HMM PoS tagging

Natural Language Processing and Deep Learning

Francisco Martínez Lasaca (frml@itu.dk)

July 18, 2020

1 Introduction

Parts of speech (PoS; also known as *word classes* or *syntactic categories*) are the different categories in which words that behave similarly within a sentence can be classified [2]. Most of them rely on the following categories: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. Identifying the parts of speech of the words in a sentence is useful, as we can infer which are the most likely types of the surrounding words, and it eases information retrieval and coreference resolution.

A program that given a sentence, tags their words appropriately is called a *PoS tagger*, there are multiple ways to implementthem. In this report, we present the methodology and analysis of an implementation using a Hidden Markov model or HMM.

2 Data

In order to create a model to work with, we need data. We use an extensive PoS-tagged corpora called Univeral Dependencies (UD).[1] In particular, we are going to use the AnCora Catalan treebank,[2] a PoS-tagged corpus made up of news written in Catalan —a language spoken in the north-east of Spain.

The corpus is made of 14969 sentences (474,445 words), which are already divided into a train corpus with 16678 sentences and a test one, with 1846 sentences. This means that the 90 % of the sentences are used for training, while the remaining 10 % is reserved for testing.

[1] https://universaldependencies.org

[2] https://github.com/UniversalDependencies/UD_Catalan-AnCora

3 Methodology

The implementation is available in a Google Colab notebook.[3]

Given a sequence of words $w_1^n = w_1, \ldots, w_n$, the goal is finding the tags $t_1, \ldots, t_n$ that are most probable to be associated to the given sequence of words. We call this estimate of the best tag sequence $\hat{t}_1^n$. We can then apply the Baye's rule.

$$\begin{aligned}\hat{t}_1^n &= \arg\max_{t_1^n} P(t_1^n \mid w_1^n) \\ &= \arg\max_{t_1^n} P(w_1^n \mid w_1^n) \cdot P(t_1^n)\end{aligned}$$

But this formulation of $\hat{t}_1^n$ is still too difficult to calculate. We can simplify it by assuming:

- That the probability of a word appearing is dependent only on its own tag and is independent of other words and tags around it.

$$P(w_1^n \mid t_1^n) \approx \prod_{i=1}^{n} P(w_i \mid t_i)$$

- The probability of a tag appearing is dependent only on the preceding tag. This is known as the *bigram assumption*.

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i \mid t_{i-1})$$

So

$$\begin{aligned}\hat{t}_1^n &= \arg\max_{t_1^n} P(t_1^n \mid w_1^n) \\ &\approx \prod_{i=1}^{n} P(w_i \mid t_i) \cdot P(t_i \mid t_{i-1})\end{aligned}$$

We can calculate $P(t_i \mid t_{i-1})$ by counting how many times we find a word of type t_i preceded by

[3] https://colab.research.google.com/drive/1Vu-a1B_myBVCOgZi9yY49UfxrbSw1JU0?usp=sharing

a word of type t_{i-1}, and divide by the number of times that we see a word of type t_{i-1}:

$$P(t_i \mid t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

A similar counting procedure can be done with $P(w_i \mid t_i)$, where we divide the number of times we see word w_i being tagged with t_i by the total number of words of type t_i:

$$P(w_i \mid t_i) = \frac{C(w_i \mapsto t_i)}{C(t_i)}$$

To implement all of this, we use the Viterbi algorithm, which is a dynamic algorithm that fills a table incrementally consuming the input word by word.

The recurrence of the algorithm is defined: $v[j,t] = 1$ if $t = 0$ and $j = 0$, and $v[j,t] = 0$ if $t = 0$ and $j \neq 0$. The general case is:

$$v[j,t] = \max_{1 \leq i \leq N} v[i, t-1] \cdot a_{ij} \cdot b_j(o_t)$$

where N is the number of tags, plus an extra tag for the start of the sequence, a_{ij} is the transition probability going from tag i to j and $b_j(o_j)$ is the state observation likelihood of the observation o_t (the current word) given the current state j.

We use a matrix P for calculating all the $P(t_i \mid t_{i-1})$ and a matrix b for $P(w_i \mid t_i)$.

If we train our model and we test it with the appropriate train a data corpora, we obtain an accuracy of 92.41 % per word and 18.74 % per complete sentence.

4 Analysis

An interesting experiment is comparing how the accuracy is affected as the available corpus is reduced. We present the results of both the per-word and per-complete-sentence accuracies plotted in Figure 1 and tabulated in Table 1.

As we can see, while the word accuracy ranges from 84.26 % to 92.41 %, the sentence accuracy ranges from 3.74 % to 18.74 %. They show a strong dependency, supported by a $R^2 = 0.9091$ between both the accuracies. The given corpus is enough to obtain good results, but for excellent results more data would be necessary.

If we take a look at the confusion matrix (Figure 2), we observe that the model performs well with most of the parts of the speech, except for interjections (`INTJ`) and particles (`PART`). On the one hand, interjections are difficult to classify because they are really sparse (there are just 14 of them in the entire corpus). On the other hand, most of the incorrectly tagged particles are the *no* particle (also the word *no* in English), which also behaves as an adverb.

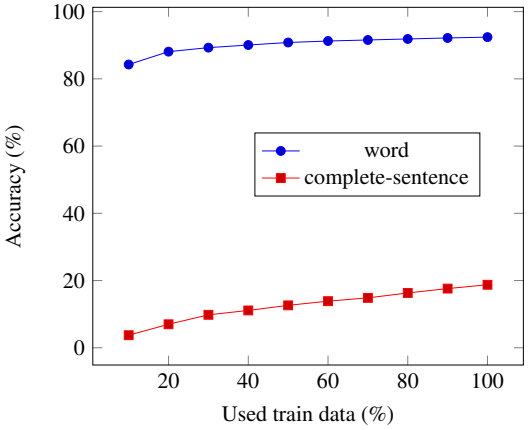

Figure 1: Accuracy over percent of used train data

Another interesting fact is the way that unknown words —or *out of vocabulary words*— are handled. For simplicity, we have opted to tag all the unknown words with the most frequent part of speech: nouns (`NOUN`) [1].

To finish, we include the longest sentence correctly tagged: "Torres/PROPN ha/AUX recordat/VERB la/DET delicada/ADJ situació/NOUN que/PRON viu/VERB el/DET sector/NOUN després/ADV de/ADP l'/DET incessant/ADJ increment/NOUN de/ADP preus/NOUN del/ADP gasoil/NOUN i/CCONJ ha/AUX advertit/VERB que/SCONJ les/DET properes/ADJ manifestacions/NOUN ,/PUNCT previstes/ADJ per/ADP al/ADP 20/NUM d'/ADP octubre/NOUN ,/PUNCT s'/PRON avançaran/VERB al/ADP setembre/NOUN si/SCONJ el/DET Govern/PROPN de/ADP l'/DET Estat/PROPN no/ADV atén/VERB les/ADP seves/DET demandes/NOUN o/CCONJ si/SCONJ continuen/VERB pujant/VERB els/DET preus/NOUN ./PUNCT"

References

[1] S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009. ISBN: 9780596555719.

[2] Dan Jurafsky and James H Martin. *Speech and language processing. Vol. 3*. 2014.

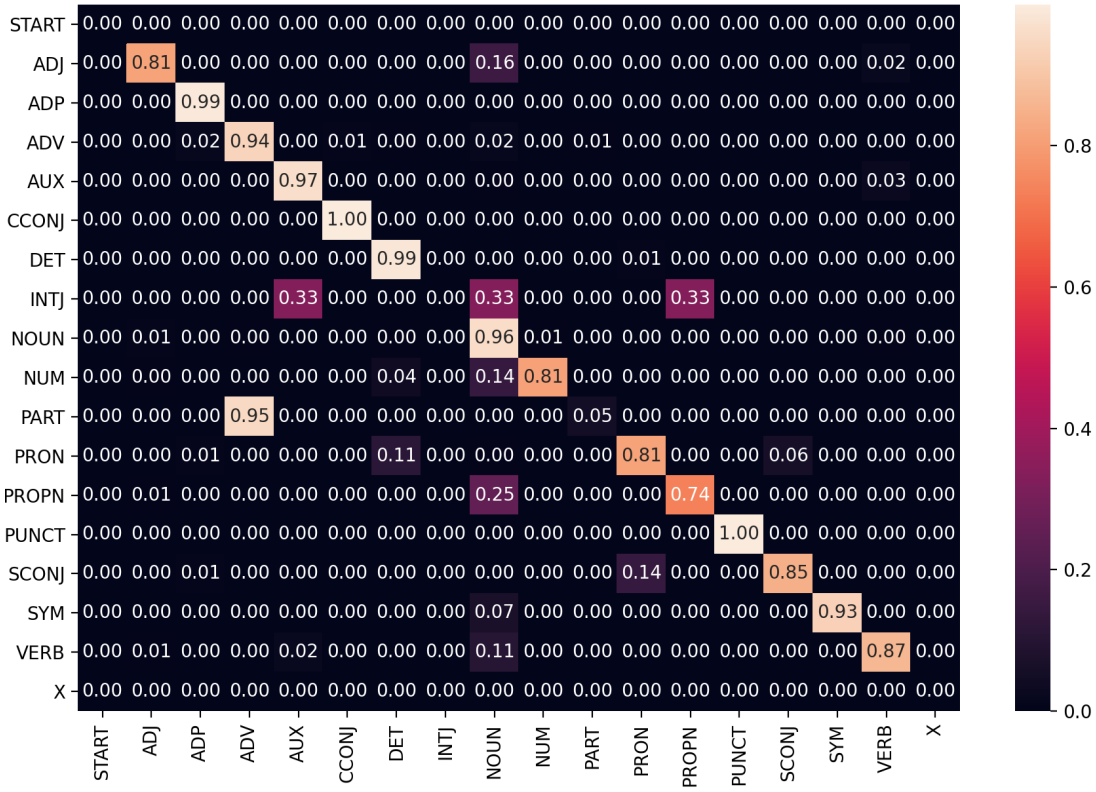

Figure 2: Confusion matrix. True tag over predicted tag. All the test data is used here.

Used tr. data (%)	**Word acc. (%)**	**Sent. acc. (%)**
10	84.261	3.738
20	88.091	6.988
30	89.284	9.804
40	90.063	11.105
50	90.816	12.622
60	91.253	13.867
70	91.564	14.843
80	91.866	16.306
90	92.161	17.606
100	92.411	18.743

Table 1: HMM accuracy given a percent of used training data