How to make the best of text analysis in law practice?

Lily Renard and William Lamazere, MSc Data Analytics & Artificial Intelligence
Master thesis supervisor: Professor Frank Fagan
May 15th, 2019

EDHEC Business School does not express approval or disapproval concerning the opinions given in this paper which are the sole responsibility of the author.

We certify that: (a) the master project being submitted for examination is our own research, the data and results presented are genuine and actually obtained by me during the conduct of the research and that we have properly acknowledged using parenthetical indications and a full bibliography all areas where we have drawn on the work, ideas and results of others and that the master project has not been presented to any other examination committee before and has not been published before.

Abstract

This master thesis addresses the different applications of Natural Language Processing in the French law. The links between law and data analysis are very recent even though the business applications are numerous. This master thesis focuses mainly on topic modelling and word embedding on unlabeled data. The corpus of texts has been scrapped from Legifrance, the French official website for the publication of legal documents: it is composed of decisions of French highest private decision court, that is the “cour de Cassation”.

Special focus is made on visualisation of high-dimensional data and how to detect information from it. As well, the authors put forward some possible business applications through the above-mentioned technics, whether there would be aimed at litigants as persons, law firms or legal departments in companies.

Table of Contents

Abbreviations & choose of words

- q.v = quo vide (to refer to another part of the thesis)
- parameter = when the value is estimated by the model itself
- Hyperparameter = the configuration of the value needs to be chosen by the user itself
- M&A = mergers & acquisition
- NLP = Natural Language Processing

Problem statement

There are two main approaches to conduct an analysis on data. Either in a tabular way (which we call structured data) meaning we have information by observation (row) and by feature (column), on which you can directly apply a toolset of algorithms, or either on unstructured data. Real-life work on data always belongs to the latter category as information is never "naturally" found in a matrix, but rather as pixels, observations retrieved from sensors or text.
Analysing text is not something new, as the essence of AI studies has been since its inception to mimic the way people speak and interact. One can even trace it back to Turing's paper (q.v bibliography) in 1950 Computing machinery and Intelligence which led to Turing's test, still relevant nowadays, that aims at distinguishing whether the conversation is with a computer program or an actual human.

What is however more recent, even though it can be traced back to the early 1960' (Lawlor 1963), is the interaction between law and data analysis. And before talking about how these two worlds are now linked to each other, one might first consider why this interaction is not so obvious.
First of all, text analysis is a sub-domain of *Natural Language Processing* (NLP) which is a discipline at the crossroads of linguistics and computer science. Simply put, we are trying to make a computer understand how humans speak and try to replicate it - for the most advanced forms. Yes, the virtual assistant software Siri (known for its release on IOS by Apple in 2011 and was by the way originally designed for military purposes - q.v bibliography) is the state-of-the-art in the human-machine field of NLP, based on convolutional neural networks even before they became famous. In fact, in this field, the easier the interaction human-machine seems to be and the more difficult it becomes. Why is that? Well, it is because this reasoning associates easiness with what is innate. But the concept of speech, comprehension and transmission is more human than anything else and teaching it to a machine is nothing but easy.

And this is why analysis on judicial decisions makes sense and for many reasons:

- In the law, there is a huge amount of data,
- In France, October 7th, 2016's law (article 20 and 21) states that all judicial decisions shall be open, and for free, to the public (q.v bibliography),

- Law texts have a structured syntax and the writing is codified, which helps finding regular patterns and therefore focus on specific parts of texts like the decision or do topic modelling.

What are the possible applications of NLP in law? We found a great number of them:

- Text classification for companies that are trying to reorganise their digital folders,
- Text summary in case of large corporate actions (Mergers & Acquisitions, Fundings, these important steps in a company's life require the readings of hundreds of Pages, very rarely read by the decision makers, but rather by interns…),
- Decision prediction,
- Translation, to our knowledge there is, to this point, no company placed on translation of legal documents (and it is a task that takes a lot of time of law firms, requiring the help of professional translators who often have no expertise in law),
- Anomaly detection, so as to notice if small inconsistencies remain before signing a legal and complying act),
- Etc...

How is that possible? This paper aims at tackling this question. We will first explain how we chose a relevant corpus of text and the methods to pre-process textual data. The second part will focus on the different techniques to transform textual data and their applications.

Part 1: How to pre-process textual data

A. Chose a relevant corpus of texts

How to access Legal texts in France ?

There are plenty of sources to analyze on the internet and the little experience we have had in working in Legal departments and Law firms is that a majority of the time spent is for searching online for the source that will confirm one's thought or infirm it.
There is the official government website Légifrance that contains the legislative and regulatory texts and court decisions of the supreme courts and appeals under French law. Naturally, you would not make much of this source if you were a law firm or a legal department in a company as:

- first instance decisions are missing for a large part in this database, and
- these entities alone would be totally lost without access to what the best specialist have written on very specific subjects, and this can be found on private sources such as Lexisnexis, Francis Lefèbre, Dalloz…

But of course, you need at one point to have these texts on your local computer/server. If some Application Programming Interface (API) directly exist for text analysis online, like Google Natural Language, surely at one point Google needs to extract the text on its servers and put it into a matrix to run its algorithms.

This is why we had to do some *Web Scraping* as downloading bulk data in a digest way was not possible to do in France. We therefore decided to run a small script that uses 2 main logics:

- The first one was to run several loops, one for fetching all weblinks containing every decision, and one for opening all them and writing them in separate files,
- The second one needed a tool that would support dynamic content, as the texts that appear on the user's interface naturally come from the search query that is sent to the server. We therefore needed another package as Beautifulsoup package was only based on html language (static web development).

Selenium package is mainly used for browser automation and scrapping. As said, we needed some service that would support extraction of dynamic content (based on javascript language).

The syntax of this library is very convenient as you can either find tags written in html and either elements directly as they appear on your webpage. For more information, you can refer at the Python script in Appendix 1.

B. Exploration of the corpus

Once we have scrapped our corpus of texts from Légifrance for 2018 (which corresponds to more than 8700 texts only corresponding of decisions of the cour de Cassation), we want to learn more about the composition of the texts. It would be helpful to analyze the vocabulary used.

In order to do so, we use the *tokenization* process. Tokenization will separate the elements of a sentence, called *tokens*. More precisely we use the function word_tokenize that works as following :

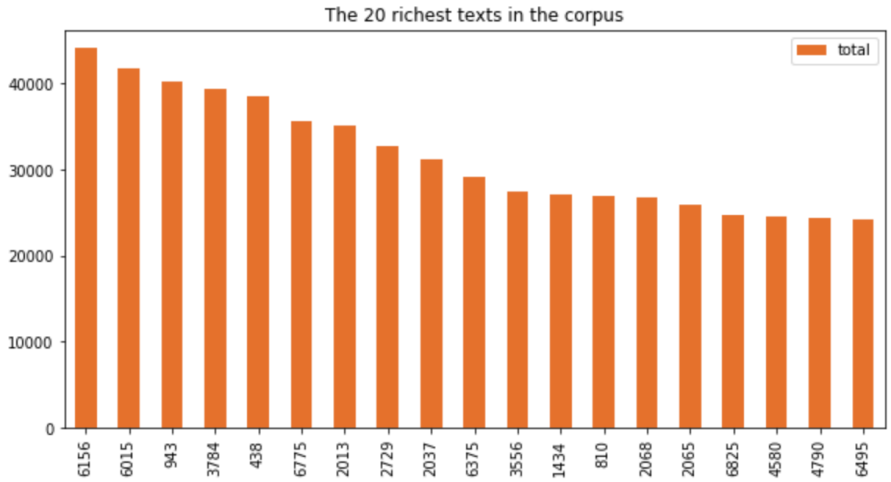
```
In [23]: test = "Bonjour, je sert d'exemple pour la thèse de Master. Essayons !"

         nltk.word_tokenize(test)

Out[23]: ['Bonjour',
          ',',
          'je',
          'sert',
          "d'exemple",
          'pour',
          'la',
          'thèse',
          'de',
          'Master',
          '.',
          'Essayons',
          '!']
```

The function takes the sentence and chops the elements in an array. We can observe several problems to this function: the punctuation is considered as a token (not very useful to analyse the content of a text) and the apostrophes are considered as part of a word and concatenate two different words (“d’exemple ” instead of “de” and “exemple”).

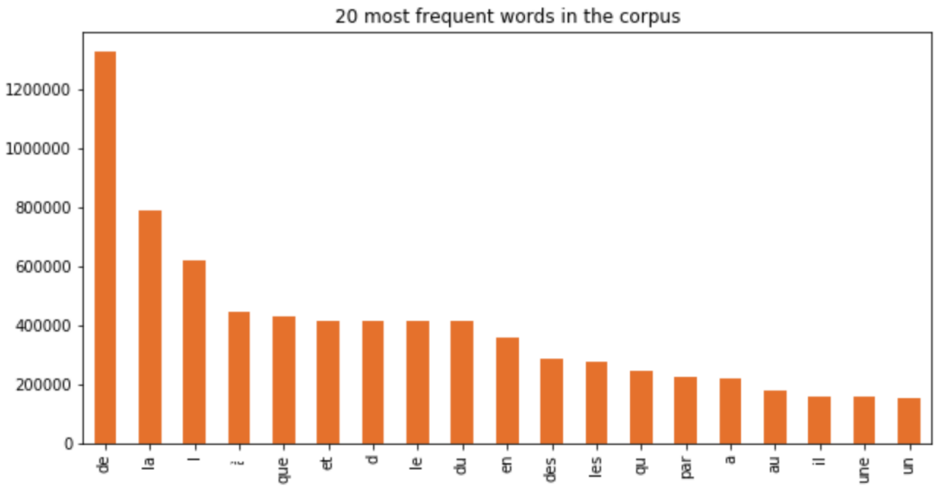
The process of correcting the punctuation errors is called *text normalization*. In order to remove non-alphanumeric characters, using *regular expressions* is enough. Once we have normalized the corpus of text, we create a function that will count the frequency of words as well as the total number of words. In order to calculate the frequency of each word, we use the *bag-of words* model. In the bag-of-words model, we count the occurrences of each word for all the

documents of the corpus. The bag-of-words model will further on be used in document classification and topic classification.

Thanks to this function we can see that our corpus is quite rich in words as the richest text contains about 40000 words. We can also notice that very rapidly, the number of words remains more or less the same, which is quite logical as legal texts are standardized.



When we plot the most frequent words, we can't help but noticing that we only plotted the most common words in French language. As such, the most frequent words do not help us analyse further the corpus and the meaning behind each text. If we only look at that bar chart, we could not even imagine that our corpus is composed of legal texts.



C. Cleaning of the corpus

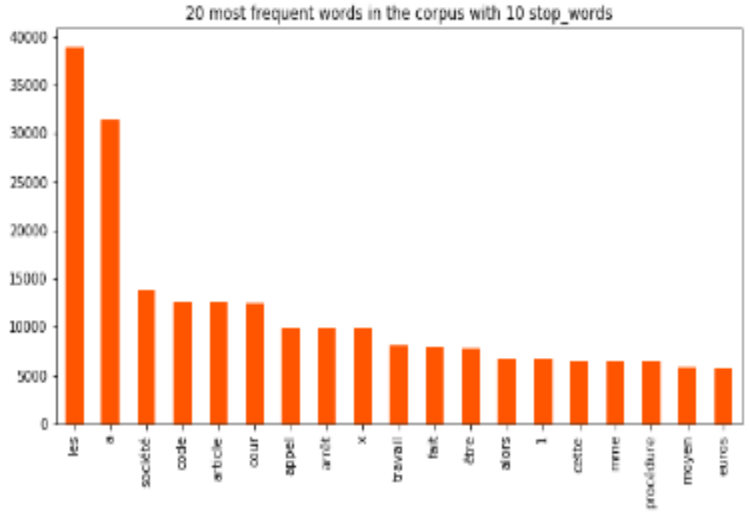
The first step in cleaning the corpus is to remove the *stopwords*. The stopwords are those very common words in a language like the ones shown in the chart above (“de”, “la”... in French) and that give absolutely no information on the meaning of the text. They slow down our work and thus, we want to delete them.

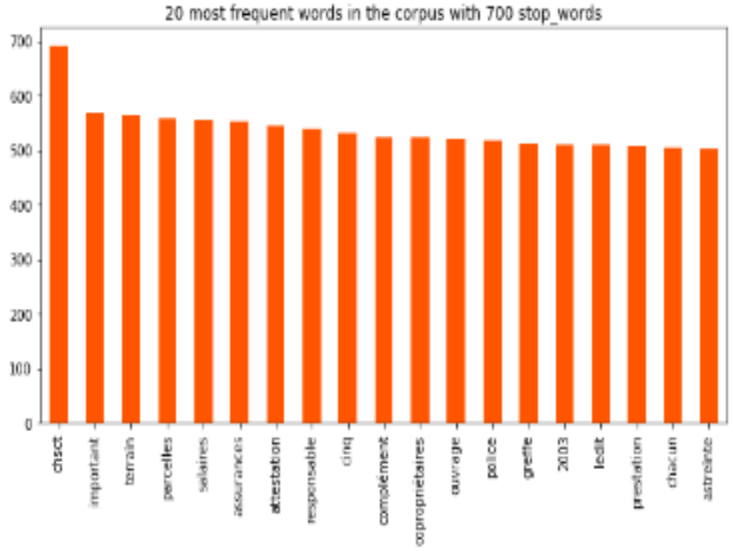
In the NLTK library there is a list of stopwords for each language. In addition to the French stopwords, we are going to remove the 85 most frequent words as well. In our case, the most frequent words in our corpus will be words specific to law such as “procédure”, “article”, “cassation”, etc… By deleting the most frequent words, we assume that they are not meaningful to the understanding of the texts.
Why not delete 10, or 700 or even 2000 stopwords instead of 85? This is in fact a crucial hyperparameter in this whole processus: take a value too important and you might lose too much information (as you remove some words that would helped you to better represent your text), but a figure too low would leave too much noise in your text.
Indeed, trying with different values (10, 700 and 2000) applies various transformations on our corpus, and we clearly see that with the highest value there remains only rare words and even figures: applying any kind of machine learning on this corpus would definitely lead to some overfitting, as generalisation would be quite hard.

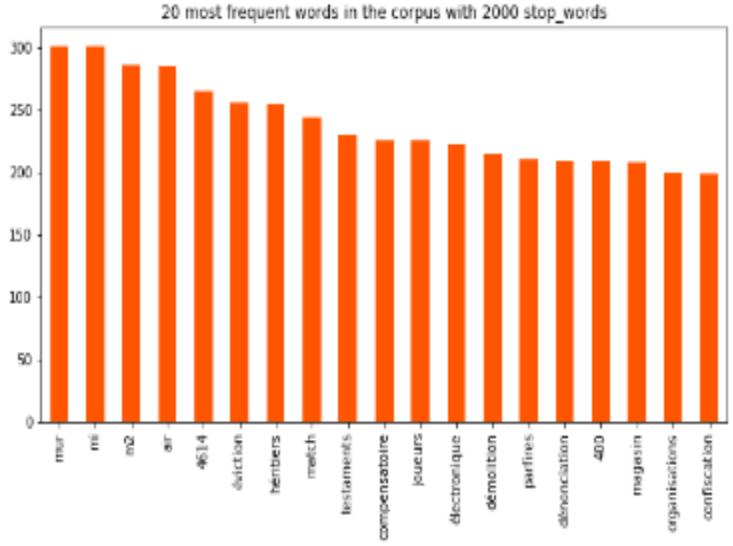
We however see a clear improvement (based on the frequency of words appearing the most in our corpus) when looking at more reasonable values, such as 85 or 700 stopwords removal.

Finally, removing most frequent words is one method, and maybe the most commonly used as the intuition behind is very logic: you considere that words appearing the most in all the corpus don’t add much sense to understand one specific text, if they don’t distinguish it from the others. But one other solution if you know very well your corpus, is to manually create your stopwords list so as to be sure not to remove some very interesting and differentiating words, and get rids of those of which we are certain that they appear almost everywhere. For instance if we are in a corpus dealing only with texts related to mergers & acquisition, one would want to create a stopwords list containing words typically used for this domain (“transaction”, “ownership”, “strategic”,”operating”,”equity”,”assets,”stocks” etc…). What is interesting here is that if we had a corpus containing several sub-domains of law (for instance public law and

civil law only related to M&A), these same stopwords would have been the ones that would have made possible to cluster apart these sub-domains using unsupervised classification techniques (see part 2).







The final step of the cleaning process is *lemmatization* and *stemming*. For both processes, the goal is reduce the derivations of a word to a single one. As an example:

sees, saw, seen → see

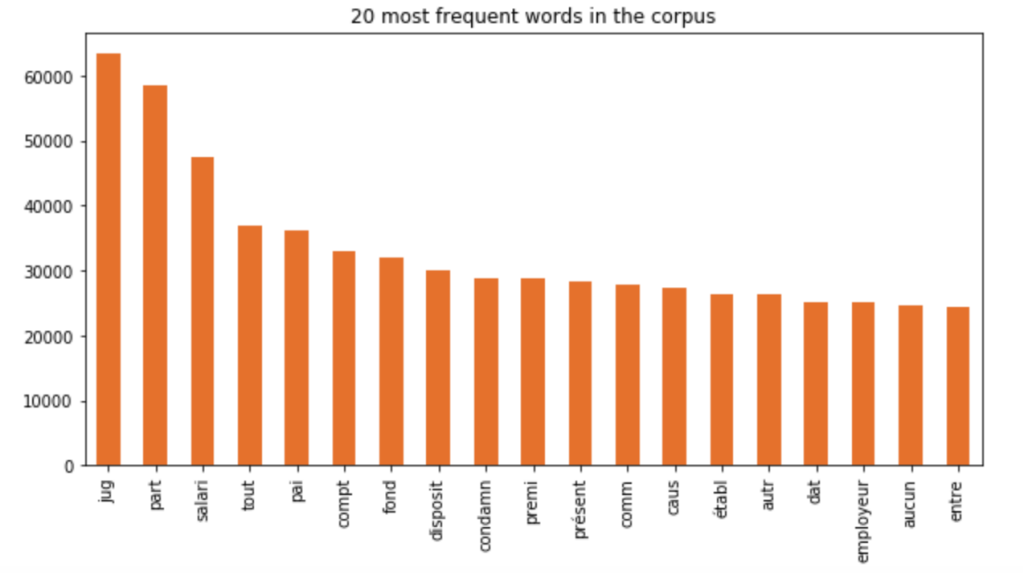
cat, cats, cat's, cats' → cat

Here again, we want to focus on the meaning of a word. The difference between lemmatization and stemming is that stemming will delete suffixes and prefixes to extract the root of the word. As such:

Rule	*Example*
SSES → SS	caresses → caress
IES → I	species → speci
SS → SS	caress → caress
S →	cats → cat

It is less effective that lemmatization that will analyse grammatically the word to associate it to the most relevant form, know as the lemma. For example, with the token "saw", stemming might return just "s", whereas lemmatization will return either "see" or "saw" depending if it's a verb or a noun.

Unfortunately, there is no French lemmatization function in the NLTK library, we decided to use the stemming function. When we plot the 20 most frequent words in the corpus we can clearly see law vocabulary showing up such as "condamn", "compt" etc…



Part 2: Transform textual data

A. Represent the corpus of text as a "bag-of-words"

The *bag-of-word*s approach is very intuitive: we simply consider every Cour de cassation decision as a whole of words, without any consideration of contexte (order, how it is used…). After steps previously discussed of stopwords removal and stemming, we finally aim at creating a term-document matrix in which:

- each ruling will represent a line vector of length size of the vocabulary of the whole corpus,
- each value represents the frequency of the word in the document, and
- there will be as many lines as there are documents in our corpus.

Now that this term-document matrix is created, another step is necessary for this approach to make sense. Indeed, in language analysis it is essential to capture the co-dependencies of the words in a text.

One technique (unused in this thesis though, but worth mentioning), that could add some context to words, is splitting the text in several group of words. This group of n words is called *n-grams*. Is it a way of looking at was is going on around words, as we now do not only consider one word, but two or three or n of these. For example, in the sentence "I write a thesis", you can extract the following bigarms: {(I, write), (write, a), (a, thesis)}.
By doing such, we interest ourselves in the probability of appearance of words according to the context. As an example, the pronoun "I" will have a high probability of being followed by a verb. This probability could be assigned as followed:

$$\mathbb{P}(write|I) = \frac{\mathbb{P}(write, I)}{\mathbb{P}(write)\mathbb{P}(I)}$$

We look at the probability that "write" appears and that "I" precede it.

However, with the n-grams technique, we are looking so far at counting word appearance in one document, without any consideration of what is happening in the whole corpus. A solution to that is to weigh up or down the importance of a word in a text relatively to the whole corpus of texts - if they only appear in this text, it should be noticed and weighed up, and the opposite

if they in fact appear in most of the documents. This technique is called the *Term Frequency-Inverse Document Frequency (TF-IDF)*.

As its name indicates it, the weight is measured as followed:

$$weight = frequency\ of\ appearance\ the\ word \times indicator\ of\ similarity$$

This indicator of similarity is the inverse document frequency i.e. the inverse of the frequency of appearance in all the documents. The goal is then to take the frequency of appearance of a word and weight it by its global appearance.

When applying TF-IDF to our corpus, the result is then a sparse matrix, each row representing an "arrêt" and each column a token of the corpus. The sparse matrix format is very useful for mathematical calculation purposes and more efficient storage wise.

```
<7006x119230 sparse matrix of type '<class 'numpy.float64'>'
        with 3732265 stored elements in Compressed Sparse Row format>
```

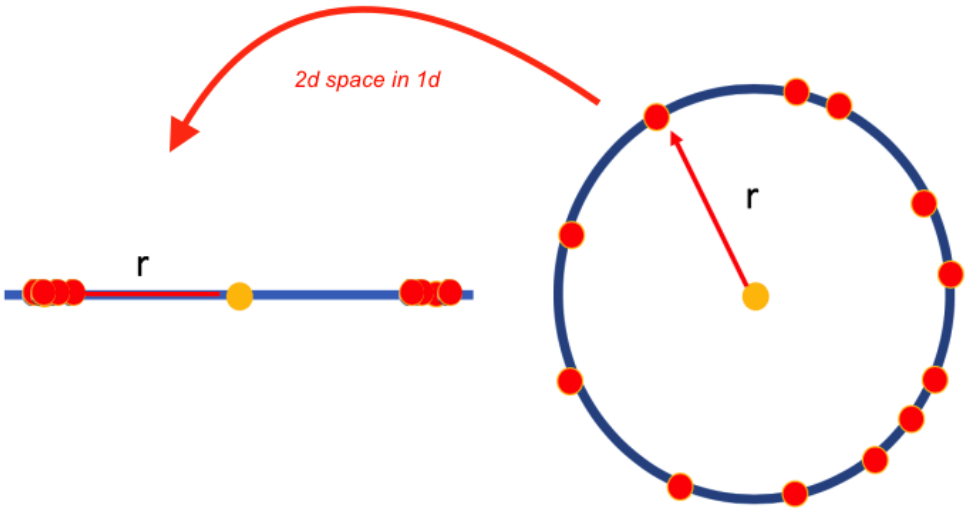
However, the corpus of texts for the year 2018 is way too voluminous for our machines. We decide to make the analysis on a sample size of 1038 textes selected randomly.

As TF-IDF is quite accurate, we can use it to do non-supervised representations, like *t-SNE*, to bring together “arrêts” with similar TF-IDF indicators.

Visualisation is a key tool with unsupervised learning and a key challenge is to have some idea of what is happening in dimensions higher than 3. The 2 researchers Maaten and Hinton introduced in 2008 a way to explore high-dimensional data with a tool called *t-SNE,* which stands for t-distributed Stochastic Neighbor Embedding and that produces a mapping function of your input high-dimensional matrix into a 2 or 3 featured one. This mathematical framework allows to preserve the local structure in this transformation, as many other existing techniques of data dimensionality reduction often squash local particularities and aims more at preserving the global information/variance of the data, like Principal Component Analysis (Hotelling, 1933) which is the most-used dimension reduction technique (q.v bibliography).

Back to our judicial decisions which we aim to analyse through text analysis, the choice of t-SNE makes sense because we remain in a high-dimensional data space (as a reminder not just 30 variables, but hundreds of them as each word-token is a variable) and the *crowding problem* rapidly arrives in these specific data reduction cases. Indeed, this comes from the simple fact that the surface of a sphere grows much faster than its radius (the analogy works in spaces

bigger than 2 and 3 dimensions). This entails that when you place yourself in a lower space, points will be completely stacked to each other.



t-SNE deals with this squasing issue by optimizing the spread-out distance between points that have been embedded in lower space.

t-SNE creates this mapping function in 2 steps:

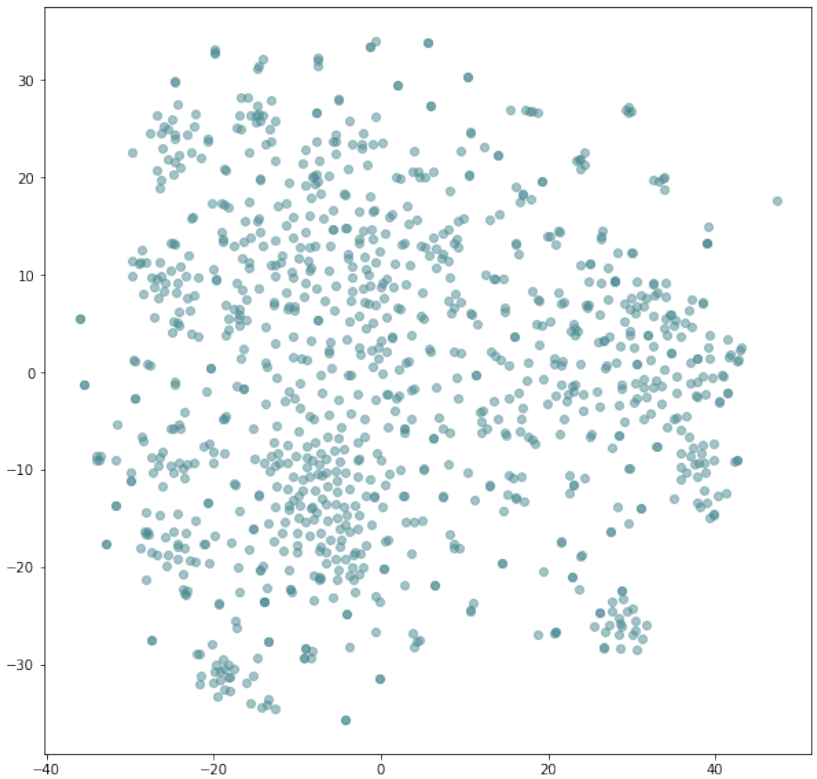
- Step 1, create a probability distribution *Pij* of points in the original data space by assigning to each point *Xi* a probability of being close to a a fixed point *Xj* and based on a gaussian distribution, so that it captures similarity while not assigning null-probabilities (as tails of a gaussian curve flatten very slowly),
- Step 2, create a probability distribution *Qij* in the low-dimensional data space (2 or 3 dimensions) thanks to Student t-distribution with 1 degree of Freedom which is a distribution that has longer tails than normal distribution. This distribution is iteratively optimized through a gradient descent on the Kullback-Leibler divergence (which corresponds to a dissimilarity between two distributions) between these 2 distributions Pij and Qij.

Hyperparameters to be considered are:

- The perplexity (σ_i that appears in the gaussian distribution, we want it to be low in densely populated areas of the space, and high in rarely inhabited parts of the space), which often is between 5 and 50, higher numbers will put more focus on global structure and will give results more like PCA
- Optimization parameters such as the learning rate, the number of iterations (as at step 2 we create a first sample in the embedded low-dimensional space, but there will be iterations with gradient descent), the momentum.

A thorough investigation of t-SNE in our text analysis visualisation could also point its limitations, even though it has overcame many (if not all) of existing techniques such as PCA, SNE (Hinton and Roweis, 2002), Sammon mapping (Sammon 1969), Isomap mapping (Tenenbaum, Silva, Langford, 2000). One drawback remains the non-convexity of the optimization function, so the gradient descent can get stuck in local minima. This is why we say t-SNE is a non-deterministic algorithm as you can run it several times and get different results each time.

A first visualization of our corpus is the following:



We can clearly see some clusters appearing that could correspond to common thematics. We will try in the next part to label these clusters by different methods and try different methods of topic modelling.

We could have used other methods to push further the recognition of patterns and relations within the words (Name Entity Recognition, Relation Extraction, Event Extraction, Part-Of-Speech tagging) before the identification of topics in the corpus but they were not relevant in our case.

B. Topic modelling with unsupervised methods

As we mentioned it before, one application of NLP in law could be topic modelling. Law companies are dealing with a lot of texts and a huge part of their work is to find similar law texts to the current case to evaluate the chances of success. In our corpus of scrapped texts, we are dealing with unlabeled data that's why we will focus on unsupervised methods.

B.1. Latent Dirichlet Allocation (LDA) Method

The first effective method is named *Latent Dirichlet Allocation* (LDA). Let's start with an example to understand this method. We have the following sentences:

- I ate a very good chicken curry with rice at the canteen yesterday.
- I like eating chicken and potatoes.
- I love reading books that are in my bookshelf.
- My sister read a very interesting book yesterday.
- Look this beef tournedos Rossini recipe on this cooking book.

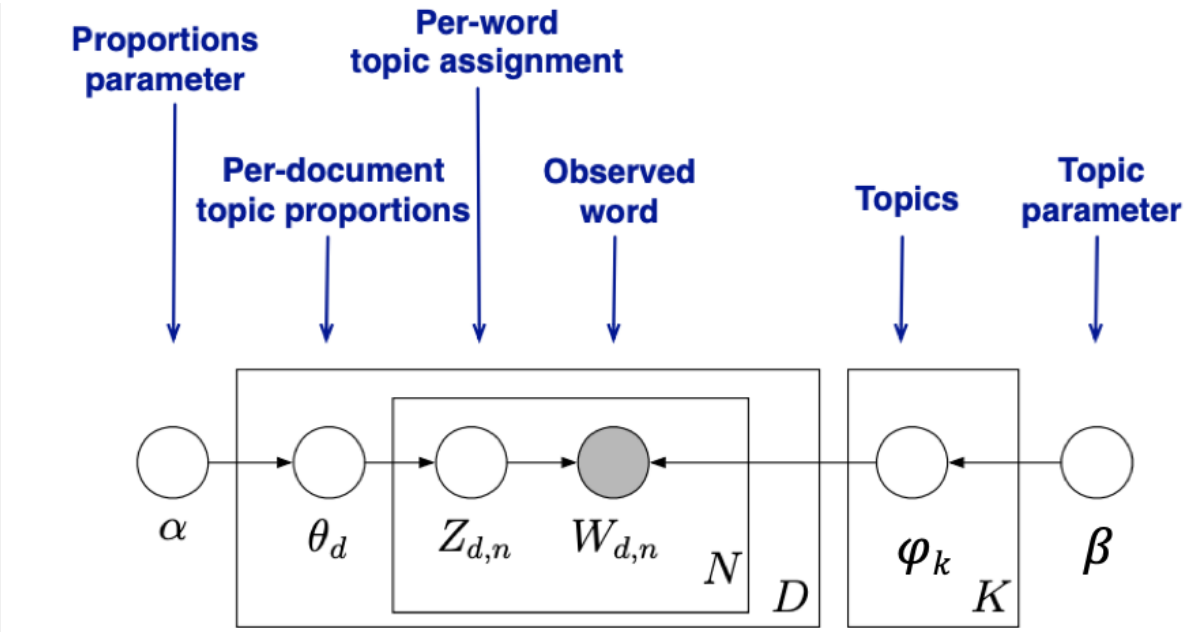
Thanks to Latent Dirichlet allocation, we will be able to know the different topics of these sentences. If we set the hyperparameter - number of topics at 2, we might get:

- Sentences 1 and 2: 100% Topic A,
- Sentences 3 and 4: 100% Topic B,
- Sentence 5: 60% Topic A, 40% Topic B,
- Topic A: 30% chicken, 15% rice, 10% curry, 10% canteen, … Topic A would mostly be about food,
- Topic B: 20% book, 20% bookshelf, 20% interesting, 15% recipe, ... Topic B would mostly be about books.

If we now take a closer look at the theory, LDA, as defined by David M. Blei, Andrew Y. Ng and Michael I. Jordan, is a generative probabilistic model with several hypothesis:

- each document of the corpus is a bag-of-words i.e. the position of the words does not matter,
- each document (m) is a mixture of an underlying set of topics of proportion $p(\theta_m)$,
- topics are identified on the basis of automatic detection of the likelihood of term co-occurrence. Indeed, each word has a distribution associated to a topic $p(\phi_k)$,
- a word (w_n) may occur in different topics $(z_1, z_2, \dots)$, however, with a different typical set of neighboring words in each topic.

LDA can be modelized as following:



With,

M: number of documents

N: length of documents

K: number of topics

α: a sparse vector composed of weights of topics in a document (document concentration)

β: a sparse vector composed of weights of words in a topic (topic concentration)

θ: distribution of topics in document d

φ: distribution of words in topic k

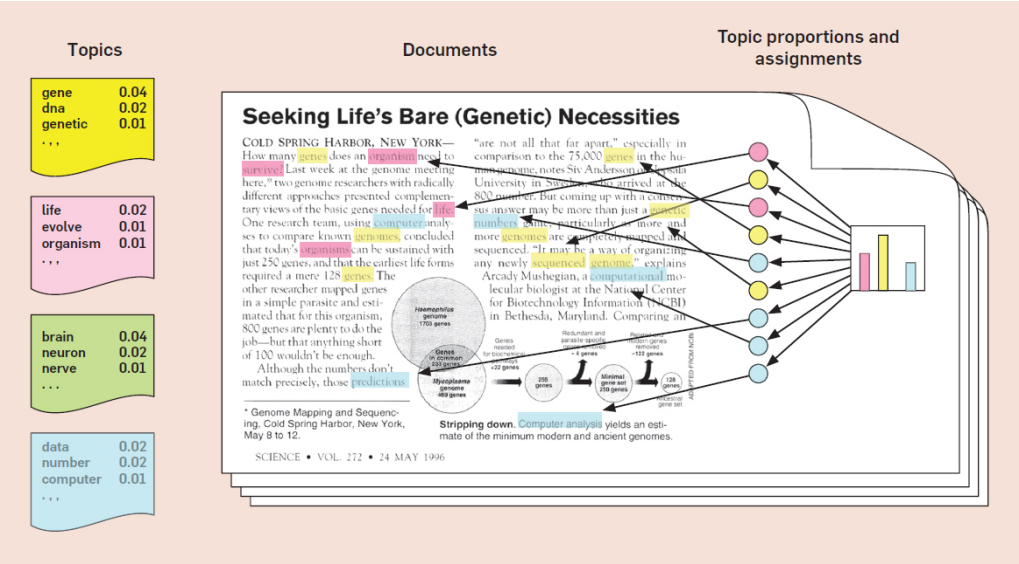
Z: identity of topic of all words in all documents

W: identity of all words in all documents

We can clearly see that there are three levels for the LDA representation. The two parameters α and β are to be sampled once in the process of generating a corpus. The parameters **θ** and d are at the document level, sampled once per document. Finally $Z_{d,n}$ and $W_{d,n}$ are at a word level and sampled once for each word for each document.

To sum this modelisation up, the algorithm will go through the generative process for each document that can be described as followed:

- When writing a document d, the algorithm chooses the number of words N the document will have (according to a Poisson distribution),
- Chose the topic mixture (**θ**) according to a Dirichlet distribution over the fixed set of K topics
- For each word w_n of the N words:
    - Pick a topic,
    - Use the topic to generate the word.



For a corpus of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the corpus.

To modelise this, we will use the LDA topic model algorithm from the package Scikit-Learn. This algorithm uses a word matrix generated by the function CountVectorizer that we have

configured according to the corpus. Most cells are zeros and the percentage of non-zeros in the matrix is very low:

```
Sparsicity:  6.3703052745280555 %
```

We build our LDA model based on this matrix with a predetermined number of topics that we set at 10. It is essential to validate this first model, as it has been selected a priori. We use the log-likelihood and the perplexity to evaluate the goodness-of-fit.

```
Log Likelihood:  -8794964.636971315
Perplexity:  1995.15677693723
```

Our model has a high log-likelihood and low perplexity, indicating it is rather good.

It is also crucial to vary the parameters to identify the best model. In LDA, the core parameter is the number of topics and, in addition, we chose to search the learning rate[1] as well. The results are the following:

```
Best Model's Params:  {'learning_decay': 0.7, 'n_components': 5}
Best Log Likelihood Score:  -3059105.3935975623
Model Perplexity:  2113.297467133195
```

Thanks to this operation, we see that the number of topics in our corpus is in fact 5 with a function of the learning rate which is 0.7 (it is in fact the default value).

Once we know which LDA model is the best, we check the topics allocation. We built a table that underlines in green the highest probability scores of a given topic in a document and thus selects the most relevant topic for the document:

[1]The learning rate is defined on the Online LDA, Hoffman et al., 2010 as “kappa”

	Topic0	Topic1	Topic2	Topic3	Topic4	dominant_topic
Doc0	0	0.08	0	0	**0.92**	**4**
Doc1	**0.23**	0	0	**0.18**	**0.59**	**4**
Doc2	0	**0.25**	0	**0.75**	0	**3**
Doc3	**0.46**	0	**0.54**	0	0	**2**
Doc4	0	**0.32**	**0.25**	**0.13**	**0.3**	**1**
Doc5	0	**0.99**	0	0.01	0	**1**
Doc6	**0.69**	**0.14**	0	**0.12**	0.04	0
Doc7	**0.49**	0	**0.33**	0	**0.18**	0
Doc8	0.06	0	**0.61**	0	**0.32**	**2**
Doc9	**0.34**	**0.39**	0	**0.28**	0	**1**
Doc10	**0.22**	0.02	**0.52**	0	**0.24**	**2**
Doc11	0	**0.98**	0	0.02	0	**1**
Doc12	0.06	0	**0.94**	0	0	**2**
Doc13	**0.12**	0	**0.58**	0	**0.3**	**2**
Doc14	**0.6**	0	0	**0.29**	0.1	0

Below, we check the distribution of topics across the corpus:

	Topic Num	Num Documents
0	4	342
1	0	255
2	2	216
3	1	174
4	3	50

The topic allocation is quite balanced except for the last topic that concerns only a minority of documents.

Let’s check the different topics and their key words:

```
Topic 0:
jugement parties civil vente compte montant 000 date acte paiement
Topic 1:
travaux garantie civil préjudice responsabilité dommages assureur parties faute assurance
Topic 2:
salarié employeur licenciement sociale poste salaire accord entreprise salariés sécurité
Topic 3:
parcelle bail indemnité comme parcelles date activité zone expropriation biens
Topic 4:
pénale instruction 2017 juge tribunal faits mise ordonnance articles examen
```

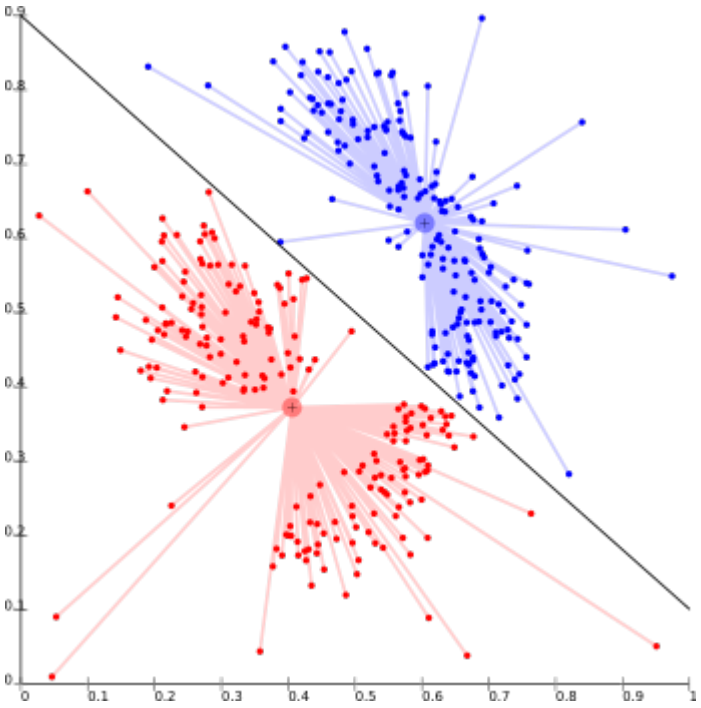
This model seems to be pretty accurate. We can clearly see that the topics put under the spotlight different branches of the law:

- Topic 0 deals with general civil law,
- Topic 1 deals with liability law,
- Topic 2 deals with labour law,
- Topic 3 deals with property law,
- Topic 4 deals with criminal law.

Once we have defined our best model and found the ideal number of topics, we would like to plot the results. To this end, we use cluster algorithm, a basic class of unsupervised machine learning models. Clustering divides the data in groups according to their similarities and their properties. We will use *k-means clustering* from the package Scikit-Learn, as it is the most frequently used.

The k-means algorithm aims to partition unlabeled data in a certain number of clusters, determined a priori. This algorithm has two hypothesis:

- The algorithm defines each cluster thanks to its center, that is the mean between each point of the cluster,
- The algorithm uses the euclidean distance to determine the distance between each point and the center and builds the clusters in consequences.



Coming back to our corpus of text, we use the k-means algorithm with 5 clusters - the number of topics of our ideal LDA model previously found. It would have also been possible to define the clusters with the probability scores calculated in the above table. To plot the graph, we need our two dimensions that correspond to the axis of the graph. To do so, we have to build a *truncated Singular Value Decomposition* (SVD) to reduce dimensionality and apply it to our best LDA model data. Truncated Singular Value Decomposition (SVD) is a matrix factorization technique, very similar to PCA, except that the factorization is done on the data matrix whereas for PCA it is done on the covariance matrix.

Mathematically, we have for a given matrix M, the truncated SVD:

$$M_k = U_k \sum_k V_k^T$$

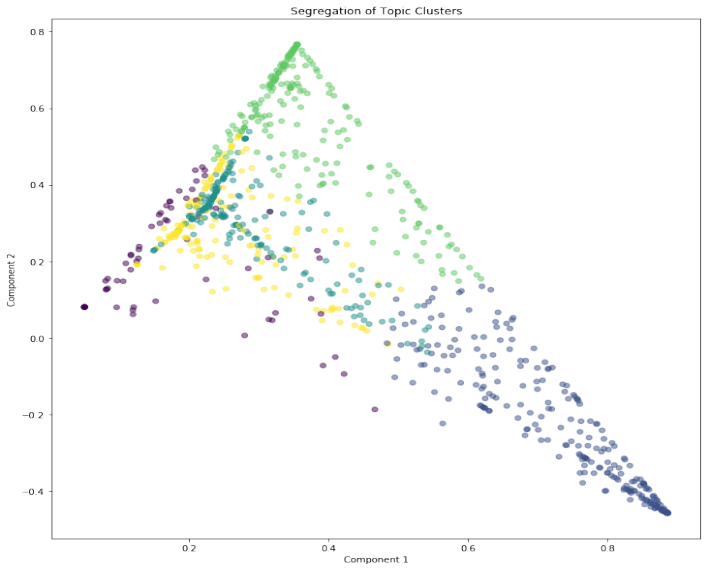
with $\Sigma_k V_k^\top$ the transformed set with $k$dimensions.

In our case, we want to reduce dimensionality to 2 dimensions, our two axis.

We print the weights of the 5 columns of our LDA data for each dimension and the percentage of total information in our LDA data explained by the two dimensions:

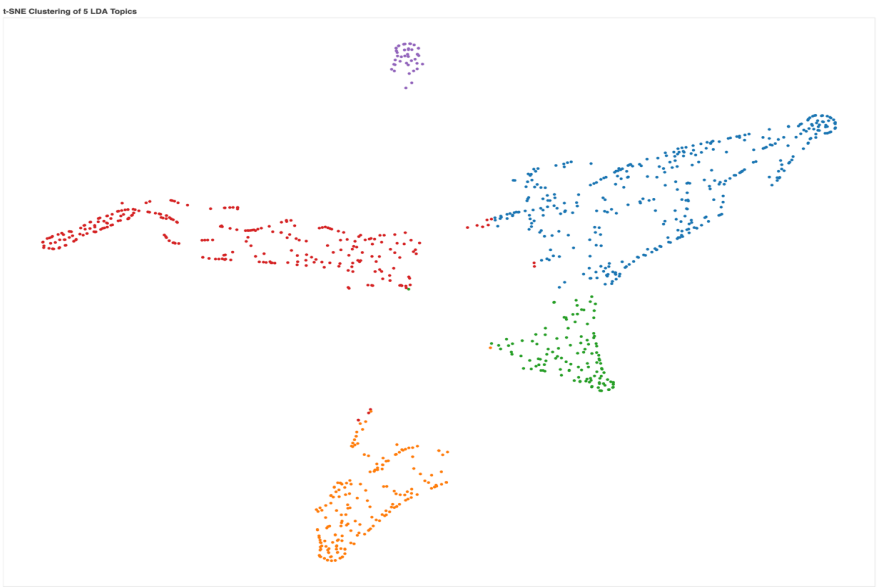
```
Component's weights:
 [[ 0.36  0.18  0.23  0.05  0.89]
 [ 0.77  0.28  0.35  0.08 -0.46]]
Perc of Variance Explained:
 [0.14 0.3 ]
```

The final plotting gives the following result:



We can clearly observe the different clusters and the shape of the data. We can recognize the Dirichlet distribution shape.

We wish to visualize the clusters of documents in a 2D space using t-SNE algorithm mentioned in Part 2. A.



We can clearly observe the 5 clusters.

B.2. Non-negative Matrix Factorisation (NMF) Method

We try another method: the *Non-negative Matrix Factorisation* (NMF) method. The Non-negative Matrix Factorisation method, whose name was given by Lee and Seung in their paper *Algorithms for Non-negative Matrix Factorization*, is a useful method for dimensionality reduction and data analysis. Similar to PCA, NMF has the advantage of working with non-negative vectors.

In practice, NMF is commonly used in topic modelling because it uses context, improving topic prediction. As an example, if we consider the verb "eat" in a document, it could have different meanings:

1. I eat three times a day,
2. I eat my favorite meal on my birthday,
3. I eat proteins to increase my time when running.

In this case, the word "eat" is used as a verb in the three sentences but has different meanings:

"eat" + "three times a day" => sustaining for survival
"eat" + "my favorite meal on my birthday" => pleasure of food
"eat" + "proteins to increase my time when running" => sport diet

To understand the process of the algorithm, we have to come back to the theory. Mathematically, Non-negative Matrix Factorization can be defined as followed:

> Given a non-negative matrix V , we can find non-negative matrix factors W and H such that:

$$V \approx WH$$

In other words, each data vector v is approximated by a linear combination of the columns of W,
weighted by the components of h, where v and h are the corresponding columns of V and H.
In topic modeling, V, W and H represent:

- The document-word matrix, composed of all the words in the corpus of text (V),
- The basis vectors, composed of the topics (clusters) of the document (W),

- The coefficient matrix, composed of the weights of each topic in the documents and gives the cluster membership (H).

As defined, NMF has an inherent clustering property.$W$and $H$are calculated iteratively so that the product approaches$V$. The algorithm stops either when the approximation error converges or when the predefined number of iterations is reached.

We decide to apply NMF to our corpus of texts. The results are the following:

```
Topic 0:
vente travaux sci civil époux monsieur préjudice 000 parties garantie
Topic 1:
pénale conseiller 2018 instruction criminelle greffier contre faits statuant ordonnance
Topic 2:
licenciement salarié employeur salaire indemnité salariée sérieuse rupture réelle poste
Topic 3:
sécurité sociale caisse cotisations chsct maladie accord charge recours redressement
Topic 4:
acte 2018 pourvoi formé déclaré contre rapport france 700 condamne
```

We can again assign clearly different branches of the law to the topics, except for the last topic:

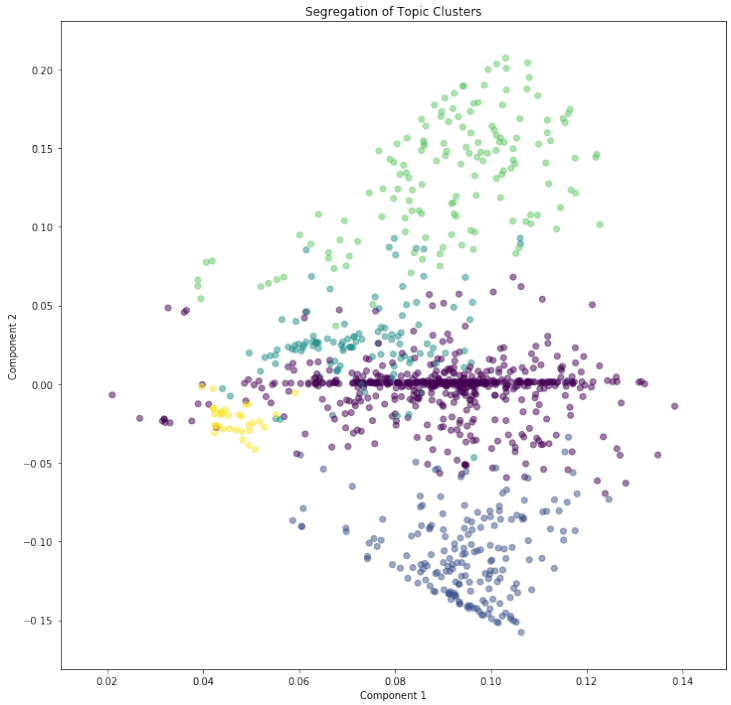
- Topic 0 deals with property law,
- Topic 1 deals with criminal law,
- Topic 2 deals with labour law,
- Topic 3 deals with social security law,
- Topic 4 remains quite hard to interpret.

When we take a closer look at topic allocation, a low number of documents are represented by the topic 4.

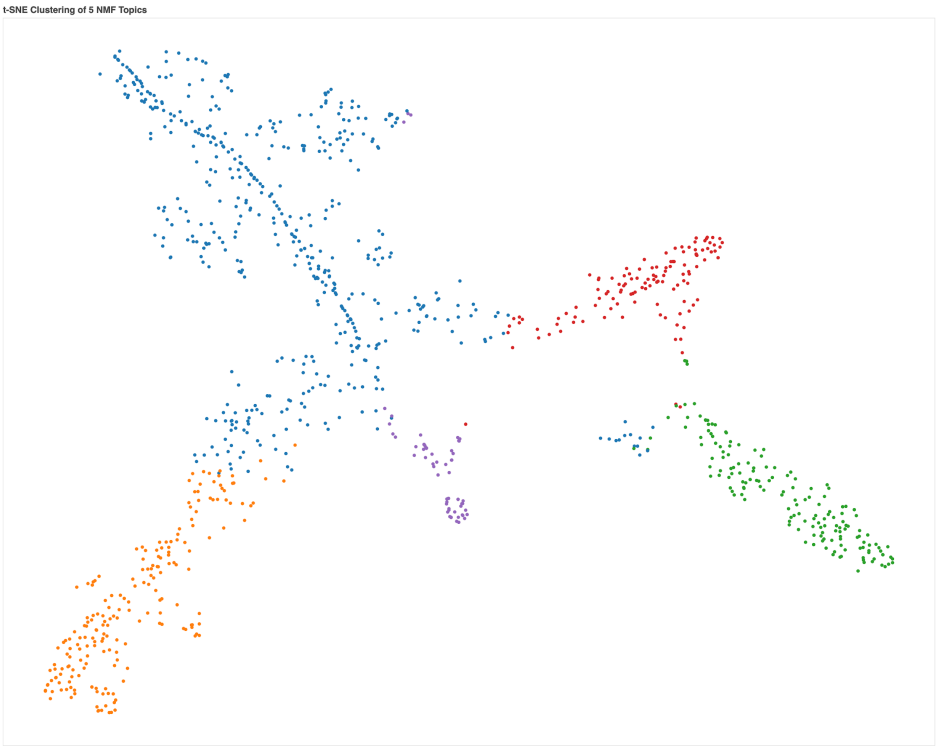
	Topic Num	Num Documents
0	0	502
1	1	197
2	2	158
3	3	122
4	4	58

When we compare the topics allocation with LDA, we notice three commun topics were spotted by the two algorithms (property, criminal and labour law) however in very different proportions. For example if we look at criminal law topic, 255 documents were allocated for LDA and 58 for NMF.

We decide to use again k-means algorithm to plot the clusters.



We finally plot the clusters of documents in a 2D space using t-SNE algorithm.



We have again our 5 clusters but not as well determined as with LDA.

C. Word embedding

We here look at another paradigm in which we no more consider our judicial decisions as a bag of preprocessed words transformed into a sparse matrix containing adjusted frequencies (part 2.A), but rather looking at a representation of an embedded space in which words that are similar to each other now share proximity in the new space. Again, so as to put the emphasis on this difference of mindset, in the Bag of Words approach we used direct counting methods to create document vectors, as in word-embedding the idea is to create a contained space in which similar words are not far from each other in terms of statistic distance. For instance, the words “endure” and “tolerate” would probably be close in a word embedding model, whereas the vectors in a classic approach (simple frequency or tf-idf) would naturally be very far (same meaning but the starting letter of both are different, this suffices to distance these vectors off from each other).

This latter model (word embedding) has a new champion: word2vec (Mikoloc & al. 2013), a model patented by a former employee of google. This is the simplified theory behind the learning phase:

-step 1, a random vector is assigned to each word in the corpus, as well as a “context” around it consisting in a sample of 5 or so words around it,

-step 2, we compute the probability of having a word in the context of another word as the sigmoïd function of the scalar product of both word vectors, or

$$\mathbb{P}(\text{have word in context}) = \frac{1}{1 + e^{-v_{context} \cdot v_{word}}}$$

This step is especially working well as the inputs of the model, either the context itself in the case of CBOW model (standing as continuous bag of words) or the target word in the case of Skipgram model, are fed to a shallow neural-network with only one hidden layer with a linear activation function, leading itself to the output layer activated with softmax function so as to end up with probability of having a word (CBOW) or having the context (Skipgram).

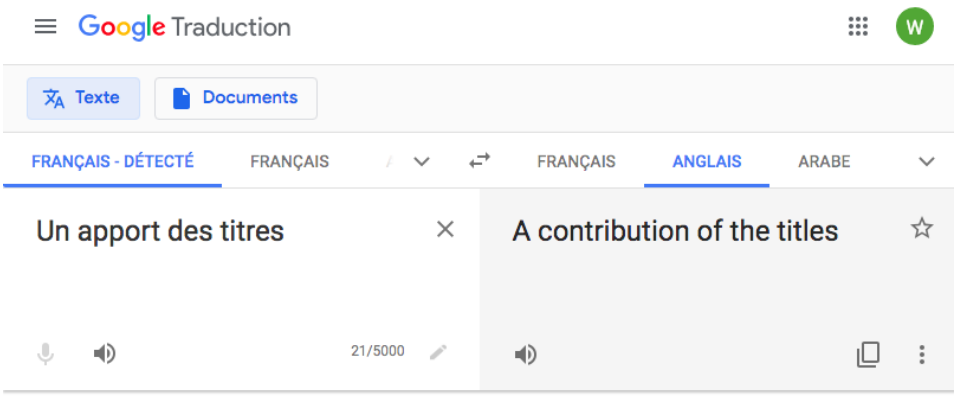
-step 3, we construct the objective function which aims at representing the distribution of all word probabilities in their context, so as to maximise terms when there correctly placed in context and the contrary if not in context, or

$$argmax_{(word,context)} \sum \log \mathbb{P}(\text{have word in context}) + \sum \log \mathbb{P}(\text{have word not in context})$$

This objective function is then optimized through gradient descent in order at the end to find the optimal values of all word vectors that maximise the above function. This procedure should normally bring closer words that are semantically related and push back words that have nothing to do with each other in a given context.

This word representation model is quite interesting as it has the capacity of creating a space that brings closer some words even though they maybe never were close to each other in the corpus! This is due to the fact it is context oriented.

As mentioned in the problem statement, many legal fields remain still untouched by these text analysis techniques, especially regarding legal texts translations. One possible business implementation could be done with Word2vec to create a specific translator dedicated to legal texts. Indeed, the translation of legal documents could be largely improved by training the translation model only on a specific legal corpus. Indeed, even if all well-known translators trained models on multibillion-word corpus, these were composed of a wide variety of texts, thus lowering the quality of speech and the unrestrained proposition of words.
For instance, when translating the French sentence " un apport des titres" on google translate, the output is the following:



In French, "titres" is mainly used for movies so as to specify the name of the movie (aka "the title"). As the majority of the corpus upon which google translate model has learned was not only driven on legal subjects, Google translate picked a choice as it had the highest probability of success.

Of course, we are not blaming its technology, as it surely more advanced than the word2vec model (patented in 2013 by Google employees!), but it can make some sense for law firms or legal departments to use a translating service that has been only trained on a legal corpus so as to avoid such pitfalls.

The appendix 2 also shows a small use-case of word2vec through the gensim library, the model was trained on our corpus scraped of 8717 texts, which amounts to more than 147 millions words. The results show already the creation of an embedded space in which close words are also semantically close. For instance, “titres” is the most closely related to “actifs” in terms of probabilistic distance, which would definitely improve the translation after towards english (and thus not choosing “titles”).

```
model.most_similar(positive=['preuve'], topn=5)
```

```
scrapping.py:5: DeprecationWarning: Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_simil
r() instead).


[('démonstration', 0.7434113621711731),
 ('preuve-', 0.5647727251052856),
 ('justification', 0.5171993374824524),
 ("d'alléguer", 0.5152993202209473),
 ('prouver', 0.5120017528533936)]
```

Conclusions and Recommendations

We decided in this this thesis to have a broad approach on:
-how one deals with the life cycle of analysing a text with computer science techniques (pre-processing and transformation of words into vectors, and how this could be used as first step for machine learning frameworks, especially with unsupervised concerns), and
-possible business cases (cited in the problem statement, and one developed in 2.C for translation).

But by the way, why choosing only decisions from the French highest judicial decision court, or "Cour de cassation" ? The main reason is that it is a highly structured way of writing. There are strong codes, and every French law professional could recognise a text from this entity only by looking at the structure. Without entering too much into detail, decisions of the cour de Cassation are twofold, either the court rejects the "pourvoi" of the party that was contesting the decision in previous court, either it is a censure of the decision ruled in previous court.
This means that developing a tool based mainly on regular patterns of words ("regular expression") or syntax could also by a way of making the best of computer power for decision prediction: indeed, one would have to first retrieve the relevant documents regarding ones legal issue and then keep into memory all that mattered (number of winning rulings, which main arguments were useful, average damages gained, best courts...etc).

But of course, we are here talking about complex reasonings, factual and law-related information, various steps of decisions and a vocabulary not often used. Especially, the result of a decision in not always easy to understand: is it a fine? An over-rulement? A suspensive decision? In fact, understanding rulements takes a whole university cursus and one could be more than puzzled when these new legal techs achieve high accuracy score on predicting decisions in only a couple seconds.

Bibliography

- Van der Maaten, Hinton, *Visualizing Data using t-SNE,* Journal of Machine Learning Research, 2579-2605, 2008
- Stéphane Moussie, *Les origines de Siri*, https://www.igen.fr/iphone/les-origines-de-siri-104696, 2013
- LOI n° 2016-1321 du 7 octobre 2016 pour une République numérique, https://www.legifrance.gouv.fr/eli/loi/2016/10/7/ECFI1524250L/jo/texte
- https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
- Alan Turing, *Computing machinery and intelligence*, Mind journal, n°236, 1960
- https://openclassrooms.com/fr/courses/4470541-analysez-vos-donnees-textuelles
- Nikolaos Aletras, Dimitrios Tsarapatsanis3, Daniel Preoţiuc-Pietro, Vasileios Lampos, *Predicting judicial decisions of the European Court of Human Rights: a Natural Language Processing perspective*, *PeerJ – Computer Science*, 2:e93; DOI 10.7717, 2016
- Thomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, E*fficient estimation of Word Representations in Vector Space*, 2013
- Hotelling, *Analysis of a complex of statistical variables into principal components*, *Journal of Education Psychology,* 25: 417-441, 1933
- D. M. Blei. *Probabilistic Topic Models*. In Communications of the ACM.
- Bietti, A. (2012). *Latent Dirichlet Allocation*. mai 2012, working paper, http://alberto. bietti. me/files/rapport-lda. pdf.
- Chen, K. Y. M., & Wang, Y. (2011). *Latent dirichlet allocation*. Technical report, Department of Electrical and Computing engineering. University of California, San Diego 4: 87-110.
- Hoffman, M., Bach, F. R., & Blei, D. M. (2010). Online learning for latent dirichlet allocation. In *advances in neural information processing systems* (pp. 856-864).

- Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems* (pp. 556-562).
- Sra, S., & Dhillon, I. S. (2006). Generalized nonnegative matrix approximations with Bregman divergences. In *Advances in neural information processing systems* (pp. 283-290).
- https://wiki.ubc.ca/Course:CPSC522/Latent_Dirichlet_Allocation
- https://www.machinelearningplus.com/nlp/topic-modeling-python-sklearn-examples/

Appendices

1. Legifrance scrapping (script)

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from selenium.common.exceptions import NoSuchElementException

option = webdriver.ChromeOptions()
option.add_argument(" — incognito")

browser = webdriver.Chrome(executable_path="/Users/lamazere/Library/Application Support/Google/chromedriver", options=option)

browser.get("https://www.legifrance.gouv.fr/rechJuriJudi.do?reprise=true&page=1")

#search
s_input = input("What are you lookning for ? ")
search = browser.find_element_by_id("champ7")
search.send_keys(s_input)

#time period
t_input = input("Which year?")
time = browser.find_element_by_id("champDateDecision1A")
time.send_keys(t_input)

#lauch search
search_button = browser.find_element_by_name("bouton")
search_button.click()


i=2
all_links1=[]
all_links2=[]
all_linkslast=[]

def get_links(text,liste):
    links_elts = browser.find_elements_by_partial_link_text(text)
    links = [x.get_attribute('href') for x in links_elts]
    for link in links:
        liste.append(link)
```

```
while True:
    try:
        next_button =
browser.find_element_by_xpath("""//*[@id="result"]/div[4]/ul/li[{}]/a""".format(i))
        get_links('Cour de cassation,',all_links1)
        i+=1
    except NoSuchElementException:
        break
    next_button.click()

while True:
    try:
        get_links('Cour de cassation,',all_links2)
        next_button = browser.find_element_by_xpath("""//*[@id="result"]/div[4]/ul/li[4]/a""")
    except NoSuchElementException:
        break
    next_button.click()

browser.find_element_by_xpath("""//*[@id="result"]/div[4]/ul/li[5]/a""").click()
get_links('Cour de cassation,',all_linkslast) #search only decisions of "cour de cassation"

all_links = all_links1 + all_links2 + all_linkslast

pick_dir = input("name of directory ?")
path = "/Users/lamazere/Desktop/Pour mémoire/%s" %pick_dir

#create file of scraped texts
os.mkdir(path)
os.chdir(path)

i=1
for links in all_links:
    page = requests.get(links)
    parse = BeautifulSoup(page.content, 'html.parser') #parses the pages
    text = parse.find("contenu").get_text().replace("\n"," ")
    open("text{}.txt".format(i),"w").write(text)
    i+=1
```

2. Jupyter Notebook

Entrée [3]:

```
#NLTK package
import nltk
from nltk import FreqDist
from nltk.corpus import stopwords
from nltk.stem.snowball import FrenchStemmer
from nltk.tokenize import word_tokenize

#for vizualisation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#for dark theme
#from jupyterthemes import jtplot
#jtplot.style(theme='chesterish')

#so that warnings don't appear
import warnings
warnings.filterwarnings("ignore")

#for text exploration
import string
import os
from collections import defaultdict

# t-SNE and Scikit-Learn package
from sklearn.manifold import TSNE
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation, TruncatedSVD, NMF
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans
import seaborn as sns

# bockeh plotting for t-SNE
import bokeh.plotting as bp
from bokeh.plotting import save
from bokeh.models import HoverTool
from bokeh.io import output_file, show
from bokeh.plotting import figure
import matplotlib.colors as mcolors

#for scrapping
from bs4 import BeautifulSoup
import urllib
import requests

#for word embeding - word2vec
import gensim
import glob
```

executed in 5.42s, finished 18:39:08 2019-05-17

```
/Users/Lamazere/miniconda3/lib/python3.6/site-packages/smart_open/ssh.
py:34: UserWarning: paramiko missing, opening SSH/SCP/SFTP paths will
be disabled.  `pip install paramiko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be
disabled.  `pip install paramiko` to suppress')
```

1 Part 1: How to pre-process textual data

1.1 A. Chose a relevant corpus of texts

There are plenty of sources to analyze on the internet and the little experience we have had in working in Legal departments and Law firms is that a majority of the time spent is for searching online for the source that will confirm one's thought or infirm it. There is the official government website Légifrance that contains the legislative and regulatory texts and court decisions of the supreme courts and appeals under French law.

We therefore decided to run a small script that uses 2 main logics:

- the first one was to run several loops, one for fetching all weblinks containing every decision, and one for opening all them and writing them in separate files.
- the second one was needed a tool that would support dynamic content with Selenium package, as the texts that appear on the user's interface naturally come from the search query that was sent to the server.

Entrée [ ]:

```
path = "/Users/lamazere/Desktop/Pour mémoire"
os.chdir(path)

%run -i 'scrapping.py' #scrapping.py is the python file, which, once runed, asks the
```

executed in 4.22s, finished 21:51:49 2019-05-12

1.2 B. Exploration of the corpus

Once we have scrapped our corpus of texts from Légifrance for 2018, we want to learn more about the composition of the texts. In order to do so, we use the tokenization process. Tokenization will separate the elements of a sentence, called tokens. More precisely we use the function word_tokenize that works as following :

Entrée [4]:

```
test = "Bonjour, je sert d'exemple pour la thèse de Master. Essayons !"

nltk.word_tokenize(test)
```

executed in 36ms, finished 18:39:20 2019-05-17

Out[4]:

```
['Bonjour',
 ',',
 'je',
 'sert',
 "d'exemple",
 'pour',
 'la',
 'thèse',
 'de',
 'Master',
 '.',
 'Essayons',
 '!']
```

We can observe several problems to this function: the punctuation is considered as a token and the apostrophes are considered as part of a word and concatenate two different words.

The process of correcting the punctuation errors is called text normalization. Once we have normalized the corpus of text, we create a function that will count the frequency of words as well as the total number of words. In order to calculate the frequency of each word, we use the bag-of words model.

1.2.1 Without transformation

Entrée [21]:

```
tokenizer=nltk.RegexpTokenizer(r'\w+')

# Creation of a dictionary database with arrays of tokens of each text
path="/Users/lamazere/Desktop/Pour mémoire/all_2018"

def load_all_texts(number_texts=len(os.listdir(path))):
    db = {}
    for i in range(1,number_texts):
        try:
            droit=open(path+"/text%s.txt"%i,"r").read()
        except FileNotFoundError:
            droit=open(path+"/text%s.txt"%(6968),"r").read()
        db[i]=tokenizer.tokenize(droit.lower())
    return db

db = load_all_texts();
```

executed in 25.7s, finished 19:45:37 2019-05-17

Entrée [22]:

```
print('Loading of {} texts in the db'.format(len(db.keys())))
```

executed in 6ms, finished 19:45:41 2019-05-17

```
Loading of 8717 texts in the db
```

Entrée [28]:

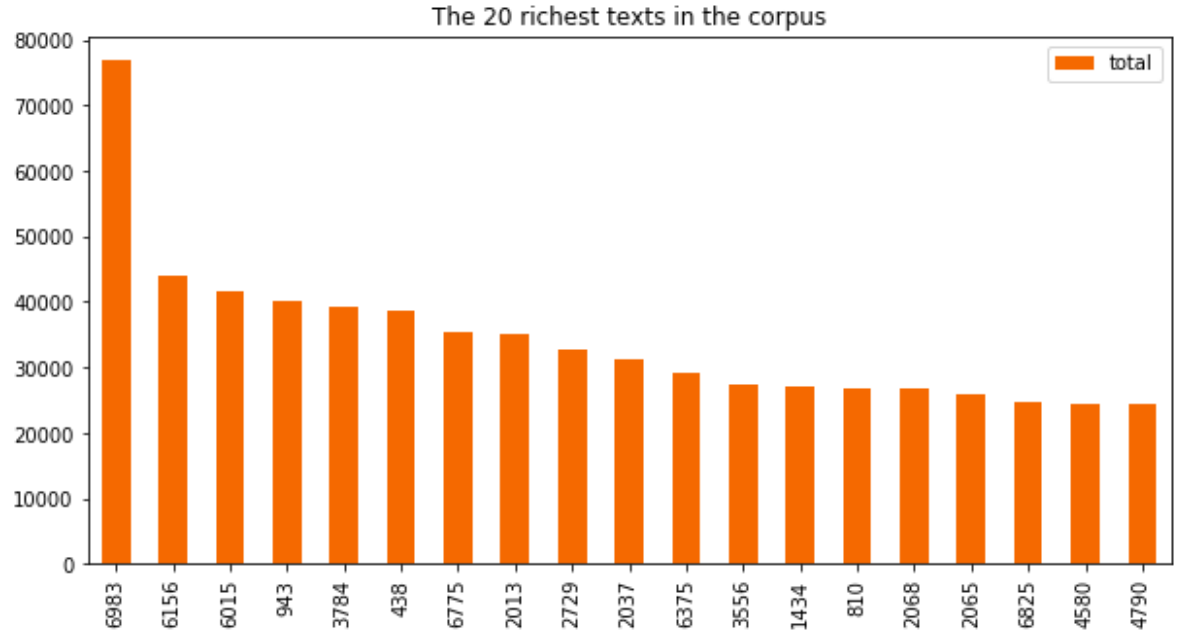
```
# Creation of a function that counts the frenquency and the total length of the corp
def freq_stats_corpora():
    corpora = db

    stats, freq = dict(), dict()
    freq_totale = nltk.Counter()

    for k, v in corpora.items():
        freq[k] = fq = nltk.FreqDist(v)
        stats[k] = {'total': len(v) }
        freq_totale += freq[k]
    return (freq_totale, freq, stats, corpora)

freq_totale, freq, stats, corpora = freq_stats_corpora()
```

executed in 2m 1s, finished 20:03:56 2019-05-17

Entrée [29]:

```
# Recuperation of the length
df_st = pd.DataFrame.from_dict(stats, orient='index')

# Print the 20 richest texts in the corpus
df1_st = df_st.sort_values('total', ascending=False)
df2_st = df1_st.iloc[0:19,:].copy()
df2_st.plot(kind='bar', color="#f56900", title='The 20 richest texts in the corpus',
```

executed in 606ms, finished 20:04:07 2019-05-17

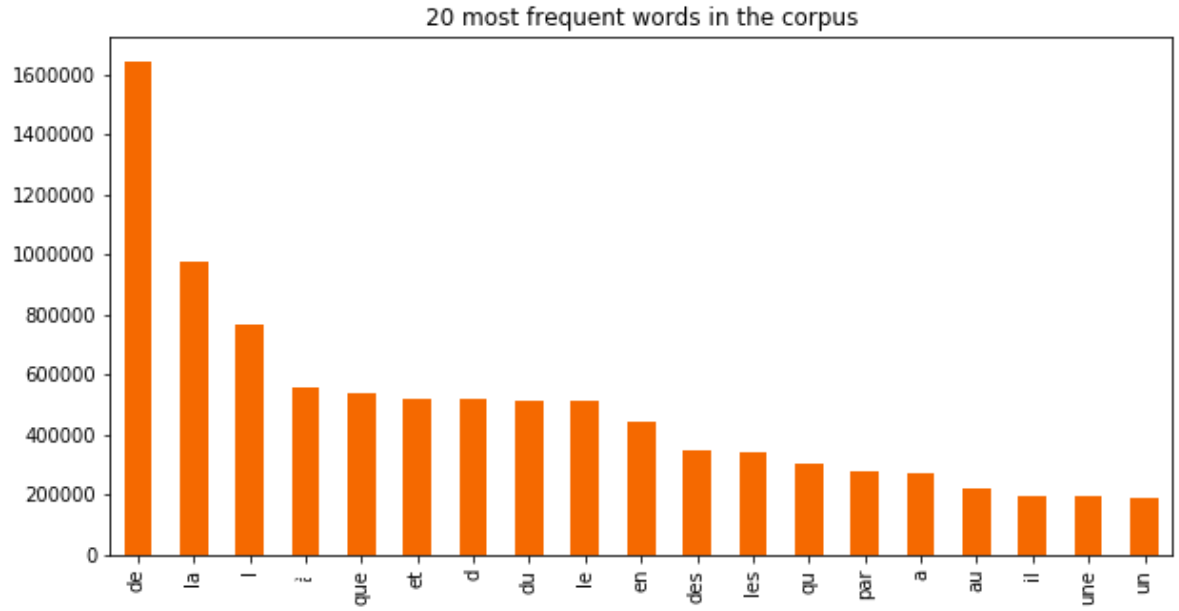


Thanks to this function we can see that our corpus is quite rich in words as the richest text contains about 40000 words. We can also notice that very rapidly, the number of words remains more or less the same, which is quite logical as legal texts are quite standardized.

Entrée [30]:

```
# Recuperation of the frequency
freq_totale, freq, stats, corpora = freq_stats_corpora()
df_fq = pd.DataFrame.from_dict(freq_totale, orient='index')

# Print the 20 most frequent words in the corpus
df1_fq = df_fq.sort_values(0, ascending=False)
df2_fq = df1_fq.iloc[0:19,:].copy()
df2_fq.plot(kind='bar', color="#f56900",legend=False, title='20 most frequent words
```

executed in 2m 0s, finished 20:10:23 2019-05-17



When we plot the most frequent words, we can't help but noticing that we only plotted the most common words in French language.

At this stage, we clearly sea that we need to clean our tokens as almost all most frequent words are the ones that are meaningless. Let's remove all stopwords and use some stemming to clean our corpus. To do that, let's find the most frequent words in our whole corpus (in all our texts).

1.2.2 With stopwords removal

Entrée [64]:

```
most_freq = next(zip(*freq_totale.most_common(85)))

print(most_freq)
```

executed in 556ms, finished 14:49:03 2019-05-20

```
('de', 'la', 'l', 'à', 'que', 'et', 'd', 'du', 'le', 'en', 'des', 'le
s', 'qu', 'par', 'a', 'au', 'il', 'une', 'un', 'pour', 'est', 'n', 'su
r', 'pas', 'dans', 'société', 'ne', 'm', 'qui', 'cour', 'aux', 'articl
e', 'code', 'ce', 'son', 'y', 'x', 'sa', 'appel', 'été', 'arrêt', 'o
u', 's', 'fait', 'elle', 'travail', 'être', 'cette', 'avait', 'alors',
'était', 'mme', '1', 'procédure', 'ses', 'moyen', 'se', 'ainsi', 'avoi
r', 'ces', 'sans', 'euros', 'demande', 'titre', 'cassation', 'civile',
'contrat', '2', 'motifs', 'lui', 'décision', 'attendu', 'même', 'ave
c', 'leur', 'droit', 'dont', 'ont', 'sont', 'somme', '3', 'non', 'cham
bre', 'deux', 'peut')
```

From the most frequent words we decide to create our own personalized stopwords composed of the 85 most frequent words as well as the default stop words in the NLTK library.

Entrée [65]:

```
sw = set()
default = nltk.corpus.stopwords.words('french')

sw.update(most_freq)
sw.update(tuple(default))
```

executed in 25ms, finished 14:49:36 2019-05-20

Let's now create another function to count again the most freq words without those most commun words.

Entrée [34]:

```
path="/Users/lamazere/Desktop/Pour mémoire/all_2018"

def load_all_texts2(number_texts=len(os.listdir(path))):
    db = {}
    for i in range(1,number_texts):
        try:
            db[i]=open(path+"/text%s.txt"%i,"r").read().lower()
        except FileNotFoundError:
            db[i]=open(path+"/text%s.txt"%(6968),"r").read().lower()
    return db

db2 = load_all_texts2();
```

executed in 8.61s, finished 20:11:28 2019-05-17

Entrée [24]:

```
print('Loading of {} texts in the db'.format(len(db2.keys())))
```

executed in 7ms, finished 19:52:28 2019-05-17

```
Loading of 8717 texts in the db
```

Entrée [38]:

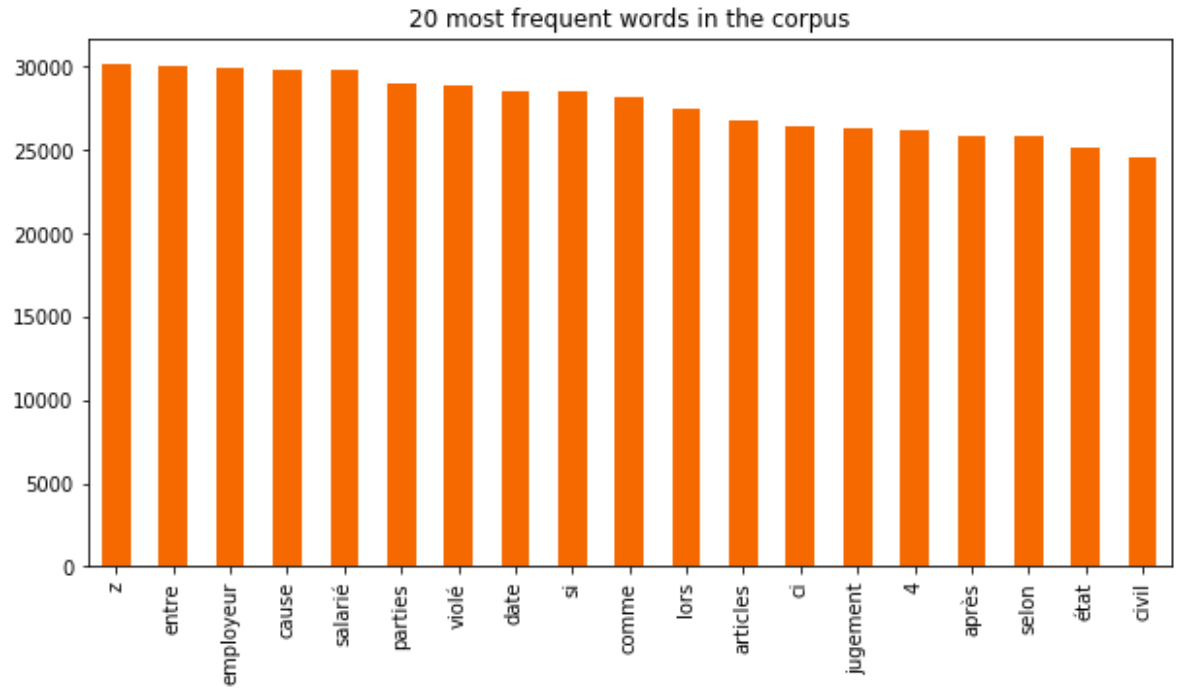
```
def freq_stats_corpora2(lookup_table=[], database = db2):
    corpora = defaultdict(list)

    for num, text in database.items():
        tokens = tokenizer.tokenize(text)
        corpora[num] += [w for w in tokens if w not in list(sw)]

    stats, freq = dict(), dict()
    freq_totale = nltk.Counter()

    for k, v in corpora.items():
        freq[k] = fq = nltk.FreqDist(v)
        stats[k] = {'total': len(v) }
        freq_totale += freq[k]
    return (freq_totale, freq, stats, corpora)

freq_totale2, freq2, stats2, corpora2 = freq_stats_corpora2()
```

executed in 9ms, finished 20:18:10 2019-05-17

Entrée [17]:

```
df_fq2 = pd.DataFrame.from_dict(freq_totale2, orient='index')

df1_fq2 = df_fq2.sort_values(0, ascending=False)
df2_fq2 = df1_fq2.iloc[0:19,:].copy()
df2_fq2.plot(kind='bar', color="#f56900", legend=False, title='20 most frequent word
```

executed in 562ms, finished 22:15:32 2019-05-12

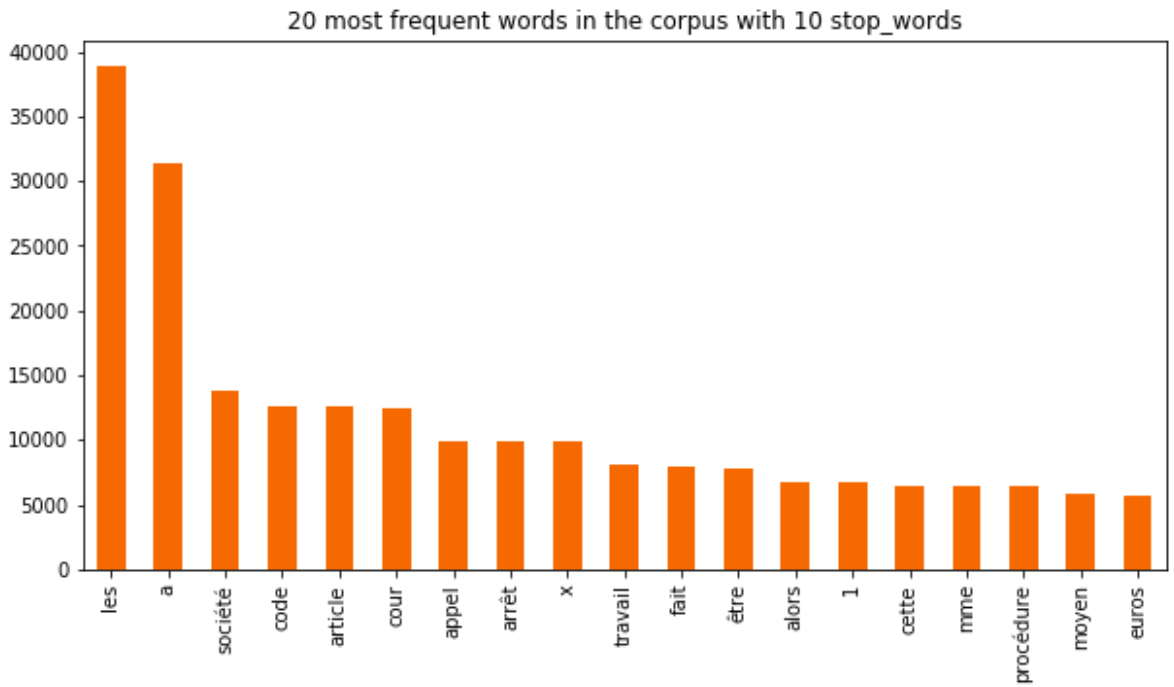


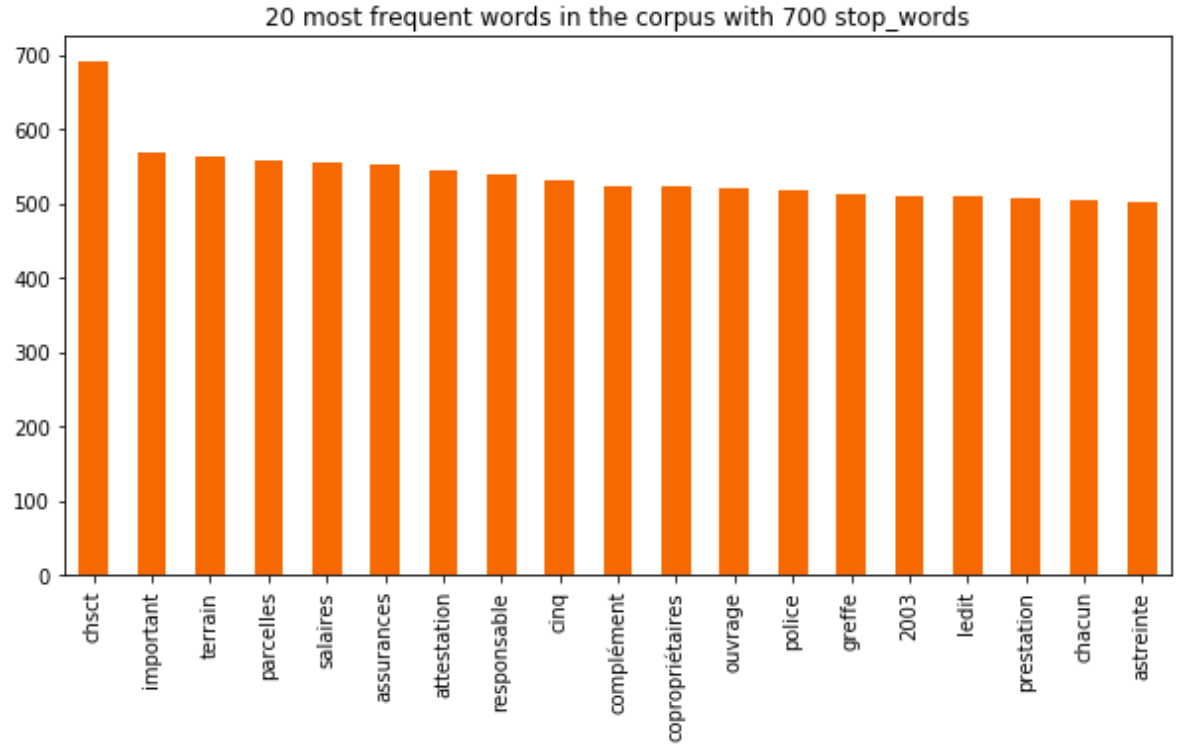
1.2.2.1 Number of Stopwords - hyperparameter

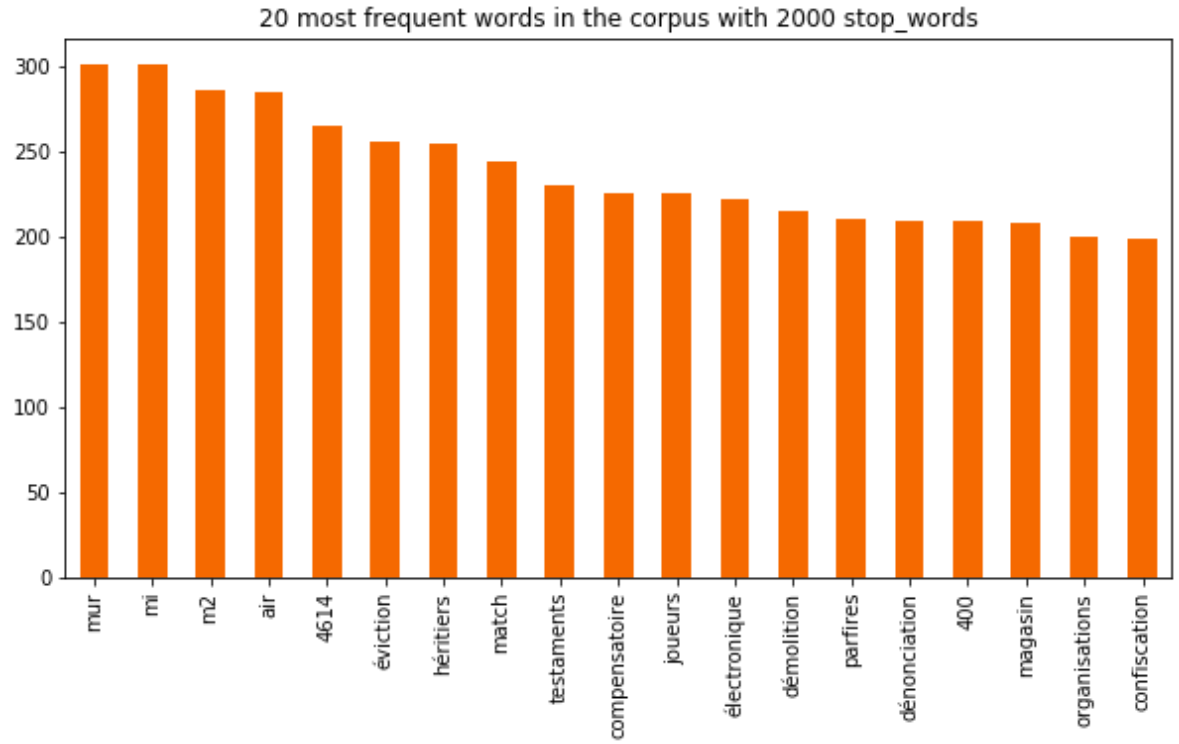
Entrée [41]:

```
#we look at the effects of number of stopwords on the quality of out texts
#we try on a smaller corpus

db_reduced = load_all_texts2(number_texts = 1000)

for nb_stopwords in [10,700,2000]:
    most_freq = next(zip(*freq_totale.most_common(nb_stopwords)))
    sw = set()
    default = nltk.corpus.stopwords.words('french')
    sw.update(most_freq)
    sw.update(tuple(default))

    #load data
    freq_totale2, freq2, stats2, corpora2 = freq_stats_corpora2(database = db_reduce

    #plot
    df_fq2 = pd.DataFrame.from_dict(freq_totale2, orient='index')
    df1_fq2 = df_fq2.sort_values(0, ascending=False)
    df2_fq2 = df1_fq2.iloc[0:19,:].copy()
    df2_fq2.plot(kind='bar',
                 color="#f56900",
                 legend=False,
                 title='20 most frequent words in the corpus with %d stop_words' % r
                 figsize=(10,5));
```

executed in 4m 51s, finished 20:32:19 2019-05-17







We see a clear improvement when looking at the 20 most frequent words in the corpus with stopwords revowals. The final step of the cleaning process is lemmatization and stemming. For both processes, the goal is reduce the derivations of a word to a single one. Stemming is less effective that lemmatization that will analyse grammatically the word to associate it to the most relevant form, know as the lemma. Unfortunately, there is no French lemmatization function in the NLTK library, we decided to use the stemming function.

1.2.3 Lemmatization & Stemming

Entrée [42]:

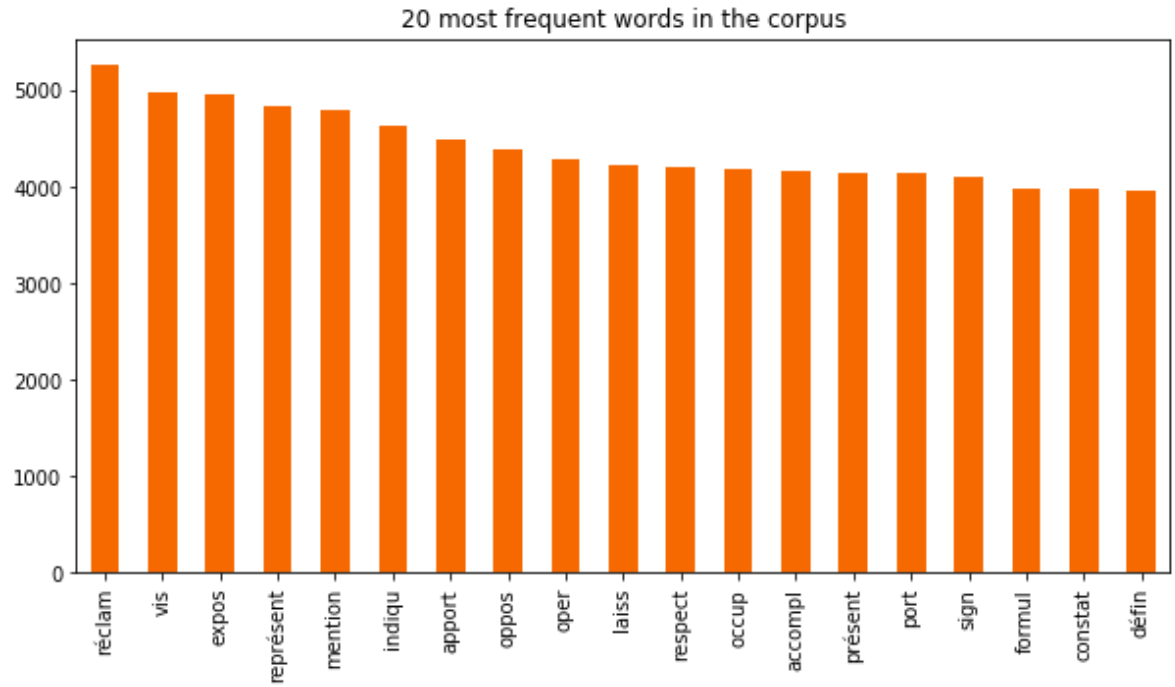
```
stemmer = FrenchStemmer()

def freq_stats_corpora3(lookup_table=[]):
    corpora = defaultdict(list)

    for num, text in db2.items():
        tokens = tokenizer.tokenize(text)
        corpora[num] += [stemmer.stem(w) for w in tokens if w not in list(sw)]

    stats, freq = dict(), dict()
    freq_totale = nltk.Counter()

    for k, v in corpora.items():
        freq[k] = fq = nltk.FreqDist(v)
        stats[k] = {'total': len(v) }
        freq_totale += freq[k]
    return (freq_totale, freq, stats, corpora)

freq_totale3, freq3, stats3, corpora3 = freq_stats_corpora3()
```

executed in 36m 24s, finished 21:40:08 2019-05-17

Entrée [43]:

```
df_fq3 = pd.DataFrame.from_dict(freq_totale3, orient='index')

df1_fq3 = df_fq3.sort_values(0, ascending=False)
df2_fq3 = df1_fq3.iloc[0:19,:].copy()
df2_fq3.plot(kind='bar', color="#f56900",legend=False, title='20 most frequent words
```

executed in 2.61s, finished 22:07:45 2019-05-19



When we plot the 20 most frequent words in the corpus we can clearly see law vocabulary showing up such as “condamn”, “compt” etc...

2 Part 2: Transform textual data

2.1 A. Represent the corpus of text as a “bag-of-words”

The bag of words approach is very intuitive: we simply consider every Cour de cassation decision as a whole of words, without any consideration of contexte (order, how it is used…).

Now that this term-document matrix is created, another step is necessary for this approach to make sense. Indeed, in language analysis it is essential to capture the co-dependencies of the words in a text.

One technique is to weigh up or down the importance of a word in a text relatively to the whole corpus of texts - if they only appear in this text, it should be noticed and weighed up, and the opposite if they in fact appear in most of the documents. This technique is called the Term Frequency-Inverse Document Frequency (TF-IDF).

Entrée [67]:

```
#run with the list of 85 stopwords
translator = str.maketrans('', '', string.punctuation)

path = "/Users/lamazere/Desktop/Pour mémoire/Test2018"
os.chdir(path)

def tokenize(text):
    tokens = nltk.word_tokenize(text)
    return tokens

token_dict = dict()

for subdir, dirs, files in os.walk(path):
    for file in files:
        file_path = subdir + os.path.sep + file
        shakes = open(file_path, 'r', encoding = "ISO-8859-1")
        text = shakes.read()
        lowers = text.lower()
        no_punctuation = lowers.translate(translator) #removes punctuation
        token_dict[file] = no_punctuation


tfidf = TfidfVectorizer(tokenizer=tokenize, stop_words=sw)
values = tfidf.fit_transform(token_dict.values())
```

executed in 49.0s, finished 14:52:06 2019-05-20

Entrée [45]:

```
values
```

executed in 15ms, finished 22:12:09 2019-05-19

Out[45]:

```
<1056x38311 sparse matrix of type '<class 'numpy.float64'>'
        with 329260 stored elements in Compressed Sparse Row format>
```

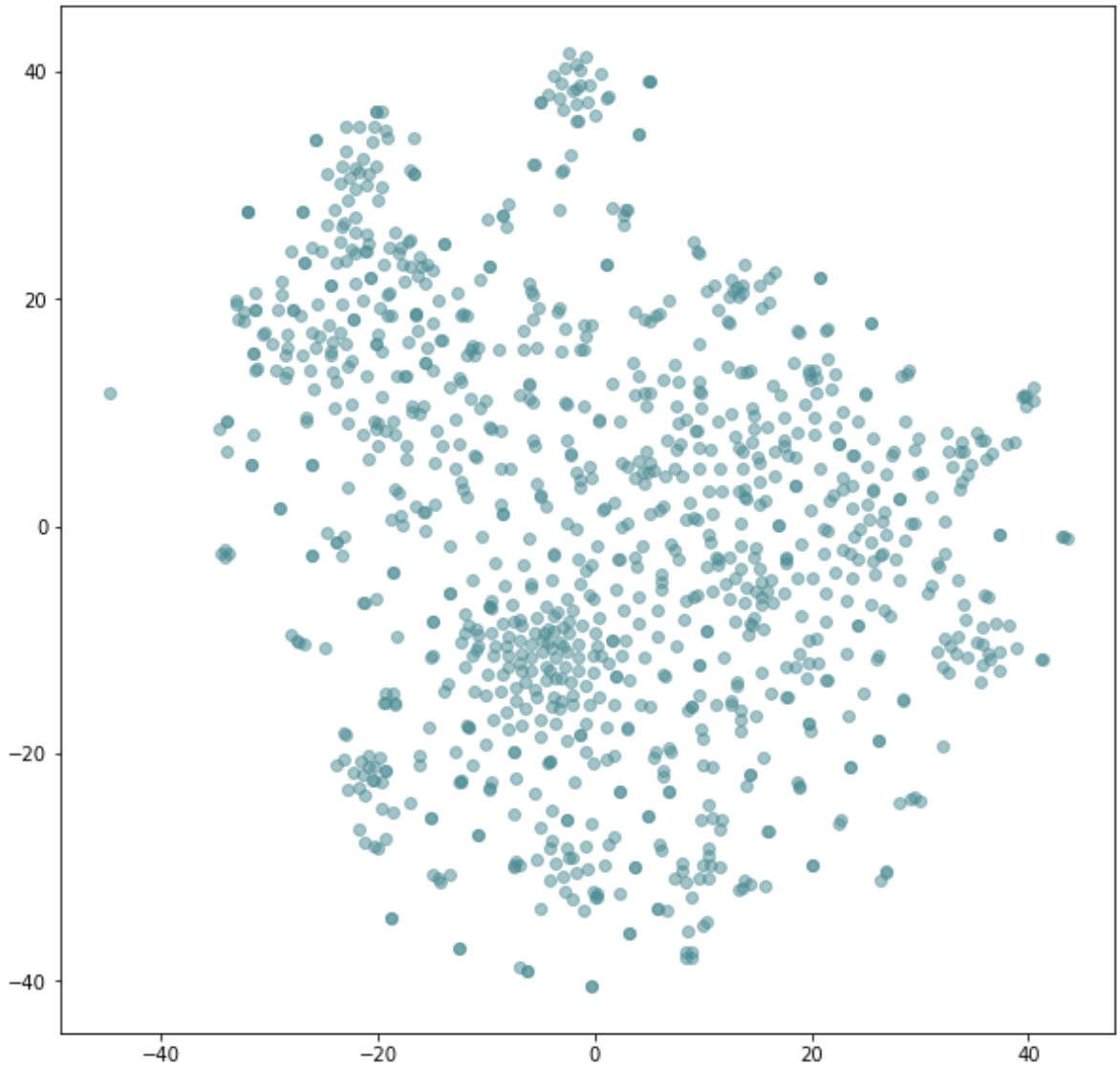
When applying TF-IDF to our corpus, the result is then a sparse matrix, each row representing an "arrêt" and each column a token of the corpus. The sparse matrix format is very useful for mathematical calculation purposes and more efficient storage wise. As TF-IDF is quite accurate, we can use it to do non-supervised

representations, like t-SNE, to bring together “arrêts” with similar TF-IDF indicators.

Entrée [68]:

```
X_tsne = TSNE(n_components=2, perplexity = 60,  learning_rate= 60, n_iter = 8000).fi

df_tsne = pd.DataFrame(X_tsne,columns=["X","Y"])

fig,ax = plt.subplots(figsize=(10,10))
ax.scatter(df_tsne["X"], df_tsne["Y"], alpha = 0.5, color = "#4A8B92");
```

executed in 2m 44s, finished 14:56:02 2019-05-20



We can clearly see some clusters appearing that could correspond to common thematics. As our dataset is not labelled, we will try next to determine those clusters by different methods.

2.2 C. Topic modelling with unsupervised methods

2.2.1 C.1. Latent Dirichlet Allocation (LDA) Method

The LDA topic model algorithm requires a document word matrix as the main input. As a result, we need to change the format of our database to this purpose.

Entrée [69]:

```
def load_all_texts_array():
    db = []
    for i in range(1,1038):
        droit=open("/Users/lamazere/Desktop/Pour mémoire/all_2018/text%s.txt"%i,"r")
        db.append(droit)
    return db

db = load_all_texts_array();
```

executed in 880ms, finished 15:02:07 2019-05-20

We create the document word matrix using CountVectorizer. In the below code, we have configured the CountVectorizer to consider words that have occurred at least 10 times (min_df) and ingnore terms that have occured more than 95% of the documents, remove our own stopwords.

Entrée [70]:

```
tf_vectorizer = CountVectorizer(max_df=0.95,
                                min_df=10,                          # minimum reqd occu
                                stop_words=sw,
                                max_features=50000)                 # max number of uni
tf = tf_vectorizer.fit_transform(db)
```

executed in 3.04s, finished 15:02:34 2019-05-20

Since most cells in this matrix will be zero, we want to know what percentage of cells contain non-zero values. We can see it is very low.

Entrée [71]:

```
# Materialize the sparse data
data_dense = tf.todense()

# Compute Sparsicity = Percentage of Non-Zero cells
print("Sparsicity: ", ((data_dense > 0).sum()/data_dense.size)*100, "%")
```

executed in 79ms, finished 15:02:52 2019-05-20

```
Sparsicity:  6.4204097829658915 %
```

We build our LDA model based on this matrix with the hyperparameter "number of topics" set to 10.

Entrée [72]:

```
lda_model = LatentDirichletAllocation(n_components=10,           # Number of topics
                                      max_iter=10,               # Max learning iter
                                      learning_method='online',
                                      random_state=100,          # Random state
                                      batch_size=128,            # n docs in each le
                                      evaluate_every = -1,       # compute perplexit
                                      n_jobs = -1,               # Use all available
                                     )
lda_output = lda_model.fit_transform(tf)

print(lda_model)  # Model attributes
```

executed in 24.5s, finished 15:03:34 2019-05-20

```
LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
             evaluate_every=-1, learning_decay=0.7,
             learning_method='online', learning_offset=10.0,
             max_doc_update_iter=100, max_iter=10, mean_change_tol=0.0
01,
             n_components=10, n_jobs=-1, n_topics=None, perp_tol=0.1,
             random_state=100, topic_word_prior=None,
             total_samples=1000000.0, verbose=0)
```

It is essential to validate this first model, as it has been selected a priori. We use the log-likelihood and the perplexity to evaluate the goodness-of-fit.

Entrée [73]:

```
# Log Likelyhood: Higher the better
print("Log Likelihood: ", lda_model.score(tf))

# Perplexity: Lower the better. Perplexity = exp(-1. * log-likelihood per word)
print("Perplexity: ", lda_model.perplexity(tf))

# See model parameters
print(lda_model.get_params())
```

executed in 2.10s, finished 15:03:57 2019-05-20

```
Log Likelihood:  -9047081.067993758
Perplexity:  2006.1431944561532
{'batch_size': 128, 'doc_topic_prior': None, 'evaluate_every': -1, 'le
arning_decay': 0.7, 'learning_method': 'online', 'learning_offset': 1
0.0, 'max_doc_update_iter': 100, 'max_iter': 10, 'mean_change_tol': 0.
001, 'n_components': 10, 'n_jobs': -1, 'n_topics': None, 'perp_tol':
0.1, 'random_state': 100, 'topic_word_prior': None, 'total_samples': 1
000000.0, 'verbose': 0}
```

We are going to look for the best model. n LDA, the core parameter is the number of topics and, in addition, we chose to search the learning rate as well.

Entrée [74]:

```
# Define Search Param
search_params = {'n_components': [5, 10, 15, 20, 25], 'learning_decay': [.5, .7, .9]

# Init the Model
lda = LatentDirichletAllocation()

# Init Grid Search Class
model = GridSearchCV(lda, param_grid=search_params)

# Do the Grid Search
model.fit(tf)
```

executed in 18m 18s, finished 15:22:36 2019-05-20

Out[74]:

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=LatentDirichletAllocation(batch_size=128, doc_topic_p
rior=None,
             evaluate_every=-1, learning_decay=0.7,
             learning_method='batch', learning_offset=10.0,
             max_doc_update_iter=100, max_iter=10, mean_change_tol=0.0
01,
             n_components=10, n_jobs=None, n_topics=None, perp_tol=0.
1,
             random_state=None, topic_word_prior=None,
             total_samples=1000000.0, verbose=0),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_components': [5, 10, 15, 20, 25], 'learning_deca
y': [0.5, 0.7, 0.9]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Entrée [75]:

```
# Best Model
best_lda_model = model.best_estimator_

# Model Parameters
print("Best Model's Params: ", model.best_params_)

# Log Likelihood Score
print("Best Log Likelihood Score: ", model.best_score_)

# Perplexity
print("Model Perplexity: ", best_lda_model.perplexity(tf))
```

executed in 1.93s, finished 15:23:25 2019-05-20

```
Best Model's Params:  {'learning_decay': 0.5, 'n_components': 5}
Best Log Likelihood Score:  -3134752.236905373
Model Perplexity:  2123.7372780493597
```

Once we know which LDA model is the best, we check the topics allocation. We built a table that underlines in green the highest probability scores of a given topic in a document and thus selects the most relevant topic for the document.

Entrée [76]:

```
# Create Document - Topic Matrix
lda_output = best_lda_model.transform(tf)

# column names
topicnames = ["Topic" + str(i) for i in range(best_lda_model.n_components)]

# index names
docnames = ["Doc" + str(i) for i in range(len(db))]

# Make the pandas dataframe
df_document_topic = pd.DataFrame(np.round(lda_output, 2), columns=topicnames, index=

# Get dominant topic for each document
dominant_topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['dominant_topic'] = dominant_topic

# Styling
def color_green(val):
    color = 'green' if val > .1 else 'black'
    return 'color: {col}'.format(col=color)

def make_bold(val):
    weight = 700 if val > .1 else 400
    return 'font-weight: {weight}'.format(weight=weight)

# Apply Style
df_document_topics = df_document_topic.head(15).style.applymap(color_green).applymap
df_document_topics
```

executed in 2.28s, finished 15:24:00 2019-05-20

Out[76]:

	Topic0	Topic1	Topic2	Topic3	Topic4	dominant_topic
Doc0	0	**0.14**	**0.83**	0.03	0	**2**
Doc1	**0.62**	0	0	**0.16**	**0.22**	0
Doc2	0	0	0	**0.94**	0.06	**3**
Doc3	**0.66**	0	0	0	**0.34**	0
Doc4	0	0.08	**0.19**	0	**0.72**	**4**
Doc5	0	**1**	0	0	0	**1**
Doc6	0.01	**0.58**	0	**0.41**	0	**1**
Doc7	0	0	0	0.05	**0.95**	**4**
Doc8	**0.27**	0	0	**0.2**	**0.53**	**4**
Doc9	0	0	0	**0.4**	**0.6**	**4**
Doc10	0	0	0	0	**1**	**4**
Doc11	0	**1**	0	0	0	**1**
Doc12	0.04	0	0	**0.26**	**0.7**	**4**
Doc13	0	0	**0.4**	0	**0.59**	**4**
Doc14	0	0	**1**	0	0	**2**

We also check the distribution of topics across the corpus.

Entrée [77]:

```
df_topic_distribution = df_document_topic['dominant_topic'].value_counts().reset_inc
df_topic_distribution.columns = ['Topic Num', 'Num Documents']
df_topic_distribution
```

executed in 28ms, finished 15:24:43 2019-05-20

Out[77]:

	Topic Num	Num Documents
0	1	362
1	2	301
2	4	193
3	3	127
4	0	54

We want to show some keywords for each topic.

Entrée [78]:

```
# Show the n keywords for each topic
def display_topics(model, feature_names, no_top_words):
    for topic_idx, topic in enumerate(model.components_):
        print("Topic %d:" % (topic_idx) )
        print(" ".join([feature_names[i] for i in topic.argsort()[:-no_top_words - 1

display_topics(best_lda_model, tf_vectorizer.get_feature_names(), 10)
```

executed in 80ms, finished 15:25:04 2019-05-20

```
Topic 0:
poste parcelle entre activité comme parcelles convention complément zo
ne indemnité
Topic 1:
civil vente parties acte jugement sci travaux 000 date violé
Topic 2:
instruction pénale juge examen tribunal mise ordonnance faits avocat 2
018
Topic 3:
préjudice éléments monsieur employeur faute assurance dommages 2012 00
0 temps
Topic 4:
salarié employeur licenciement accord sociale entreprise salaire cause
conditions dispositions
```

This model seems to be pretty accurate. We can clearly see that the topics put under the spotlight different branches of the law:

- Topic 0 deals with property law
- Topic 1 deals with liability law
- Topic 2 deals with criminal law
- Topic 3 deals with general civil law
- Topic 4 deals with social law (= labour law)

Once we have defined our best model and found the ideal number of topics, we would like to plot the results. To this end, we use cluster algorithm, a basic class of unsupervised machine learning models. Clustering divides the data in groups according to their similarities and their properties. We will use k-means clustering from the package Scikit-Learn, as it is the most frequently used.

We use the k-means algorithm with 5 clusters - the number of topics of our ideal LDA model previously found. It would have also been possible to define the clusters with the probability scores calculated in the above table. To plot the graph, we need our two dimensions that correspond to the axis of the graph. To do so, we have to build a truncated Singular Value Decomposition (SVD) to reduce dimensionality and apply it to our best LDA model data.

Entrée [79]:

```
# Construct the k-means clusters
clusters = KMeans(n_clusters=5, random_state=100).fit_predict(lda_output)

# Build the Singular Value Decomposition(SVD) model
svd_model = TruncatedSVD(n_components=2)  # 2 components
lda_output_svd = svd_model.fit_transform(lda_output)

# X and Y axes of the plot using SVD decomposition
x = lda_output_svd[:, 0]
y = lda_output_svd[:, 1]

# Weights for the 5 columns of lda_output, for each component
print("Component's weights: \n", np.round(svd_model.components_, 2))

# Percentage of total information in 'lda_output' explained by the two components
print("Perc of Variance Explained: \n", np.round(svd_model.explained_variance_ratio_
```

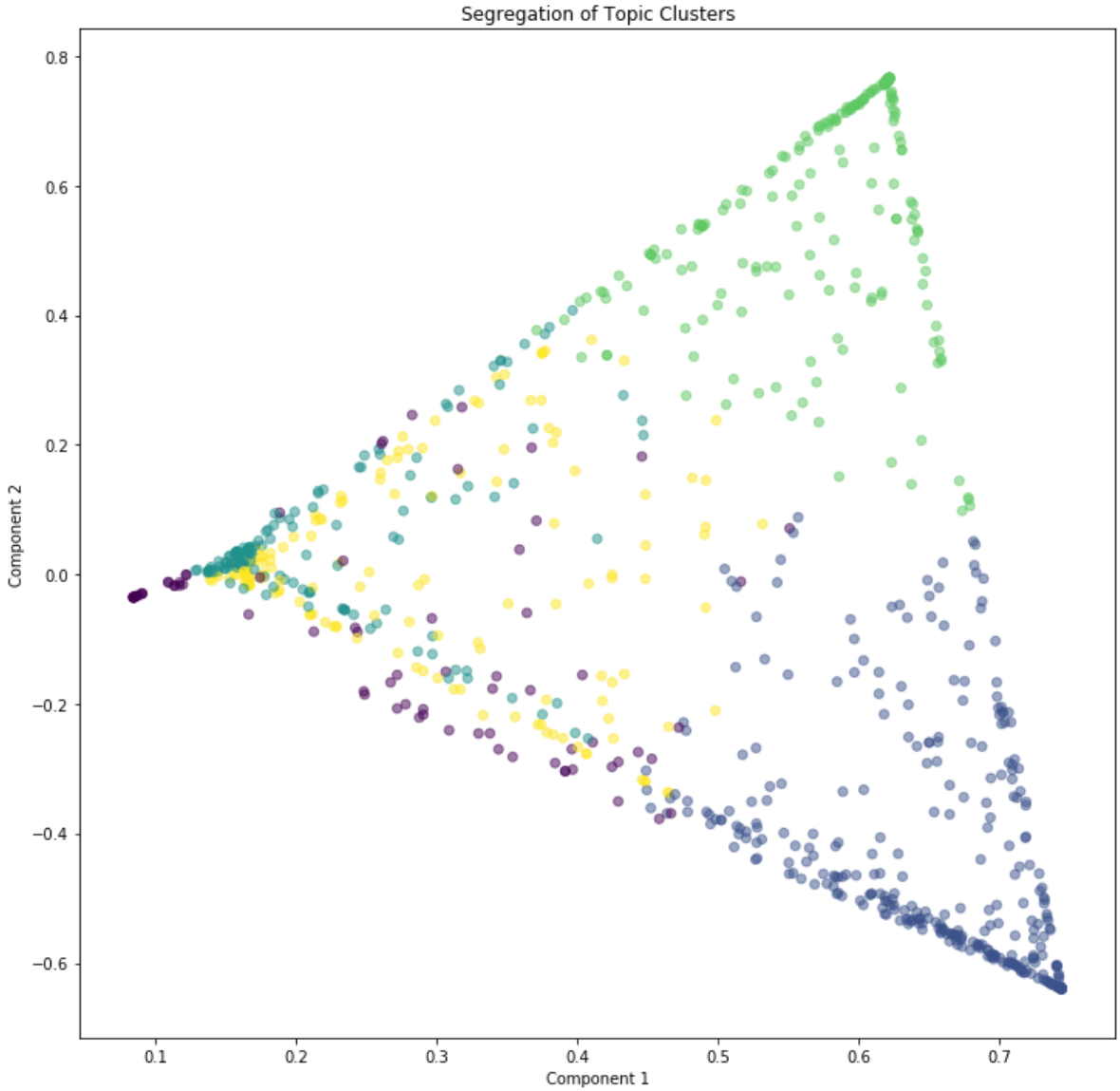
executed in 199ms, finished 15:28:00 2019-05-20

```
Component's weights:
 [[ 0.08  0.74  0.62  0.17  0.16]
 [-0.04 -0.64  0.77 -0.02  0.03]]
Perc of Variance Explained:
 [0.1  0.45]
```

Entrée [80]:

```
# Plot
fig, ax = plt.subplots(figsize=(12, 12))
ax.scatter(x, y, c=clusters, alpha = 0.5);
ax.set_xlabel('Component 1')
ax.set_ylabel('Component 2')
ax.set_title("Segregation of Topic Clusters");
```

executed in 553ms, finished 15:28:24 2019-05-20



We wish to visualize the clusters of documents in a 2D space using t-SNE algorithm mentioned in Part 2. A.

Entrée [81]:

```
colormap = np.array([color for name, color in mcolors.TABLEAU_COLORS.items()])

X_topics = best_lda_model.fit_transform(tf)
X_tsne = TSNE(n_components=2, perplexity = 60,  learning_rate= 60, n_iter = 8000).fi
```

executed in 49.2s, finished 15:29:38 2019-05-20

Entrée [82]:

```
_lda_keys = []
for i in range(X_topics.shape[0]):
    _lda_keys +=  X_topics[i].argmax(),

# create the dictionary with all the information
num_example = len(X_topics)

plot_dict = {
        'x': np.array(X_tsne[:num_example, 0]),
        'y': np.array(X_tsne[:num_example, 1]),
        'colors': colormap[_lda_keys][:num_example],
        'content': db[:num_example],
        'topic_key': _lda_keys[:num_example]
        }

# create the dataframe from the dictionary
plot_df = pd.DataFrame.from_dict(plot_dict)

#remove all the Na values and set everything to float
plot_df = plot_df.fillna(0)

plot_df['x'] = pd.to_numeric(plot_df['x'])
plot_df['y'] = pd.to_numeric(plot_df['y'])


# declare the source
source = bp.ColumnDataSource(data=plot_df)
title = 't-SNE Clustering of 5 LDA Topics'

# initialize bokeh plot
plot_lda = bp.figure(plot_width=1400, plot_height=1100,
                     title=title,
                     tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave",
                     x_axis_type=None, y_axis_type=None, min_border=1)

# build scatter function from the columns of the dataframe
plot_lda.scatter('x', 'y', color='colors', source=source);


# hover tools
hover = plot_lda.select(dict(type=HoverTool))
hover.tooltips = {"topic": "@topic_key"}

output_file("plot_lda.html")
show(plot_lda)
```

executed in 1.91s, finished 15:34:56 2019-05-20

2.2.2 C.2. Negative Matrix Factorisation (NMF) Method

We try another method : the Non-negative Matrix Factorisation (NMF) method. In practice, NMF is commonly used in topic modelling because it uses context, improving topic prediction. We decide to apply NMF to our corpus of texts. The results are the following:

Entrée [83]:

```
# NMF is able to use tf-idf
tfidf_vectorizer = TfidfVectorizer(max_df=0.95, min_df=2, max_features=500, stop_wor
tfidf = tfidf_vectorizer.fit_transform(db)
tfidf_feature_names = tfidf_vectorizer.get_feature_names()


# Run NMF
nmf_model = NMF(n_components=5, random_state=1, alpha=.1, l1_ratio=.5, init='nndsvd
nmf_output = nmf_model.fit(tfidf)

no_top_words = 10
display_topics(nmf_output, tfidf_feature_names, no_top_words)
```

executed in 7.49s, finished 15:35:48 2019-05-20

```
Topic 0:
vente travaux époux civil sci monsieur 000 parties préjudice entre
Topic 1:
pénale conseiller 2018 instruction greffier contre faits ordonnance fo
i examen
Topic 2:
licenciement salarié employeur salaire salariée indemnité sérieuse pos
te rupture réelle
Topic 3:
sécurité sociale chsct caisse accord maladie établissement recours cha
rge conditions
Topic 4:
acte 2018 pourvoi formé déclaré contre france rapport sociale 700
```

We can again assign clearly different branches of the law to the topics, except for the last topic:

- Topic 0 deals with property law
- Topic 1 deals with criminal law
- Topic 2 deals with social law (= labour law)
- Topic 3 deals with social security law
- Topic 4 remains quite hard to interpret

Entrée [84]:

```
# Create Document - Topic Matrix
nmf_output = nmf_model.fit_transform(tfidf)

# column names
topicnames = ["Topic" + str(i) for i in range(nmf_model.n_components)]

# index names
docnames = ["Doc" + str(i) for i in range(len(db))]

# Make the pandas dataframe
df_document_topic = pd.DataFrame(np.round(nmf_output, 2), columns=topicnames, index=

# Get dominant topic for each document
dominant_topic = np.argmax(df_document_topic.values, axis=1)
df_document_topic['dominant_topic'] = dominant_topic
```

executed in 269ms, finished 15:37:24 2019-05-20

When we take a closer look at topic allocation, a low number of documents are represented by the topic 4.

Entrée [85]:

```
df_topic_distribution = df_document_topic['dominant_topic'].value_counts().reset_inc
df_topic_distribution.columns = ['Topic Num', 'Num Documents']
df_topic_distribution
```

executed in 52ms, finished 15:37:45 2019-05-20

Out[85]:

	Topic Num	Num Documents
0	0	507
1	1	193
2	2	171
3	3	109
4	4	57

We decide to use again k-means algorithm to plot the clusters. We finally plot the clusters of documents in a 2D space using t-SNE algorithm.

Entrée [86]:

```
# Construct the k-means clusters
clusters = KMeans(n_clusters=5, random_state=100).fit_predict(nmf_output)

# Build the Singular Value Decomposition(SVD) model
svd_model = TruncatedSVD(n_components=2)  # 2 components
nmf_output_svd = svd_model.fit_transform(nmf_output)

# X and Y axes of the plot using SVD decomposition
x = nmf_output_svd[:, 0]
y = nmf_output_svd[:, 1]

# Weights for the 5 columns of lda_output, for each component
print("Component's weights: \n", np.round(svd_model.components_, 2))

# Percentage of total information in 'lda_output' explained by the two components
print("Perc of Variance Explained: \n", np.round(svd_model.explained_variance_ratio_
```

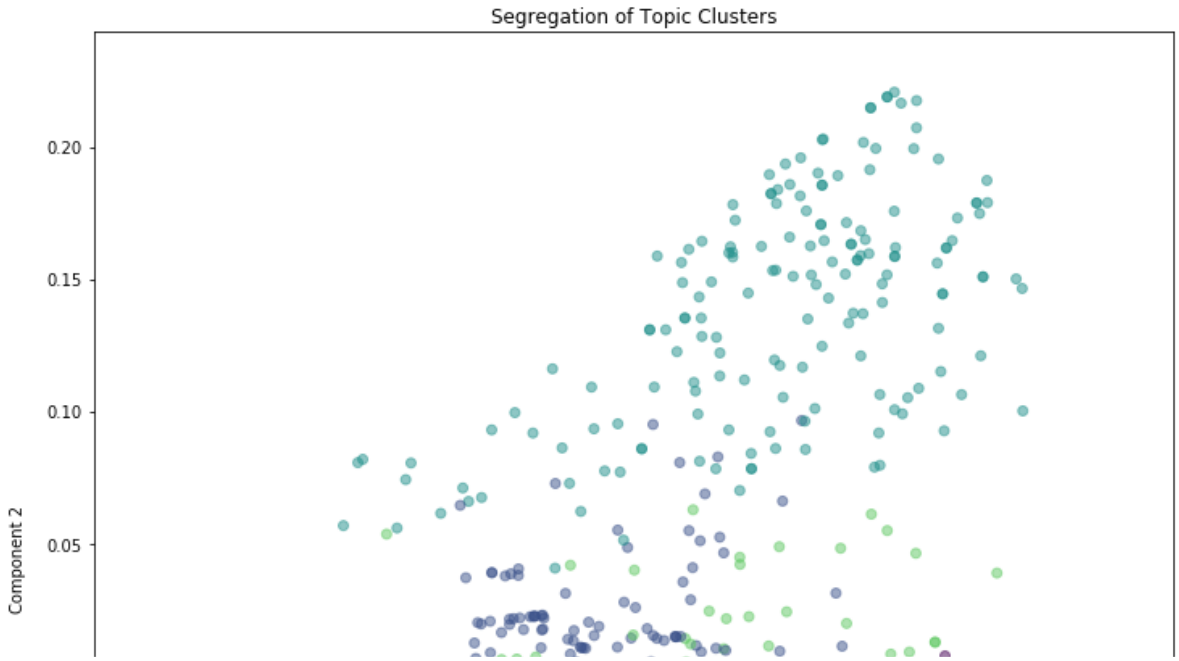
executed in 110ms, finished 15:38:06 2019-05-20

```
Component's weights:
 [[ 0.76  0.42  0.4   0.24  0.14]
 [-0.14 -0.55  0.81  0.08 -0.1 ]]
Perc of Variance Explained:
 [0.02 0.34]
```

Entrée [87]:

```
# Plot
fig, ax = plt.subplots(figsize=(12, 12))
ax.scatter(x, y, c=clusters, alpha = 0.5);
ax.set_xlabel('Component 1')
ax.set_ylabel('Component 2')
ax.set_title("Segregation of Topic Clusters");
```

executed in 542ms, finished 15:38:25 2019-05-20



Entrée [ ]:

```
X_topics = nmf_model.fit_transform(tfidf)
X_tsne = TSNE(n_components=2, perplexity = 60,  learning_rate= 60, n_iter = 8000).fi
```

execution queued 21:51:40 2019-05-12

Entrée [ ]:

```
colormap = np.array([color for name, color in mcolors.TABLEAU_COLORS.items()])

_lda_keys = []
for i in range(X_topics.shape[0]):
    _lda_keys +=  X_topics[i].argmax(),

# create the dictionary with all the information
num_example = len(X_topics)

plot_dict = {
        'x': np.array(X_tsne[:num_example, 0]),
        'y': np.array(X_tsne[:num_example, 1]),
        'colors': colormap[_lda_keys][:num_example],
        'content': db[:num_example],
        'topic_key': _lda_keys[:num_example]
        }

# create the dataframe from the dictionary
plot_df = pd.DataFrame.from_dict(plot_dict)

#remove all the Na values and set everything to float
plot_df = plot_df.fillna(0)

plot_df['x'] = pd.to_numeric(plot_df['x'])
plot_df['y'] = pd.to_numeric(plot_df['y'])


# declare the source
source = bp.ColumnDataSource(data=plot_df)
title = 't-SNE Clustering of 5 NMF Topics'

# initialize bokeh plot
plot_nmf = bp.figure(plot_width=1400, plot_height=1100,
                     title=title,
                     tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave",
                     x_axis_type=None, y_axis_type=None, min_border=1)

# build scatter function from the columns of the dataframe
plot_nmf.scatter('x', 'y', color='colors', source=source);


# hover tools
hover = plot_nmf.select(dict(type=HoverTool))
hover.tooltips = {"topic": "@topic_key"}

output_file("plot_nmf.html")
show(plot_nmf)
```

execution queued 21:51:40 2019-05-12

2.3 Word Embeding

Entrée [60]:

```
path="/Users/lamazere/Desktop/Pour mémoire/all_2018"
os.chdir(path)

try:
    model = gensim.models.Word2Vec.load("word2vec.model")  # gets back our stored mo
except IOError: # re-train only if required.
    model = gensim.models.Word2Vec(tokenized_sentences, min_count=1, workers=24)
```

executed in 2.90s, finished 22:39:32 2019-05-19

Entrée [61]:

```
read_files = glob.glob("*.txt")
temporary_file_name = "concatenated_texts.txt"
#creates a file containing all texts
with open(temporary_file_name, "wb") as outfile:
    for f in read_files:
        with open(f, "rb") as infile:
            outfile.write(infile.read())

all_texts_str = open(temporary_file_name,"r").read() #variable containing all texts
os.remove(temporary_file_name) #delete the file

sentences = nltk.sent_tokenize(all_texts_str)
tokenized_sentences = [nltk.word_tokenize(sentence) for sentence in sentences]

print("there in total this number of words in our corpus",len(all_texts_str))
```

executed in 8h 38m 16s, finished 07:17:57 2019-05-20

```
there in total this number of words in our corpus 147272258
```

Entrée [62]:

```
model.most_similar(positive=['titres'], topn=5)
```

executed in 641ms, finished 12:07:51 2019-05-20

Out[62]:

```
[('actifs', 0.7816277742385864),
 ('immeubles', 0.7470481395721436),
 ('biens', 0.7315032482147217),
 ('parts', 0.7016721963882446),
 ('cessions', 0.6811667680740356)]
```