



UNIVERSITY OF LIÈGE
SCHOOL OF ENGINEERING AND COMPUTER SCIENCE
FACULTY OF APPLIED SCIENCES

NLP Methods for Insurance Document Comparison

Author
Adrien SCHOFFENIELS

Supervisor
Ashwin ITTOO

Master's thesis carried out to obtain the degree of Master of Science in
Computer Science and Engineering by Schoffeniels Adrien

Academic year 2020-2021

Abstract

This work aims to study the different steps of a process that would allow to compare 2 different versions of a document. This process is decomposed into 4 parts: text extraction, text segmentation, text matching and text comparison, which have been the subject of research and experiments. Especially, one show that comparing the sections of the documents rather than the complete documents improve the quality of the comparison.

The text matching task, which is the part studied in more depth, is a variant of the classification task, with the difference that there are no general categories from which we try to classify. Instead, each document has a unique set of classes, corresponding to each section, that can not be known in advance. This has many implications, mainly the fact that traditional classifiers cannot be used, as one cannot create training data for this task.

Different natural language processing (NLP) methods have been compared on the text matching task. For this purpose, a small dataset of pairs of documents with their matching has been built, and metrics inspired from the confusion matrix for the classification task has been designed, to be able to assess the performances of the different models. The models compared are term frequency (TF), TF-IDF, Word2vec combined with the Word Mover's distance, Doc2vec, BERT and RoBERTa. The different experiments show that more complex models are not suited for this matching task, and that it is preferable to use simple statistical models.

Acknowledgements

I would like to thank the data science team at NRB, that allowed me to work on this subject, and to come to occupy their office whenever I wanted to. In particular, I would like to thank Samy Doloris, who made his proof of concept available to me, who was always quickly available to answer my questions, for his proofreading of my work, as well as for his precious advise.

Likewise, I would like to thank Pr. Ashwin Ittoo for helping me to define the scope of the work, for his recommendations, and for his answers to my questions.

Finally I would like to thank my family and friends for supporting and encouraging me during this period.

Contents

1	Introduction	1
1.1	Context	1
1.2	Problem Definition	1
1.3	Contributions	2
2	Previous work	5
2.1	Text extraction	5
2.2	Text segmentation	5
2.3	Section matching	6
2.3.1	Angular similarity	6
2.3.2	Frequency vectors	7
2.3.3	Word embeddings	9
2.3.4	Paragraph vectors	14
2.3.5	Contextual word embeddings	16
2.4	Sections comparison	22
3	Practical approach	24
3.1	Text extraction	24
3.1.1	Text extraction packages	24
3.1.2	Preprocessing	25
3.2	Text segmentation	26
3.2.1	Identify the sections	26
3.2.2	Separate titles and sections	27
3.3	Section matching	27
3.3.1	General implementation	28
3.3.2	NLP methods	30
3.4	Sections comparison	32
4	Results and observations	33
4.1	Dataset	33
4.2	Metrics	33
4.3	Experiments for the matching task	34
4.3.1	Comparison of different ways to use Word2vec	34
4.3.2	Matching without thresholding	35
4.3.3	Matching with thresholding	38
4.3.4	Non-exclusive matching	40
4.3.5	Speed comparison	42
4.3.6	Combining titles and sections	42

4.4	Text segmentation issues	43
4.5	Qualitative analysis of the output	43
5	Conclusion	47
A	Results	53

Chapter 1

Introduction

1.1 Context

Standard form contracts, terms and conditions, company policies, all of these are part of our daily lives. Driving your car, going to work, browsing on internet, all these actions are subject to rules and conditions written in these documents. However, these documents often suffer from a lack of interest: a study conducted in 2016 by the European commission on consumers' attitudes towards terms and conditions states that less than 1% to 65% of people questioned read them depending on the situation, and that 26.6% of them already faced a problem due to insufficient knowledge about them [Joa+16].

Moreover, these types of documents are updated frequently, and it is uncommon to find some sort of "changelog", a document that only points out the modifications that were made from one version of a document to another (as we can usually find for software programs). However, this could be quite useful, as the length of these documents is one of the main reasons why people usually do not read them [Joa+16]. The existence of such a summary of the modifications could encourage people to learn about the changes. The applications of such a tool that will bring to light the modifications made to a document does not limit to this usage: a company could also use it for example to keep track of the novelties and changes of its competitors, to be able to adapt their offers and services.

In this thesis, one will explore and compare different approaches using natural language processing methods (NLP) to address this issue and develop a tool that will show the modifications that were made from one version of a document to another, specifically for insurance documents. An example of the desired output of this tool is shown in Figure 1.1.

1.2 Problem Definition

The development of the tool that will highlight the modifications between 2 versions of a same document can be decomposed into different sub-modules:

- **Text extraction:** The first one consists in the extraction of the useful text from a pdf file, to be able to process it.
- **Text segmentation:** The second one will segment the raw text into smaller text

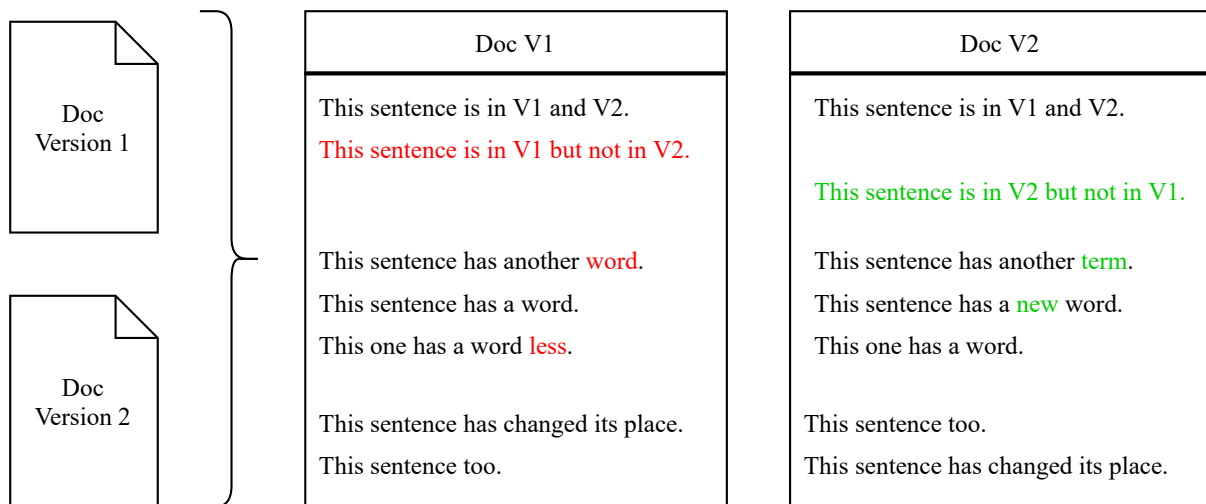


Figure 1.1: An example of result that the desired tool would output, allowing to easily observe the modifications that have been made to a document through its different versions.

Sentences that changed their place are not new content and should therefore not be highlighted.

chunks, for then doing the comparison between these smaller text pieces rather than between two big monolithic text sequence.

- **Matching task:** The third one will find which text fragment from the second version of a document corresponds to which text fragment of the first version of the same document. This part requires the use of NLP methods.
- **Comparison task:** The last module will then compare the pairs of corresponding text fragments by finding the differences between them. This module also displays these modifications properly.

A schematic view of the decomposition of the tool is shown in [Figure 1.2](#)

1.3 Contributions

These different modules raise several questions that will be the subject of research and experiments:

- How can we segment a text to obtain smaller text chunks that are suitable for comparison ?
- Is it really useful to perform this segmentation to compare two documents ?
- How can we match efficiently the obtained text chunks from one document to the corresponding text chunks from another version of this document ?
- How can we measure the quality of these matchings ?
- How to highlight the modifications that were made from one text chunk to its corresponding updated chunk ?

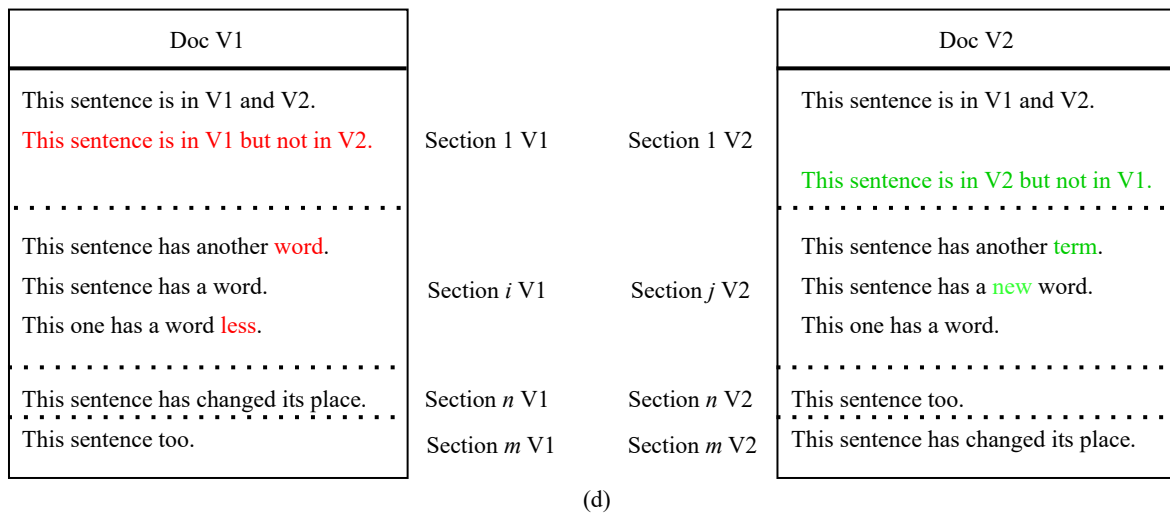
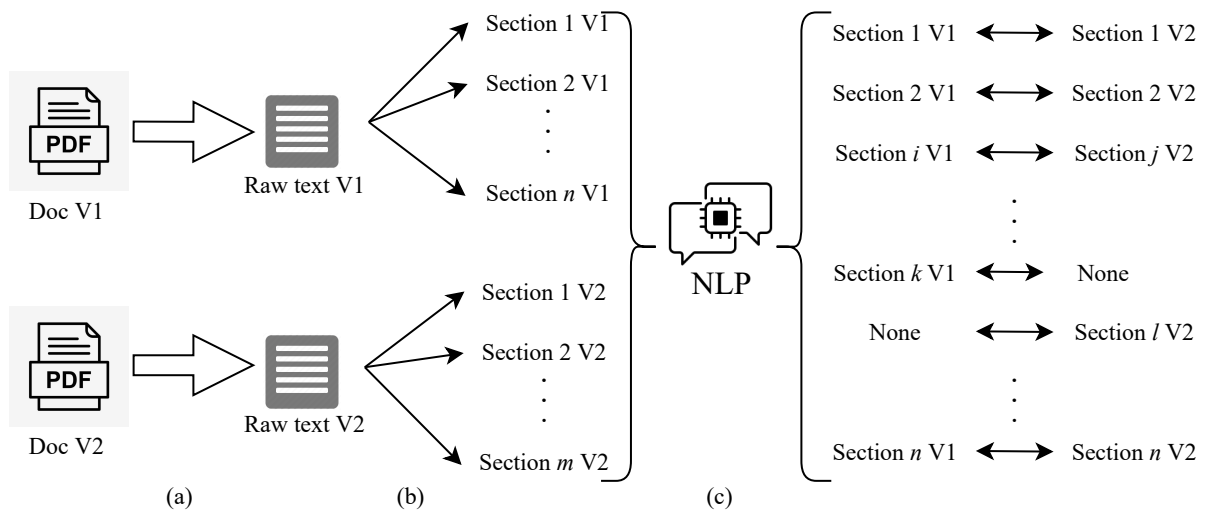


Figure 1.2: Decomposition of the tool in different parts.

- (a) parses pdf documents to extract the raw text,
- (b) segments the raw text into (sub)sections,
- (c) matches the corresponding sections of the two versions of the document,
- (d) compares the pairs of matching sections and highlight the modifications.

- How to deal with the impossibility of creating a training dataset (since the categories for each document are unique) ?

Several contributions arise from the efforts to answer these questions. Especially, the matching module addresses a task for which no studies involving recent NLP methods have been found. The matching task is a variant of a classification task, except that the classes are not known in advance and are unique to a document, which has many implications. For instance it is impossible to create a training dataset which prevents the use of traditional classification models requiring a training phase. The different contributions are listed below:

- The review and comparison of different NLP methods applied to the matching task. The methods studied for this purpose are Term Frequency (TF), TF-IDF, Word2vec with the Word Mover's Distance, Doc2vec, BERT and RoBERTa.
- The finding that more complex models, that are usually better in the majority of NLP tasks, are not the most appropriate ones for the matching task.
- The creation of a small dataset of pairs of insurance documents with the correct matchings, to assess the performances of the different models
- The design of different metrics (inspired from the confusion matrix and accuracy of the classification task), to be able to compare quantitatively the different NLP methods implemented.

Chapter 2

Previous work

This section aims at reviewing the previous work that has been done to tackle the different tasks addressed by the modules of the tool. Especially, emphasis will be placed on the work achieved in the field of natural language processing to compare and classify text documents, since the purpose of the section matching module is a derivative of a text classification task.

2.1 Text extraction

The text extraction is the first step of the process and is therefore extremely important, as the quality of the raw text extracted will impact on the performances of the other modules (and the NLP models). Although, since it is a common task, a lot of different packages implement text-extraction features by either transforming the pdf pages into images and scanning the characters, or by directly using the raw code from the pdf file. Since the latter method seems to be more reliable, different text-extractor packages based on this method have been tried and tested. Their performance will be discussed later.

2.2 Text segmentation

To be able to compare two versions of a document, one assume that it is preferable to decompose the documents and compare their sections. Since some documents may be several dozens of pages long, determining the differences between these complete documents would be a complex task, and the illustration of the differences would be unreadable, providing no real useful information. Moreover, the algorithm of the last module used to compare two text chunks, makes a syntactic comparison of these chunks. We are thus interested in comparing small text blocks that are similar, rather than monolithic text chunks.

Common text segmentation procedures are unfortunately not useful for the case at hand. The most common ones are used to segment a text into words or sentences, which is not adequate to compare sections.

Other segmentation techniques based on the idea of segmenting a document into its topics have been explored in the past years. These techniques include segmentation using Hidden Markov Models [BM03], using clustering [Lam+07], or based on topic modeling algorithms

[Mis+11] using for instance derivatives of Multinomial Mixtures (MM) or of the Latent Dirichlet Allocation (LDA) [BNJ03], usually used for doing topic modeling.

Unfortunately, these topics based segmentation techniques cannot be used in the case at hand. These methods require to know the different topics of the document beforehand, which is impossible. Moreover, the sections have contents that are too similar to be able to make small text blocs with meaningful separations.

Therefore, another solution must be used. The one employed consists in taking advantage of the section numbers that delimit the sections, by using regular expressions to identify them in the raw text.

2.3 Section matching

The segmentation module allows to extract the sections from the raw text. However, to be able to compare the section from one document to its corresponding section in the other document, one first need to find which section of the first version of the document corresponds to which section of the second version of the same document.

This task is more complicated than it seems. Trivial solutions are in fact not efficient. The first idea would be to simply match the section numbers, as we rely on these to segment the text. However, this method does not work if the location of a section in the document changes from one version to another, or if a new subsection is introduced in-between two sections for example.

The second idea would be to extend the first one by matching the titles of the sections. Although this may seem more reliable than the first approach, it still suffers from critical limitations. It can happen that the title of a section changes from one version of a document to another. In addition, and even worse, sometimes sections do not start with a title.

A more elaborated approach consists in computing the similarity between sections, and match the sections that are the most similar. Measuring text similarity is a quite old task that has been the subject of numerous studies in recent years since the emergence of neural networks. The different approaches to measure text similarity are reviewed below.

2.3.1 Angular similarity

Most of the text similarity techniques rely on the angular similarity. The angular similarity, sometimes referred to as the cosine similarity, is a similarity measure based on the cosine of the angle between two vectors in an Euclidean space. It can be computed using the formula of the dot product between two vectors \mathbf{x} and \mathbf{y} :

$$\begin{aligned}\mathbf{x} \cdot \mathbf{y} &= \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta \\ \Rightarrow \text{sim}(\mathbf{x}, \mathbf{y}) &= \cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}\end{aligned}$$

where θ is the angle between the two vectors \mathbf{x} and \mathbf{y} , and $\|\mathbf{x}\|$ is the Euclidean norm of the vector $\|\mathbf{x}\|$, defined as

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \dots + x_p^2}$$

where \mathbf{x} is a vector of dimension p .

Intuitively, if the 2 vectors point in the same direction, meaning that they have the same components (apart from a factor), then the angle θ is equal to 0 and therefore the similarity measure will be equal to 1.

On the other hand, if the 2 vectors are orthogonal, meaning that they have none non-zero component in common, then the angle θ is equal to $\pi/2$ and therefore the similarity measure will be equal to 0.

2.3.2 Frequency vectors

To be able to apply the cosine similarity measure to documents, one need to represent these documents as vectors. The simplest approaches consists in statistical methods.

Term-frequency (TF)

Term-frequency [HKP12] consists in creating vectors from documents by simply counting the occurrences of each word in the documents.

This is done by first creating one-hot-encoding vectors of the vocabulary of the corpus¹. Each word of the vocabulary is mapped to a vector which has only one value set to one (the other being set to 0). The position of this value set to 1 is different for each word.

The term-frequency vector for a document is then obtained by summing up the vectors corresponding to the words appearing in this document. This implies that each component of this vector represents a word from the vocabulary of the corpus, and that the value of each component is equal to the frequency with which the corresponding term appears in the document. An illustration of one-hot-encoding vectors and term-frequencies vectors for a corpus of 2 sentences can be found in Figure 2.1.

Note that once these vectors are built, one are not limited to the angular similarity to measure the similarity between them. For example the Euclidean distance could also be used, but it is not recommend as the score will highly depend on the length of the documents. This is not the case of the cosine similarity, making it more reliable to measure the similitude between documents. Other measures may be more suitable depending on the nature of the documents. For example, when dealing with high-dimensional data applications, the improved sqrt-cosine metric performs better [SW17].

The advantages of combining the term-frequencies vectors and the angular similarity are multiple. Firstly, it is really easy to implement. It can thus be used as a first draft, to make experiments and quickly get results. Another usage that benefits of its simplicity would be to have a quick and rather efficient way to check if two documents are identical. It is more suitable than a pure string comparison, as it does not output a binary response, and therefore can handle space changes, typos, errors that have been corrected and so on. Another advantage, compared to more sophisticated methods that will be discussed later on is that it does not suffer of the out-of-vocabulary problem. The vocabulary is obtained directly from the corpus used, which is not necessarily the case when using more complex models. It can therefore be suitable to compare small text portions, such as titles. Indeed, titles are made up of few words, and a common approach in NLP consists

¹A corpus is a set of documents

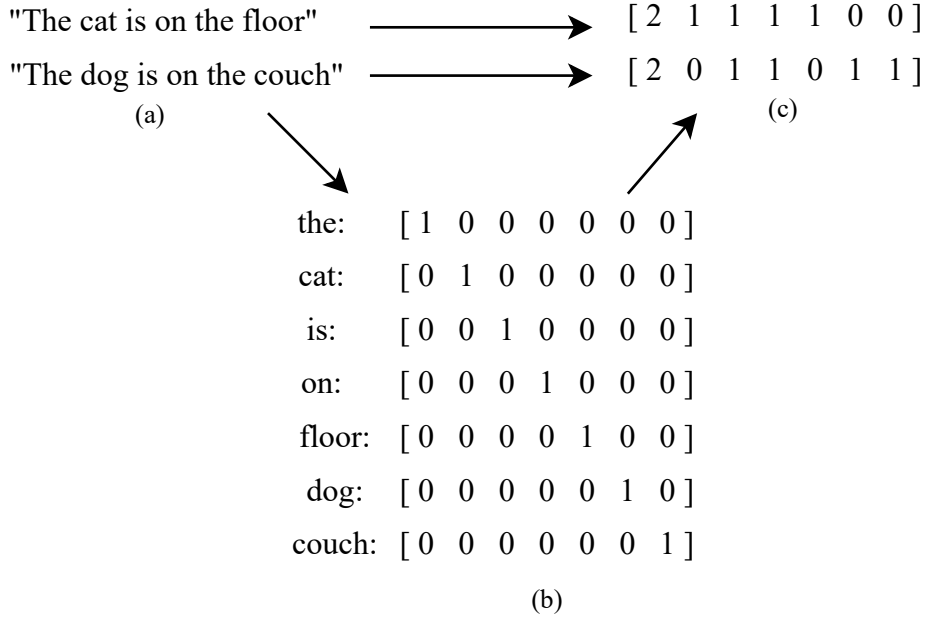


Figure 2.1: Example of Term-frequencies vectors and associated one-hot-encoding vectors. (a) is the corpus, defining the vocabulary of 7 terms, (b) are the one-hot-encoding vectors associated to the words of the vocabulary, (c) are the term-frequencies vectors of the 2 sentences of the corpus, obtained using TF.

in removing stop words², which implies that one can end up with comparing text inputs of 1 or 2 words. If these words are out-of-vocabulary for complex models, they can not be used by these models and they have nothing left to compare. Concerning the shortcomings, the term-frequencies vectors do not capture any semantic information about the words, which could be worthwhile in a classification task. Moreover, the dimension of the vectors is directly linked to the size of the vocabulary of the corpus, which involves having to deal with very large vectors.

Term frequency - inverse document frequency (TF-IDF)

Term frequency - inverse document frequency (TF-IDF) overcomes an important limitation of the term-frequency method used to create the term-frequencies vectors, which is the fact that it assumes that each word in the vocabulary has the same importance. This is obviously not the case: a word that appears in every document does not provide any useful information for a classification task. Instead, a word that appears in very few documents gives more information about the content of the document than common words, and is therefore more likely to be discriminant. Consequently, TF-IDF takes into account this information to build the frequency vectors, by multiplying the term frequency by a function of the inverse of the number of documents in which the term appears. The component for the term t of the document d defined as follows:

$$tf-idf = tf_{t,d} \log \frac{N}{df_t} \tag{2.1}$$

²Stop words are words that are commonly used in a language, such as "a" or "the" and that do not convey a lot of information.

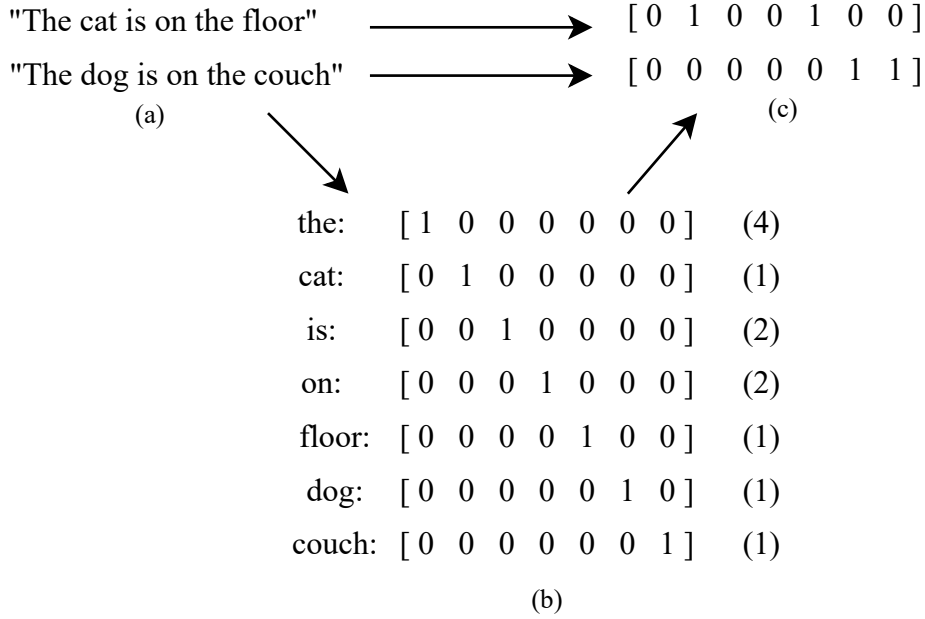


Figure 2.2: Example of TF-IDF with a corpus of size 2.

(a) is the corpus, defining the vocabulary,

(b) are the words of the vocabulary, with their one-hot-encoding vectors and their frequency in the corpus,

(c) are the term-frequencies vectors computed using TF-IDF, with a log of base 2.

In this case, the words that are in both sentences of the corpus lead to the logarithm equal to 0 (otherwise, the logarithm is equal to 1).

where $tf_{t,d}$ is the frequency of term t in the document d , N is the number of documents in the corpus, and df_t is the number of documents from the corpus in which term t appears.

Intuitively, this value is higher when t occurs frequently in a small subset of the corpus, and lower when t occurs rarely in a document or in a large subset of the corpus.

An example of TF-IDF applied to a corpus of 2 sentences is shown in Figure 2.2.

Although TF-IDF usually allows to enhance the performances of the term-frequency method on classification tasks, the limitations mentioned for TF are not resolved, which means that alternative methods to the frequency vectors may be explored.

2.3.3 Word embeddings

Term-frequencies vectors are able to represent words as discrete symbols, which combined with a similarity metric can quite well perform at a classification task. However, these one-hot-encoding vectors suffer from some limitations that would be nice to be solved. The first one concerns the dimension of the vectors. Since there is one vector component for each word of the vocabulary, one end up with very large and sparse vectors. The other one is that such vectors are not semantically intuitive: they do not capture the meaning of the words. However, one can easily understand that words that appear in the same context may have a similar meaning.

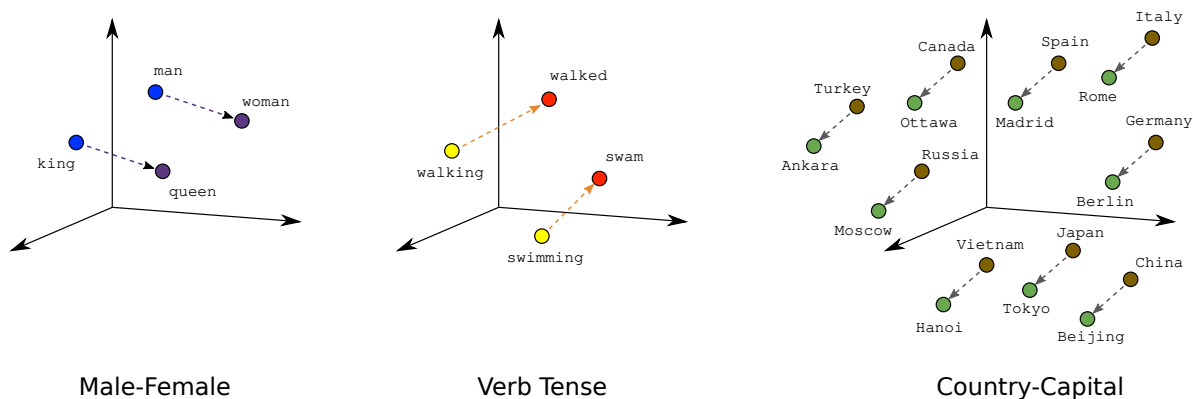


Figure 2.3: Visualization of the geometrical relationships between semantically related words in the vector space of word embeddings (source: <https://developers.google.com/machine-learning/crash-course/images/linear-relationships.svg>)

To handle these limitations, word embeddings have been introduced. Word embeddings are generally defined as unsupervisedly learned numerical representation vectors of words, whose relative similarity correlates with semantic similarity [MS16]. This similarity can be observed by performing some algebraic operations on vectors [Mik+13]. For example, with proper embeddings, one can compute

$$X = v(\text{"king"}) - v(\text{"man"}) + v(\text{"woman"})$$

where $v(w)$ represents the embedding of the word w .

When searching for the closest word to X in the embedding space (using the cosine distance), one finds the word "queen". Needless to say, the similarities captured are not limited to the genre of the words. Figure 2.3 shows different visualizations of the geometrical relationships between semantically related words.

There are 2 main categories of methods to create word embeddings:

- Count-based methods
- Neural-based (or predictive) methods

Both categories are based on the idea that the meaning of a word can be defined by its neighbors.

Count-based methods, which were popular in the 90s, are based on vectors representing explicitly co-occurrence statistics between words, as well as reweighting heuristics to lower the dimensions of the embeddings. The main approaches in this category of methods include Latent Semantic Analysis/Indexing (LSA/LSI) [Dee+90] and Pairwise Mutual Information (PMI) [CH02].

These methods are decreasing in popularity nowadays, since the numerous advances in the field of neural-based methods. Moreover, comparative studies have shown that neural-based methods are more efficient and perform usually better than count-based methods

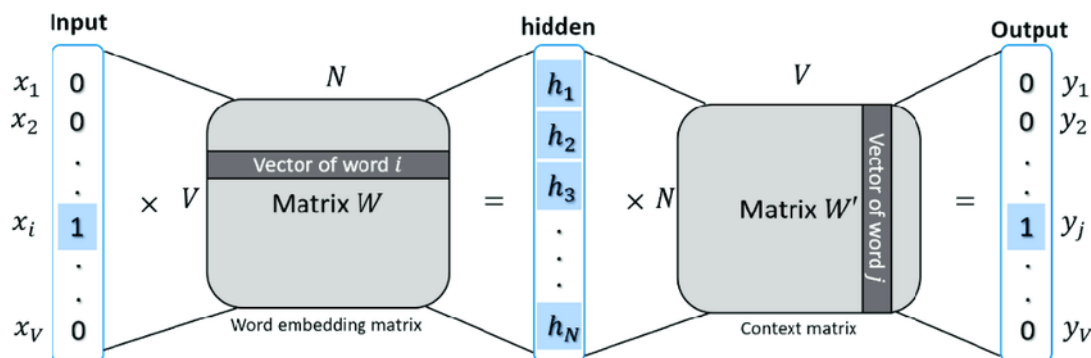


Figure 2.4: Architecture of the Word2vec Model (for the Skip Gram model). The embeddings are the vectors of the word embedding matrix, the internal representations of the word of the neural network. (from [OY19])

[NCB17; MYZ13; Zhi+13]. Therefore, count-based methods were not considered as potential solutions for this comparative study.

Neural-based methods construct embeddings in a different way than count-based methods, which is non convex. The word vector consists in the internal representation of a neural network, trained for instance to predict a word based on its neighbors. These methods became very popular with the introduction of Word2vec [Mik+13], and later with the follow up works, GloVe [PSM14] and FastText[Boj+16]. For this work, Word2vec has been employed as it is the most popular implementation, which in addition makes it easier to find pre-trained models. Therefore, this model will be reviewed in more details.

Word2vec

Word2Vec, introduced in 2013 by Mikolov et al. [Mik+13], is based on a shallow neural network model of one hidden layer, parametrized by word vectors. Its architecture is detailed in Figure 2.4.

It is trained by iteratively optimizing a given objective function and using backpropagation. Basically, it learns to distinguish actually co-occurring groups of words, from random groups of words. There are actually 2 algorithms to do this:

- Continuous Bags of Words (CBOW)
- Skip-Gram (SG)

CBOW learns to predict a target word from its neighbors. SG however, does the opposite, by learning to predict the neighboring words, given some target word. An illustration of these two architecture is shown in Figure 2.5. SG is the most popular implementation. In both architectures, the input(s) and output(s) of the models are words represented using one-hot-encoding vectors.

The mathematical details of the model are rigorously explained in a paper written by Xin Rong called "word2vec Parameter Learning Explained" [Ron16].

Once the model is trained sufficiently (it predicts correctly most of the words), the embeddings are obtained by taking the representation of a word at the first layer after the input.

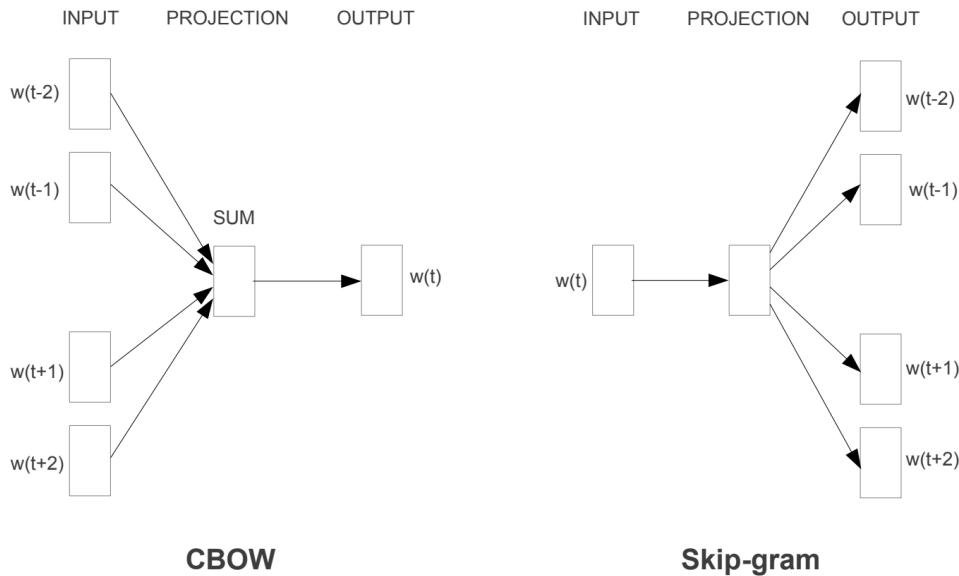


Figure 2.5: Difference between the Continuous Bag of Words (CBOW) and the Skip-gram architectures of the Word2vec model. $w(t)$ is the target word. (From [Mik+13])

Word Movers' Distance

The word embeddings obtained with word2vec are highly effective at capturing similarities between words. However, in the case at hand, one would like to compare sections, rather than just words. A first approach discussed to handle this issue is called the Word Mover's Distance.

The Word Mover's Distance [PW08; PW09; Kus+15] is a distance function working at the document level. In its original paper, it is defined as a metric "*measuring the dissimilarity between two text documents as the minimum amount of distance that the embedded words of one document need to 'travel' to reach the embedded words of another document*" [Kus+15]. In other words, it computes the optimal way to move from the distribution of one document to another one. It is in fact an instance of the well-known Earth Mover's distance, a metric based on a linear optimization problem introduced in 2000 [RTG00], for which many efficient solvers have been developed since then.

The measure of dissimilarity is computed by summing the Euclidean distances between the words from the first document and their corresponding word in the second document. To find the correspondences of the words between the two documents, one also rely on the Euclidean distance, and assume that the corresponding words are the one with the smallest Euclidean distance between them. An illustration of the calculation of this metric is shown in Figure 2.6.

The complete mathematics of the Word Mover's Distance are detailed in the paper that introduced it [Kus+15].

The whole point of using word embeddings instead of term-frequencies vectors is also shown in Figure 2.6: since word embeddings allow to find similarities between synonyms and close-related words, the Word Mover's Distance between the sentences "Obama speaks to the media in Illinois" and "The President greets the press in Chicago" is not null. However, if using term-frequencies vectors and removing stop-words, a similarity metric

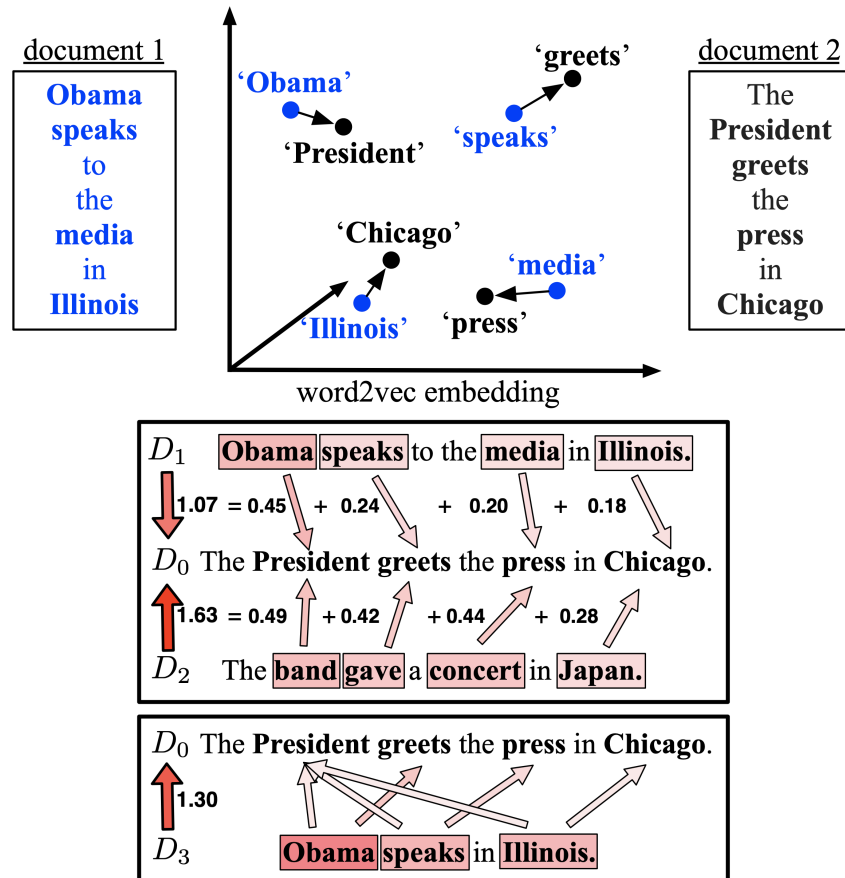


Figure 2.6: Example of the calculation of the Word Mover's Distance between documents. The image at the top shows how the correlation between the non-stop words of 2 documents are found (smallest Euclidean Distance between the words), The middle image shows how the metric it is computed (by summing the Euclidean Distances of the matched words). Note that the Word Mover's distance is a dissimilarity measure, meaning that D_1 is the most similar to D_0 . The image at the bottom shows what happens when the documents are not of the same length: the distance is computed by moving words to multiple similar words. (from [Kus+15])

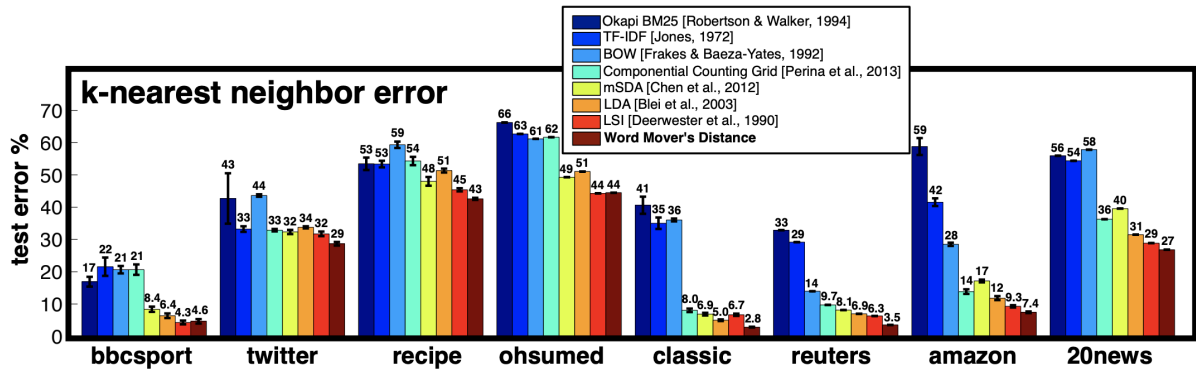


Figure 2.7: Evaluation of different canonical and state of the art method, including the Word Mover’s distance, for kNN -classification. (from [Kus+15])

would give a result of 0, since the sentences have no words in common.

The authors of the paper introducing the Word Mover’s Distance [Kus+15] evaluated this metric on 8 datasets and compared it to 8 canonical and state of the art methods for kNN -classification. The results show that the Word Mover’s Distance perform the best on 6 out of the 8 datasets, and almost as good results as the best method for the 2 other datasets. These results are summarized in Figure 2.7.

2.3.4 Paragraph vectors

Another solution to deal with the issue of word embeddings not capturing the similarities between text segments but words is based on paragraph vectors. They have been introduced in a paper called "Distributed Representations of Sentences and Documents" [LM14].

The model for creating paragraph vectors is inspired from, and therefore very similar to, the model presented by Mikolov et al [Mik+13], namely word2vec. The additional feature is the introduction of a paragraph vector, which is similar to other word vectors, except that it is unique to each paragraph (whereas word vectors are shared among all paragraphs), acting as a memory remembering what is missing from the current topic of the paragraph.

As for the word2vec, there are 2 different algorithms to train the model:

- Distributed Memory Model of Paragraph Vectors (PV-DM)
- Distributed Bag of Words version of Paragraph Vector (PV-DBOW)

PV-DM is based on the Continuous Bags of Words version of word2vec. In this case, the paragraph vector is concatenated with context words and trained to predict the next word, as shown in Figure 2.8.

PV-DBOW is based on the Skip-Gram version of word2vec. In this version, the model is trained by using the paragraph vector to classify if words belong to the paragraph or not, as illustrated in Figure 2.9. This version of the model is more efficient (and therefore more used) than the Distributed Memory model.

Once the model has been trained using one of these techniques, the word embeddings and the classifier parameters (used in the DBOW version) are no more needed for the

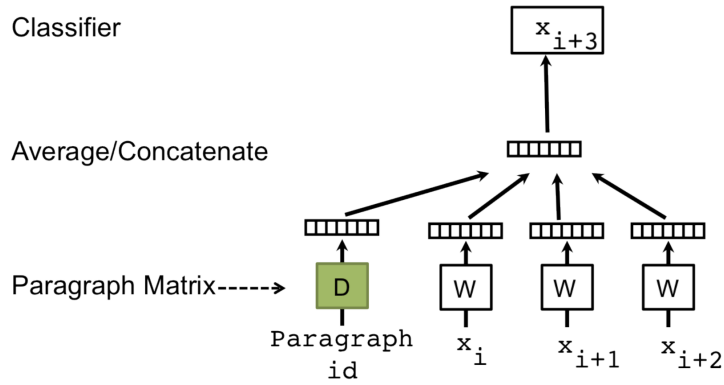


Figure 2.8: Distributed Memory Model of Paragraph Vectors. The paragraph vector is concatenated with the context words to predict the next word in the sentence. (from [DOL15])

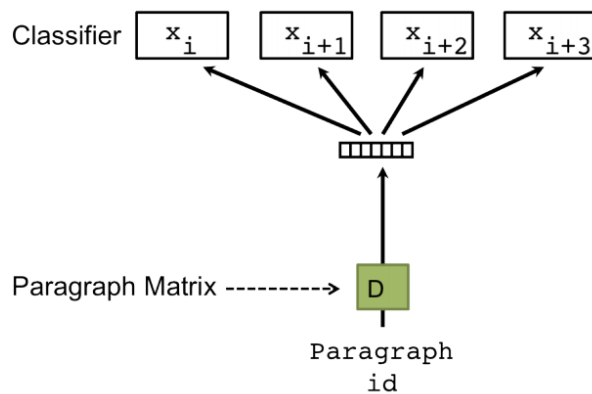


Figure 2.9: Distributed Bag of Words version of Paragraph Vector. The paragraph vector is used to classify if random words belongs to the paragraph or not. (from [DOL15])

inference task. Instead, the paragraph vectors are initialized randomly, and fine-tuned using backpropagation.

The advantages of this method compared to the usage of word2vec with the Word Mover's Distance includes the fact that there is no need of a complex metric to measure the similarity between text sections: the cosine similarity is all it takes to do it. Moreover, since the paragraph vector is created based on the entire paragraph, it is able to capture the context of it, whereas word embeddings are the same regardless of the paragraph it is located in. However this implies that the paragraph vectors must be learned specifically for the type of corpus they are used for, which is not the case of word2vec for which one can pre-train a model and use it for every instance.

Doc2vec

A paper focusing on paragraph vectors at the document level [DOL15], presents experiments conducted on 2 datasets. They considered complete documents as *paragraphs*, and showed that these (document) paragraph vectors perform better or at least as well as other methods at determining which documents are the most similar to each other. However, no comparison was made with the usage of the combination of word2vec and the Word Mover's Distance, making it interesting to evaluate both methods for this section matching task.

2.3.5 Contextual word embeddings

The introduction of word embeddings have led to important improvements in many tasks of NLP, including text classification, as developed in section 2.3.3. However, word embeddings have still one big flaw, which is that the embedding for a word is unique, regardless of its context.

For example, the word "mean" appears in the sentences "You have been mean to me" and "The mean of the results is 5". However, it is clear that they do not have the same meaning. Nevertheless, they will have the same word embedding. In other words, word embeddings are unable to deal with homonyms, because it does not take into account the context in which the words appear.

As a consequence, contextual word embeddings have been introduced more recently. The first approaches to create contextual word embeddings were based on the creation of one word embedding per word sense, by extending the skip-gram model of word2vec (each word is associated to a global vector, and each sense of word is associated to an embedding and a context cluster) [Nee+15]. Another approach relies on the enrichment of the word embeddings using sub-word information [Wie+16].

BERT

The most popular approach nowadays is called BERT (Bidirectional Encoder Representations from Transformers) [Dev+19]. It takes the advantages of several previous works in the domain of contextual word embeddings, especially through the use of transformers.

The transformer is a model architecture presented in a paper called "Attention Is All You Need" [Vas+17]. This architecture was originally designed for machine translation by

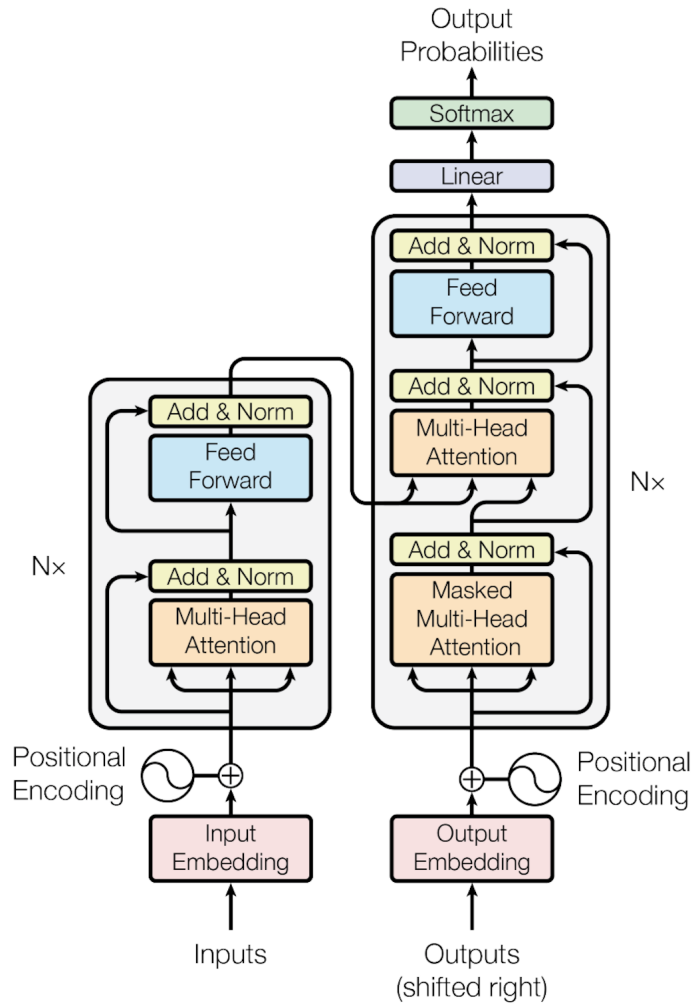


Figure 2.10: Architecture of the Transformer model. The decoder (right) has access to the previous words it predicted, as well as the complete input sequence through the output of the encoder (left). (from [Vas+17])

using an encoder-decoder structure. The encoder reads the input sequence and map it to a higher dimension vector, while the decoder uses the output sequence (shifted from one position) as well as the result of the encoder to produce the translation. This architecture therefore allows to take into account the entire input sequence to predict a word. The transformer architecture is shown in Figure 2.10.

As the name of the paper suggests, the architecture uses attention mechanisms. These mechanisms were introduced in a paper called "Neural Machine Translation by Jointly Learning to Align and Translate" [BCB16]. These mechanisms are used when trying to translate a target word to put emphasis on the words of the input sequence that are related to this target word. That way, when the decoder makes a prediction for a certain word, it has access to the previous input words and knows which words are the most related to this target word. For example, in the sentence "He is driving his car", the attention mechanism will learn during the training stage that "driving" and "car" are related in this sentence. Consequently, when predicting a word to translate the word "car", it will take into account the other words of the sentence, but will give more importance to the word "car" to make its decision.

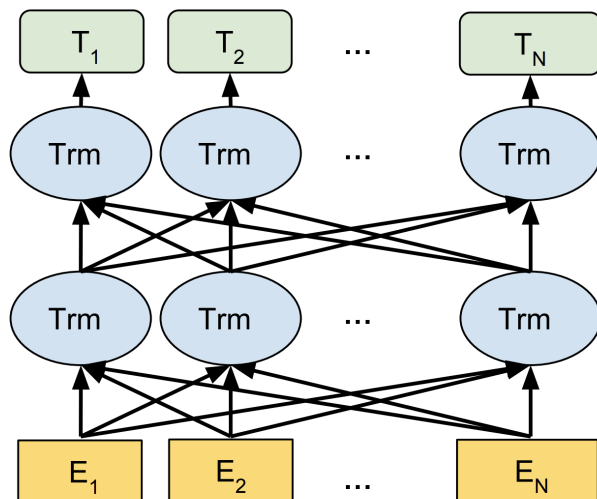


Figure 2.11: BERT’s architecture. It uses a bidirectional transformer (the representations are based on the left and right context at each layer). (from [Dev+19])

Since the goal of BERT is to generate a language model, from which one can create the contextual word embeddings, only the encoder part of the transformer is needed (as a reminder, the decoder was used to generate the translation of the input sequence). The output of the model will therefore be a set of vectors (one per input word), which are not (yet) the word embeddings of the input words. This allows to use the model for various tasks, by adding a module at the output of the model and fine-tuning this part, on the basis of a language model that has already been trained on a very large corpus. Since the transformer uses the whole input sequence at once, it is said to be bidirectional (since it can predict a word based on its whole surroundings, left and right). The architecture of the model is detailed in Figure 2.11.

The training of BERT is done using 2 strategies:

- Masked Language Modeling (MLM)
- Next Sentence Prediction (NSP)

The Masked Language Modeling strategy consists in learning to predict a word from a sentence that has been masked (or hidden). A complete sentence is given as input, from which the target word has been replaced by a token "masked". The output vector corresponding to the position of the masked word is then used to predict the masked word. An illustration of this strategy is shown in Figure 2.12.

The Next Sentence Prediction strategy is done at the same time. During the training phase, the model is actually given a pair of sentences (both having a masked word), separated by a special token. This way, the model can also learn to predict if the second sentence is the subsequent sentence to the first one in the original corpus or not. The embedding for each word that is fed to the model is a combination of the token’s embedding, the sentence embedding (indicating if the word belongs to the first or the second sentence), and the positional embedding, indicating the position of the token in the sentence. This strategy is illustrated in Figure 2.13.

During the training, the two strategies are used at the same time by optimizing over a combination of their losses.

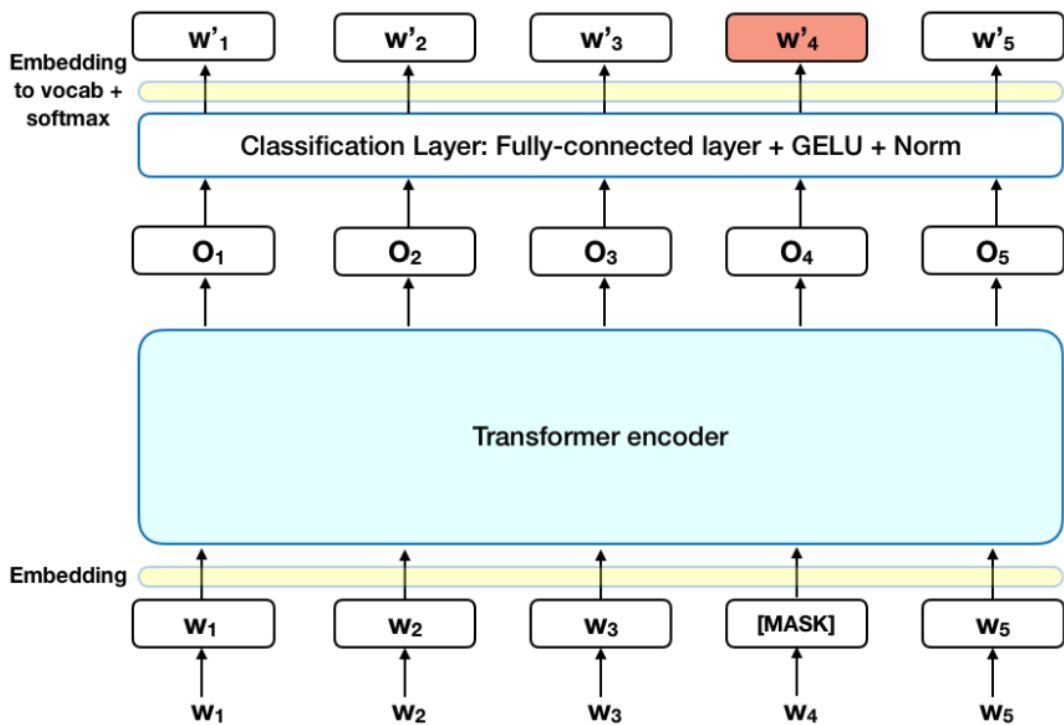


Figure 2.12: Mask Language Modeling task for training BERT. A random word (w_4) of a sequence is marked as "masked". The whole sequence is fed to the transformer. The output is used with a classification layer, the embedding matrix (to get the vocabulary dimension) and softmax to get the probability for each word of the vocabulary. (from <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>)

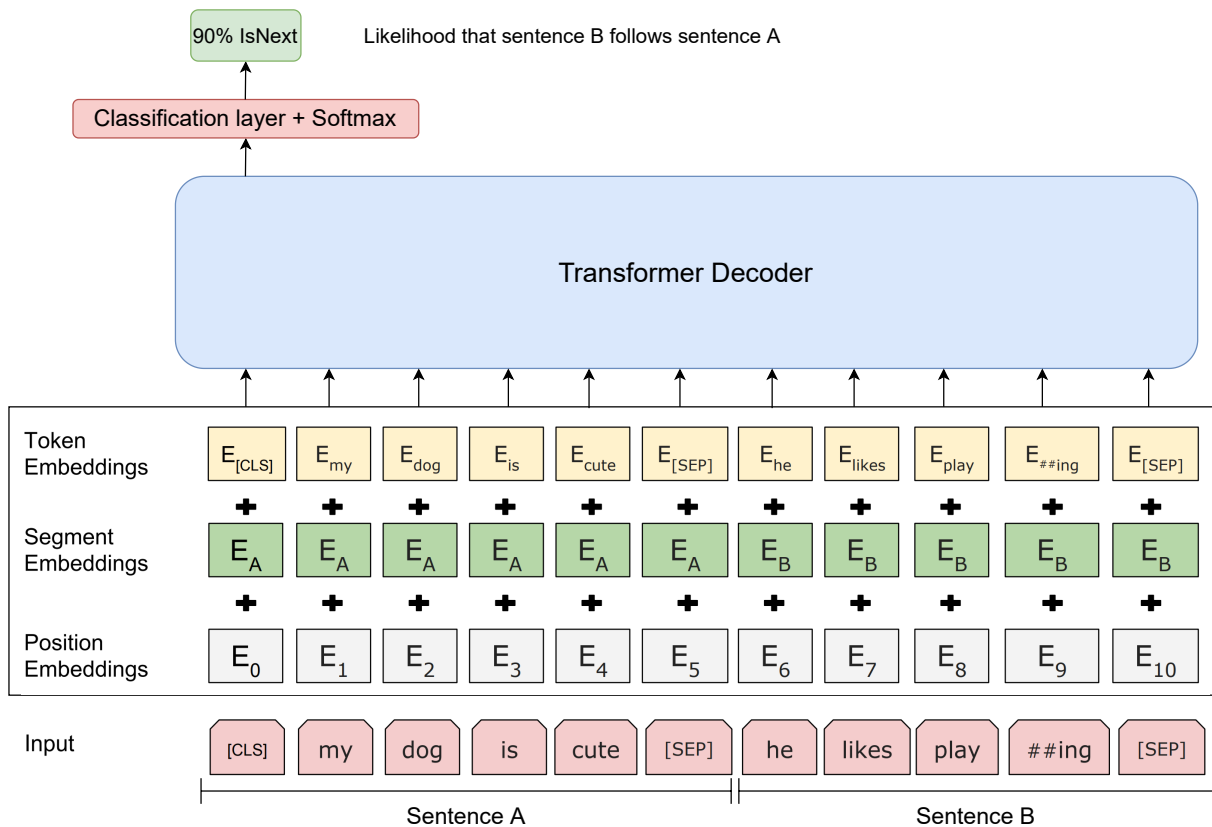


Figure 2.13: Next Sentence Prediction task for training BERT. The input embeddings are linear combinations of the token embeddings, the segment embeddings and the position embeddings. The output corresponding to the [CLS] token is used with a classification layer and softmax to get the probability that the second sentence follows the first one. (partly from [Dev+19], modified)

Once BERT has been trained, it is easy to adapt it for another NLP task, by only adding a small layer to the model. For example, in the case of a sentiment analysis task, one must only add a classification layer on top of the output corresponding to the [CLS] token used for the Next Sequence Prediction task (similarly as for the Next Sequence Prediction task). Using their trained model, the authors of the original paper introducing BERT fine-tuned it for 11 NLP tasks, and achieved new state-of-the-art results for these tasks.

A French version of BERT, called FlauBERT [Le+20] has been made publicly available in 2020. This pretrained model will be the one used in this work.

To perform the section comparison task, one take the output vectors corresponding to the [CLS] tokens for each section, to obtain section embeddings, and then one use a similarity measure (the cosine similarity, defined in section 2.3.2) to match the sections between them.

RoBERTa

Following the release of BERT (which is not only the architecture, but the complete pretrained model with its weights that the authors of the paper made publicly available), several papers proposed methods and architectures to enhance BERT, such as XLM [LC19] and XLNET [Yan+20]. At the time of the realisation of this work, the most recent version which was available with a French pretraining was RoBERTa [Liu+19], for Robustly Optimized BERT Approach. The authors of the paper claimed to achieve better or at least as good results as the aforementioned methods.

The core idea behind RoBERTa is to train more robustly BERT, which is according to the authors significantly under-trained. To reach this objective, they brought 4 modifications during the training phase, compared to BERT:

- First, they trained the model over more data.
- Secondly, they removed the Next Sequence prediction Task of the training stage.
- Next, they trained the model over longer text sequences.
- Finally, they introduced dynamic masking. For the BERT model, the input sequences are duplicated and masked randomly 10 times, before the start of the training stage. This means that the model will only see 10 versions of each sequence. Dynamic masking (in RoBERTa) is done during the training phase, which means that a sequence can be used with more than 10 different masking versions.

Since the Next Sequence Prediction task has been removed, and as we use the token that is originally used in BERT for this task to generate the section embeddings, it may be interesting to try the two versions, BERT and RoBERTa to verify if the removal of the Next Sequence Prediction task during the training phase actually decreases the performance of the model for the task at hand.

The French version of RoBERTa has also been made publicly available in 2020, and is called CamemBERT [Mar+20]. This pretrained model will be used in this work, to compare with the other methods described earlier.

2.4 Sections comparison

The principal algorithm used in practice to find the differences between two versions of the same text sequence is Myer's Diff algorithm [Mye86]. The algorithm has been proposed in 1986 and is still widely used nowadays. For example, it is the default algorithm used by Git, the open-source software used to track changes in sets of files.

The algorithm is based on the well-known longest common subsequence problem, which consists in finding the longest subsequence of characters that is common to a set of sequences (usually 2). A subsequence is different from a substring in the sense that the characters of the subsequence do not need to be consecutive in the initial sequences.

The longest common subsequence problem is dual to the shortest edit script problem which consists in finding the smallest set of operations that converts a file into another. This set of operations corresponds therefore to the modifications that were made to a file to get the other one, and can be used for the task at hand to show the modifications made from one version of a document to another.

Myer's diff algorithm relies on the fact that finding the longest common subsequence is equivalent to finding the shortest path in a graph, as explained in Figure 2.14. Myer proposed a convenient greedy algorithm to solve this problem.

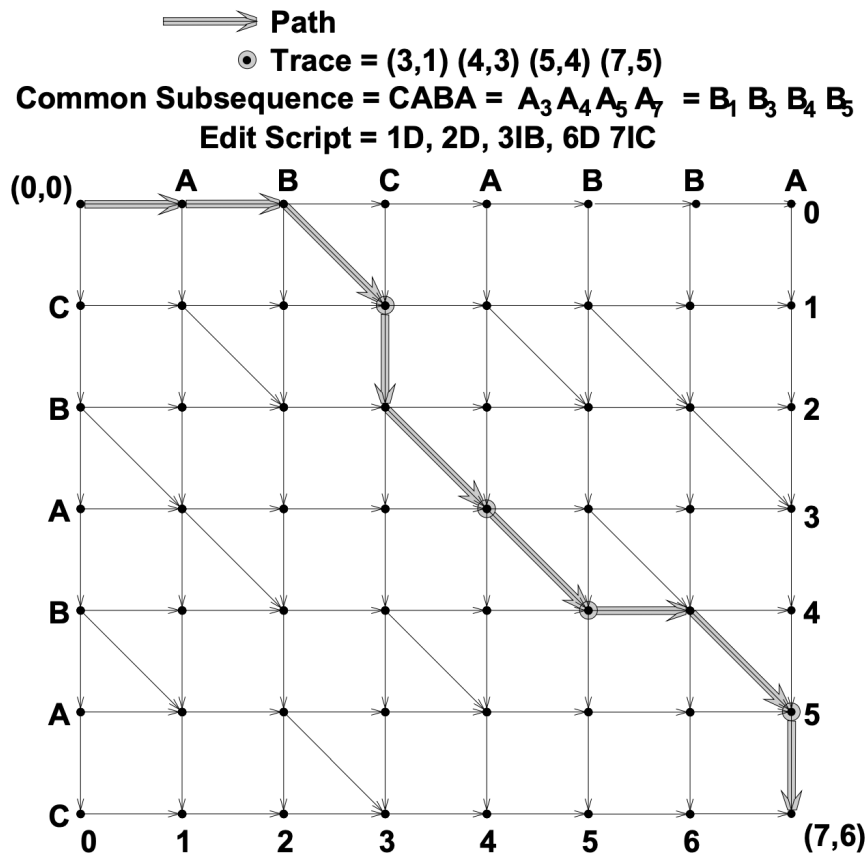


Figure 2.14: Edit graph used to solve the longest common subsequence in Myer's diff algorithm. The first sequence is "ABCABBA" and is written at the top. The second one is "CBABAC" and is written at the left side. The goal is to go from the top left corner to the bottom right one, by taking the shortest path. Vertical edges correspond to a deletion from the first sequence, while horizontal edges represent insertions in the second sequence. Diagonal edges are common characters. (from [Mye86])

Chapter 3

Practical approach

The purpose of this section is to explain from a more practical point of view how the different modules of the tool have been imagined and how the methods discussed in Chapter 2 have been implemented and used.

3.1 Text extraction

3.1.1 Text extraction packages

Several packages exist in python to extract the text from pdf files. 3 of them have been tested for this work:

- PyPDF2 (<https://pythonhosted.org/PyPDF2/>).
- Pdfminersix (<https://pdfminersix.readthedocs.io/en/latest/>).
- PyMuPDF (<https://pymupdf.readthedocs.io/en/latest/>).

Their performances are discussed below.

PyPDF2

PyPDF2 is a pure python package based on PyPDF, which is claimed to be an advantage as it should run on all python platforms. However it is quite old, as its last version has been published in 2016. Moreover, it is more a tool aimed to edit files. It is not specially focused on the extraction of content.

The results obtained using this package are really poor. It fails at extracting the full sentences, and only gets few words (as can be seen in Figure A.1 in the appendix). The package is therefore unusable.

Pdfminersix

Pdfminersix is a tool that focuses on getting and analyzing text data from pdf files. It may therefore be more suited than PyPDF2. It is also written in pure python. The package is a fork maintained by the community, meaning that there are more recent releases than for PyPDF2.

The results are more satisfying, but nonetheless still not optimal. Sometimes, some whitespace characters are missing (as shown in Figure A.2 in the appendix). This would obviously causes issues in the other modules of the tool: these words would be out-of-vocabulary when using word2vec, or words will be missing for the term frequencies vectors.

To try to solve the issue, a dynamic programming approach has been tried. Its goal is to infer the whitespace locations using a dictionary of existing words. Simply said, each word is associated to a cost, and the algorithm tries to place the whitespaces in order to minimize the sum of the costs using backtracking. To determine the cost of a word, one processed as follows: if the sequence of characters forms an unknown word, the cost is infinite. If the sequence forms an existing word, the cost is inversely proportional to the frequency of this word in the entire set of the corpus. To check if a sequence of character forms an existing word, one rely on a list of 336531 french words (<http://www.pallier.org/liste-de-mots-francais.html>), based on the French Gutenberg dictionary. Although this algorithm works fine most of the time, it is not optimal since it won't work if there is an unknown word in the sequence of characters. Therefore, another approach than PyPDF2 has been tried.

PyMuPDF

PyMuPDF is different from the two other packages. It is in fact a python binding to MuPDF (<https://www.mupdf.com/>), an open-source software framework written in C, that especially implements a pdf file parsing and rendering engine. It is widely used and well-known for its top performance and high quality rendering. Unlike the other packages, this one successfully extracts the whole text from the documents of the corpus, without missing any whitespace character. Therefore, this one has been chosen as the extracting package to use.

3.1.2 Preprocessing

Once the whole text has been extracted, some observations were made. Some parts of the text are unnecessary for the task (such as the front page, the table of content and so on). Some parts may even be considered as noise, such as the page numbers and the headers that are extracted and put in the middle of sections. Therefore, before extracting the content of the pdf, a small preprocessing is done.

When using the `pdf_to_txt` module implemented, a document version and the number of pages to remove from the beginning and from the end of the document are required. This way, using PyMuPDF, one can restrict the regions of the pages to extract the text from (by excluding the header and footer parts), and remove the unnecessary pages at the beginning and at the end of the document. The regions of extraction have no other choice, unfortunately, than being hardcoded. However, the documents of a company are in general in the same format. For example, for the documents of AXA, every document from 2013 to 2018 were in the same format. The appearance changed in 2019, but since, they are also all the same. Therefore, the function `pdf_to_txt` may take the argument `doc_v`, which indicates the year of the document as well as the company it was issued from (for example, it can be 'AXA <= 2018', for the documents of AXA pre 2019). The `doc_v` can also be determined automatically if the name of the company and the year of

the documents are written in the file name. If the argument is not given, the user is asked to enter the different parameters manually.

Using these preprocessing and PyMuPDF, one can thus extract only the useful portions of text in a proper way, which is primordial for the other parts of the tool.

3.2 Text segmentation

As a reminder, the purpose of this module is to separate the entire text into its sections and subsections.

As explained in section 2.2, common text segmentation techniques such as word/sentence segmentation or topic based segmentation do not work in the case at hand, since it is impossible to know the topics of the sections in advance, and since the topics of the different sections are too similar. One thus have to do it in a different way.

3.2.1 Identify the sections

The approach used consists in using regular expressions to match the sections numbers, which usually follow a certain pattern. For example, in the case of the documents from AXA, each section number is of the form "1.2.3.", which is a number followed by a dot, repeated a certain number of times. This can easily be matched using a regular expression of the form " $(\d\\.)+$ ". However, this also implies that it must be hardcoded at some point for each document type. This can also be done using the `doc_v` argument introduced for the text extraction module, which can be associated with a certain type of section pattern.

The regular expressions works fine at finding the section numbers, but it is unable to guess if a matched string is indeed a section number or not. For example, there can be references to other sections anywhere in the text, as well as numbered lists inside a section, which will also be matched.

The first idea explored to try to solve this issue was to use the table of contents to extract the section numbers. This idea quickly fell by the wayside since too long section numbers (in particular *paragraphs*¹ for AXA documents) are not included in the table of contents. Therefore, another approach used to solve this issue is to match the numbers in the reading direction, and consider a number as a section number only if it logically follows the previous one. For example, if the current section number is "2.3.", the allowed next matchings are:

- "2.3.1",
- "2.4",
- "3."

However, this solution does not completely solve the issue. For example, if there is a numbered list in the first section ("1."), then the second item (which is identified by "2.") will be matched (as well as the other following items). Needless to say that this is an undesired behaviour.

¹Paragraphs have section numbers of the form "X.X.X.X."

To try to solve this issue, one rely on the idea that matching a numbered list would create a shorter list of section numbers. Indeed, these numbered lists are usually short and therefore if they were to be matched, this would probably happen at the beginning of the document rather than at the end, skipping a lot of sections. As a result of this observation, one would like to match as much logically following section numbers as possible. This can be done using an adaptation of a recursive implementation of the longest path algorithm.

Once the sections numbers have been correctly identified using the described solution, one can use the consecutive section numbers to identify the starting and ending positions of each section. In this perspective, one do not only store the section numbers matched, but the `match` object returned by the regular expression package, which stores the position of the match in the text.

3.2.2 Separate titles and sections

Some further observations may be done before simply providing these complete sections to the next module, which matches these sections. Firstly, it may be convenient to separate the section title from the section body. This would allow for example to match sections based on their title, as one can reasonably think that sections having the same, or closely related titles should be matched. Furthermore, sections that only consists of a title (for example if a section starts directly with a subsection) do not have content to compare, and are therefore useless.

In order to split the title from the rest of the section, one again relies on regular expressions and observe that a title ends with one ore more newline character, followed by an uppercase letter (indicating the start of a new sentence). The regular expression used to match the end of a title is the following:

$$\backslash n \backslash s * [A - Z][^A - Z? \backslash s] * \backslash s [^?]$$

This regular expression tries to take into consideration a rare case where the title is written in uppercase, by not matching a newline character followed by a complete word in uppercase.

The sections for which this regular expression does not match anything are treated differently depending on the case. If the text chunk ends with a dot or with a semicolon, it means that the section does not have a title, and the text chunk must be used in the section matching module. The title and the section content for these sections are set to be the same in this case, to not have to deal with empty titles in the next module. In the other hand, if the text chunk ends with something else, then the section probably only consists of a title, and is therefore discarded.

3.3 Section matching

The section matching module is certainly the most important one, as it is the one that will allow to compare the different NLP methods discussed in section 2.3, and allow to answer to most of the research questions addressed in this work.

3.3.1 General implementation

A class has been created for each NLP model described in Chapter 2 (TF, TF-IDF, Word2vec, Doc2vec, Bert, RoBERTa). Except for the Doc2vec class, each one is similar and has similar methods.

The general procedure to match the sections follows the following pattern:

1. Create the tokens for the model
2. Create the vectors/embeddings
3. Compute a cost/similarity matrix
4. Match the most similar sections
5. (Unmatch not similar enough sections)

This procedure is detailed below.

Create the tokens for the model

The first step is to transform the text sequence into something that is meaningful to the model, because each method does not require the same input. For example, the word2vec model takes as input an array of words, whereas the BERT model requires the use of its builtin tokenizer to create the input tokens, that takes as input a string of all the words separated by a space.

However, what is common to each method, is the addition of the possibility to remove the stopwords from the text (which, as a reminder, are words that do not provide any useful information, such as "the", "a" and so on). Usually, pre-trained models simply discard them, so there is no need to pass them to the models.

Create the vectors/embeddings

The second step consists in using the model to create frequency vectors, word embeddings, paragraph embeddings or contextual word embeddings depending on the model, using the tokens created in the previous step.

Compute a cost/similarity matrix

The next task consists in using the vectors/embeddings and a distance or similarity measure to build respectively a cost matrix or a similarity matrix.

A cost (similarity) matrix is a matrix where each row correspond to a section of a version of a document while each column correspond to a section of the other version of the document. The value of a cell is equal to the dissimilarity (similarity) between the sections of the 2 versions of the document. An example of a cost matrix is shown in Figure 3.1.

This matrix is computed iteratively, by computing the distance (similarity) between each section of the first version of the document and each section of the second version of the document.

V2 \ V1	1.1.	1.2.	1.3.	2.
1.1.	0.01	0.2	0.91	0.9
1.2.	0.25	0.03	0.93	0.85
1.3.	0.85	0.75	0.26	0.97
1.4.	0.72	0.84	0.07	0.76
2.	0.94	0.86	0.77	0

Figure 3.1: Example of a cost matrix. The rows correspond to the sections of the second version of the document, while the columns correspond to the sections of the first version of the document. A cell corresponds to the dissimilarity between the corresponding sections of the 2 versions of the document. In the case at hand, a new section has been introduced in the new version of the document at position "1.3", shifting the old section of the same number. Section "2." did not change at all.

Match the most similar sections

The following step uses the cost (or similarity) matrix that has just been computed to match the closest (or the most similar) sections.

There are two ways to obtain these matchings: do an exclusive matching, or do a non-exclusive matching. An exclusive matching means that a section may be matched only to one other section (this is depicted in Figure 3.1, where the section "1.3." of V2 does not match with a section of V1). The non-exclusive matching means that a section could be matched to two or more sections (in the case of the cost matrix of Figure 3.1, the section "1.3." of V2 could also have matched the section "1.3." of V1). This second possibility has been implemented with the understanding that from one version to another, a section could have been split into two sections, or that two sections may have been merged together.

The exclusive matching is done using the `linear_sum_assignment` function of the well-known SciPy library (<https://www.scipy.org/>). This function solves the linear sum assignment problem, consisting in matching the vertices of a bipartite graph while minimizing the sum of the weights of the vertices. The two versions of the document are in this instance of the problem the two sets of the bipartite graph, and the cost matrix represents the weights of the vertices.

The non exclusive matching is done by simply taking the minimum (maximum) value of each row of the cost (similarity) matrix.

Unmatch not similar enough sections

The last step is not mandatory, but consists in unmatching the matched sections that have a similarity that is not significant enough. This is done because by default, the previous step makes the most matches possible, even if for that it needs to match sections that are not really similar. This has been done by adding a threshold value. For instance, in the example of Figure 3.1, if a matching threshold was set at 0.05, then sections "1.3."

of V1 and "1.4." of V2 would not match. Likewise, if we used the non-exclusive matching feature, the sections "1.3." of both versions would have matched, although the similarity is less significant compared to the similarities of the other matchings. Using the threshold would prevent this from happening. In the next Chapter, the performances of the methods using this threshold or not will be compared.

3.3.2 NLP methods

Term-frequency (TF)

The angular similarity method is quite simple and has therefore been implemented from scratch. The vocabulary is created by taking the union of the words of the two sections one would like to estimate the similarity (to reduce the sparsity of the vectors, since the words that do not appear in both sections do not provide any information). The 2 term-frequencies vectors are then created by counting the number of times each word appears in each of the 2 sections to compare. Finally, the cosine similarity is computed between these 2 vectors. This operation is repeated for each pair of sections possible to fill in the cost matrix. Since the vocabulary is defined on the basis of a pair of sections, the term-frequency vector of a section has to be recomputed for each other section it is associated with.

TF-IDF

The term-frequency inverse-document-frequency method is also quite simple and has also been implemented from scratch. This time, the vocabulary must be the union of the words of both versions of the document (to be able to create the *inverse document frequency* terms). The frequencies vectors are therefore computed only once per section, and the cosine similarity is computed between each pair of frequency vectors of the sections of the 2 versions of the document.

Word2vec with the Word Mover's Distance

For word2vec's embeddings to be strong, one needs to train the model on a huge corpus. Unfortunately, the available corpus of insurance documents is quite small, and training word2vec only on these documents would probably lead to overfitting.

Instead of training a model over the few available documents, different approaches have been implemented:

- The first one consists in training a Word2vec model *on the fly* during run time, using the sections of the 2 documents one would like to compare. Of course such models will very likely overfit the text input, but this is not really a problem since the model is trained only with the aim of creating word embeddings for this same text input.
- Training a model each and every time is not really convenient and time consuming. Therefore, the second method is the conventional one, which is to train the data on a large corpus once and for all, and use this model in every instance. The data used for this purpose is a corpus made available by the European Commission, consisting of 8947709 sentences originating from *Légifrance*, France's official website for publishing legislation, regulations and legal information (<https://tinyurl.com/5dpeb97u>).

- The last approach is to use a pre-trained model, which is a model trained by someone else or by a group of persons on a very large quantity of data, and made publicly available. The pre-trained model used has been trained on *frWaC*, a corpus of 1.6 billion words obtained by crawling ".fr" websites, by Jean-Philippe Fauconnier [Fau15].

The performances of these different approaches will be discussed later.

In order to train these models or to use the pre-trained model, one rely on the open-source topic modeling library Gensim [RS10].

As explained in section 2.3, one must use the Word Mover's Distance to be able to compare sets of word embeddings. The Gensim library also implements the Word Mover's Distance, and it has therefore been used as is as dissimilarity measure between the embeddings of sections, since the library has already been used to create the embeddings.

Doc2vec

Doc2vec works differently than the other models, which proceeds by computing a cost-matrix. In fact, doc2vec must be trained using as input a document and its class. It goes without saying that finding a pretrained French model with one category per section of every document does not exists. Such a pretraining could simply not be created since we can not know in advance the categories of the sections of a document. This is the same problem as mentioned for the segmentation task earlier, that prevents the use of topic based segmentation techniques

Therefore, one proceeds differently. Each time one need to compare 2 versions of a document, a doc2vec model is trained (during 250 epochs) on the sections of the first version of the document, all of which have been assigned a different number as category. Then, using this freshly trained model, one create the embeddings of the sections of the second version of the document. Finally, instead of computing the similarity between the embeddings of the 2 versions, one just check to which category (which are the indexes associated to a section of the first version of the document) the embedding of a section of the second version of the document is the closest to.

To perform all these operations, the Gensim library is again used, as it implements a Doc2Vec class.

BERT and RoBERTa

The French implementations of BERT and RoBERTa (namely FlauBERT and CamemBERT) are available in the Huggingface library [Wol+20]. As explained in section 2.4, these models are not only neural network architectures. They are the complete, pre-trained model. Therefore one import and use these complete pre-trained models.

To create the similarity matrices for these models, one give the tokens of a complete section as input to the model, which outputs, as explained in section 2.4, one vector per token in the section. The first token always correspond to the [CLS] token, which is used to indicate the beginning of a sequence. It is also well-known to represent the complete input sequence. Therefore, this output vector is used as section embedding, and the cosine similarity is computed between each pair of sections to create the similarity matrix.

3.4 Sections comparison

As explained in section 2.4, the most used algorithm to compare chunks of text is Myer's Diff algorithm.

For this tool, one uses Google's implementation of it: the open-source Diff-Match-Patch library (<https://opensource.google/projects/diff-match-patch>). This version of the implementation contains speedups and cleanups, improving both the performance and the output quality of the algorithm. This makes it a clear choice for this module. Note that the version used has been modified by Samy Doloris to make the results more human readable.

This module outputs a html file containing two columns (one for each document version), and where each section of each pair of matched sections are put side to side. Deleted words are highlighted in red and crossed out, while new words are highlighted in green and underlined. Sections that were not matched are placed alone on a line, at the end of the document. An example of such an html file generated is shown in figure A.3 in the appendix.

Chapter 4

Results and observations

This section will address the experiments established, the dataset created, the metrics designed and the results obtained with the tool, based on the different NLP methods discussed and implemented.

4.1 Dataset

The main obstacle to measure the performances of the different models on the matching task is that there does not exist any dataset to do it.

To address this problem, a small dataset of pairs of insurance documents, along with the matching of their sections has been built. Since the matchings must be manually found and encoded for each pair of documents, it takes *a lot* of time to create the *ground-truth* of pairs of sections for each pair of versions of documents. It is for this reason that the dataset is quite small, but significant enough to be able to compare the different NLP methods implemented.

The dataset is made up of 15 pairs of documents of the insurance company "AXA", concerning car or home insurances. These 15 pairs of documents contains 1059 sections, forming 515 pairs of matching sections, and 29 individual sections (these sections have been added or removed from one version of a document to the other).

4.2 Metrics

In addition to the absence of dataset, there is no real metric designed to assess the performances of a model for a matching task. Therefore, besides the creation of the dataset, different metrics have been imagined to be able to compare the different methods implemented. These are inspired of the confusion matrix often use to described the performances of a classification model. However, the regular confusion matrix can not be used as the matching task is a little bit more elaborated than the classification task. In the case at hand, there are no predefined categories (sections) that are common to every document, and therefore one can not count the number of sections that have been correctly classified.

Instead, The modified confusion matrix imagined emphasizes on the correct and incorrect

matches, which does not require to have different categories. This matrix is composed of the following values:

- TP = True Positives: corresponds to the number of correctly matched sections.
- FP = False Positives: corresponds to the number of instances where sections matched incorrectly (in other words, when two sections matched when they did not have to).
- TN = True Negatives: corresponds to the number of sections that did not match any section, and that should indeed not match any (because these sections have been added or removed from one section to another).
- FN = False Negatives: corresponds to the number of sections that did not match to another section, while it should have matched a section.

The objective will be to maximize the number of Trues while minimizing the number of Falses. They are related but it is not always possible to achieve both objectives (the number of False Negatives may increase when maximizing the number of True Negatives).

A *correctness* metric is also introduced, and is defined as follows:

$$correctness = \frac{TP + TN}{total}$$

where *total* is the number of matchings and non-matched sections of the groundtruth. In fact, it corresponds the proportion of correctly classified sections. It is similar to the accuracy metric used for the classification task.

The modified confusion matrix will allow to have a better understanding of the weaknesses and advantages of the different models, than simply using the *correctness* (the proportion of correctly classified sections). For instance, this will allow to verify if a model tends to match too frequently sections that have no correspondence in the other version of the document, and adjust the threshold value accordingly.

4.3 Experiments for the matching task

4.3.1 Comparison of different ways to use Word2vec

As explained in section 3.3, different approaches of word2vec have been implemented. The results obtained using these different ways of training word2vec on the dataset are summarized in Table 4.1.

As one can observe, using a pre-trained model has only advantages compared to the other models. It has the best modified confusion matrix values, and is the fastest to use.

This observation raises the question of the usefulness of making a custom training of word2vec for the matching task. It is necessary to remind that there is no large corpus of insurance documents available to train rigorously the model. Maybe if such a corpus existed, the trained model would outperform any other method. The only way to be really sure would be to create such a corpus.

For the comparisons with the other NLP methods, the pre-trained model has been used, as it is the one that performs the best among the 3 versions implemented.

Training method	TP	TN	FP	FN	Training time	Matching time
Sections	511	7	25	4	30 s	224.12 s
Légifrance	512	7	24	3	8 h	198.44 s
Pre-trained	514	15	8	2	None	130.19 s

Table 4.1: Comparison of the different methods implemented to train Word2vec.

Sections corresponds to training the model at run time, on the sections that need to be compared and matched.

Légifrance corresponds to the model trained on text data collected from *Légifrance*, France’s official website for publishing legislation, regulations and legal information.

Pre-trained consists in using a pre-trained model, trained on 1.6 billions words.

Method	TF	TF-IDF	Word2vec	Doc2vec	BERT	RoBERTa	Mean
Titles	0.954	0.954	0.946	0.639	0.870	0.881	0.874
Sections	0.958	0.964	0.964	0.928	0.923	0.954	0.949

Table 4.2: Mean *correctness* achieved using the different methods implemented, by matching based on the titles or based on the sections.

4.3.2 Matching without thresholding

The different methods have first been tested on the dataset created without thresholding in 2 different ways:

- The first consists in determining the matching based on the titles of the sections only.
- The second consists in identifying the matchings based on the contents of the sections only.

The values of the different metrics imagined earlier in section 4.2 have been plotted in Figures 4.1 and 4.2 for these two approaches, to be able to compare their performances. The precise values of these results can be found in Table A.1 in the appendix.

The first observation one can make is that, overall, basing the matchings on the contents of the sections rather than on the titles of the sections leads to better results. This is consistent with the reasoning done in section 3.2: titles may change from one version of a document to another, which implies that matching using the titles in these cases is very hard, if not impossible if not using contextual or word embeddings. In this case, the best mean *correctness* achieved by matching the titles is equal to 0.954, while it is equal to 0.964 by matching the titles, as can be seen in Table 4.2.

The second observation is that Doc2vec, although promising from a theoretical point of view, is not suitable for the task at hand. Logically, it performs very poorly if using only the titles (the titles are too short to be able to train correctly the model). In addition, it is the method producing the most incorrect matchings, as shown by the high number of False Positives and False Negatives, as well as its low *correctness*.

Surprisingly, one can also notice that the most efficient methods for this task are not the most recent ones. In particular, TF-IDF and word2vec give the best results, and work equally well. On the other hand, BERT and RoBERTa, which are the most complex

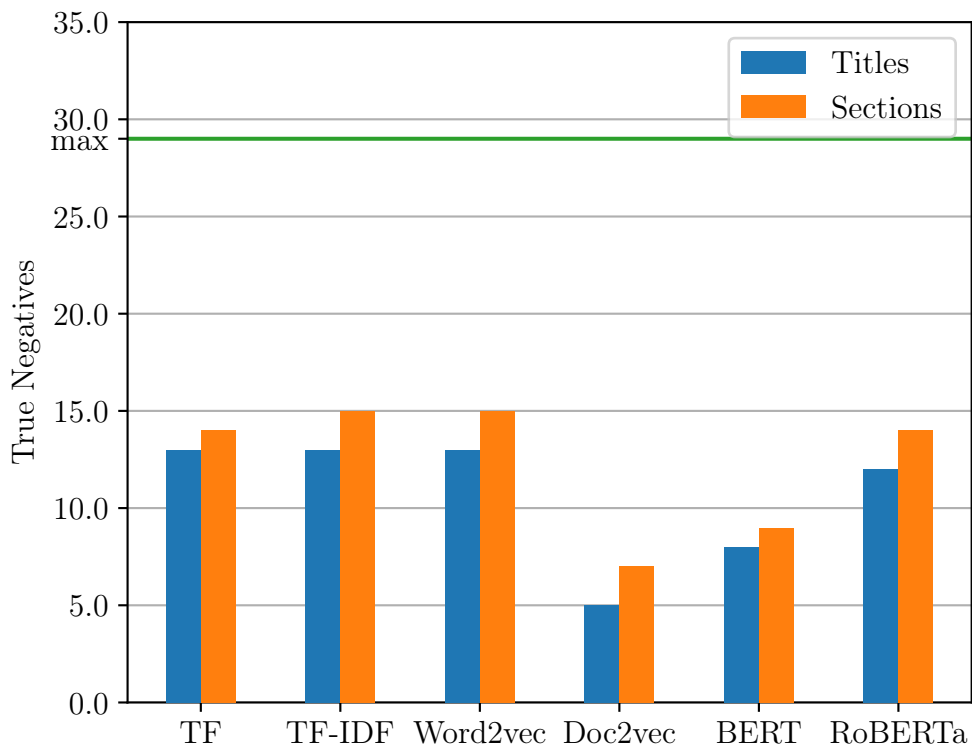
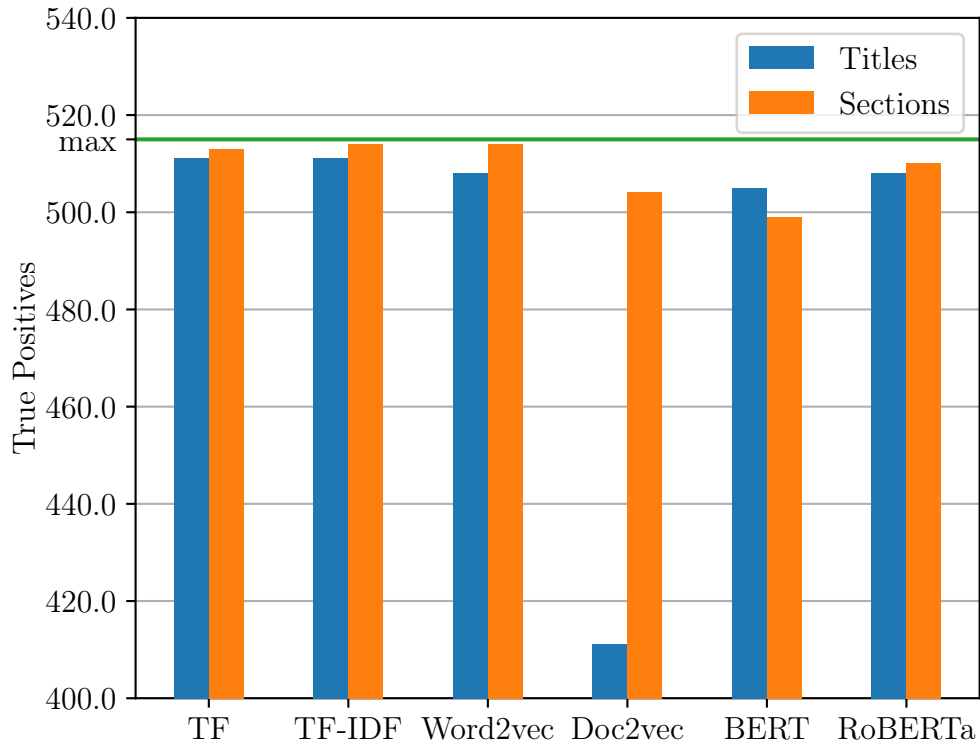


Figure 4.1: Comparison of the number of True Positives and True Negatives obtained by matching based on the titles or based on the sections' content, using the different methods implemented.

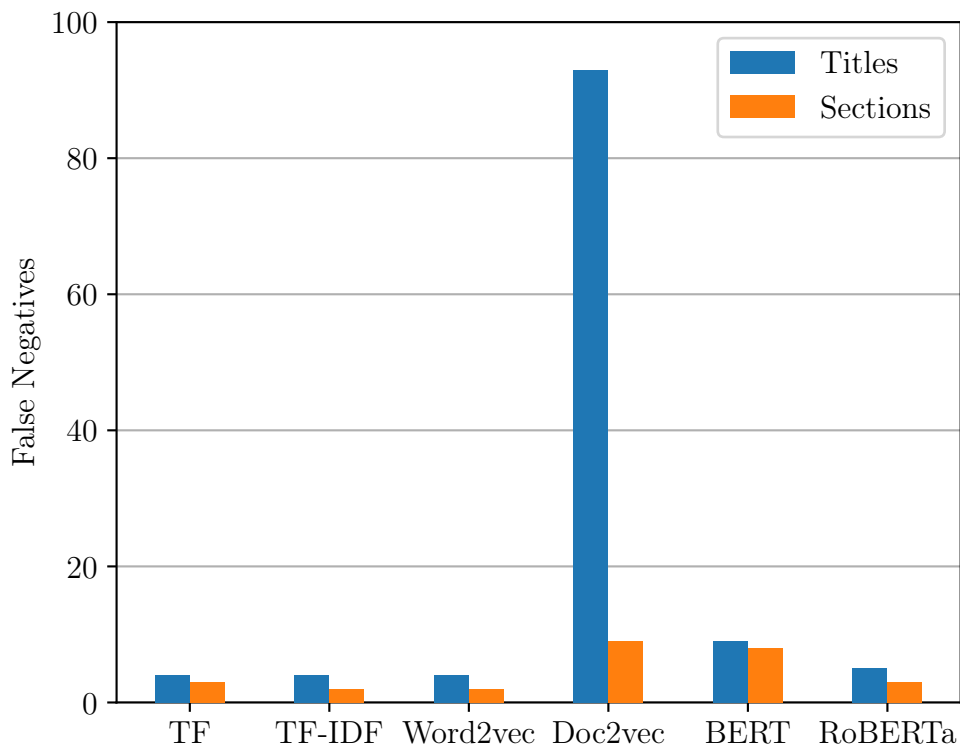
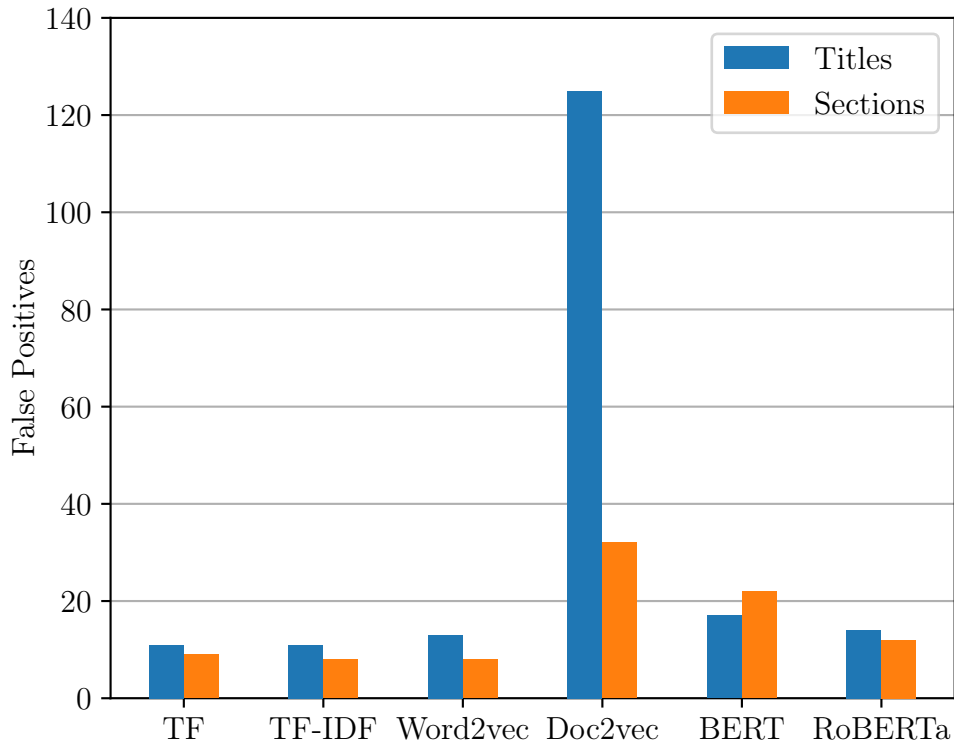


Figure 4.2: Comparison of the number of False Positives and False Negatives obtained by matching based on the titles or based on the sections' content, using the different methods implemented.

models have fewer True Positives and more False Positives, meaning that they produce in fact results that are less discriminant.

One reason that could explain the poor performance of the BERT model is that the vector used to compare the sections is used during the training stage to learn to predict if a sentence could follow another one, as explained in section 2.3. Therefore, vectors of text inputs that are related must be similar, even if they are not identical or nearly so. Although using this vector for a regular document classification task would work well, the sections to classify in this case are all related since they refer to insurance related stuff, which much increase the similarity between each inferred embedding.

This reasoning could also explain why RoBERTa, for which the Next Sequence task has been removed during its training stage (as explained in Section 2.3), still performs better than BERT. This task being removed, the vector associated to this token should be exclusively dependent of the text input at hand. Nevertheless, this model does not compete with TF-IDF or Word2vec. A reason for that could be that the usage of contextual word embeddings increase the mean similarity between every pair of section. The section embedding used represents a good summary of the text sequence, but once again since the context plays a major role in BERT and RoBERTa, these embeddings must be closer than when comparing word embeddings or frequency vectors.

In order to further investigate why simple methods seem to outperform complex models, the heat maps of the cost (or similarity) matrices of each method have been plotted (except for Doc2vec which does not compute such a matrix). One can see in Figures 4.3 and 4.4 that, as assumed in the previous paragraph, the similarity between each pair of sections are less pronounced using the contextual word embeddings (of RoBERTa in this case), whereas one can clearly identify the most similar sections from the heat map of the cost matrix obtained using frequencies vectors (of TF-IDF in this case). Moreover, the values of the cost matrix of TF-IDF are much more distributed, ranging between 0 and 1, where as the values of the similarity matrix of RoBERTa are between a much smaller range, being comprised between 0.75 and 1. This supports the idea that complex models are in fact *too good* for the matching task, and that pure statistical methods are more suited. The heat maps for the other models can be found in Figures A.8, A.9 and A.10 in the appendix.

4.3.3 Matching with thresholding

Without thresholding, one can observe that there are very few True Negatives and $\sim 2\%$ of False Positives. These pairs of sections will be problematic for the last module of the tool. Unrelated sections will be compared, and most of the time only few words will be in common in the two sections. The comparison would therefore produce something that is unsightly, and not useful at all.

One can reduce the number of False Positives (and increase the number of True Negatives at the same time since they are related) by setting the threshold discussed in Section 3.3. False positives are most of the time pairs of sections that have a lower similarity than the others.

For each method, different threshold values have been tried. The most interesting cases are discussed below. The results for the other cases are shown in Figures A.4, A.5, A.6

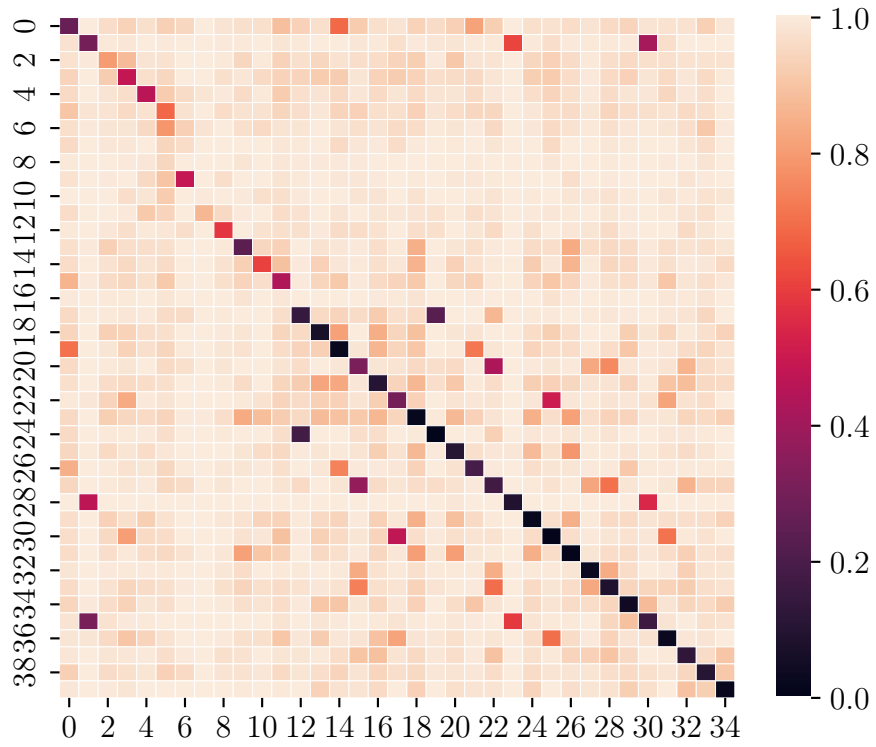


Figure 4.3: Heat map of the cost matrix obtained using **TF-IDF**.

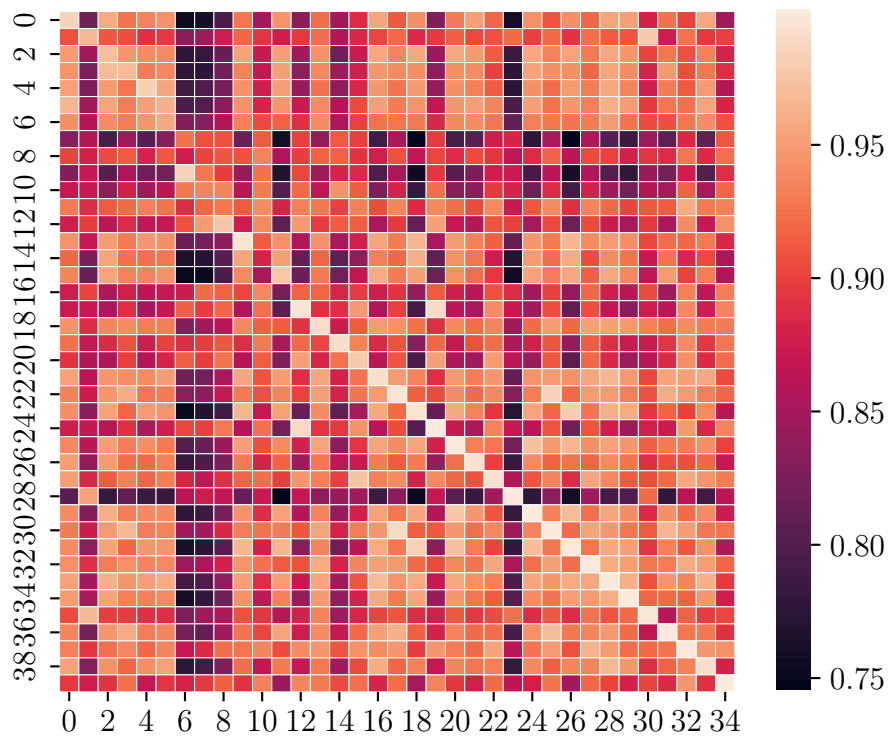


Figure 4.4: Heat map of the similarity matrix obtained using **RoBERTa**.

Threshold	TP	TN	FN	FP
0.9	514	19	10	2
0.87	514	24	5	2
0.85	513	24	5	4
Exclusive matching	514	29	0	4

Table 4.3: Results for the non-exclusive matching approach using TF-IDF with different threshold values, compared with the results for the exclusive matching method with TF-IDF and a threshold of 0.9.

and A.7 in the appendix. The precise values obtained for the different models with the best threshold¹ are listed in Table A.1 of the appendix.

Concerning TF-IDF, Figure 4.5 shows that it is possible to have no False Positives while still having the maximum number of True Positives achievable, as well as the correct number of True Negatives. This however comes at a cost of a little increase in the number of False Negatives. Nevertheless, using a threshold value of around 0.9 leads to a *correctness* equal to 0.999, the highest value reached regardless of the method used. TF-IDF is the only method for which it is possible to maximize the number of True Positives while not having any False Positive. For every other method, the number of True Positives first decreases before that the number of False Negatives cancels out, as can be seen for Word2vec in Figure 4.6, which is the second best performing method using the threshold.

4.3.4 Non-exclusive matching

As explained in section 3.3, 2 methods of matching have been implemented: an exclusive matching approach, that matches every section to maximum one other section, and a non-exclusive matching method, that matches a section to its most similar one, regardless of whether this other section has already been matched or not.

The previous experiments have all been done using the exclusive matching approach, which is considered as the default one. As a reminder, the non-exclusive method has been implemented with the idea that sections may have been split or merged. This second matching approach has been tried, using TF-IDF (the model giving the best results during the previous experiments), with 3 different threshold values. The results are listed in Table 4.3, and compared with the best performing model of the exclusive matching approach.

As one can observe, the non-exclusive matching approach can not match the performances of the exclusive matching one, as one can not obtain the maximum number of TP while having no FN at all.

Unfortunately, the dataset being small, there are very few occurrences of sections being merged or split. Among the 15 documents, only 2 cases of this event were observed. Therefore, assessing the performances of the attempts to match a section to 2 sections

¹The best threshold here is defined as the threshold for which the number of TP is the highest while minimizing the number of FP.

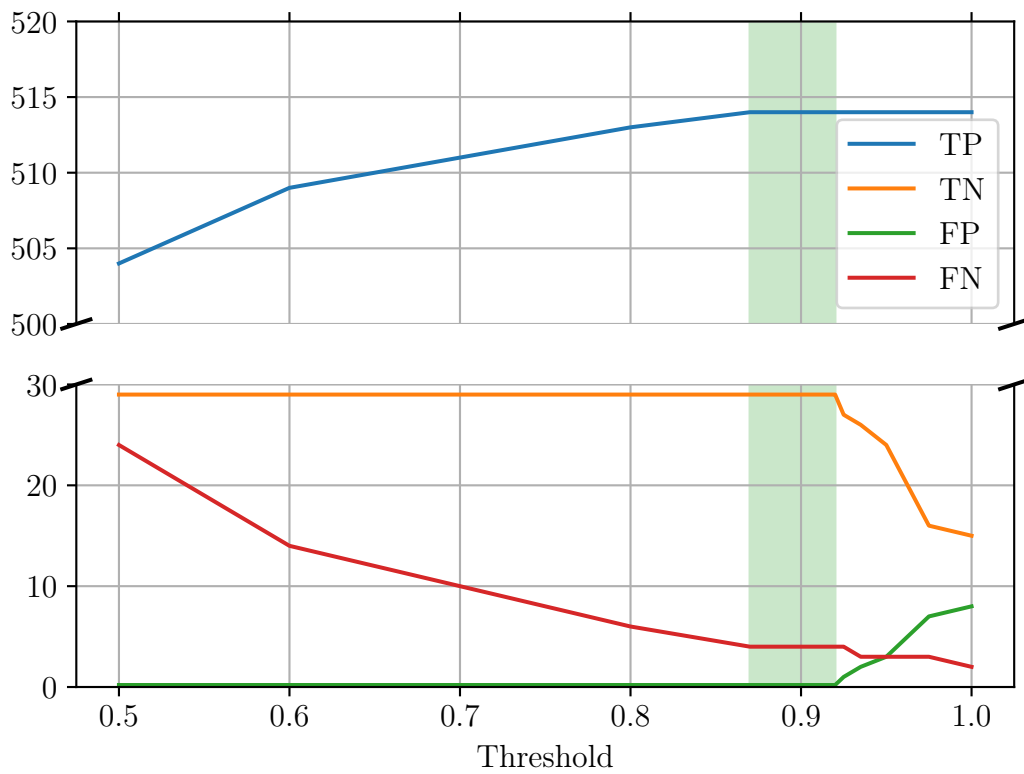


Figure 4.5: Results obtained with **TF-IDF** by varying the threshold value.

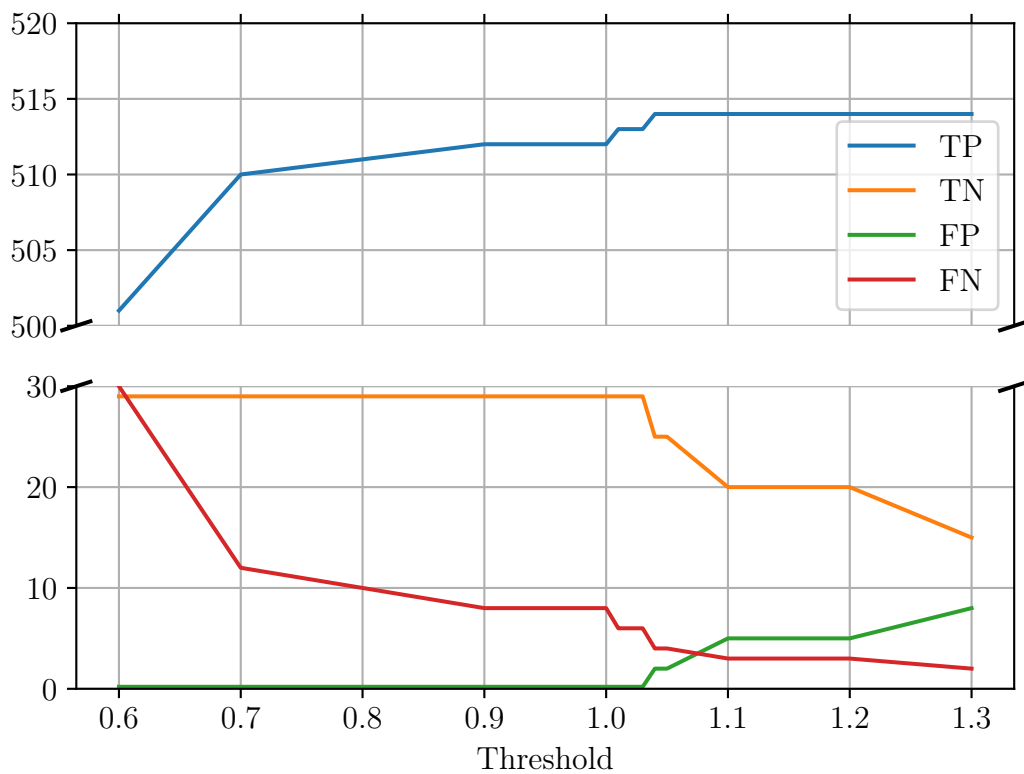


Figure 4.6: Results obtained with **Word2vec** by varying the threshold value.

Method	Tf	Tf-idf	Word2vec	Doc2vec	BERT	RoBERTa
Duration (s)	54.23	1.67	130.19	3.46	77.54	24.73

Table 4.4: Mean processing time needed for each method to compute the similarities between the pairs of sections and match the most similar ones.

if one was split or if two were merged can not be done rigorously. Anyway, using TF-IDF with the non-exclusive matching approach and a threshold of 0.87 (the value that minimizes the number of FN while keeping the number of FP maximum) was not able to do these matchings in the 2 occurrences observed.

The non-exclusive approach is therefore ineffective and does not deserve to be investigated further. Instead, a completely different approach could be considered.

4.3.5 Speed comparison

Since the different methods are intended to be used inside a tool, the comparison of their processing speed may also be interesting, as nobody likes to use something that is time-consuming. Therefore, each method has been applied on the 15 pairs of documents, and the mean processing time has been computed. This process has been done 5 times, and again the mean has been computed in order to have a reliable value. These mean processing speeds are listed in Table 4.4.

One can observe that TF-IDF is not only the most efficient, but it is also the fastest method to find the matching sections. As one would expect, more complex methods generally take longer, because they need to transform the words into their corresponding tokens and embeddings first.

RoBERTa being faster than BERT can be explained easily. The version of BERT used (BERT-large) has 2 times more encoding layers than the version of RoBERTa used (RoBERTa-base).

Word2vec is the slowest, as it need to compare the euclidean distance between each word in addition to the distances between each sections.

Doc2vec is quite fast, which can be explained by the fact that the model is trained on a very small corpus, and need to produce only one embedding per section.

Finally, TF takes a lot of time compared to TF-IDF. This must be because the vectors are created on the basis of a pair of sections, whereas the vectors for TF-IDF are based on a single section, allowing to reuse the vectors for any pair of section.

Based on these observations, TF-IDF seems to be the best candidate once again

4.3.6 Combining titles and sections

During the implementation of the different methods, the possibility to match the sections by first matching the titles then by matching the remaining sections was implemented. As explained in section 3.3, the intuition behind this approach was to use a simple method such as TF or TF-IDF with a high threshold to match the titles to identify the most obvious matches, then use a more complex method such as Word2vec or BERT to match

the remaining sections based on their content, with the aim of alleviating the work of this more complex model.

However, it was not expected that the best performing method was a simple model, which moreover is the fastest one. Therefore this approach has been abandoned.

4.4 Text segmentation issues

When first evaluating the performances of the NLP methods on the dataset, one pair of documents in particular was more problematic than every other one. After analysis, it turns out that the problem originates from the segmentation of the text into sections. In fact, the documents are really small, and unfortunately they contain a numbered list, that has more items than there are sections in these documents. Since the segmentation module looks for the longest match of numbers (see section 3.2), it considers the items of the numbered list as sections, rather than the actual sections.

This indicates that the segmentation module could still be enhanced, or that another approach than using regular expressions could be explored. In order to not distort the results of the evaluations of the NLP methods, this pair of document has been removed from the dataset before doing the different experiments.

A small piece of code has also been written to check if the segmentation module worked correctly overall. One can observe using Table A.1 (a) and (c) that there are 2 sections too many (if there are 515 TP, 29 TN and 0 FP, there can only be 2 FN, unless there are 2 sections too many). It turns out that there were in fact 4 incorrect sections too many, and 2 sections were missing. This was due to unfortunate facts such as, for instance, that "section 2.3". contains a reference to the "Article 2.4.", which is matched by the regular expression, and validated by the longest matching algorithm. The 2 sections missing are not matched because they have no title, and because they end with a ":". Therefore, they are considered by the segmentation module as being only composed of a title and are discarded.

Therefore, if these sections are not taken into account, TF-IDF used with a threshold between 0.87 and 0.92 actually makes zero mistakes over the dataset.

4.5 Qualitative analysis of the output

A few observations and comments are made by observing the html files output by the tool developed. Note that the analysis of the overall quality of the developed tool can only be done in a qualitative manner.

First, as shown in Figure A.3, decomposing the text into its sections before doing the syntactic comparison leads to far better results compared to doing the comparison on the complete text input. The few differences are immediately apparent, nicely displayed, and it is easy to spot the correspondence of the changes displayed on the left and on the right.

However, the output is not always as good. For example, when many modifications have been made to a section, some parts of the text are incorrectly marked as modified, as one can see in Figure 4.7. Likewise, when the section contains a table, the addition of

elements to this table can change the arrangement of the text inside the table, and change the extraction order of these text chunks. This case is illustrated in Figure 4.8.

Therefore, other extraction or comparison strategies could be explored. For instance, one could try to further break down a section into smaller chunks when the modifications are numerous, match these chunks (for example one could consider sentence segmentation) using any method discussed earlier, and compare the smaller chunks that matched.

[Doc 1]: Section 1.3. - Quelles sont les garanties obligatoires liées à la garantie Responsabilité ?

Nous couvrons la responsabilité des assurés qui serait engagée à l'occasion de l'usage dans la circulation du véhicule assuré. Nous indemnisons, conformément à la loi, les conséquences des dommages corporels résultant pour un usager faible d'un accident de circulation dans lequel le véhicule assuré est impliqué. Nous avançons aussi le cautionnement exigé, en vue de la protection des personnes lésées pour lever la saisie du véhicule désigné ou pour la mise en liberté sous caution de l'assuré. Notre garantie est Pour les dommages résultant de lésions corporelles : illimitée. Toutefois, si au jour du sinistre, la réglementation nous autorise à limiter notre garantie pour ces dommages, celles-ci sera limitée, par sinistre à 120.067.670 EUR ou, s'il lui est supérieur, au montant le plus bas auquel la réglementation autorise la limitation de garanties. Pour les dommages matériels (autres que ceux visés au point ci-après) : limitée à 120.067.670 EUR par sinistre ou, s'il lui est supérieur, au montant le plus bas auquel la réglementation au jour du sinistre autorise la limitation de garanties. Pour les dommages occasionnés aux vêtements et bagages personnels des passagers du véhicule assuré : limitée à 2.977 EUR par passager ou, s'il lui est supérieur, au montant le plus bas auquel la réglementation au jour du sinistre autorise la limitation de garanties. Pour le cautionnement : limitée à 62.000 EUR pour le véhicule désigné et l'ensemble des assurés.

[Doc 2]: Section 1.3. - Que couvre la garantie Responsabilité ?

Nous couvrons la responsabilité civile encourue par les assurés à la suite d'un sinistre causé par le véhicule assuré à l'occasion de son usage dans la circulation. Cette couverture est conforme à la loi du 21 novembre 1989 ou, le cas échéant, à la législation étrangère applicable. Nous indemnisons en outre certaines victimes d'accidents de la route, à savoir : Les usagers faibles, conformément à l'article 29 bis de la loi du 21 novembre 1989 Celui-ci prévoit la réparation des dommages qui sont subis par les usagers faibles et leurs ayants droit et qui résultent de lésions corporelles ou du décès, y compris les dommages aux vêtements et les dommages occasionnés aux prothèses fonctionnelles, en cas d'accident de la circulation impliquant le véhicule assuré et sur lequel le droit belge est d'application, à l'exclusion des accidents survenus dans un pays qui n'est pas mentionné sur le certificat d'assurance. Les victimes innocentes, conformément à l'article 29 ter de la loi du 21 novembre 1989 Celui-ci prévoit l'indemnisation de tous les dommages subis par les victimes innocentes et leurs ayants droit, c'est-à-dire les personnes sur lesquelles ne pèse manifestement aucune responsabilité, lorsque le véhicule assuré est impliqué avec un ou plusieurs autres véhicules dans un accident de circulation en Belgique et qu'il n'est pas possible de déterminer quel véhicule a causé l'accident. Nous avançons aussi le cautionnement exigé par une autorité étrangère, en vue de la protection des personnes lésées, pour lever la saisie du véhicule désigné ou pour la mise en liberté sous caution de l'assuré suite à un sinistre dans un pays repris sur le certificat d'assurance (autre que la Belgique). Notre garantie est : Pour les dommages résultant de lésions corporelles: illimitée. Pour les dommages matériels (y compris les dommages occasionnés aux vêtements et bagages personnels des passagers du véhicule assuré): limitée à 120.067.670 EUR par sinistre Pour le cautionnement: limitée à 62.000 EUR pour le véhicule désigné et l'ensemble des assurés, majorée des frais de constitution et de récupération du cautionnement qui sont à notre charge

Figure 4.7: Comparison based on a section which has been heavily modified. As shown by the sentences boxed in blue, the module is less successful at determining the real modifications, sometimes marking identical sentences as modified.

[Doc 1]: Section 1.2. - Quelle est l'étendue territoriale de la garantie Responsabilité ?

Andorre France Liechtenstein Autriche Tchèque Belgique ARYM (Macédoine)
Lituanie Pologne Tunisie Bosnie Herzégovine Grèce Luxembourg Portugal Turquie
Bulgarie Hongrie Malte Roumanie Vatican Chypre(*) Irlande Maroc Saint-Marin
Royaume-Uni Danemark Islande Monaco Serbie(*) Suède Allemagne Italie
Monténégro Slovénie Suisse Estonie Croatie Pays-Bas Slovaquie Finlande Lettonie
Norvège Espagne (*) Nous n'offrons une couverture que dans les parties
géographiques de Chypre et de la Serbie qui sont sous le contrôle des
gouvernements respectifs.

[Doc 2]: Section 1.2. - Quelle est l'étendue territoriale de la garantie Responsabilité ?

Sauf mention contraire reprise sur votre certificat d'assurance, notre garantie est
accordée pour un sinistre survenu dans les pays suivants : Allemagne Andorre
Autriche Belgique Bosnie- Herzégovine Bulgarie Chypre (*), Cité du Vatican Croatie
Danemark Espagne Estonie Finlande France Grèce Hongrie Irlande Islande Italie
Lettonie Liechtenstein Lituanie Luxembourg Macédoine du Nord Malte Maroc
Monaco Monténégro Norvège Pays-Bas Pologne Portugal Roumanie Royaume-Uni
Saint-Marin Serbie (*), Slovénie Slovaquie Suède Suisse Tchèque Tunisie Turquie
(*) Nous n'offrons une couverture que dans les parties géographiques de Chypre
et de la Serbie qui sont sous le contrôle des gouvernements respectifs. Notre
garantie Responsabilité est accordée pour les sinistres survenus sur la voie
publique ou les terrains publics ou privés

Figure 4.8: Comparison based on a section from which the text has been extracted from tables. The extraction order is not the same, causing some words to be incorrectly considered as removed or added, as the ones boxed in blue.

Chapter 5

Conclusion

In order to compare 2 versions of a document, the segmentation of these documents into their sections and subsections and then make the comparisons between each section allows to improve the quality of the comparison.

To determine the best way to match the corresponding sections, different NLP methods have been compared. A small dataset of 15 pairs of documents with their correct matchings, as well as different metrics inspired from the confusion matrix often used in classification tasks have been designed, to be able to compare the performances of the different methods. The outcome is that the most complex methods are actually too good in language modeling for the task at hand. Contextual embeddings (BERT, RoBERTa) perform very well on regular text classification tasks, because they are very good at comparing the general subject, or context, of a text sequence independently of the words used. However, in the case at hand, most of the sections have contents that are similar, and the embeddings produced are therefore less discriminant compared to embeddings or vectors of simple methods. Paragraph embeddings are also not suitable for the matching task, because they need to be trained on classes of documents. However, in the case at hand it is not possible to create meaningful classes for each section, that can be shared among every document. Word embeddings with Word Mover's distance in contrast perform very well, using a pretrained model. Training a model on a huge corpus of insurance documents could possibly improve the performances. Nevertheless, excellent results can be obtained using simple statistical methods, such as TF-IDF, which works much faster than the aforementioned complex methods, and that does not require a specific training for a new type of document. These results and observations strongly encourage to use TF-IDF for the matching task.

Since TF-IDF, which produces frequencies vectors using a statistical approach, performs better than more complex methods, further work may be done by investigating the performances of word embeddings based on statistical approaches, such as Latent Semantic Analysis, on the matching task.

Basing the text segmentation module on regular expressions and on the idea of matching the longest sequence of characters is not the best solution, as there are in some rare cases undesired behaviours. These do not appear frequently at all, but nonetheless some other approaches may be investigated for this module as well, such as OCR and computer vision methods that would recognize the titles during the text extraction phase.

Concerning the highlighting of the changes made to a section, Myer's Diff algorithm does the trick most of the time. It sometimes struggles when a section has been too heavily modified, or when a section contains a table that has been modified. Further work could try to handle these cases. For instance, when noticing that a section has been modified a lot, one could further divide this section into smaller pieces of text, such as sentences, and use the matching task to find the correspondences between the two sections.

Some other improvements could focus on the threshold value used during the matching task to reject matchings that are not significant enough. The threshold values used for the comparisons were the same for every pair of sections of the dataset. One could imagine a procedure that would determine automatically a threshold value, by computing it using a specific function and the similarity values obtained for a pair of documents.

Bibliography

- [Joa+16] Luzak Joasia, Loos Marco, Elsen Millie, Leenheer Jorna, and Elshout Maartje. “Study on consumers’ attitudes towards Terms and Conditions (T&Cs)”. In: 2016. ISBN: 978-92-9200-706-5 (page 1).
- [BM03] David Blei and Pedro Moreno. “Topic Segmentation with an Aspect Hidden Markov Model”. In: (Sept. 2003). DOI: [10.1145/383952.384021](https://doi.org/10.1145/383952.384021) (page 5).
- [Lam+07] Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, and Frédéric Saubion. “ClassStruggle: a clustering based text segmentation”. In: Jan. 2007, pp. 600–604. DOI: [10.1145/1244002.1244140](https://doi.org/10.1145/1244002.1244140) (page 5).
- [Mis+11] Hemant Misra, François Yvon, Olivier Cappé, and Joemon Jose. “Text segmentation: A topic modeling perspective”. In: *Information Processing & Management* 47.4 (2011), pp. 528–544. ISSN: 0306-4573. DOI: <https://doi.org/10.1016/j.ipm.2010.11.008>. URL: <https://www.sciencedirect.com/science/article/pii/S0306457310000981> (page 6).
- [BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. “Latent Dirichlet Allocation”. In: 3.null (Mar. 2003), pp. 993–1022. ISSN: 1532-4435 (page 6).
- [HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. “2 - Getting to Know Your Data”. In: *Data Mining (Third Edition)*. Ed. by Jiawei Han, Micheline Kamber, and Jian Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, 2012, pp. 39–82. ISBN: 978-0-12-381479-1. DOI: <https://doi.org/10.1016/B978-0-12-381479-1.00002-2>. URL: <https://www.sciencedirect.com/science/article/pii/B9780123814791000022> (page 7).
- [SW17] Sahar Sohngir and Dingding Wang. “Improved sqrt-cosine similarity measurement”. In: *Journal of Big Data* 4.1 (July 2017), p. 25. ISSN: 2196-1115. DOI: [10.1186/s40537-017-0083-6](https://doi.org/10.1186/s40537-017-0083-6). URL: <https://doi.org/10.1186/s40537-017-0083-6> (page 7).
- [MS16] Amit Mandelbaum and Adi Shalev. *Word Embeddings and Their Use In Sentence Classification Tasks*. 2016. arXiv: [1610.08229](https://arxiv.org/abs/1610.08229) [cs.LG] (page 10).
- [Mik+13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: [1301.3781](https://arxiv.org/abs/1301.3781) [cs.CL] (pages 10–12, 14).
- [Dee+90] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. “Indexing by latent semantic analysis”. In: *Journal of the American Society for Information Science* 41.6 (1990), pp. 391–407. DOI: [https://doi.org/10.1002/\(SICI\)1097-4571\(199009\)41:6<391::AID-ASI1>3.0.CO;2-9](https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9). eprint: [https://asistdl.onlinelibrary.com/doi/pdf/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9](https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9). URL: <https://asistdl.onlinelibrary.com/doi/pdf/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9>.

[wiley.com/doi/abs/10.1002/9781119900929.ch10](https://doi.org/10.1002/9781119900929.ch10) (page 10).

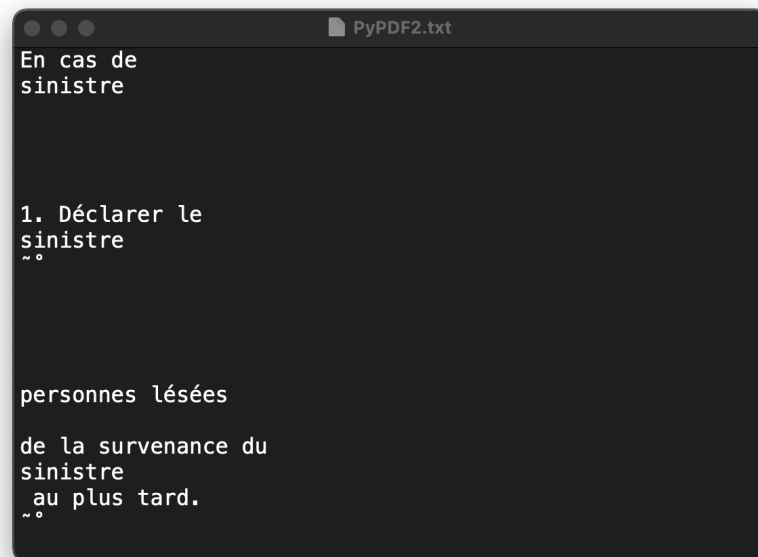
- [CH02] Kenneth Church and Patrick Hanks. “Word Association Norms, Mutual Information, and Lexicography”. In: *Computational Linguistics* 16 (July 2002). DOI: [10.3115/981623.981633](https://doi.org/10.3115/981623.981633) (page 10).
- [NCB17] Marwa Naili, Anja Habacha Chaibi, and Henda Hajjami Ben Ghezala. “Comparative study of word embedding methods in topic segmentation”. In: *Procedia Computer Science* 112 (2017). Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 21st International Conference, KES-20176-8 September 2017, Marseille, France, pp. 340–349. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2017.08.009>. URL: <https://www.sciencedirect.com/science/article/pii/S1877050917313480> (page 11).
- [MYZ13] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. “Linguistic Regularities in Continuous Space Word Representations”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 746–751. URL: <https://aclanthology.org/N13-1090> (page 11).
- [Zhi+13] Alisa Zhila, Wen-tau Yih, Christopher Meek, Geoffrey Zweig, and Tomas Mikolov. “Combining Heterogeneous Models for Measuring Relational Similarity”. In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Atlanta, Georgia: Association for Computational Linguistics, June 2013, pp. 1000–1009. URL: <https://aclanthology.org/N13-1120> (page 11).
- [PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. “GloVe: Global Vectors for Word Representation”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: <http://www.aclweb.org/anthology/D14-1162> (page 11).
- [Boj+16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. “Enriching Word Vectors with Subword Information”. In: *arXiv preprint arXiv:1607.04606* (2016) (page 11).
- [OY19] Korawit Orkphol and Wu Yang. “Word Sense Disambiguation Using Cosine Similarity Collaborates with Word2vec and WordNet”. In: *Future Internet* 11 (May 2019), p. 114. DOI: [10.3390/fi11050114](https://doi.org/10.3390/fi11050114) (page 11).
- [Ron16] Xin Rong. *word2vec Parameter Learning Explained*. 2016. arXiv: [1411.2738](https://arxiv.org/abs/1411.2738) [cs.CL] (page 11).
- [PW08] Ofir Pele and Michael Werman. “A Linear Time Histogram Metric for Improved SIFT Matching”. In: *Computer Vision – ECCV 2008*. Ed. by David Forsyth, Philip Torr, and Andrew Zisserman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 495–508. ISBN: 978-3-540-88690-7 (page 12).
- [PW09] O. Pele and M. Werman. “Fast and robust Earth Mover’s Distances”. In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 460–467. DOI: [10.1109/ICCV.2009.5459199](https://doi.org/10.1109/ICCV.2009.5459199) (page 12).
- [Kus+15] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. “From Word Embeddings To Document Distances”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July

- 2015, pp. 957–966. URL: <http://proceedings.mlr.press/v37/kusnerb15.html> (pages 12–14).
- [RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas Guibas. “The Earth Mover’s Distance as a metric for image retrieval”. In: *International Journal of Computer Vision* 40 (Jan. 2000), pp. 99–121 (page 12).
- [LM14] Quoc V. Le and Tomas Mikolov. “Distributed Representations of Sentences and Documents”. In: *CoRR* abs/1405.4053 (2014). arXiv: [1405.4053](https://arxiv.org/abs/1405.4053). URL: <http://arxiv.org/abs/1405.4053> (page 14).
- [DOL15] Andrew M. Dai, Christopher Olah, and Quoc V. Le. “Document Embedding with Paragraph Vectors”. In: *CoRR* abs/1507.07998 (2015). arXiv: [1507.07998](https://arxiv.org/abs/1507.07998). URL: <http://arxiv.org/abs/1507.07998> (pages 15, 16).
- [Nee+15] Arvind Neelakantan, Jeevan Shankar, Alexandre Passos, and Andrew McCallum. *Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space*. 2015. arXiv: [1504.06654](https://arxiv.org/abs/1504.06654) [cs.CL] (page 16).
- [Wie+16] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. “Charactergram: Embedding Words and Sentences via Character n-grams”. In: *CoRR* abs/1607.02789 (2016). arXiv: [1607.02789](https://arxiv.org/abs/1607.02789). URL: <http://arxiv.org/abs/1607.02789> (page 16).
- [Dev+19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: [1810.04805](https://arxiv.org/abs/1810.04805) [cs.CL] (pages 16, 18, 20).
- [Vas+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: [1706.03762](https://arxiv.org/abs/1706.03762) [cs.CL] (pages 16, 17).
- [BCB16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: [1409.0473](https://arxiv.org/abs/1409.0473) [cs.CL] (page 17).
- [Le+20] Hang Le, Loic Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoıt Crabbe, Laurent Besacier, and Didier Schwab. *FlauBERT: Unsupervised Language Model Pre-training for French*. 2020. arXiv: [1912.05372](https://arxiv.org/abs/1912.05372) [cs.CL] (page 21).
- [LC19] Guillaume Lample and Alexis Conneau. *Cross-lingual Language Model Pre-training*. 2019. arXiv: [1901.07291](https://arxiv.org/abs/1901.07291) [cs.CL] (page 21).
- [Yan+20] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. *XLNet: Generalized Autoregressive Pretraining for Language Understanding*. 2020. arXiv: [1906.08237](https://arxiv.org/abs/1906.08237) [cs.CL] (page 21).
- [Liu+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: [1907.11692](https://arxiv.org/abs/1907.11692) [cs.CL] (page 21).
- [Mar+20] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suarez, Yoann Dupont, Laurent Romary, Eric de la Clergerie, Djame Seddah, and Benoıt Sagot. “CamemBERT: a Tasty French Language Model”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (2020). DOI: [10.18653/v1/2020.acl-main.645](https://doi.org/10.18653/v1/2020.acl-main.645). URL: <http://dx.doi.org/10.18653/v1/2020.acl-main.645> (page 21).
- [Mye86] Eugene W. Myers. “An O(ND) Difference Algorithm and Its Variations”. In: *Algorithmica* 1 (1986), pp. 251–266 (pages 22, 23).

- [Fau15] Jean-Philippe Fauconnier. *French Word Embeddings*. 2015. URL: <http://fauconnier.github.io> (page 31).
- [ŘS10] Radim Řehůřek and Petr Sojka. “Software Framework for Topic Modelling with Large Corpora”. English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. <http://is.muni.cz/publication/884893/en>. Valletta, Malta: ELRA, May 2010, pp. 45–50 (page 31).
- [Wol+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. “Transformers: State-of-the-Art Natural Language Processing”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: <https://www.aclweb.org/anthology/2020.emnlp-demos.6> (page 31).
- [Pet+18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. *Deep contextualized word representations*. 2018. arXiv: [1802.05365](https://arxiv.org/abs/1802.05365) [cs.CL].

Appendix A

Results



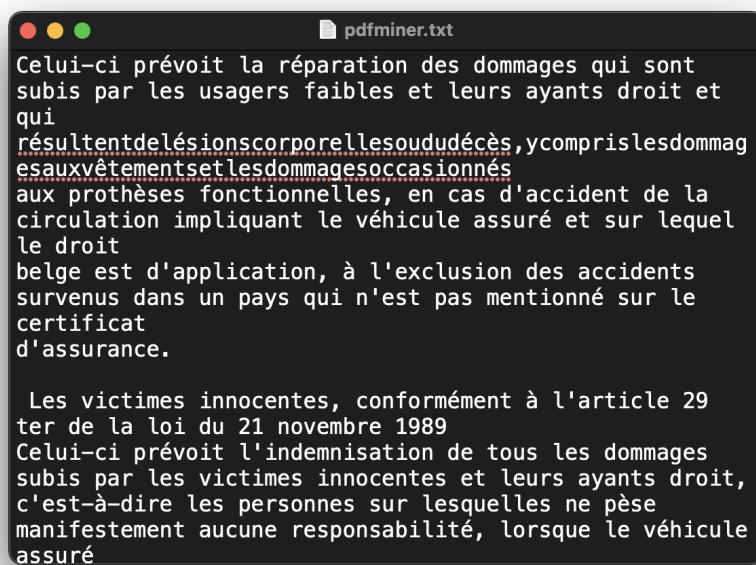
```
En cas de
sinistre

1. Déclarer le
sinistre
~°

personnes lésées

de la survenance du
sinistre
au plus tard.
~°
```

Figure A.1: Text extracted using the package PyPDF. The results are very poor, as it extracts only a few words.



```
pdfminer.txt
Celui-ci prévoit la réparation des dommages qui sont
subis par les usagers faibles et leurs ayants droit et
qui
résultent de lésions corporelles ou d'un décès, y compris les dommages
es aux vêtements et les dommages occasionnés
aux prothèses fonctionnelles, en cas d'accident de la
circulation impliquant le véhicule assuré et sur lequel
le droit
belge est d'application, à l'exclusion des accidents
survenus dans un pays qui n'est pas mentionné sur le
certificat
d'assurance.

Les victimes innocentes, conformément à l'article 29
ter de la loi du 21 novembre 1989
Celui-ci prévoit l'indemnisation de tous les dommages
subis par les victimes innocentes et leurs ayants droit,
c'est-à-dire les personnes sur lesquelles ne pèse
manifestement aucune responsabilité, lorsque le véhicule
assuré
```

Figure A.2: Text extracted using Pdfminersix. The package sometimes misses whitespace characters.

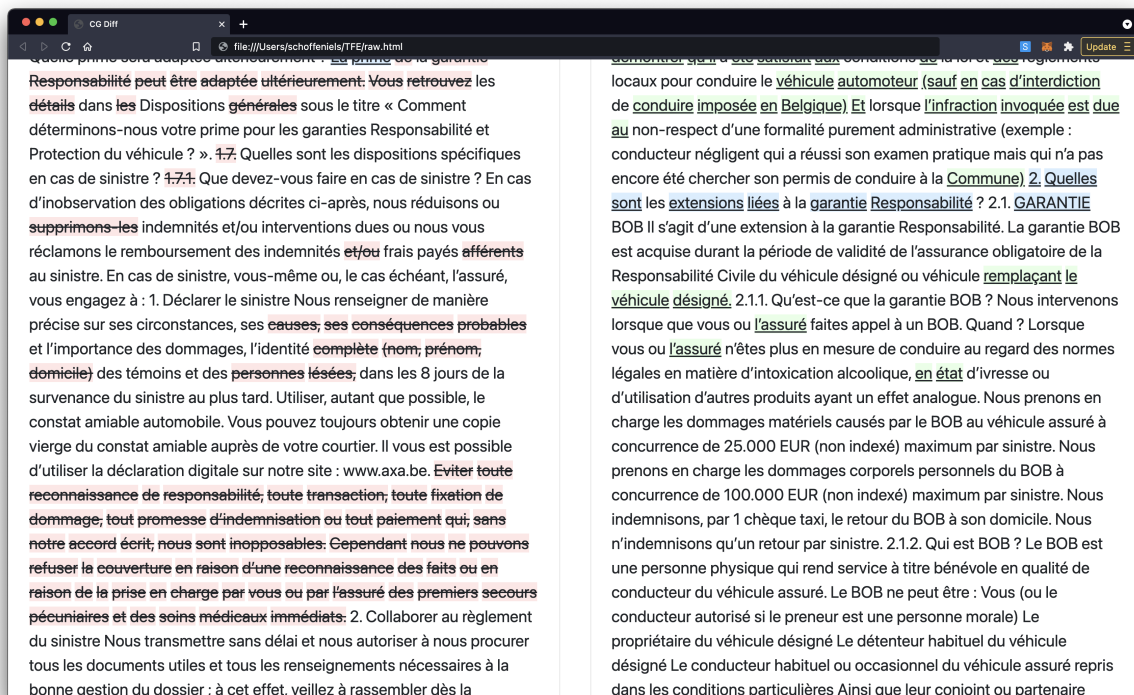
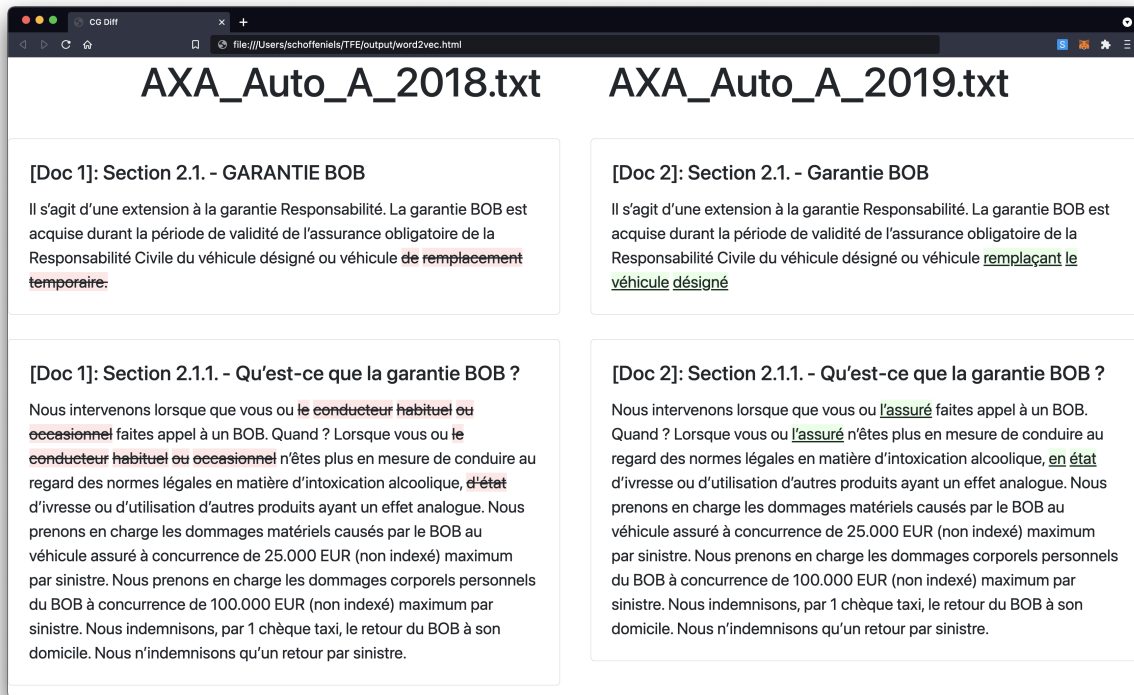


Figure A.3: Comparison of using the comparison module on sections (at the top) and on the complete documents (at the bottom). The output of the tool implemented is cleaner, more readable. Using the comparison module on the complete document makes it hard to find to which modifications from the left side correspond the one from the right side.

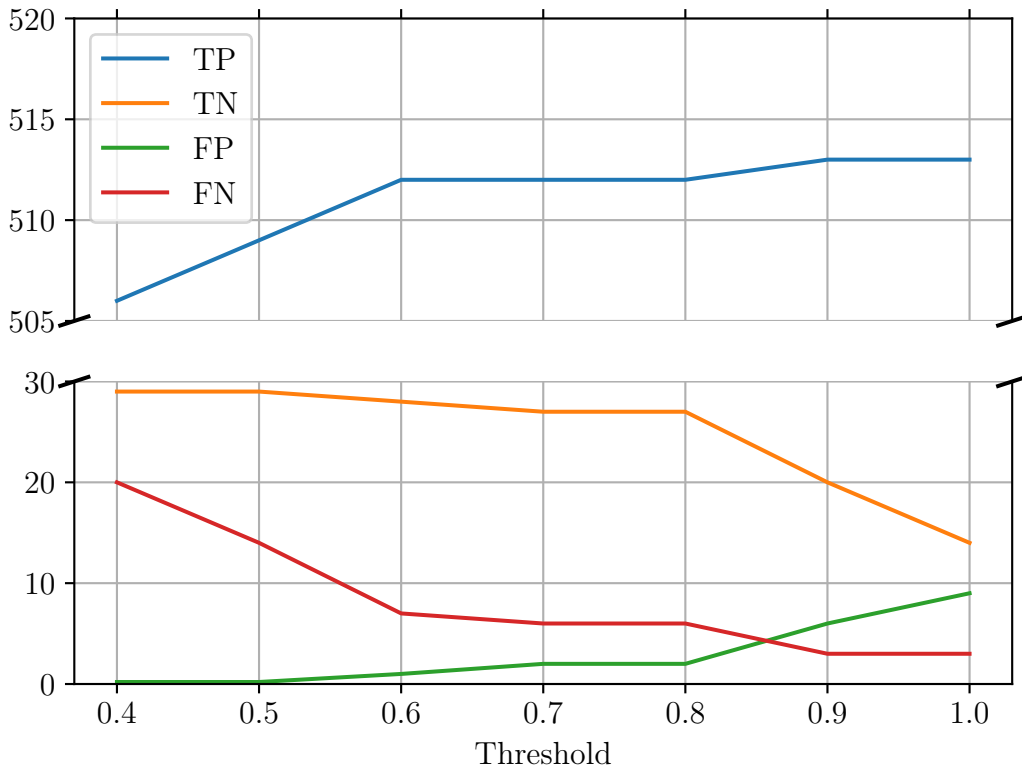


Figure A.4: Results obtained with **TF** by varying the threshold value.

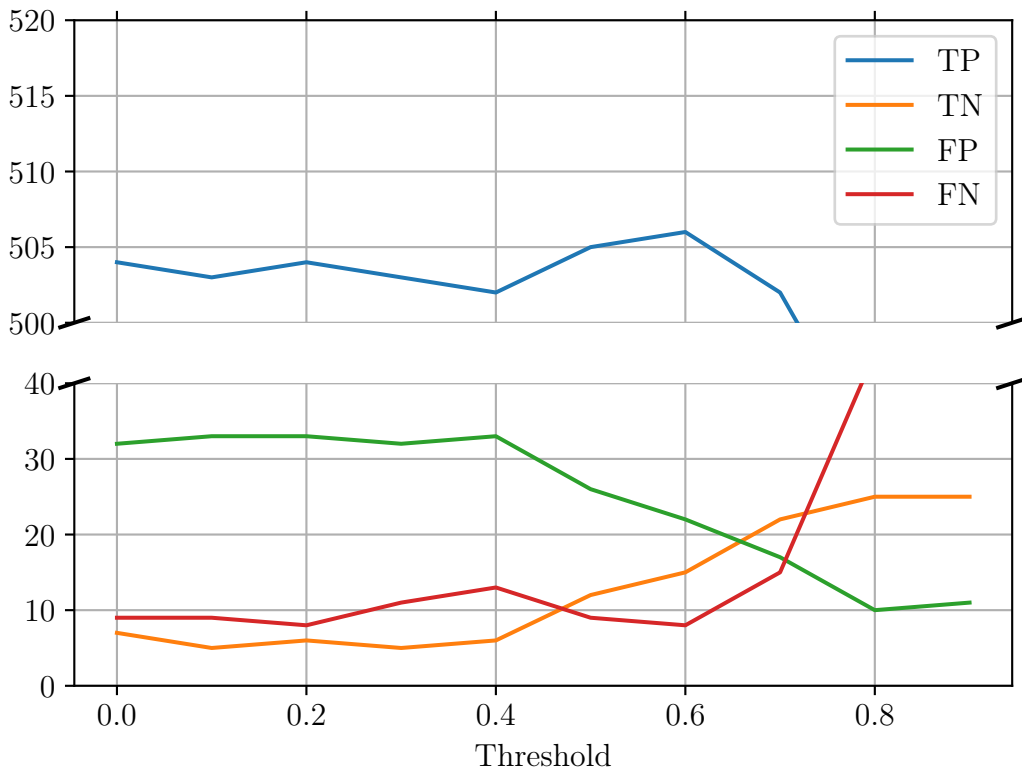


Figure A.5: Results obtained with **Doc2vec** by varying the threshold value.

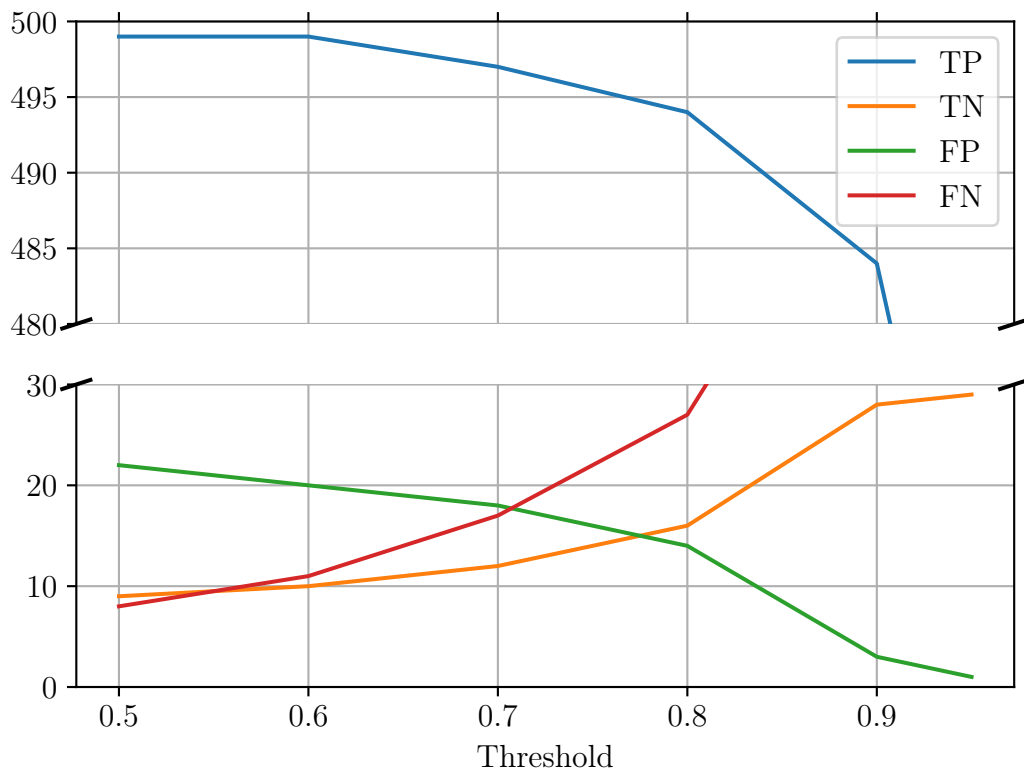


Figure A.6: Results obtained with **BERT** by varying the threshold value.

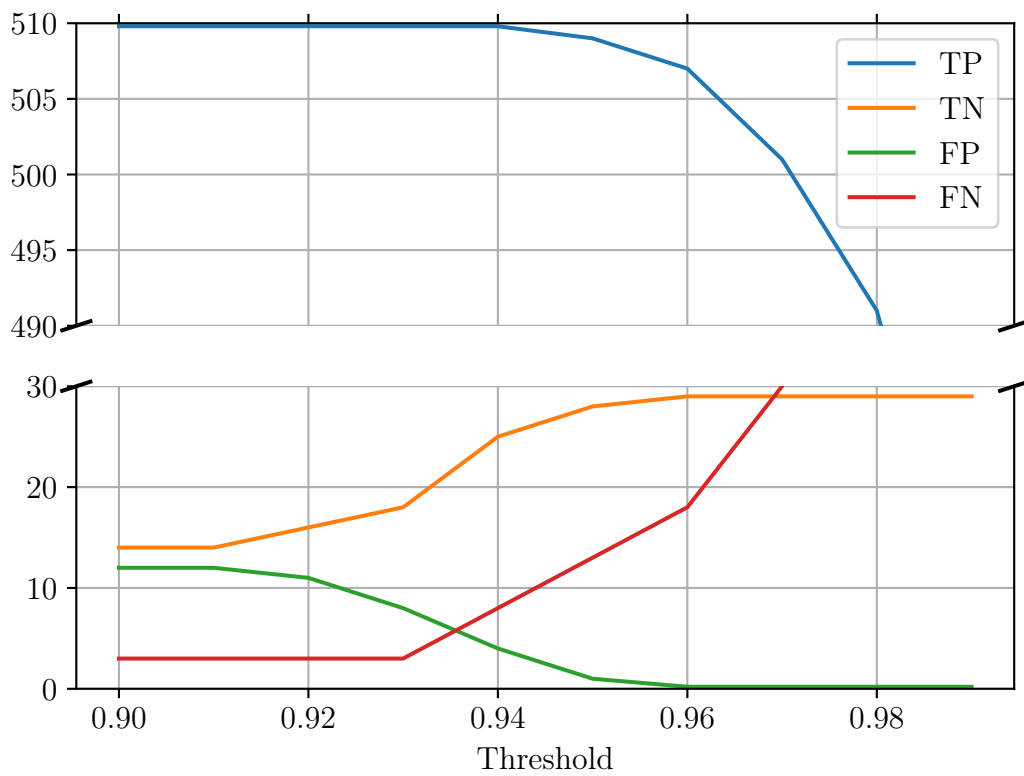


Figure A.7: Results obtained with **RoBERTa** by varying the threshold value.

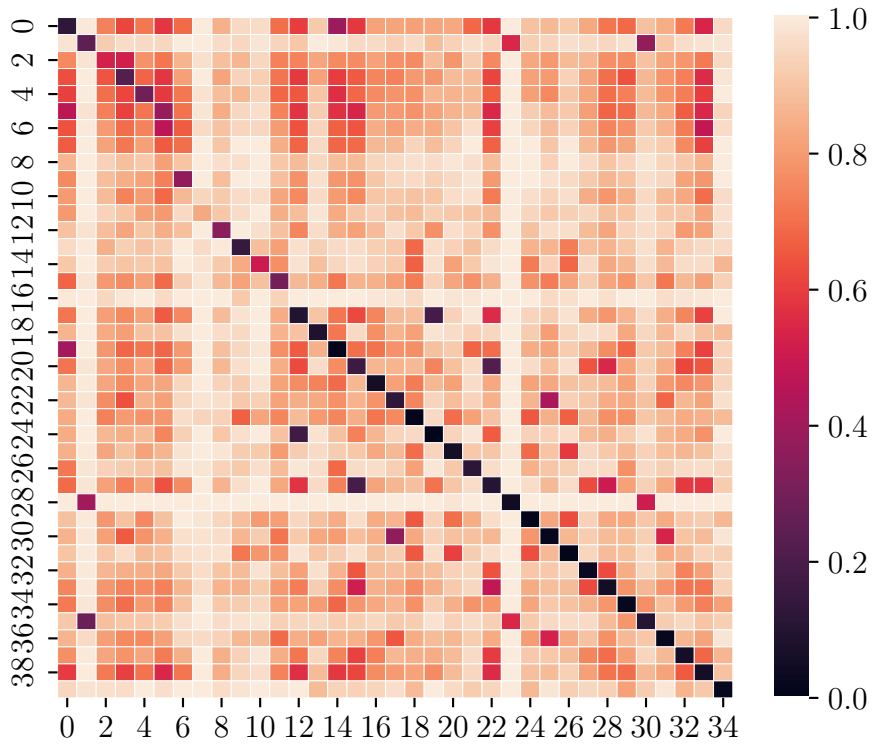


Figure A.8: Heat map of the cost matrix obtained using **TF**.

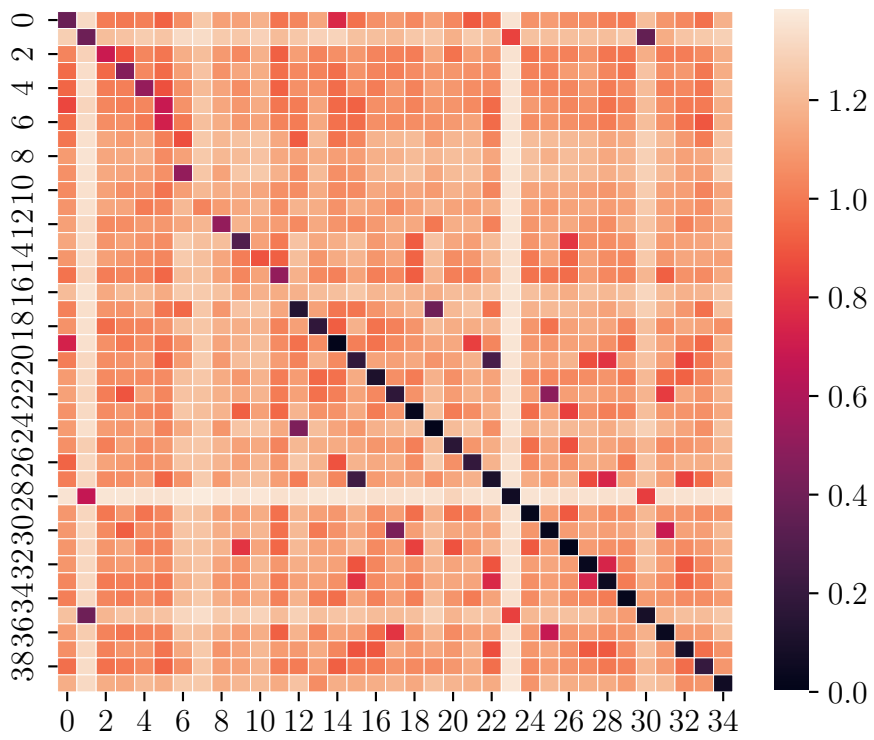


Figure A.9: Heat map of the similarity matrix obtained using **Word2vec**.

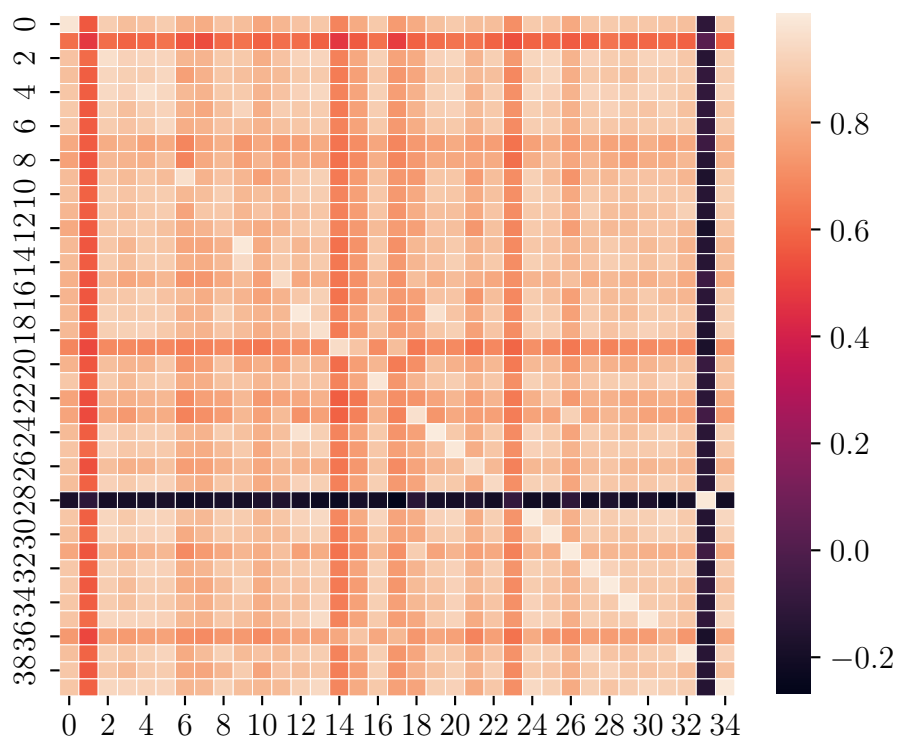


Figure A.10: Heat map of the cost matrix obtained using **BERT**.

TP	TN	FP	FN
515	29	0	0

(a) **Ground truth**

Method	TP	TN	FP	FN
Titles	511	13	11	4
Sections	513	14	9	3
Threshold	509	29	0	14

(b) **TF** ($th = 0.5$)

Method	TP	TN	FP	FN
Titles	511	13	11	4
Sections	514	15	8	2
Threshold	514	29	0	4

(c) **TF-IDF** ($th \in [0.87 - 0.92]$)

Method	TP	TN	FP	FN
Titles	508	13	13	4
Sections	514	15	8	2
Threshold	513	29	0	6

(d) **Word2vec** ($th = 1.03$)

Method	TP	TN	FP	FN
Titles	411	5	125	93
Sections	504	7	32	9
Threshold	491	25	10	44

(e) **Doc2vec** ($th = 0.8$)

Method	TP	TN	FP	FN
Titles	505	8	17	9
Sections	499	9	22	8
Threshold	456	29	1	118

(f) **BERT** ($th = 0.95$)

Method	TP	TN	FP	FN
Titles	508	12	14	5
Sections	510	14	12	3
Threshold	507	29	0	18

(g) **RoBERTa** ($th = 0.96$)

Table A.1: Results of the different models used for the different experiments done. The results for the threshold experiments are the one obtained using the threshold value (th) that gives the least amount of FP, while maximizing the number of TP.