Bar Ilan University
Natural Language Processing

Or Levitas

December 2022

1 Part 1

1. The reason that the sentences are so long is because of the rule:

   1 NP NP PP.

   That actually translate with the rule of:

   1 PP Prep NP.

   To:

   1 NP NP Prep NP.

   Which meaning that NP is called twice. Since every sentence go thorough *NP* rule and only 2 of those exists. The sentences becoming long.

2. Every sentence go thorough *NP* rule and *NP* rule has 50% to go to the "finishing" rule: *1 NP Det Noun*. From this "finishing" rule, there is 6 rules of the kind: *1 Noun SOMETHING*, which only one of them have "Adj" in it. Therefore 16.67% to be chosen every time. The probability for For n "Adj" in a row is: $0.16^{n-1} \times 0.84$.

3. The changes:

   (a) *9 NP Det Noun*. So there is only 10% to duplicate the *NP*.

   (b) *5 Noun Adj Noun*. For 50% to add "Adj".

4. Change the weights of the adjectives and verbs to probabilistic create more realistic English.

   The nouns are: president, sandwich, pickle, chief of staff, floor. In total 5 words.

   (a) The weight changes for the *adjectives*:

      i. **fine** - Can be in proper context with all the nouns. The weight is 1.

      ii. **delicious** - Can be in proper context on "sandwich" and "pickle" out of 5 noun words. The weight is 0.4.

iii. **perplexed** - Can be in proper context on “president” and “chief of staff” out of 5 noun words. The weight is 0.4.

iv. **pickled** - Can be in proper context on “pickle” out of 5 noun words. The weight is 0.2.

I normalize the four adjectives weights to 1:

i. **fine** - 0.5.

ii. **delicious** - 0.2.

iii. **perplexed** - 0.2.

iv. **pickled** - 0.1.

(b) The verb come between two “NP” ($S \longrightarrow NP\ Verb\ NP$), so the weights are actually capture the conditional probability. Given a noun, what the probability to choose a verb that correctly applied on the second noun. The weight changes for the *verbs*:

i. **ate** - The verb can be apply in proper context on “sandwich” and “pickle” out of 5 noun words. The weight is 0.4.

ii. **wanted** - The verb can be apply in proper context with all the nouns. The weight is 1.

iii. **kissed** - The verb can be apply in proper context with all the nouns. The weight is 1.

iv. **understood** - The verb can be apply in proper context on “president” and “chief of staff” out of 5 noun words. The weight is 0.4.

v. **pickled** - The verb can be apply in proper context on “pickle” out of 5 noun words. The weight is 0.2.

Because the rule: “Noun Adj Noun” has weight of 5 (probability of 50%). I normalize the five verbs weights to 5:

i. **ate** - 0.66.

ii. **wanted** - 1.66.

iii. **kissed** - 1.66.

iv. **understood** - 0.66.

v. **pickled** - 0.33.

2 Part 2

All changes with explanations in the file: “grammer2”. Before explain why (b) and (h)/(i) can interact in a bad way. First let’s see an example: “Sally and the president wanted and is lazy.” The problem is: *CC Verb* (In the example translate to: and is), which not suitable verb.

3 Part 3

Tree structures in file “part3.gen”.

4 Part 4

1. I chosen *(a) "a" vs. "an"*. To logic was to split every rule with "det" into 3 rules. (1)rule that have all the "det" except "an" and 'a". (2) rule that have"detA" which is "a" that can become Nouns or adjectives that starts without vowel ("detA"/"adjA"). (3) rule that have"detAn" which is "an" that can become Nouns or adjectives that starts with vowel ("detAn"/"adjAn").

   Examples:

   (a) every pickle wanted an actual desk .

   (b) a pickle is eating every sandwich .

   (c) is it true that a sandwich perplexed eating a sandwich ?

   (d) an apple ate under a floor !

2. I chosen *(b) Yes-no questions.* To logic was to come up with multiple questions format in English and apply it to a rule. The rules added are: (1)*ROOT Q ?*. (2)*Q question NNP VP.*(3)*Q question NP VP.* (4)*question did.*(5)*question was.*(6)*question will*

   Examples:

   (a) was every president pickled an apple ?

   (b) will an eel understood a proposal ?

   (c) was a floor perplexed Sally ?

   (d) did the umbrella understood eating an oven ?

5 Part 5

Since in previous exercise, a POS dictionary already have been built on an input training file. I am that dictionary to built the vocabulary and grammar rules. Generally the dictionary record the POS for a sentence with a sliding window of size 3. The sliding window create rules of a structure: $POS_1 \longrightarrow POS_2\ POS_3$. Where POS_1 is the POS of the most left word in the sliding window, POS_2 is the POS of the middle word in the sliding window and POS_3 is the POS of the most right word in the sliding window. In Addition structures like $POS \longrightarrow Word$ are saved.

Those structures are saved while traversing the train file, as well as their frequencies. Those rules are added to the initial rules to built a POS from a sentence.

For more details on creating the grammar or the dictionary, please look at the appendix.

Examples:

1. the trading correct the gain !

2. is it true that An department encourage the trade ?
3. these consultant lead Both profitability .

6 Appendix

Running the code:

```
python create_grammar.py --input INPUT --output OUTPUT
```

Code 1: Creating the grammar.

INPUT - Is the train POS file from previous task.
OUTPUT - Is the location of the ouptput file.

sentences from grammar:

```
python generate.py OUTPUT
```

Code 2: Creating sentences from the grammar.

Dictionary implementation:
While passing over the sentence, word increase the dictionary counts by +1 in four nodes:

- Word POS counts. Number *1* in Fig. 1.
- Right POS counts for that word. Number *2* in Fig. 1.
- Left POS and Right POS counts for that word. Number *3* in Fig. 1.
- Left POS counts for that word. Number *4* in Fig. 1.

To get better understanding of the dictionary structure see Fig. 2. To how does it look in the code for the word "intends".

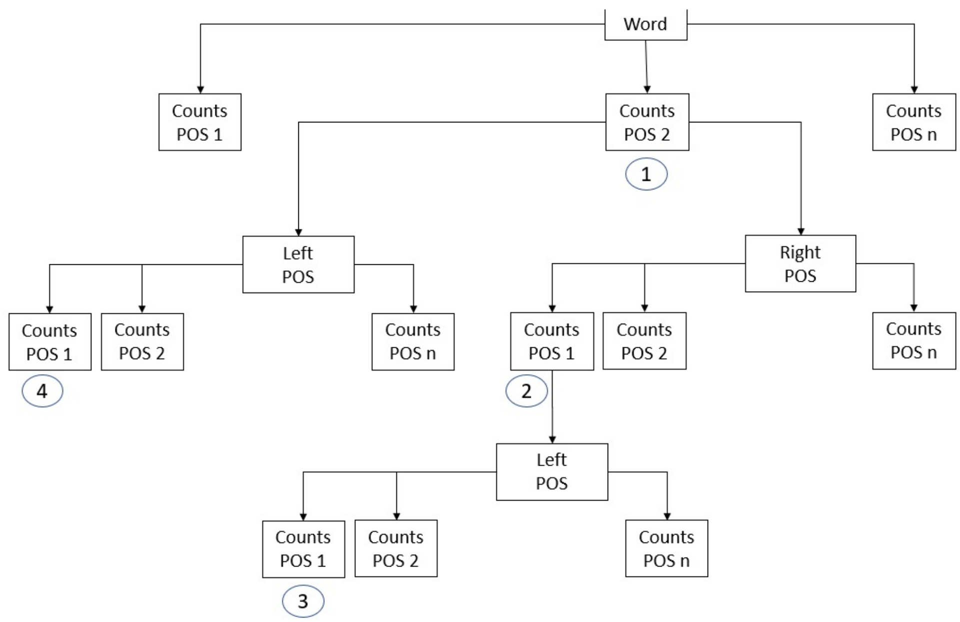

Figure 1: Illustration of the dictionary that saves the counts. Each circled number correspond to different scenario. *1* - POS of the word. *2* - POS of the word and right POS. *3* - left POS and Right POS counts for that word. *4* - POS of the word and left POS.

```
{
    "VBZ": {
        "left_pos": {
            "NNP": {
                "left_count": 1
            },
            "PRP": {
                "left_count": 1
            }
        },
        "right_pos": {
            ":": {
                "left_pos": {},
                "right_count": 1
            },
            "TO": {
                "left_pos": {
                    "NNP": {
                        "left_count": 1
                    },
                    "PRP": {
                        "left_count": 1
                    }
                },
                "right_count": 43
            },
            "``": {
                "left_pos": {},
                "right_count": 1
            }
        },
        "wp_count": 45
    }
}
```


Figure 2: Example of the dictionary, as it look in the code, for the word "intends".