CS 679: Natural Language Processing

Lecture #20: The Probabilistic CKY (PCKY) Parsing Algorithm

Thanks to Dan Klein of UC Berkeley, Noah Smith of CMU, and Andrew Rosenberg of Columbia for many of the slides in this presentation.

This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License.

Objectives

- Understand the CKY parsing algorithm
- Understand the Probabilistic CKY (PCKY) parsing algorithm from many perspectives
- Analyze PCKY
- Reinforce your understanding of efficient algorithms for inference with probabilistic models

Convention

- Sentence: $\mathbf{w} = (w_1, w_2, \ldots, w_n)$
- Positions:
  - Word $w_i$ occupies position $i$
  - To cover $(w_i, w_{i+1}, \ldots, w_k)$
    - We span positions $i$ through $k$
    - We build “edges” or constituents $X(i,k)$

CKY Parser: Visually

CKY Parser: Declaratively

Parsing Outline

1. Introduction to parsing natural language: Probabilistic Context-Free Grammars (PCFGs)
2. Independence Assumptions
3. Parsing Algorithm: Probabilistic CKY
4. PCFG Transformations
5. Markov grammars and other generative models
6. (Extra) Agenda-Based Parsing
7. Dependency Parsing
Probabilistic CKY

- Assign probabilities to constituents as they are completed and placed in the chart
- Keep the best scoring constituent with a given label in each cell in the chart
- Maintain back pointers to recover the parse

\[ \text{Input: } \text{Back} \]
\[ \text{Output: } \text{Assign} \]

\[ \text{Score: } \delta(i,j,k) = \log P(X_i \rightarrow X_j) \]
\[ \text{Path: } \pi(i,j,k) = \text{argmax}_{\pi(i,j,k)} \delta(i,j,k) \]
\[ \text{Goal: } C(1,n) = \langle \delta(1, n), \pi(1, n) \rangle \]

```
PCKY: Pseudo-code
```

```
int PCKY(CNF n, PCFG G) {
    n = \[C,R,L,B]\n    Create and fill the chart C(i,j,k)
    for each cell in chart:
        \( \delta(i,j,k) = \log P(X_i \rightarrow X_j) \)
        \( \pi(i,j,k) = \text{argmax}_{\pi(i,j,k)} \delta(i,j,k) \)
    return the filled chart C(1,n), C(1,n)
}
```

```
// base case
for i = 1 to n:
    \( \delta(i,i) = \infty \)
    \( \pi(i,i) = \text{null} \)
// recursion case
for i = 1 to n - span + 1:
    for j = i to n - span - 1:
        for k = i to j - 1:
            if span*span > 2 \( \delta(i,j,k) \) & \( \delta(j+1,i,k) \) & \( \log P(X_j \rightarrow X_k) \)
                \( \delta(i,j,k) = \max \delta(i,j,k), \delta(i,j,k) + \log P(X_j \rightarrow X_k) \)
                \( \pi(i,j,k) = \text{argmax}_{\pi(i,j,k)} \delta(i,j,k) \)
        return the filled chart C(1,n)
Example

- John called Mary from Denver

Grammar:
- Written on whiteboard

Base Case: X → w (Span 1)
Base Case: $X \rightarrow w$ (Span 1)

Recursive Cases: $X \rightarrow Y Z$ (Span 2)
Recursive Cases: X→Y Z (Span 2)

Recursive Cases: X→Y Z (Span 3)

Recursive Cases: X→Y Z (Span 3)

Recursive Cases: X→Y Z (Span 4)
Many ways to build $X(2,5)$

Here’s one way

Competing Analysis

Keep the Best

Recursive Cases: $X ightarrow Y Z$ (Span 5)

Reached the Goal
Extract the Best Parse

Big Picture

- PCKY is not “building a tree” bottom-up
- It is scoring partial hypotheses bottom-up
- You can assume nothing about the best tree until you get to the end!
- Follow the back-pointers
- Sound familiar? How?

Unary Productions?

- Caveat: a raw Treebank grammar is not in CNF
  - We provide code to help you binarize
- There are unary productions: X → Y, where X & Y are non-terminals

How to use Unary Productions

Analysis of PCKY

Analysis

How do we fill in C(1,2)?
Put together C(1,1) and C(2,2).
Analysis

How do we fill in C(1,3)?

1 2 3 n

One way ...

Analysis

How do we fill in C(1,3)?

1 2 3 n

Another way.

Analysis

How do we fill in C(1,3)?

1 2 3 n

Analysis

How do we fill in C(1,n)?

1 2 3 n

Analysis

How do we fill in C(1,n)?

1 2 3 n

- Run-time:
- Space:
### Analysis

- Run-time: \( O(n^2 \cdot |w| \cdot n) = O(n^4 |w|) \)
- Space: \( O(n^2 \cdot |w|) \)
- How does it remind you of Viterbi?

### Next

- Breaking out of the PCFG independence assumptions
  - Binarization
  - Markovization
  - Unlexicalized features
- Breaking out of the PCFG mold: Markov grammars
- Extra topics:
  - More inference for PCFGs: Faster search methods