Université Paris Cité

Evaluating a PoS Tagger

Guillaume Wisniewski
guillaume.wisniewski@u-paris.fr

September 2022

- **you must submit a report in pdf format before September 29 9am. The report must be sent by email and include, for each question, either the code allowing to answer it or a precise answer.**
- **using google colab might a good idea if you do not have access to a linux computer.**
- **do not hesitate to send me any corrections, comments or suggestions to improve this lab!**

This lab assumes that you know either how to use google colab (see `https://colab.research.google.com/#` for an introduction) or python from the command line. While the latter solution is far better (it is not always desirable to give your data to google and knowing how to use a terminal is a useful skill in NLP[1]), the former allows you to start working on your lab without installing anything on your computer.

You should also know:

- basic python programming (defining and using functions, iterations and conditions)
- how to use a dictionary.

1 Context

Part-of-Speech (PoS) tagging is the task that consists in assigning to each word of a sentence a label describing its *grammatical category* (noun, verb, ...). Here are two sentences whose words are labeled by their PoS:

[1] at least according to people of my generation.

(1) J' aimerai peindre la porte en majorelle .
I love paint the door in majorelle .
PRON VERB VERB DET NOUN ADP NOUN PUNCT

(2) Iel la porte avec panache .
They it wear with style .
PRON PRON VERB ADP NOUN PUNCT

In this two examples, the PoS allow to distinguish the two meanings of *la porte*.

A PoS tagger can be seen as a function that takes in a sentence and returns a list of labels such that:

- there is a pre-defined set of possible labels (the tagset);
- the i-th element of the output list corresponds to the label of the i-th word of the sentence.

In practice, the input sentence has to be tokenized so that words can be easily (and unambiguously) identified and the PoS tagger inputs is generally either a string in which "words" are all separated by a space or directly a list of string, each string corresponding to a "word". For instance the sentence "I'm a great admirer of Ms. O'Neil's work" will (generally) be tokenized either as `"I 'm a great admirer of Ms. O'Neil 's work"` or as `["I", "'m", "a", "great", "admirer"`
Note that transforming these two forms of tokenized sentences is trivial with the `join` and `split` functions:

```
>>> s = "I 'm a great admirer of Ms. O'Neil 's work"
>>> ss = s.split()
>>> ss
['I', "'m", 'a', 'great', 'admirer', 'of', 'Ms.', "O'Neil", "'s", 'work']
>>> " ".join(ss)
"I 'm a great admirer of Ms. O'Neil 's work"
```

2 Using Spacy PoS tagger

In this lab, we will use the PoS tagger of `Spacy`, one of the most popular NLP library in python.

2.1 Installing Spacy

The following instructions can be used to install `Spacy` on a Unix-environment (Linux, MacOS and, with some small modificaitons, Colab[2]):

[2] `Spacy` is already installed on Colab, but you have to download the French model by executing a cell containing the command: `!python -m spacy download fr_core_news_sm`

```
> python3 -m venv local
> ./local/bin/pip install spacy
> ./local/bin/python -m spacy download fr_core_news_sm
```

The first command will create a virtual environment, the second one install `Spacy` and the third download the model we are going to use.

More information about virtual environments can be found on the moodle of the lecture. The only thing that you need to know, is that to use the `Spacy` in your virtual environment you need to run your scripts with python interpreter you just installed (`./local/bin/python`).

2.2 Predicting the PoS tags of a sentence

To use `Spacy` in your program you have to import the library and load a model:

```
import spacy
fr_model = spacy.load('fr_core_news_sm')
```

By default, `Spacy` takes, as input, a "raw" sentences and predict several kind of information: tokenization of the sentence into words, PoS tagging, dependency analysis, ... The following code will predict only the PoS of sentence that has already be tokenized:

```
1 from spacy.tokens import Doc
2
3 def predict_pos(sentence, model):
4     model.tokenizer = lambda x: Doc(model.vocab, x.split())
5     return [token.pos_ for token in model(sentence)]
6
7 print(predict_pos("J' aime le chocolat", fr_model))
```

The `model` parameter corresponds to a `Spacy` model (the object returned by the `spacy.load` function).

1. What is lambda function in python and why do we use such a function in line 4 ?
2. Explain the last line of the `predict_pos` function.
3. Write a function, that takes as input a list of sentences and returns a list of PoS annotations (i.e. a list of lists of PoS)

3 Metrics for PoS Tagging

In the following, we will assume we have a *test set* that contains pairs of (tokenized) sentences and their gold PoS labels. We will denote $\mathbf{x}^{(i)}$ the i-th sentence of our test set and $\mathbf{y}^{(i)}$ the corresponding sequence of labels. The test set is made of n sentences: $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)_{i=1}^{n}$. We will also denote x_j the j-th element of the sequence $\mathbf{x}$. So the j-th word of the i-th sentence of the test set is $x_j^{(i)}$ and its label $y_j^{(i)}$. We will use an hat to distinguish between predicted label (such as $\hat{y}_j$ and the corresponding gold label y_j.

Several metrics can be defined to evaluate the quality of a PoS tagger:

- the **(sentence) accuracy** corresponds to the proportion of sentences of the test set for which all labels are predicted correctly:

$$\frac{1}{n} \times \sum_{i=1}^{n} \mathbb{1}\left\{\mathbf{y}^{(i)} = \hat{\mathbf{y}}^{(i)}\right\} \tag{1}$$

  where $\mathbb{1}$ is the indicator function that returns 1 if its predicate is true and 0 otherwise.

- the **micro (word) accuracy** corresponds to the proportion of word in the test set for whose label is correctly predicted:

$$\frac{1}{\#\text{words}} \times \sum_{i} \sum_{j} \mathbb{1}\left\{y_j^{(i)} = \hat{y}_j^{(i)}\right\} \tag{2}$$

- the **macro (word) accuracy** corresponds to the average of the accuracy computed for each possible PoS label.

4. Why do we (have to) consider tokenized sentences to evaluate a PoS tagger?

5. Implement each of the evaluation metric defined in this section.

Sometimes it is more meaningful to evaluate a PoS tagger in terms of error rates:

- the **(sentence) error rate** corresponds to the proportion of sentences of the test set for which at least one label is mispredicted:

$$\frac{1}{n} \times \sum_{i=1}^{n} \mathbb{1}\left\{\mathbf{y}^{(i)} \neq \hat{\mathbf{y}}^{(i)}\right\} \tag{3}$$

- the **(word) error rate** corresponds to the proportion of words of the test set for which the label is mispredicted:

$$\frac{1}{\#\text{words}} \times \sum_{i} \sum_{j} \mathbb{1}\left\{y_j^{(i)} \neq \hat{y}_j^{(i)}\right\} \tag{4}$$

6. How can these new metrics be computed easily (i.e. by using the functions you have already implemented)

4 Estimating PoS Tagging Performance

In this section we will consider the corpus contained in the file `sequoia.test.json`. This corpus can easily be loaded in python with the following code:

```
import json
corpus = json.load(open("sequoia.test.json", "r"))
```

The variable `corpus` is a list of dictionaries, each dictionary contains two keys, `tokenized_sentence` and `gold_tags` the meaning of wich is obvious.

7. How many sentences and words are there in the Sequoia corpus?

8. Evaluate the performance achieved by the tagger described in Section 2 on the corpus. Interpret.

9. Write a function that evaluate the performance of a PoS tagger (with a user-provided evaluation function) on $n\%$ of the sentence chosen randomly from a corpus (both n and the corpus should be parameter of the function). Remember that:
   - the `sample` function of the `random` module can be used to sample elements from a list.
   - its possible to pass function as parameters in python:

```
def add(a, b):
    return a + b

def print_squared(func, a, b):
    print(func(a, b) ** 2)

print_squared(add, 2, 2)
```

In the two following questions, we will only consider the word error rate metric.

10. Draw randomly 1,000 corpora containing half of the sentences of the original corpus and evaluate the PoS tagger on each of these corpus. What are the largest and the smallest scores achieved? Draw an histogram representing the scores distribution. What can you conclude?

11. Compute the accuracy of the PoS tagger estimated on 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80% and 90% of the test set. For each corpus size, you will report the accuracy averaged over 10 samples of the original data as well as the standard deviation. How close is this accuracy to the accuracy measured on the whole corpus. Conclude.

5 Out-of-Domain Evaluation

We will now consider three additional test sets: the first one, `Minecraft` contains sentences extracted from discussions between Minecraft players, the second one `GSD` contains sentences grabbed from the web (e.g. from Wikipedia) and the last one `ParisStories` is a corpus of oral French. The last two corpora are part of the Universal Dependencies project (`https://universaldependencies.org/`)

12. Do you think that the error rate estimated on `Sequoia` is representative of the performance that the PoS tagger will obtain on these test sets?

13. Evaluate the performance of the PoS tagger on these two corpora. What can you conclude?