Natural Language Processing

Lecture 14—10/20/2015
Jim Martin

Today

- Finish CKY (Chapter 13)
- Start Statistical Parsing (Chapter 14)

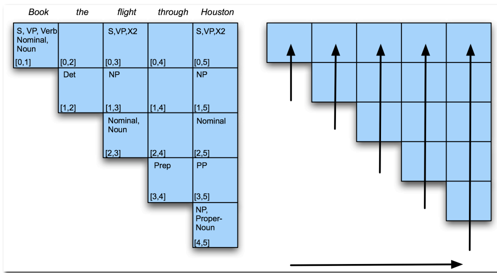
CKY Algorithm

function CKY-PARSE(*words, grammar*) **returns** *table*

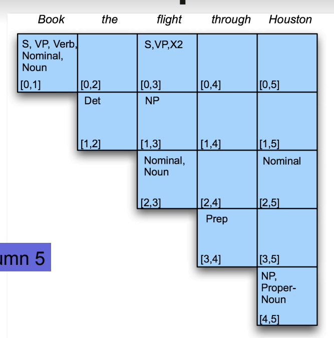
for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do** — Looping over the columns
  $table[j-1, j] \leftarrow \{A \mid A \rightarrow words[j] \in grammar\}$ — Filling the bottom cell
  for $i \leftarrow$ **from** $j-2$ **downto** 0 **do** — Filling row i in column j
    for $k \leftarrow i+1$ **to** $j-1$ **do** — Looping over the possible split locations between i and j.
      $table[i,j] \leftarrow table[i,j] \cup$
        $\{A \mid A \rightarrow BC \in grammar,$
          $B \in table[i,k],$
          $C \in table[k,j]\}$

Check the grammar for rules that link the constituents in [i,k] with those in [k,j]. For each rule found store the LHS of the rule in cell [i,j].

Example

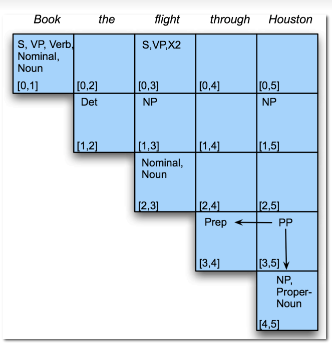


Example

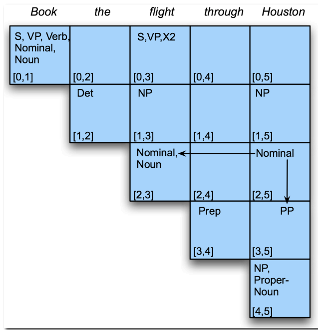


Filling column 5

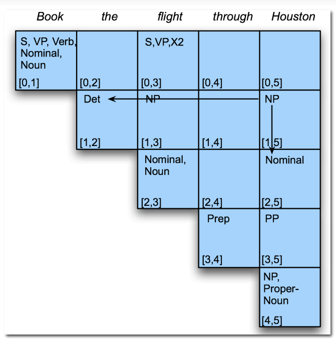
Example



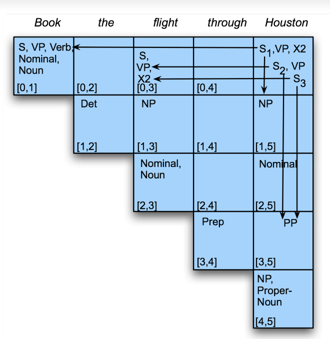
Example



Example

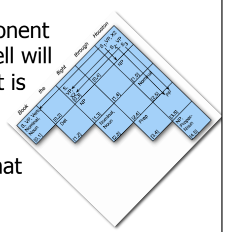


Example

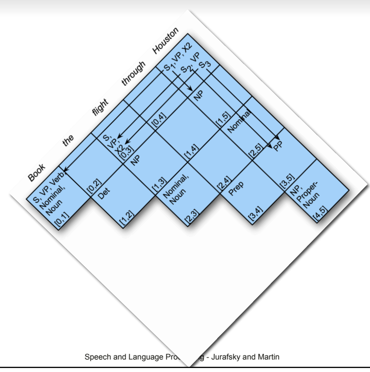


Note

- An alternative is to fill a diagonal at a time.
  - That still satisfies our requirement that the component parts of each constituent/cell will already be available when it is filled in.
  - Filling a diagonal at a time corresponds naturally to what task?



One last thing



Getting the Best Parse

- CKY ends with a table packed with all the possible parses (anchored by an S in [0,n])
- It can not select the "right" parse out of all the possible parses
  - Even if we knew what "right means"
- If we interpret "right" to mean "most probable" parse" we get our old friend
  - Argmax P(Parse|Words)

Probabilistic CFGs

1 The framework (Model)
- How to define the probability of a parse tree

2 Training the model (Learning)
- How to acquire estimates for the probabilities specified by the model

3 Parsing with probabilities (Decoding)
- Given an input sentence and a model how can we efficiently find the best (or *N* best) tree(s) for that input

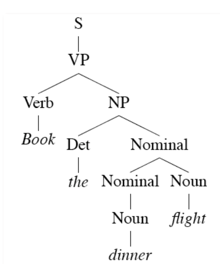
Simple Probability Model

- 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.

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

Example

- How many "rules" are in this derivation?



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.

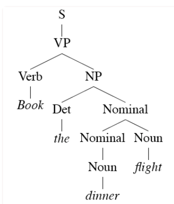
Parsing (Decoding)

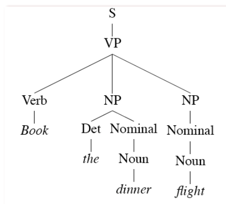
- So to get the best (most probable) parse for a given input

1. Enumerate all the trees for a sentence
2. Assign a probability to each using the model
3. Return the best (argmax)

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

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

Dynamic Programming

- Of course, as with normal parsing we don’t really want to do it that way...
- Instead, we need to exploit dynamic programming
  - For the parsing (as with CKY)
  - And for computing the probabilities and returning the best parse (as with Viterbi and HMMs)

Probabilistic CKY

- Alter CKY so that the probabilities of constituents are stored in the table as they are derived
  - Probability of a new constituent *A* derived from the rule *A → B C* :
    - P(*A → B C* | *A*) * P(*B*) * P(*C*)
    - Where P(B) and P(C) are already in the table given the way that CKY operates
    - What we store is the MAX probability over all the A rules for a given cell [i,j]

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 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

- 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 the exactly correct parse?

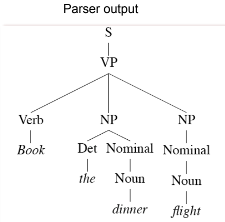
Evaluation

- 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

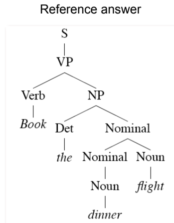
Example

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

Parser output



Reference answer



Evaluation

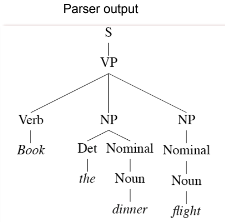
- Precision
  - What fraction of the sub-trees in our parse match corresponding sub-trees in the reference answer
    - How much of what we're producing is right?
- Recall
  - What fraction of the sub-trees in the reference answer did we actually get?
    - How much of what we should have gotten did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

Example

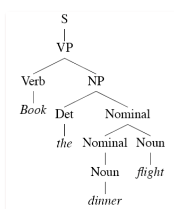
- Consider...
  - *Book the dinner flight*

Precision: 6/10
Recall: 6/9

Parser output



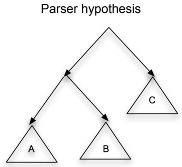
Reference answer



Evaluation

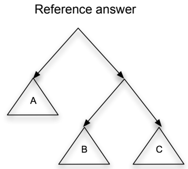
- Crossing brackets

Parser hypothesis



((A B) C)

Reference answer

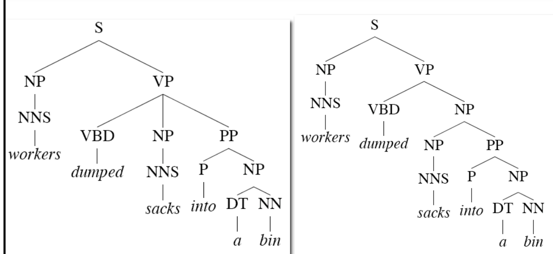


(A (B C))

Sources of Problems

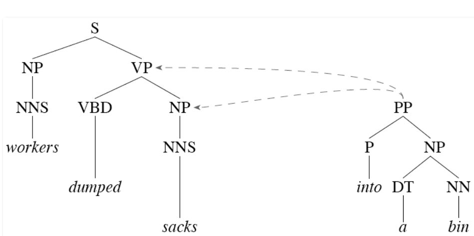
- Attachment ambiguities
  - PP attachment
  - Coordination problems

PP Attachment



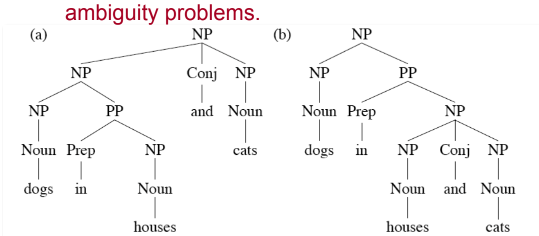
PP Attachment

- 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.