { A | A → BC ∈ grammar,
                            and table[i,k,B] > 0 and table[k,j,C] > 0 }
                 if (table[i,j,A] < P(A → BC) × table[i,k,B] × table[k,j,C]) then
                    table[i,j,A] ← P(A → BC) × table[i,k,B] × table[k,j,C]
                    back[i,j,A] ← {k,B,C}
     return BUILD_TREE(back[1, LENGTH(words), S]), table[1, LENGTH(words), S]
```

Problems with Basic PCFGs

- The probability model we're using is just based on the the bag of rules in the derivation…
  1. Doesn't take the actual words into account in any useful way.
  2. Doesn't take into account *where* in the derivation a rule is used
  3. *Doesn't work terribly well*
     - That is, the most probable parse isn't usually the right one (the one in the treebank test set).

Evaluation

- First, how do we measure how well a parser is working?
  - Assume we have a training/dev set from a treebank so we have "reference" answers for some set of trees.
- We could look for straight accuracy across a test set of sentences
  - How many sentences received exactly the correct parse?

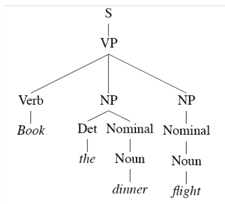
Evaluation

- That's too depressing
- And not informative enough --- we might be making useful changes to the system and not see any improvement given this metric
  - The trees are getting better, but they're still not right.
- A better metric looks at the contents of the reference tree and the hypothesis tree

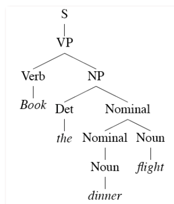
Example

- Consider...
  - *Book the dinner flight*

Parser output



Reference answer



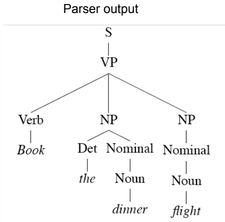
Evaluation

- Precision
  - What fraction of the sub-trees in the hypothesis match corresponding sub-trees in the reference answer?
    - How much of what we're producing is right?
- Recall
  - What fraction of the sub-trees in the reference answer did we actually get?
    - How much of what we should have gotten did we actually get?

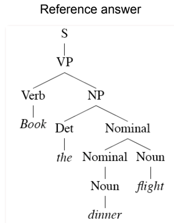
Example

- Consider...
  - *Book the dinner flight*

Parser output



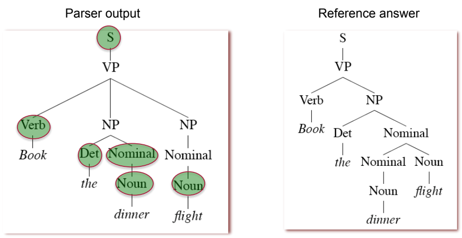
Reference answer



Example: Precision

- Consider...
  - *Book the dinner flight*

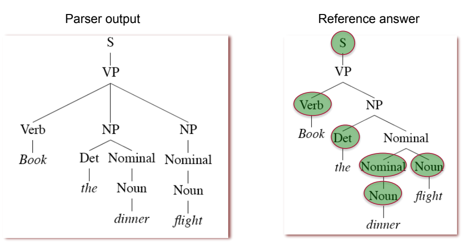
Precision: 6/10



Example: Recall

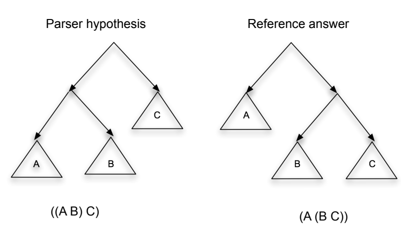
- Consider...
  - *Book the dinner flight*

Precision: 6/11



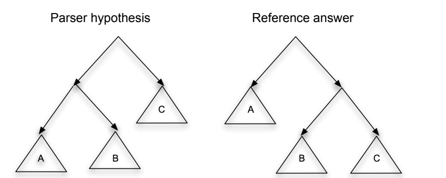
Evaluation

- Crossing brackets



Evaluation

- Crossing brackets

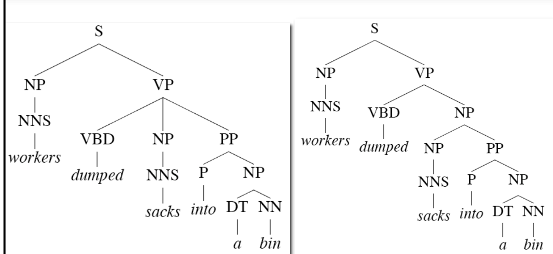


(book (the dinner) flight) (book (the (dinner flight)))

Sources of Difficulty for PCFGs

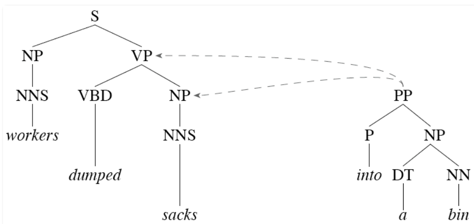
- Attachment ambiguities
  - PP attachment
  - Coordination problems

PP Attachment



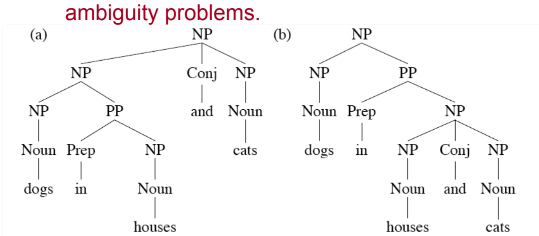
PP Attachment

- Another view.



Coordination

Most grammars have a rule (implicitly) of the form
X -> X and X. This leads to massive ambiguity problems.



Better Statistical Parsing

- We'll look at two approaches to overcoming these shortcomings
  1. Rewriting the grammar to better capture the dependencies among rules
  2. Integrate lexical dependencies into the model
     1. And come up with the independence assumptions needed to make it work.

Solution 1: Rule Rewriting

- The grammar rewriting approach attempts to better 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… that is, split the NP rules into separate rules

Motivation: NPs

- Our CFG rules for NPs don't condition on where in a tree the rule is applied (that's why they're context-free)
- 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%

Example: NPs

- So that comes down to
  - NP --> Pronoun
- Gets replaced with something like
  - NP_Subj --> Pronoun
  - NP_Obj --> Pronoun

Separate rules, with different counts in the treebank and therefore different probabilities

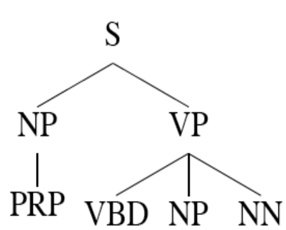
Rule Rewriting

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

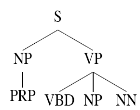
Local Context Approach

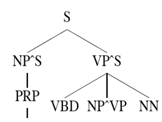
- Condition the rules based on parent nodes
  - Splitting based on tree-context captures some of the linguistic intuitions we saw with the NP example

Parent Annotation



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…
  - NP^S -> PRP
  - NP^VP -> DT
  - VP^S -> NP^VP

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.

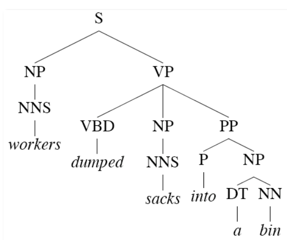
Auto Rewriting

- 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.

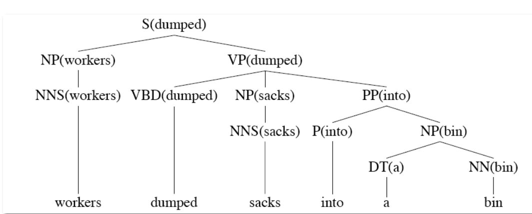
Solution 2: Lexicalized Grammars

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

Dumped Example



Dumped Example



How?

- We used to have
  - VP -> V NP PP   P(this 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 just count and divide

Use Independence

- When stuck, exploit independence and collect the statistics you can…
- There are a large 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 of a VP
  - 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

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

is estimated as

$$P_H(VBD|VP, dumped) \times P_L(STOP|VP, VBD, dumped)$$
$$\times P_R(NP(sacks,NNS)|VP, VBD, dumped)$$
$$\times P_R(PP(into,P)|VP, VBD, dumped)$$
$$\times P_R(STOP|VP, VBD, dumped)$$

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

Last Point

- Statistical parsers are getting quite good, but its still quite silly to expect them to come up with the correct parse given only syntactic information.
- But its not so crazy to think that they can come up with the right parse among the top-N parses.
- Lots of current work on
  - Re-ranking to make the top-N list even better
- What's the problem with this argument?

Finally

- In case someone hasn't pointed this out yet, the lexicalization stuff is a thinly veiled attempt to incorporate semantics into the syntactic parsing process…
  - Duhh..,. Picking the right parse requires the use of semantics.
  - Which we'll get to real soon now.