

Projet Programmation : **Analyse comparative de corpus**

Anne-Sophie Koch et Tom Duffes

Année universitaire 2020-2021
ICOM, Université Lyon 2 , Master 1 Informatique

Table des matières:

Lien du dépôt GitHub	2
Spécifications	2
Analyse	2
Environnement de travail et données identifiées dans la spécification	2
Diagramme UML	3
Conception	3
Partage des tâches	3
Bibliothèques requises	4
Explication de quelques algorithmes spécifiques	4
nettoyer_texte(text)	4
tfidf(voca)	5
Liste des variables utiles à l'analyse	6
Problèmes rencontrés	6
Utilisation	8
Tests et validation	9
Maintenance	10
Limites	10
Améliorations	10

1. Lien du dépôt GitHub

Voici le lien de notre projet : <https://github.com/Ansoko/Analyse-corpus-tf-idf>

Il comporte 3 fichiers :

- `Projet_analyse.py` (contient le code à exécuter, qui fait appel à `Projet_classes.py`)
- `Projet_classes.py` (contient les différentes classes et méthodes de notre projet)
- `Projet_tests.py` (contient les tests des différentes méthodes de `Projet_classes.py`)

2. Spécifications

L'objectif de ce projet était de concevoir un outil qui, dans un premier temps, prépare les données. Les données, issus de documents disponibles sur internet, sont de véritables mines d'informations si l'on parvient à en extraire ce que l'on souhaite.

En effet, il est par exemple possible de connaître l'auteur ou la date de centaines de milliers de documents sur Reddit, la fréquence et l'importance de différents mots clés dans un corpus regroupant des articles scientifiques avec ArXiv, et bien d'autres.

Pour notre projet, nous avons choisi de récupérer des documents provenant de Reddit et d'ArXiv, et de travailler sur le contenu du document (les différents mots).

Dans un second temps, ce projet avait pour but de concevoir un outil efficace qui permettrait d'analyser comparativement deux corpus de documents.

Pour faire nos analyses comparatives, nous avons choisi de travailler avec les fréquences des mots, ainsi qu'avec une analyse du vocabulaire à l'aide d'un score, score calculé à partir de la mesure du TF-IDF.

Une description des résultats de l'analyse sera disponible dans la partie XX du rapport.

3. Analyse

a. Environnement de travail et données identifiées dans la spécification

Les spécifications données imposent de travailler sous Python, nous avons donc travaillé Python avec Spyder. Les données importées pour constituer les corpus viennent de Reddit et d'Arxiv.

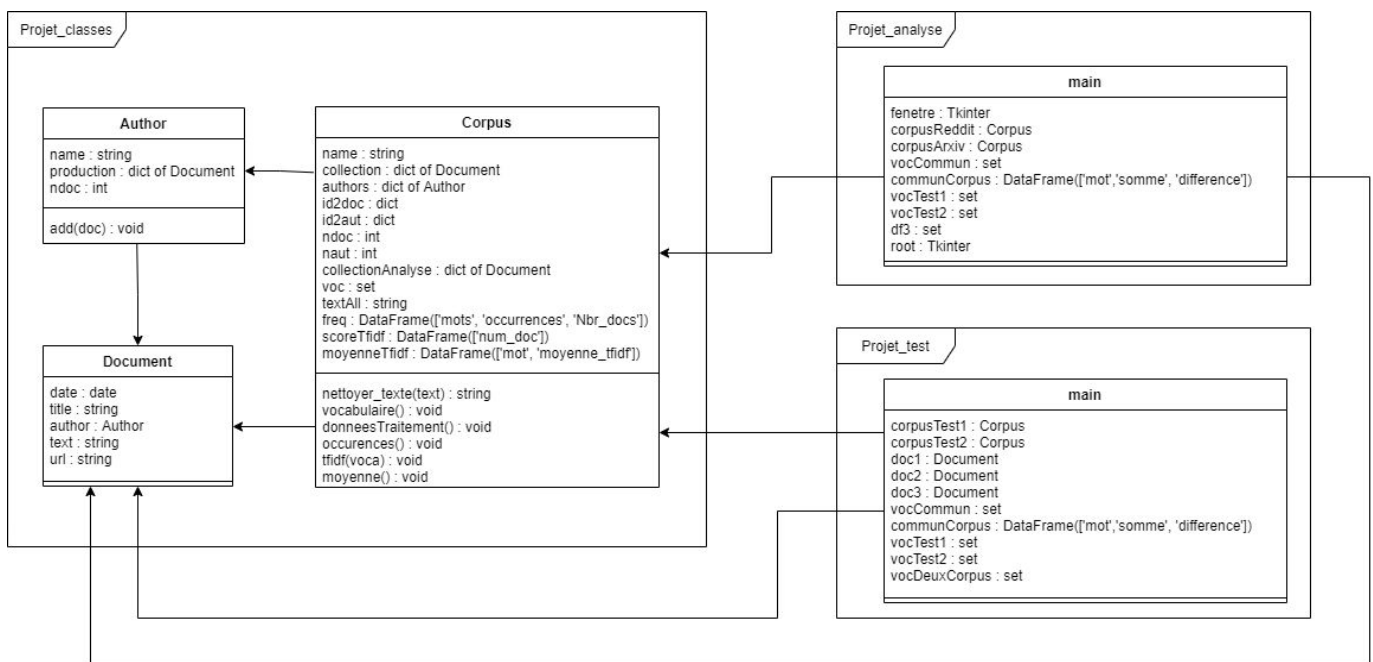
Les corpus traités sont modélisés par une classe `Corpus` qui va également contenir toutes les méthodes d'analyse.

Dans la spécification, il est vivement suggéré de comparer plusieurs corpus à l'aide d'identificateurs, comme le score TF-IDF qui calcule l'importance d'un mot dans un corpus. Les tableaux contenant ces scores sont spécifiques à un corpus, ce sont donc des variables de classe de `Corpus`.

Le score TF-IDF (term frequency-inverse document frequency) est une méthode de pondération utilisée en linguistique pour mesurer l'importance d'un mot dans un corpus. Il faut donc implémenter une fonction permettant son calcul en l'adaptant à notre projet : le corpus est considéré comme un gros document, et chaque document est considéré comme une longue phrase.

Aussi, le texte importé de Reddit ou d'Arxiv est brut : afin de l'analyser au mieux, il faudra lui appliquer un pré-traitement qui supprimera toutes les données superflues (nombres, symboles...) et optimisera au maximum l'analyse (regroupement des mots de même famille...).

b. Diagramme UML



Sur ce diagramme UML, les flèches représentent l'utilisation des autres classes (par exemple, le main du Projet_test utilise les attributs et les méthodes de Corpus et de Document).

Le projet est donc composé d'un fichier contenant les différentes classes (Corpus, Author et Document), d'un fichier de test et d'un fichier contenant les données des corpus, le déroulement de l'analyse et l'affichage.

4. Conception

a. Partage des tâches

Les tâches prévues et les attributions sont :

- Créer la méthode de calcul du score TF-IDF (Anne-Sophie)
- Réaliser une méthode d'analyse (à deux)
- Afficher les résultats et création de la mini-interface (Tom)

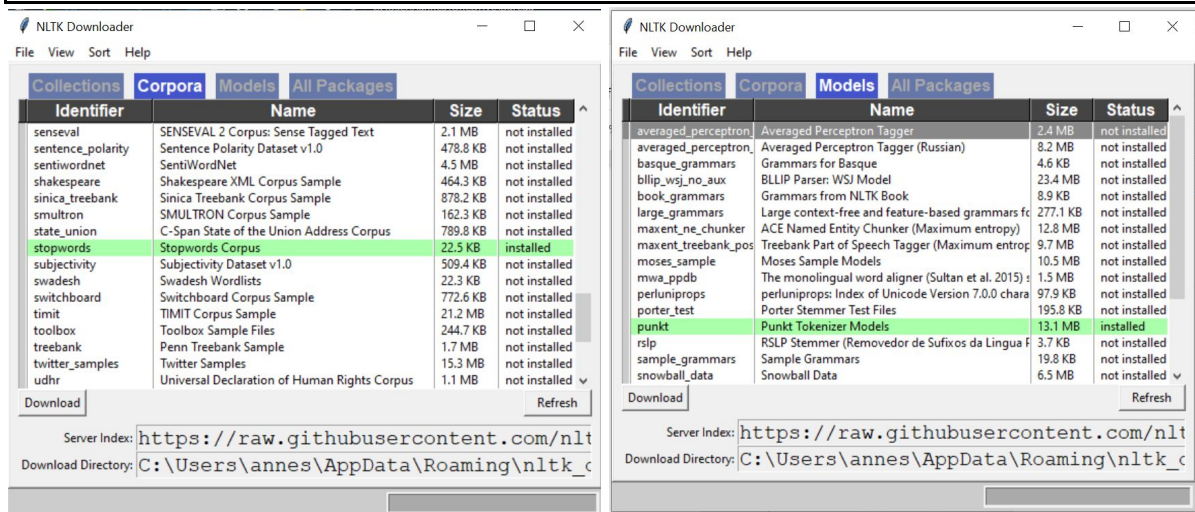
b. Bibliothèques requises

Pour ce projet, les bibliothèques à installer sont :

- pandas
- urllib3
- xmltodict
- praw
- nltk :

Avant de lancer le programme, il faut installer stopwords et punkt en lançant la bibliothèque depuis la console python :

```
import nltk
print('stopwords et punkt à télécharger')
nltk.download()
```



c. Explication de quelques algorithmes spécifiques

Les méthodes les plus spécifiques réalisées pour répondre au besoin sont :

- [nettoyer texte](#)
- [tfidf](#)

i. nettoyer_texte(text)

Cette fonction permet de nettoyer du texte de façon à ce que l'analyse soit optimale. Elle retourne une valeur de type str.

Tout d'abord, nous avons repris la fonction de l'exercice 4.4, qui permettait la mise en minuscules, le remplacement des passages à la ligne ainsi que le remplacement des signes de ponctuation et des chiffres à l'aide d'expressions régulières.

Afin de procéder à un nettoyage plus profond, nous nous sommes aidé de ce tutoriel :

<https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089> (William Scott). Dans cet article, le processus de nettoyage est expliqué plus longuement :

En plus des étapes précédentes, on peut supprimer les mots de liaison qui n'ajoutent aucun sens supplémentaire au document (comme 'is', 'for', 'when', ...).

Ces mots sont regroupés dans la variable `stopword` de la librairie `nlTK`. Pour chaque mot du document, on va tester s'il fait partie de l'un deux, et si c'est le cas, il va être supprimé du document.

Dans cette même librairie, il est également possible de "stemming" les mots, c'est à dire de les réduire à leur racine. En faisant cela, les mots conjugués ou de même famille deviennent identiques. Cela rend plus précis le scoring `tf-idf` par la suite.

Enfin, afin d'éviter les mots vides potentiellement créés précédemment, on supprime les multiples espaces et les espaces en début et en fin de texte, grâce à des expressions régulières.

ii. `tfidf(voca)`

Cette fonction permet de calculer le score `tf-idf` d'un corpus et d'enregistrer ces scores dans la variable de classe `scoreTfidf`.

Ce score étant calculé à l'aide d'une part le "TF" (Term Frequency, la fréquence des termes dans un document) et d'autre part du "IDF" (Inverse Document Frequency, qui mesure le niveau informatif d'un terme), on va d'abord les calculer (dans les mêmes boucles afin d'éviter de parcourir le corpus plusieurs fois) avant de calculer le score final en multipliant les deux valeurs.

On va commencer par faire une boucle sur tous les documents du corpus, puis séparer les mots du texte du document et boucler sur ces mots.

Pour le `tf` :

On incrémente la valeur du nombre d'occurrence de ce mot dans la ligne du dataframe correspondant au numéro du document. Si ce mot ne fait pas partie de l'étiquette des colonnes, on l'ajoute et on lui attribue la valeur 1.

Après avoir fini la boucle sur les mots du document, on va diviser chaque terme par la longueur du document.

Pour l'`idf` :

On va tout d'abord créer un nouveau dataframe "`idf`" composé d'une colonne '`mots`' et d'une autre '`Nbr_docs`'.

Pour chaque document, on va créer un ensemble vide. Dès que l'on boucle sur un nouveau mot, on l'ajoute à cet ensemble. Ainsi, pour éviter d'incrémenter plusieurs fois la valeur qui compte le nombre de documents dans lequel le mot apparaît, on va d'abord tester si le mot n'a pas déjà été rencontré dans le texte en utilisant l'ensemble.

Si le mot n'a jamais été rencontré dans le corpus auparavant, on ajoute une ligne au dataframe `idf`. Sinon, on vérifie que le mot n'a pas déjà été rencontré dans le document. Si ce n'est pas le cas, on incrémente la valeur.

Enfin, lorsque l'on a parcouru tous les documents, on applique le calcul de l'`idf` sur chaque ligne : $\ln(\text{nombre de document} / \text{nombre de document contenant ce mot})$

Pour le tf-idf :

Maintenant que l'on a les scores tf et idf pour chaque mot, il suffit de multiplier chaque valeur du dataframe contenant les fréquences par l'idf de ce mot.

d. Liste des variables utiles à l'analyse

OCCURRENCE :

corpusReddit.freq : dataframe contenant les occurrences de chaque mots de Reddit

corpusArxiv.freq : dataframe contenant les occurrences de chaque mot de Arxiv

SCORE TF-IDF :

corpusReddit.scoreTfidf : dataframe contenant le score pour chaque mots de Reddit

corpusArxiv.scoreTfidf : dataframe contenant le score pour chaque mots de Arxiv

corpusReddit.moyenneTfidf : dataframe contenant le score moyen des mots de Reddit

corpusArxiv.moyenneTfidf : dataframe contenant le score moyen des mots de Arxiv

ANALYSE

communCorpus (somme, difference, plus_important) : dataframe contenant la somme des scores tf-idf de chaque mot, la différence absolue des scores tf-idf de chaque mot et dans quel corpus il est le plus important.

VOCABULAIRE :

corpusReddit.voc : ensemble de tout le vocabulaire de Reddit

corpusArxiv.voc : ensemble de tout le vocabulaire de Arxiv

vocCommun : ensemble de tout le vocabulaire des deux corpus réunis

vocReddit : ensemble du vocabulaire unique à Reddit

vocArxiv : ensemble du vocabulaire unique à Arxiv

df3 : ensemble du vocabulaire commun aux deux corpus

e. Problèmes rencontrés

Premier problème rencontré : Comprendre l'intérêt du TF-IDF, savoir l'utiliser.

Résolu : oui

Une fois le calcul du TF-IDF fonctionnel, nous ne nous étions jamais vraiment posé la question de ce que nous allions pouvoir analyser avec ce dernier. Nous avons décidé de faire la moyenne des scores de chaque mot pour chaque corpus, puis d'en faire la somme et la différence entre les deux corpus, et nous pensons que c'est une utilisation plutôt intéressante du score. Cela permet de nous donner d'une part la liste des mots les plus importants des deux corpus, et d'autre part dans quel corpus ce mot est le plus important par rapport à l'autre.

Deuxième problème rencontré : Le temps d'exécution de l'outil pour un très grand nombre de documents est très long.

Résolu : non

Dès le départ, nous avons travaillé sur une base d'une vingtaine de documents (pour chaque corpus), que nous avons réduit à cinq pour accélérer le processus lors des tests. Lors de test sur 100 documents pour chaque corpus, le programme prend plusieurs minutes à s'exécuter.

Dans ce sens, notre outil n'est pas réellement utilisable sur un très grand nombre de documents. Ce problème est devenu une limite de notre outil.

Troisième problème rencontré : L'affichage des données est très compliqué.

Résolu : partiellement (L'affichage des données reste très sommaire).

En effet, nous nous sommes un peu entêtés à garder en sortie des tableaux sous formes de pandas.dataframe, structure que tkinter ne gère pas du tout et n'affiche pas. (Normalement, on aurait pu créer un widget « canvas », puis insérer nos résultats par exemple).

Au final, après beaucoup de temps de recherche, le seul code viable que nous avons trouvé vient de ce lien : <https://stackoverflow.com/fr/q/7308298>

En plus d'être peu compréhensible, même l'auteur de la réponse ne peut expliquer son code.

Ce fut compliqué d'adapter ce code à notre projet, et le rendu pose problème car le tableau apparaît bien dans la fenêtre, mais, n'appartenant à aucun widget, il nous est impossible de rendre l'aspect graphique plus intéressant.

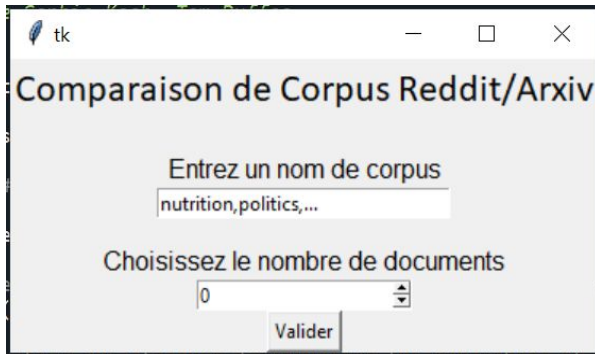
De plus, tout nouveau widget écrase l'espace consacré au dataframe, et l'affichage doit donc se limiter à du langage python, à savoir la fonction print().

Finalement, l'affichage est correct mais manque cruellement d'ergonomie.

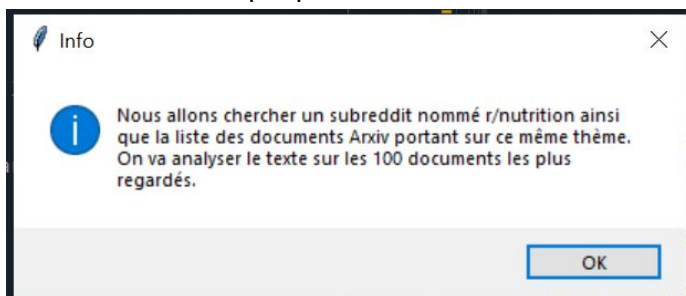
Ce bout de code est peut être la cause d'un autre problème: à la fin de l'analyse, il faut reboot le noyau ou relancer spyder pour pouvoir relancer le programme.

f. Utilisation

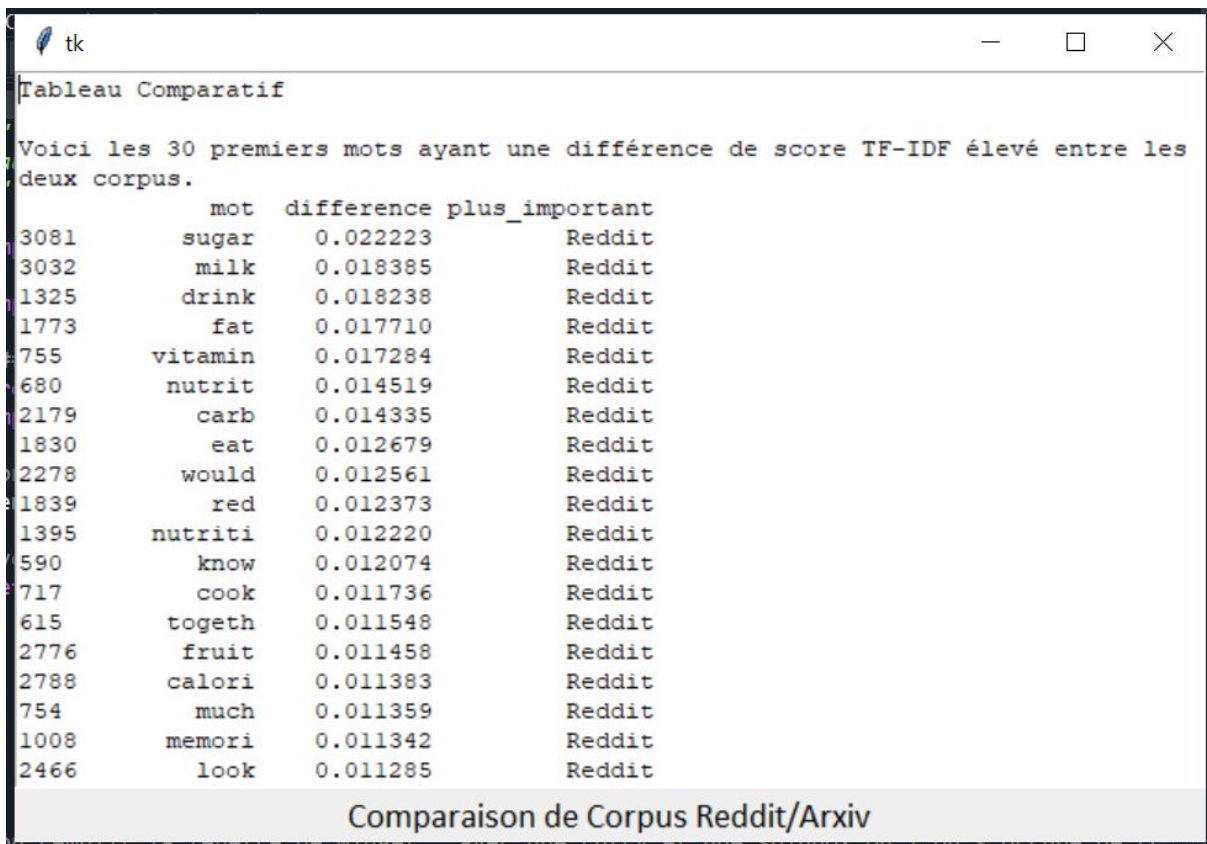
Lancez le fichier "Projet_analyse.py". Une fenêtre va alors s'ouvrir :



Entrez le thème du corpus, ainsi que le nombre de documents voulus dans les corpus dans les différents champs, puis validez. Une nouvelle fenêtre informative va s'ouvrir :



Une fois cette dernière fermée, l'analyse commence. elle peut prendre plus ou moins de temps, selon la taille du corpus. Une fois l'analyse terminée, la fenêtre de résultat s'ouvre :



	mot	difference	plus_important
3081	sugar	0.022223	Reddit
3032	milk	0.018385	Reddit
1325	drink	0.018238	Reddit
1773	fat	0.017710	Reddit
755	vitamin	0.017284	Reddit
680	nutrit	0.014519	Reddit
2179	carb	0.014335	Reddit
1830	eat	0.012679	Reddit
2278	would	0.012561	Reddit
1839	red	0.012373	Reddit
1395	nutriti	0.012220	Reddit
590	know	0.012074	Reddit
717	cook	0.011736	Reddit
615	togeth	0.011548	Reddit
2776	fruit	0.011458	Reddit
2788	calori	0.011383	Reddit
754	much	0.011359	Reddit
1008	memori	0.011342	Reddit
2466	look	0.011285	Reddit

Il suffit de défiler vers le bas dans la fenêtre pour avoir accès au reste des informations.

Ci dessous une rapide présentation des résultats:

Le premier tableau présente les 30 mots ayant la différence de score TF-IDF la plus élevée (entre les deux corpus).

Ce tableau représente donc les 30 mots ayant un très grand score TF-IDF dans un des deux corpus, soustrait à un faible score dans l'autre corpus. C'est-à-dire les 30 mots importants dans un corpus, mais peu importants dans l'autre.

Le deuxième tableau présente les mots avec les sommes des scores TF-IDF les plus élevées et les plus faibles.

C'est-à-dire les mots les plus importants et les moins importants dans l'ensemble des deux corpus

Ensuite, la liste de tous les mots que l'on retrouve à la fois dans le corpus Reddit et dans le corpus Arxiv.

Enfin, un dernier tableau, qui regroupe les deux premiers tableaux de l'affichage, mais seulement sur les mots que l'on retrouve dans la liste des mots communs entre les deux corpus. (trié par la somme la plus élevée, soit le deuxième tableau).

Ce dernier tableau est intéressant lorsqu'on effectue l'analyse avec peu de documents, car beaucoup de mots peuvent avoir un grand score sans forcément être présent dans l'autre corpus.

5. Tests et validation

Afin de tester nos méthodes, nous avons créé un fichier spécialement pour les tests (Projet_test.py).

Dans ce programme python, chaque méthode est testée individuellement et affiche ce qu'elle retourne afin de vérifier son bon déroulement.

Par exemple, ci-dessous la vérification du prétraitement des documents :

```
In [4]: print("Traitement des documents")
...: print(corpusTest1.collection[0].get_text())
...: corpusTest1.donneesTraitement()
...: print(corpusTest1.collectionAnalyse[0].get_text())
Traitement des documents
When you have eliminated all which is impossible, then whatever remains,
however improbable, must be the truth. (common)
elimin imposs whatev remain howev improb must truth common
```

On peut alors remarquer que les symboles, les mots génériques ont bien été retirés. Les mots ont été raccourcis à leur racine.

Autre exemple, calcul des scores TF-IDF :

```

score tf-idf
tf terminé
idf terminé
tf-idf terminé
num_doc    dr    armstrong    rais    ...    way    perfectli    clear    us
0          0.0  0.051344    0.051344  0.025672  ...  0.00000    0.00000    0.00000    0.00000
1          1.0  0.000000    0.000000  0.000000  ...  0.04621    0.04621    0.04621    0.04621

```

Après avoir calculé à la main à côté pour certains de ces mots, nous nous sommes rendu compte que le score correspond bien.

Lors de ces tests, nous avons également soulevé un cas particulier : lorsque le corpus ne comporte qu'un seul document, le score est faussé car le score de l'idf est alors égal à 0 ($\ln(1/1) = 0$).

```

score tf-idf
tf terminé
idf terminé
tf-idf terminé
num_doc    elimin    imposs    whatev    remain    howev    improb    must    truth    common
0          0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0

```

6. Maintenance

a. Limites

On a pu remarquer deux limites à notre outil :

Tout d'abord, comme indiqué dans les problèmes rencontrés, la complexité en temps de l'algorithme est assez conséquente, ce qui ne nous permet pas d'analyser des énormes quantités de documents. En effet, il faut déjà plusieurs minutes de calcul pour cent documents (pour chaque corpus), alors il nous est impossible d'effectuer l'analyse sur plusieurs milliers de documents. (En réalité, c'est surtout le calcul du TF-IDF pour chacun des mots qui est le plus gourmand.)

Une deuxième limite qu'on pourrait trouver est plutôt liée au calcul du TF-IDF.

En effet, le "TF" se calcule en trouvant le nombre d'occurrence d'un mot, divisé par le nombre total de mot dans le document.

Les documents de Reddit comportant en moyenne moins de mots que les documents de Arxiv, le TF, et donc le score du TF-IDF sera souvent plus important sur des mots provenant de Reddit (comme on peut le voir lors de l'affichage des résultats).

b. Améliorations

Nous avons aussi trouvé deux pistes d'amélioration de notre outil.

On pourrait par exemple proposer un choix de mot différents pour chacun des corpus.

Pour ce faire, il faudrait simplement attribuer une nouvelle variable et modifier la mini-interface, cela serait assez facile.

On pourrait aussi avoir le choix de la provenance des documents ; en effet la méthode de nettoyage de texte et de calcul du TF-IDF fonctionne sur n'importe quel texte en entrée. On peut imaginer qu'il serait possible de trouver d'autres sites qui donnent accès à leurs données et qu'il serait possible de les récupérer via python. Cela resterait largement faisable, mais il faudrait d'abord se renseigner sur des sites donnant libre accès à leurs données, données qui devraient rester pertinentes dans notre outil d'analyse