# Algorithms in Data Science: Dracula

# Text Pre-processing

In order to be able to apply most algorithms to any kind of text, a pre-processing step is necessary.

Throughout the report, we will use different kinds of division for the book, sometimes by blocks, sometimes by chapter, but the pre-processing step will always be needed and will always be the same.

Raw text can be very difficult to interpret due to many factors, the most important being the fact that some words are basically the same but are written differently. For example, car and cars should be considered the same word in most cases, the same goes for Car and car.

First we are going to reduce every word to lowercase, this is self-explanatory.

Then we will remove punctuation and most special characters, including eventual HTML tags (this one is done for reproducibility on other documents since our particular text doesn’t contain any).

Finally, with our text free from capital letters and punctuation, we can proceed the the stemming of words. Stemming is the process of reducing a word to its root form, so that similar words can be considered the same by our algorithms.

# Topic extraction : LDA and NMF

## Topic extraction

Topic extraction is a machine learning statistical model that aims for discovering abstract topics in some documents. We can easily interpret it, a document had a topic, and if it is from that topic it means that some words will be present in the said document more often than others. But a document can have multiple topics, the goal here is to find those topics.

## LDA

Latent Dirichlet Allocation, often referred to as LDA, is a topic extraction technique based on a statistical model (probabilistic) almost equal to probabilistic latent semantic analysis that uses a Dirichlet distribution. This distribution helps to introduce the fact that a document only cover a small amount of topics and that a topic only contains few frequent words.

LDA takes into input a bag of word matrix where rows are documents and columns are words. It then outputs two matrices :

* The document to topic matrix
* The topic to document matrix : It is often used to display topics by taking the n words with highest weights ( = most frequent into that particular topic)

## NMF

Non negative matrix factorisation is an algorithm based on linear algebra where we factorize a matrix into two matrices. All matrices must be non negative, which explains the name given to that algorithm.

The non negativity property makes interpretation more intuitive.

NMF also takes a bag of word matrix as input and outputs the same two matrices (same function but different results obviously).

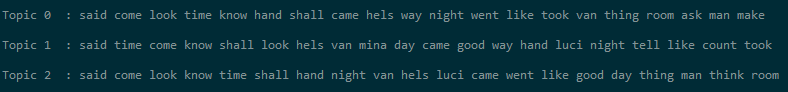
## Project

We are working on the book Dracula, so we are going to proceed this way :

* Pre-process the data as explained before
* Divide the document into chapters
* Apply LDA to our documents and then try to extract topics.
* Try to interpret results and maybe go further

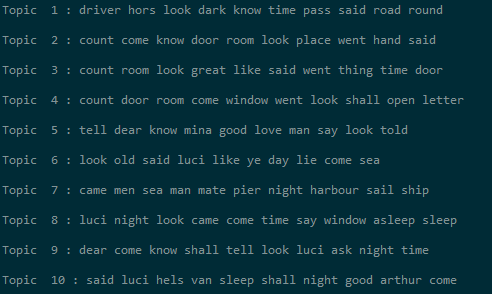
For LDA (same goes for NMF), we need to specify the number of abstract topics we are looking for.

After a few tries, 3 topics seemed like a good option so we will use this one. For each topic we are using the topic to document matrix to display the words with highest weights in order to help us interpret the results.



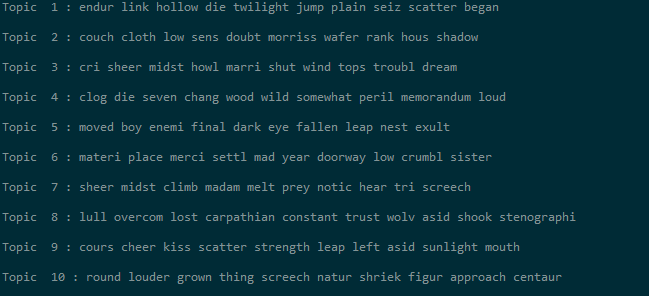
Trying to interpret these results is still hard since they are very similar. In all topics it seems like people are talking (said), which is understandable given that our data is a book but they are so similar that we almost would want to extract only one topic from the book.

After these results we wanted to try something else, in order to get a better insight on the different topics of the book, so we are going to try LDA again, but this time we will use it on each chapter individually, extracting exactly one topic per chapter. This way we hopefully will capture some better information about how the different parts of the book are articulated. We will only show the 10 first topics in this report for a matter of space, but others can be seen with the code.



This looks a lot better, we can now derive some information from the chapters, and we could imagine for example trying to guess what a given chapter is about based on these. Chapter 4 as an example could be about a letter, or chapter 5 about Mina (Dracula’s loved one).

Another algorithm that is often used in topic extraction is NMF, we are now going to use NMF to compare the results with LDA.



One interesting thing to notice is that topics generated in chapters from NMF are a lot more diversified than LDA. Words with higher weights here are a bit more specific, making the understanding of the differences between chapters easier, yet a bit messier than LDA.

The results between the two techniques differ quite a lot. This might be explained by the Dirichlet distribution in LDA that assumes fewer frequent words.

# Graph representation of characters

## Relation between characters

To approximate the relation between character, we are going to count the times where a character is mentioned in the same document as another one. So for example, if Dracula mentions Mina in the book, we will increment the relation [Dracula => Mina] of 1.

Given that our data is a whole book and not some well divided documents, we will need to divide it ourselves. But dividing by chapters here might seem like a very bad idea since chapters are very long and therefore a lot of characters will appear in every chapter.

To avoid this, we are going to divide the document in blocks of N lines. (here 10 lines).

The result is a matrix counting the frequency of two characters being mentioned together, this matrix will estimate the link between main characters.

For graph drawings, we will use a second matrix that is essentially the same but with the introduction of a threshold, where if a value is inferior to the threshold it is assigned 0 as value, this way we only draw pertinent links.

## PageRank

Pagerank is originally an algorithm used by google to generate a ranking between web pages. It mesures the poopularity of a web page but pagerank is not the only algorithm used in this case, it is just one indicator used to order the results of a research.

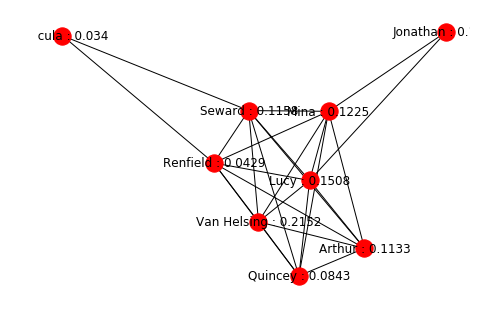
It works by assigning each page a score proportional to how many times an user would get on the said page by cliquing on random links all pages.

Therefore a page is linked by a lot of other pages is going to have an high score. We can say that pagerank is a random walk on the graph. The output is a probability distribution of the likelihood that the user gets on a said page by randomly clicking.

The algorithm therefore calculates which pages are the most "central" ones, in our case it can be interpreted as which characters are the centrals ones, and are connected most to other main characters.

It means that we can use it to make a ranking of what are the most important characters throughout the story.

For PageRank, we will use the matrix without threshold, so that all links are included in the calculations.



(Upper left is Dracula and value for Jonathan is 0.1213)

Advantages: we can represent most important characters of the story very easily, it is a fast and iterative algorithm that converges to a solution.

Cons: As we can see, if a character is not directly linked to other characters it won't have a high score. This is easily explained by the fact that PageRank is based on the links between characters. But in our story Dracula is the main "bad guy" of the story and yet is ranked last... In our case it also means that if a character refers to another one as “my dear” or a similar formulation, the algorithm won’t take that into account.

Why? Well it is simply because he rarely appears in the book and therefore is not connected to a lot of other characters. So if a character is important but stays away from the other main characters it will not be ranked as high as he normally should be. We can add to that the fact that the author writing style

Although this gives us a good estimation on the written text itself, informing us that Dracula is not appearing a lot in the book. Without even reading the book, a lot of information can be extracted from this graph. First off, the main character isn't Dracula as the name of the book suggest, but the book is more about Harker and Van Helsing.

Second, Dracula is very absent from the books, he has some connections with Seward and Van Helsing (who is tracking him). But it doesn't seem like he is omnipresent.

# Simplified search engine

To go further, we thought that we could use some matrix properties in order to create a simple search engine for the book.

## Tf-idf cosines

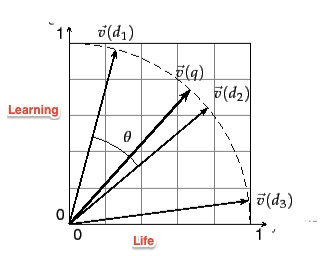
To create this section, we will use the tf-idf cosines similarity then return the highest value as result for the search, which is the most similar document according to the algorithm.

* Term Frequency (tf): Vector measuring the number of times a word appears in a document.
* Inverse Document Frequency (idf):

1 + log(Number of docs/Number of document with term in it)

The cosine similarity is then simply given by multiplying TF by IDF.

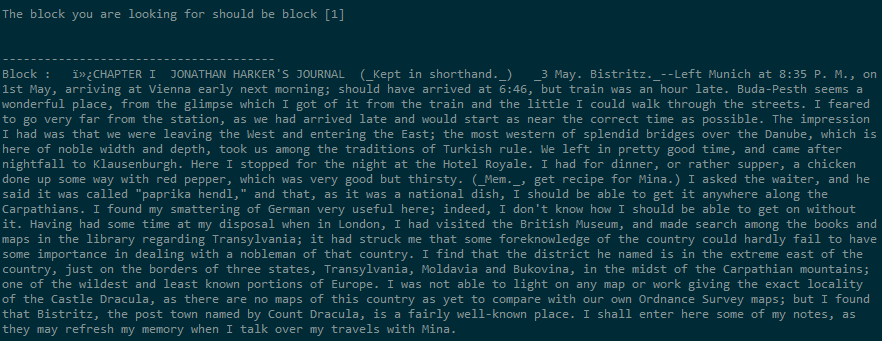
The goal here is to convert words into numbers that we can work with. Imagine a n-dimensional space with vectors for each documents representing their attributes. The document vector closest to query vector is going to be the most similar one. It means that the angle between them will be small, and as angle gets smaller, the cosines gets bigger. This is why the tf-idf cosines can be used as a measure to rank document by similarity to a query.



To try out our algorithm we are going to make a simple query that is related to the beginning of the book, hoping that it returns the first block of text as output.

Our query will be "I stopped at hotel royale, we had a good diner and i then visited a museum."

The algorithm outputs :



Which is exactly what we expected. The tf-idf cosines similarity is simple yet it proved to be quite efficient when it comes to search for some particular information in a book.

We can easily see the use of such algorithms, making the search into unread books much easier. However we can point a limitation (at least in our implementation): it doesn’t take synonyms into account.