Flagged CYK Algorithm

Ali Ahmadi Esfidi

December 2024

Explanation

The `Flagged CYK Algorithm` is an extension of the traditional CYK algorithm used for parsing sentences in formal grammar. This algorithm calculates a table of probabilities for different non-terminal symbols that can generate parts of a given sentence, incorporating flagged symbols that modify the probabilities based on the context.

- **Input:** The algorithm receives a sentence (a sequence of words) and a flag ratio that influences the probabilities of flagged non-terminal symbols.
- **Output:** The output is a table of probabilities, where each entry represents the likelihood of a non-terminal symbol generating a substring of the input sentence.
- **Steps:**
    1. Preprocess the input by removing flags from words and tracking how many flags were removed up to each word in the sentence.
    2. Initialize a probabilities table with zeros.
    3. Apply unary grammar rules to populate the table for single-word substrings.
    4. Process binary grammar rules iteratively, combining previously computed probabilities and adjusting for flagged non-terminals by modifying their probabilities according to the flag ratio.
- **Flagged Non-Terminals:** The algorithm specifically handles flagged non-terminals, adjusting their probability according to how many flags were removed from the corresponding words.

This algorithm is useful in probabilistic parsing tasks where sentences may contain markers (flags) that need to influence parsing probabilities.

Algorithm 1: Flagged CYK Algorithm

Input : A sentence, Flag ratio
Output: Table of probabilities

Function `FlaggedCYK`(*words, flag_ratio*)**:**
  Data: $filtered\_words$, $flagged\_counts$, $grammar$, $rule\_probs$
  Result: Table of probabilities P
  Step 1: Preprocess Input;
  Remove flags from words to get $filtered\_words$;
  Compute $flagged\_counts[i]$ to track flags removed up to the i-th word in $filtered\_words$;
  Set $length =$ size of $filtered\_words$;
  Step 2: Initialize Probabilities Table;
  Initialize P with default value 0.0;
  Step 3: Handle Unary Rules;
  for $i \leftarrow 1$ ***to*** $length$ **do**
    for $(A \rightarrow w) \in grammar.unary\_rules$ **do**
      if $w = filtered\_words[i-1]$ **then**
        $P[i, i, A] \leftarrow rule\_probs[A, w]$;
  Step 4: Process Binary Rules;
  for $l \leftarrow 2$ ***to*** $length$ **do**
    for $i \leftarrow 1$ ***to*** $length + 1 - l$ **do**
      Set $j \leftarrow i + l - 1$;
      for $k \leftarrow i$ ***to*** $j - 1$ **do**
        for $(A \rightarrow BC) \in grammar.binary\_rules$ **do**
          if $P[i, k, B] > 0$ ***and*** $P[k + 1, j, C] > 0$ **then**
            $Prob \leftarrow P[i, k, B] \cdot rule\_probs[A, B, C] \cdot P[k+1, j, C]$;
            if A *is flagged_nonterminal* **then**
              $end \leftarrow flagged\_counts[j - 1]$;
              $start \leftarrow flagged\_counts[i - 1]$;
              $flagged\_count \leftarrow end - start$;
              $Prob \leftarrow Prob \cdot (flag\_ratio^{flagged\_count})$;
            if $Prob > P[i, j, A]$ **then**
              $P[i, j, A] \leftarrow Prob$;
  return P;