Introduction to NLP

Probabilistic Parsing

Dr. John McCrae
Data Science Institute, NUI Galway

Overview

Parsing with Probabilistic Context-free Grammars (PCFGs)

Probabilistic Cocke-Younger-Kasami (CYK) algorithm

Problems with and solutions for PCFGs

Comparison of LMs, HMMs and PCFGs

Summary

Overview

Parsing with Probabilistic Context-free Grammars (PCFGs)

Probabilistic Cocke-Younger-Kasami (CYK) algorithm

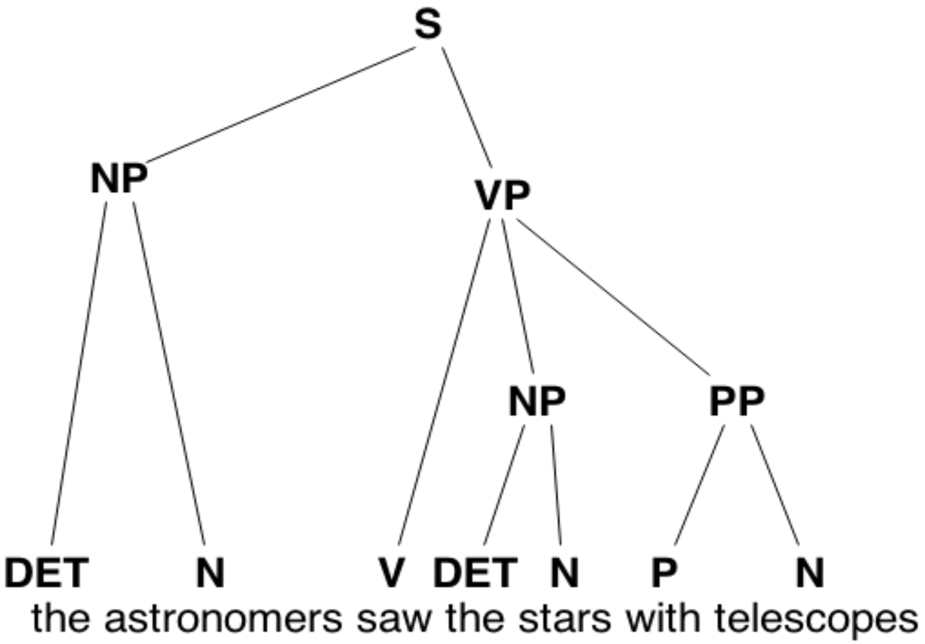
Problems with and solutions for PCFGs

Comparison of LMs, HMMs and PCFGs

Summary

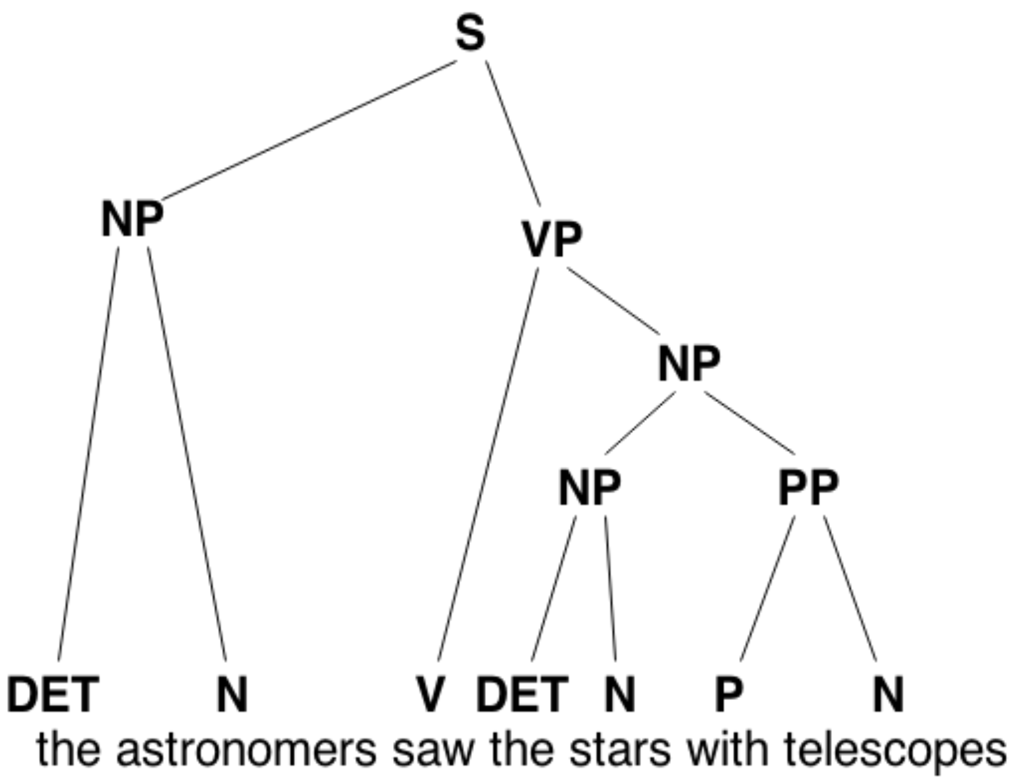
Parsing

Parsing is the problem of finding the tree structure of a sentence



Ambiguity

Parses are often ambiguous



Probabilistic Grammars: Motivation

Probabilities on parses allow us to choose the best (most-likely) parse tree

Probabilities also allow parsers to be language models

- Better handling of language structure
- Generalization over part-of-speech
- Constrain next word candidates
- More complexity

No framework has become truly standard

Context-free grammars

Recall, a context-free grammar G=(N,Σ,P,S) consists of:

A set of non-terminal symbols N

e.g., 'N', 'VP', 'S'

A set of terminal symbols Σ

e.g., 'cat', 'astronomer', 'the'

A set of productions P

e.g., 'S $\longrightarrow$ NP VP'

A start symbol

Normally 'S'

Probabilistic context-free grammar

A probabilistic context-free grammar $G=(N,\Sigma,P,S,D)$ consists of:

- N, Σ, P, S as for a CFG
- A function $D:P\rightarrow[0,1]$ which assigns a probability to each production

A PCFG is consistent iff

$$\sum_{\{\beta\,:\,A\rightarrow\beta\in P\}} D(A\rightarrow\beta) = 1 \quad \forall A\in N$$

And there are no infinite derivations for any finite string (e.g., $S\rightarrow S$)

Notation
Lowercase letters: Terminal (word), e.g. a,the
Capital letters: Non-terminal, e.g., N, V
Greek letters: Sequence of terminal and non-terminals
ϵ : Empty sequence

NUI Galway
OÉ Gaillimh

Example: PCFG

Rule	Prob	Rule	Prob
S $\longrightarrow$ NP VP	0.80	VP $\longrightarrow$ V NP	0.90
S $\longrightarrow$ Aux NP VP	0.20	VP $\longrightarrow$ V NP NP	0.10
NP $\longrightarrow$ PN	0.45	N $\longrightarrow$ flights	1.00
NP $\longrightarrow$ Nom	0.05	V $\longrightarrow$ book	1.00
NP $\longrightarrow$ Pro	0.50	Aux $\longrightarrow$ can	1.00
Nom $\longrightarrow$ N	0.95	PN $\longrightarrow$ Lufthansa	1.00
Nom $\longrightarrow$ PN Nom	0.05	Pro $\longrightarrow$ you	1.00

Calculating the probability of a parse

We wish to know how likely a parse T is, given input S

$$P(T|S) = P(T,S)/P(S)$$

Ergo, the best parse is $$T^* = \arg\max_T P(T,S)$$

And as the parse tree contains all words in the sentence

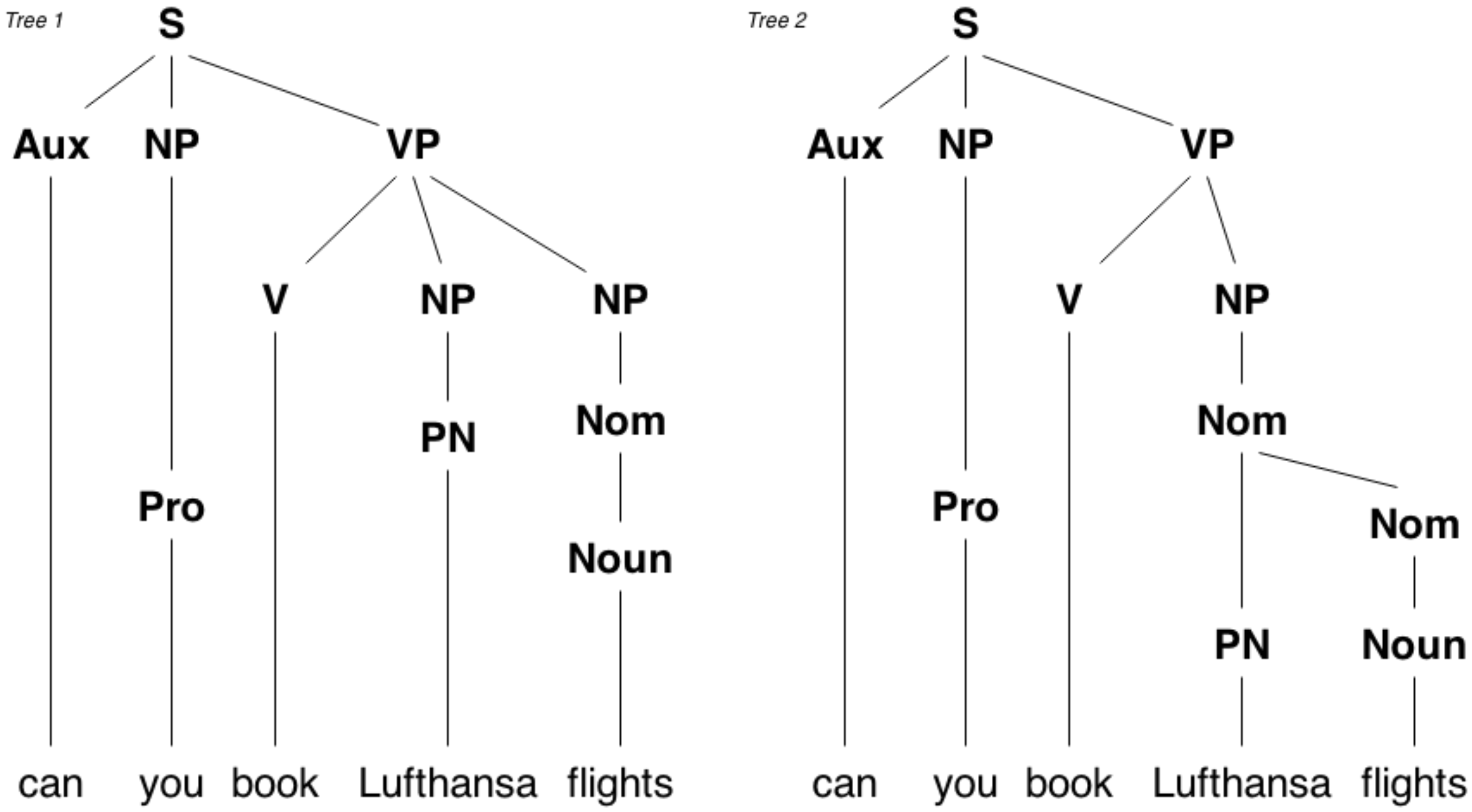
$$P(T,S) = P(T)P(S|T) = P(T)$$

Hence

$$P(T,S) = \prod_{n \in T} D(r(n))$$

Where r(n) is the rule used to generate n

Example: Parsing with PCFG



Example: Parsing with PCFG

Tree 1

$S \longrightarrow Aux\ NP\ VP$	0.20	$Aux \longrightarrow can$	1.00
$NP \longrightarrow Pro$	0.50	$Pro \longrightarrow you$	1.00
$VP \longrightarrow V\ NP\ NP$	0.10	$V \longrightarrow book$	1.00
$NP \longrightarrow Nom$	0.05	$PN \longrightarrow Lufthansa$	1.00
$NP \longrightarrow PN$	0.45	$N \longrightarrow flights$	1.00
$Nom \longrightarrow N$	0.95		

P(Tree 1) = 0.2 × 0.5 × 0.1 × 0.05 × 0.45 × 0.95 × 1.0 × 1.0 × 1.0 × 1.0 × 1.0

= 0.00021

Exercise: Tree 2

Overview

Parsing with Probabilistic Context-free Grammars (PCFGs)

Probabilistic Cocke-Younger-Kasami (CYK) algorithm

Problems with and solutions for PCFGs

Comparison of LMs, HMMs and PCFGs

Summary

Finding the best parse

It is not practical to find all parse trees (exponential number)

Recall, we can find the parse of a CFG in $\mathcal{O}(N^3)$

We can also find the best parse in $\mathcal{O}(N^3)$

Two most commonly used algorithms are easily adaptable to PCFG

- Earley algorithm
- Cocke-Younger-Kasami (CYK) algorithm

Probabilistic Cocke-Younger-Kasami Algorithm

CYK is essentially a bottom-up parser

Can be adapted to PCFGs with dynamic programming

Input:

- A PCFG
- n words $w_1,\ldots,w_n$

Data structure:

- A table $t_{i,j,a} \in \mathbb{R}$ where $1 \leq i < j \leq n$ and $a \in N$

Output:

- The best parse $T = \arg\max_T P(T)$

CYK

can	you	book	Lufthansa	flights	*j - i*
Aux: 1.0	Pro: 1.0	V: 1.0	PN: 1.0	N: 1.0	1
					2
					3
					4
					5

Rule:
Aux → can : 1.0
Pro → you : 1.0
V → book : 1.0
PN → Lufthansa : 1.0
N → flights : 1.0

CYK

can	you	book	Lufthansa	flights	*j - i*
Aux: 1.0	Pro: 1.0 NP: 0.5	V: 1.0	PN: 1.0 NP: 0.45	N: 1.0 Nom: 0.95 NP: 0.0475	1
					2
					3
					4
					5

Rule:
NP → Pro : 0.5
NP → PN : 0.45
Nom → N : 0.95
NP → Nom : 0.05

CYK

can	you	book	Lufthansa	flights	*j - i*
Aux: 1.0	Pro: 1.0 NP: 0.5	V: 1.0	PN: 1.0 NP: 0.45	N: 1.0 Nom: 0.95 NP: 0.0475	1
			VP: 0.41	Nom: 0.10 NP: 0.005	2
					3
					4
					5

Rule:
VP → V NP : 0.90
Nom → PN Nom : 0.10
NP → Nom : 0.05

CYK

can	you	book	Lufthansa	flights	*j - i*
Aux: 1.0	Pro: 1.0 NP: 0.5	V: 1.0	PN: 1.0 NP: 0.45	N: 1.0 Nom: 0.95 NP: 0.0475	1
			VP: 0.41	Nom: 0.10 NP: 0.005	2
				VP: 0.004 ~~VP: 0.002~~	3
					4
					5

Rule:
VP → V NP NP : 0.10
VP → V NP : 0.90

Choose VP → V NP as
0.9 × 1.0 × 0.005 >
0.1 × 1.0 × 0.45 ×0.0475

CYK

can	you	book	Lufthansa	flights	*j - i*
Aux: 1.0	Pro: 1.0 NP: 0.5	V: 1.0	PN: 1.0 NP: 0.45	N: 1.0 Nom: 0.95 NP: 0.0475	1
			VP: 0.41	Nom: 0.10 NP: 0.005	2
			S: 0.16	VP: 0.004	3
			S: 0.04	S: 0.0017	4
				S: 0.00043	5

Rule:
S → Aux NP VP : 0.20
S → NP VP : 0.80

CYK Algorithm

Set $t_{i,j,a}$ = -∞ for all values

For $i = 1, \ldots, n$

  For $A \rightarrow w_i \in P$

    $t_{i,i+1,A} = D(A \rightarrow w_i)$

For $k = 1, \ldots, n$; $i = 1, \ldots, n - k + 1$; $j = i + k$

  For $A \rightarrow \beta \in P$

    If β matches between i and j

      $S = D(A \rightarrow \beta) \times \prod_{i',j',A'} t_{i',j',A'}$ where $\{i',j',A\}$ are the matches

      If $s > t_{i,j,A}$

        $t_{i,j,A} = s$

Chomsky Normal Form

First, we define Chomsky Normal Form as a grammar such that every rule is of the form:

$A \longrightarrow BC$
$A \longrightarrow a$
$A \longrightarrow \epsilon$

It is known that for any CFG G there is a weakly equivalent grammar CNF(G) in Chomsky Normal Form.

Chomsky Normal Form

CYK is more efficient and easier to implement with a CNF grammar

To convert to normal formal:

Merge rules with a single non-terminal RHS

e.g., Nom→N:0.95, N→flights:1.0 to Nom→flights:0.95

Split rules with more than two non-terminal RHSs

e.g., S→Aux NP VP:0.20 to S→Aux X:0.20, X→NP VP:1.0

CYK for language modelling

The CYK algorithm can be adapted to generate language models scores

How? (Think about Forward Algorithm last week)

Supervised learning of PCFGs

If we have a gold-standard corpus of known trees we can learn from this

- Corpus of trees is called a **treebank**

PCFG probabilities can be obtained by counting

$$D(A \rightarrow \beta) = \frac{c(A \rightarrow \beta)}{c(A)}$$

Smoothing can be applied.

Unsupervised learning of PCFGs

- Suppose we have the grammar but not the probabilities
- We cannot just apply our grammar as there are many ambiguous parse
- We can apply a method called the **Inside-Outside algorithm,** analogous to the forward-backward algorithm
  - Learn probabilities by **expectation maximization.**
- Can be useful in a semi-supervised setting
  - Learning from a treebank and a larger unannotated corpus

Overview

Parsing with Probabilistic Context-free Grammars (PCFGs)

Probabilistic Cocke-Younger-Kasami (CYK) algorithm

Problems with and solutions for PCFGs

Comparison of LMs, HMMs and PCFGs

Summary

Lexical Dependencies

In a CFG the expansion of one non-terminal is independent of any other non-terminal

In Francis et. al (1999)

Subjects are

- 91% pronouns
- 9% lexical noun phrases

Direct objects are

- 34% pronouns
- 66% lexical noun phrases

Ergo, we would need to distinguish between *NP* in subject and direct object positions

Francis, H. S., Gregory, M. L., and Michaelis, L. A. (1999). Are lexical subjects deviant?. In CLS-99.

Lexical Attachment

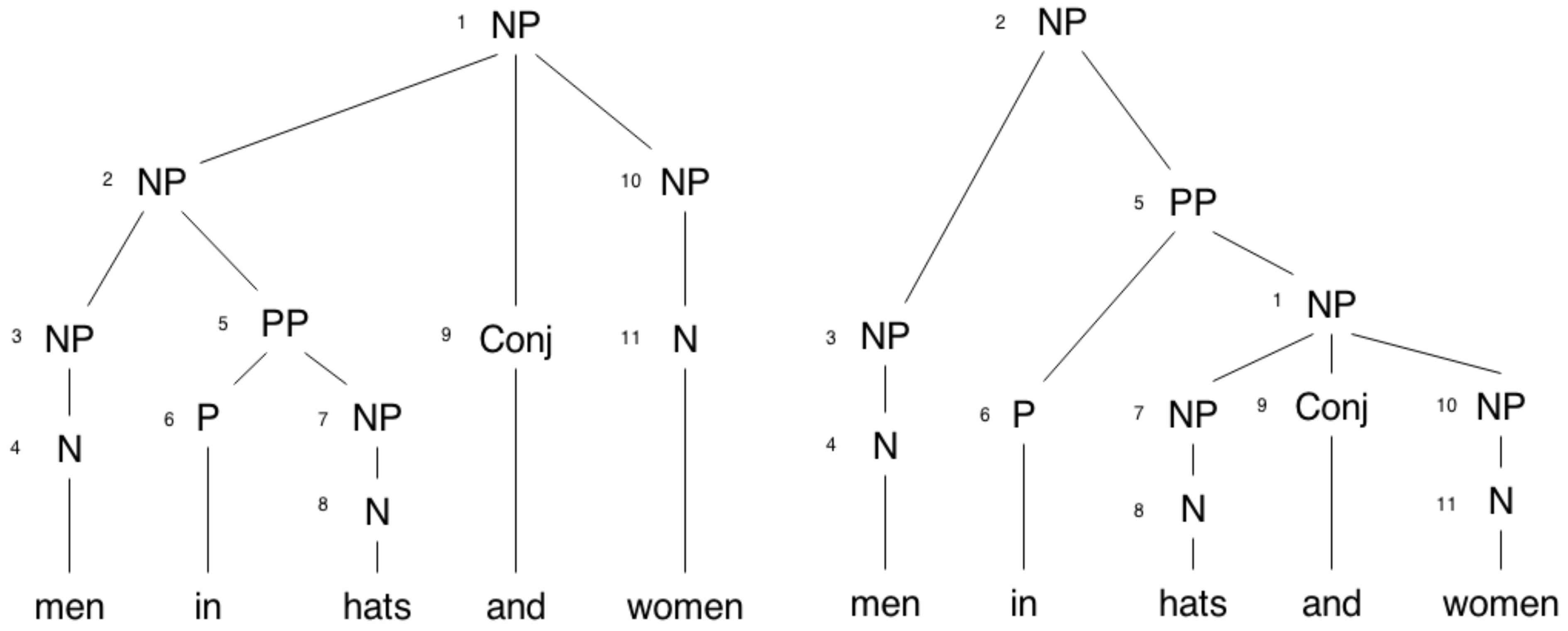
Lexical attachment is very word dependent

For example:

he hit the man with a bat
he hit the man with a hat

PCFGs cannot model lexical dependencies

Parse ambiguity



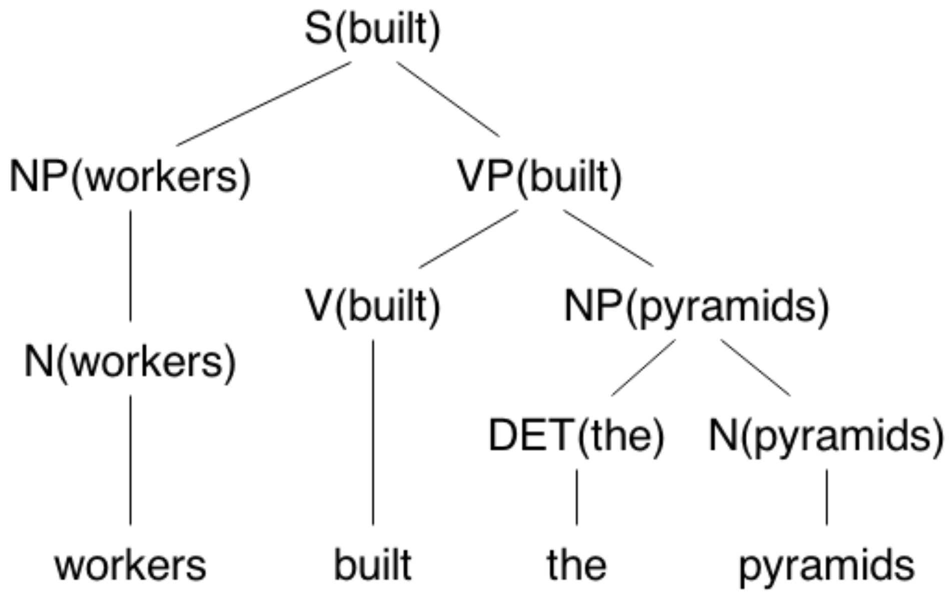
Formalisms for statistical parsing

We will look at two solutions for parsing:

- Lexicalized PCFGs
  - Further differentiate non-terminals by associating them with lexical items
  - Leads to increased sparsity
- Dependency grammars
  - Model links between individual words in the parse
  - Complex to combine with standard CFG

Lexicalized PCFGs

In a lexicalized grammar each non-terminal is further distinguished with a terminal



Heuristic rules decide which non-terminal is the head of the parent

Lexicalized PCFGs

Lexicalizing the grammar creates many more rules

- Extremely sparse

It is common to make some partial independence assumption, e.g.,

$$P(VP(\text{built}) \rightarrow V(\text{built})\ NP(\text{pyramids})) \simeq$$
$$P(VP(\text{built}) \rightarrow V(\cdot)\ NP(\cdot))P(VP(\text{built})|NP(\text{pyramids}))$$

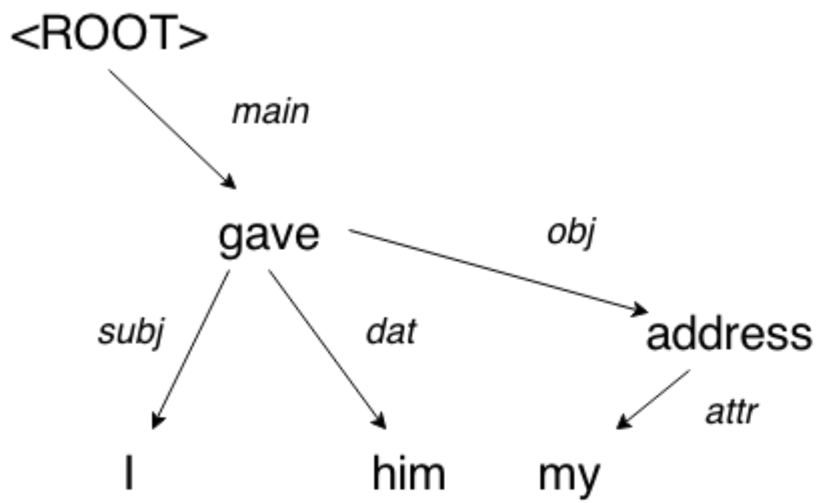
In addition, most parses still apply significant smoothing methods (see lecture 3)

Dependency grammars

Dependency grammars are different to constituency grammars as

- Links are directed and between words
- Links have labels (e.g., ‘subj’)
- They form an acyclic graph

Dependency grammars are especially good for languages with *free word order*



Evaluation of Parsing

Parsers are generally evaluated according to the PARSEVAL metrics

$$\text{recall} = \frac{\#\ \text{correct constituents in parse}}{\#\ \text{consituents in treebank}}$$

$$\text{precision} = \frac{\#\ \text{correct constituents in parse}}{\#\ \text{consituents in parse}}$$

In addition we report cross-brackets

This is where the parser outputs ((A B) C) but the candidate is (A (B C))

Modern parsers achieve 90%+ precision and recall and 1% cross-bracketing

Overview

Parsing with Probabilistic Context-free Grammars (PCFGs)

Probabilistic Cocke-Younger-Kasami (CYK) algorithm

Problems with and solutions for PCFGs

Comparison of LMs, HMMs and PCFGs

Summary

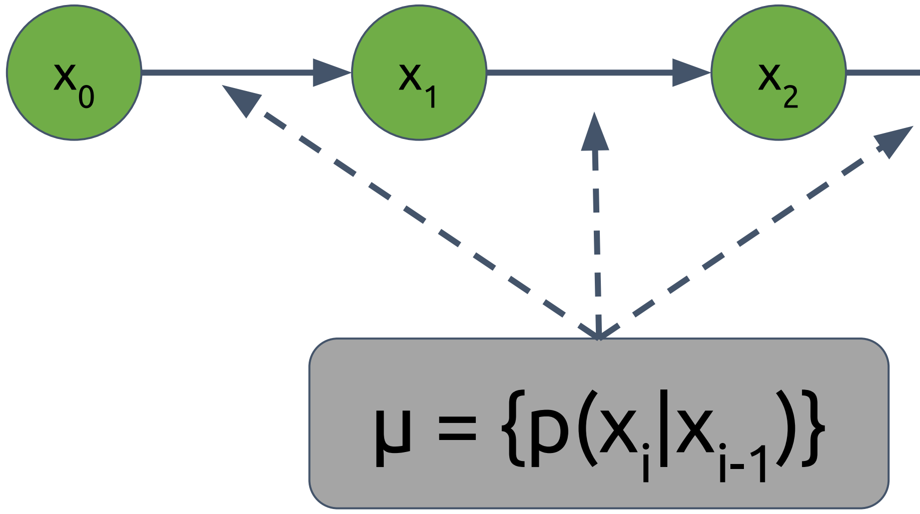
LMs, HMMs and PCFGs

Language models, **Hidden Markov Models** and **Probabilistic Context Free Grammars** are special cases of **probabilistic graphical models**

They consist of

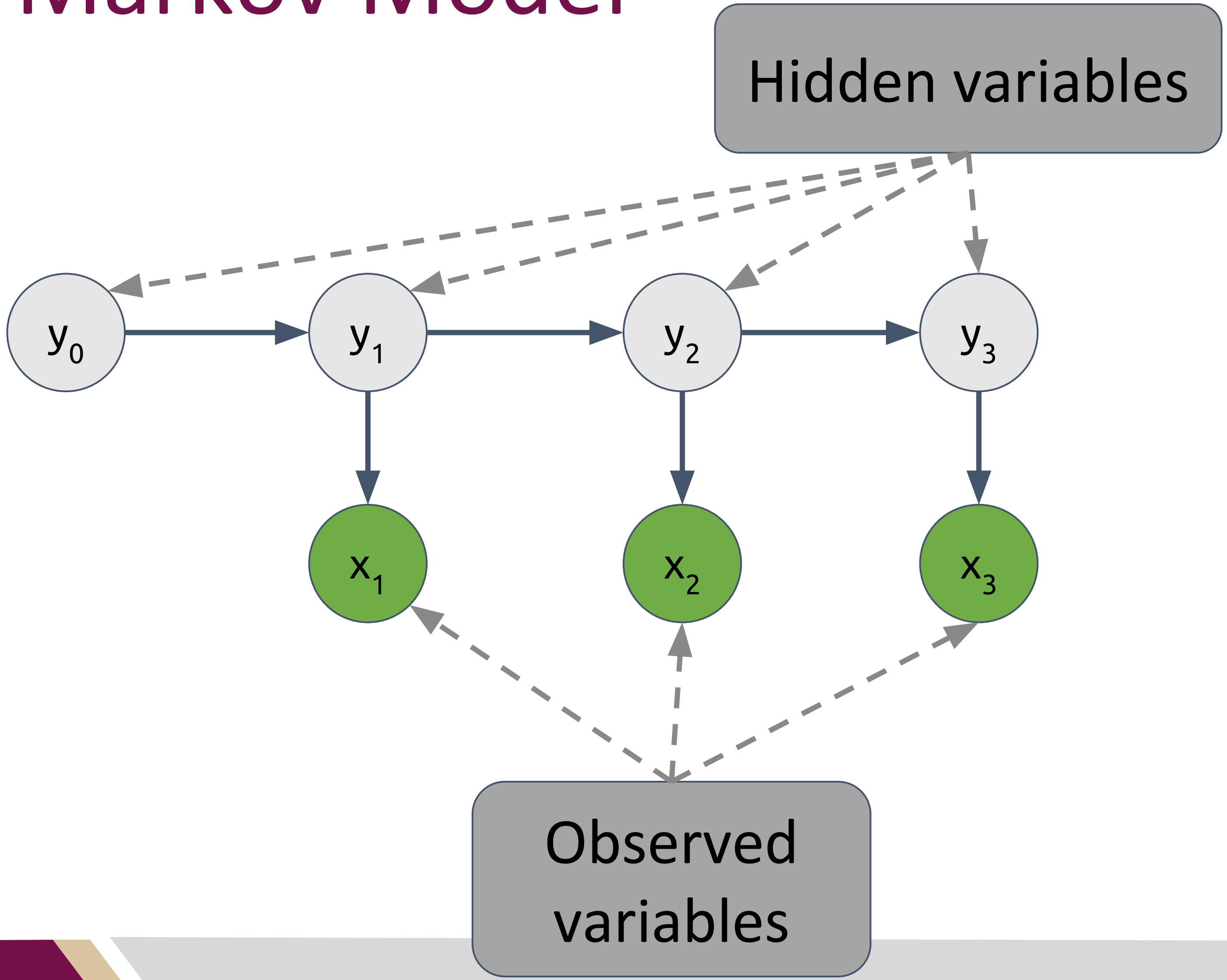
- A set of observed variables X
- A set of hidden variables Y
- A set of parameters (probabilities) μ
- An independence assumption - this generates the structure of the model

Bigram Language Model

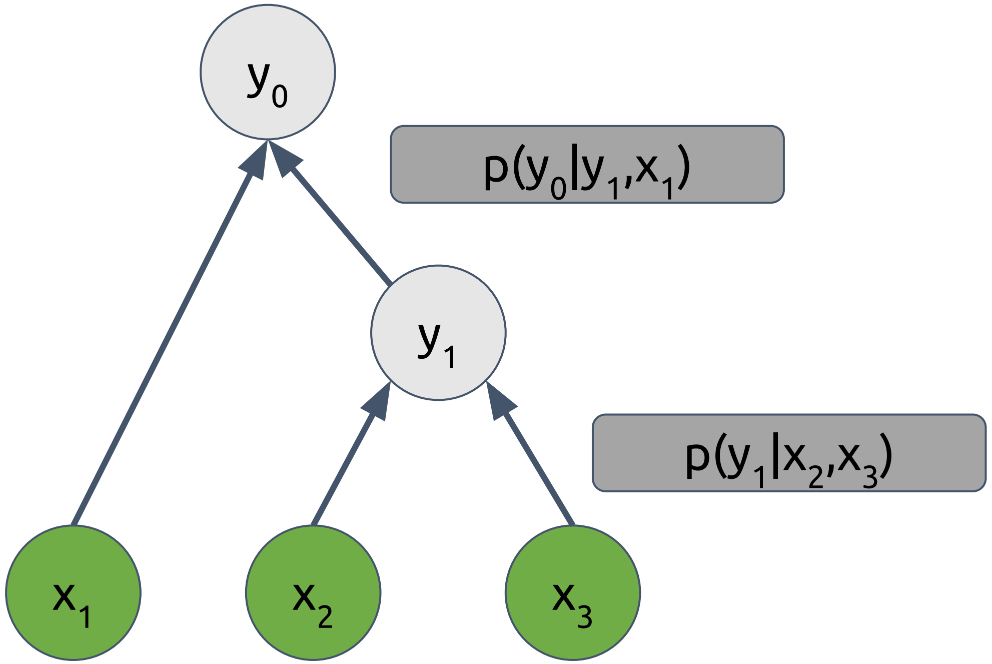


No hidden variables!

Hidden Markov Model



Probabilistic Context-Free Grammar



Problems for Probabilistic Graphical Models

Probability of observed sequence

$p(X|\mu) = \Sigma_Y\, p(X,Y|\mu)$

Most likely hidden state

$\max_Y\, p(X,Y|\mu)$

Dynamic Programming

Learning problem

Supervised: $\max_\mu\, p(X,Y|\mu)$

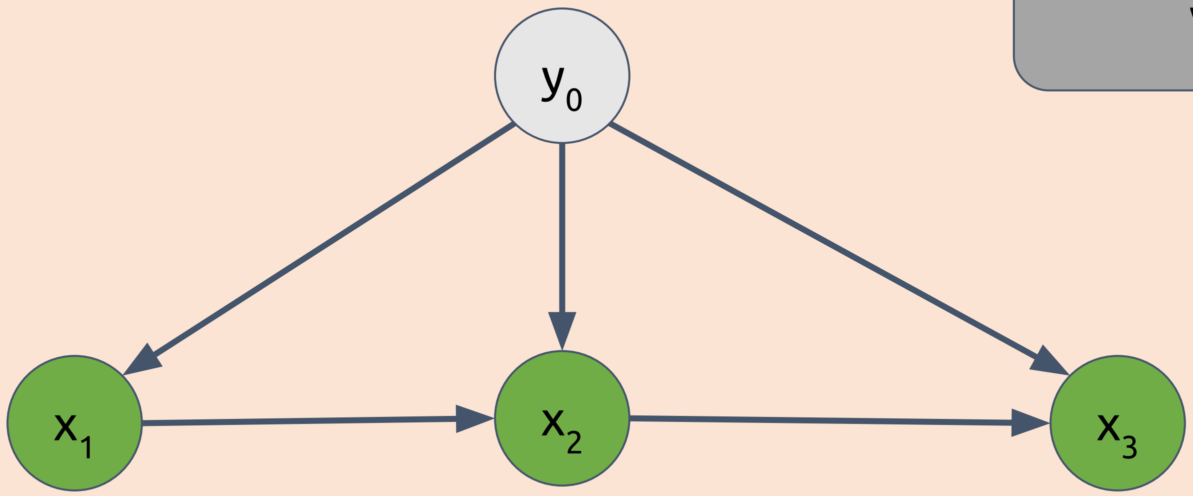
Unsupervised: $\max_{\mu,Y}\, p(X,Y|\mu)$

Counting

E-M algorithm

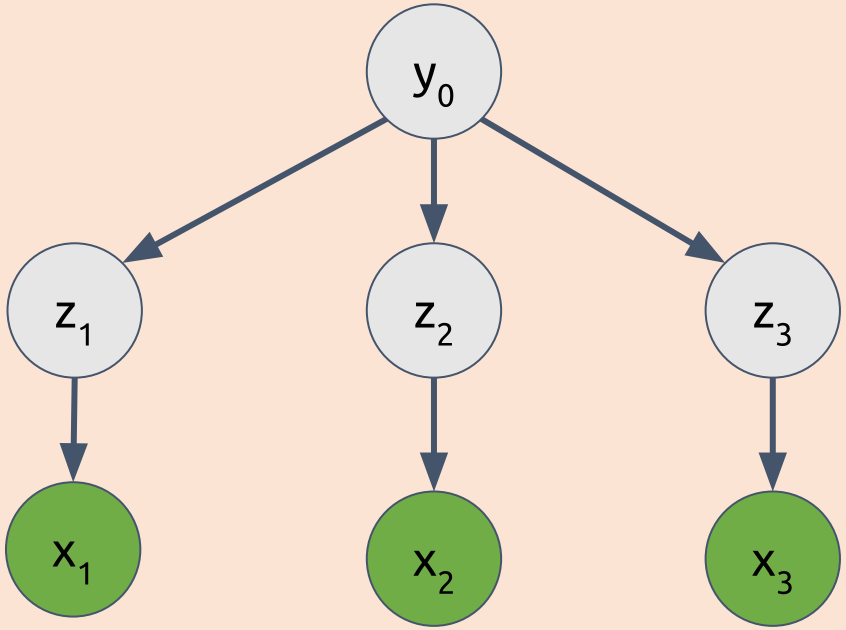
NUI Galway
OÉ Gaillimh

Topic Models



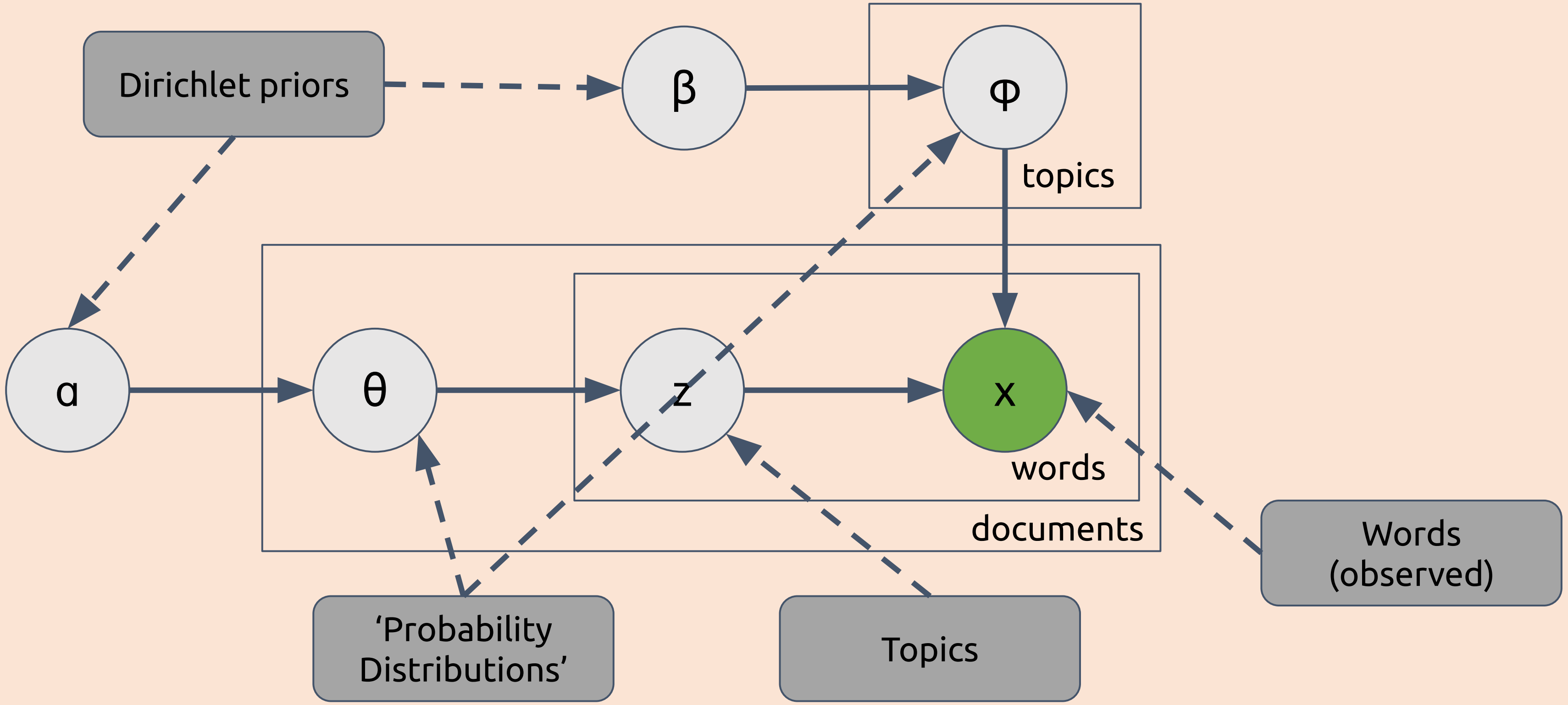
Per document 'topic' variable

Topic Models



Per document and per word ‘topic’ variable

Latent Dirichlet Allocation



$$P(X, Z, \Theta, \Phi; A, B) = \prod_i p(\phi_i; \beta) \prod_j p(\theta_j; \alpha) \prod_t p(z_{j,t}; \theta_j) p(x_{i,t} | z_{j,t}; \phi_i)$$

Overview

Parsing with Probabilistic Context-free Grammars (PCFGs)

Probabilistic Cocke-Younger-Kasami (CYK) algorithm

Problems with and solutions for PCFGs

Comparison of LMs, HMMs and PCFGs

Summary

Summary

PCFGs extends CFGs by adding weights to each production
The best parse tree can be found in cubic time
PCFGs have problems
- Cannot capture non-direct dependencies
- Cannot capture sibling dependencies
- Sometimes, cannot distinguish unlike trees

Lexicalized PCFGs fix these issues by adding a head word to each non-terminal
Dependency Grammars instead incorporate generalized links between words

Lab of this Week

Exercises on probabilistic parsing using NLTK

QA