COMP 6490: Document Analysis Assignment 3: NLP

By U6135394

Q2

According to Kneser Ney smoothing for Bigrams

$$P_{KN}(x_i|x_{i-1}) = \frac{\max(count(x_i, x_{i-1}) - d, 0)}{count(x_{i-1})} + \lambda(x_{i-1})P_{continuation}(x_i)$$

Where $\lambda(x_{i-1}) = \frac{d}{count(x_{i-1})} |\{x|count(x_{i-1}, x) > 0\}|$

$$P_{continuation}(x_i) = \frac{|\{x_{i-1}|count(x_{i-1}, x_i) > 0\}|}{|\{(x_{j-1}, x_j)|count(x_{j-1}, x_j) > 0\}|}$$

d = 0.75

$$count(x_i, x_{i-1}) = count(Sam, am) = 2$$

$$count(x_{i-1}) = count(am) = 3$$

$$\max(count(x_i, x_{i-1}) - d, 0) = \max(2 - 0.75, 0) = 1.25$$

$$|\{x|count(x_{i-1}, x) > 0\}| = 2$$

$$|\{x_{i-1}|count(x_{i-1}, x_i) > 0\}| = 3$$

$$|\{(x_{i-1}, x_i)|count(x_{j-1}, x_j) > 0| = 14$$

$$P_{continuation}(x_i) = \frac{|\{x_{i-1}|count(x_{i-1}, x_i) > 0\}|}{|\{(x_{i-1}, x_i)|count(x_{j-1}, x_j) > 0\}|} = \frac{3}{14}$$

$$\lambda(x_{i-1}) = \frac{d}{count(x_{i-1})} |\{x|count(x_{i-1}, x) > 0\}| = \frac{0.75}{3} * 2 = 0.5$$

$$P_{KN}(x_i|x_{i-1}) = \frac{\max(count(x_i, x_{i-1}) - d, 0)}{count(x_{i-1})} + \lambda(x_{i-1})P_{continuation}(x_i)$$

$$= \frac{1.25}{3} + 0.5 * \frac{3}{14} = \frac{11}{21} \approx 0.5238$$

Q3

A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple **(N, T, P, S)** where

N is a set of non-terminal symbols.
T is a set of terminals where N ∩ T = NULL.
P is a set of rules, P: N→ $(N \cap T)^*$.
S is the start symbol.

Thus, the errors in the giving context-free grammar rules are as flowing:

(1) PNP → nounEndWithS' (False)
Because this does not include the case that nounEndWith'S, such as uncle's.
(2) Det Nominal → Det Noun (False)
Because the left of the arrow should not have more than one non-terminal, also, the non-terminal Det hasn't given any further description in this grammar rules.
(3) Nominal → Noun
It hasn't provided further description of Noun, thus this grammar rules lack of terminals.
(4) There is no start symbols in this grammar rules.

The correct version is as following:

English possessive noun phrases context-free grammar:

N: {Poss-NP, PRP$, PNP, Article, Det, Nominal, Noun}

T: {my, uncle's bicycle, companies' workers, a, car, his, books, the, bus, stop}

S: Poss-NP

P:

Poss-NP → Det Nominal

Nominal → Det Nominal

Nominal → Nominal Noun

Nominal → Noun

Det → PRP$ | PNP | Article

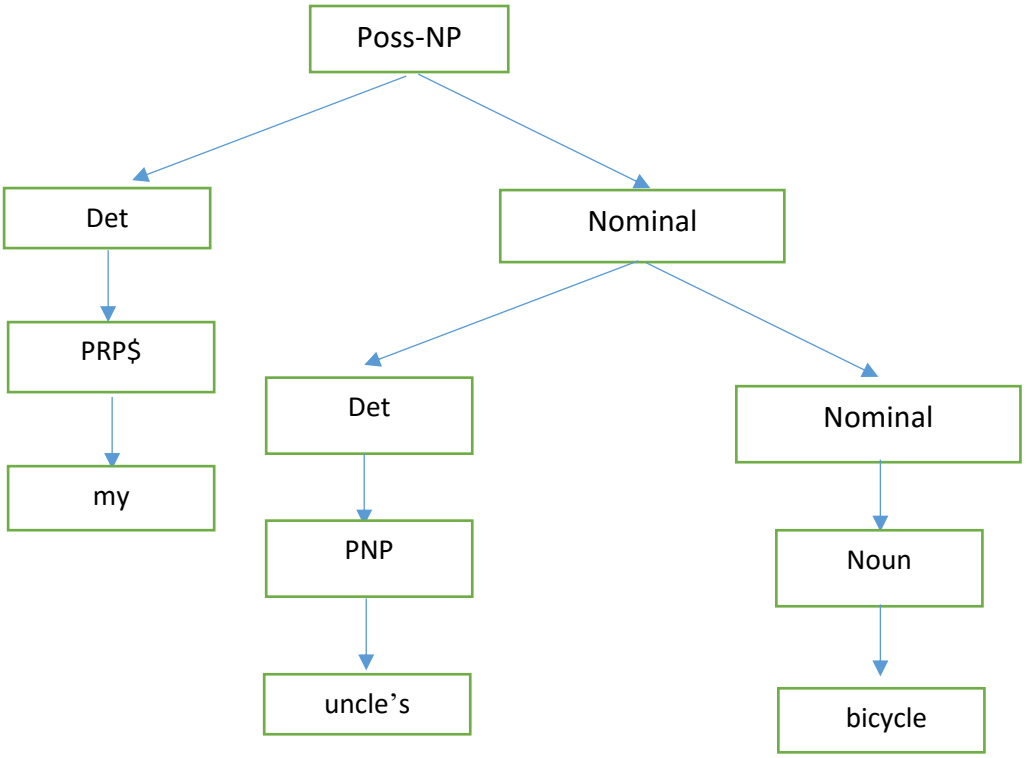
PRP$ → my | his | her | its

PNP → uncle's | companies'

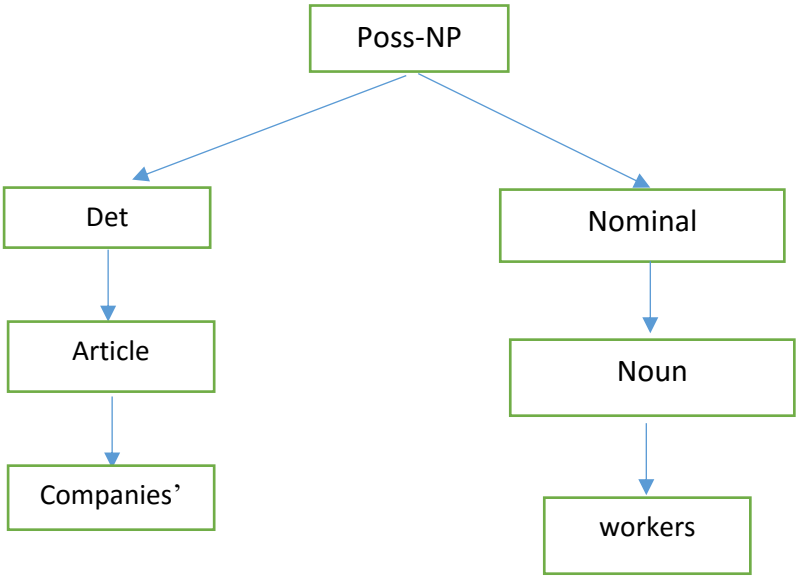
Article → a | the

Noun → bicycle | workers | car | books | bus | stop

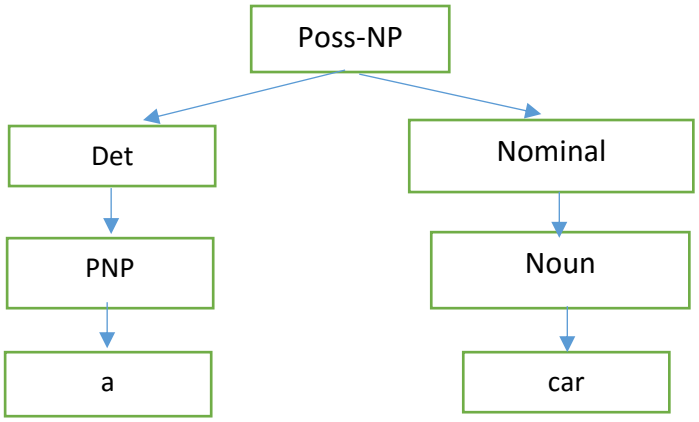
my uncle’s bicycle



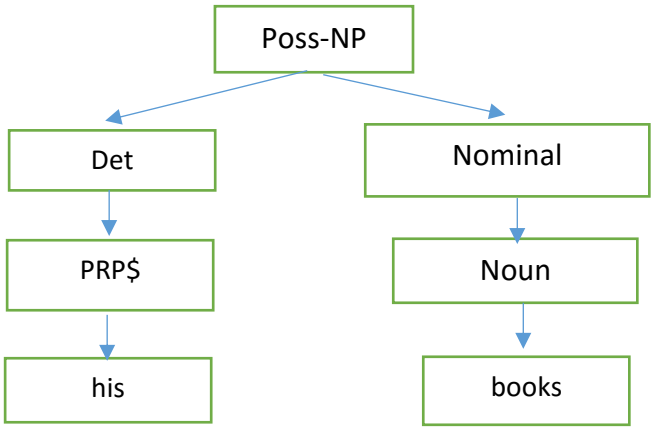
Companies’ workers



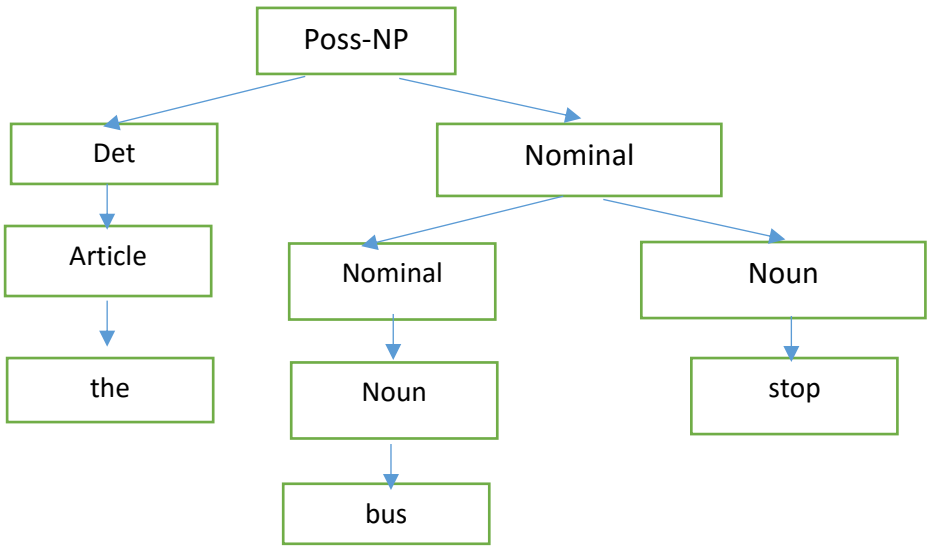
a car



his book



the bus stop



Q4

According to Collobert et al.., there are two simple ways to deal with the unseen words embedding, one is **to use a single, generic embedding for all unseen words**, the other is **to use their initial embeddings** [1]. However, Madhyastha et al. point out that these two options are not ideal because **using a single unknown embedding conflates many words**, while **the initial embeddings may be in a space that is not comparable to the trained embedding space** [1].

Thus in their paper [1], they solved this problem by **training a neural network mapping function** that takes initial word embeddings and map them to task-specific embeddings that are trained for the given task, via a multi-loss objective function.

My own solution to this problem is to **use the contexts' embeddings of the unknown words to predict the unknown words' embeddings**. According to the Distributional Hypothesis, words that appear in the same contexts share semantic meaning [2]. Thus I want to use the unknown word's last word's most likely following word's embedding to represent the unknown word or to use the unknown word's next word's most likely prior word's embedding to represent the unknown word.

For example:

There is a sentence: ……..W1 W W2…….

W1 and W2 are the word in the training set while W is the unseen world.

If we use W1 to predict the embedding of W,

$$P(x_j) = \frac{count(\ x_i,\ x_j)}{|\{x_{i+1}|count(\ x_i, x_{i+1}) > 0\}|}$$

Then I use the embedding of x_j that has the largest $P(x_j)$ among all the $\{\ x_{i+1}\ \}$ to represent the embedding of W.

Q5

(1)

According to Joakim Nivre [3], a well-formed dependency graph D = (N_w, A) need to satisfied the following four conditions:

Single head $(\forall \mathrm{n}\, n'\, n'')\ (\mathrm{n} \rightarrow n' \wedge n'' \rightarrow n') \Longrightarrow \mathrm{n} = n''$

Acyclic $(\forall \mathrm{n}\, n')\ \neg\ (\mathrm{n} \rightarrow n' \wedge n'' \rightarrow {}^{*}n)$

Connected $(\forall \mathrm{n}\, n')\ \mathrm{n} \leftrightarrow {}^{*}n'$

Projective $(\forall \mathrm{n}\, n'\, n'')\ (\mathrm{n} \leftrightarrow n' \wedge n < n'' < n') \Longrightarrow (\mathrm{n} \rightarrow {}^{*}n'' \vee n' \rightarrow {}^{*}n'')$

Left-Arc (LA) $< v_i \mid S, v_j \mid I, A > \Rightarrow < S, v_j \mid I, A \cup \{(v_j, v_i)\} >$ means the leftmost word I[0] of the list of remaining input word become the head of the topmost node S[0] of the stack and provided that the graph does not contain an arc $v_k \rightarrow v_i$ (according to the single head). An arc is added to the dependency set A and the topmost node S[0] of the stack is removed from the stack.

The reason for removing the topmost element from the stack in Left-Arc transition is **to eliminate the possibility of adding an arc $v_i \rightarrow v_j$, which would create a cycle in the graph**.

Right-Arc (RA) $< v_i \mid S, v_j \mid I, A > \Rightarrow < v_j \mid v_i \mid S, I, A \cup \{(v_i, v_j)\} >$ means the topmost node S[0] of the stack become the head of the leftmost word I[0] of the list of remaining input word and provided that the graph does not contain an arc $v_k \rightarrow v_j$ (according to the single head). An arc is added to the dependency set A and shift v_i onto the stack.

The reason for shift the leftmost element of the list to the stack is also **to eliminate the possibility of adding an arc $v_j \rightarrow v_i$, which would create a cycle in the graph**. (Since v_j must be reduced before v_i can become topmost again, no further arc linking these nodes can ever be added.)

(2)

The space complexity of Nivre's parsing algorithm is **O(n)**, where n is the length of the input sentence.

In Nivre's parsing algorithm, only one configuration C = (S, I, A) needs to be stored at any given time. Assuming that a single node can be stored in some constant space, the space need to store S and I, which the max needed space is n, respectively is bounded by the number of nodes. The same holds for A, given that a single arc can be stored in constant space, because the number of arcs in a dependency forest is bounded by the number of nodes. Hence, the word-case space complexity is O(n).

References:

[1] Madhyastha P S, Bansal M, Gimpel K, et al. Mapping Unseen Words to Task-Trained Embedding Spaces[J]. arXiv preprint arXiv:1510.02387, 2015.

[2] Vector Representations of Words. 2010, https://www.tensorflow.org/versions/r0.10/tutorials/word2vec/index.html#ve. Accessed 20 Sept. 2016.

[3] Nivre J. An efficient algorithm for projective dependency parsing[C]//Proceedings of the 8th International Workshop on Parsing Technologies (IWPT. 2003.