JADH Presentation

What is a word-vector

The main idea of vector representation of terms is that each word is not only defined by the characters combined in it, but essentially **by the company it keeps**. That is to say that a word is defined by its context, which helps vectorizers define what its probable context should be, and what it should'nt be.

Basically, on a most simple level, a word is represented by the presence of its **cooccurrents** in a given **moving window** (usually 5 to 10 words before and after the word). That is to say that if a certain word appears 3 times in the context of a word, the value at its index is 3. Each word is then represented as a vector, that is to say a **series of values** at different indexes of its cooccurrents. Based on this most simple definition, the vectorizer we used, word2vec developped by google, is capable of **predicting probable context** from word vectors already built (skip-gram model), and the other idea of word2vec is to maximise the similarity (dot product) between the vectors for words which appear close together in text, and minimise the similarity of words that do not (**negative sampling**).

How do we define vector similarity

Each word can be represented on a semantic space of *n* dimensions. For each word we can define an **angle** that separates it from all the other word-vectors. The **cosine similarity** helps us define the angle proximity between different vectors. For example, on a semantic space, the word "littérature" is more likely to have a higher cosine proximity to a word like poetry than "astrophysics" : that is to say that the angle between the vector representing literature and the vector representing poetry is smaller than the one between literature and astrophysics.

Why are those vector similar ?

The main idea is to say that if a vector is **similar** to another, it's used in aproximately the same way. That's not to say that these words are equivalent, but that they are relatively close in the vector space.

Diachronical vectors ?

Logically, if we train vectors on different slides of time, lets say 20-year slides, we should be able to **map the evolution** of the vectors through time, as the use of the words, and thus the vectors, is likely to change. That is what we did with word2vec. Word2Vec can have its flaws (if you find it useful we can discuss it afterwards) but it's very effective for two reasons : first it is capable of handling a **very large amount of data** which is not so important for the French corpora, but which takes quite another dimension for the BPO and any Hathitrust corpus), but even more important for us, it is language independant : this aspect was essential for our experience as we had to be sure we would use the same methods for both languages.

Intro to the code

On the code open-source that you see here (and that can be downloaded from github), you can decide which model for which corpus you want to use for your analysis : the only thing you need is a set of vectors on the model of the word2vec ones : you can **easily adapt** your own vectorizer to these models. Here by default, we use the critical corpus which Alexandre presented within the years 1820-1940, as well as the BPO which JD presented. We tested over 10-year slides, it's clearly much less effective due to the small size of the French corpora. After the load, put the path to your directory of models, and then select your range in "year in range".

Similarity of a list of key terms to "littérature"

The cell below (which calls the methods above), shows how two words differ from one another through time, with **cosine similarity**. Here, we show how a list of selected concepts evolves compared to "littérature". You can manually change both in the code as well.

At this point, we'll do the same thing as above, and calculate, for each token in the 200 nearest terms to the main entry, the proximity of this term and its significance. The significance is calculated with the **p value**, that is to say that, below a certain threshold (0.05) we have a strong likelyhood that the result is sure and significant.

Increasing

In this part, we want to show what terms **emerge** more and more with the main entry, that is to say each word of the given test list. The "**slope**" is the degree of progress, and the **"p" value** its efficiency. So here, the main emergence with "littérature" which is significant is "humanisme". All terms seem to be significant, except "fédéralisme", "welschinger", "maniere", "hennet", "réapparition", "deffence", "bourgin", "colonie", "naturalisme", "réalisme", "sillery", "gréco", "compétence", "symbolisme", "catholique", "japonais", "manuel", "romand", "topographie, "organisme", "prédominance". That is to say that those terms can be nearest, but that statistically they are not significant enough to be sure, while the others are more certain.

In this following cell, we show how the top ten most similar vectors change through time compared to the words in the test list : "humanisme" for example seems to be very rare before 1860, and then becomes more and more similar to "littérature". Those show the terms that **were not similar in the beginning, but tend to be more and more related** to "littérature". You should keep in mind the p values associated to each vector.

Decreasing

This is the same process here : we want to see which terms tend to **disassociate** themselves from "littérature" and other terms in the list above (by default, which you can change with the trained models). Then again, you have to check the p values. For « littérature » for example, "transplantation", "choeur" and "philé" are not considered significant, "chaldéen" is, and "destination", "morceau", etc. are as well. The fact that those are less significant is logical : the fewer the terms, the more erratic their series tend to be.

Intersected neighbors

In this part, we show which significant terms tend to be, through time, the **persisting nearest neighbours** to the main entry. That is to say that these vectors follow the **same evolution** through time as the main entry and are very near to the "littérature" vector. At this stage, we only ask for significant terms (filter above : "if fit.pvalues[1] < 0.05").