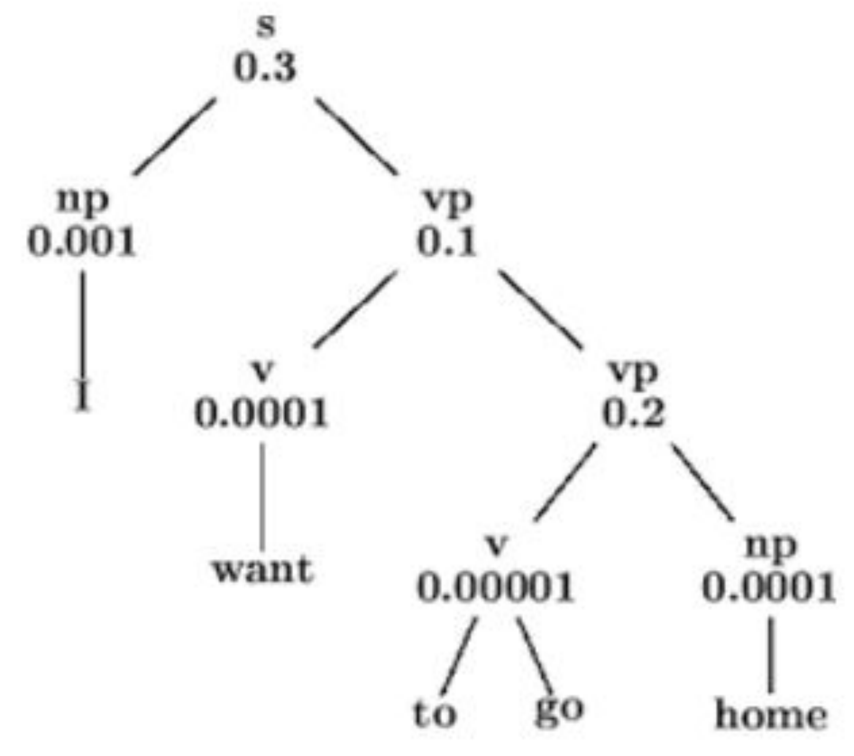
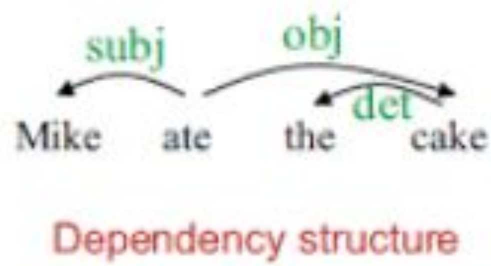
Topic 9 (Pt 2) :
Statistical Parsing





Statistical Parsing

- Uses a probabilistic model of syntax in order to **assign probabilities to each parse** tree
- Provides principled approach to resolving syntactic ambiguity
- Allows supervised learning of parsers from tree-banks of parse trees provided by human linguists
- Also allows unsupervised learning of parsers from unannotated text, but the accuracy of such parsers has been limited

Probabilistic Context Free Grammar (PCFG)

- A probabilistic version of a CFG where each production rule is assigned a probability
- Probabilities of all productions rewriting a given non-terminal must add to 1, defining a distribution for each non-terminal
- The generations of string is now probabilistic where production probabilities are used to non-deterministically select a production for rewriting a given non-terminal

PCFG for ATIS English
(Air Travel Information System Corpus)

Grammar	Prob	
S → NP VP	0.8	+ 1.0
S → Aux NP VP	0.1	
S → VP	0.1	
NP → Pronoun	0.2	+ 1.0
NP → Proper-Noun	0.2	
NP → Det Nominal	0.6	
Nominal → Noun	0.3	+ 1.0
Nominal → Nominal Noun	0.2	
Nominal → Nominal PP	0.5	
VP → Verb	0.2	+ 1.0
VP → Verb NP	0.5	
VP → VP PP	0.3	
PP → Prep NP	1.0	

Lexicon

Det → the | a | that | this
0.6 0.2 0.1 0.1

Noun → book | flight | meal | money
0.1 0.5 0.2 0.2

Verb → book | include | prefer
0.5 0.2 0.3

Pronoun → I | he | she | me
0.5 0.1 0.1 0.3

Proper-Noun → Houston | NWA
0.8 0.2

Aux → does
1.0

Prep → from | to | on | near | through
0.25 0.25 0.1 0.2 0.2

Treebanks

- Treebanks are corpora in which each sentence has been paired with a parse tree (with the assumption that it is the right one).
- Example: Penn Treebank
- Most well known is the *Wall Street Journal* with 1 million words between 1987-1989.

Penn Treebank Example

- Sentence :

“We would have to wait until we have collected on those assets”, he said.

```
( (S (‘‘ ‘‘)
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those)(NNS assets)))))))))))))
    (, ,) (’’ ’’)
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Probabilistic Parsed Trees with NLTK Chart Parser

Sentence: "Book the flight through Houston"

- Define the grammar with probabilities (PCFG):

```
>>> grammar = nltk.PCFG.fromstring("""
S -> NP VP [0.8] | Aux NP VP [0.1] | VP[0.1]
NP -> Det Nominal [0.6]| Pronoun [0.2] | Proper-Noun [0.2]
Nominal -> Noun [0.3] | Nominal Noun [0.2] | Nominal PP [0.5]
VP -> Verb [0.3] | Verb NP [0.2] | VP PP [0.5]
PP -> Prep NP [1.0]
Det -> 'the' [0.6] | 'a' [0.2] | 'that' [0.1] | 'is' [0.1]
Verb -> 'book' [0.5] | 'include' [0.2] | 'prefer' [0.3]
Noun -> 'book' [0.1] | 'flight' [0.5] | 'meal' [0.2] | 'money' [0.2]
Proper-Noun -> 'Houston' [0.8] | 'NWA' [0.2]
Prep -> 'from' [0.25] | 'to' [0.25] | 'near' [0.1] | 'through' [0.2] |'on'
[0.2]
""")
```

Probabilistic Parsed Trees with NLTK Chart Parser

- Re-display grammar with production rules

```
>> print(grammar)
```

- Parse grammar using pchart (chart with prob.)

```
>> from nltk.parse import pchart
>> parser = pchart.InsideChartParser(grammar)
```

- Define sentence

```
>> sent = "book the flight through Houston"
```

- Generate all possible trees based on sentence

```
>> trees = parser.parse(sent.split())
```

- Print all possible trees

```
>> for t in trees:
   print(t)
```

PCFG Grammar & Bracketed Trees

```
Grammar with 31 productions (start state = S)
    S -> NP VP [0.8]
    S -> Aux NP VP [0.1]
    S -> VP [0.1]
    NP -> Det Nominal [0.6]
    NP -> Pronoun [0.2]
    NP -> Proper-Noun [0.2]
    Nominal -> Noun [0.3]
    Nominal -> Nominal Noun [0.2]
    Nominal -> Nominal PP [0.5]
    VP -> Verb [0.3]
    VP -> Verb NP [0.2]
    VP -> VP PP [0.5]
    PP -> Prep NP [1.0]
    Det -> 'the' [0.6]
    Det -> 'a' [0.2]
    Det -> 'that' [0.1]
    Det -> 'is' [0.1]
    Verb -> 'book' [0.5]
    Verb -> 'include' [0.2]
    Verb -> 'prefer' [0.3]
    Noun -> 'book' [0.1]
    Noun -> 'flight' [0.5]
    Noun -> 'meal' [0.2]
    Noun -> 'money' [0.2]
    Proper-Noun -> 'Houston' [0.8]
    Proper-Noun -> 'NWA' [0.2]
    Prep -> 'from' [0.25]
    Prep -> 'to' [0.25]
    Prep -> 'near' [0.1]
    Prep -> 'through' [0.2]
    Prep -> 'on' [0.2]
>>>
```

```
(S
  (VP
    (Verb book)
    (NP
      (Det the)
      (Nominal
        (Nominal (Noun flight))
        (PP (Prep through) (NP (Proper-Noun Houston))))))) (p=8.64e-06)
(S
  (VP
    (VP (Verb book) (NP (Det the) (Nominal (Noun flight))))
    (PP (Prep through) (NP (Proper-Noun Houston))))) (p=8.64e-06)
```

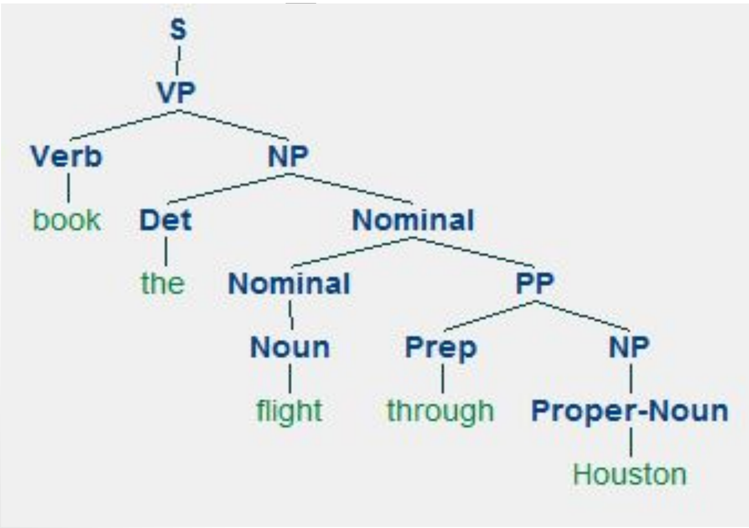
Draw PCFG Parsed Trees

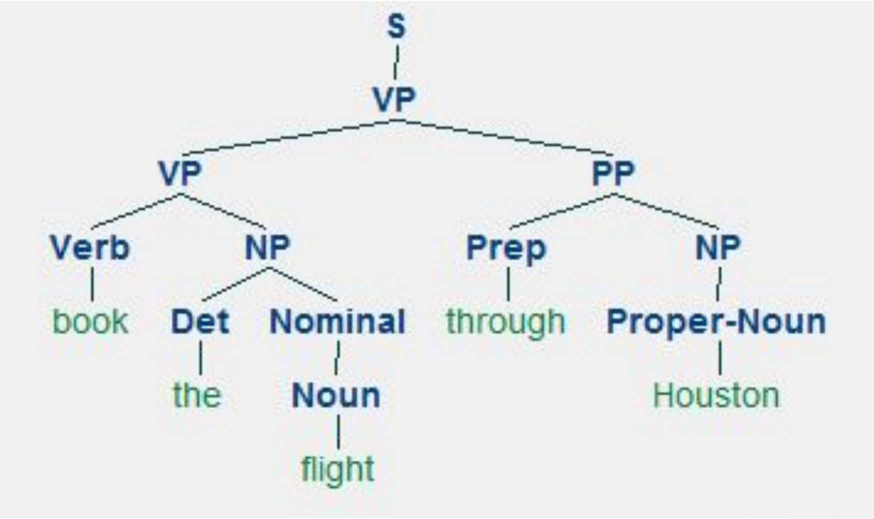
- Import draw_tree package

```
>> from nltk.draw.tree import draw_trees
>> from nltk.parse import pchart
```

- Draw all possible trees

```
>> for t in trees:
   draw_trees(t)
```



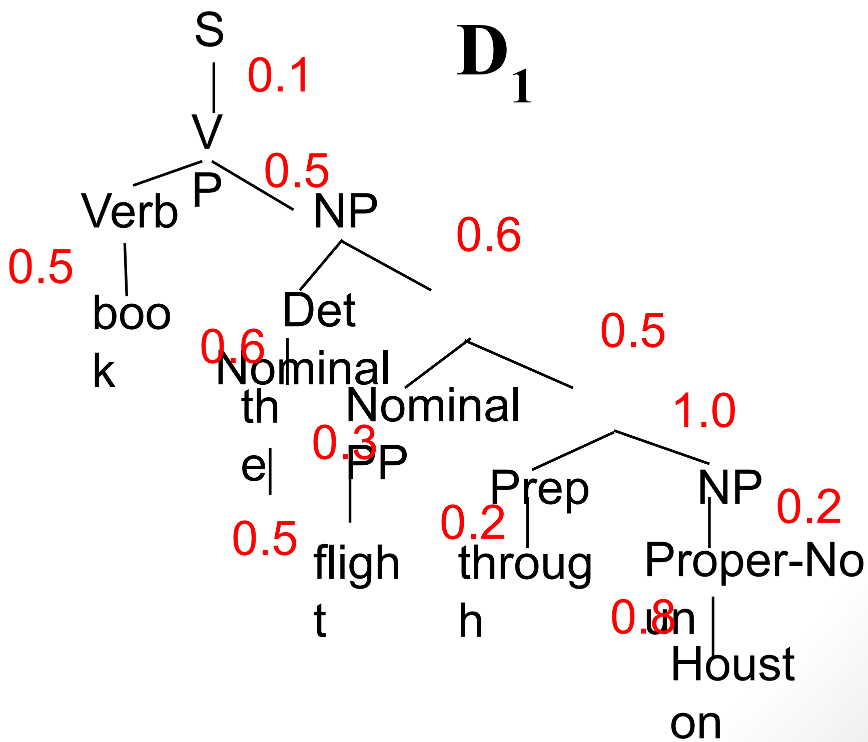


Sentence Probability

- Assume that the productions for each node are chosen independently
- Probability of derivation is the product of the probabilities of its productions

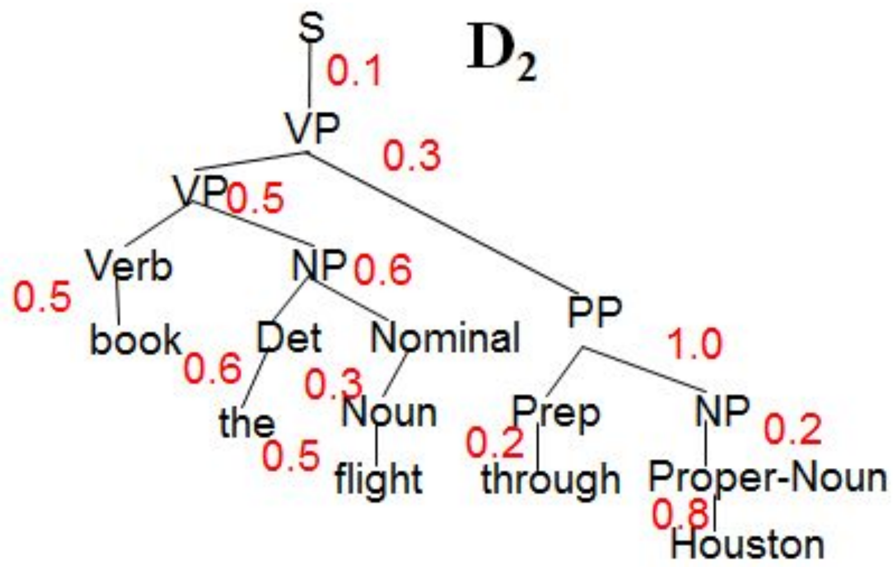
$$P(D_1) = 0.1 \times 0.5 \times 0.5 \times 0.6 \times 0.6 \times 0.5 \times 0.3 \times 1.0 \times 0.2 \times 0.2 \times 0.5 \times 0.8$$

$$= \mathbf{0.0000216}$$



Syntactic Disambiguation

- Resolve ambiguity by picking most probable parse tree



$$P(D_2) = 0.1 \times 0.3 \times 0.5 \times 0.6 \times 0.5 \times 0.6 \times 0.3 \times 1.0 \times 0.5 \times 0.2 \times 0.2 \times 0.8$$

$$= \mathbf{0.00001296}$$

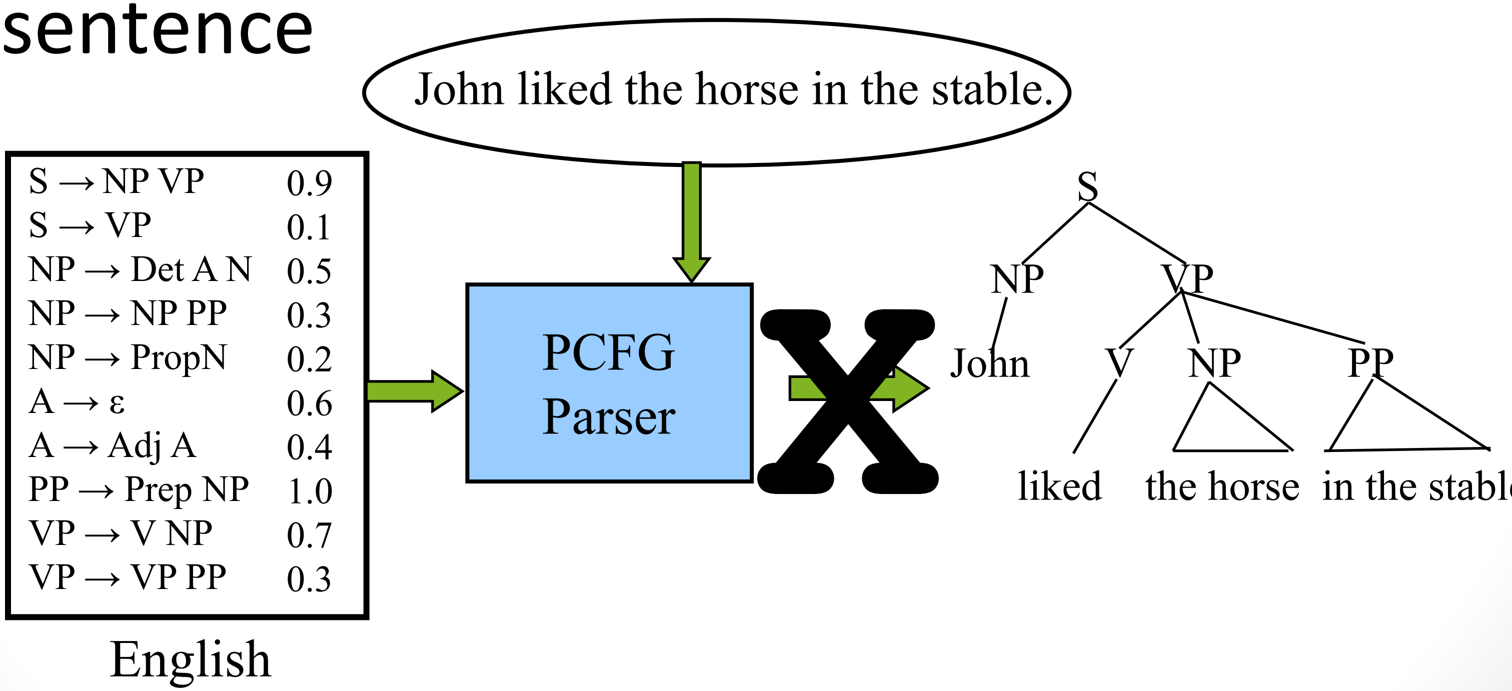
Sentence Probability

- Probability of a sentence is the **sum of the probabilities of all of its derivations**

P("book the flight through Houston") =
$P(D_1) + P(D_2) = 0.0000216 + 0.00001296$
$= \mathbf{0.00003456}$

PCFG: Most Likely Derivation (Viterbi)

- There is an analog(referent) to the Viterbi algorithm to efficiently determine the most probable derivation (parse tree) for a sentence

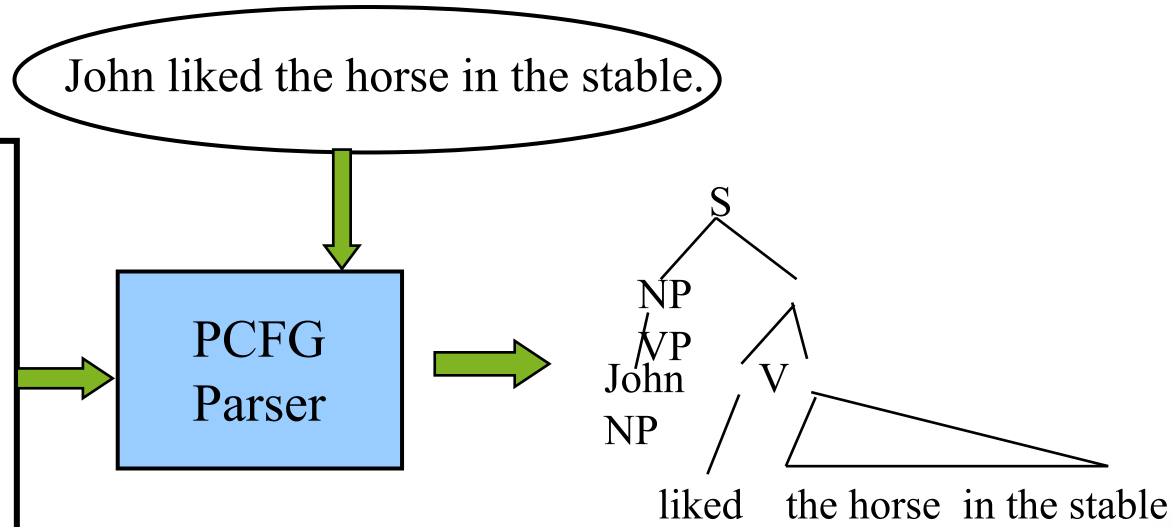


PCFG: Most Likely Derivation

- There is an analog(referent) to the Viterbi algorithm to efficiently determine the most probable derivation (parse tree) for a sentence

$S \rightarrow NP\ VP$	0.9
$S \rightarrow VP$	0.1
$NP \rightarrow Det\ A\ N$	0.5
$NP \rightarrow NP\ PP$	0.3
$NP \rightarrow PropN$	0.2
$A \rightarrow \varepsilon$	0.6
$A \rightarrow Adj\ A$	0.4
$PP \rightarrow Prep\ NP$	1.0
$VP \rightarrow V\ NP$	0.7
$VP \rightarrow VP\ PP$	0.3

English



Probabilistic Parsed Trees with NLTK Viterbi Parser

Sentence: "John liked the horse in the stable"

- Repeat the steps in Slide 7.
- Replace the InsideChartParser in Slide 8 with Viterbi parser

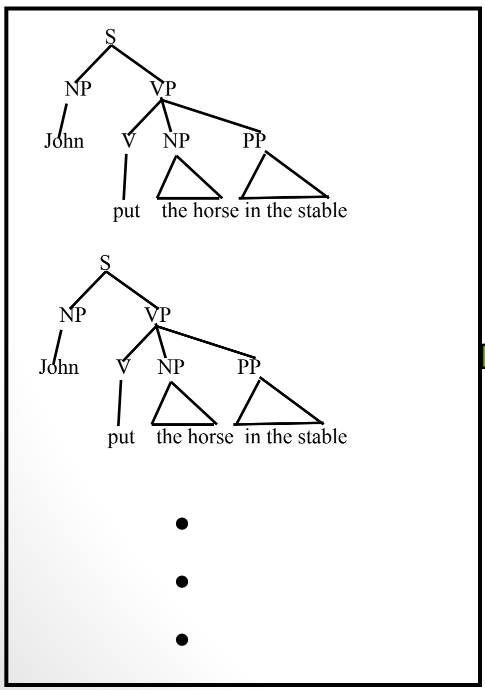
```
>> parser = nltk.ViterbiParser(grammar)
```

- Repeat the rest of the steps in Slide 7-10
- Find the probability of the most likely derivation of the parsed tree

PCFG: Supervised Training

- If parse trees are provided for training sentences, a grammar and its parameters can be can all be estimated directly from counts accumulated from the tree-bank (with appropriate smoothing).

Tree Bank



Supervised PCFG Training

S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

English

Estimating Production Probabilities

- Set of production rules can be taken directly from the set of rewrites in the treebank.
- Parameters can be directly estimated from frequency counts in the treebank.

$$P(\alpha \to \beta \mid \alpha) = \frac{\text{count}(\alpha \to \beta)}{\sum_{\gamma} \text{count}(\alpha \to \gamma)} = \frac{\text{count}(\alpha \to \beta)}{\text{count}(\alpha)}$$

PCFG: Maximum Likelihood Training

- Given a set of sentences, induce a grammar that maximizes the probability that this data was generated from this grammar.
- Assume the number of non-terminals in the grammar is specified.
- Only need to have an unannotated set of sequences generated from the model. Does not need correct parse trees for these sentences. In this sense, it is unsupervised.

PCFG: Maximum Likelihood Training

Training Sentences

John ate the apple
A cat bit Mary
Mary hit the cat
John gave Mary the cat.

•
•
•

PCFG Training

S → NP VP	0.9
S → VP	0.1
NP → Det A N	0.5
NP → NP PP	0.3
NP → PropN	0.2
A → ε	0.6
A → Adj A	0.4
PP → Prep NP	1.0
VP → V NP	0.7
VP → VP PP	0.3

English

Head Words

- Syntactic phrases usually have a word in them that is most "important" to the phrase.
- Linguists have defined the concept of a lexical **head** of a phrase.
- Simple rules can identify the head of any phrase by percolating head words up the parse tree.
  - Head of a VP is the main verb
  - Head of an NP is the main noun
  - Head of a PP is the preposition
  - Head of a sentence is the head of its VP

Parser Evaluation

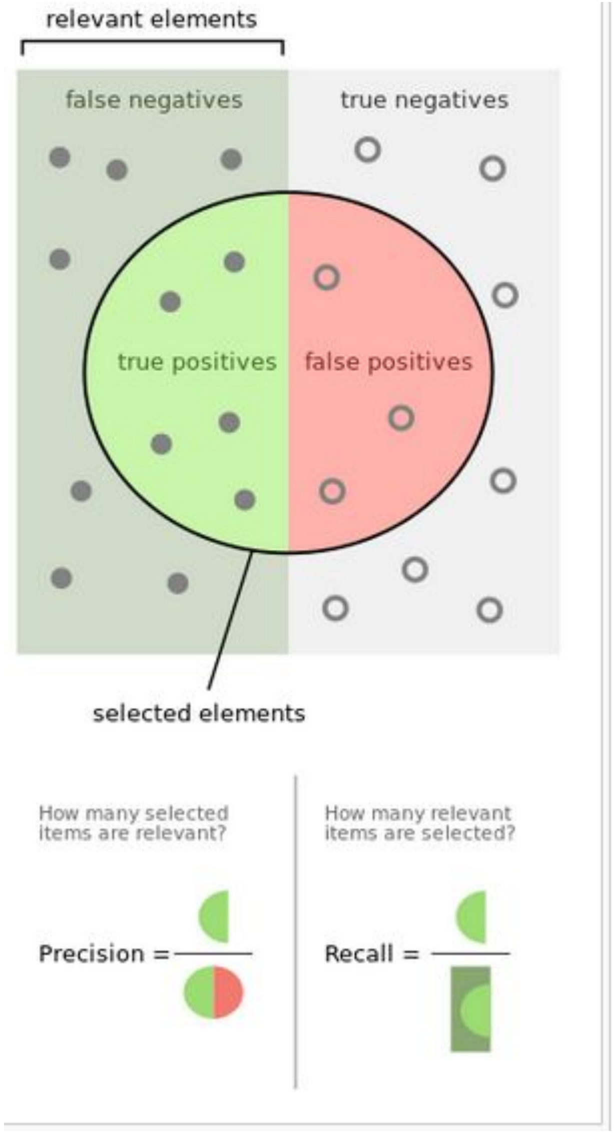
- PARSEVAL metrics measure the fraction of the constituents that match between the computed and human parse trees. If *P* is the system's parse tree and *T* is the human parse tree (the "gold standard"):
  - ***Recall*** = $\dfrac{(\#\text{ correct constituents in } P)}{(\#\text{ constituents in } T)}$
  - ***Precision*** = $\dfrac{(\#\text{ correct constituents in } P)}{(\#\text{ constituents in } P)}$

Parser Evaluation

- ***Labeled Precision*** and ***labeled recall*** require getting the non-terminal label on the constituent node correct to count as correct.
- ***F_1*** (F-measure) is the harmonic mean of precision and recall.

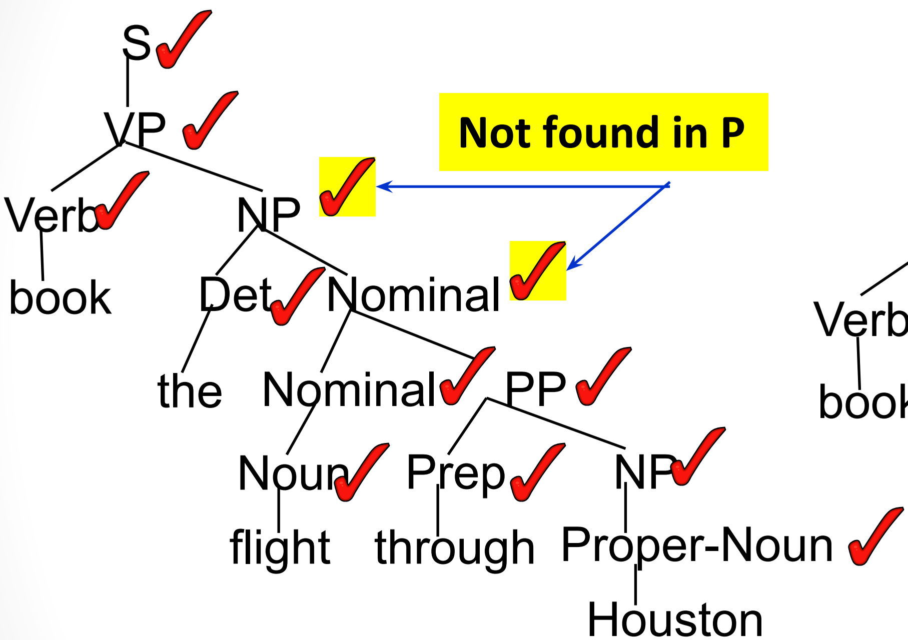
$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Precision vs Recall

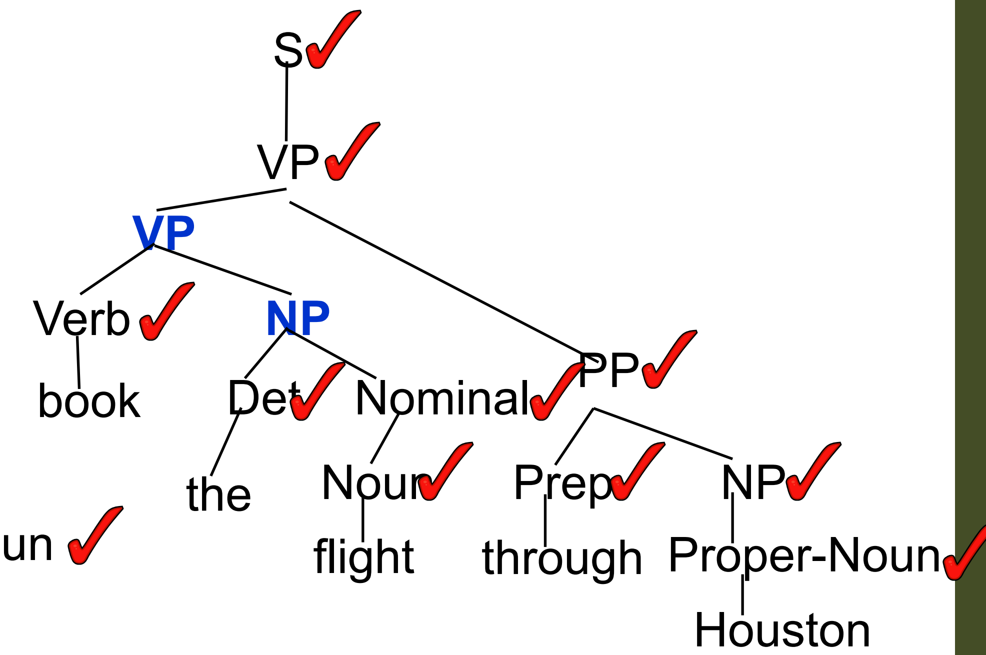


Parser Evaluation Example

Correct Tree T



Computed Tree P



Constituents: 12

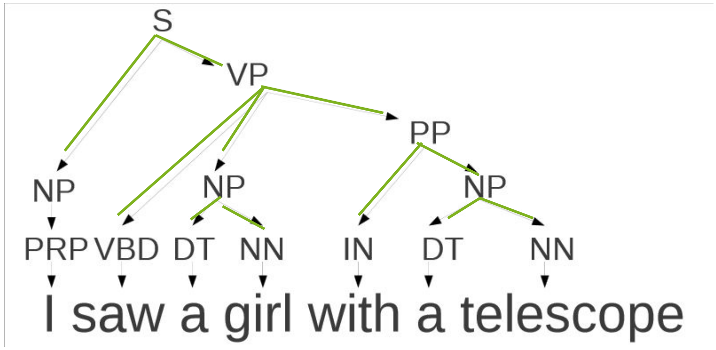
Constituents: 12

 # Correct Constituents: 10

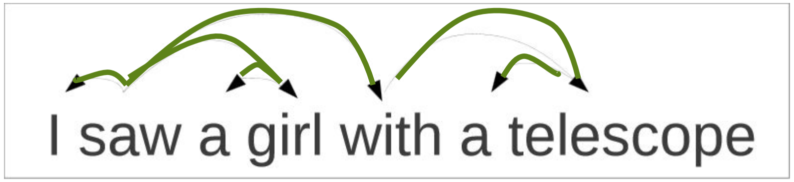
Recall = 10/12= 83.3% **Precision = 10/12=83.3%** **F_1 = 83.3%**

Syntactic Parsing (Revisit)

- Two types of parsing:
  - Phrase structure:
    - focuses on identifying phrases and their recursive structure



  - **Dependency**
    - focuses on relations between words



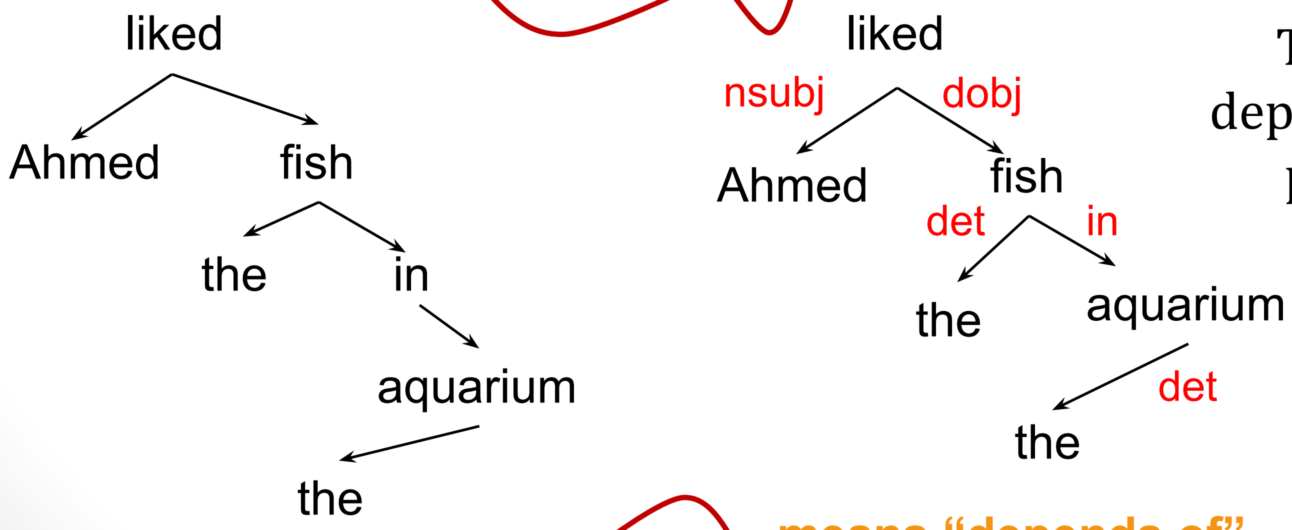
Dependency Grammar

- Dependency grammar assumes that syntactic structure consists only of dependencies
- DG is often used for free word order languages
- Dependencies are (labeled) asymetrical binary relations between two lexical items (words).
- Dependencies form a graph over the words in a sentence.
- This graph is connected (every word is a node) and (typically) acyclic (no loops)
- Dependency trees do not specify the order of words in a sentence

Dependency Grammar

- An alternative to Phrase-Structure Grammar is to define a parse as a directed graph between the words of a sentence representing ***dependencies*** between the words.

"Ahmed liked the fish in the aquarium"

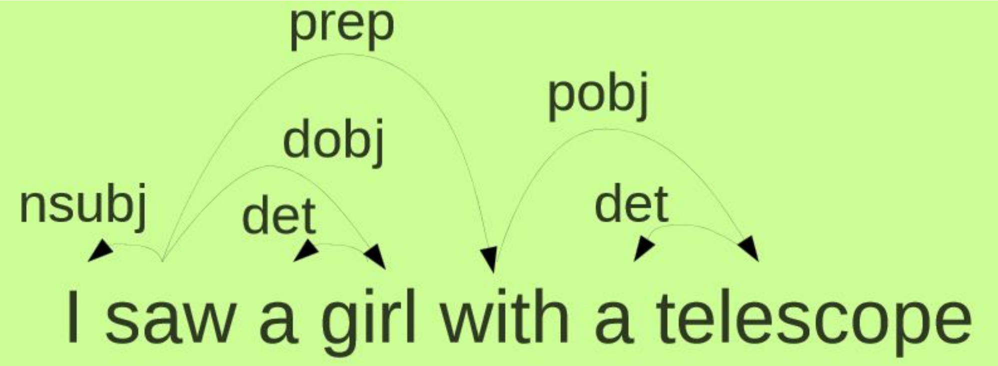


Typed dependency parse

means "depends of"

Dependencies

- Typed : Labels indicate relationship between words



- Untyped: Only which words depend

From Parse Tree to Dependency Grammar

- Convert a phrase structure parse to a dependency tree by **making the head of each non-head child of a node depend on the head of the head child**.