CS447: Natural Language Processing
http://courses.engr.illinois.edu/cs447

Lecture 16: Statistical Parsing with PCFGs

Julia Hockenmaier
juliahmr@illinois.edu
3324 Siebel Center

Where we're at

Previous lecture:

Standard **CKY** (for non-probabilistic CFGs)

The standard CKY algorithm finds all possible parse trees τ for a sentence $S = w^{(1)} \ldots w^{(n)}$ under a CFG G in Chomsky Normal Form.

Today's lecture:

Probabilistic Context-Free Grammars (PCFGs)

– CFGs in which each rule is associated with a probability

CKY for PCFGs (Viterbi):

– CKY for PCFGs finds the most likely parse tree $\tau^* = \text{argmax } P(\tau \mid S)$ for the sentence S under a PCFG.

Previous Lecture: CKY for CFGs

CKY for standard CFGs

CKY is a bottom-up chart parsing algorithm that finds all possible parse trees τ for a sentence $S = w^{(1)} \ldots w^{(n)}$ under a CFG *G* in Chomsky Normal Form (CNF).

- **CNF**: *G* has two types of rules: $X \longrightarrow Y\ Z$ and $X \longrightarrow w$ (X, Y, Z are nonterminals, w is a terminal)
- CKY is a **dynamic programming** algorithm
- The **parse chart** is an n×n upper triangular matrix: Each cell chart[i][j] ($i \leq j$) stores **all subtrees** for $w^{(i)} \ldots w^{(j)}$
- Each cell chart[i][j] has at most **one entry for each nonterminal** X (and **pairs of backpointers** to each pair of (Y, Z) entry in cells chart[i][k] chart[k+1][j] from which an X can be formed
- Time Complexity: $O(n^3 |G|)$

Recap: CKY algorithm

1. Create the chart

(an *n*×*n* upper triangular matrix for an sentence with *n* words)

– Each cell chart[i][j] corresponds to the substring $w^{(i)} \ldots w^{(j)}$

2. Initialize the chart (fill the diagonal cells chart[i][i]):

For all rules X → $w^{(i)}$, add an entry X to chart[i][i]

3. Fill in the chart:

Fill in all cells chart[i][i+1], then chart[i][i+2], …, until you reach chart[1][n] (the top right corner of the chart)

– To fill chart[i][j], consider all binary splits $w^{(i)} \ldots w^{(k)} | w^{(k+1)} \ldots w^{(j)}$

– If the grammar has a rule X → YZ, chart[i][k] contains a Y and chart[k+1][j] contains a Z, add an X to chart[i][j] with two backpointers to the Y in chart[i][k] and the Z in chart[k+1][j]

4. Extract the parse trees from the S in chart[1][n].

Additional unary rules

In practice, we may allow other unary rules, e.g.

NP → Noun

(where Noun is also a nonterminal)

In that case, we apply all unary rules to the entries in chart[i][j] after we have checked all binary splits (chart[i][k], chart[k+1][j])

Unary rules are fine as long as there are no "loops" that could lead to an infinite chain of unary productions, e.g.:

X → Y and Y → **X**

or: **X** → Y and Y → Z and Z → **X**

CKY so far…

Each entry in a cell chart[i][j] is associated with a nonterminal X.

If there is a rule X → YZ in the grammar, and there is a pair of cells chart[i][k], chart[k+1][j] with a Y in chart[i][k] and a Z in chart[k+1][j],
we can add an entry X to cell chart[i][j], and associate one pair of backpointers with the X in cell chart[i][k]

Each entry might have multiple pairs of backpointers.
- When we extract the parse trees at the end,
- we can get all possible trees.
- We will need probabilities to find the single best tree!

How do you count the **number of parse trees** for a sentence?

1. For each **initial item** (e.g. $V \rightarrow$ eat): #trees = 1

$$\text{trees}(V_{V \rightarrow \text{eat}}) = 1$$

2. For each **pair of backpointers** (e.g. $VP \rightarrow V\ NP$): **multiply** #trees of children

$$\text{trees}(VP_{VP \rightarrow V\ NP}) = \text{trees}(V) \times \text{trees}(NP)$$

3. For each **list of pairs of backpointers** (e.g. $VP \rightarrow V\ NP$ and $VP \rightarrow VP\ PP$): **sum** #trees

$$\text{trees}(VP) = \text{trees}(VP_{VP \rightarrow V\ NP}) + \text{trees}(VP_{VP \rightarrow VP\ PP})$$

Exercise: CKY parser

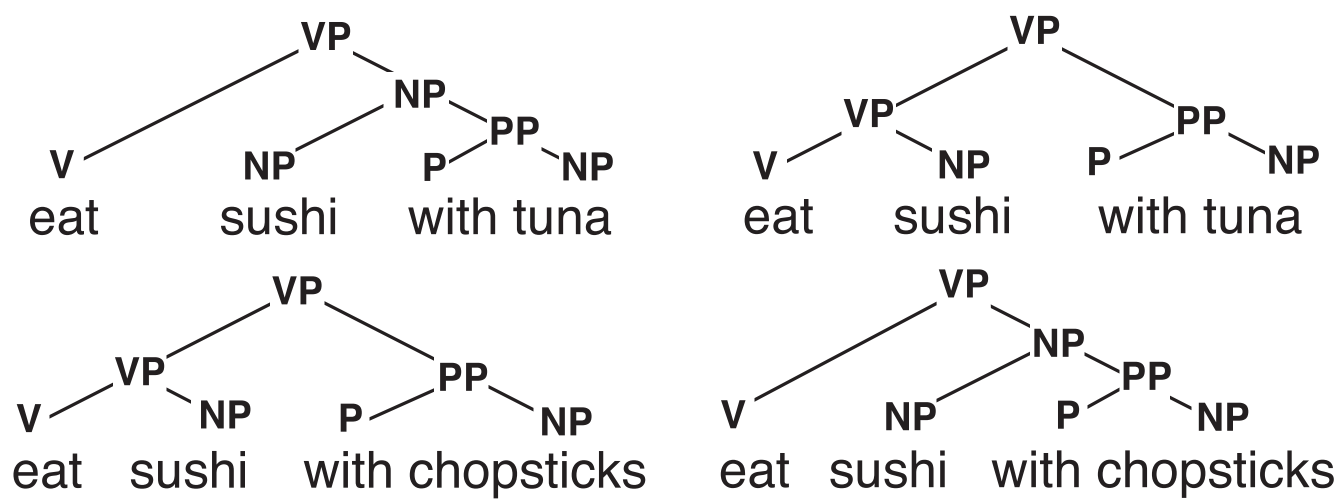
I eat sushi with chopsticks with you

$$
\begin{array}{lll}
S & \longrightarrow & NP \quad VP \\
NP & \longrightarrow & NP \quad PP \\
NP & \longrightarrow & \text{sushi} \\
NP & \longrightarrow & \text{I} \\
NP & \longrightarrow & \text{chopsticks} \\
NP & \longrightarrow & \text{you} \\
VP & \longrightarrow & VP \quad PP \\
VP & \longrightarrow & Verb \quad NP \\
Verb & \longrightarrow & \text{eat} \\
PP & \longrightarrow & Prep \quad NP \\
Prep & \longrightarrow & \text{with}
\end{array}
$$

Dealing with ambiguity: Probabilistic Context-Free Grammars (PCFGs)

Grammars are ambiguous

A grammar might generate multiple trees for a sentence:



What's the most likely parse τ for sentence S ?

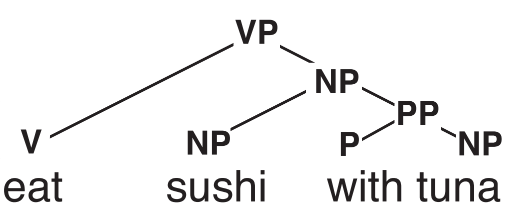
We need a model of $P(\tau \mid S)$

Computing P(τ | S)

Using Bayes' Rule:

$$
\begin{aligned}
\arg\max_{\tau} P(\tau|S) &= \arg\max_{\tau} \frac{P(\tau, S)}{P(S)} \\
&= arg \max_{\tau} P(\tau, S) \\
&= arg \max_{\tau} P(\tau) \quad \text{if} \quad \mathrm{S} = \mathrm{yield}(\tau)
\end{aligned}
$$

The **yield of a tree** is the string of terminal symbols that can be read off the leaf nodes

yield(

) = *eat sushi with tuna*

Computing P(τ)

T is the (infinite) set of all trees in the language:

$$L = \{s \in \Sigma^* \mid \exists \tau \in T : \text{yield}(\tau) = s\}$$

The set T is generated by a context-free grammar:

S $\longrightarrow$ NP VP	VP $\longrightarrow$ Verb NP	NP $\longrightarrow$ Det Noun
S $\longrightarrow$ S conj S	VP $\longrightarrow$ VP PP	NP $\longrightarrow$ NP PP
S $\longrightarrow$	VP $\longrightarrow$	NP $\longrightarrow$ .....

We need to define P(τ) such that:

$$\forall \tau \in T : \quad 0 \leq P(\tau) \leq 1$$

$$\sum_{\tau \in T} P(\tau) = 1$$

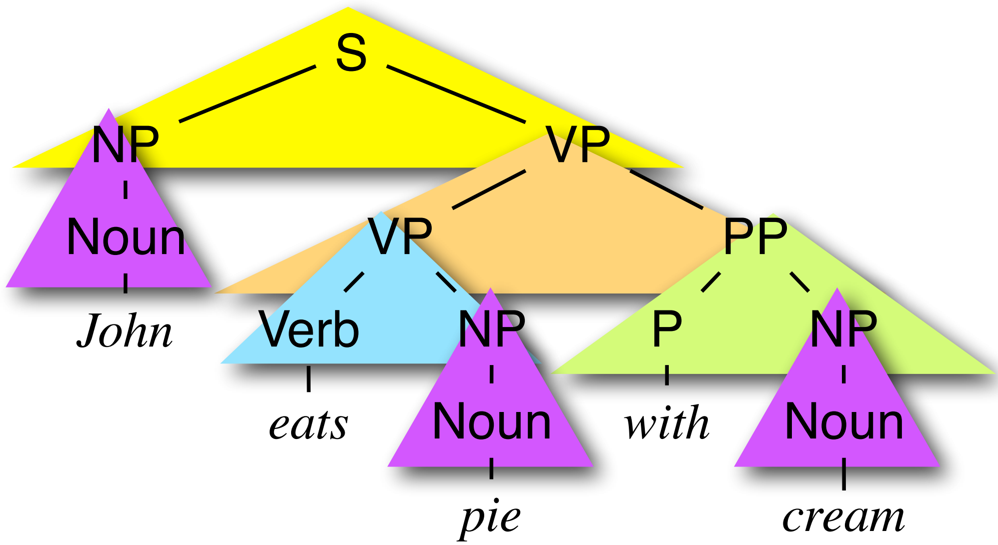
Probabilistic Context-Free Grammars

For every nonterminal X, define a probability distribution $P(X \rightarrow \alpha \mid X)$ over all rules with the same LHS symbol X:

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.4
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.2
PP	$\longrightarrow$	P NP	1.0

Computing $P(\tau)$ with a PCFG

The probability of a tree τ is the product of the probabilities of all its rules:



S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.4
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.2
PP	$\longrightarrow$	P NP	1.0

$$P(\tau) = 0.8 \times 0.3 \times 0.2 \times 1.0 \times 0.2^3$$

$$= \mathbf{0.00384}$$

Learning the parameters of a PCFG

If we have a treebank (a corpus in which each sentence is associated with a parse tree), we can just count the number of times each rule appears, e.g.:

S → NP VP . (1000)   S → S conj S . (220)
etc.

and then we divide the observed frequency of each rule X → Y Z by the sum of the frequencies of all rules with the same LHS X to turn these counts into probabilities:

S → NP VP .   (p = 1000/1220)
S → S conj S . (p = 220/1220)

More on probabilities:

Computing $P(s)$:

If $P(\tau)$ is the probability of a tree τ,
the probability of a sentence s is the sum of the probabilities of all its parse trees:

$$P(s) = \sum_{\tau:\text{yield}(\tau) = s} P(\tau)$$

How do we know that $P(L) = \sum_{\tau} P(\tau) = 1$?

If we have learned the PCFG from a corpus via MLE, this is guaranteed to be the case.
If we just set the probabilities by hand, we could run into trouble, as in the following example:

S → S S (0.9) S → w (0.1)

PCFG parsing (decoding): Probabilistic CKY

Probabilistic CKY: Viterbi

Like standard CKY, but with probabilities.

Finding the most likely tree is similar to Viterbi for HMMs:

Initialization:

- [*optional*] Every chart entry that corresponds to a **terminal** (entries w in `cell[i][i]`) has a Viterbi probability $P_{\mathrm{VIT}}(\mathrm{w}_{[i][i]}) = 1$ (*)
- Every entry for a **non-terminal** `X` in `cell[i][i]` has Viterbi probability $P_{\mathrm{VIT}}(\mathrm{X}_{[i][i]}) = \mathrm{P}(\mathrm{X} \rightarrow \mathrm{w} \mid \mathrm{X})$ [and a single backpointer to $\mathrm{w}_{[i][i]}$ (*)]

Recurrence: For every entry that corresponds to a **non-terminal** `X` in `cell[i][j]`, keep only the highest-scoring pair of backpointers to any pair of children (`Y` in `cell[i][k]` and `Z` in `cell[k+1][j]`):

$P_{\mathrm{VIT}}(\mathrm{X}_{[i][j]}) = \mathrm{argmax}_{\mathrm{Y,Z,k}}\, P_{\mathrm{VIT}}(\mathrm{Y}_{[i][k]}) \times P_{\mathrm{VIT}}(\mathrm{Z}_{[k+1][j]}) \times P(\mathrm{X} \rightarrow \mathrm{Y\ Z} \mid \mathrm{X})$

Final step: Return the Viterbi parse for the start symbol S in the top `cell[1][n]`.

*this is unnecessary for simple PCFGs, but can be helpful for more complex probability models

Probabilistic CKY

Input: POS-tagged sentence

`John_N eats_V pie_N with_P cream_N`

John	eats	pie	with	cream	
Noun NP 1.0 0.2	S $0.8 \cdot 0.2 \cdot 0.3$	S $0.8 \cdot 0.2 \cdot 0.06$		S $0.2 \cdot 0.0036 \cdot 0.8$	**John**
	Verb VP 1.0 0.3	VP $1 \cdot 0.3 \cdot 0.2$ $= 0.06$		VP $\max(1.0 \cdot 0.008 \cdot 0.3,$ $0.06 \cdot 0.2 \cdot 0.3)$	**eats**
		Noun NP 1.0 0.2		NP $0.2 \cdot 0.2 \cdot 0.2$ $= 0.008$	**pie**
			Prep 1.0	PP $1 \cdot 1 \cdot 0.2$	**with**
				Noun NP 1.0 0.2	**cream**

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	Prep NP	1.0
Prep	$\longrightarrow$	P	1.0
Noun	$\longrightarrow$	N	1.0
Verb	$\longrightarrow$	V	1.0

How do we handle flat rules?

S	$\longrightarrow$	NP VP	0.8
S	$\longrightarrow$	S conj S	0.2
NP	$\longrightarrow$	Noun	0.2
NP	$\longrightarrow$	Det Noun	0.4
NP	$\longrightarrow$	NP PP	0.2
NP	$\longrightarrow$	NP conj NP	0.2
VP	$\longrightarrow$	Verb	0.3
VP	$\longrightarrow$	Verb NP	0.3
VP	$\longrightarrow$	Verb NP NP	0.1
VP	$\longrightarrow$	VP PP	0.3
PP	$\longrightarrow$	Prep NP	1.0

S	$\longrightarrow$	**S ConjS**	**0.2**
ConjS	$\longrightarrow$	**conj S**	**1.0**

Define a new nonterminal (ConjS) and add a new rule