COMS W4705: Natural Language Processing (Fall 2018)
Problem Set #2

Wenbo Gao - `wg2313@columbia.edu`

October 4, 2018

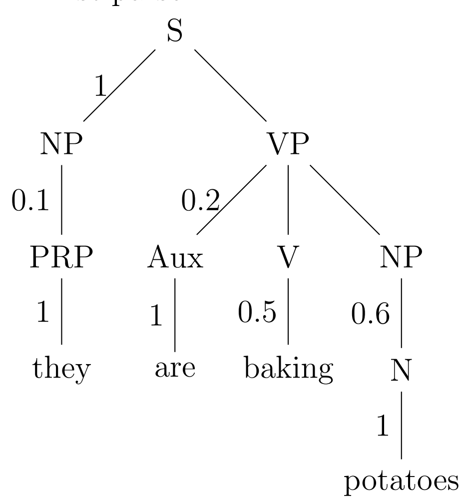
Problem 1 - PCFGs and HHMs

Both PCFGs and HMMs can be seen as generative models that produce a sequence of POS tags and words with some probability (of course the PCFG will generate even more structure, but it will also generate POS tags and words).

(a)

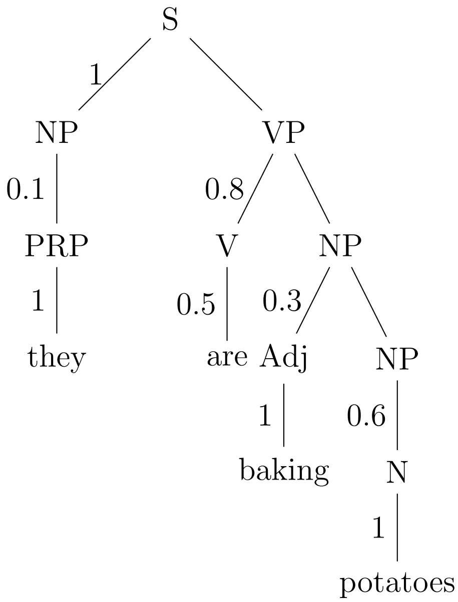
Problem. Revisit the example sentence "*they are baking potatoes*" and grammar from Problem 2. For each sequence of POS tags that is possible for this sentence according to the grammar, what is the joint probability $P(\text{tags}, \text{words})$ according to the PCFG? Hint: consider all parses for the sentence.

Solution.
first parse:



$$
\begin{aligned}
&P[tags, words = \{they, are, baking, potatoes\}] \\
=&P[S \to NP\ VP] \cdot P[NP \to PRP] \cdot P[VP \to Aux\ V\ NP] \cdot P[PRP \to they] \\
&\cdot P[Aux \to are] \cdot P[V \to baking] \cdot P[NP \to N] \cdot P[N \to potatoes] \\
=&1 \cdot 0.1 \cdot 0.2 \cdot 1 \cdot 1 \cdot 0.5 \cdot 0.6 \cdot 1 \\
=&0.006
\end{aligned}
$$

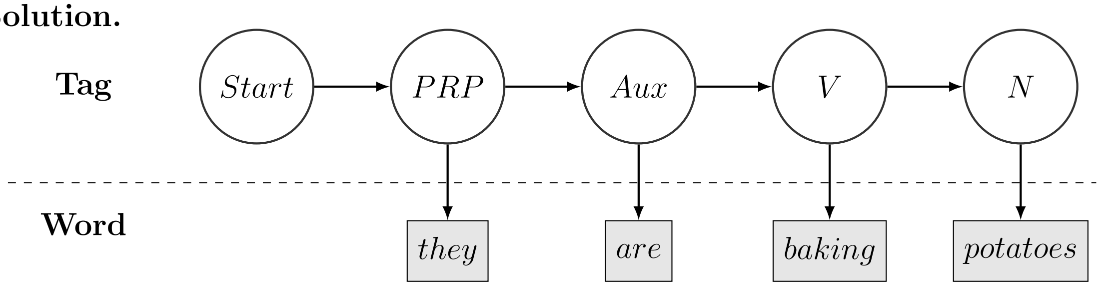
second parse:



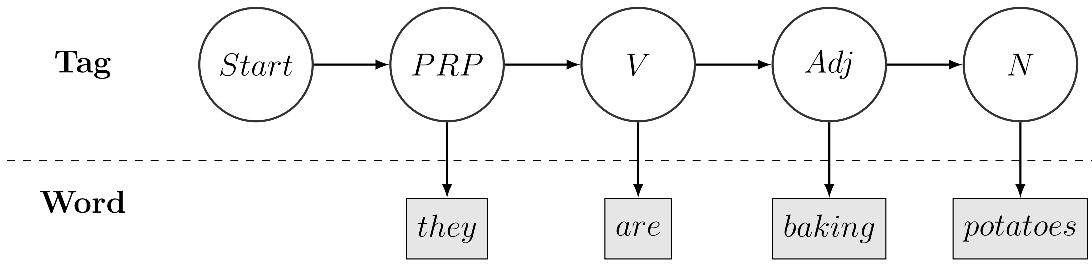
$$
\begin{aligned}
&P[tags, words = \{they, are, baking, potatoes\}] \\
=&P[S \to NP\ VP] \cdot P[NP \to PRP] \cdot P[VP \to V\ NP] \cdot P[PRP \to they] \\
&\cdot P[V \to are] \cdot P[NP \to Adj\ NP] \cdot P[Adj \to baking] \cdot P[NP \to N] \cdot P[N \to potatoes] \\
=&1 \cdot 0.1 \cdot 0.8 \cdot 1 \cdot 0.5 \cdot 0.3 \cdot 1 \cdot 0.6 \cdot 1 \\
=&0.0072
\end{aligned}
$$

(b)

Solution.



$$
\begin{aligned}
&P[tags = \{PRP, Aux, V, N\}, words = \{they, are, baking, potatoes\}] \\
=&P[tag = PRP|tag = Start] \cdot P[word = they|tag = PRP] \\
&\cdot P[tag = Aux|tag = PRP] \cdot P[word = are|tag = Aux] \\
&\cdot P[tag = V|tag = Aux] \cdot P[word = baking|tag = V] \\
&\cdot P[tag = N|tag = V] \cdot P[word = potatoes|tag = N] \\
=&0.006
\end{aligned}
$$



$$
\begin{aligned}
&P[tags = \{PRP, V, Adj, N\}, words = \{they, are, baking, potatoes\}] \\
=&P[tag = PRP|tag = Start] \cdot P[word = they|tag = PRP] \\
&\cdot P[tag = V|tag = PRP] \cdot P[word = are|tag = V] \\
&\cdot P[tag = Adj|tag = V] \cdot P[word = baking|tag = Adj] \\
&\cdot P[tag = N|tag = Adj] \cdot P[word = potatoes|tag = N] \\
=&0.0072
\end{aligned}
$$

Here is my HMM:

1. tag-word transition distributions (the same as in PCFG):
   - $P[word = they|tag = PRP] = 1$
   - $P[word = are|tag = Aux] = 1$
   - $P[word = baking|tag = V] = 0.5$, $P[word = are|tag = V] = 0.5$
   - $P[word = potatoes|tag = N] = 1$
2. tag-tag transition distributions:
   - $P[tag = PRP|tag = Start] = 1$
   - $P[tag = Aux|tag = PRP] = 0.5$, $P[tag = V|tag = PRP] = 0.5$
   - $P[tag = N|tag = V] = 0.5$, $P[tag = Adj|tag = V] = 0.5$
   - $P[tag = V|tag = Aux] = 0.048$, $P[tag = PRP|tag = Aux] = 0.952$
   - $P[tag = N|tag = Adj] = 0.0576$
   - $P[tag = PRP|tag = N] = 1$

(c)

Problem. In general, is it possible to translate any PCFG into an HMM that produces the identical *joint* probability P(tags, words) as PCFG (i.e. not just for a single sentence)? Explain how or why not. No formal proof is necessary. Hint: This has nothing to do with probabilities, but with language classes.

Solution. The hidden/latent state space in HMM can be formally described by a finite state machine which is strictly less expressive than pushdown automaton / context-free grammar in PCFG.

Problem 2

Consider the following probabilistic context free grammar.

$$
\begin{aligned}
S &\rightarrow NP\ VP & [1.0] \\
NP &\rightarrow Adj\ NP & [0.3] \\
NP &\rightarrow PRP & [0.1] \\
NP &\rightarrow N & [0.6] \\
VP &\rightarrow V\ NP & [0.8] \\
VP &\rightarrow Aux\ V\ NP & [0.2] \\
PRP &\rightarrow they & [1.0] \\
N &\rightarrow potatoes & [1.0] \\
Adj &\rightarrow baking & [1.0] \\
V &\rightarrow baking & [0.5] \\
V &\rightarrow are & [0.5] \\
Aux &\rightarrow are & [1.0]
\end{aligned}
$$

(a)

Problem. Using this grammar, show how the **Earley algorithm** would parse the following sentence.

they are baking potatoes

Write down the complete parse chart. The chart should contain $n + 1$ entries where n is the length of the sentence. Each entry i should contain *all* parser items generated by the parser that end in position i. You can ignore the probability for part (a).