



# Rapport du projet NLP

## RECHERCHE DE MOTS-CLÉS ET EXTRACTION DE LA PHRASE RÉSUMANT LE MIEUX LE DOCUMENT

**Mbathe Mekontchou Paul** **Julian Sliva** **Alexandre Desgrées du Loû** **Alexandre Movsessian**

27 novembre 2024

**ABSTRACT**

Ce projet, réalisé dans le cadre du cours de traitement du langage naturel (NLP), porte sur l'extraction de mots-clés et la synthèse de documents à l'aide de diverses approches. Dans une première phase, nous avons implémenté trois techniques d'extraction de mots-clés : une méthode basée sur TF-IDF et une approche par graphes. La deuxième phase s'est concentrée sur la synthèse en identifiant la phrase la plus représentative de chaque document. Pour cette tâche, nous avons évalué plusieurs méthodes, incluant une synthèse basée sur les mots-clés, une combinaison de TF-IDF et des sémantiques de FastText, une approche de synthèse basée sur BERT, ainsi que trois variantes exploitant l'algorithme d'embedding de mots Roberta pour identifier la phrase du texte qui résume au mieux le texte.

Notre jeu de données se composait d'articles web sur le thème de l'éthique de l'intelligence artificielle, collectés via un web scraping. L'analyse des résultats a montré que certaines méthodes apportaient des améliorations significatives dans l'identification des phrases résumant le mieux les documents. Ce travail montre l'efficacité de la combinaison de techniques statistiques, sémantiques et basées sur l'apprentissage profond pour les tâches de NLP.

***Keywords*** Text Summarization, TF-IDF, PageRank, FastText, BERT, RoBERTa, ScoreBERT, NLP

# Introduction

Ce projet s'inscrit dans le cadre du cours de Natural Language Processing (NLP), pour explorer des méthodes modernes de traitement de texte appliquées à deux problématiques fondamentales : l'extraction de mots-clés et la génération de résumés de documents. Ces deux tâches sont essentielles dans de nombreux domaines, tels que l'analyse de contenu, la veille technologique et la gestion de l'information, où l'objectif est de permettre une compréhension rapide et efficace de larges volumes de données textuelles.

La première partie du projet se concentre sur l'extraction de mots-clés, visant à identifier les termes les plus représentatifs des documents. Pour ce faire, deux approches ont été mises en œuvre : une méthode basée sur les scores TF-IDF et approche utilisant des graphes pour capturer les relations entre les mots.

Dans un second temps, le projet se tourne vers la génération de résumés, consistant à identifier la phrase qui résume le mieux chaque document. Plusieurs stratégies ont été testées : une méthode fondée sur les mots-clés extraits, une approche intégrant TF-IDF et sémantique grâce à FastText, une méthode utilisant le modèle BERT, ainsi que trois variantes basées sur les embeddings de RoBERTa. Ces approches offrent une perspective complémentaire sur l'utilisation de modèles classiques et de modèles profonds pour la tâche de summarization.

Les évaluations ont été réalisées sur un corpus d'articles web portant sur l'éthique de l'intelligence artificielle, une thématique à la fois riche et actuelle. Les résultats obtenus permettent d'identifier les forces et les limites de chaque approche, tout en offrant des pistes pour améliorer les performances dans des contextes similaires.

Ce rapport détaille les méthodologies adoptées, les résultats obtenus et les enseignements tirés, tout en mettant en avant les contributions de ce projet à une meilleure compréhension des concepts et des outils en NLP.

# 1 Téléchargement du corpus et prétraitement

La première étape de notre projet a consisté à télécharger le jeu de données et à le nettoyer afin de pouvoir l'exploiter convenablement.

## 1.1 Téléchargement du corpus

Pour le téléchargement des fichiers constituant notre dataset, nous avons utilisé le dépôt **mapai** mis à notre disposition sur GitLab : `https://gitlab.telecom-paris.fr/tiphaine.viard/mapaie`. Ce dépôt contenait plusieurs fichiers Python couvrant l'ensemble des étapes nécessaires, depuis la collecte initiale des documents jusqu'à leur prétraitement pour des analyses avancées.

Le script `dl_docs.py` a été utilisé pour télécharger automatiquement un grand nombre de documents à partir d'une liste prédéfinie d'URLs. Ce script gère les erreurs HTTP et les métadonnées, tout en supportant plusieurs formats de fichiers tels que PDF et HTML. Une fois les documents collectés, le fichier `parse_docs.py` s'appuie sur la classe `Parser` pour extraire les contenus textuels. Les fichiers HTML sont analysés à l'aide de la bibliothèque `BeautifulSoup`, tandis que les fichiers PDF sont traités avec `PyPDF2`. Ce processus assure une extraction efficace et fidèle du contenu textuel.

Après avoir exploré les scripts fournis dans le dépôt GitLab, nous avons décidé de les adapter pour mener notre propre recherche de mots-clés. Toutefois, nous avons rapidement constaté que certains fichiers du corpus étaient rédigés dans des langues variées, notamment l'espagnol, le chinois et l'italien. Cela a nécessité une étape supplémentaire de traitement linguistique afin d'uniformiser les données et garantir leur compatibilité avec nos outils.

## 1.2 Prétraitement

Dans la phase de prétraitement, nous avons d'abord sélectionné uniquement les fichiers en anglais, puis nous avons travaillé sur les stop words, le nettoyage du texte, la lemmatisation et l'extraction des entités nommées.

### 1.2.1 Sélection des fichiers anglais

Pour cela, nous avons utilisé la bibliothèque langdetect afin de détecter automatiquement la langue de chaque document. Une fois la détection effectuée, nous avons filtré l'ensemble des fichiers pour ne conserver que ceux rédigés en anglais. Au total, 49 fichiers ont été exclus en raison de leur langue différente de l'anglais. Ce processus nous a permis de garantir une homogénéité linguistique dans notre corpus de travail (voir image terminale).". 1

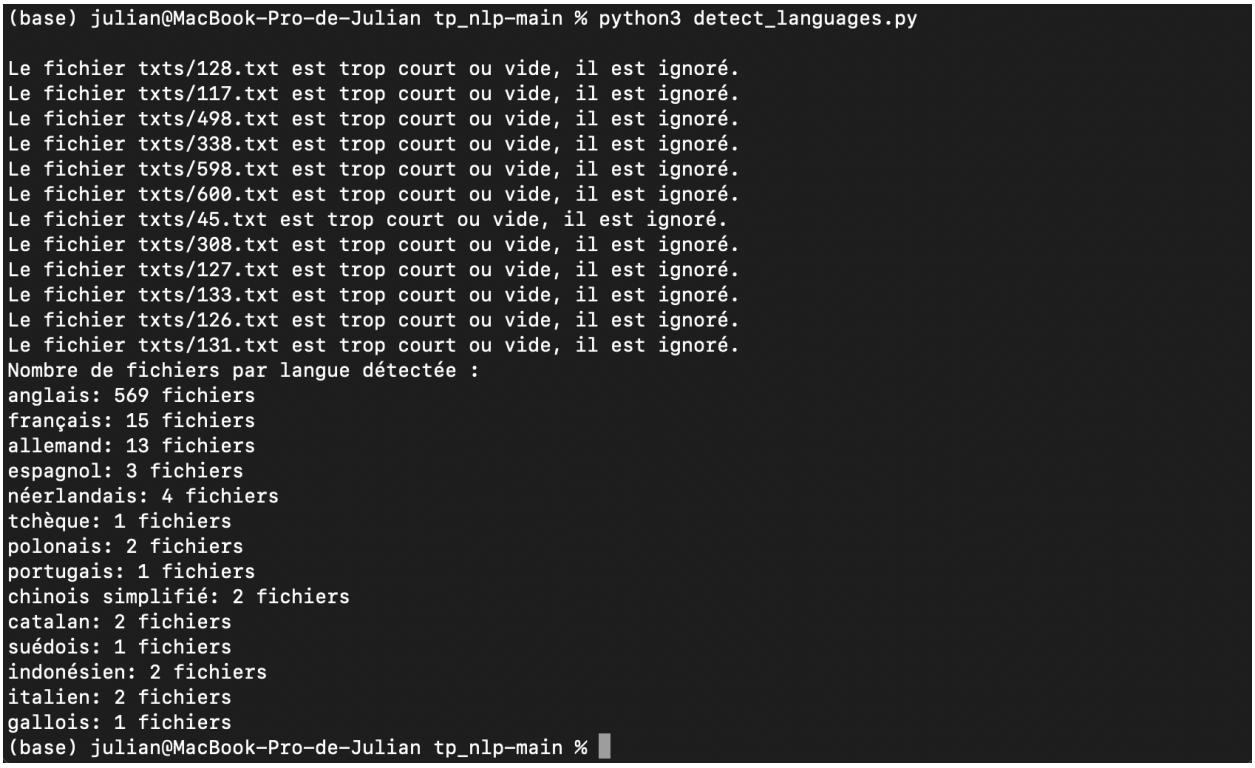



FIGURE 1 – Aperçu des résultats obtenus avec la bibliothèque `langdetect` pour la détection de langue.

### 1.2.2 Suppression des caractères spéciaux et Stop words

Nous récupérons le corpus sous forme d'une liste de texte ayant au préalable été scraper avec *scrap*, *beautifulsoup*. Une fois les documents extraits, nous avons rassemblé l'ensemble des données et appliqué des traitements préliminaires à l'aide d'expressions régulières afin de normaliser le texte et de supprimer les éléments non pertinents :

- Suprresion des caractères spéciaux
- Suppression des #hashtags
- Suppression des @usernames
- Suppression des @mails
- Suppression des urls
- Suppression des digits et des nombres
- Suppression des balises HTML
- Suppression des lignes latex

Une fois cette phase de nettoyage terminée. Pour certaines méthode comme le TF-IDF, les graphes et le clustering, nous avons effectué une tokenisation, qui désigne le découpage en mots des différents documents qui constituent notre corpus.

Parmi les tokens obtenus, les **stop words** anglais, qui sont des mots très fréquents, mais peu informatifs (tels que « and », « or » ou « the »), ont été éliminés. Ces mots sont généralement ignorés, car ils ne contiennent habituellement pas autant d'information que les autres mots du document. Puis, nous supprimons tous les mots qui ne contiennent qu'un seul caractère et ceux dont la fréquence vaut 1. L'ensemble des tokens finaux constitue une partie des mots de notre vocabulaire. Pour compléter ces tokens, nous avons trouvé tous les n-grams, qui sont des groupes de mots du corpus ayant une fréquence supérieure à 1. Après avoir identifié ces tokens et ces n-grams, nous avons effectué une reconnaissance d'entités nommées pour exclure certains mots que nous avons estimés ne pas être pertinents, en particulier pour les techniques basées sur les mots-clés et le clustering.

### 1.2.3 Reconnaissance des entités nommées

La reconnaissance d'entités nommées est une sous-tâche de l'activité d'extraction d'information dans des corpus documentaires qui consistent à chercher les entités nommées mentionnées dans un texte non structuré en catégories prédéfinies, telles que personnes, noms, organisations, lieux, codes médicaux, expressions temporelles, quantités, valeurs monétaires et pourcentages. La reconnaissance des entités nommées nous permet ici de labelliser les tokens pour améliorer nos algorithmes et trouver les mots-clés pertinents.

## 2 Extration des mots-clés

Pour l'extraction de mots-clés, nous nous sommes concentrés sur trois principales approches : une approche utilisant les poids TF-IDF, une approche basée sur FastText et le clustering, et enfin une approche basée sur les graphes.

### 2.1 TF-IDF

Cette technique permet de mettre en évidence les mots importants dans un document ou un groupe de documents en déterminant les poids $tf_idf$ de ces mots dans l'ensemble des phrases qui constituent le document ou les groupes de documents. Pour cela, nous déterminons les poids tf-idf([Salton and McGill(1988)], [Baeza-Yates and Ribeiro-Neto(1999)], [Zhang et al.(2015)]) de chaque mot dans une phrase du document, puis nous trouvons le poids définitif de chaque mot dans le document en faisant la somme des poids des mots dans chaque document où il apparaît. Ensuite, nous trions ces mots-clés et prenons les $top\_n$ mots qui ont les meilleurs scores. Le document résumé est celui dont la somme des scores des mots parmi les $n\_top$ mots-clés est maximale

#### 2.1.1 Calcul du score TF-IDF

Après avoir effectué un préprocessing sur le corpus comme présentée à la section 1.2, Pour chaque token final, nous déterminons, un score $tf\_idf$ dans chaque phrase. Dans notre cas d'usage, phras est considérée comme un document du texte global constituant notre corpus. Le score TF-IDF d'un mot $w$ dans une phrase $p$ est donné par la formule suivante :

$$tf\_idf(p, w) = tf_{p,w} * idf_t = tf_{p,w} * log_2(\frac{N}{N_w})$$

dans cette formule, $N$ représente le nombre de documents du corpus,$N_w$ représente le nombre de phrases qui contienne le mot $w$, $tf_{p,w}$ répresente la fréquence du mot $w$ dans la phrase $p$

#### 2.1.2 Identification des mots-cles

Pour identifier les mots que nous pouvons considérr comme mots-clés dans le corpus de document ou un texte, il nous faut trouver un score de ce mot dans l'ensemble des phrases constituant le corpus, pour cela, nous calculons ce score comme la somme des poids $tf\_idf$ de ce mot dans chaque document. En supposant que notre corpus soit constitué de $n$ phrase, nous calculons ce score comme suit :

$$f(w) = \sum_{i=1}^{n} tf\_idf(w, p_i)$$

#### 2.1.3 Détermination des mots clés

Une fois les poids de chaque mot dans l'ensemble des phrases du corpus déterminés, pour trouver les mots-clés, nous avons deux approches, une première utilisant un nombre de mots fixé à l'avance $top\_n$ mots, ou en utilisant les quantiles.

**Nombre de mots n fixé**

Pour cette approche, Les mots sont triés par ordre décroissant de poids donnés par :

$$W_{\text{triés}} = \{w_1, w_2, \ldots, w_k \quad | \quad f(w_1) \geq f(w_2) \geq \ldots \geq f(w_k)\}$$

Pour un mot $w$ sélectionné, nous pouvons dire que :

$$w \in \{w_1, w_2, \ldots, w_n\}$$

$$\exists w \in W : w \in \{w_1, w_2, \ldots, w_n\} \text{ avec } W_{\text{triés}} = \{w_1, w_2, \ldots, w_k\} \text{ et } f(w_1) \geq f(w_2) \geq \ldots \geq f(w_k)$$

cela revient à trouver les $n$ mots les ayant les poids les plus élévés.

**Utilisation des quantiles**

Pour déterminer les mots pertinents, nous calculons un quantile $q$ (où $0 < q < 1$) des scores $f(w)$. Nous définissons alors :

$$Q_q = \text{quantile}(f(w) \,|\, w \in W, \text{ pour } f(w) \text{ trié par ordre croissant})$$

Les mots pertinents peuvent être exprimés par l'ensemble :

$$P = \{w \in W \,|\, f(w) \geq Q_q\}$$

## 2.2 Méthode basée sur les graphes

L'approche TF-IDF précédente a donné des résultats intéressants, mais il existe aussi d'autres possibilités pour extraire des mots-clefs d'un texte.

Dans cette partie, nous utiliserons plusieurs approches qui se basent sur l'utilisation d'une représentation du texte sous forme de graphe.

Ces approches sont recensées dans le compte-rendu de conférence "Graph-based Text Representations : Boosting Text Mining, NLP and Information Retrieval with Graphs" [Fragkiskos D. Malliaros(2017)]

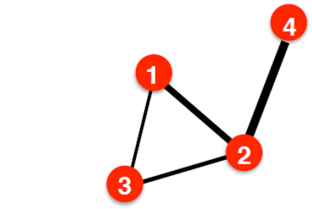
Pour créer un graphe à partir d'un texte, on considère chaque mot comme un noeud. Chaque juxtaposition entre deux mots est considérée comme une arête entre leurs nœuds. Ces arêtes sont pondérées, c'est-à-dire qu'elles possèdent une valeur représentant le nombre de fois où deux mots sont proches dans un texte.

Cette notion de proximité entre deux mots est définie par un paramètre max_distance. Ce paramètre est un entier qui définit la distance maximum à laquelle deux mots sont considérés comme proches.

Par exemple, pour max_distance = 2, dans la phrase « Le chat mange la souris », « chat » est proche de « Le », « mange », « la », mais pas de souris qui est à une distance de 3.

À partir de cette représentation par graphe, on peut aussi créer une représentation sous forme matricielle, en tant que matrice de co-occurrence, ou matrice d'adjacence.

À noter que cette matrice d'adjacence est parfois transformée par l'inverse, car nous appliquons nos algorithmes aux distances entre les mots, qui sont donc considérées comme l'inverse du nombre d'occurrences (Plus des mots sont proches, plus, ils apparaissent souvent ensemble).




$$A_{ij} = \begin{pmatrix} 0 & 2 & 0.5 & 0 \\ 2 & 0 & 1 & 4 \\ 0.5 & 1 & 0 & 0 \\ 0 & 4 & 0 & 0 \end{pmatrix}$$

$$A_{ii} = 0 \qquad A_{ij} = A_{ji}$$

FIGURE 2 – Graphe et matrice d'adjacence associée[Fragkiskos D. Malliaros(2017)]

### 2.2.1 Algorithme HITS

L'algorithme HITS (Hyperlink-Induced Topic Search) est un algorithme initialement conçu pour analyser les structures de liens hypertextes sur le web, mais il peut également être adapté pour déterminer les mots les plus importants dans un texte, en fonction de leur rôle dans la structure du réseau des relations entre les mots.

L'algorithme HITS repose sur deux notions clés : hub et authorities. Ces concepts sont utilisés pour classer les éléments d'un réseau. Dans le cadre de l'analyse de texte, cela peut être appliqué de manière suivante :

1. **Hub** : Un mot est considéré comme un "hub" si, à partir de ce mot, il existe de nombreuses connexions vers d'autres mots importants.
2. **Authority** : Un mot est considéré comme une "autorité" s'il est fortement relié à de nombreux hubs.

Dans notre cas, nous recherchons les mots avec la plus forte Authority.

**Algorithm 1** Algorithme HITS (Hyperlink-Induced Topic Search)

**Entrée :** Graphe orienté $G = (V, E)$, où $V$ est l'ensemble des nœuds (mots) et $E$ est l'ensemble des arêtes (co-occurrences)
**Sortie :** Scores de Hub et d'Autorité pour chaque nœud de $V$
Initialiser les scores de Hub : $h(v) = 1$ pour tous les $v \in V$
Initialiser les scores d'Autorité : $a(v) = 1$ pour tous les $v \in V$
**repeat**
    **for** chaque nœud $v \in V$ **do**
        Mettre à jour le score d'Autorité de $v$ :

$$a(v) = \sum_{u \in In(v)} h(u)$$

    **for** chaque nœud $v \in V$ **do**
        Mettre à jour le score de Hub de $v$ :

$$h(v) = \sum_{u \in Out(v)} a(u)$$

    Normaliser les scores de Hub et d'Autorité :

$$h(v) = \frac{h(v)}{\max_{v' \in V} h(v')} \quad \text{et} \quad a(v) = \frac{a(v)}{\max_{v' \in V} a(v')}$$

**until** Convergence OU maximum d'itérations atteint
**Retourner :** Les scores finaux d'Autorité pour chaque nœud $v$

Cet algorithme converge lentement et son temps de calcul augmente très vite avec la taille des données, ce qui fait qu'il est peu adapté à des textes longs. Il est aussi très sensible aux cycles intérieurs aux graphes, et peut ne pas converger à cause de tels cycles.

En revanche, cet algorithme est très efficace sur un texte très hétérogène. Il permet de clairement séparer les hubs, soit les mots les plus centraux du texte, des autorités, qui sont les mots les plus importants. Un texte avec une distinction importante entre ces deux catégories donnera de bons résultats.

De plus, cet algorithme est particulièrement adapté aux graphes orientés. Nous n'en ferons pas l'étude ici, mais utiliser un graphe orienté en liant les mots uniquement aux mots suivants (ou précédents) permet de garder beaucoup d'information au niveau sémantique. Nous aurions alors une matrice de co-occurrence qui n'est pas symétrique. En l'absence de ces graphe orientés, l'algorithme HITS se rapproche beaucoup de l'algorithme du TextRank.

### 2.2.2 Algorithme TextRank

Le PageRank repose sur une idée simple : Si une page web A contient un lien vers une autre page web B, cela peut être considéré comme une forme de "référence" ou de "recommandation" pour la page B. L'algorithme suppose que les pages les plus référencées (ou les pages ayant des liens provenant de pages importantes) sont probablement les plus pertinentes.

TextRank est un algorithme inspiré de PageRank, mais il est spécifiquement conçu pour l'analyse de texte. Il est fait sur le principe selon lequel les mots les plus importants du texte seront connectés avec d'autres mots importants.

**Algorithm 2** Algorithme TextRank pour l'extraction de mots-clés

**Entrée :** Document texte $D$
**Sortie :** Mots-clés classés $K$
Construire un graphe $G = (V, E)$, où $V$ est l'ensemble des mots et $E$ représente les co-occurrences entre les mots
Pour chaque paire de mots $(w_1, w_2)$ dans le document, ajouter une arête entre $w_1$ et $w_2$ avec un poids basé sur leur co-occurrence
Inverser les poids des arêtes pour obtenir la proximité entre les mots
**for** chaque mot $w \in V$ **do**
    Initialiser $score(w) = 1.0/|V|$
**repeat**
    **for** chaque mot $w \in V$ **do**
        Mettre à jour le score de $w$ :

$$score(w) = (1 - d) + d \times \sum_{w' \in N(w)} \frac{score(w')}{\text{degré}(w')}$$

    où $N(w)$ est l'ensemble des voisins de $w$, et $d$ est le facteur d'amortissement (damping factor)
**until** Convergence OU maximum d'itérations atteint
Trier les mots $w \in V$ par leur score $score(w)$ en ordre décroissant.
**Retourner :** Les mots-clés les mieux classés $K$

Cet algorithme utilise un damping factor, ou facteur d'amortissement. Ce facteur permet d'éviter les cycles et les boucles infinies en introduisant une probabilité de saut aléatoire entre deux nœuds. En général, il est choisi autour de 0.85.

L'algorithme du TextRank est très efficaces, que les textes soient courts ou longs. Il est aussi très efficace indépendamment du langage utilisé.

En revanche, il est très sensible au bruit dans le texte. Le texte a besoin d'être nettoyé proprement, car certains mots fréquents, mais sans information sémantique, comme les stopwords ou les conjonctions de coordination, sont logiquement considérés comme centraux et obtiennent un très bon score devant d'autres mots plus importants, mais moins centraux.

De la même façon, si un texte contient des mots qui sont fréquemment répétés, TextRank peut surévaluer leur importance, même si ces mots ne sont pas réellement significatifs pour le contenu du texte.

### 2.2.3 Algorithme K-Core

L'algorithme du K-Core consiste essentiellement à répéter un processus de suppression de nœuds dans un graphe, en fonction du nombre de voisins qu'ils ont, jusqu'à ce qu'on obtienne un sous-graphe où tous les nœuds restants ont au moins K voisins.

Une fois qu'un nœud est supprimé, ses voisins sont affectés, car le nombre de leurs voisins diminue. On procède donc par itérations.

**Algorithm 3** Algorithme K-Core

**Entrée :** Graphe $G = (V, E)$, où $V$ est l'ensemble des nœuds et $E$ est l'ensemble des arêtes
**Sortie :** Le sous-graphe $G_k$ représentant le k-core du graphe $G$
Initialiser $k = 1$
Initialiser $G_k = G$
**repeat**
    **for** chaque nœud $v \in V(G_k)$ **do**
        **if** degré($v$) < k **then**
            Supprimer le nœud $v$ et toutes ses arêtes de $G_k$
    Mettre à jour le sous-graphe $G_k$
    Incrémenter $k$
**until** Il n'y a plus de nœuds à supprimer dans $G_k$
**Retourner :** Le sous-graphe $G_k$

Cet algorithme est extrêmement simple à comprendre et à mettre en place. Il est aussi très efficace sur de longs textes. Il identifie facilement des clusters de mots importants. Il est utile comme première approche, car il permet de se représenter les parties les plus importantes du graphe, sans pour autant les classer.

En revanche, ses résultats sous forme de clusters peuvent être difficiles à interpréter et dépendent fortement du choix de k. De plus, il ne prend absolument pas en compte la pondération des arêtes du graphe, il compte uniquement le nombre de liaisons qu'un mot a avec les autres.

### 2.2.4 Comparaison des résultats

Pour comparer les performances de nos algorithmes, nous les avons utilisés sur chaque document et en avons reçu les dix mots les plus fréquents, par document, par algorithme. À partir de là, nous avons créé un histogramme des fréquences pour chaque algorithme, afin de déterminer les mots-clefs les plus fréquents du corpus. Les résultats complets sont disponibles sous la forme d'un tableur Excel sur le GitHub :

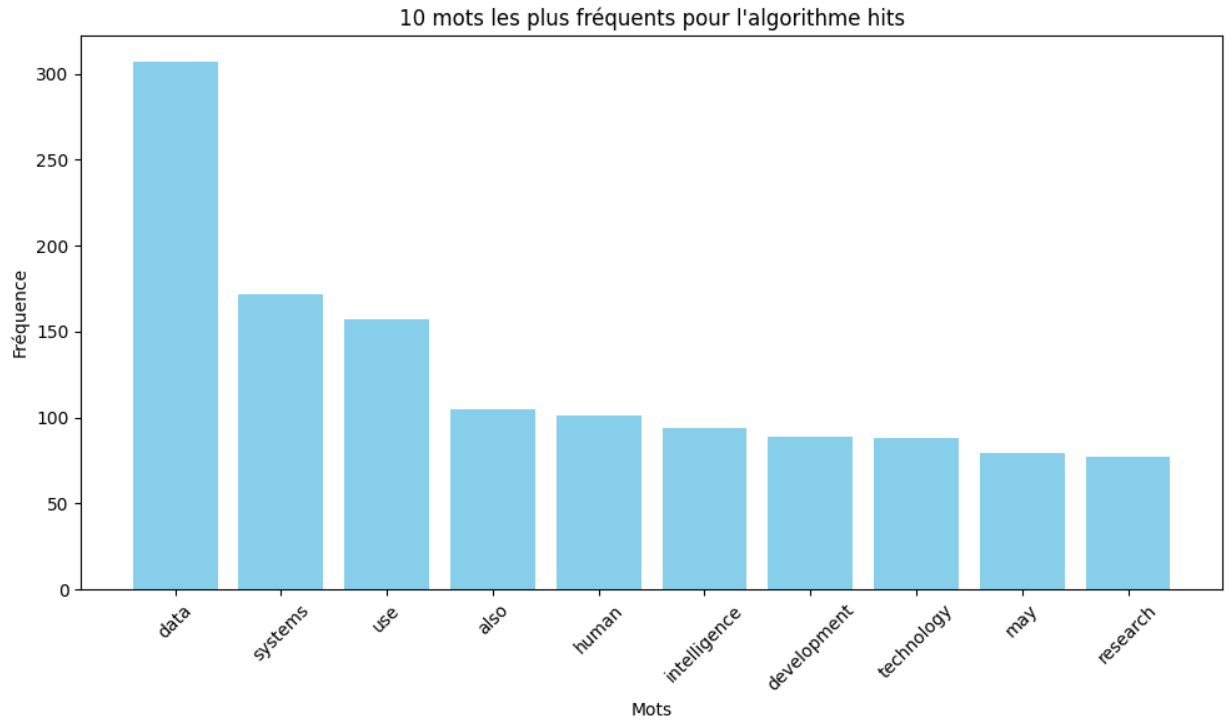



FIGURE 3 – Les 10 mots les plus fréquents dans le corpus entier selon l'algorithme HITS

On observe que l'algorithme HITS donne d'excellents résultats sur le corpus entier, les mots-clefs qu'il a produits correspondent pour la plupart aux thématiques adressées par le corpus de document Mapaie. On observe cependant un "also" en 4 position et un "may" en 9e position, ce qui nous indique que le corpus aurait pu être encore plus nettoyé pour enlever les mots de liaisons et les auxiliaires qui viennent polluer les résultats.

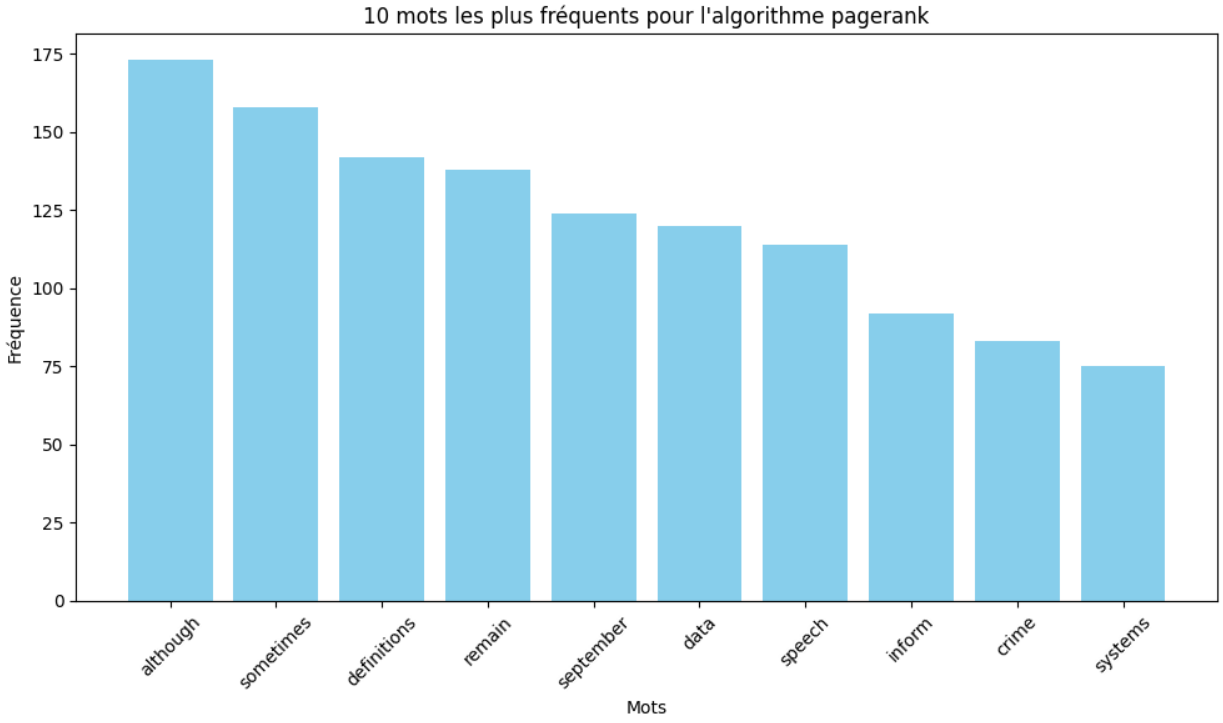



FIGURE 4 – Les 10 mots les plus fréquents dans le corpus entier selon l'algorithme TextRank

Comme on l'avait dit plus haut, l'algorithme du TextRank est très sensible au bruit, et c'est ce qu'on constate ici. Certains mots parasites comme "although" et "sometimes", très utilisés dans tous les contextes, prennent une place prépondérante face à autres motsts plus porteur de sens, mais moins utilisés. On constate cependant que cet algorithme donne de très différents résultats du HITS, même en ignorant les mots inutiles. Les seuls mots en commun sont "data" et "systems", et se retrouve à la 6e et 10e place alors qu'ils étaient 1 et 2e auparavant.

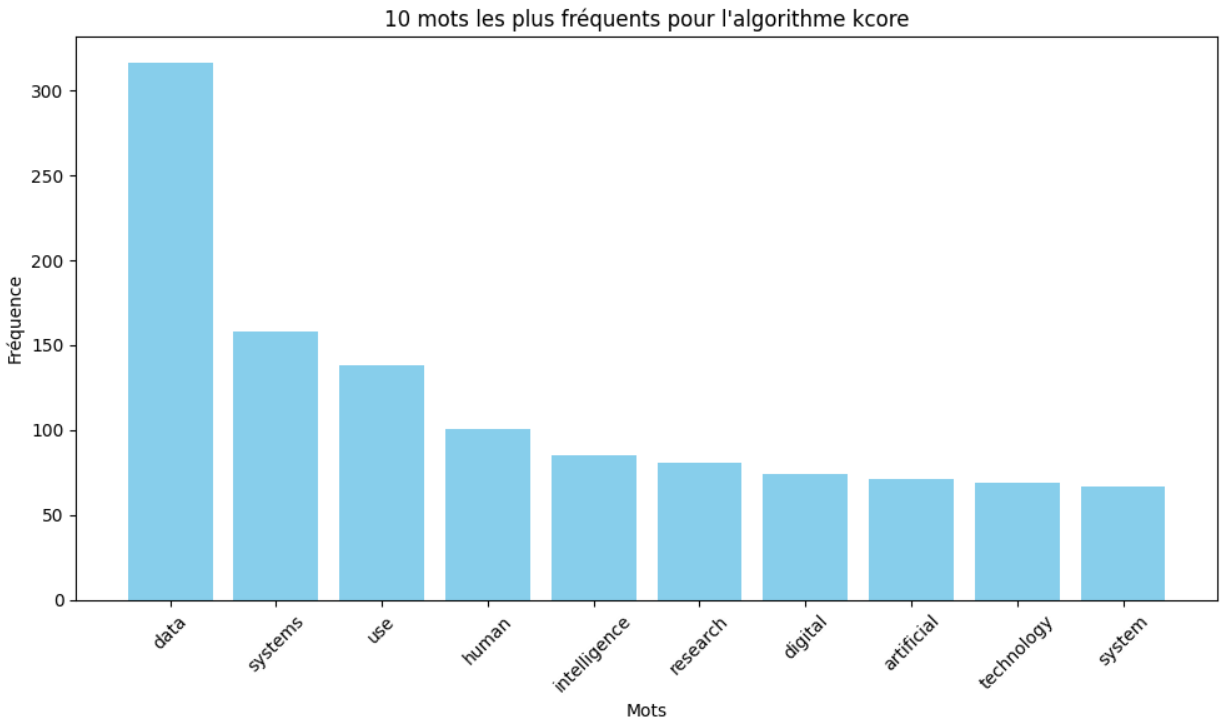



FIGURE 5 – Les 10 mots les plus fréquents dans le corpus entier selon l'algorithme K-Core

L'algorithme du K-Core est celui qui donne les meilleurs résultats sur l'ensemble de notre corpus. Il donne des résultats similaires à l'algorithme HI, maisis ignore les mots redondants comme les auxiliaires. Les 10 mots-clefs les plus fréquents donnés par l'algorithme correspondent exactement aux thématiques du corpus mapaie.

## 3 Résumé de documents

Notre méthodologie pour résumer les documents ici, consiste à trouver la phrase la plus pertinente qui résume au mieux le document, pour cela, nous avons exploré plusieurs approches.

### 3.1 Approche utilisant TF-IDF

Cette technique exploite les mots-clés retournés par la technique utilisant les poids $tf\_idf$. Pour cela, nous déterminons les poids tf-idf de chaque mot dans chaque phrase du document, puis nous trouvons le poids définitif d'un mot dans le document en faisant la somme des poids de ce mot dans chaque phrase où il intervient. Ensuite, nous trions ces mots-clés et prenons les top_n mots qui ont les meilleurs scores. La phrase qui résume au mieux le document est celle dont la somme des scores des mots parmi les n_top mots-clés est maximale.

#### 3.1.1 Représentation des documents

Une fois que nous avons trouvé les mots-clés en utilisant la méthode de la session 2.1, les phrases sont représentées par des vecteurs dans l'espace formé par les mots-clés trouvés. Par exemple, en supposant que nous ayons trouvé $m$ mots-clés, une phrase $p_i$ sera représentée par un vecteur $(w_i^1, w_i^2, \ldots, w_i^m)$ où $w_i^j$ représente le score $tf$-$idf$ du mot-clé $j$ dans le document $i$.

### 3.2 Détermination des documents pertinents

Une fois les phrases représentées dans cet espace, le poids global d'une phrase est donné par la formule $f(p_i) = \sum_{j=1}^{m} w_i^j$. Une fois les poids de chaque phrase calculés, nous trions ces phrases par ordre de poids décroissant et nous prenons les top_$k$ premières phrases comme celles qui résument le mieux le document dans cet ordre.

| | mots 1 | mots 2 | ... | mots m |
|---|---|---|---|---|
| phrase 1 | 0.4 | 0.2 | | 0 |
| phrase 2 | 0 | 0.7 | | 0.1 |
| ⋮ | ⋮ | ⋮ | | |
| phrase n | 0.9 | 0.8 | | 0.1 |
| som | 0.7 | 0.5 | | 0.2 |

FIGURE 6 – Resume de document basé sur mots-clés

### 3.3 Approche utilisant TF-IDF+Clustering+Fasttext

Dans cette approche, nous utilisons FastText pour réaliser un embedding sémantique des mots du corpus, puis nous regroupons sémantiquement les mots en clusters en nous basant sur leur signification. Pour le regroupement, nous appliquons un seuil avec l'algorithme DBSCAN. Une fois que les clusters ont été construits, nous calculons les scores TF-IDF de chaque cluster dans les phrases du corpus, en considérant la somme des scores TF-IDF de chaque mot appartenant au cluster, multiplié par un coefficient de pondération. Une fois les scores TF-IDF de chaque cluster dans les phrases calculé, nous identifions les clusters les plus pertinents en cherchant ceux dont la somme des poids dans toutes les phrases est maximale. Une fois ces $n_{\text{top\_clusters}}$ sélectionnés, les meilleures phrases sont celles dont la somme des scores TF-IDF des mots présents dans le cluster est maximale.

#### 3.3.1 Reconnaissance d'entité nommées

La reconnaissance des entités nommées permet d'étiqueter les mots du texte afin d'améliorer l'algorithme de regroupement des mots en grappes. En effet, les entités nommées n'ont pas nécessairement de synonymes et peuvent être une source d'ambiguïté. Au regard de la manière dont les word embeddings sont entraînés, les noms propres peuvent être représentés par des vecteurs proches, car ils sont généralement utilisés dans le même contexte (par exemple, « Astérix » et « Obélix »). Il en va de même pour les noms de lieux et autres noms. La reconnaissance des entités nommées permet ici d'éviter ces ambiguïtés qui auraient eu un impact négatif sur notre méthode. Ainsi, les entités nommées sont isolées dans leur cluster et ne se superposent pas à d'autres mots.

### 3.4 Groupement de mots en clusters

Le texte annoté est transformé en jetons pour constituer le vocabulaire du corpus. La liste de tokens qui en résulte est utilisée pour construire des clusters.Cependant, certaines contraintes doivent être prises en compte :

- Le nombre de clusters ne doit pas être connu à l'avance. En effet, la taille du vocabulaire n'est pas fixe et peut donc évoluer.
- L'algorithme doit être incrémental afin de permettre l'ajout de nouveaux mots aux clusters sans avoir à modifier globalement les grappes déjà formées.
- Le centroïde doit être fixe pour éviter qu'un groupe ne prenne en compte des mots de sens différents.
- Les clusters formés doivent être linéairement indépendants d'un point de vue sémantique pour qu'ils forment un espace vectoriel.

— Les mots rares (RW) et les entités nommées (NE) doivent être isolés dans leurs groupes.

Compte tenu de ces différentes contraintes, nous proposons un algorithme pour la construction des clusters, basé sur l'algorithme à passage unique (Single Pass Algorithm) [Gupta and Grossman(2004)].

**Algorithm 4** Grouping words from the collection into clusters

**Require:** $\epsilon \geq 0$

$Cluster\ \ new\_cluster$ ▷ Create a new cluster
**if** $clusters.length = 0\ \ or\ \ w.label = "NE - RW"$ **then**
    $new\_cluster.centroid \leftarrow w.vecteur$
    $add\_word(new\_cluster.words,\ w.text)$
    $add\_cluster(clusters,\ new\_cluster)$
**else**
    $cluster\_index \leftarrow None$
    $cluster\_index \leftarrow find\_closest\_cluster(clusters, w, \epsilon)$
    **if** cluster_index = None **then**
        $new\_cluster.centroid \leftarrow w.vecteur$
        $add\_word(new\_cluster.words.add, w.text)$
        $add\_cluster(clusters,\ new\_cluster)$
    **else**
        $add\_word(clusters[cluster\_index].words,\ w.text)$

### Description

— Si la liste des clusters est vide ou si le mot traité est une entité nommée ou un mot rare, un nouveau cluster est créé.
— Sinon, nous recherchons le cluster le plus proche du mot avec le seuil $\epsilon$. Ceci se fait en comparant l'intégration de ce mot avec chaque centroïde de cluster et en vérifiant si le score de similarité est inférieur à $\epsilon$.
— Si le mot n'appartient à aucun groupe, nous créons un groupe.
— Si le mot correspond à un groupe, il est ajouté à la liste des mots de ce groupe sans que le centroïde soit modifié.
— Les fonctions *add_word(L,w)* et *add_cluster(L,w)* permettent d'ajouter un mot $w$ à la liste des mots ou au cluster respectivement.

En recherchant le cluster le plus proche du mot plutôt que le premier groupe dans lequel le mot pourrait être ajouté, on réduit l'ambiguïté en s'assurant qu'un mot n'appartient pas à plus d'un groupe.

### Sélection du seuil $\epsilon$

Le paramètre $\epsilon$ est déterminé en sélectionnant $N$ paires de mots de la langue cible, qui sont connus pour être synonymes. Le paramètre $\epsilon$ est la moyenne des scores obtenus pour toutes les paires.

### Complexité

L'ajout d'un mot nécessite, dans le pire des cas, de passer par tous les clusters pour trouver celui qui est le plus proche. Pour chaque itération de la boucle, il faut calculer la distance entre le centroïde du cluster et le vecteur associé au mot. L'algorithme s'exécute donc en $O(n \cdot \beta)$, où $n$ est le nombre de clusters au moment de l'insertion du mot,ot et $\beta$ est la complexité pour calculer le cosinus entre le centroïde et le vecteur du mot.

La figure la 7


FIGURE 7 – exemple de clusters

Dans notre implémentation, nous utilisons les enchâssements *fastText* pour construire les clusters. Ces clusters sont ensuite sauvegardés dans Elasticsearch avec leurs vecteurs centroïdes.

### 3.5 Représentation d'une phrase dans l'espace des clusters

Selon la contrainte sur la construction des clusters, ceux-ci sont linéairement indépendants et forment un espace vectoriel. Nous représentons donc chaque phrase dans cet espace.
En supposant que l'on dispose de $n$ clusters construits à partir des phrases du corpus traité, la phrase $p_j$ est donnée par le vecteur :

$$\vec{p_j} = (\alpha_1^j, \alpha_2^j, \ldots, \alpha_n^j)$$

où $\alpha_i^j$ est le poids du cluster $i$ dans la phrase $p_j$.

**Calcul du poids $\alpha_i^j$ du cluster $i$ dans la phrase $j$**

Pour le calcul du poids $\alpha_i^j$, nous utilisons le principe de la pondération *TF-IDF* que nous adaptons de manière originale au cas des clusters. Le poids $\alpha_i^j$ du cluster $i$ dans la phrase $j$ est calculé avec la formule suivante :

$$\alpha_i^j = \beta \sum_{w \in C_i} tf\_idf(w, j)$$

— $\beta$ est un facteur qui représente la proportion de mots dans le cluster $C_i$ qui appartiennent à la phrase $p_j$.
— $tf\_idf(w, j)$ est le poids tf_idf du mot $w$ dans la phrase $j$

**Justification**

Nous avons ajouté le facteur $\beta$ pour augmenter le poids d'un cluster s'il contient plusieurs mots distincts. En calculant la fréquence d'un cluster comme la somme des fréquences des mots qu'il contient, nous favorisons ainsi les clusters qui ont plusieurs mots apparaissant plusieurs fois dans une phrase. En effet, l'objectif ici est de minimiser le poids des clusters qui ont trop de mots et qui partagent peu de mots avec un document, car ces clusters vont créer de l'ambiguïté. Supposons que nous ayons deux clusters $C_1$ et $C_2$ et un document $d$. Le cluster $C_1$ contient 10 mots et le cluster $C_2$ en contient 5. En supposant que le cluster $C_1$ contienne 3 mots dans le document $d$ et que $C_2$ contiennent également trois mots dans ce document, en calculant le facteur $\beta$ de chacun, nous obtenons $\frac{3}{10}$ et $\frac{3}{5}$ respectivement pour chacun des clusters. Notre idée ici est de minimiser le poids des clusters qui ont trop de mots et qui partagent peu de mots avec un document donné, car ces clusters créeront de l'ambiguïté.

#### 3.5.1 Détermination des phrases pertinentes pour le résumé

Une fois les phrases représentées dans l'espace des clusters, le poids global d'une phrase est donné par la formule $f(p_i) = \sum_{j=1}^{m} \alpha_i^j$. Une fois les poids de chaque document calculés, nous trions ces phrases par ordre de poids décroissant et nous prenons les top_$k$ premières phrases comme celles qui résument le mieux le document dans cet ordre.

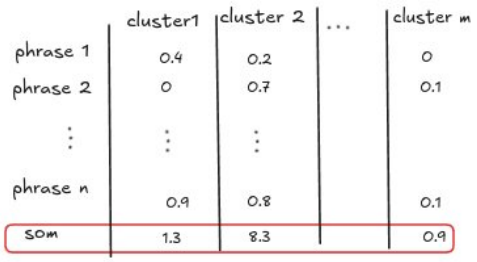



FIGURE 8 – Résume basé sur Fasttext et le clustering

### 3.6 Approche utilisant BERT

BERT (Bidirectional Encoder Representations from Transformers) est un modèle de traitement du langage naturel proposé par Google AI en 2018 [Devlin et al.(2019)]. Il était innovant dans la compréhension contextuelle des séquences de texte grâce à son entraînement bidirectionnel, permettant de capturer le contexte à la fois à gauche et à droite d'un mot cible. BERT est basé sur l'architecture Transformer, initialement introduite par Vaswani et al. [Vaswani et al.(2023)], qui repose sur un mécanisme d'auto-attention pour traiter les dépendances globales entre les mots. Contrairement aux approches unidirectionnelles précédentes, comme les modèles de langage traditionnels, cette architecture permet une modélisation plus riche du contexte.

La pré-formation de BERT repose sur deux tâches principales : le *Masked Language Modeling* (MLM) et la *Next Sentence Prediction* (NSP). Dans le MLM, certains mots d'une séquence sont masqués de manière aléatoire, et le modèle doit les prédire en s'appuyant sur le contexte environnant. Cette méthode d'entraînement bidirectionnelle permet à BERT de dépasser les limites des modèles traditionnels qui ne considèrent le contexte que dans une seule direction. La tâche NSP, quant à elle, consiste à prédire si deux phrases données sont consécutives dans un corpus, ce qui améliore la compréhension des relations entre les phrases.

Dans ce projet, BERT a été utilisé sans modifications structurelles. Une attention particulière a été portée sur l'exploitation des embeddings contextuels générés par le modèle. Ces représentations capturent non seulement les significations lexicales des mots, mais aussi leurs relations sémantiques et syntaxiques au sein de la phrase. Contrairement aux méthodes basées sur des vecteurs statiques comme Word2Vec ou GloVe, BERT produit des embeddings dynamiques qui varient selon le contexte dans lequel les mots apparaissent.

#### 3.6.1 Representation vectorielle des phrases et résumé

Soit $D$ un document composé de $n$ phrases :

$$D = \{P_1, P_2, \ldots, P_n\}$$

Nous utilisons BERT pour obtenir l'embedding de chaque phrase $P_i$ :

$$E_i = \text{BERT}(P_i) \quad \text{pour } i = 1, 2, \ldots, n$$

L'embedding du document $E_D$ est obtenu en faisant la moyenne des embeddings de toutes les phrases :

$$E_D = \frac{1}{n} \sum_{i=1}^{n} E_i$$

Pour chaque phrase $P_j$ dans le document, nous calculons son embedding $E_j$ :

$$E_j = \text{BERT}(P_j) \quad \text{pour chaque } P_j \in D$$

Nous calculons ensuite la similarité cosinus entre l'embedding du document et chaque embedding de phrase :

$$\text{similarity}(E_D, E_j) = \frac{E_D \cdot E_j}{\|E_D\| \|E_j\|}$$

où $\cdot$ représente le produit scalaire et $\| \cdot \|$ représente la norme du vecteur.

La phrase qui résume le mieux le document est celle dont la similarité cosinus est maximale :

$$P^* = \arg\max_{P_j \in D} \text{similarity}(E_D, E_j)$$

### 3.7 Approche utilisant Roberta

RoBERTa (Robustly Optimized BERT Pretraining Approach) est un modèle de traitement du langage naturel développé par Facebook AI en 2019. Il s'inscrit dans la continuité du modèle BERT (Bidirectional Encoder Representations from Transformers), en optimisant et en adaptant ses méthodes d'entraînement pour de meilleures performances sur diverses

tâches de compréhension de texte. RoBERTa conserve l'architecture de BERT, mais introduit plusieurs améliorations, notamment une pré-formation plus longue sur de grands corpus non étiquetés, une augmentation de la taille des mini-lots, et une optimisation du masque appliqué aux séquences pendant l'apprentissage. Ces ajustements permettent à RoBERTa d'atteindre de meilleurs résultats à ceux de BERT sur plusieurs benchmarks standard. Cependant, contrairement à BERT, RoBERTa ne repose pas sur la tâche de prédiction de la prochaine phrase [Liu et al.(2019)], [Face(2020)].

Contrairement à des approches traditionnelles, nous n'avons pas appliqué de suppression des *stop words* lors de l'utilisation de RoBERTa. Cette décision repose sur le fait que les *embeddings* générés par RoBERTa encapsulent des informations contextuelles riches, y compris pour des mots courants comme « and », « or », ou « the ». Ces mots, bien que souvent considérés comme non pertinents dans des méthodes statistiques, contribuent à la compréhension globale du contexte et des relations syntaxiques dans les modèles de type transformer. RoBERTa est particulièrement efficace pour d'autres tâches pour l'analyse de sentiment, la classification de texte, la compréhension des relations entre les mots...

### 3.7.1 Similarité cosinus entre chaque phrase et le texte entier

La première méthode consiste à calculer la similarité cosinus entre les embeddings de chaque phrase et ceux du texte entier, obtenus à l'aide de RoBERTa. Les embeddings capturent des informations contextuelles, ce qui permet de représenter chaque phrase et le document global sous forme de vecteurs dans un espace multidimensionnel. La similarité cosinus mesure la proximité sémantique entre ces vecteurs. La phrase ayant la plus grande similarité cosinus avec le document complet est considérée comme la plus représentative. Cette approche exploite la capacité de RoBERTa à intégrer à la fois le sens lexical des mots et leur contexte dans le texte, permettant d'identifier des phrases pertinentes même dans des documents complexes.

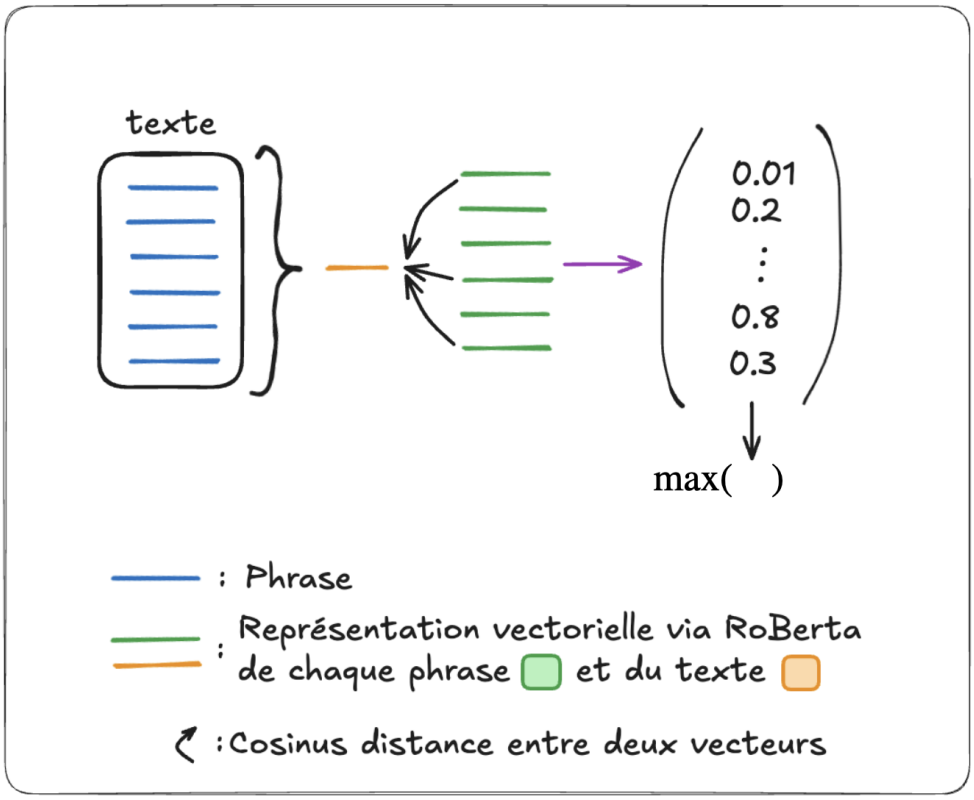



FIGURE 9 – Schéma de la première méthode de RoBERTa

## Représentation des Embeddings

Soit $T$ un texte complet et $P_i$ une phrase extraite du texte. Nous désignerons par $\mathbf{v}_T$ l'embedding du texte $T$ obtenu avec RoBERTa, et par $\mathbf{v}_{P_i}$ l'embedding de la phrase $P_i$ obtenue avec RoBERTa.

## Calcul de la Similarité Cosinus

La similarité cosinus entre les vecteurs $\mathbf{v}_T$ et $\mathbf{v}_{P_i}$ est définie par la formule suivante :

$$\text{similarité\_cosinus}(P_i, T) = \frac{\mathbf{v}_{P_i} \cdot \mathbf{v}_T}{\|\mathbf{v}_{P_i}\| \|\mathbf{v}_T\|} \tag{1}$$

où :
- $\mathbf{v}_{P_i} \cdot \mathbf{v}_T$ est le produit scalaire des deux vecteurs.
- $\|\mathbf{v}_{P_i}\|$ et $\|\mathbf{v}_T\|$ sont les normes (ou longueurs) des vecteurs $\mathbf{v}_{P_i}$ et $\mathbf{v}_T$, respectivement.

## Identification de la Phrase la Plus Proche

Pour chaque phrase $P_i$ dans le texte, nous calculons la similarité cosinus avec le texte complet $T$. La phrase la plus proche est celle qui maximise cette similarité :

$$P^* = \arg\max_{P_i} \text{similarité\_cosinus}(P_i, T) \tag{2}$$

où $P^*$ est la phrase ayant la plus grande similarité cosinus avec le texte.

### 3.7.2 Similarité cosinus entre chaque phrase entre elles

La deuxième méthode se concentre sur les relations entre les phrases du texte pour identifier la plus représentative. Elle repose sur la comparaison des embeddings associés à chaque phrase avec ceux de toutes les autres phrases du document. Ces embeddings, obtenus grâce à l'utilisation de RoBERTa, servent à calculer les similarités cosinus, permettant ainsi de mesurer la proximité sémantique entre chaque paire de phrases. L'objectif est de déterminer la centralité de chaque phrase, c'est-à-dire d'évaluer dans quelle mesure une phrase est représentative des autres. La phrase considérée comme la plus centrale, car présentant la plus grande similitude globale avec les autres, est ensuite sélectionnée comme résumé. Cette méthode tire parti de la structure interne du texte en identifiant les phrases qui sont les plus connectées aux autres, ce qui les rend représentatives des idées principales du document.

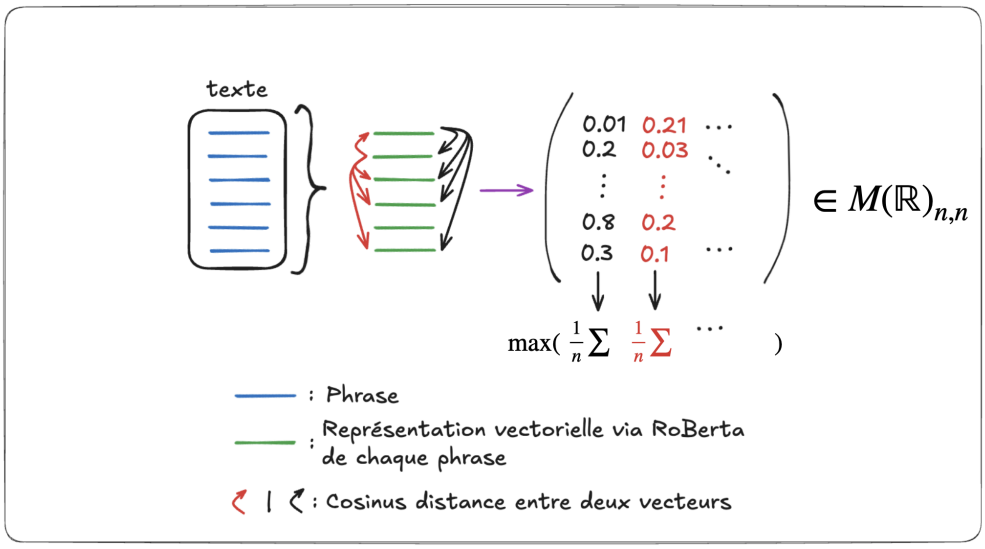



FIGURE 10 – Schéma de la deuxième méthode de RoBERTa

#### Représentation des Embeddings

Soit $P_i$ et $P_j$ deux phrases du document. Les embeddings de ces phrases sont notés $\mathbf{v}_{P_i}$ et $\mathbf{v}_{P_j}$, obtenus grâce à RoBERTa.

**Calcul de la Similarité Cosinus**

La similarité cosinus entre les phrases $P_i$ et $P_j$ est calculée comme suit :

$$\text{similarité\_cosinus}(P_i, P_j) = \frac{\mathbf{v}_{P_i} \cdot \mathbf{v}_{P_j}}{\|\mathbf{v}_{P_i}\| \|\mathbf{v}_{P_j}\|} \tag{3}$$

où :
- $\mathbf{v}_{P_i} \cdot \mathbf{v}_{P_j}$ est le produit scalaire des vecteurs des phrases.
- $\|\mathbf{v}_{P_i}\|$ et $\|\mathbf{v}_{P_j}\|$ sont les normes des vecteurs $\mathbf{v}_{P_i}$ et $\mathbf{v}_{P_j}$, respectivement.

**Détermination de la Centralité des Phrases**

Pour chaque phrase $P_i$, nous calculons la somme des similarités cosinus avec toutes les autres phrases $P_j$ du document :

$$C(P_i) = \sum_{j \neq i} \text{similarité\_cosinus}(P_i, P_j) \tag{4}$$

où $C(P_i)$ représente la centralité de la phrase $P_i$.

**Sélection de la Phrase la Plus Centrale**

La phrase la plus centrale, qui est la plus représentative des autres, est déterminée par :

$$P^* = \arg\max_{P_i} C(P_i) \tag{5}$$

où $P^*$ est la phrase ayant la plus grande centralité.

### 3.7.3 Approche hybride : combinaison des deux méthodes précédentes

La méthode hybride combine les avantages des deux approches précédentes pour améliorer la qualité de la phrase résumant le texte. Dans un premier temps, les phrases les plus pertinentes sont présélectionnées à l'aide de la première méthode, qui évalue la similarité cosinus entre chaque phrase et le texte entier. Cette étape permet d'isoler un ensemble restreint de phrases candidates (les top N) qui reflètent le mieux le contenu global. Ensuite, la deuxième méthode est appliquée à cet ensemble pour identifier la phrase la plus centrale, c'est-à-dire celle qui est la plus connectée aux autres phrases du sous-ensemble. En combinant ces deux étapes, cette méthode optimise à la fois la sélection des phrases pertinentes et leur cohérence avec le reste du document.

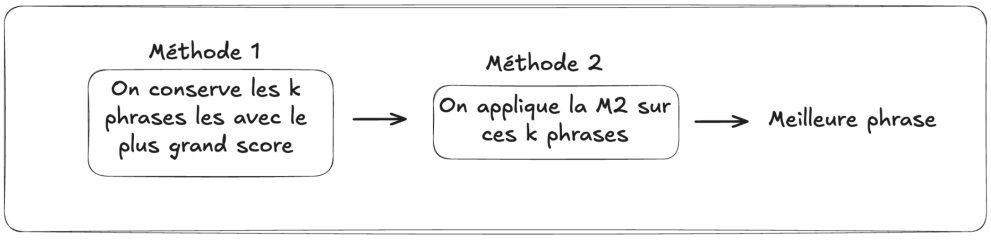



FIGURE 11 – Schéma de la méthode hybride de RoBERTa

**Étape 1 : Sélection des Phrases Pertinentes**

Dans un premier temps, nous appliquons la première méthode, qui évalue la similarité cosinus entre chaque phrase $P_i$ et le texte complet $T$. Cela se fait comme suit :

$$\text{similarité\_cosinus}(P_i, T) = \frac{\mathbf{v}_{P_i} \cdot \mathbf{v}_T}{\|\mathbf{v}_{P_i}\| \|\mathbf{v}_T\|} \tag{6}$$

Nous sélectionnons ensuite les $N$ phrases les plus pertinentes, notées $\{P_1, P_2, \ldots, P_N\}$, qui maximisent la similarité cosinus :

$$\{P_1, P_2, \ldots, P_N\} = \arg\max_{P_i} \text{similarité\_cosinus}(P_i, T) \tag{7}$$

**Étape 2 : Identification de la Phrase la Plus Centrale**

Ensuite, nous appliquons la deuxième méthode à l'ensemble restreint de phrases candidates $\{P_1, P_2, \ldots, P_N\}$. Pour chaque phrase $P_i$ dans cet ensemble, nous calculons la centralité :

$$C(P_i) = \sum_{j \neq i} \text{similarité\_cosinus}(P_i, P_j) \quad \text{pour } P_j \in \{P_1, P_2, \ldots, P_N\} \tag{8}$$

La phrase la plus centrale, qui est la plus connectée aux autres phrases du sous-ensemble, est déterminée par :

$$P^* = \arg \max_{P_i \in \{P_1, P_2, \ldots, P_N\}} C(P_i) \tag{9}$$

où $P^*$ est la phrase sélectionnée comme résumé.

# 4 Evaluation et Résultat

## 4.1 Score BERT

Le *Score BERT* [Zhang et al.(2020)] est une méthode non supervisée utilisée pour mesurer la similarité entre des textes en exploitant les modèles d'*embeddings* contextuels tels que BERT (*Bidirectional Encoder Representations from Transformers*). Contrairement aux approches classiques basées sur la distance de mots ou de phrases, cette méthode utilise les représentations contextuelles fournies par BERT pour capturer des nuances sémantiques et contextuelles.

### 4.1.1 Principe de fonctionnement

Pour calculer le Score BERT entre une référence $R$ et un candidat $C$, la démarche se décompose en plusieurs étapes :

1. **Génération des vecteurs d'*embedding*** : Chaque phrase est passée dans le modèle BERT pour produire des vecteurs d'*embedding* contextuels pour chacun des mots. Soit :
$$\mathbf{E}_R = [\mathbf{e}_{R,1}, \mathbf{e}_{R,2}, \ldots, \mathbf{e}_{R,n}], \quad \mathbf{E}_C = [\mathbf{e}_{C,1}, \mathbf{e}_{C,2}, \ldots, \mathbf{e}_{C,m}],$$
les vecteurs d'*embedding* des mots de $R$ et $C$ respectivement.
2. **Calcul de la matrice de similarité** : Une matrice de similarité $\mathbf{S} \in \mathbb{R}^{n \times m}$ est calculée entre les vecteurs d'*embedding* de $R$ et $C$ en utilisant la similarité cosinus :
$$\mathbf{S}_{i,j} = \frac{\mathbf{e}_{R,i} \cdot \mathbf{e}_{C,j}}{\|\mathbf{e}_{R,i}\| \|\mathbf{e}_{C,j}\|}, \quad \forall i \in [1, n], \quad \forall j \in [1, m].$$
3. **Alignement optimal et calcul du score** : Pour chaque mot de la référence $R$, on identifie le mot du candidat $C$ ayant la similarité maximale. L'alignement optimal est pondéré par les poids IDF (*Inverse Document Frequency*) pour refléter l'importance relative des mots. Soit $idf(w_i)$ le poids IDF du mot $w_i$ de $R$, le Score BERT est donné par :
$$R_{\text{BERT}} = \frac{\sum_{i=1}^{n} [\max_j \mathbf{S}_{i,j}] \cdot idf(w_i)}{\sum_{i=1}^{n} idf(w_i)}.$$

### 4.1.2 Cas d'utilisation

Le Score BERT peut être utilisé pour comparer différents types de textes :

— **Phrase contre phrase** : Par exemple, la comparaison entre *« The weather is cold today »* et *« It is freezing today »*. Chaque mot est comparé individuellement en utilisant la similarité cosinus entre les vecteurs d'*embedding* correspondants, ce qui permet de construire la matrice de similarité $\mathbf{S}$. Le score final est calculé en identifiant l'alignement optimal des mots entre les deux phrases.

— **Phrase contre texte** : Deux approches sont possibles :

  — **Considérer le texte comme une seule phrase** : On traite le texte entier comme une phrase longue et on applique la méthode de *Phrase contre Phrase*.

  — **Comparer la phrase à chaque phrase du texte** : La phrase candidate est comparée individuellement à chaque phrase du texte. Pour chaque paire *(phrase candidate, phrase du texte)*, un Score BERT est calculé. Cela génère un vecteur de scores, à partir duquel le score global peut être obtenu via différentes méthodes d'agrégation :

    — **Moyenne arithmétique** : Calculer la moyenne simple des scores individuels. Cette méthode est rapide mais peut être influencée par des phrases contenant des mots courants ou des répétitions, car elle considère que toutes les phrases ont le même poids sémantique.

    — **Moyenne pondérée par IDF** : Pondérer chaque score individuel par l'IDF de la phrase 4.1.5 correspondante. Ainsi, les phrases contenant des mots moins courants auront un poids plus élevé, ce qui permet de mieux capturer la spécificité des informations et d'ignorer les phrases à faible valeur sémantique.

    — **Maximum des scores** : Sélectionner le score le plus élevé parmi tous les scores individuels. Cela est pertinent lorsque l'on cherche à identifier la phrase du texte la plus similaire à la phrase d'entrée.

  Ce type de comparaison est utile pour évaluer dans quelle mesure une phrase est présente ou évoquée dans un texte plus long, comme dans les tâches de recherche d'information ou d'analyse de documents.

— **Texte contre texte** : Deux méthodes principales peuvent être appliquées :

  — **Considérer les textes comme de longues phrases** : Traiter chaque texte comme une phrase unique et appliquer la méthode de *Phrase contre Phrase*.

— **Comparer chaque phrase du premier texte à chaque phrase du second texte** : Cela génère une matrice de similarité entre les phrases des deux textes. L'agrégation des scores pour obtenir une mesure globale de similarité peut se faire de différentes manières :

  — **Moyenne des alignements optimaux** : Pour chaque phrase du premier texte, on identifie la phrase la plus similaire dans le second texte (en utilisant l'étape 3 du calcul du Score BERT). Puis on applique une moyenne sur (éventuellement pondérée par l'IDF des phrases) les scores obtenus.
  — **Moyenne sur la matrice de similarité** : Calculer la moyenne des scores sur les lignes de la matrice de similarité, puis appliquer une moyenne (éventuellement pondérée) sur l'IDF des phrases du texte de référence.

La comparaison *Texte contre Texte* est couramment utilisée dans des tâches telles que la détection de plagiat, la comparaison de documents ou la recherche de correspondances entre des résumés et des textes sources.

### 4.1.3 Illustration

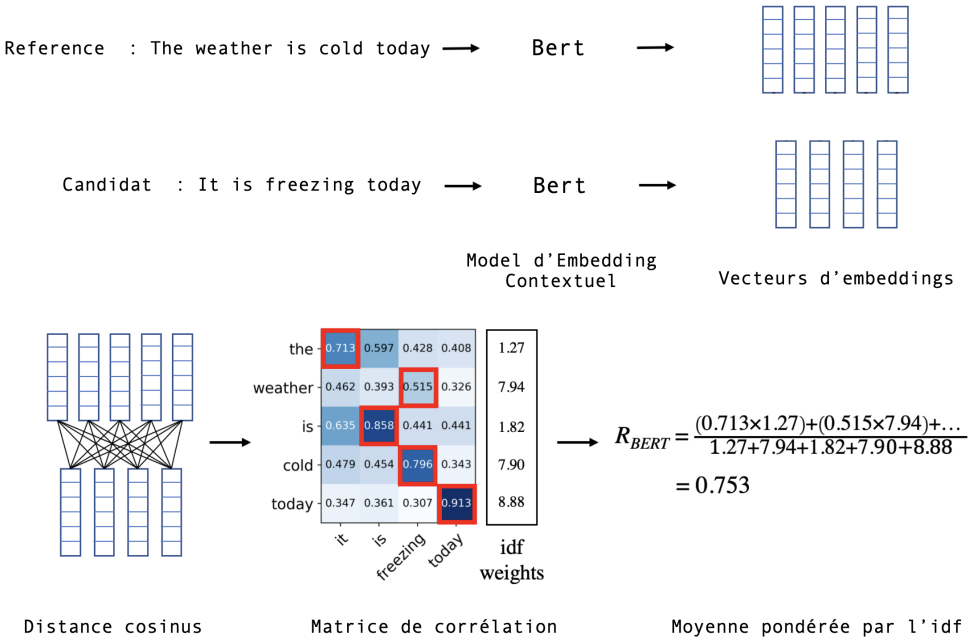



FIGURE 12 – Calcul des embeddings et similarité entre référence et candidat.

La Figure 12 illustre le processus complet du calcul du Score BERT, depuis la génération des vecteurs d'*embedding* jusqu'à l'obtention du score final à l'aide de la pondération par IDF.

### 4.1.4 Avantages et limites

Le Score BERT présente plusieurs avantages et limitations :

— **Avantages** :
  — **Capture des similarités sémantiques fines** : Grâce aux embeddings contextuels de BERT, le Score BERT est capable de saisir des nuances sémantiques même en présence de variations lexicales ou syntaxiques.
  — **Flexibilité** : Peut être appliqué à différents niveaux de granularité (mot, phrase, texte) et adapté à divers cas d'utilisation.
— **Limites** :
  — **Complexité computationnelle** : Les comparaisons *Phrase contre Texte* et *Texte contre Texte* sont coûteuses en termes de calcul, surtout pour des textes longs, en raison de la nécessité de comparer chaque phrase individuellement. Cela peut rendre l'approche impraticable pour des documents volumineux sans optimisation appropriée.

— **Sensibilité au contexte** : Bien que BERT capture le contexte des mots, il peut avoir des difficultés avec des phrases très ambiguës ou lorsque les phrases comparées sont sémantiquement éloignées. L'utilisation des poids IDF réduit cette limitation, mais ne l'élimine pas complètement.
— **Alignement incorrect** : Dans le cas de textes très différents, le processus d'alignement peut associer des phrases qui ne sont pas réellement similaires, réduisant la fiabilité du score global. Des approches alternatives, telles que l'utilisation de modèles *cross-encoder*, peuvent améliorer la qualité de l'alignement, mais au prix d'une complexité accrue.

#### 4.1.5 Calcul de l'IDF d'une phrase

Pour obtenir l'IDF d'une phrase, on peut utiliser différentes méthodes d'agrégation des IDF des mots qui la composent. Les méthodes les plus courantes sont :

— **Moyenne arithmétique des IDF des mots** :

$$idf(\text{phrase}) = \frac{1}{n} \sum_{i=1}^{n} idf(w_i),$$

où $n$ est le nombre de mots dans la phrase, et $idf(w_i)$ est l'IDF du mot $w_i$.
— **Somme des IDF des mots** :

$$idf(\text{phrase}) = \sum_{i=1}^{n} idf(w_i).$$

Cette méthode accentue davantage l'importance des phrases contenant plusieurs mots rares.
— **Maximum des IDF des mots** :

$$idf(\text{phrase}) = \max_i \{idf(w_i)\}.$$

Ici, l'IDF de la phrase est déterminé par le mot le plus rare qu'elle contient.
— **Médiane des IDF des mots** :

$$idf(\text{phrase}) = \text{médiane}\left(\{idf(w_1), idf(w_2), \ldots, idf(w_n)\}\right).$$

Cette approche réduit l'influence des valeurs extrêmes.
— **Moyenne pondérée** : On peut également pondérer les IDF des mots en fonction de leur importance syntaxique ou de leur position dans la phrase.

La méthode choisie dépend du contexte et des objectifs de l'analyse. En agrégeant les IDF des mots pour obtenir l'IDF d'une phrase, on cherche à estimer l'importance ou la spécificité de cette phrase dans le corpus. Les phrases contenant des mots rares (avec des IDF élevés) auront un IDF de phrase plus élevé, indiquant qu'elles apportent des informations plus spécifiques ou moins communes.

Dans le contexte du Score BERT, l'utilisation de l'IDF de la phrase permet de pondérer les scores de similarité entre phrases en fonction de leur importance. Ainsi, les phrases plus informatives (avec un IDF élevé) contribuent davantage au score global que les phrases composées principalement de mots fréquents.

### 4.2 Résultats

Puisque notre dataset n'est pas conçu pour faire des résumés de documents, nous avons rencontré un problème : nous ne savions pas comment déterminer avec exactitude si un document était le meilleur pour faire un résumé ou non. Pour pouvoir évaluer nos différentes solutions, nous avons utilisé la métrique `score_bert` de la section ci-desssus en calculant le score `score_bert` entre le document original et la phrase sélectionnée par chaque méthode pour résumer le document. Pour éviter d'avoir des résultats subjectifs, nous avons utilisé les rangs de chaque méthode en les classant par ordre de F-mesure décroissante. Pour les six méthodes, la première est celle qui a la F-mesure la plus élevée, la seconde est celle qui a la deuxième F-mesure la plus élevée, et ainsi de suite. Les figures 13 et 15 présentent la distribution des rangs des différentes méthodes sur les 570 documents de notre corpus.

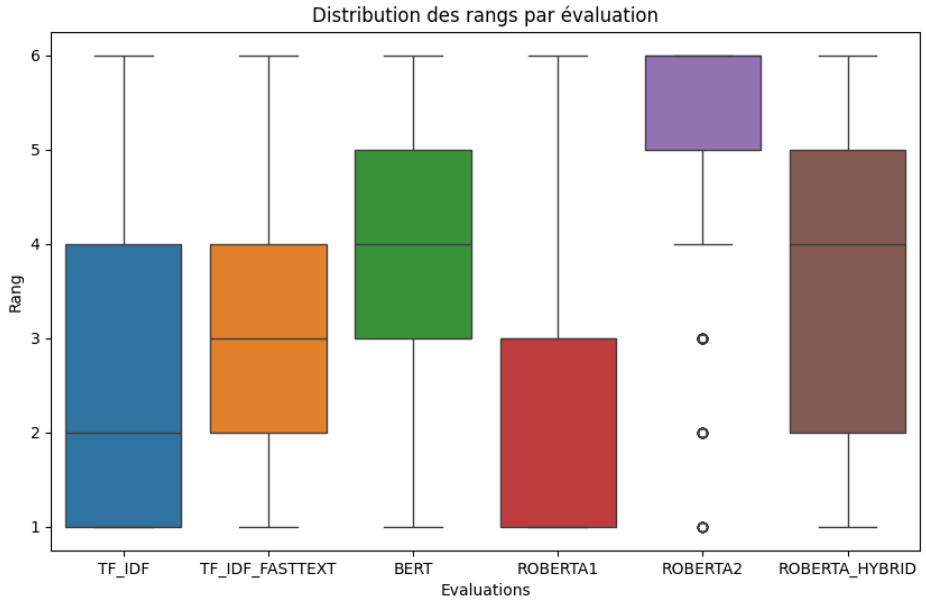



FIGURE 13 – Distribution de rangs

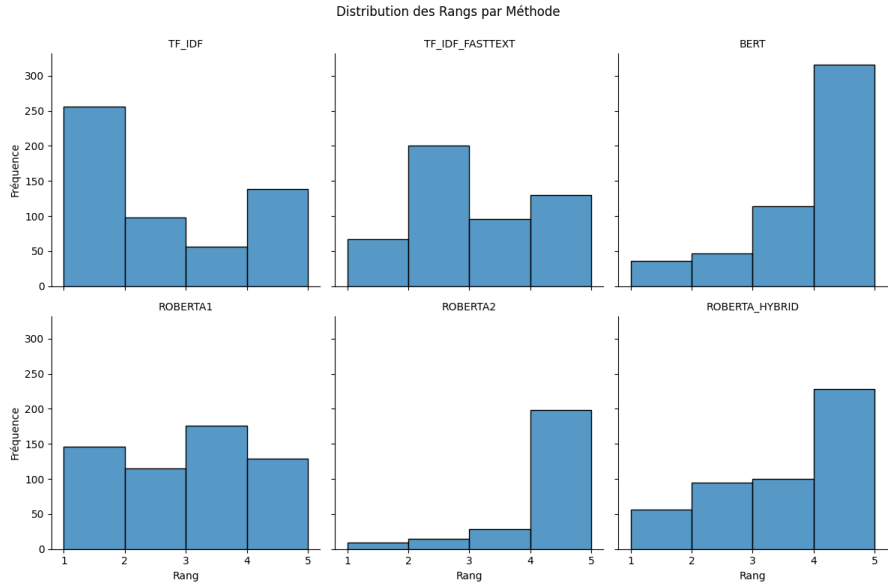



FIGURE 14 – Histogramme de rangs

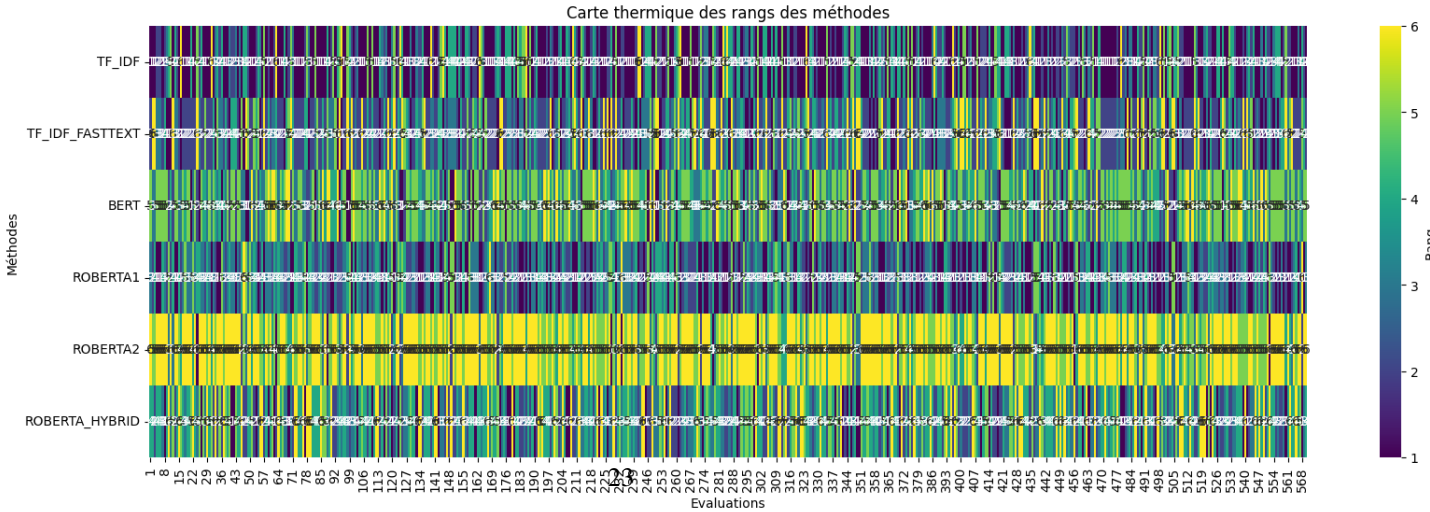



FIGURE 15 – Carte thermique de rangs

L'analyse de ses résultats montre que la méthode TF-IDF présente de meilleurs résultats que l'ensemble des autres méthodes. Elle est suivie de TF-IDF-FASTTEXT, ROBERTA1, ROBERTA-HYBRIDE et BERT.

TABLE 1 – Overall performance (Precision, Rappel, F-mesure, Rang)

| | **Pre** | **Racall** | **F-mesure** | **Rang** |
|---|---|---|---|---|
| *TF-IDF* | 0.59 | 0.703 | 0.63 | 2.36 |
| Clustering+*TF-IDF* | **0.68** | **0.62** | **0.754** | **2.6** |
| Bert | 0.471 | 0.62 | 0.53 | 3.12 |
| Roberta1 | *0.65* | *0.74* | *0.69* | *3.62* |
| Roberta2 | **0.43** | **0.60** | **0.49** | **4.02** |
| Roberta_hybrid | **0.54** | **0.64** | **0.58** | **5.24** |

La table 1 presente la précision, le rappel, la F-mesure et le rang moyen sur les 570 documents. Nous pouvons constater que la méthode TF-IDF à des meilleurs résultats par rapport aux autres méthodes,

#### 4.2.1 Commentaire

- Représentation des mots avec TF-IDF : La méthode TF-IDF donne des très bon résultats car elle se concentre sur les termes les plus fréquents et distinctifs d'un document. Cette approche est robuste pour les résumés, car elle permet d'identifier rapidement les mots-clés importants qui capturent le contenu essentiel, même dans des documents plus simples ou plus courts. Cette approche est plus sensible aux mots clés car Les résumés basés sur TF-IDF peuvent mieux se concentrer sur les mots qui sont véritablement représentatifs du sujet du document, ce qui peut conduire à des résumés plus clairs et plus pertinents.
- TF-IDF et FastText Combinaison des Forces : Bien que la méthode TF-IDF + FastText prenne en compte les relations sémantiques entre les mots, la qualité de l'embedding dépend fortement des clusters formés et de la manière dont les mots sont regroupés. Si les clusters ne capturent pas bien les nuances de signification, cela peut limiter la performance. Bien que FastText est conçu pour capturer des informations sémantiques, si les clusters sont mal définis ou si le document contient des termes spécifiques qui ne sont pas bien représentés par les embeddings, cela peut nuire à la qualité du résumé ce qui explique les résultats.
- Modèles Basés sur BERT BERT et RoBERTa prennent en compte le contexte, mais cela peut parfois mener à des résumés qui manquent de clarté ou qui incluent des informations superflues, car ils essaient de capturer des relations plus complexes qui ne sont pas toujours utiles dans le cadre d'un résumé.

## Conclusion

Notre objectif dans ce rapport était de proposer des solutions pour, d'une part, déterminer les mots-clés dans un corpus de documents et, d'autre part, identifier la phrase qui résume au mieux chaque document. À cet effet, nous avons exploré plusieurs approches. Pour l'extraction des mots-clés, nous avons proposé une méthode basée sur les poids TF-IDF de chaque mot dans les phrases, ainsi qu'une approche reposant sur les graphes de mots.

Concernant la génération de résumés, nous avons proposé différentes stratégies : une approche exploitant les mots-clés extraits en utilisant les poids TF-IDF, une méthode basée sur la sémantique des mots via les embeddings FastText et leur regroupement sémantique, une approche exploitant BERT pour générer des embeddings des phrases et du document afin d'identifier la phrase la plus proche de la représentation globale du document, et enfin, plusieurs variantes utilisant Roberta.

L'expérimentation de ces différentes approches sur notre jeu de données portant sur l'éthique de l'intelligence artificielle a permis de mettre en évidence plusieurs points. Tout d'abord, l'approche basée sur la méthode TF-IDF s'est révélée très performante pour identifier les mots clés et extraire les phrases résumant les documents. Ensuite, la méthode combinant TF-IDF et les embeddings sémantiques a également produit des résultats significatifs, en permettant de trouver des phrases résumant le document, même lorsque les mots de ces phrases sont des synonymes des termes clés. Enfin, la méthode Roberta a montré d'excellentes performances, confirmant son efficacité pour cette tâche complexe.

Cependant, une perspective d'amélioration pourrait consister à explorer l'utilisation de modèles d'IA générative pour produire des résumés. Cette approche permettrait de générer non seulement des résumés basés sur des phrases existantes dans les documents, mais également de créer de nouvelles phrases qui synthétisent et résument le contenu de manière plus approfondie. Cela ouvrirait la voie à des résumés plus variés, adaptatifs et mieux adaptés à des contextes spécifiques, enrichissant ainsi les applications potentielles dans ce domaine.

## Références


[Baeza-Yates and Ribeiro-Neto(1999)] Ricardo Baeza-Yates and Bernardo Ribeiro-Neto. 1999. Modern Information Retrieval : The Concepts and Technology Behind Search. In *Addison-Wesley*. Includes a section on TF-IDF.

[Devlin et al.(2019)] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv :1810.04805 [cs.CL] `https://arxiv.org/abs/1810.04805`

[Face(2020)] Hugging Face. 2020. RoBERTa. `https://huggingface.co/transformers/model_doc/roberta.html`

[Fragkiskos D. Malliaros(2017)] Michalis Vazirgiannis Fragkiskos D. Malliaros. 2017. Graph-based Text Representations : Boosting Text Mining, NLP and Information Retrieval with Graphs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Copenhagen, Denmark, Numéro des pages.

[Gupta and Grossman(2004)] Chetan Gupta and Robert Grossman. 2004. Genic : A single pass generalized incremental algorithm for clustering. In *Proceedings of the 2004 SIAM International Conference on Data Mining*. SIAM.

[Liu et al.(2019)] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa : A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv :1907.11692* (2019).

[Salton and McGill(1988)] Gerard Salton and Michael J. McGill. 1988. Introduction to Modern Information Retrieval. *McGraw-Hill* (1988). Chapters on TF-IDF.

[Vaswani et al.(2023)] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv :1706.03762 [cs.CL] `https://arxiv.org/abs/1706.03762`

[Zhang et al.(2015)] Jie Zhang, Jie Zhou, and Zhi Wang. 2015. TF-IDF : A Comprehensive Review. *International Journal of Computer Science and Information Security* 13, 5 (2015), 123–128.

[Zhang et al.(2020)] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore : Evaluating Text Generation with BERT. arXiv :1904.09675 [cs.CL] `https://arxiv.org/abs/1904.09675`