自然语言处理

week-5

凌震华

2024年3月28日

- Probabilistic CFGs
- Structural Dependencies
- Parser Evaluation

Why Statistical Parsing

- Solve problems of syntactic ambiguity
  - Parsing algorithms can identify it, but cannot resolve it
- Model human parsing which is known to be probabilistic
- Extend parsing to poorly understood languages for which data is available
- Efficient parsing for sub-languages restricted to specific domains

Probabilistic CFGs

- The probabilistic model
  - Assigning probabilities to parse trees
- Getting the probabilities for the model
- Parsing with probabilities
  - Slight modification to dynamic programming approach
  - Task is to find the max probability tree for an input

Probability Model

- Attach probabilities to grammar rules
- The expansions for a given non-terminal sum to 1

VP -> Verb	.55
VP -> Verb NP	.40
VP -> Verb NP NP	.05

  – Read this as P(Specific rule | LHS)
    - P(VP-> Verb|VP) = .55

Probability Model

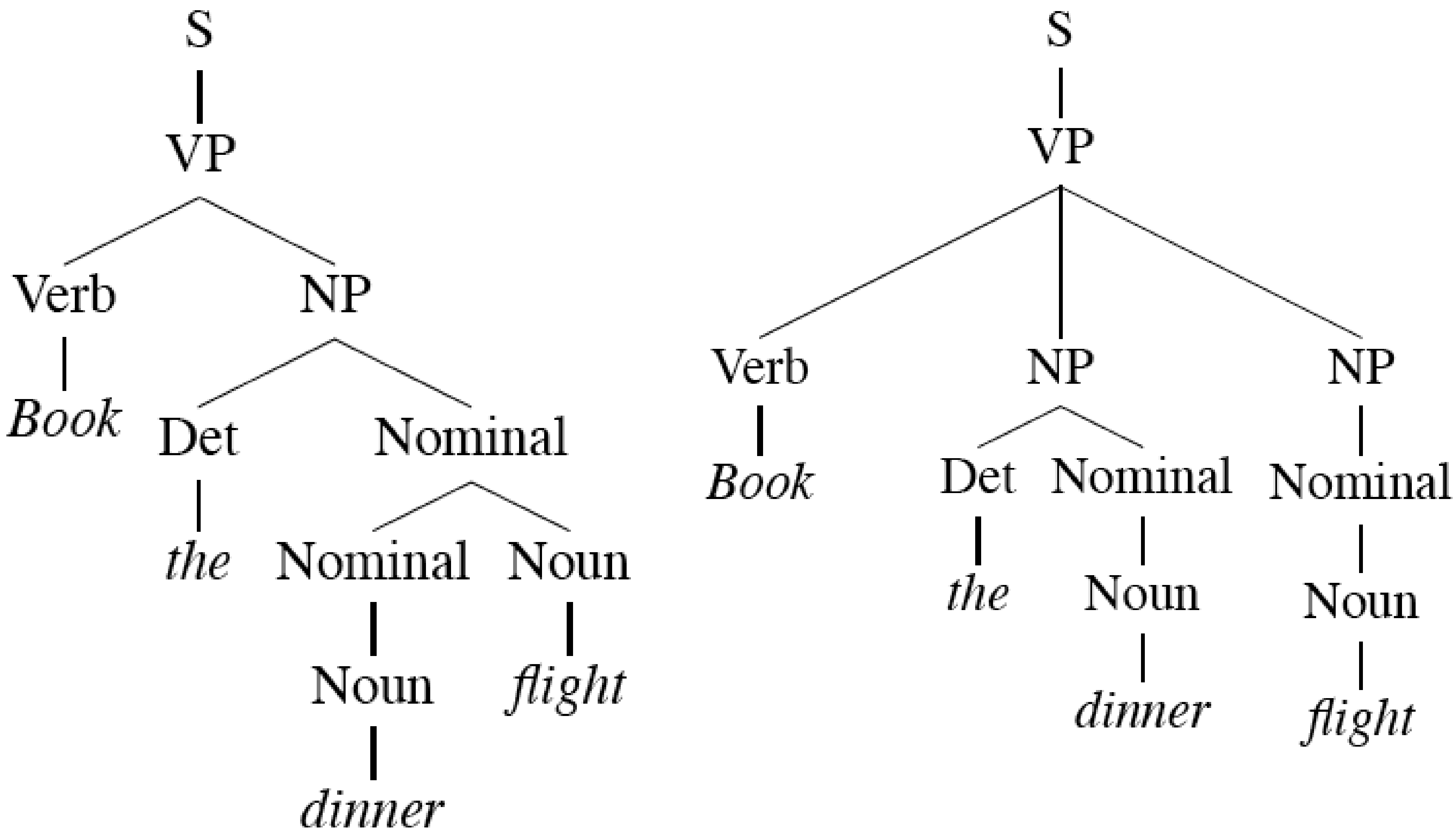
- A derivation (tree) consists of the bag of grammar rules that are in the tree
- The probability of a tree is just the product of the probabilities of the rules in the derivation.

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

Probability Model

- The probability of a word sequence (sentence) is the probability of its tree in the unambiguous case.
- It’s the sum of the probabilities of the trees in the ambiguous case.
- Since we can use the probability of the tree(s) as a proxy for the probability of the sentence…
  – PCFGs give us an alternative to N-Gram models as a kind of language model.

Example


S
VP
Verb
NP
Book
Det
Nominal
the
Nominal
Noun
Noun
flight
dinner
S
VP
Verb
NP
NP
Book
Det
Nominal
Nominal
the
Noun
Noun
dinner
flight

Rule Probabilities

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

$2.2 * 10^{-6}$ $6.1 * 10^{-7}$

Getting the Probabilities

- From an annotated database (a treebank)
  - So 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 VPs overall.

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

Getting the Probabilities

- If we don’t have a treebank, but we do have a grammar can we get reasonable probabilities?
- Yes. Use a prob parser to parse a large corpus and then get the counts as above.
  - In the unambiguous case we’re fine
  - In ambiguous cases, weight the counts of the rules by the probabilities of the trees they occur in.
  - Where do those probabilities come from?
  - Make them up. And then re-estimate them.

Assumptions

- We're assuming that there is a grammar to be used to parse with.
- We're assuming the existence of a large robust dictionary with parts of speech
- We're assuming the ability to parse (i.e. a parser)
- Given all that... we can parse probabilistically

Typical Approach

- Use CKY as the backbone of the algorithm
- Assign probabilities to constituents as they are completed and placed in the table
- Use the max probability for each constituent going up

Clarifying last point…

- Say we’ re talking about a final part of a parse
  - $S\text{->}{}_0NP_iVP_j$

  The probability of this S is…

  P(S->NP VP)*P(NP)*P(VP)

  The green stuff is already known if we're using some kind of sensible DP approach.

Max

- I said the P(NP) is known.
- What if there are multiple NPs for the span of text in question (0 to i)?
- Take the max

CKY

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

  for j ← from 1 to LENGTH(words) do
     table[j − 1, j] ← {A | A → words[j] ∈ grammar }
     for i ← from j − 2 downto 0 do
        for k ← i + 1 to j − 1 do
           table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                             B ∈ table[i,k],
                             C ∈ table[k,j] }
```

Prob 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 rules in the derivation…
  - Doesn't take into account where in the derivation a rule is used (structural issue)
  - Doesn't use the words in any real way
  - Doesn't really work
    - Most probable parse isn't usually the right one (the one in the treebank test set).

- Probabilistic CFGs
- Structural Dependencies
- Parser Evaluation

Structural dependencies

- Strength of CFG -- rules are "context-free" – becomes a weakness
- Cannot model context-sensitive probabilities
- See section 14.4.1
  – NP that is a subject is more likely to be a pronoun
    - NP → PRP (.91 in subject pos, .34 in object pos)
    - Context-free, it resolves to .25 probability

One solution

- Parent annotation 父结点标注 (section 14.5)

- Split non-terminals to add context
- Parent info added to every non-terminal
  - Subject NP becomes NP^S; Object NP becomes NP^VP

- Split pre-terminal POS nodes
  - Note that in parsing, POS categories are fixed and come from “outside” the grammar

Parent Annotation

- Increases size/complexity of grammar
- Requires more data than is typically available
- Split-and-merge algorithm now available for automatically adjusting the size
- Gives the best published result on penn-treebank.

Solution for using word info

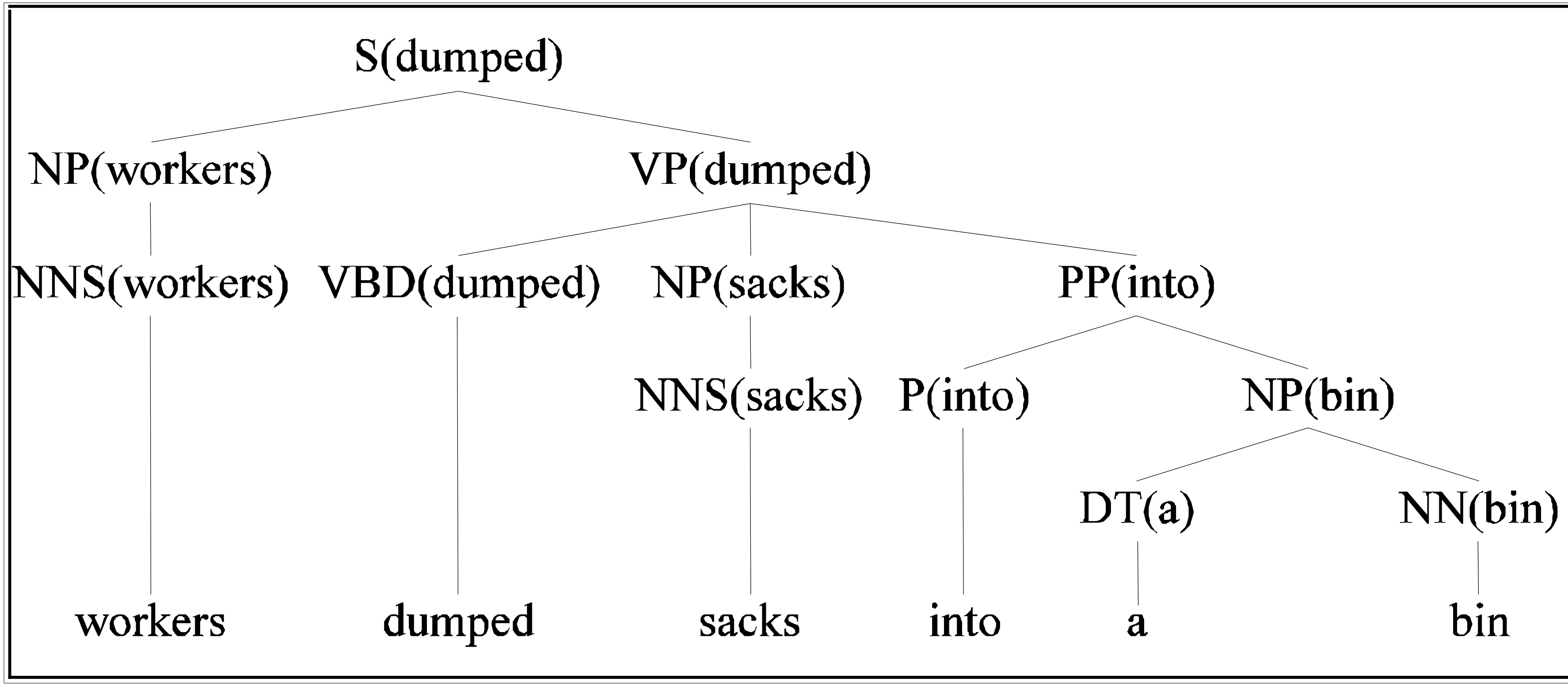
- Add lexical dependencies to the scheme…
  - Integrate the preferences of particular words into the probabilities in the derivation
  - i.e. Condition the rule probabilities on the actual words

Heads

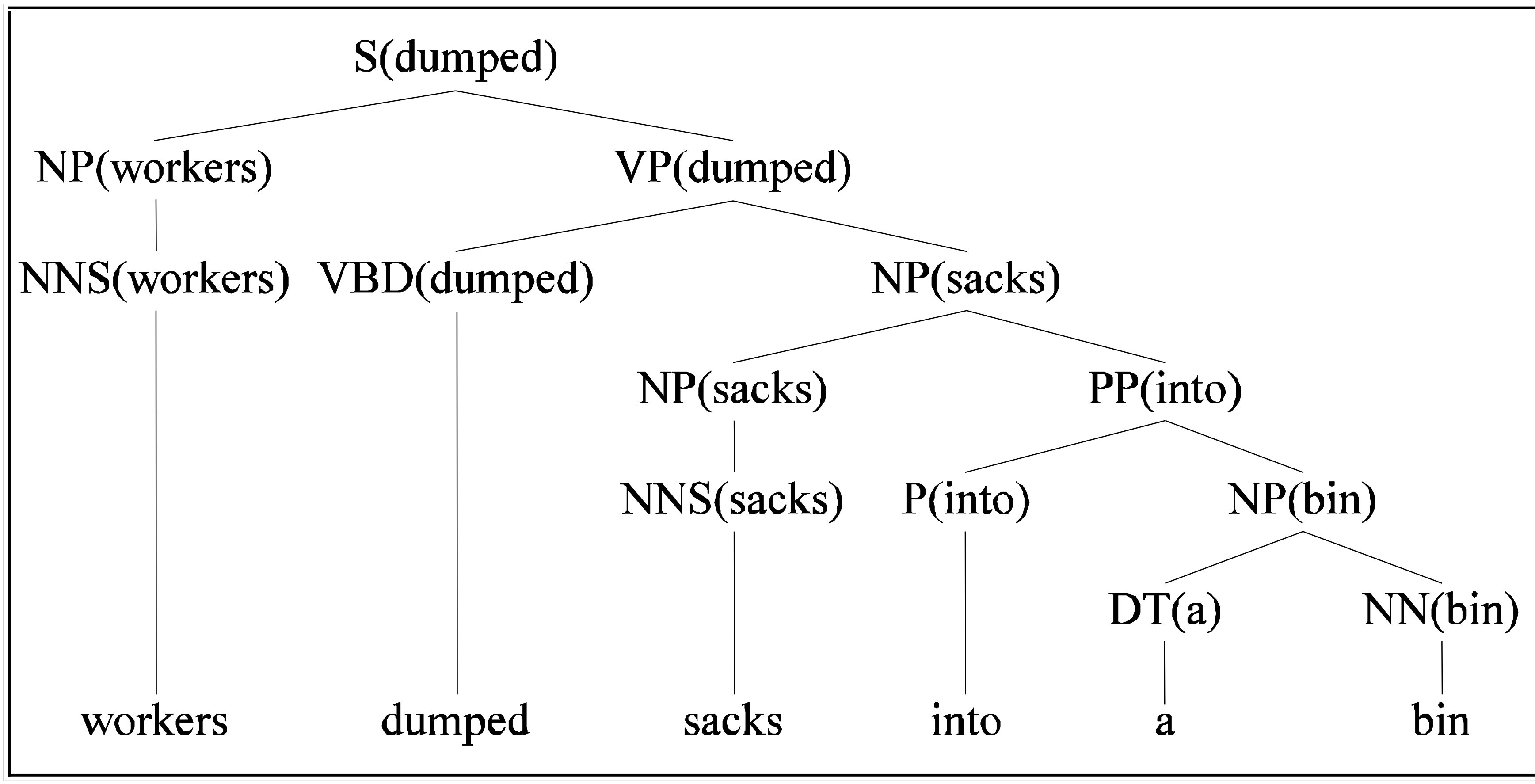
- To do that we' re going to make use of the notion of the head 中心词 of a phrase
  - The head of an NP is its noun
  - The head of a VP is its verb
  - The head of a PP is its preposition

  (It's really more complicated than that but this will do.)

Example (sensible parse)



Example (incorrect parse)



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
  - VP(dumped)-> V(dumped) NP(sacks)PP(in)
  - P(r|VP ^ dumped is the verb ^ sacks is the head of the NP ^ in is the head of the PP)
  - Not likely to have significant counts in any treebank
  - Make independence assumptions to bread down each rule

Subcategorization

- Condition particular VP rules on their head… so

  r: VP -> V NP PP P(r|VP)

  Becomes

  P(r | VP ^ dumped)

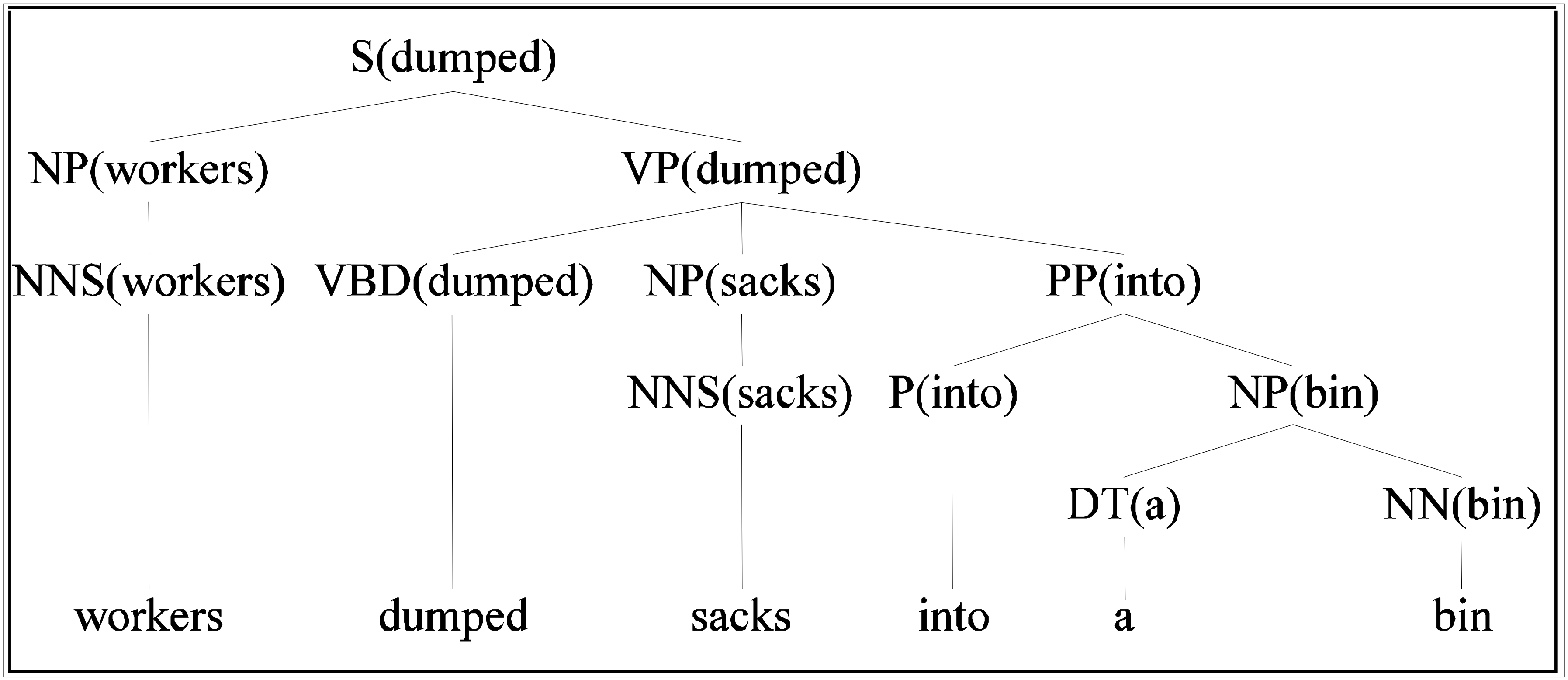
  What's the count?

  How many times was this rule used with dump, divided by the number of VPs that dump appears in total

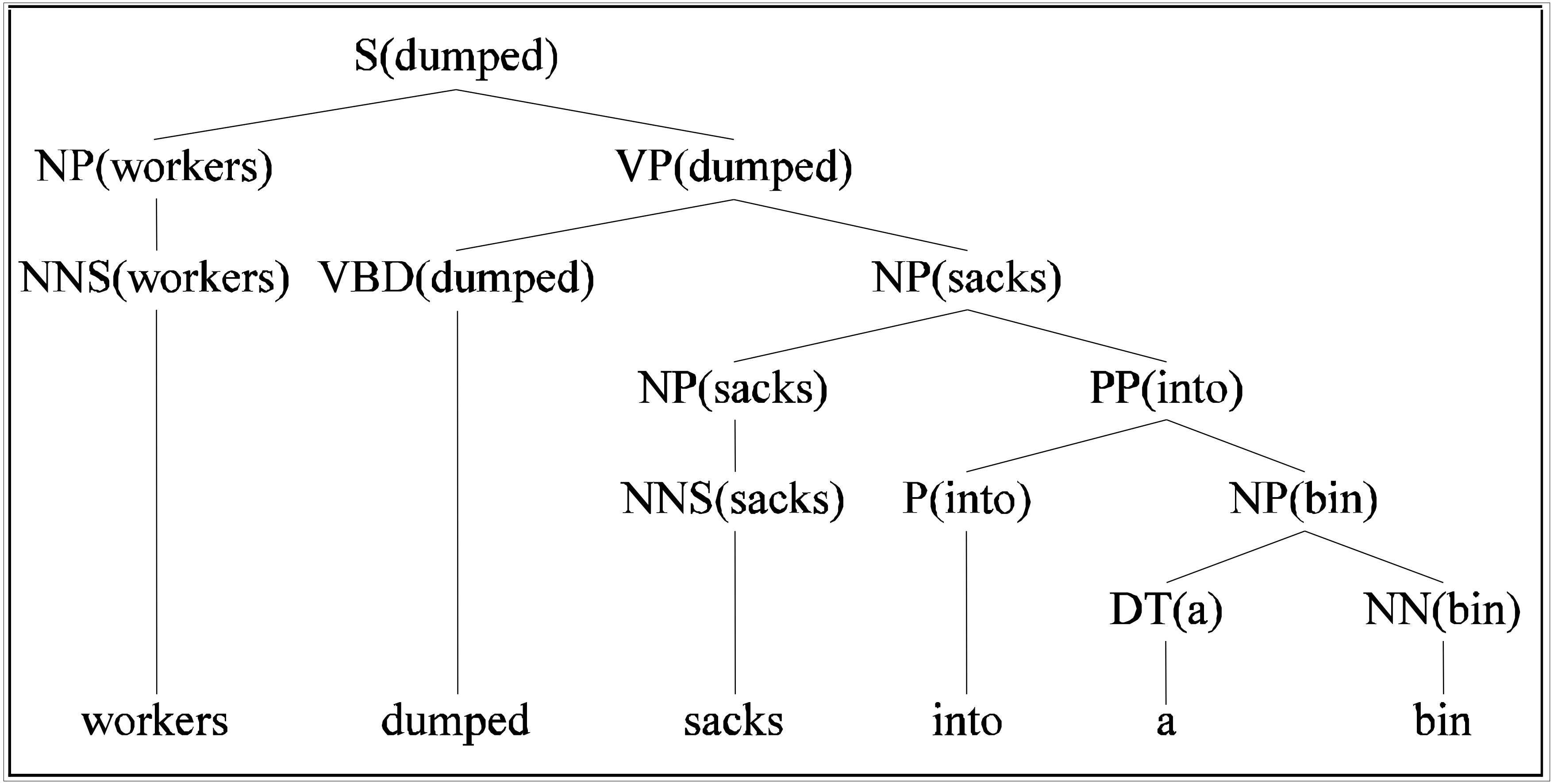
Preferences

- Subcat captures the affinity between VP heads (verbs) and the VP rules they go with.
- What about the affinity between VP heads and the heads of the other daughters of the VP
- Back to our examples…

Example (right)



Example (wrong)



Preferences

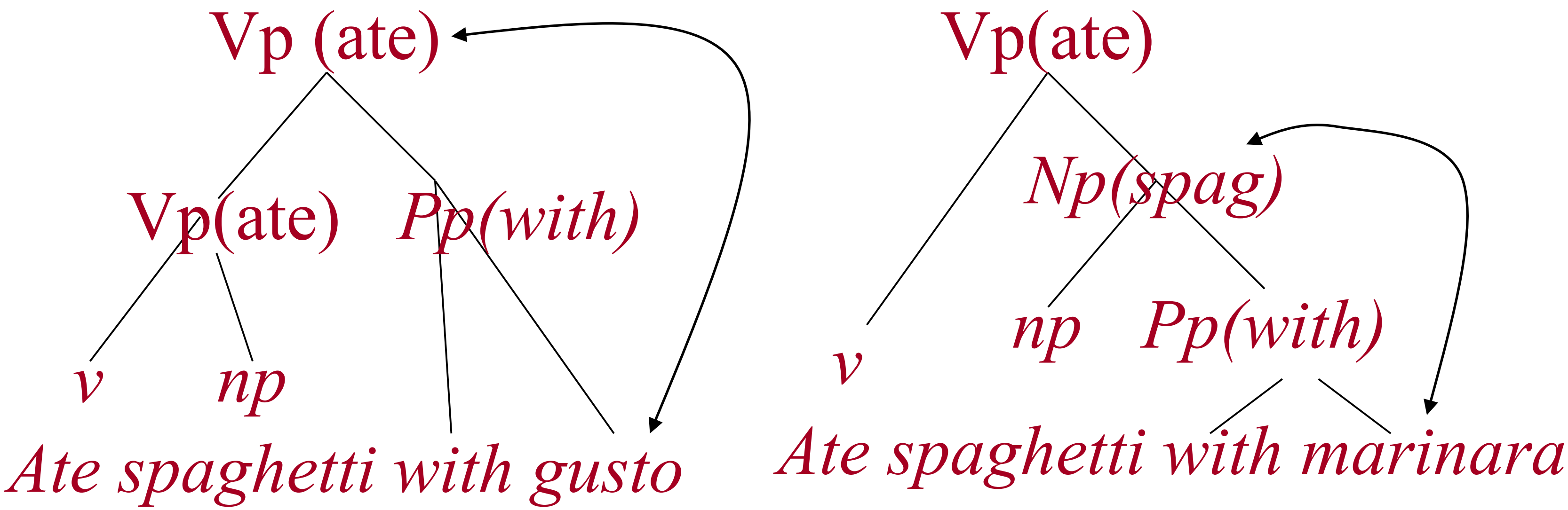
- The issue here is the attachment of the PP. So the affinities we care about are the ones between dumped and into vs. sacks and into.
- So count the places where dumped is the head of a constituent that has a PP daughter with into as its head and normalize
- v.s. the situation where sacks is a constituent with into as the head of a PP daughter.

Preferences

- Consider the VPs
  - Ate spaghetti with gusto
  - Ate spaghetti with marinara
- The affinity of gusto for eat is much larger than its affinity for spaghetti
- On the other hand, the affinity of marinara for spaghetti is much higher than its affinity for ate

Preferences

- Note the relationship here is more distant and doesn’t involve a headword since gusto and marinara aren’t the heads of the PPs.



- Probabilistic CFGs
- Structural Dependencies
- Parser Evaluation

Parser Evaluation

- See 14.7 (many parts are a repeat of 13.5.3 because chunking is partial parsing)

- Constituent-level evaluation
  – Sentence-level would be too coarse

- Cross brackets
  – number of brackets in the candidate parse which cross brackets in the treebank parse

Constituent Evaluation - Recall

$$\frac{\#\text{correct nodes in candidate parse}}{\#\text{nodes in treebank parse}}$$

Correct node = node in candidate parse which:

– has same node label as in treebank

– spans the same words as in treebank

Constituent Eval - Precision

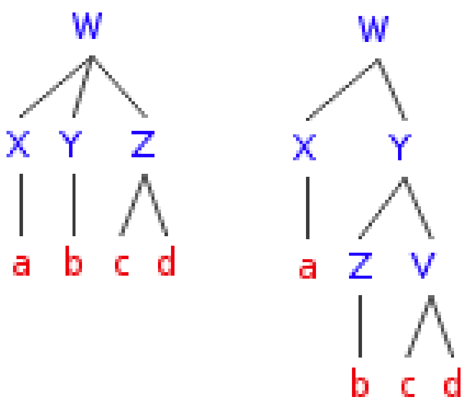
$$\frac{\#\,\text{correct nodes in candidate parse}}{\#\,\text{nodes in candidate parse}}$$

Correct node = node in candidate parse which:

– has same node label as in treebank

– spans the same words as in treebank

Example



Candidate Parse

Golden Standard

Candidate	gold
X:a	X:a
Y:b	Z:b
Z:cd	V:cd
--	Y:bcd
W:abcd	W:abcd

Labeled Recall = 2/5; Labeled Precision = 2/4

Cross brackets

- Number of brackets in candidate parse that cross brackets in the treebank parse
  - e.g. treebank has ((X Y) Z) and candidate has (X (Y Z))
- Unlike precision/recall, this is an objective function to minimize

Drawbacks of PARSEVAL

- Rewards shallow/safe analyses better than those that make more claims but a few mistakes.
- Some "single" errors can hurt the score repeatedly, for example a single misplaced node may trigger multiple crossing brackets and incorrect nodes.
- Weights all nodes evenly, rather than making crucial semantical relations more important.

Metrics for Dependency Parsing

- Head Attachment Score (percent of nodes which are correctly attached to their parent)
- Label Precision (percent of nodes whose dependency labeled is predicted correctly)
- Labeled Attachment Score (percent of node for which both of the above are true)
- Branch Precision (percent of the Paths (from root to leaf) that are being classified correctly)
- Correct trees precision (percent of the sentences from the eval corpus which have been parsed flawlessly)