POS tagging

Nil Biescas, Joan Lafuente and Xavi Soto

February 26, 2024

1 Exercise 1

In our first task, we explored different ways of stemming the Firefox text to see how it affected the size of the vocabulary. To begin, we displayed the original size of our vocabulary, which was 9300 words. After that, when we passed it through the different stemmers, we noticed a high reduction on the vocabulary size:

Porter Stemmer vocabulary size: 5847

Snowball Stemmer vocabulary size: 5702

We can see that when using the snowball stemmer the size is lower, this is because although they work very similarly, the Snowball stemmer is more aggressive than the Porter due to its broader set of rules and language-specific optimizations, leading to heavier truncation of word forms.

Additionally, one important thing to do when analysing word frequency, is to remove stop words. This is done because stop words like "the" or "and" are vastly used making them always be the most frequent ones, despite not providing any relevant information about the specific text, the specific stop words can be seen on the notebook provided. We can see that after removing them from the original vocabulary, we reduce it to 9187. Additionally, we can add some specific words due to a high presence in the text, while providing irrelevant information as the stop words, for these reasons, we decided to remove also the word "I".

After we remove all the words that we consider not to be relevant, we computed a frequency distribution of the words and created a word cloud for them, which can be seen in Figure 1. As we could expect, being the text about Firefox, the most frequent words are Firefox itself and some related words to it.

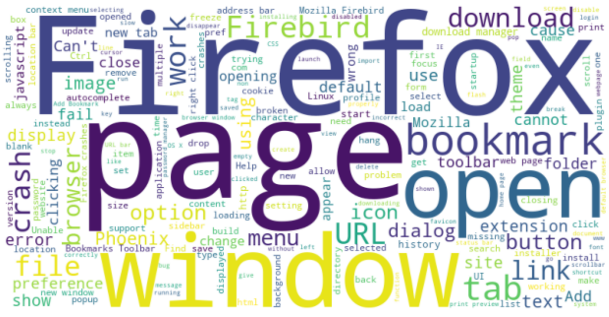

Figure 1: A word cloud of the Firefox text, with the stop words removed.

2 Exercise 2

For the second exercise, we trained a Random Forest model to perform Part-of-Speech (POS) tagging, analysing both its performance metrics and the most important features for the model while using it on the *treebank corpus*.

To train the model, the first step was to separate the sentences on the text and then tokenize them. At this point we decided to add start and end of sequence tokens to the sentences. Then we needed to extract the token features, for this we used the ufeatures function previously provided to us, facilitating the subsequent analysis. Although we tried doing some modifications on the features, at the end we kept the original version as there was not any noticeable variance in the performance of the model with our changes. To obtain the ground-truth, we used the universal tag set already existing in the NLTK Treebank.

Once we had the text features extracted and the ground truth, we separated the corpus into train (80% of the tokens) and test (20% of the tokens) sets and vectorized the features.

When training the model we wanted to use the optimal parameters, for this reason we evaluated the performance of the model for different number of estimators and maximum depths of the trees, using cross validation. Out of the parameters tested the best performing were not limiting the depth of the trees and using 100 of them.

2.1 Results

After our model was trained, we predicted the POS tags of the test set, making a classification report for each of the different classes. To perform this evaluation we have not considered the performance of the two tags created by us ($' < s >', ' < /s >'$). There is a summary of the results in Table 1. More detailed metrics, for each of the different tags, can be found in the attached notebook. From the results we can see a good performance of the model on those common tags, and worse performances on those rarely occurring tags.

	precision	recall	f1-score	accuracy
macro avg	0.88	0.84	0.85	-
weighted avg	0.95	0.95	0.95	-
accuracy	-	-	-	0.95

Table 1: Summary of the results, on the 20183 test tokens, for the model trained.

Moreover, we show some qualitative results for some random sentences and compare them with a prebuilt POS tagger obtaining the same results, predicting correctly the POS tag. These examples can be seen in Table 2. Furthermore, we created a scatter plot with annotations to visually represent each word alongside its predicted tag for one of the above sentences which can be seen on Figure 2.

Sentence	The	cat	is	sleeping			
Our model	DT	NN	VBZ	VBG			
NLTK Perceptron	DT	NN	VBZ	VBG			
Sentence	The	dog	is	barking	at	the	mailman
Our model	DT	NN	VBZ	VBG	IN	DT	NN
NLTK Perceptron	DT	NN	VBZ	VBG	IN	DT	NN
Sentence	She	danced	gracefully	under	the	moonlit	sky
Our model	PRP	VBD	RB	IN	DT	NN	NN
NLTK Perceptron	PRP	VBD	RB	IN	DT	NN	NN

Table 2: Comparison between our model and the NLTK averaged_perceptron_tagger

2.2 Feature importance

The last step of this exercise was to measure the importance of each of the features used. To perform this evaluation, we have used the mean decrease in impurity (MDI). In Figure 3 can be seen the importance of each feature as well as the standard deviation of these importances in the different trees of the ensemble.

We can see that the most important feature is suffix3, which contains the last 3 characters, closely followed by the entire token importance. The following most important features are those

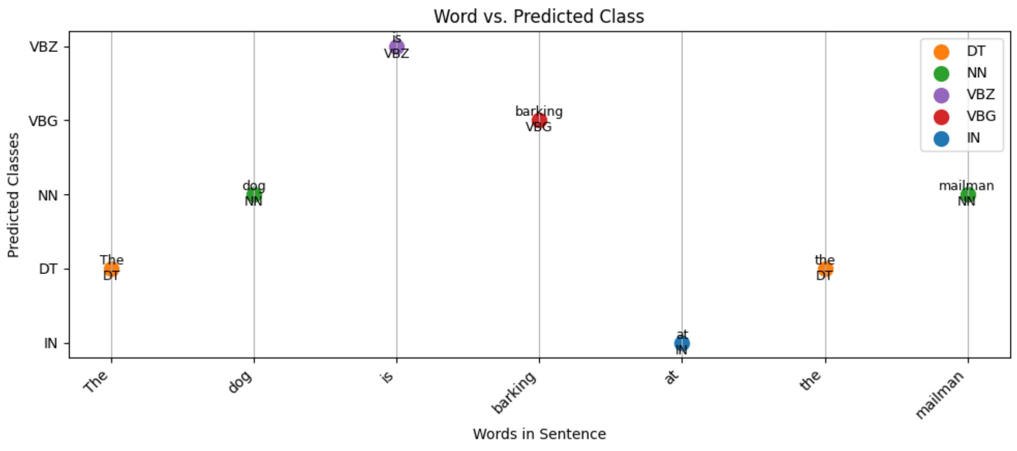

Figure 2: A visual representation of the tags predicted by our model.

that contain the different suffixes and prefixes of the word. After that we can see the context words, next and previous, and finally the rest of the features which, considering MDI, are much less relevant than the other features. So, as we could expect, the word itself is the most important factor for the model.

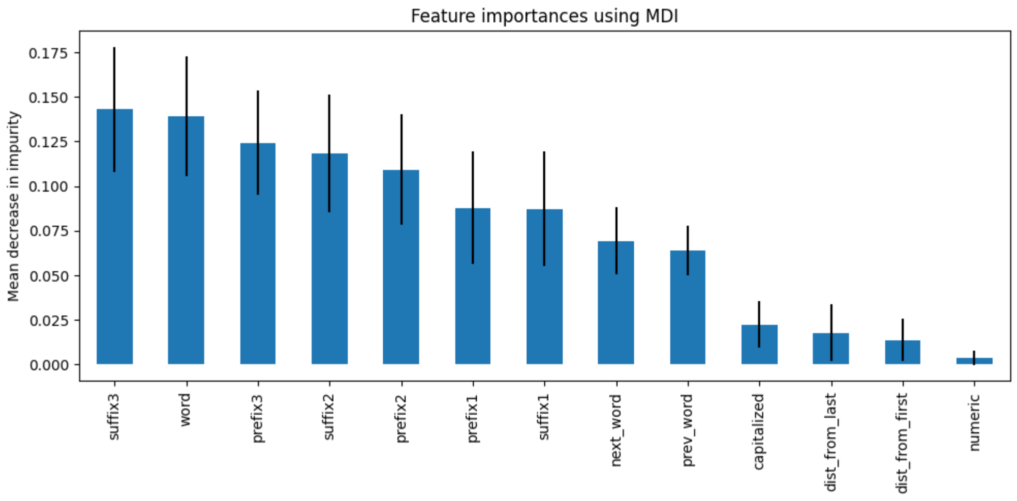

Figure 3: Feature importance using mean decrease in impurity, sorted from more to less importance.