December 16, 2022

1 Part of Speech Tagging

1.1 No word vectors

The algorithm described here is the core for all following test, therefore it described in great length, to make sure it well understood.

The description on the POS tagging algorithm is splitted into two parts:

- **Part 1** - Creating the dictionary form the training corpus.
- **Part 2** - Using the dictionary from Part 1. Tag each word with it's relevant part of speech.

Part 1

While passing over the sentence, count each word appearance in three ways as in Fig. Generally, 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".

So now we have dictionary that records the POS in a window of size 3.

Part 2

First, the algorithm select locations based on quality. As the location have more surrounding tag POS so it quality increased. Secondly, all the location in quality tier, are examined to select the highest count. If no match found in quality tier, all the second tier is examined and so on until the last tier.

If the word is missing from the dictionary, repeat the same process, but instead of focusing on one word that has a surrounding POS, search all words that has this surrounding POS.

More specifically, given a quality tier the algorithm is as follows:

1 If the word in the dictionary. Otherwise(2).

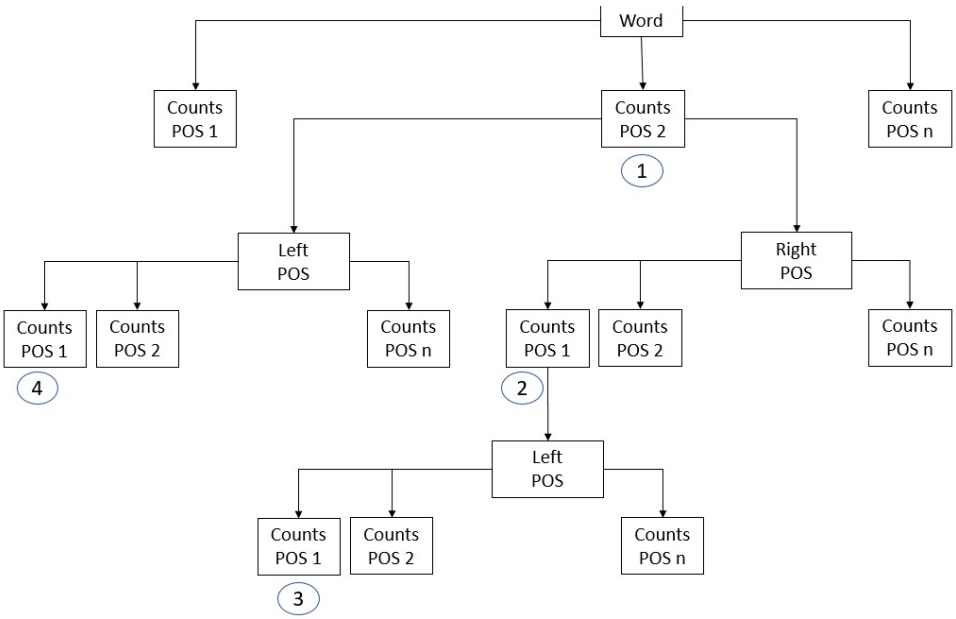

Figure 1: Illustration of the dictionary that saves the counts. Each circled number correspond to different scenario. *1* - POS of the word. *2* - POS of the word and right POS. *3* - left POS and Right POS counts for that word. *4* - POS of the word and left POS.

1.1 If right and left words are tagged and the POS are recorded in the dictionary then match the most frequent POS. otherwise(1.2)

1.2 If right word is tagged and the POS is recorded in the dictionary then match the most frequent POS. otherwise(1.3)

1.3 If left word is tagged and the POS is recorded in the dictionary then match the most frequent POS. otherwise(1.4)

1.4 Match the most frequent POS.

2 For missing words, the same process is repeated but instead of an specific word, it search for all words with that pattern.

```
(1) If the word in the dictionary. Otherwise(2).
    (1.1) If right and left words are tagged and the POS are
    recorded in the dictionary then match the most frequent POS.
    otherwise(1.2)
    (1.2) If right word is tagged and the POS is recorded in the
    dictionary then match the most frequent POS. otherwise(1.3)
    (1.3) If left word is tagged and the POS is recorded in the
    dictionary then match the most frequent POS. otherwise(1.4)
```

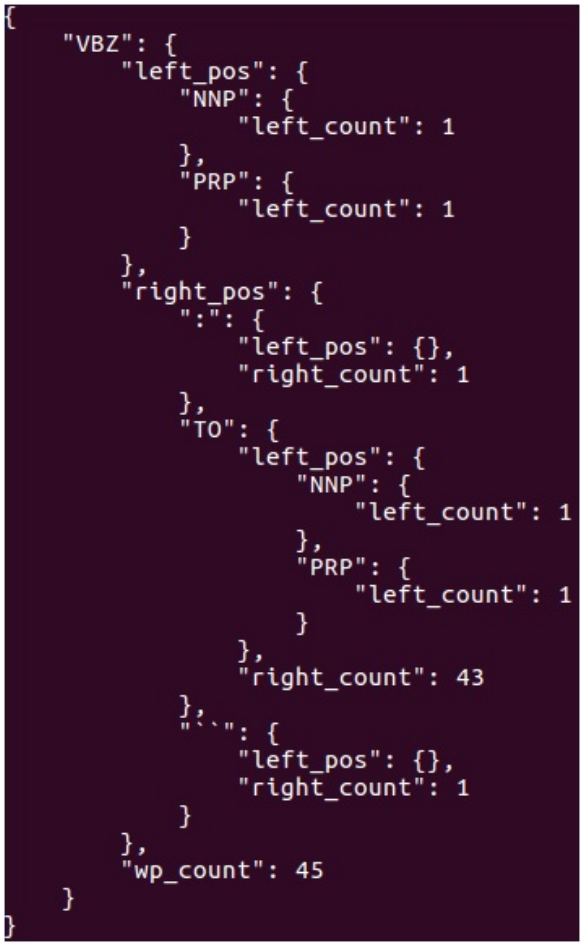

Figure 2: Example of the dictionary, as it look in the code, for the word "intends".

```
    (1.4) Match the most frequent POS.
(2) For missing words, the same process is repeated but instead of
    an specific word, it search for all words with that pattern.
```

Code 1: Tagging algorithem for a selected location.

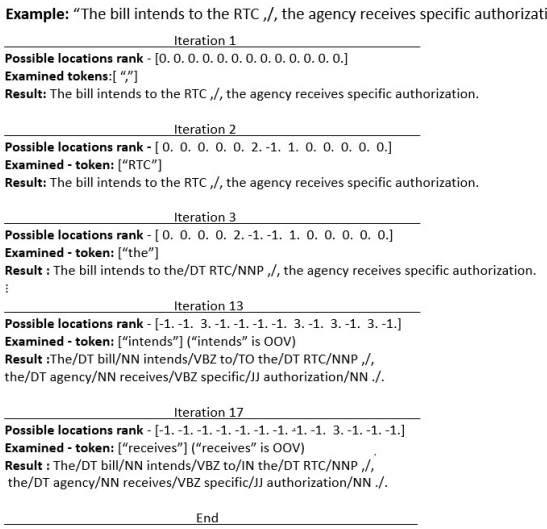

Figure 3: Example of the POS tagging.

The quality tiers have different priority number:

- **3** - Left and right words have POS.
- **2** - Only left word have POS tags.
- **1** - Only right word have POS tags.
- **0** - No surrounding POS tags.
- **-1** - Words already tagged.

In less formal description, and more intuitive way, the process can be looked at in the following way: While not all the words are tagged, the iteration continues. At the beginning all locations have value of zero, the word with the highest count is chosen and essentially become a "seed" for tagging (because, now it's left location have the value of 2 and the right location have the value of 1). The tagging continue to the left side, until unknown word is encountered or beginning of sentence. Then the tagging is continued to the right. If another unknown words or end of sentence is encountered, then a new "seed" is planted. This process continue until all words in the sentence are tagged.

Example of tagging process, with the proposed locations (quality tier) and the POS tag are described in Fig. 3.

The accuracy on the development set is: $\boxed{92.4\%}$

1.2 Static word vectors

Since the OOV words are 3% of the test and development files. I tried to tackled this problem directly. The algorithm is the same as described in previous

sections. However now, when facing a missing word, the algorithm search for the most similar word to that. If the missing word from the training set also missing in the word dictionary, then search for all words with that surrounding POS. Section (2) in the Algo. 1.

different embedding dictionaries were tested:

- With model: **"word2vec-google-news-300"**. The result on the development set: 92.6%
- With model: **"glove-twitter-50"**. The result on the development set: 92.9%
- With model: **"glove-wiki-gigaword-50"**. The result on the development set: 93.2%

For the static word vectors: "glove-wiki-gigaword-50". Different thresholds were tested, when the cosine similarity required to be greater than the threshold.

- **Threshold=0.5**. The result on the development set: 93.2%
- **Threshold=0.6**. The result on the development set: 93.2%
- **Threshold=0.7**. The result on the development set: 93.0%
- **Threshold=0.8**. The result on the development set: 92.8%

Relative the huge vocabulary sizes: "word2vec-google-news-300" has vocabulary size of 3000000 and "glove-twitter-50" has 1193514. The "glove-wiki-gigaword-50" has only 400000. Although this huge difference, "glove-wiki-gigaword-50" get better results. Perhaps more missing words in the development set are existing in "glove-wiki-gigaword-50". A further research in this matter is needed. In addition, different thresholds seems to deteriorate the results, which strengthen the intuition that finding similar word and then search for a pattern is better than search for POS surrounding pattern.

The accuracy on the development set is: $\boxed{93.2\%}$

1.3 Contextualized word vectors

Continue the same line of thought. When facing a missing word, the K top most similar words are suggested with "roberta-base". Those suggestion are searched in the word count dictionary. If the missing word from the training set also missing in the word dictionary, then search for all words with that surrounding POS. Section (2) in the Algo. 1.

Multiple test were examined:

- K=20 - The result on the development set: 93.4%
- K=10 - The result on the development set: 93.4%
- K=5 - The result on the development set: 93.4%

- K=3 - The result on the development set: 93.4%
- K=2 - The result on the development set: 93.3%
- K=1 - The result on the development set: 93.3%

For the highest score result, the lowest K were chosen. K=3. Different thresholds were tested, where the score returned from the model for each word required to be greater than the threshold.

- **Threshold=0.3**. The result on the development set: 92.6%
- **Threshold=0.2**. The result on the development set: 92.8%
- **Threshold=0.1**. The result on the development set: 93.0%
- **Threshold=0.05**. The result on the development set: 93.1%
- **Threshold=0**. The result on the development set: 93.4%

The The accuracy on the development set is: $\boxed{93.4\%}$

2 Named Entities Recognition

The best algorithm from all the trails, the algorithm from 1.3 with 3-top suggestions and threshold of 0, is applied here.

The results from the "ner_eval.py":

- **Accuracy** - $\boxed{95.90\%}$
- **All-types** - Precision: 76.90% Recall: 78.11%.

All F1 : 77.50%

ORG - Precision: 64.12%, Recall: 68.75%, F1: 66.35%.

PER - Precision: 82.36%, Recall: 78.83% F1: 80.55%.

LOC - Precision: 81.98%, Recall: 85.19% F1: 83.56%.

MISC - Precision: 75.89%, Recall: 76.14% F1: 76.02%.