Context-Free Grammars and CKY Algorithm

CS114B Lab 10

Kenneth Lai

April 7, 2022

Formal Language Theory

- A language is a set of strings

Formal Language Theory

- A language is a set of strings
- How to define a language?

Formal Language Theory

- A string is in a language iff:
  - It is recognized by a machine for that language

Formal Language Theory

- A string is in a language iff:
  - It is recognized by a machine for that language
  - It is generated by a grammar for that language

The Chomsky Hierarchy

- Classification of languages by the complexity of their machines/grammars

The Chomsky Hierarchy

- Classification of languages by the complexity of their machines/grammars
  - How hard is it to decide membership in a language?

The Chomsky Hierarchy

- Classification of languages by the complexity of their machines/grammars
  - How hard is it to decide membership in a language?
  - How much structure does the language define for its strings?

The Chomsky Hierarchy

Grammar	Language	Machine
Unrestricted (Type 0)	Recursively enumerable	Turing machine
Context-sensitive (Type 1)	Context-sensitive	Linear-bounded automaton
Context-free (Type 2)	Context-free	Pushdown automaton
Regular (Type 3)	Regular	Finite-state automaton

The Chomsky Hierarchy

- Regular languages: useful for describing linear structure

The Chomsky Hierarchy

- Regular languages: useful for describing linear structure
  - Example: implementing an HMM using FSTs

The Chomsky Hierarchy

- Regular languages: useful for describing linear structure
  - Example: implementing an HMM using FSTs
- Context-free languages: useful for describing hierarchical structure

Phrase structure grammars = context-free grammars

- $G = (T, N, S, R)$
  - T is set of terminals
  - N is set of nonterminals
    - For NLP, we usually distinguish out a set $P \subset N$ of preterminals, which always rewrite as terminals
    - S is the start symbol (one of the nonterminals)
    - R is rules/productions of the form $X \to \gamma$, where X is a nonterminal and γ is a sequence of terminals and nonterminals (possibly an empty sequence)
- A grammar G generates a language L.

Phrase structure grammars = context-free grammars

- $S \rightarrow aSb$
- $S \rightarrow \epsilon$

Phrase structure grammars = context-free grammars

Grammar	Lexicon
$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid the \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Pronoun$	$Pronoun \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid NWA$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	$Preposition \rightarrow from \mid to \mid on \mid near \mid through$
$Nominal \rightarrow Nominal\ Noun$	
$Nominal \rightarrow Nominal\ PP$	
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	
$VP \rightarrow Verb\ NP\ PP$	
$VP \rightarrow Verb\ PP$	
$VP \rightarrow VP\ PP$	
$PP \rightarrow Preposition\ NP$	

Figure 13.1 The $\mathscr{L}_1$ miniature English grammar and lexicon.

Chomsky Normal Form

- Every rule is of the form
  - $S \rightarrow \epsilon$
  - $A \rightarrow BC$
  - $A \rightarrow a$
- Where S is the start symbol, A is a nonterminal, B and C are nonterminals (except for S), and a is a terminal

Chomsky Normal Form

- Every rule is of the form
  - $S \to \epsilon$
  - $A \to BC$
  - $A \to a$
- Where S is the start symbol, A is a nonterminal, B and C are nonterminals (except for S), and a is a terminal
- Every CFG is equivalent to a CFG in Chomsky normal form

Converting a CFG into Chomsky Normal Form

- Add a new start symbol

Converting a CFG into Chomsky Normal Form

- Add a new start symbol
- Remove ϵ-rules

Converting a CFG into Chomsky Normal Form

- Add a new start symbol
- Remove ϵ-rules
- Remove unary rules

Converting a CFG into Chomsky Normal Form

- Add a new start symbol
- Remove ϵ-rules
- Remove unary rules
- Break up rules with more than 3 things on the right hand side

Converting a CFG into Chomsky Normal Form

- Add a new start symbol
- Remove ϵ-rules
- Remove unary rules
- Break up rules with more than 3 things on the right hand side
- Replace terminals with nonterminals and add new rules as needed

Converting a CFG into Chomsky Normal Form

- Add a new start symbol
- Remove ϵ-rules
- Remove unary rules
- Break up rules with more than 3 things on the right hand side
- Replace terminals with nonterminals and add new rules as needed

  - We can modify CKY algorithm to handle unary rules

Probabilistic CFGs

- Augment each rule in R with a conditional probability

Probabilistic CFGs

- Augment each rule in R with a conditional probability
  - $P(\text{LHS} \rightarrow \text{RHS}) = P(\text{RHS}|\text{LHS})$

Probabilistic CFGs

- Augment each rule in R with a conditional probability
  - $P(\text{LHS} \rightarrow \text{RHS}) = P(\text{RHS}|\text{LHS})$
- The probability of a parse T is the product of the probabilities of all of the n rules used to generate T

Probabilistic CFGs

- Augment each rule in R with a conditional probability
  - $P(\text{LHS} \rightarrow \text{RHS}) = P(\text{RHS}|\text{LHS})$
- The probability of a parse T is the product of the probabilities of all of the n rules used to generate T
  - $P(T) = \prod_{i=1}^{n} P(\text{RHS}_i|\text{LHS}_i)$

Probabilistic CFGs

- Augment each rule in R with a conditional probability
  - $P(\text{LHS} \to \text{RHS}) = P(\text{RHS}|\text{LHS})$
- The probability of a parse T is the product of the probabilities of all of the n rules used to generate T
  - $P(T) = \prod_{i=1}^{n} P(\text{RHS}_i|\text{LHS}_i)$
- In other words, $P(T)$ decomposes into a product of local parts

Probabilistic CFGs

- Augment each rule in R with a conditional probability
  - $P(\text{LHS} \rightarrow \text{RHS}) = P(\text{RHS}|\text{LHS})$
- The probability of a parse T is the product of the probabilities of all of the n rules used to generate T
  - $P(T) = \prod_{i=1}^{n} P(\text{RHS}_i|\text{LHS}_i)$
- In other words, $P(T)$ decomposes into a product of local parts
- This allows us to use dynamic programming

Probabilistic CFGs

- Augment each rule in R with a conditional probability
  - $P(\text{LHS} \rightarrow \text{RHS}) = P(\text{RHS}|\text{LHS})$
- The probability of a parse T is the product of the probabilities of all of the n rules used to generate T
  - $P(T) = \prod_{i=1}^{n} P(\text{RHS}_i|\text{LHS}_i)$
- In other words, $P(T)$ decomposes into a product of local parts
- This allows us to use dynamic programming
- Does this look familiar?

Probabilistic CFGs

Grammar		Lexicon
$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow that\ [.10] \mid a\ [.30] \mid the\ [.60]$
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow book\ [.10] \mid trip\ [.30]$
$S \rightarrow VP$	[.05]	$\mid meal\ [.05] \mid money\ [.05]$
$NP \rightarrow Pronoun$	[.35]	$\mid flight\ [.40] \mid dinner\ [.10]$
$NP \rightarrow Proper\text{-}Noun$	[.30]	$Verb \rightarrow book\ [.30] \mid include\ [.30]$
$NP \rightarrow Det\ Nominal$	[.20]	$\mid prefer\ [.40]$
$NP \rightarrow Nominal$	[.15]	$Pronoun \rightarrow I\ [.40] \mid she\ [.05]$
$Nominal \rightarrow Noun$	[.75]	$\mid me\ [.15] \mid you\ [.40]$
$Nominal \rightarrow Nominal\ Noun$	[.20]	$Proper\text{-}Noun \rightarrow Houston\ [.60]$
$Nominal \rightarrow Nominal\ PP$	[.05]	$\mid NWA\ [.40]$
$VP \rightarrow Verb$	[.35]	$Aux \rightarrow does\ [.60] \mid can\ [.40]$
$VP \rightarrow Verb\ NP$	[.20]	$Preposition \rightarrow from\ [.30] \mid to\ [.30]$
$VP \rightarrow Verb\ NP\ PP$	[.10]	$\mid on\ [.20] \mid near\ [.15]$
$VP \rightarrow Verb\ PP$	[.15]	$\mid through\ [.05]$
$VP \rightarrow Verb\ NP\ NP$	[.05]	
$VP \rightarrow VP\ PP$	[.15]	
$PP \rightarrow Preposition\ NP$	[1.0]	

Figure C.1 A PCFG that is a probabilistic augmentation of the $\mathscr{L}_1$ miniature English CFG grammar and lexicon of Fig. **??**. These probabilities were made up for pedagogical purposes and are not based on a corpus (any real corpus would have many more rules, so the true probabilities of each rule would be much smaller).

CKY Algorithm

- Given tables *table* and *back*:
  - Base case
    - Fill in *table* cells $[i, i+1]$ with all possible nonterminals that can generate that word

CKY Algorithm

- Given tables *table* and *back*:
  - Base case
    - Fill in *table* cells $[i, i+1]$ with all possible nonterminals that can generate that word
  - Recursive case
    - In *table*, if $A \rightarrow BC$ and B is in cell $[i, j]$ and C is in cell $[j, k]$, fill in cell $[i, k]$ with A
    - In *back*, fill in cell $[i, k]$ with backpointers (e.g. A: j, B, C)

Probabilistic CKY Algorithm

- Given tables *table* and *back*:
  - Base case
    - Fill in *table* cells $[i, i+1]$ with all possible nonterminals that can generate that word, and their probabilities
  - Recursive case
    - In *table*, if $A \rightarrow BC$ and B is in cell $[i, j]$ and C is in cell $[j, k]$, and $table[i, k, A] < P(A \rightarrow BC) \times table[i, j, B] \times table[j, k, C]$, fill in cell $[i, k]$ with A: $P(A \rightarrow BC) \times table[i, j, B] \times table[j, k, C]$
    - In *back*, fill in cell $[i, k]$ with backpointers (e.g. A: j, B, C)

Probabilistic CKY Algorithm

- Unary rules
  - In *table*, if $A \to B$ and B is in cell $[i, i+1]$, fill in cell $[i, i+1]$ with A: $P(A \to B) \times table[i, i+1, B]$
  - In *back*, fill in cell $[i, i+1]$ with backpointers (e.g. A: B)