句法分析

句法分析的常用方法(CFG,PCFG)

万泱

句法结构分析

句法结构分析是指对输入的单词序列 $w_1 w_2 \ldots w_n$（通常为句子）判断其构成是否合乎给定的语法，分析合乎语法的句子的句法结构。

- 句法分析可以简单的分为基于规则的分析方法和基于统计的分析方法，其中：
  - 基于规则的分析方法（例：**上下文无关文法（CFG）**）多用于计算机程序设计语言的编译器设计任务。
  - 基于统计的分析方法（例：**基于概率的上下文无关法（PCFG）**）多用于自然语言的句法解析任务。
- 思考：
  - 为什么要研究上下文无关法?
  - 如何定义无关?

上下文无关文法（Context-Free Grammars）

上下无关文法是一种生成式的方法，其短语结构文法可以表示成一个四元组G = (N, Σ, R, S):

- N: “非终结符”，表示在句子中不同类型的短语或子句（通常为大写字母）。
- Σ: “终结符”，表示句子的实际内容，与N无交集，即为词汇表（通常为小写字母）。
- S: “开始变量”，表示整个句子（程序），为N中的元素。
- R: $V \rightarrow \omega$ 的规则，其中 $\omega \in (V \cup \Sigma)^*$。

S -> NP VP	NP -> NP PP	NP -> stars
PP -> P NP	NP -> astronauts	NP -> telescope
VP -> V NP	NP -> eyes	P -> with
VP -> VP PP	NP -> saw	V ->saw

{"ab", "c"}* = {ε, "ab", "c", "abab", "abc", "cab", "cc", "abababab"...}

上下文无关文法（Context-Free Grammars）

将 →看成一个运算符，表示[生成]，生成规则的左侧始终是非终结符，即语法记号不会出现在生成的正式语言中。

- A $\rightarrow \alpha$ .....(1)
- A →BC .....(2)
- BAC $\rightarrow \alpha$

如果一个CFG是 ε-free的 (ε:空串)，且其语法规则只有(1)(2)两种形式,则称它为Chomsky Normal Form(CNF)，所有上下文无关文法都可以有效的变换成等价的Chomsky范式的文法。
转换方法：

- step1：将A $\rightarrow B\beta$转换为A →BC，C $\rightarrow \beta$
- step2：将A →BCD转换为A →BX，X → CD

CFG产生语句的基本方法

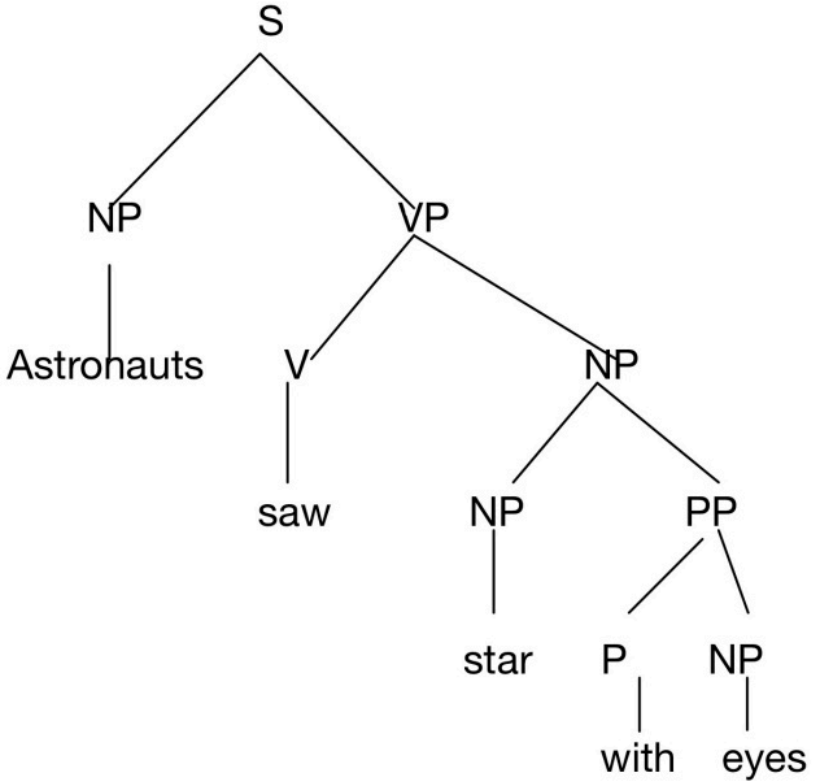
最左派生规则(left-most derivations)

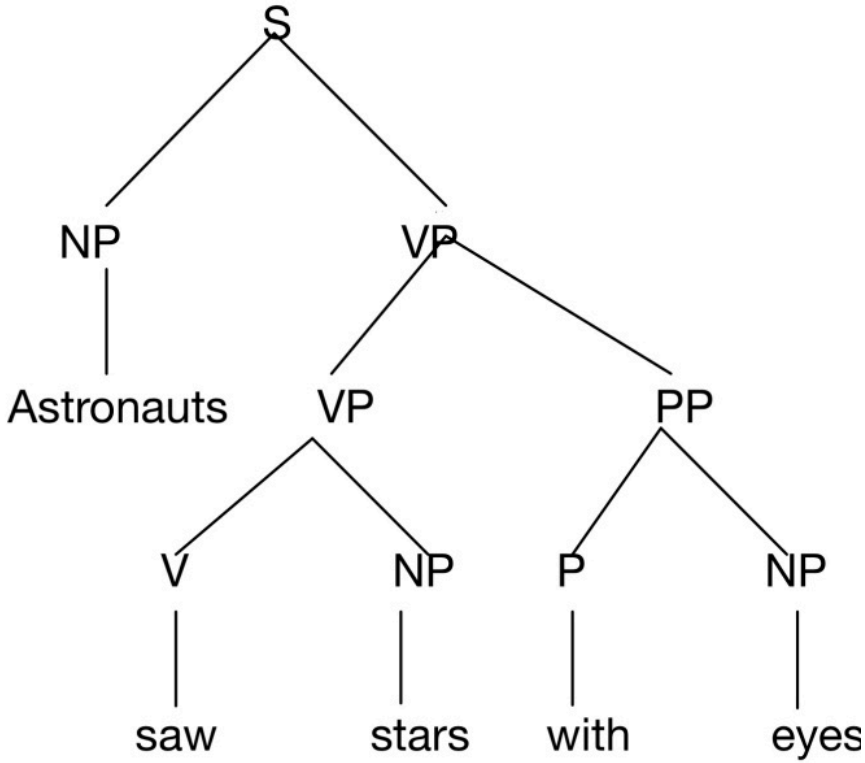
- 最左派生规则即对于一个序列 $s_1 s_2 \ldots s_n$，序列的开头是 s_1=S,其后每一个元素 s_i都是取前一个元素 s_{i-1}中从左数第一个非终结符，对其按照R中的某一规则进行派生，序列的结尾 s_n则是一个全部由终结符构成的字符串。

S -> NP VP	NP -> NP PP	NP -> stars
PP -> P NP	NP -> astronauts	NP -> telescope
VP -> V NP	NP -> eyes	P -> with
VP -> VP PP	NP -> saw	V ->saw

推导示例: [S][NP VP][astronauts VP][astronauts V NP][astronauts saw NP][astronauts saw stars]

CFG语法解析的局限性





- 一次推导对应着一棵语法解析树，故由于生成规则没有优先级和结合性的规定，每次推导的过程都对应着不同的语法树，则会产生歧义。
  比如：Astronauts saw stars with eyes.其中with eyes 这一介词短语既可以修饰动词

基于概率的上下文无关法（PCFG）

- 为了解决上面的歧义问题，故为每棵语法树赋予一个对应的概率值，概率值最大的则被认为是这个句子的语法解析树。
- 如何对每课语法树概率赋值？（对每个规则概率赋值+独立性假设）

PCFG的短语结构文法可以表示成一个五元组G = (N, Σ, R, S，P):

- N: “非终结符”，表示在句子中不同类型的短语或子句（通常为大写字母）。
- Σ: “终结符”，表示句子的实际内容，与N无交集，即为词汇表（通常为小写字母）。
- S: “开始变量”，表示整个句子（程序），为N中的元素。
- R: V $\rightarrow \omega$ 的规则，其中 $\omega \in$ (V∪Σ)*。
- P: 代表每个产生规则的概率。
  形式：A $\rightarrow \alpha$ P
  约束: $\sum_{\alpha}$P(A $\rightarrow \alpha$)=1

利用PCFG计算分析树的概率

- 独立性假设
  - 位置不变性：子树的概率与其管辖词在整个句子的位置无关。
  - 上下文无关性：子树的概率与子树管辖以外的范围词无关。
  - 祖先无关性：子树的概率与推导出该子树的结点无关。

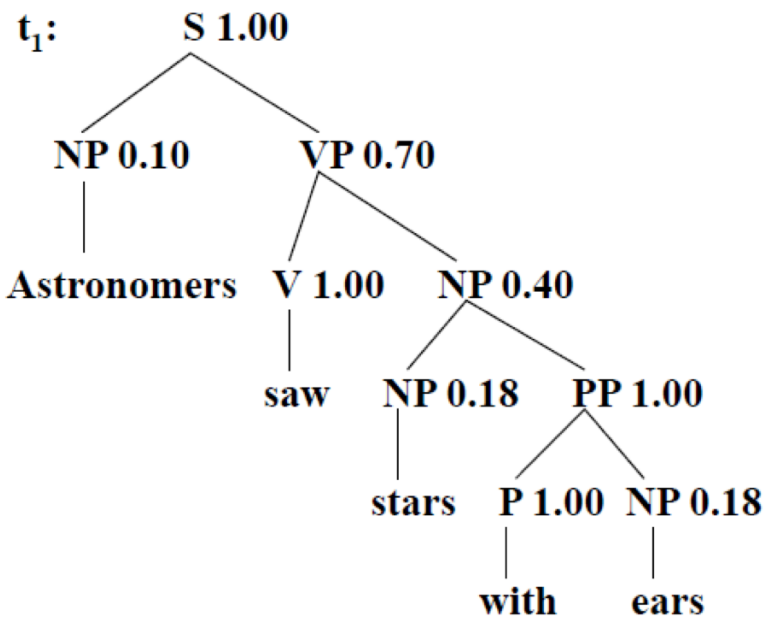
  规则集：

S -> NP VP	1.0	NP -> NP PP	0.4	NP -> stars	0.18
PP -> P NP	1.0	NP -> astronauts	0.1	NP -> telescope	0.1
VP -> V NP	0.7	NP -> eyes	0.18	P -> with	1.0
VP -> VP PP	0.3	NP -> saw	0.04	V -> saw	1.0

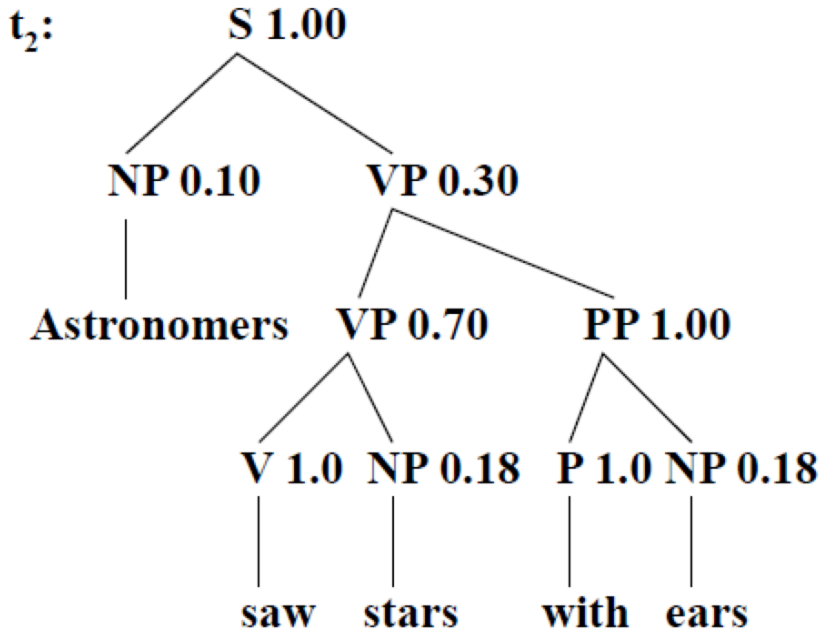
相较于CFG,PCFG对最左推导方法的每一步都添加概率，使得整棵句法分析树的概率就是所有这些独立的概率的乘积。理论上应当计算所有可能的树结构概率，取最大值对应的树作为该句子的句法分析结果。

利用PCFG计算分析树的概率

分别计算两颗语法树的概率如下：



P(T1)=1.0x0.1x0.7x1.0x0.4x0.18x
1.0x1.0x0.18=0.0009072



P(T2)=1.0x0.1x0.3x0.7x1.0x0.18x
1.0x1.0x0.18=0.0006804

P(T1)>P(T2),则分析结果T1比T2正确的可能性更大，故选择T1作为最终的句法树。

P("Astronauts saw stars with eyes.")=P(S)= $\sum_t$P(S,t)

向内算法

定义向内变量: β_A(p,q)=P(A $\rightarrow w_p w_{p+1} \ldots w_q$)=P($w_p w_{p+1} \ldots w_q$ | A),即非终结符A推导出句子中字符串的概率。 故:

- β_S(1,n)=P(S $\rightarrow w_1 w_2 \ldots w_n$)
- β_A(k,k)=P(A $\rightarrow w_k$) 如何计算 β_A(p,q) p<q ?
- 重写规则A $\rightarrow$BC （Chomsky范式）
- 假设子串 w_{pq}在位置d被分为两个部分，分别由B，C支配。

向内算法

- β_A(p,q)需考虑所有以A为左部的重写规则，以及考虑d作为分割点的不同情况，故内向算法的步骤如下：
  1.初始化
- β_A(k,k)=P(A $\rightarrow w_k$)
  2.归纳计算
- β_A(p,q)= $\sum_{B,C}\sum_d$P(A $\rightarrow BC$)β_B(p,d) β_C(d+1,q)
  3.终止
- P($w_1 w_2 \ldots w_n$)= β_S(1,n)
  向内算法通过为自底而上的递归计算句子的概率，可由CYK算法实现。

CYK算法

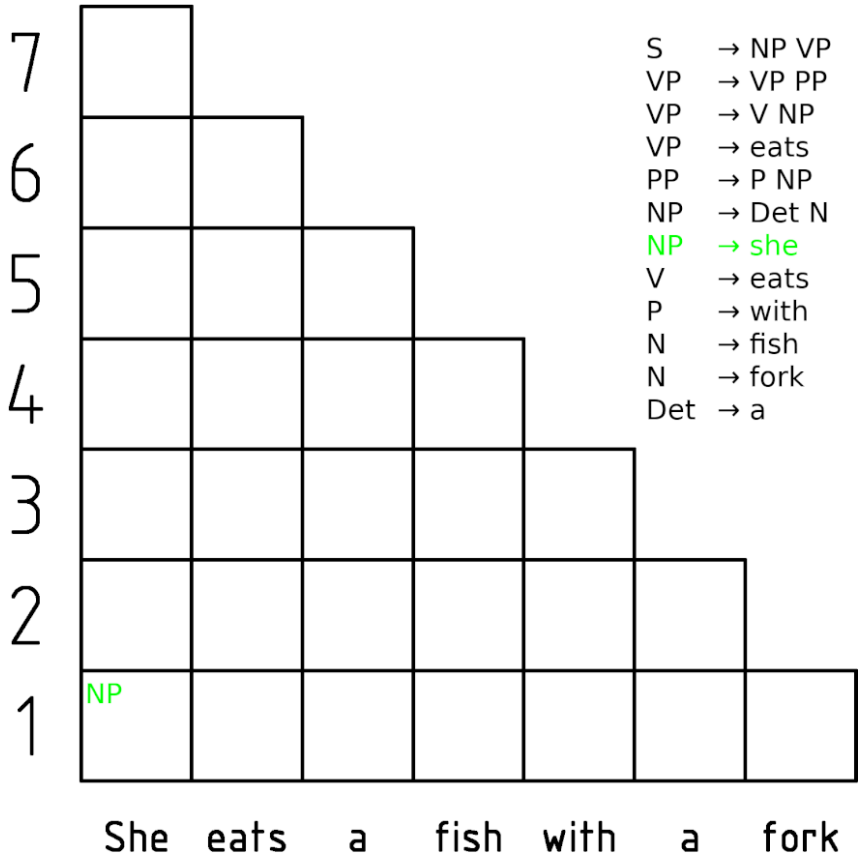
CYK算法最初基于“动态规划”算法设计思想，用于判定任意给定的字符串是否属于上下文无关文法，该算法可以在O(n^3)的时间内得出结果。
数据结构：

- 一个二维矩阵{P(i,j)}
- 每一个元素对应于输入句子某一跨度上所有可能形成短语的非终结符集合

CYK算法

算法描述：

- 对于第一行，对每一条规则A $\rightarrow w_i$，将非终结符A加入P(i,1)。
- 对于j=2...n,i=1...n-j+1,k=1..j-1,
  对每一条规则A $\rightarrow BC$ 如果B $\in$ P(i,k),C $\in$ P(i+k,j-k),那么非终结符加入P(i,j)。
- 如果S $\in$ P(1,n)，则分析成功。
  CYK算法的特点： 自底向上，广度优先，不需要回溯，时间复杂度为O(n^3)
  由于广度优先，若歧义较多时，只有分析到最后才知道结果



PCGF相关拓展

- 如何选择最佳句法树?
  - 将CYK算法拓展为概率CYK算法，即将规则概率代入递归的步骤，向上递归时只选择当前概率最大的组合情况，与Viterbi算法相似。
- 如何为文法规则选择参数?
  - 有监督（树库），count，给频率<t的规则赋值。
  - 无监督，EM算法。
- PCGF评价
  - 利用概率减少搜索空间
  - 对概率较小的子枝剪枝，加快分析效率
  - 定量比较两个语法
  - 概率计算条件太过苛刻

Thank you

`

CYK

for l in range(1, len(seg)): for i in range(len(seg)-l): j = i + l for x in self.non_terminal: tmp_best_x = {'prob':0, 'path':None} for key, value in self.rules_prob[x].iteritems(): if key[0] not in self.non_terminal: break for s in range(i, j): tmp_prob = value *best_path[i][s][key[0]]['prob']* best_path[s+1][j][key[1]]['prob'] if tmp_prob > tmp_best_x['prob']: tmp_best_x['prob'] = tmp_prob tmp_best_x['path'] = {'split': s, 'rule': key} best_path[i][j][x] = tmp_best_x self.best_path = best_path