

Analyse de textes en utilisant les bibliothèques du langage Python

(Calcul de similarité entre deux documents)

Réalisé par :
ZEFFATE NAJOUA

Encadrée par :
Pr.A.MOULOUDI

Remerciement

Nous souhaitons remercier dans un premier temps notre professeur encadrant monsieur MOULOUDI Aziz pour le temps qu'il nous a consacré tout au long de cette période.

Notre remerciement va aussi à tous le corps pédagogique de la faculté des sciences de Kenitra, nos professeurs, et tous les responsables de la filière Sciences Mathématiques et Informatiques.

Nous tenons à remercier tout particulièrement et à témoigner toute mes reconnaissances à nos familles qui nous ont soutenues, tout au long du chemin physiquement et moralement, sans aucune plainte ou aucun regret.

Table de matière

Remerciement.....	1
Liste des figures.....	3
Introduction.....	4
Chapitre I : Introduction à l'analyse de textes.....	5
1. Pourquoi l'analyse de texte ?.....	6
2. Traitement automatique du langage naturel.....	6
3. Python & Analyse de textes.....	8
3.1.Les merveilles des librairies du langage Python.....	9
3.2.Bibliothèque NLTK.....	9
3.3.Bibliothèque SPACY.....	10
3.4.Bibliothèque Numpy.....	10
Chapitre II : Etapes/Méthodes d'analyse de textes.....	11
1. Méthodes de pré traitement de données textuelles.....	12
1.1. Tokenisation.....	12
1.2. Stemming.....	12
1.3. Lemmatisation.....	13
1.4. Stopwords(Mots vides).....	14
2. Implémentation en Python.....	15
3. Méthode d'analyse de texte.....	16
3.1. Similarité syntaxique.....	17
3.2. Modèle d'espace vectoriel.....	17
3.2.1. Extraction des termes pertinents (lemmatisation).....	18
3.2.2. Calcul de poids.....	18
4. Mesures de similarité existante.....	21
4.1. Similarité cosinus.....	21
4.2. Distance Euclidienne	22
5. Analyse Sémantique.....	23
5.1. Approche vectorielle.....	23
5.2. Approche Statistique.....	24
Chapitre III : Mise en place d'une application pour l'analyse de textes.....	25
1. Objectif.....	26
2. Algorithme de test de similarité.....	26
3. Implémentation en Python.....	27
4. Interface graphique.....	33
5. Outils.....	34
Conclusion.....	35
Bibliographie.....	36

Liste des figures

Figure 1 : les librairies les plus utilisées en NLP	9
Figure 2 : importation des librairies nécessaires	10
Figure 3 : Récupération/Lecture des documents à tester	10
Figure 4 : Passer les deux documents en minuscules.....	10
Figure 5 : Chargement du model français	10
Figure 6 : Récupération/Ajout des mots vides	10
Figure 7 : lemmatisation	10
Figure 8 : Affichage des listes de mots lemmatisés de chaque document.....	10
Figure 9 : nettoyage des deux documents des mots vides	10
Figure 10 : Retourner les nouveaux documents après nettoyage.....	10
Figure 11 : Résultat de nettoyage.....	10
Figure 12 : tokenization	10
Figure 13 : Résultat de tokenisation	10
Figure 14 : Retourner les nouveaux documents après prétraitement	10
Figure 15 : Union des deux documents.....	10
Figure 16 : création de dictionnaire de mots et leur occurrence pour chaque document.....	10
Figure 17 : transformation des résultats sous format trames de données	10
Figure 18 : affichage des deux vecteurs des deux documents	10
Figure 19 : Calcul de la similarité	10
Figure 20 : fenêtre2	10
Figure 21 : fenêtre1	10

Introduction

Depuis les débuts de l'informatique, l'homme cherche à communiquer avec les machines. Si les nombreux langages de programmation permettent une forme d'échange entre l'homme et la machine, on aimerait que cette communication se fasse de façon plus naturelle. Pour que cela soit possible, il faut d'abord que la machine "comprenne" ce que l'utilisateur lui dit puis qu'elle soit capable de répondre d'une manière compréhensible par l'homme.

La discipline dernière ce processus s'appelle le Natural Language Processing (NLP) ou Traitement Automatique du Langage Naturel (TALN) en français. Elle étudie la compréhension, la manipulation et la génération du langage naturel par les machines. Par langage naturel, on entend le langage utilisé par les humains dans leur communication de tous les jours par opposition aux langages artificiels comme les langages de programmation ou les notations mathématiques.

De nos jours le domaine de recherche est passé brusquement d'une situation de pénurie de données ou, du moins, d'une situation où les données étaient difficiles à assembler, à une situation où les données sont apparemment massives, disponibles et faciles d'accès. Cette masse de données n'est pas sans poser problème : la plupart du temps, les données ne sont pas directement exploitables, elles doivent être triées, filtrées, organisées ; elles reflètent des points de vue particuliers qui ne sont pas obligatoirement ceux visés par le chercheur. Nous garderons ces difficultés en tête lors de cette étude, mais nous nous focaliserons surtout sur le cas des données textuelles : celles-ci constituent une source incomparable de connaissances, mais offrent dans le même temps les plus grandes difficultés d'accès.

En effet, comme chacun le sait, un texte ne saurait être assimilé à une masse de connaissances directement exploitable par la machine. Il faut dans un premier temps prévoir des traitements complexes pour identifier l'information pertinente, la normaliser, la catégoriser et éventuellement la mettre en contexte. Alors seulement l'ordinateur ou l'expert sera capable d'en tirer parti pour mener à bien ses analyses. Mais comment procéder pour extraire l'information pertinente de la masse textuelle ? Quels outils utiliser ? Pour quelle pertinence ?

Chapitre I : Introduction à l'analyse de textes

1. Pourquoi l'analyse de texte ?

Big Data et « Content Analytics » Analyse de contenu en français, sont sur toutes les bouches comme si une nouvelle révolution était en marche. Une réelle évolution sans doute, du fait des réseaux sociaux et des nouvelles applications qui permettent à tout un chacun de participer à la construction des contenus. Chacun est en mesure de dire ce qu'il pense sur Internet, de dire ce qu'il « aime », de dire où il se trouve et ce qu'il fait.

Toute cette information constituée par des millions d'internautes représente un volume de contenu colossal disponible et accessible sur la toile. Au-delà de la capacité à gérer ces volumes gigantesques et à pouvoir les interroger même s'ils sont disparates, le véritable enjeu est celui de l'analyse de ces masses d'informations structurées (données relationnelles) et non structurées (Word, PDF, texte, logs). Le but avoué ou inavoué de cette analyse des contenus consiste à permettre une compréhension et une synthèse comportementale et temporelle fournie par l'analyse des tendances et l'analyse de la pensée. La difficulté est grande. Le chemin est semé d'embûches.

Avant de pouvoir analyser les contenus, il faut en faire l'acquisition et collecter les informations où elles se trouvent. Il faut extraire de cette masse d'information souvent non structurée, une forme compréhensible et intelligible d'information. L'analyse syntaxique, sémantique et morphologique des contenus et la réconciliation est alors possible en prenant en compte l'identification des exceptions. Le résultat de ce travail ne serait pas audible sans la mise en œuvre de systèmes de représentation qui permettent une navigation progressive et multidimensionnelle dans la connaissance extraite.

Sans la mise en œuvre des technologies d'analyse des contenus sur les volumes produits par Internet, ce but reste inaccessible. La quantité d'information à traiter est trop vaste pour être synthétisée par une organisation. Seuls des processus informatiques dont la puissance de calcul ne cessent de s'accroître, sont capables d'orchestrer la production d'indicateurs renvoyés par des innombrables nœuds intelligents d'analyse et de traitement des contenus.

La transformation des données en contenu, du contenu en indicateurs, d'indicateurs en tendances, de tendances en situations, de situations en décision, représente l'enjeu du marché du Content Analytics. Un enjeu essentiel pour aider à comprendre l'humanité d'aujourd'hui et celle de demain, à l'aider dans les décisions qu'elle devra prendre, en lui fournissant des outils de synthèse encore jamais mis à la disposition de l'Homme.

2. Traitement automatique du langage naturel

Le Natural Language Processing, ou Natural Language Programming (abrégié NLP) peut être traduit en français par *traitement automatique du langage naturel*. Il s'agit d'une technologie dont l'objectif est la communication entre l'ordinateur et l'être humain sur un même niveau. Le NLP allie les connaissances issues de la linguistique et les méthodes les plus récentes de l'informatique et de l'intelligence artificielle.

➤ Historique :

Le développement du NLP débute dans les années 1950 à l'époque où le scientifique Alan Turing publie un article titré *Computing Machinery and Intelligence*. Il y présente une méthode de mesure de l'intelligence artificielle. Le "Turing Test" existe encore de nos jours.

Les chercheurs avaient déjà réussi dès 1954 à traduire à l'aide d'une machine une soixantaine de phrases du russe à l'anglais. Dans l'euphorie des débuts, beaucoup de chercheurs informatiques pensaient que la traduction automatique n'était plus qu'une question de temps. Il a cependant fallu attendre jusque dans les années 1980 pour voir se développer les premiers systèmes de traduction automatique basés sur les statistiques. Entre temps, quelques idées furent trouvées pour traduire les informations du monde "réel" en langue informatique.

Un grand pas fut fait à la fin des années 1980, à l'époque où l'apprentissage automatique est devenu populaire. Les algorithmes, alliés à la puissance informatique de plus en plus importante des ordinateurs, purent alors être utilisés pour le NLP. L'un des pionniers de ce domaine fut et est encore aujourd'hui le linguiste Noam Chomsky. L'entreprise de logiciels IBM a elle aussi joué un rôle dans le développement du Natural Language Processing.

Les programmes informatiques basés sur le NLP ne sont aujourd'hui plus en mesure de profiter des fichiers de données collectés manuellement, mais sont cependant capables d'analyser directement et tout seuls des corpus de textes comme des sites web ou de la langue parlée.

➤ Conditions :

Le NLP est fondé sur l'idée de base que chaque forme de langue, parlée ou écrite, doit tout d'abord être reconnue. La langue est cependant un système très complexe de signes. Ce qui importe n'est pas seulement le mot en lui-même mais le rapport qui existe avec d'autres mots, des phrases entières ou encore les faits.

Pour comprendre ce que les êtres humains apprennent naturellement dès la naissance, les ordinateurs doivent utiliser des algorithmes. Si l'Homme peut tirer des réponses de son expérience de vie, l'ordinateur doit lui avoir recours à des expériences créées de manière artificielle. Le défi pour le traitement automatisé du langage naturel est finalement de comprendre la langue plus que de la produire.

➤ Fonctionnalité :

Le NLP moderne se base sur des algorithmes fondés eux-mêmes sur des **statistiques** de la **machine learning**. La particularité ici est que les ordinateurs, de cette manière, sont non seulement capables d'apprendre des problèmes résolus autrefois, mais également de les reconnaître de façon autonome et de résoudre des secteurs problématiques en se basant sur d'importants corpus de documents. Les ordinateurs n'apprennent pas de cette manière à trouver une solution pour chaque problème mais plutôt des tendances générales grâce

auxquelles ils pourront traiter des questions individuelles. Cela fait du NLP un précurseur de l'intelligence artificielle.

➤ **Exemple d'application :**

L'avantage considérable de cette méthode est le fait que plus les ordinateurs obtiennent des informations, plus ils sont performants. Un bon exemple est la fonction de traduction de Google. Le projet ne fut pas pris au sérieux à ses débuts. Aujourd'hui, le programme est capable de traduire un grand nombre de textes différents, et même de traduire le mot parlé de manière relativement fluide.

Le "Rank Brain" de Google utilise également la méthode de NLP pour fournir des résultats de recherche correspondants à des requêtes de recherche sans précédent. L'"interprétation" des données est complétée par l'intelligence artificielle.

Les programmes informatiques qui se basent sur le NLP doivent pouvoir remplir les tâches suivantes :

- ✓ Racinisation
- ✓ Simplification de textes
- ✓ Comparaison de textes
- ✓ Convertir du texte en langue parlée
- ✓ Convertir la langue parlée en texte
- ✓ Comprendre la recherche en langue naturelle
- ✓ Reconnaître les questions élargies et celles de suivi
- ✓ Vérifier la plausibilité des réponses

3. Python & Analyse de textes

Python est le langage de programmation open source le plus employé par les informaticiens. Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données ou dans le domaine du développement de logiciels. En effet, parmi ses qualités, Python permet notamment aux développeurs de se concentrer sur ce qu'ils font plutôt que sur la manière dont ils le font. Il a libéré les développeurs des contraintes de formes qui occupaient leur temps avec les langages plus anciens. Ainsi, développer du code avec Python est plus rapide qu'avec d'autres langages.

Python est un langage de programmation libre de droits, populaire et utilisé dans énormément de domaines. Il est utilisé dans le domaine scientifique pour l'analyse de données, et plus généralement la Data Science, mais aussi pour développer des applications web, des jeux, des interfaces graphiques d'outils, ou encore pour faire du développement logiciel en général. Il propose un ensemble complet de bibliothèques dédiées à la Data Science et donc des fonctions facilitant la manipulation de données, les analyses statistiques sur ces données et leur visualisation, notamment.

Son concurrent direct est le langage R, langage dédié à l'analyse statistique pensé par les statisticiens, contrairement à Python qui est un langage très généraliste et applicable à de larges domaines. Ces deux langages, R et Python, sont très utilisés en Data Science, mais depuis quelques années, Python est devenu plus populaire auprès du Data Scientist, en fait, depuis l'apparition des librairies dédiées à la Data Science.

3.1. Les merveilles des librairies du langage Python

Le langage Python (en) est couramment utilisé pour faire du NLP. Il existe plusieurs librairies permettant de travailler avec le langage naturel. Les plus connues et utilisées sont Gensim (en), NLTK (en) et plus récemment SpaCy (en).



Figure 1 : les librairies les plus utilisées en NLP

3.2. NLTK

Le traitement du langage naturel (TLN) est un domaine qui vise à rendre le langage humain naturel utilisable par des programmes informatiques.

Pendant longtemps, NLTK (Natural Language ToolKit) a été la librairie Python standard pour le NLP. Elle regroupe des algorithmes de classifications, de Part-of-Speech Tagging, de stemming, de tokenisation (en) de mots et de phrases. Elle contient également des corpora de données et permet de faire de la reconnaissance d'entités nommées (NER) et de l'analyse de sentiments. Même s'il y a des mises à jour régulières, la librairie NLTK commence un peu à dater (2001) et montre quelques limites notamment en termes de performance.

3.3. Spacy

Une librairie plus récente (2015) semble avoir pris le relais de NLTK, il s'agit de SpaCy. Cette librairie écrite en Python et Cython regroupe les mêmes types d'outils que NLTK : tokenisation, POS-tagging, NER, analyse de sentiments (toujours en développement), lemmatisation. Elle possède également des vecteurs de mots pré-entraînés et des modèles statistiques dans plusieurs langues (anglais, allemand, français et espagnol jusqu'ici).

3.4. Numpy

NumPy est une bibliothèque Python qui fournit une structure de données simple mais puissante : le tableau à n dimensions. C'est sur cette base que repose presque toute la puissance de la boîte à outils de science des données de Python, et l'apprentissage de NumPy est la première étape du parcours de tout scientifique de données Python. Ce didacticiel nous fournira les connaissances dont nous avons besoin pour utiliser NumPy et les bibliothèques de niveau supérieur qui en dépendent.

Voici les quatre principaux avantages que NumPy peut apporter à votre code :

- Plus de vitesse : NumPy utilise des algorithmes écrits en C qui se terminent en nanosecondes plutôt qu'en secondes.
- Moins de boucles : NumPy nous aide à réduire les boucles et à éviter de vous emmêler dans les index d'itération.
- Code plus clair : sans boucles, notre code ressemblera d'avantage aux équations que nous essayons de calculer.
- Meilleure qualité : des milliers de contributeurs s'efforcent de maintenir NumPy rapide, convivial et sans bogue.

En raison de ces avantages, NumPy est la norme de facto pour les tableaux multidimensionnels dans la science des données Python, et la plupart des bibliothèques les plus populaires sont construites dessus. NumPy est un excellent moyen d'établir une base solide à mesure qu'on puisse développer nos connaissances dans des domaines plus spécifiques de la science des données.

Conclusion :

Comme nous avons pu le voir dans ce premier chapitre, le NLP est un champ de recherche en plein développement. Sa grande complexité, liée à l'ambiguïté du langage naturel et à la très grande diversité des langues existantes, en fait un domaine très intéressant qui va continuer à évoluer et à s'améliorer. Cette évolution passe notamment par le développement de Frameworks et l'enrichissement des données disponibles.

Chapitre II : Etapes/Méthodes d'analyse de textes

1. Méthodes de pré traitement de données textuelles :

1.1. Tokenisation :

Le traitement d'un gros morceau de texte n'est généralement pas la meilleure façon de procéder. Comme on dit toujours « **diviser pour mieux régner** ». Le même concept s'applique également aux tâches de NLP.

Lorsque nous avons des textes, nous les séparons en différents jetons et chaque jeton représente un mot. Il sera plus facile de traiter (faire du stemming ou de la lemmatisation) et de filtrer les jetons inutiles (comme les caractères spéciaux ou les mots vides).

➤ Exemple :

Test= « Bouygues a eu une coupure de réseau à Marseille »

Le résultat de tokenisation de la phrase Test est :

['Bouygues', 'a', 'eu', 'une', 'coupure', 'de', 'réseau', 'à', 'Marseille']

1.2. Stemming (racinisation ou désuffixation) :

Dans la morphologie linguistique et la recherche d'information, le *stemming* est le processus qui consiste à réduire les mots infléchis (ou parfois dérivés) à leur souche, leur base ou leur racine – généralement une forme écrite.

En termes plus simples, le *stemming* est le processus qui consiste à réduire un mot à sa « racine ».

Par exemple : **marcher, marches, marchons, marcheur, ...** seront réduits à **march** et ils partageront ainsi la même signification dans le texte.

marche marchait marcherez marché marcheraient



march

Il existe différents algorithmes qui implémentent le *stemming* : Lovins Stemmer, Porter Stemmer, Paice Stemmer, etc. Chacun a sa propre façon de récupérer le *stemma* d'un mot. La plupart de ces algorithmes fonctionnent avec la langue anglaise.

Il existe cependant des algorithmes de *stemming* qui ont été implémentés pour le **Français** et la bibliothèque *NLTK* de Python dispose d'un *Stemmer* en français.

Testons ce *Stemmer* avec une phrase Test:

Test= "Bouygues a eu une coupure de réseau à Marseille"

Après l'avoir passé au **Stemmer** de **NLTK** nous obtenons le résultat suivant :

['bouygu', 'a', 'eu', 'une', 'coupur', 'de', 'réseau', 'à', 'marseil']

En lisant les résultats après l'extraction, nous pouvons clairement voir qu'après avoir réduit les mots, il y en a plusieurs qui n'existent pas dans le dictionnaire Français : **coupure....**

À la raison pour laquelle cela peut arriver, c'est la meilleure réponse que nous avons pu trouver en ligne :

« Il est souvent considéré comme une erreur grossière qu'un algorithme de stemming ne laisse pas un mot réel après avoir enlevé la tige. Mais le but du Stemming est de rassembler les différentes formes d'un mot, et non de faire correspondre un mot à sa forme paradigmatique. »

En résumé, le *Stemming* sert donc à regrouper de manière "*brute*" plusieurs mots partageant le même sens en enlevant le genre, le nombre, la conjugaison, etc.

Les algorithmes ne sont pas, cependant, parfaits. Ils peuvent marcher pour certains cas et pour d'autres, regrouper des mots ne partageant pas le même sens.


NB : Le stemming n'est pas un concept applicable à toutes les langues. Il n'est pas, par exemple, applicable en chinois. Mais pour les langues du groupe indo-européen un modèle commun de structure des mots émerge. En supposant que les mots sont écrits de gauche à droite, la tige ou la racine d'un mot est à gauche, et zéro ou plusieurs suffixes peuvent être ajoutés à droite.

1.3. Lemmatisation :

En linguistique informatique, la lemmatisation est le processus algorithmique qui consiste à déterminer le lemme d'un mot en fonction de sa signification prévue.

Dans de nombreuses langues, les mots apparaissent sous plusieurs formes infléchies. Par exemple, en français, le verbe **marcher** peut apparaître comme **marchera**, **marché**, **marcheront**, etc. La forme de base « **marcher** », que l'on pourrait trouver dans un dictionnaire, s'appelle le lemme du mot.

Ici, le but principal de la lemmatisation, est de regrouper les différents mots d'un texte qui partagent le même « sens » en un seul mot qui est le lemme sans pour autant créer de « nouveaux » mots comme ce qui se fait dans le cas du stemming.

marche marchait marcherez marché marcheraient

marcher

Cependant, les lemmatizers sont beaucoup plus difficiles à créer parce que nous avons besoin d'un dictionnaire qui contient la plupart des mots dans notre langue et il faut aussi connaître la nature du mot en question : un verbe et un nom sont lemmatisés de façon différente.

Les moteurs de recherche peuvent aussi utiliser la lemmatisation au lieu du stemming, elle fournit dans certains cas des résultats plus précis mais est plus difficile à mettre en œuvre. Comme pour le Stemming, NLTK possède aussi un lemmatizer pour le Français.

1.4. Stopwords (Mots vides) :

En informatique, les mots vides (*stopwords*) sont des mots qui sont filtrés avant ou après le traitement des données en langue naturelle (**NLP**). Bien que le terme « mots vides » désigne généralement les mots les plus courants dans une langue, il n'existe pas de liste universelle unique des mots vides utilisés par tous les outils de traitement du langage naturel.

En effet, en fonction des tâches NLP, certains mots et expressions sont inutiles dans le cadre du travail à réaliser.

Supposons que nous voulions faire un test de similarité simple entre des documents écrits en Français. Pour effectuer cela, nous avons comme idée de compter pour chaque document, les 15 mots les plus fréquents. Si deux documents ont plus de 7 mots en commun dans leurs mots les plus fréquents on supposera qu'ils sont similaires, sinon ils sont différents.

Il s'agit d'un processus assez simple et trivial, la similarité entre des documents nécessite beaucoup plus que ces étapes-là, en guise d'exemple cependant, essayons de faire simple.

Si j'utilise des textes dans leur forme brute sans enlever les *stopwords*, on risquerait de ne se retrouver qu'avec des textes similaires. Pourquoi ?

C'est simple, les 45 à 50 mots les plus fréquents de la langue française représentent en moyenne 50% d'un texte.

Et si on pousse plus loin, les 600 mots les plus fréquents de la langue française représentent 90% d'un texte.

Liste des stopwords de la librairie **spacy** :

{'selon', 'c', 'était', 'parler', 'suffisant', 'siens', 'vous', 'té', 'quelles', 'sinon', 'toute', 'hui', 'ne', 'certes', 'vas', 'jusque', 'ceux-là', 'notamment', 'chez', 'ouverte', 'lequel', 'pense', 'pourquoi', 'certaine', 'son', 'nous', 'vous-mêmes', 'alors', 'deux', 'laquelle', 'quatrième', 'hep', 'o', 'importe', 'je', 'possible', 'environ', 'seul', 'comme', 'désormais', 'desquels', 'te', 'da', 'tres', 'quatrièmement', 'ces', 'specifique', 'specifiques', 'enfin', 'trente', 'â', 'derriere', 'd"', 'differentes', 'quelle', 'mien', 'ainsi', 'suivante', 'ayant', 'en', 'miens', 'ô', 'plusieurs', 'puisque', 'vôtre', 'elles', 'sous', 'sienne', 'quelques', 'et', 'au', 'lui', 'mille',
.....'mo
i-même', 'tels', 'etaient', 'avons', 'dehors', 'personne', 'telle', 'relativement', 'ah', 'dessous', 't', 'ta', 'a', 'étaient', 'hi', 'nul', 'treize', 'on', 'font', 'oust', 'nôtres', 'troisièmement', 'via', 'revoici', 'peu', 'lui-meme', 'jusqu', 'ont', 'sent', 'hem', 'vers', 'qui', 'suffisante', 'autrui', 'partant', 'tend', 'autrement', 'pu', 'à', 'ci'}

NB : La librairie **SPACY** contient **487** stopwords , par contre la bibliothèque **NLTK** contient que **157** stopwords.

Nous avons maintenant constaté l'utilité d'enlever les mots vides qui risqueraient de fausser la similarité.

NB : il est préférable d'utiliser ces listes comme point de départ et d'y ajouter ou supprimer des mots au besoin.

Comme l'auxiliaire avoir à sa troisième forme de singulier «a »n'existe pas dans la liste des mots vides des librairies SPACY et NLTK c'est pour cela il faut l'ajouter à la liste des mots vides.

2. Implémentation en Python

➤ Stemming :

```
#importaion des librairies
import spacy
from spacy.lang.fr.stop_words import STOP_WORDS
from nltk.stem.snowball import SnowballStemmer
#Un des stemmers les plus connus est Le Snowball Stemmer.Ce stemmer est disponible en français
stemmer = SnowballStemmer(language='french')
#phrase de test
test="Marchait marcherait marche marché "
#Récupération des mots vides
mots_vides=set(STOP_WORDS)
#conversion Majuscule/minuscule
test=test.lower()
#chargement du model français
nlp = spacy.load("fr_core_news_sm")
doc = nlp(test)
def return_stem(sentence):
    doc = nlp(sentence)
    return [stemmer.stem(X.text) for X in doc]
#Stemming
Stemming=[]
for mots in return_stem(test):
    if mots not in mots_vides:
        Stemming.append(mots)
#nettoyage de la phrase des mots vides
Clean_MotsVides=[]
for word in Stemming:
    if word not in mots_vides:
        Clean_MotsVides.append(word)
```

➤ Résultat du Stemming :

```
#La nouvelle phrase devient
test=" ".join(Clean_MotsVides)
print(test)
```

```
march march march march
```

➤ Lemmatisation :


```

#importaion des librairies
import spacy
from spacy.lang.fr.stop_words import STOP_WORDS
#chargement du model français
nlp = spacy.load("fr_core_news_sm")
#phrase de test
test="Bouygues a eu une coupure de réseau à Marseille "
#Récupération des mots vides
mots_vides=set(STOP_WORDS)
#conversion Majuscule/minuscule
test=test.lower()
doc = nlp(test)
#Lemmatisation
lemmatisation=[]
for mots in doc:
    lemmatisation.append(mots.lemma_)
#nettoyage de la phrase des mots vides
Clean_MotsVides=[]
for word in lemmatisation:
    if word not in mots_vides:
        Clean_MotsVides.append(word)
#La nouvelle phrase devient
test=" ".join(Clean_MotsVides)
print(test)

bouygue coupure réseau marseille

```

➤ Tokenisation :

```

from nltk import word_tokenize
print(word_tokenize(test))

['bouygue', 'coupure', 'réseau', 'marseille']

```

3. Méthode d'analyse de texte :

Évaluer la similarité entre documents textuels est une des problématiques importantes de plusieurs disciplines comme l'analyse de données textuelles, la recherche d'information ou l'extraction de connaissances à partir de données textuelles (Text Mining). Dans chacun de ces domaines, les similarités sont utilisées pour différents traitements :

- ✓ en analyse de données textuelles, les similarités sont utilisées pour la description et l'exploration de données ;
- ✓ en recherche d'information, l'évaluation des similarités entre documents et requêtes est utilisée pour identifier les documents pertinents par rapport à des besoins d'information exprimés par les utilisateurs ;
- ✓ en Text Mining, les similarités sont utilisées pour produire des représentations synthétiques de vastes collections de documents.

Les techniques mises en œuvre pour calculer les similarités varient bien évidemment selon les disciplines, mais elles s'intègrent cependant le plus souvent dans une même approche générale en deux temps :

1. Les documents textuels sont d'abord associés à des représentations spécifiques qui vont servir de base au calcul des similarités. Bien que la nature précise des représentations utilisées dépende fortement du domaine d'application, il faut noter que, presque dans tous les cas, les documents sont représentés sous la forme d'éléments d'un espace vectoriel de grande dimension.
2. Un modèle mathématique est choisi pour mesurer les similarités.

Il existe deux types de similarité textuelle :

3.1. Similarité syntaxique :

En mathématiques et en informatique, une mesure permettant de comparer des documents textuels, consiste à comparer des chaînes de caractères. C'est une métrique qui mesure la similarité ou la dissimilarité entre deux chaînes de caractères. Par exemple, les chaînes de caractères "Sam" et "Samuel" peuvent être considérées comme similaires. Une telle mesure sur les chaînes de caractères fournit une valeur obtenue algorithmiquement. Parmi de telles mesures de similarité, citons par exemple, la distance de Levenshtein (ou distance d'édition), le coefficient de Dice, l'indice de Jaccard, la distance euclidienne, le cosinus, ...

Une mesure de similarité syntaxique permet de comparer des documents textuels en se basant sur les chaînes de caractères qui les composent. Par exemple, les chaînes de caractères "voiture" et "voiturier" peuvent être considérées comme très proches, alors que "voiture" et "automobile" pourront être considérées comme très différentes. Dans cette partie, nous présentons les mesures de similarité syntaxique les plus utilisées, en passant par la représentation vectorielle d'un document.

3.2. Modèle d'espace vectoriel :

Afin de réduire la complexité des documents et de faciliter leur manipulation, il faut transformer chaque document, i.e. sa version textuelle intégrale, en un vecteur qui décrit le contenu du document. La représentation d'un ensemble de documents sous forme de vecteurs dans un espace vectoriel commun est connue sous le nom de modèle d'espace vectoriel (vector space model). En recherche d'information, dans un modèle d'espace vectoriel, les documents sont représentés comme des vecteurs de caractéristiques représentant les termes qui apparaissent dans la collection. On parle aussi de 'sacs de mots' où les mots sont considérés comme indépendants et où l'ordre est sans importance. La valeur de chaque caractéristique est appelé le poids du terme et est en général une fonction de fréquence de termes dans le document. Par conséquent, en utilisant la fréquence de chaque terme comme un poids, les termes qui apparaissent le plus fréquemment sont plus

importants et donc descriptifs du document. La représentation d'un document sous forme vectorielle se déroule en 2 étapes :

- ✓ Extraire les termes pertinents du document ;
- ✓ Calculer les poids des termes restants.

3.2.1. Extraction des termes pertinents (lemmatisation) :

Il s'agit de prétraiter le texte des documents textuels en supprimant les mots-vides (Stop Words), la ponctuation et les éventuels 'retours-chariots', de lemmatiser le texte et de le segmenter.

Exemple :

Dans cet exemple nous allons utiliser les librairies Python pour lemmatiser la phrase de test suivante :

« Bouygues a eu une coupure de réseau à Marseille »

Pour ce faire il faut tout d'abord installer les librairies NLTK (pip install NLTK) et SPACY (pip install SPACY) :

En suite dans un environnement de développement (Spyder,jupyter,IDLE,...) Python nous allons lemmatiser notre phrase de test.

- ✓ Premièrement nous allons importer les bibliothèques du TAL (SPACY et NLTK) .
- ✓ Charger le modèle français.
- ✓ Eliminer les mots vides.

NB : Voir (**Implémentation en Python (Page 14)**)

3.2.2. Calcul de poids :

Le poids de chaque terme dans un document peut être obtenu de différentes manières : booléenne, fréquence des termes, tf-idf (Term frequency - Inverse Document Frequency).

➤ Méthode booléenne :

De manière booléenne, si un terme existe dans un document alors la valeur qui lui correspond vaut 1, sinon 0. L'approche booléenne est utilisée lorsque chaque terme est d'égale importance et s'emploie uniquement lorsque les documents sont de petites tailles.

Pour se faire nous allons rendre les mots de chaque phrase sous leur forme de base (**lemmatisation**) ensuite à l'aide de la bibliothèque **Pandas** nous allons réaliser la représentations vectorielles de manière booléenne des deux phrase de test :

Test="Bouygues a eu une coupure de réseau à Marseille"

Test1="Le réseau sera bientôt rétabli à Marseille"

La phrase test et test1 après lemmatisation et nettoyage deviennent :

Test= « bouygue coupure réseau marseille »

Test1= « réseau bientôt rétablir marseille»

Ensuite nous allons faire l'union des deux phrases, à l'aide de la commande suivante :

```
#L'union des deux phrases
test=test.split()
test1=test1.split()
total=set(t).union(set(t1))
print(total)

{'réseau', 'marseille', 'rétablir', 'coupure', 'bientôt', 'bouygue'}
```

Ensuite nous allons créer des dictionnaires de mots et leur occurrence pour chaque phrase comme et nous allons transformer les résultats sous format trames de données suit :

```
#création de dictionnaire de mots et leur occurrence pour chaque phrase
dic1=dict.fromkeys(total,0)
dic2=dict.fromkeys(total,0)

for word in test:
    dic1[word]+=1

for word in test1:
    dic2[word]+=1
#transformation des résultats sous format trames de données
print(pd.DataFrame([dic1, dic2]))
```

	réseau	marseille	rétablir	coupure	bientôt	bouygue
0	1	1	0	1	0	1
1	1	1	1	0	1	0

➤ Fréquence des termes tf :

Pour la fréquence des termes, le poids d'un terme est obtenu en comptant les occurrences du terme dans le document : **tf_{i,j}** représente donc la fréquence du terme **i** dans le document '**j**'.

Pour cette méthode nous allons définir la fonction **computeTF** qui retourne la fréquence de chaque terme dans une phrase :

```
#TF méthode
def computeTF(wordDict, bow):
    tfDict = {}
    bowCount = len(bow)
    for word, count in wordDict.items():
        tfDict[word] = count/float(bowCount)
    return tfDict
```

Ensuite nous allons afficher le résultat sous forme d'une trame de données comme suit :

```
#exécuter nos phrases à travers la fonction tf
tfest = computeTF(dic1, test)
tfest1 = computeTF(dic2, test1)
#transformation des résultats sous format trames de données
print(pd.DataFrame([tfest,tfest1]))
```

	réseau	marseille	rétablir	coupure	bientôt	bouygue
0	0.25	0.25	0.00	0.25	0.00	0.25
1	0.25	0.25	0.25	0.00	0.25	0.00

- 1- La lemmatisation du contenu d'un texte permet de regrouper les mots d'une même famille. Chacun des mots d'un contenu se trouve ainsi réduit en une entité appelée lemme (forme canonique). La lemmatisation regroupe les différentes formes que peut revêtir un mot, soit : le nom, le pluriel, le verbe à l'infinitif, ...
- 2- Les mots considérés vides, ici, sont : pour, le, la, les, l',..., un, une, des, ..., y, à, se, s', les dérivés du verbe être, les dérivés du verbe avoir, mon, ton, son, mes, tes, ..., qu', que, qui, ...

➤ Méthode Tf-Idf :

Le tf-idf permet, quant à lui, d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection. Le poids augmente proportionnellement avec le nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans la collection. Ainsi, la fréquence inverse du document (idf) est une mesure de l'importance du terme dans l'ensemble des documents. Dans le cas du tf-idf, elle vise à donner un poids plus important aux termes les moins fréquents, considérés comme les plus discriminants. Il s'agit de calculer le logarithme de l'inverse de la proportion de documents qui contiennent le terme : $\text{idf}_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$ où $|D|$ est le nombre total de documents et $|\{d_j : t_i \in d_j\}|$ est le nombre de documents où le terme t_i apparaît. Finalement, le poids s'obtient en multipliant les deux mesures : $\text{tf idf}_{i,j} = \text{tf}_{i,j} \cdot \text{idf}_i$.

word	tf		idf	Tf*idf	
	Test	Test1		Test	Test1
bientôt	0	¼=0.25	Log (2/1)=0.693	0	0.173
rétablir	0	¼=0.25	Log (2/1)=0.693	0	0.173
marseille	¼=0.25	¼=0.25	Log (2/2)=0	0	0
coupure	¼=0.25	0	Log (2/1)=0.693	0.173	0
bouygue	¼=0.25	0	Log (2/1)=0.693	0.173	0
réseau	¼=0.25	¼=0.25	Log (2/2)=0	0	0

➤ Fonction idf :

```
#fonction idf
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict
idf= computeIDF([dic1, dic2])
```

Affichage d'idf :

```
print(idf)

{'réseau': 0.0, 'marseille': 0.0, 'rétablir': 0.6931471805599453, 'coupure': 0.6931471805599453, 'bientôt': 0.6931471805599453, 'bouygue': 0.6931471805599453}
```

➤ Fonction tf_idf :

```
#tf_idf
def computeTFIDF(tfBagOfWords, idf):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idf[word]
    return tfidf
tfidfA = computeTFIDF(tftest, idf)
tfidfB = computeTFIDF(tftest1, idf)
print(pd.DataFrame([tfidfA, tfidfB]))
```

	réseau	marseille	rétablir	coupure	bientôt	bouygue
0	0.0	0.0	0.000000	0.173287	0.000000	0.173287
1	0.0	0.0	0.173287	0.000000	0.173287	0.000000

4. Mesures de similarité existante

Une mesure de similarité est, en général, une fonction qui quantifie le rapport entre deux objets, comparés en fonction de leurs points de ressemblance et de dissemblance. Les deux objets comparés sont, bien entendu, de même type.

4.1. Similarité cosinus :

La similarité cosinus est fréquemment utilisée [Baeza-Yates and Ribeiro-Neto, 1999] en tant que mesure de ressemblance entre deux documents d1 et d2. Il s'agit de calculer le cosinus de l'angle entre les représentations vectorielles des documents à comparer. La similarité obtenue $\text{sim_cosinus}(d1, d2) \in [0, 1]$.

$$sim_{cosinus}(d_1, d_2) = \frac{\vec{d_1} \cdot \vec{d_2}}{\|\vec{d_1}\| \cdot \|\vec{d_2}\|}$$

NB : Grâce à la bibliothèque NUMPY on peut calculer facilement la norme et le produit des vecteurs.

```
#Similarité Cosinus
import numpy as np
from numpy import dot
from numpy.linalg import norm
x=np.array(pd.Series(dic1))
y=np.array(pd.Series(dic2))
cos_sim =(dot(x,y)/(norm(x)*norm(y)))*100
if cos_sim>=100:
    cos_sim=100
print("%.2f" % cos_sim,'%')
```

50.00 %

4.2. Distance Euclidienne :

La distance euclidienne calcule la similarité entre deux documents d1 et d2 comme la distance entre leurs représentations vectorielles ramenées à un seul point.

$$sim_{euclidienne}(d_1, d_2) = \|\vec{d_1} - \vec{d_2}\|$$

NB : Grâce à la bibliothèque NUMPY on peut calculer facilement la distance euclidienne entre deux phrases (vecteurs).

Pour calculer la distance euclidienne entre les deux phrases de testes (**Test** et **Test1**)

Nous avons utilisé l'exécution de ces deux phrases à travers la fonction **tf_idf** .

```
x=np.array(pd.Series(tfidfA))
y=np.array(pd.Series(tfidfB))
#Distance Euclidienne
sim_eucl=norm(x-y)*100
if sim_eucl>=100:
    sim_eucl=100
print("%.2f" % sim_eucl,'%')
```

34.66 %

NB : il existe plusieurs méthodes de calcul de similarité entre deux phrases (documents) que nous n'avons pas cité comme (la méthode **Indice de Dice** , la méthode **Distance (d'édition) de Levenshtein**, méthode **Coefficient de Jaccard** , méthode **Coefficient de corrélation de Pearson** ,.....). Nous avons choisie de présenter ces deux méthodes (la méthode **Cosinus** et la méthode **Distance Euclidienne**) car elles sont plus utilisées pour le calcul de similarité.

5. Analyse Sémantique

Analyse Sémantique est un concept selon lequel un ensemble de documents ou de termes se voient attribuer une métrique basée sur la ressemblance de leur signification. Contrairement à l'analyse lexical qui se base sur le calcul d'un score de similarité qui se base sur le nombre d'unité lexical. Par exemple, il y a une similitude évidente entre les segments de texte « Je possède un chien » et « j'ai un animal », mais la plupart des techniques de mesure lexical vont échouer à identifier une similarité entre les deux textes.

➤ Technique existante

5.1. Approche Vectorielle :

Valeur Sémantique : consulte les autres termes utilisés à ses cotes dans des phrases afin de détermine la sémantique. Une manière simple de le faire est d'utiliser des vecteurs pour représenter le sens des mots, et d'utilise ensuite des mesures de similarité vectoriel. Par exemple, supposons que nous avons 3 mots : Poulain, Mouton et Cheval. Le mot Poulain apparait avec le mot Mouton 12 fois dans le document, et avec le cheval 3 fois. Considérant un espace vectoriel avec uniquement Poulain et Mouton en tant que vecteur de base. Si nous utilisant la similarité de Cosinus, nous avons donc :

$$\text{Sim (Poulain, Mouton)} = \frac{3*0+12*1}{\sqrt{3^2+12^2*1}} = \frac{12}{\sqrt{153}} \approx \mathbf{0.97}$$

$$\text{Sim (Poulain, Cheval)} = \frac{3*1+12*0}{\sqrt{3^2+12^2*1}} = \frac{3}{\sqrt{153}} \approx \mathbf{0.24}$$

Bi-Clustering : La classification double ou Bi-Clustering est une technique d'exploration des données permettant de segmenter simultanément des lignes et des colonnes d'une matrice. L'algorithme génère des sous-ensembles de ligne (bi-clusters) qui représente un comportement similaire sur un sous-ensemble de colonnes. Exemple : Dans le tableau ci-dessous, les documents D1 et D2 n'ont aucun mot ω_i en commun. Avec une mesure de similarité syntaxique, leur similarité est égale à zéro. Pourtant, on peut observer que d1 et d2 partagent des mots avec d3, ce qui signifie que les mots ω_2 et ω_3 ont quelques similitudes dans l'espace des documents. Avec une approche de bi-clustering, il est possible d'associer une similarité entre d1 et d2, qui sera bien sûr moins flagrante que celles entre d1 et d3 ou entre d2 et d3 mais elle sera néanmoins non nulle.

M	ω_1	ω_2	ω_3	ω_4
d_1	1	1	0	0
d_2	0	0	1	1
d_3	0	1	1	0

Approches topologiques : Cette approche s'appuie sur un réseau sémantique de mots, tel que le Word Net. Deux mots, leur similarité peut être estimée à partir de leur position relative dans la hiérarchie de la base de connaissance. Pour la langue française, une base a été créée : WOLF (Word Net Libre du Français).

5.2. Approche Statistique :

- **LSA :** ou l'analyse sémantique latente qui peut utiliser pour déterminer la distance entre les mots ou ensemble de mots. La LSA consiste à créer une matrice mot/paragraphe, mot/document, mot/passage, contrairement aux approches citées au paravent d'une matrice mot/mot. Dans une modélisation latente, les documents d'une collection sont modélisés comme une combinaison pondérée des thèmes latents dans un ensemble Z. Dans l'analyse sémantique latente probabiliste (PLSA), chaque thème latent possède un modèle de langage probabiliste $P(w|z)$ représente la probabilité que le mot w puisse être généré par un mélange pondéré des modèles latents des thèmes. Si un document est modélisé par une collection de mots C . le modèle PLSA est :

$$P(C||d_i) = \prod_{w \in V} (\sum_{z \in Z} P(w||z)P(z||d_i))^{c_w}$$

➤ **Avantage et Inconvénients**

Les approches sémantiques se basent sur les corpus ou la connaissance posent des problèmes de stockage et de complexité et sont souvent spécifiques à un domaine donné.

En conclusion

Nous avons passé en revue les principales méthodes de prétraitement de données textuelles, elles sont utilisées pour faciliter la traduction d'un texte écrit en langage humain en langage machine.

La plus grande remarque faite lors des recherches pour cet article est que la plupart de ces techniques n'existent que pour les langues **principales** et ne sont « parfaites » que pour l'Anglais en général. Pour des langues moins fournies, telle que le Wolof, il devient primordial de pouvoir mettre en place toutes ces techniques pour pouvoir traiter de manière efficace des textes écrits à l'aide de ces dernières.

Chapitre III : Mise en place d'une application pour l'analyse de textes

1. Objectif :

L'objectif de ce projet est de réaliser une application qui teste la similarité entre deux documents, vu que le calcul de similarité est une tâche essentielle du Traitement Automatique de données, textuelles ou non, et a toujours reçu une attention particulière de la communauté scientifique. Si l'on se restreint aux données textuelles, le besoin est avant tout venu de besoin pour la recherche d'information. Il s'agit en effet de pouvoir d'une part de mesurer le degré de proximité entre des documents et d'autre part de pouvoir identifier, et ordonner, la liste des documents les plus pertinents à offrir en réponse à une requête dans un moteur de recherches.

Nous avons choisie comme type d'analyse de texte : l'analyse syntaxique.

2. Algorithme de test de similarité :

Cette approche calcule la similitude en mesurant le cosinus de l'angle entre deux vecteurs. Mathématiquement parlent, La similarité de cosinus est une mesure de similarité entre deux vecteurs non nuls dans une espace vectoriel. Le cosinus de 0 degré radiant est 1 et moins de 1 dans l'intervalle $[0, \pi]$. Deux vecteurs avec même orientation ont une similarité de cosinus de 1.

La similitude cosinus est avantageuse car même si les deux documents similaires sont éloignés de la distance euclidienne (en raison de la taille du document), il y a de fortes chances qu'ils soient toujours orientés plus près l'un de l'autre. Plus l'angle est petit, plus la similitude cosinus est élevée.

Pour appliquer ces algorithmes on doit tout d'abord préparer nos données (Document, texte, Paragraphe ...).

Etape 1 : Phase de prétraitement

Cette méthode utilise la fréquence des mots comme pondération des termes. Ce qui oblige de prétraitée toute sorte de texte.

On commence par la transformation tout le document on minuscule puis on élimine tous les mots vides qui ne vont pas nous servir à différencier le document. Ensuite on applique l'algorithme de lemmatisation afin de transformer chaque mot à sa racine ce qui permet de bien connaître l'occurrence de chaque terme dans le document.

Après on crée pour chaque document son propre dictionnaire de mots. Ensuite on compte le nombre d'occurrence de **TOKEN** de chaque document.

Etape 2 : Calcul de la similarité

On applique la formule de cosinus sur les deux vecteurs de chaque document :

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

Comme on a mentionné en haut, plus le résultat est proche de 1 plus il y a de similarité entre les documents.

3. Implémentation en Python

Nous avons utilisé les documents de tests suivants :

Document 1 :

« La nature est l'ensemble des êtres vivants, animaux et végétaux, ainsi que le milieu où ils se trouvent (minéraux, mers, montagnes, continents) qu'on trouve dans l'Univers.

En fait, la nature est tout ce qui n'est pas construit directement par l'homme, qui est appelé alors « artificiel ».

La nature est souvent associée à l'environnement, la forêt, ou les rivières.

C'est aussi le monde physique qui nous entoure. »

Document 2 :

« L'Univers est l'ensemble de tout ce qui existe, régi par un certain nombre de lois.

La cosmologie cherche à appréhender l'Univers d'un point de vue scientifique, comme l'ensemble de la matière distribuée dans l'espace-temps.

Pour sa part, la cosmogonie vise à établir une théorie de la création de l'Univers sur des bases philosophiques ou religieuses.

La différence entre ces deux définitions n'empêche pas nombre de physiciens d'avoir une conception finaliste de l'Univers (voir à ce sujet le principe anthropique).

Si l'on veut faire correspondre le mouvement des galaxies avec les lois physiques telles qu'on les conçoit actuellement, on peut considérer que l'on n'accède par l'expérience qu'à une faible partie de la matière de l'Univers¹, le reste se composant de matière noire.

Par ailleurs, pour expliquer l'accélération de l'expansion de l'Univers, il faut également introduire le concept d'énergie sombre.

Plusieurs modèles alternatifs ont été proposés pour faire correspondre les équations et nos observations en prenant d'autres approches. »

Etape1 : importation des librairies nécessaires

```
#importaion des Librairies nécessaires
import spacy
from spacy.lang.fr.stop_words import STOP_WORDS
import pandas as pd
import numpy as np
from numpy import dot
from numpy.linalg import norm
from nltk import word_tokenize
```

Figure 2 : importation des librairies nécessaires

Etape2 : Récupération/Lecture des documents à tester

```
#Récupération/Lecture des documents à tester
x=input("Donner le chemin du premier fichier: ")
x1=input("Donner le chemin du deuxième fichier:")
with open(x,"r",encoding="utf-8") as f:
    test=f.read()
with open(x1,"r",encoding="utf-8") as f1:
    test1=f1.read()
```

Donner le chemin du premier fichier: C:/Users/Najoua/Desktop/file1.txt
Donner le chemin du deuxième fichier:C:/Users/Najoua/Desktop/file2.txt

Figure 3 : Récupération/Lecture des documents à tester

Etape3 : Passer les deux documents en minuscules

```
#Passer Les deux documents en minuscules
test=test.lower()
test1=test1.lower()
print(test)
print("*****")
print(test1)

nature ensemble vivant animal végétal milieu trouver minéral mer montagne continent trouve univers
nature construire homme appeler artificiel
nature associer environnement forêt rivière
monde physique entoure
*****
univers ensemble existe régir nombre loi

cosmologie cherche appréhender univers point vue scientifique ensemble matière distribuer espace tem
ps
part cosmogonie vis établir théorie création univers base philosophique religieux
différence définition empêche nombre physicien conception finaliste univers voir sujet principe anth
ropique

vouloir faire correspondre mouvement galaxie loi physique concevoir actuellement
pouvoir considérer accéder expérience faible partie matière univers1
composer matière noir
ailleurs expliquer accélération expansion univers falloir également introduir concept énergie sombre
modèle alternatif proposer faire correspondre équation observation prenant approche
```

Figure 4 : Passer les deux documents en minuscules

Etape4 : Chargement du model français

```
#chargement du model français
nlp = spacy.load("fr_core_news_sm")
doc=nlp(test)
doc1=nlp(test1)
```

Figure 5 : Chargement du model français

Etape5 : Récupération/Ajout des mots vides

```
#Récupération des mots vides
fr_stop=set(STOP_WORDS)
#methode 1 d'ajout des mots vides
par={"(", ")", ",", ".", ":", ";", "!", "?", "'", '"', "#", "%", "&", ">", "[", "]", "{", "}", "/", "\\", "-", "_", "^", "~", "`", "%"}
fr_stop=fr_stop.union(par)
#methode 2
fr_stop.add("@")
```

Figure 6 : Récupération/Ajout des mots vides

Etape6 : Lemmatisation

```
#Lemmatisation
lem=[]
for word in doc:
    lem.append(word.lemma_)
lem1=[]
for word in doc1:
    lem1.append(word.lemma_)
```

Figure 7 : lemmatisation

Etape7 : Affichage des listes de mots lemmatisés de chaque document

```
#Affichage des Listes de mots Lemmatisés de chaque documents
print(lem)
print("*****")
print(lem1)

['le', 'nature', 'être', 'le', 'ensemble', 'de', 'être', 'vivant', ',', 'animal', 'et', 'végétal', ',', 'ainsi', 'que', 'le',
'milieu', 'où', 'il', 'se', 'trouver', '(', 'minéral', ',', 'mer', ',', 'montagne', ',', 'continent', ')', 'qu', 'on', 'trouv
e', 'dans', 'le', 'univers', '.,', '\n', 'en', 'fait', ',', 'le', 'nature', 'être', 'tout', 'ce', 'qui', 'n', 'être', 'pas', 'c
onstruire', 'directement', 'par', 'le', 'homme', ',', 'qui', 'être', 'appeler', 'alors', '«', 'artificiel', '»', '.,', '\n', 'l
e', 'nature', 'être', 'souvent', 'associer', 'à', 'le', 'environnement', ',', 'le', 'forêt', ',', 'ou', 'le', 'rivière', '.,',
'\n', 'ce', 'être', 'aussi', 'le', 'monde', 'physique', 'qui', 'nous', 'entoure', '.']
*****
['le', 'univers', 'être', 'le', 'ensemble', 'de', 'tout', 'ce', 'qui', 'existe', ',', 'régir', 'par', 'un', 'certain', 'nombr
e', 'de', 'loi', '.,', '\n\n', 'le', 'cosmologie', 'cherche', 'à', 'appréhender', 'le', 'univers', 'de', 'un', 'point', 'de', 'v
ue', 'scientifique', ',', 'comme', 'le', 'ensemble', 'de', 'le', 'matière', 'distribuer', 'dans', 'le', 'espace', '-', 'temps',
',', '\n', 'pour', 'son', 'part', ',', 'le', 'cosmogonie', 'vis', 'à', 'établir', 'un', 'théorie', 'de', 'le', 'création', 'd
e', 'le', 'univers', 'sun', 'un', 'base', 'philosophique', 'ou', 'religieux', '.,', '\n', 'le', 'différence', 'entre', 'ce', 'de
ux', 'définition', 'n', 'empêche', 'pas', 'nombre', 'de', 'physicien', 'de', 'avoir', 'un', 'conception', 'finaliste', 'de',
'le', 'univers', '(', 'voir', 'à', 'ce', 'sujet', 'le', 'principe', 'anthropique', ')', '.,', '\n\n', 'si', 'le', 'on', 'vouloi
r', 'faire', 'correspondre', 'le', 'mouvement', 'de', 'galaxie', 'avec', 'le', 'loi', 'physique', 'tel', 'que', 'on', 'le', 'co
ncevoir', 'actuellement', ',', '\n', 'on', 'pouvoir', 'considérer', 'que', 'le', 'on', 'n', 'accéder', 'par', 'le', 'expérien
ce', 'que', 'à', 'un', 'faible', 'partie', 'de', 'le', 'matière', 'de', 'le', 'univers1', ',', '\n', 'le', 'reste', 'se', 'comp
oser', 'de', 'matière', 'noir', '.,', '\n', 'par', 'ailleurs', ',', 'pour', 'expliquer', 'le', 'accélération', 'de', 'le', 'expa
nsion', 'de', 'le', 'univers', ',', 'il', 'falloir', 'également', 'introduire', 'le', 'concept', 'de', 'énergie', 'sombre', '.,',
'\n', 'plusieurs', 'modèle', 'alternatif', 'avoir', 'être', 'proposer', 'pour', 'faire', 'correspondre', 'le', 'équation', 'l
e', 'notre', 'observation', 'en', 'prenant', 'd', 'autre', 'approche', '.']
```

Figure 8 : Affichage des listes de mots lemmatisés de chaque document

Etape8 : nettoyage des deux documents des mots vides

```
#nettoyage des deux documents des mots vides
clean=[]
for word in lem:
    if word not in fr_stop:
        clean.append(word)
clean1=[]
for word in lem1:
    if word not in fr_stop:
        clean1.append(word)
```

Figure 9 : nettoyage des deux documents des mots vides

Etape9 : Retourner les nouveaux documents après nettoyage

```
#Retourner Les nouveaux documents après nettoyage
test="" ".join(clean)
test1="" ".join(clean1)
```

Figure 10 : Retourner les nouveaux documents après nettoyage

Etape10 : Résultat de nettoyage

```
#Résultat de nettoyage
print(test)
print("*****")
print(test1)
```

nature ensemble vivant animal végétal milieu trouver minéral mer montagne continent trouve univers
nature construire homme appeler artificiel
nature associer environnement forêt rivière
monde physique entoure

univers ensemble existe régir nombre loi

cosmologie cherche appréhender univers point vue scientifique ensemble matière distribuer espace temps
part cosmogonie vis établir théorie création univers base philosophique religieux
différence définition empêche nombre physicien conception finaliste univers voir sujet principe anthropique

vouloir faire correspondre mouvement galaxie loi physique concevoir actuellement
pouvoir considérer accéder expérience faible partie matière univers1
composer matière noir
ailleurs expliquer accélération expansion univers falloir également introduir concept énergie sombre
modèle alternatif proposer faire correspondre équation observation prenant approche

Figure 11 : Résultat de nettoyage

Etape11 : tokenisation

```
#tokenization
R=word_tokenize(test)
R1=word_tokenize(test1)
```

Figure 12 : tokenisation

Etape12 : Résultat de tokenisation

```
#Résultat de tokenisation
print(R)
print("*****")
print(R1)
```

['nature', 'ensemble', 'vivant', 'animal', 'végétal', 'milieu', 'trouver', 'minéral', 'mer', 'montagne', 'continent', 'trouve', 'univers', 'nature', 'construire', 'homme', 'appeler', 'artificiel', 'nature', 'associer', 'environnement', 'forêt', 'rivière', 'monde', 'physique', 'entoure']

['univers', 'ensemble', 'existe', 'régir', 'nombre', 'loi', 'cosmologie', 'cherche', 'appréhender', 'univers', 'point', 'vue', 'scientifique', 'ensemble', 'matière', 'distribuer', 'espace', 'temps', 'part', 'cosmogonie', 'vis', 'établir', 'théorie', 'création', 'univers', 'base', 'philosophique', 'religieux', 'différence', 'définition', 'empêche', 'nombre', 'physicien', 'conception', 'finaliste', 'univers', 'voir', 'sujet', 'principe', 'anthropique', 'vouloir', 'faire', 'correspondre', 'mouvement', 'galaxie', 'loi', 'physique', 'concevoir', 'actuellement', 'pouvoir', 'considérer', 'accéder', 'expérience', 'faible', 'partie', 'matière', 'univers1', 'composer', 'matière', 'noir', 'ailleurs', 'expliquer', 'accélération', 'expansion', 'univers', 'falloir', 'également', 'introduire', 'concept', 'énergie', 'sombre', 'modèle', 'alternatif', 'proposer', 'faire', 'correspondre', 'équation', 'observation', 'prenant', 'approche']

Figure 13 : Résultat de tokenisation

Etape13 : Retourner les nouveaux documents après prétraitement

```
#Retourner Les nouveaux documents après prétraitement
str0="" ".join(R)
str1="" ".join(R1)
print(test)
print("*****")
print(test1)

nature ensemble vivant animal végétal milieu trouver minéral mer montagne continent trouve univers
nature construire homme appeler artificiel
nature associer environnement forêt rivière
monde physique entoure
*****
univers ensemble existe régir nombre loi

cosmologie cherche appréhender univers point vue scientifique ensemble matière distribuer espace temps
part cosmogonie vis établir théorie création univers base philosophique religieux
différence définition empêche nombre physicien conception finaliste univers voir sujet principe anthropique

vouloir faire correspondre mouvement galaxie loi physique concevoir actuellement
pouvoir considérer accéder expérience faible partie matière univers1
composer matière noir
ailleurs expliquer accélération expansion univers falloir également introduir concept énergie sombre
modèle alternatif proposer faire correspondre équation observation prenant approche
```

Figure 14 : Retourner les nouveaux documents après prétraitement

Etape14 : Union des deux documents

```
#Union des deux documents
t=str0.split()
t1=str1.split()
total=set(t).union(set(t1))
print(total)

{'actuellement', 'énergie', 'concept', 'construire', 'considérer', 'nombre', 'cherche', 'artificiel',
'prenant', 'partie', 'approche', 'point', 'création', 'modèle', 'part', 'voir', 'physicien', 'appréhe
nder', 'correspondre', 'monde', 'existe', 'végétal', 'ensemble', 'ailleurs', 'noir', 'sujet', 'enviro
nnement', 'concevoir', 'accéder', 'matière', 'homme', 'établir', 'théorie', 'sombre', 'temps', 'mouve
ment', 'conception', 'cosmologie', 'galaxie', 'également', 'alternatif', 'philosophique', 'différenc
e', 'espace', 'univers', 'minéral', 'vouloir', 'accélération', 'falloir', 'continent', 'trouve', 'tro
uver', 'pouvoir', 'vivant', 'expérience', 'introduir', 'proposer', 'définition', 'nature', 'vis', 'en
toure', 'observation', 'expansion', 'rivière', 'univers1', 'montagne', 'faire', 'anthropique', 'distr
ibuer', 'milieu', 'régir', 'composer', 'scientifique', 'loi', 'associer', 'physique', 'mer', 'appele
r', 'religieux', 'principe', 'faible', 'équation', 'forêt', 'cosmogonie', 'finaliste', 'empêche', 'ba
se', 'expliquer', 'animal', 'vue'}
```

Figure 15 : Union des deux documents

Etape15 : création de dictionnaire de mots et leur occurrence pour chaque document

```
#création de dictionnaire de mots et Leur occurrence pour chaque document
dic1=dict.fromkeys(total,0)
dic2=dict.fromkeys(total,0)

for word in t:
    dic1[word]+=1

for word in t1:
    dic2[word]+=1
```

Figure 16 : création de dictionnaire de mots et leur occurrence pour chaque document

Etape16 : transformation des résultats sous format trames de données


```
#transformation des résultats sous format trames de données
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
#Modification directe de l'attribut
pd.options.display.max_columns = None
pd.options.display.max_rows = None
print(pd.DataFrame([dic1, dic2]))
```

	actuellement	énergie	concept	construire	considérer	nombre	cherche	\
0	0	0	0	1	0	0	0	
1	1	1	1	0	1	2	1	

	artificiel	prenant	partie	approche	point	création	modèle	part	voir	\
0	1	0	0	0	0	0	0	0	0	
1	0	1	1	1	1	1	1	1	1	

	physicien	appréhender	correspondre	monde	existe	végétal	ensemble	\
0	0	0	0	1	0	1	1	
1	1	1	2	0	1	0	2	

	ailleurs	noir	sujet	environnement	concevoir	accéder	matière	homme	\
0	0	0	0	1	0	0	0	1	
1	1	1	1	0	1	1	3	0	

	établir	théorie	sombre	temps	mouvement	conception	cosmologie	\
0	0	0	0	0	0	0	0	
1	1	1	1	1	1	1	1	

	galaxie	également	alternatif	philosophique	différence	espace	univers	\
0	0	0	0	0	0	0	1	
1	1	1	1	1	1	1	5	

	minéral	vouloir	accélération	falloir	continent	trouve	trouver	\
0	1	0	0	0	1	1	1	
1	0	1	1	1	0	0	0	

	pouvoir	vivant	expérience	introduire	proposer	définition	nature	vis	\
0	0	1	0	0	0	0	3	0	
1	1	0	1	1	1	1	0	1	

	entoure	observation	expansion	rivière	univers1	montagne	faire	\
0	1	0	0	1	0	1	0	
1	0	1	1	0	1	0	2	

	anthropique	distribuer	milieu	régir	composer	scientifique	loi	\
0	0	0	1	0	0	0	0	
1	1	1	0	1	1	1	2	

	associer	physique	mer	appeler	religieux	principe	faible	équation	\
0	1	1	1	1	0	0	0	0	
1	0	1	0	0	1	1	1	1	

	forêt	cosmogonie	finaliste	empêche	base	expliquer	animal	vue	\
0	1	0	0	0	0	0	1	0	
1	0	1	1	1	1	1	0	1	

Figure 17 : transformation des résultats sous format trames de données

Etape17 : affichage des deux vecteurs des deux documents

```
#récupération du vecteur du document 1
x=np.array(pd.Series(dic1))
#récupération du vecteur du document 2
y=np.array(pd.Series(dic2))
#Affichage des deux vecteurs
print(x)
print("*****")
print(y)
```




```
[0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 0 3 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 1 0]
*****
[1 1 1 0 1 2 1 0 1 1 1 1 1 1 1 1 1 2 0 1 0 2 1 1 1 0 1 1 3 0 1 1 1 1 1 1
1 1 1 1 1 1 5 0 1 1 1 0 0 0 1 0 1 1 1 1 0 1 0 1 1 0 2 1 1 0 1 1 1 2
0 1 0 0 1 1 1 1 0 1 1 1 1 1 0 1]
```

Figure 18 : affichage des deux vecteurs des deux documents

Etape18 : Calcul de la similarité

```
#Calcul de la similarité
cos_sim =(dot(x,y)/(norm(x)*norm(y)))*100
if cos_sim>=100:
    cos_sim=100
#Affichage du résultat
print("%.2f" % cos_sim, '%')
```



```
13.13 %
```

Figure 19 : Calcul de la similarité

4. Interface graphique

Tkinter (de l'anglais Tool kit interface) est la bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. Elle vient d'une adaptation de la bibliothèque graphique Tk écrite pour Tcl.

Nous avons créé deux fenêtres : une pour la récupération des chemins de documents à comparer et le déclenchement du test et l'autre fenêtre pour l'affichage du résultat de la similarité.

Nous avons utilisé une barre de progression pour présenter le niveau de similarité entre les deux documents.

Nous avons affiché juste la partie entière du pourcentage de similarité.

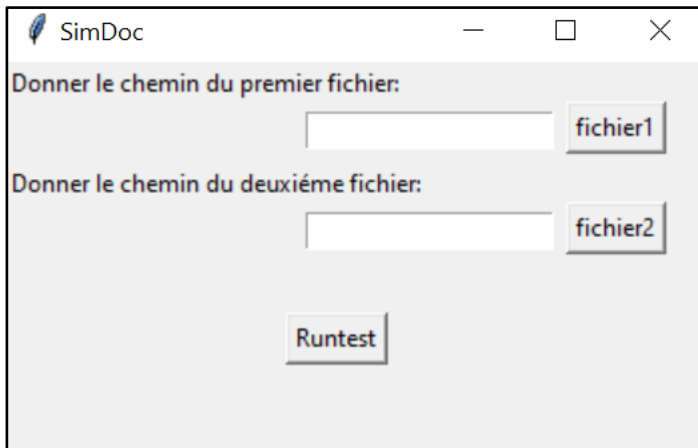


Figure 20 : fenêtre1

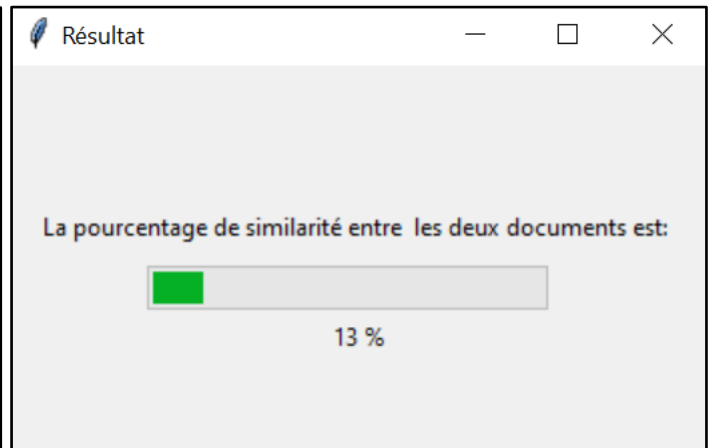


Figure 21 : fenêtre2

5. Outils :

Pour la réalisation de ce projet nous avons utilisé trois librairies : pour la partie prétraitement du texte nous avons utilisé (**NLTK** et **SPACY**) et **NUMPY** pour le calcul de la similarité.

Commande d'installation des librairies :

SPACY : PIP INSATLL SAPCY

NLTK : PIP INSTALL NLTK

NUMPY : PIP INSTALL NUMPY

Après avoir installé la bibliothèque SAPCY il faut télécharger les modèles français :

python -m spacy download fr_core_news_sm

TKINTER pour réaliser l'interface graphique.

Environnements de développements : **Spyder, Jupyter, IDLE**

Conclusion

Le NLP est un élément important dans le développement de l'intelligence artificielle. En effet, la langue joue un rôle central dans la création d'ordinateurs capables de penser de façon autonome. L'approche du Natural Language Processing a ainsi pu créer des liens importants entre l'être humain et l'ordinateur.

On utilise aujourd'hui ces techniques pour la traduction et le traitement de texte mais aussi dans les call centers. Il existe également déjà des programmes capables de créer eux-mêmes des textes. Déjà aujourd'hui, les utilisateurs peuvent " parler " à des chatbots de fournisseurs sélectionnés sur Skype pour réserver des billets ou lancer des requêtes simples.

Google souhaite également faire de son traducteur un service de traduction en live. En même temps, la technologie est utilisée par de nombreux assistants numériques de grandes sociétés Internet, comme Amazon Echo, Windows Cortana ou Siri d'Apple.

Il s'agit donc d'un secteur de recherche très florissant avec des enjeux majeurs. Il s'agit aussi et surtout d'un domaine clé pour l'analyse quali-quantitative dans la mesure où la masse de textes est là, mais nécessite des traitements précis pour avoir accès à une analyse fine.

Bibliographie

<https://www.cairn.info/revue-reseaux-2014-6-page-25.htm?contenu=article>

[https://fr.ryte.com/wiki/Natural language processing](https://fr.ryte.com/wiki/Natural_language_processing)

[https://www.gpomag.fr/web/images/stories/livres blancs/contentanalytics.pdf](https://www.gpomag.fr/web/images/stories/livres_blancs/contentanalytics.pdf)

<https://www.ekino.com/articles/introduction-au-nlp-partie-ii>

<https://hal.archives-ouvertes.fr/hal-00874280/document>

<https://tartarus.org/martin/PorterStemmer/>

<https://blog.baamtu.com/from-text-to-words/>

[https://jep-taln2020.loria.fr/wp-content/uploads/JEP-TALN-RECITAL-2020 paper 210.pdf](https://jep-taln2020.loria.fr/wp-content/uploads/JEP-TALN-RECITAL-2020_paper_210.pdf)

https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da?source=search_post-----1

<https://towardsdatascience.com/machine-learning-nlp-text-classification-using-scikit-learn-python-and-nltk-c52b92a7c73a>

<https://medium.com/@adriensieg/text-similarities-da019229c894>