

Rapport projet 1 NLP

Maud Tissot et Aurélien Pouxviel

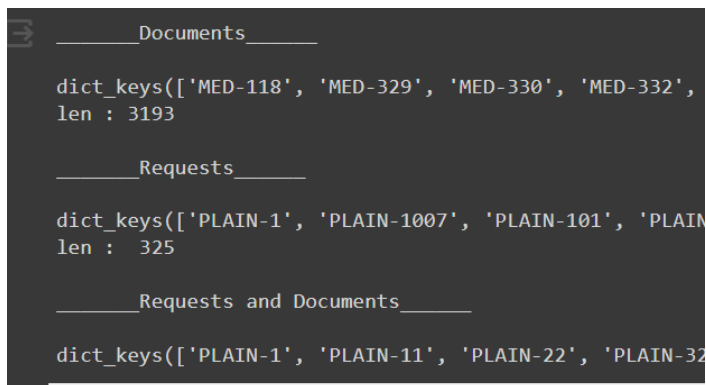
1. Présentation du problème et du dataset

- 1.1 Objectifs

Le but du projet est de développer notre propre système de recherche d'information sur un corpus spécifique. Pour ce faire, nous sommes autorisés à utiliser n'importe quel type de prétraitement et à manipuler le vocabulaire des documents. L'ensemble de données est un corpus médical. Les documents sont des résumés de publications médicales provenant de PubMed et les requêtes sont des vulgarisations sur les sujets liés aux documents de PubMed.

Il s'agit d'un corpus très complexe où les approches modernes d'apprentissage profond ne parviennent pas à faire mieux que BM25.

- 1.2 Présentation du dataset



```
➡ _____Documents_____
dict_keys(['MED-118', 'MED-329', 'MED-330', 'MED-332',
len : 3193

_____Requests_____
dict_keys(['PLAIN-1', 'PLAIN-1007', 'PLAIN-101', 'PLAIN
len : 325

_____Requests and Documents_____
dict_keys(['PLAIN-1', 'PLAIN-11', 'PLAIN-22', 'PLAIN-32
```

Ces dictionnaires contiennent des informations sur les documents, les requêtes et leurs associations :

dicDoc :

Ce dictionnaire contient des informations sur les documents médicaux. Chaque clé représente l'identifiant d'un document (par exemple, "MED-118", "MED-329"). Les valeurs correspondantes sont le contenu textuel des documents.

dicReq :

Ce dictionnaire semble contenir des simplifications d'articles précédents. Chaque clé représente un identifiant de requête (par exemple, 'PLAIN-901', 'PLAIN-912').

dicReqDoc :

Ce dictionnaire correspond à des associations entre plusieurs documents de médecine et une simplification. La valeur numérique peut indiquer une certaine forme de pertinence ou de poids (par

exemple, {'PLAIN-901' : {'MED-2421 ':2 ... }...}.

2. Explication de l'algorithme BM25

Nous avons créé des définitions pour expliquer le code donné dans le projet.

Nous avons décomposé le code de l'exemple en plusieurs fonctions pour le rendre plus facile à comprendre.

La méthode `run_bm25_only` renvoie un score pour évaluer la recherche d'information entre une requête et plusieurs articles scientifiques. A l'intérieur :

`loadNFCorpus` : charge les données dans trois dictionnaires différents.

`selectNb` : sélectionne un certain nombre de documents et requêtes pour réduire la quantité de données pour tester l'algorithme.

`getAllVocab` : renvoie pour chaque document deux listes de tout le vocabulaire présent dans les documents et dans les requêtes

`getTokenDoc` : Retourne à chaque document un token.

`getTokenReq` : Idem pour les requêtes

`bm25_ngcd` : Initie un modèle bm25 en utilisant bm25 Okapi et la liste des documents. Il calcule ensuite une liste de scores pour chaque requête. Le score doit être élevé si les documents correspondent à la demande. Enfin, ndcg score évalue la qualité des résultats. Voici un exemple :

`bm25score` = [0.9 , 0.01 , 0.8, 0,2...]

`trueDoc` = [1. , 0 , 1 , 0 ...]

=> ici le score ndcg devrait être élevé car les probabilités élevées sont associées aux vrais documents (1) et les faibles aux documents qui ne correspondent pas à la requête (0).

3. Text processing & tokenization

- 3.1 - Final processing méthodes

Dans un premier temps, nous avons rajouté des fonctions de preprocessing de la donnée. Notre fonction permet de choisir les méthodes à appliquer.

- stopwords

Nous utilisons NLTK qui permet d'enlever les 'stopwords' en anglais, et nous avons choisi pour longueur des mots >2.

- link et adresses

Nous avons trouvé en visualisant les mots qu'il y avait beaucoup de liens ou d'adresses. Ces informations ne sont pas pertinentes dans nos modèles et nous avons implémenté une fonction permettant de les supprimer ou non.

- Lemmatisation

Ensuite, si l'utilisateur le veut, il peut lemmatiser les mots. Cela réduit les mots à leur forme canonique (sa racine).

- DelWords

Enfin, nous avons codé la possibilité de supprimer des mots spécifiques qui fausseraient certaines visualisations.

Voici un exemple avec :

`dicDoc['MED-118']`

"alkylphenols human milk relations dietary habits central taiwan pubmed ncbi abstract aims study determine concentrations num nonylphenol np num octylphenol op num human milk samples examine related factors including mothers demographics dietary habits women consumed median amount cooking oil significantly higher op concentrations num ng/g consumed num ng/g num op concentration significantly consumption cooking oil beta num num fish oil capsules beta num num adjustment age body mass index bmi np concentration significantly consumption fish oil capsules beta num num processed fish products beta num num food pattern cooking oil processed meat products factor analysis strongly op concentration human milk num determinations aid suggesting foods consumption nursing mothers order protect infants np/op exposure num elsevier rights reserved \n"

- Il y a 118 mots. Voici les résultats en exécutant nos différentes fonctions de preprocessing :

```
Basics processings : 111
Delete links      : 108
Delete num word   : 95
Delete links and num word : 92
Final processing : ['alkylphenols', 'human', 'milk', 'relation',
```

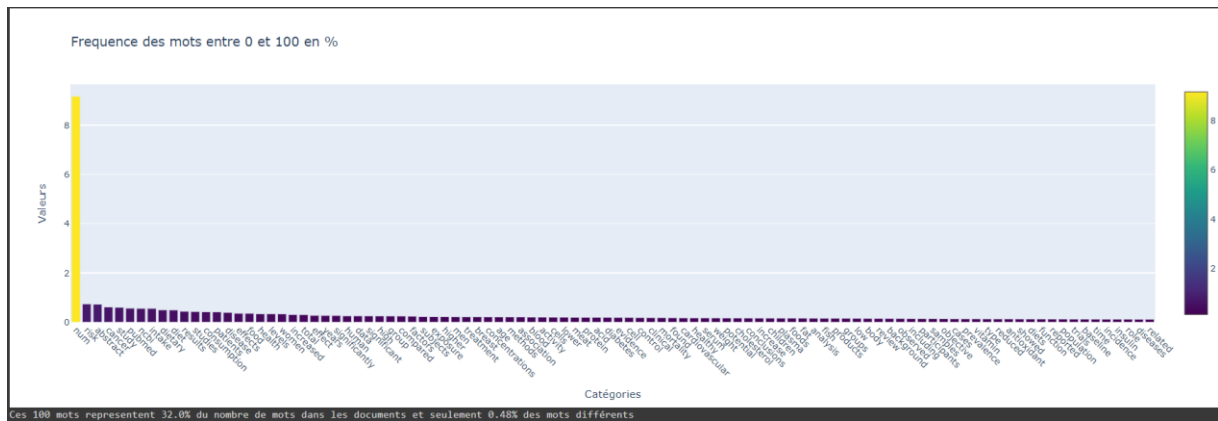
- 3.2 - Mots les plus utilisés

Nous avons codé plusieurs fonctions pour essayer d'identifier les plus les plus utilisés dans les documents et vulgarisations.

***Dans les documents :*

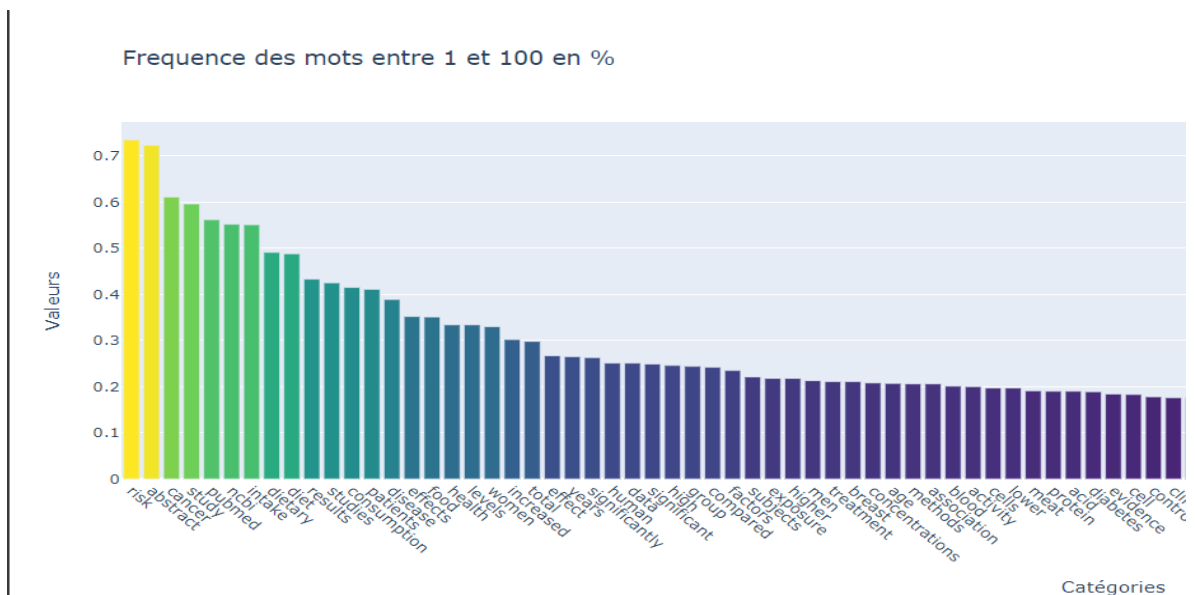
Nombre de mots : 440443

Nombre de mots différents : 20964



Ces 100 mots représentent 32.0% du nombre de mots dans les documents et seulement 0.48% des mots différents.

Comme on peut l'observer, le mot 'num' fausse toute la liste et est omniprésent. Voici quand on le retire :

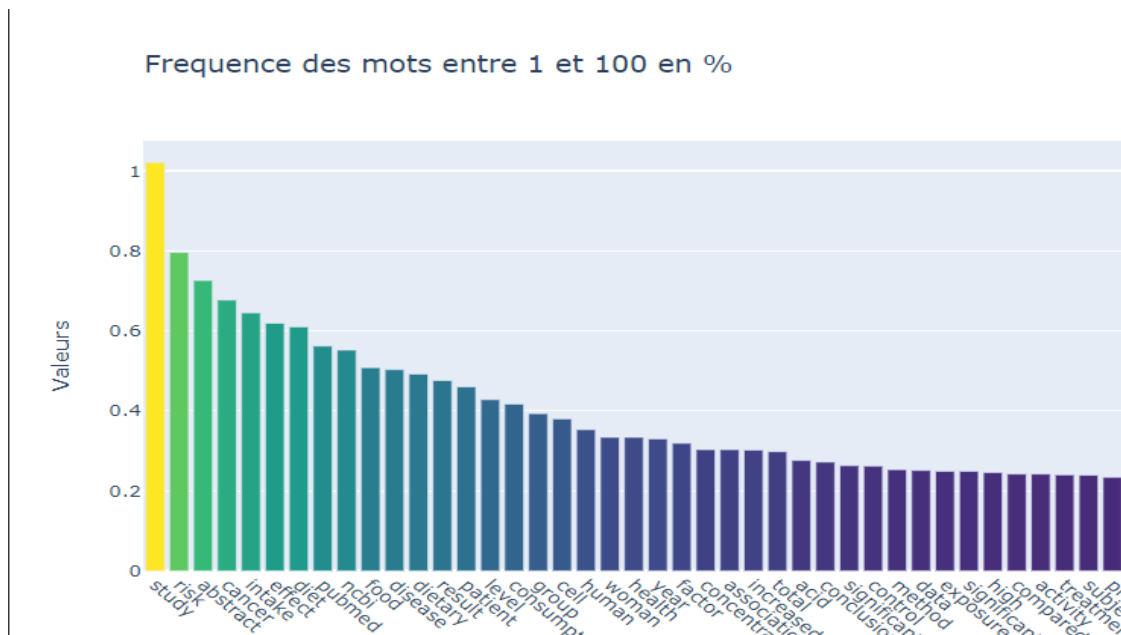


Ces 99 mots représentent 23.0% du nombre de mots dans les documents et seulement 0.47% des mots différents.

**** Maintenant appliquons la lemmatisation et regardons à nouveau le graphique :**

Nombre de mots : 440443

Nombre de mots différents : 18905

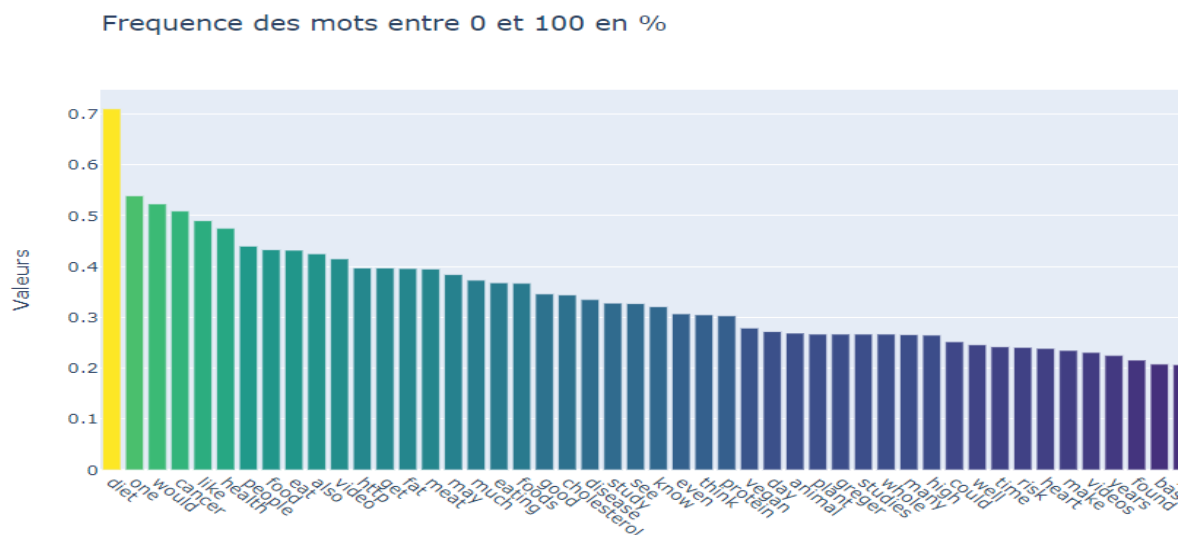


Ces 99 mots représentent 26.0% du nombre de mots dans les documents et seulement 0.52% des mots différents. On observe un changement radical de la fréquence des mots, avec 'study' qui devient le plus fréquent.

**** Maintenant passons aux vulgarisations :**

Nombre de mots : 303277

Nombre de mots différents : 22116

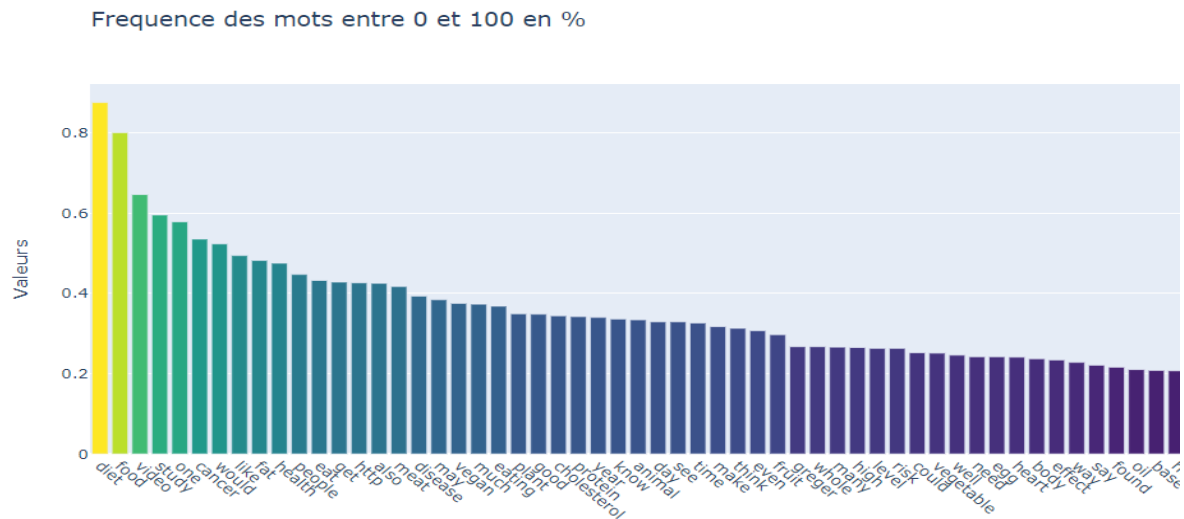


Ces 100 mots représentent 25.0% du nombre de mots dans les documents et seulement 0.45% des mots différents.

**** Après lemmatisation :**

Nombre de mots : 303277

Nombre de mots différents : 19878



Ces 100 mots représentent 27.0% du nombre de mots dans les documents et seulement 0.5% des mots différents. Certains gagnent énormément en fréquence, comme 'food' passant de 0.4 à 0.8%.

Pour récapituler : Dans tous les documents, il y a plus de 440 000 mots, dont 20 000 sont uniques, et environ 2 000 de moins une fois les mots réduits.

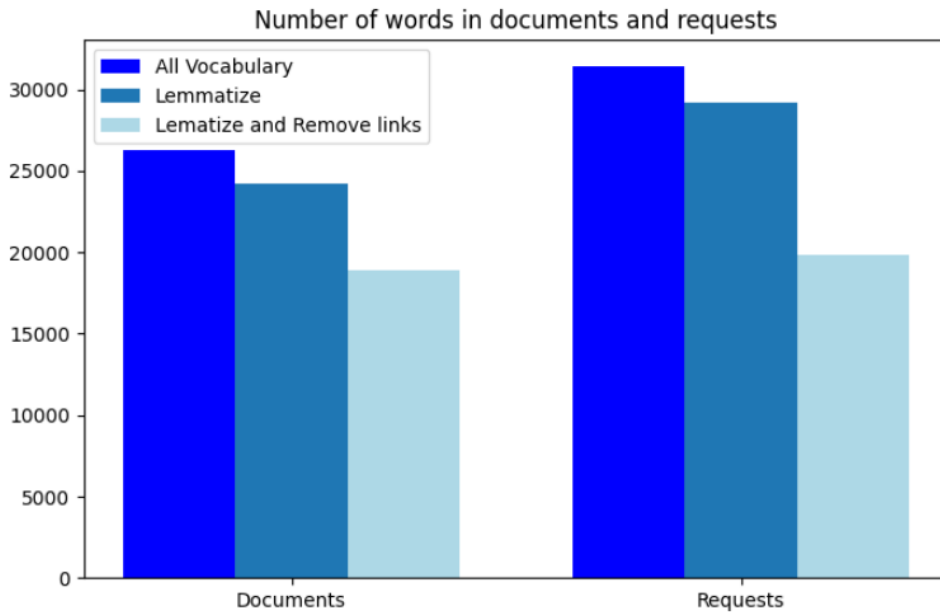
Dans toutes les requêtes, il y a plus de 300 000 mots, dont 22 000 sont uniques, et environ 1 000 de moins une fois les mots réduits.

Par conséquent, nous pourrions supprimer les N mots les plus fréquents ou seulement les mots sans signification (par exemple, "dire", "aimer") que l'on trouve le plus souvent dans les requêtes.

Nous avons donc défini une liste de mots assez fréquents que nous pourrions supprimer dans les différents preprocessing. Une liste de test à été par exemple :

['num', 'study', 'risk', 'abstract', 'cancer', 'intake', 'effect', 'diet', 'pubmed', 'ncbi', 'food', 'disease', 'dietary', 'result', 'patient', 'level', 'consumption', 'group', 'cell', 'human']

- 3.3 - Effets globaux de nos méthodes



Nous avons pu réduire drastiquement le nombre de mots à traiter pour nos modèles.

4. Embeddings sur les mots

Nous avons choisi dans un premier temps de visualiser des embeddings sur les mots de chaque document ou vulgarisation. L'ensemble des tests sur les embeddings ont été réalisés sur un échantillon restreint des données. Ensuite, nous les visualisons avec une réduction de dimension par TSNE en 3D, et pour aller un peu plus loin, nous avons implémenté une clusterisation des mots avec KMEANS sur cette réduction. Nous savons que les résultats des clusters ne seront pas optimaux, notre but était juste d'explorer.

- 4.1 - TF-IDF embeddings

Sur les documents 'doc':

TF-IDF Embeddings - 3D t-SNE Visualization with Clusters



Sur les vulgarisations 'req':

TF-IDF Embeddings - 3D t-SNE Visualization with Clusters

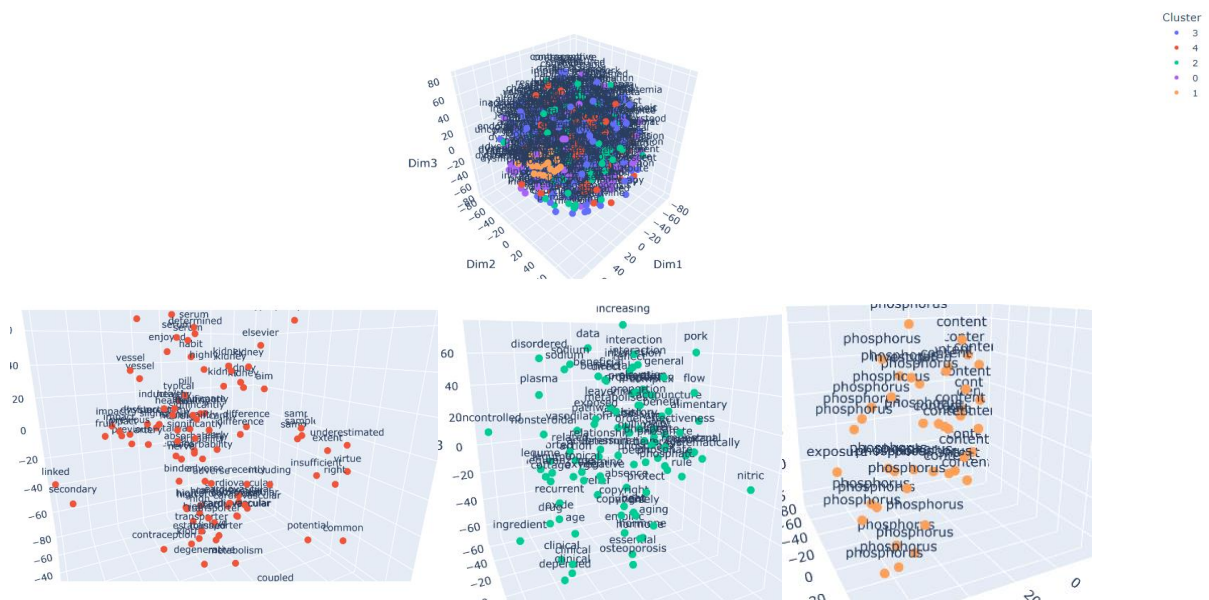


Comme on peut l'observer, TF-IDF ne génère pas de très bon embeddings, et la clusterisation ne fonctionne pas non plus. 99% des mots appartiennent à 1 seul cluster. De plus, la visualisation 3D perd beaucoup d'informations pour former des clusters.

- 4.2- Word2Vec embeddings

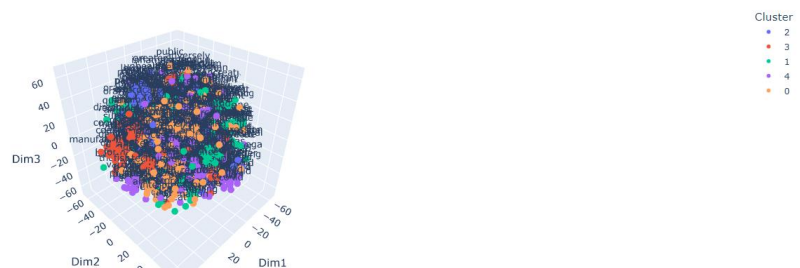
Sur les documents 'doc':

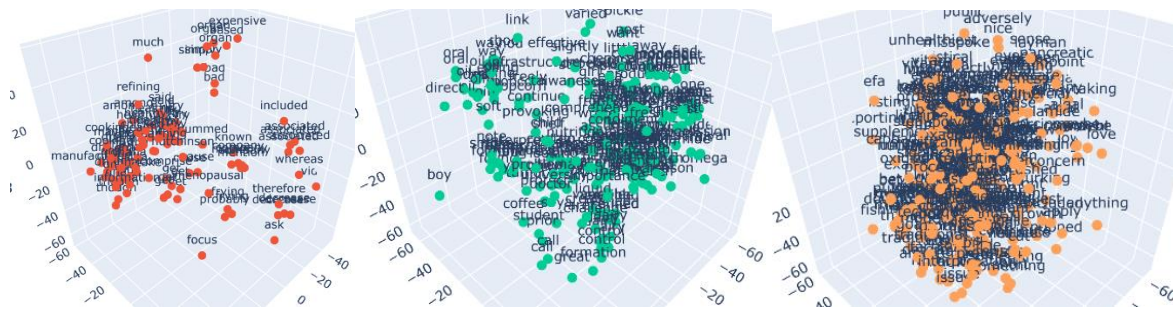
Word2Vec Embeddings - 3D t-SNE Visualization with Clusters



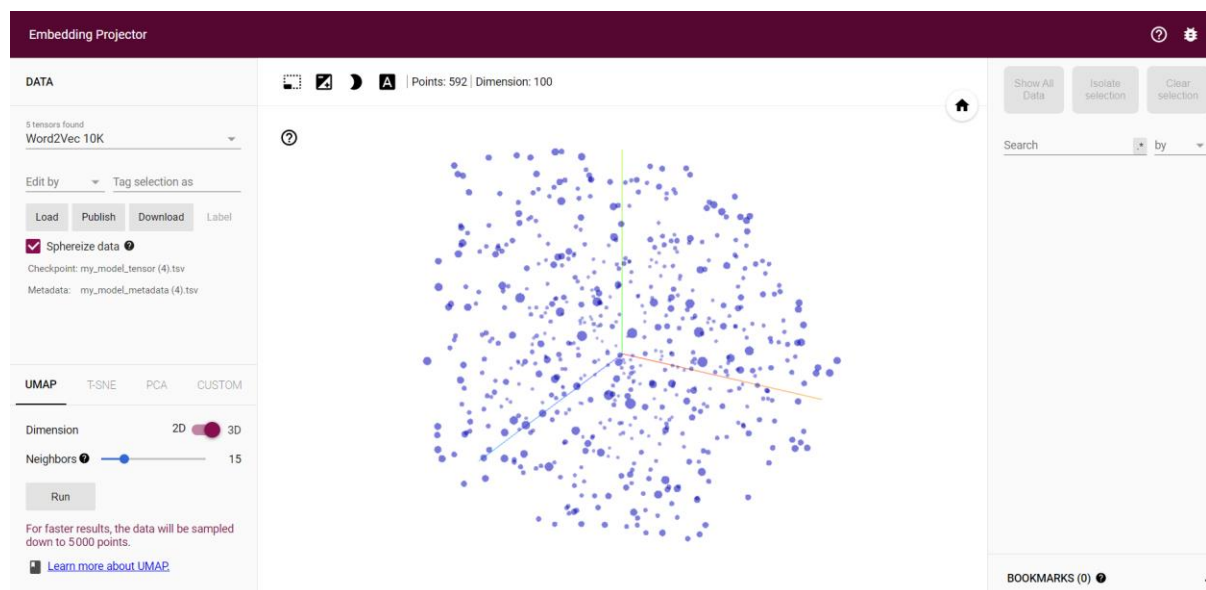
Sur les vulgarisations 'req' :

Word2Vec Embeddings - 3D t-SNE Visualization with Clusters





Les embeddings générés par word2vec sont déjà nettement plus pertinents, mais évidemment pas parfait. Nous avons continué notre exploration avec **tensorflow projector**, avec un **UMAP** au lieu de **TSNE**. La distribution est assez similaire à celle de tSNE.

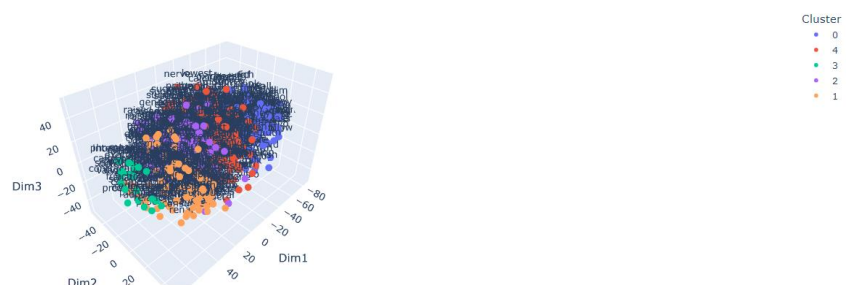


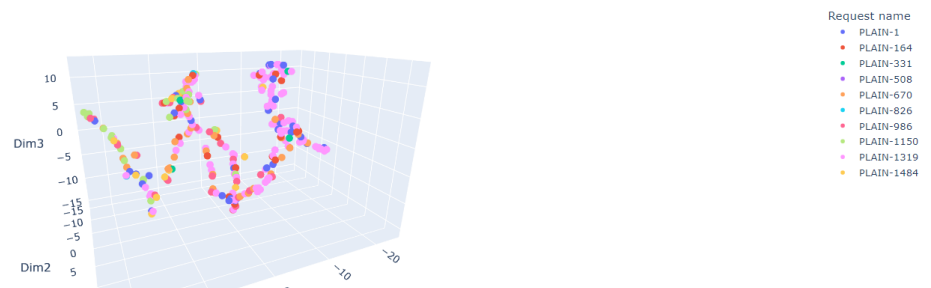
- 4.3 - FastText embeddings

FastText est une bibliothèque d'apprentissage des mots et de classification de textes créée par le laboratoire de recherche en IA de Facebook (FAIR). Nous avons voulu la tester.

Pour les documents 'doc' :

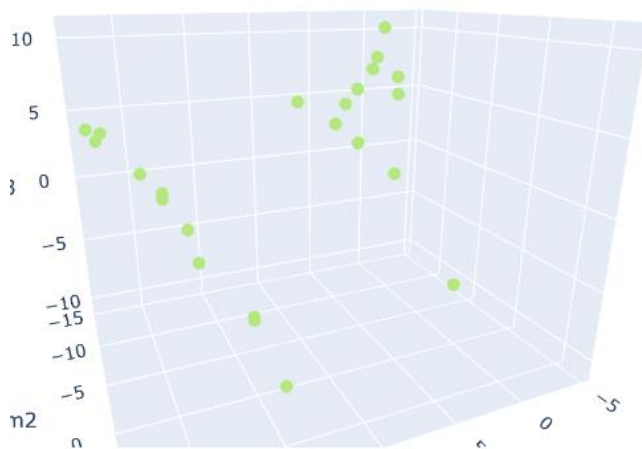
FastText Embeddings - 3D t-SNE Visualization with Clusters





Nous avons associé à chaque document sa vulgarisation :

Plain 1150 :



Cela nous a confirmé que les embeddings de FastText pouvaient être une approche intéressante pour la suite.

- 5.3 - Sbert embeddings

SentenceTransformers est un package Python pour l'intégration de phrases, de textes et d'images à la pointe de la technologie. Il sert notamment à créer des embeddings dans 100 langues différentes et c'est ce que nous avons fait.

Nous avons choisi d'utiliser le modèle sentence-transformers/paraphrase-MiniLM-L6-v2 via Hugging Face.

Il s'agit d'un modèle de transformateur de phrases : Il fait correspondre les phrases et les paragraphes à un espace vectoriel dense de 384 dimensions et peut être utilisé pour des tâches telles que le regroupement ou la recherche sémantique.

Sur les documents entiers :



Les résultats sont pas mal dutout et semblent meilleur que FastText au niveau du regroupement. Il ne reste plus qu'à tester nos modèles.

6. Evaluation de nos modèles

- 6.1 Méthodes

Nous avons notre méthode principale 'evaluateSimilarity' qui appelle de nombreuses fonctions. Le fonctionnement est le suivant

- Appel des fonctions de Load du corpus et sélection de la portion désirée
- Tokenisation avec nos méthodes de preprocessing prenant en argument les choix de l'utilisateur (traiter les liens, lemmatiser ou non, mettre une liste de mot à supprimer ou non)
- L'utilisateur choisit la méthode d'embeddings en paramètre. Il a le choix entre nos 2 meilleures : FastText et sbert.
- Quelle que soit la méthode choisie, nous créons le modèle associé et nous créons les embeddings. Ensuite nous faisons la similarité du cosinus entre les embeddings des documents et vulgarisations pour les classer, et calculer le ndgc5.

Exemple sur les 200 premiers documents, avec sbert :

```
score = evaluateSimilarity(0,200,method='sbert',lem=True,link=False)
score
(85, 384)
0.8824086793035104
```

Observations rapides : Les embeddings de SBert sont **meilleurs** que ceux de fastText et BM25 sur 0, 200. Par exemple :

```
scoreSbert = evaluateSimilarity(0,200,method='sbert',lem=True,link=False)
scoreFastText = evaluateSimilarity(0,200,method='fasttext',lem=True,link=False)
scoreBM25 = ndgcScore_bm25(0,200)

print("Score on 0 -200 documents - Sbert : ", scoreSbert)
print("Score on 0 -200 documents - FastText : ",scoreFastText)
print("Score on 0 -200 documents - BM25 : ",scoreBM25)

Score on 0 -200 documents - Sbert : 0.8824086793035104
Score on 0 -200 documents - FastText : 0.39329361136569674
Score on 0 -200 documents - BM25 : 0.8072123842662229
```

Maintenant nous allons pousser beaucoup plus nos analyses.

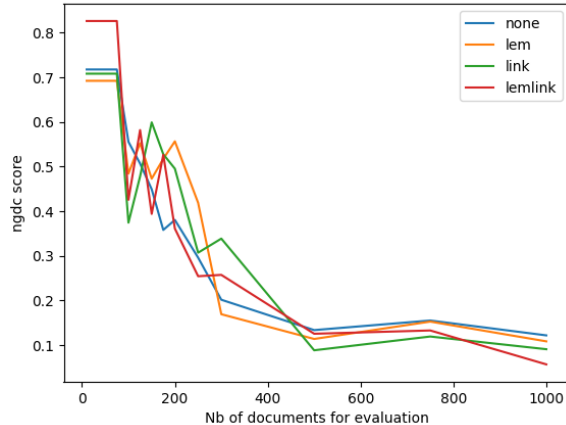
- 6.2 Evaluation selon différents processing

****Pour FastText :**

Nous avons évalué nos méthodes de processing pour mesurer l'impact sur le score final en utilisant les embeddings de FastText d'abord. Nous avons observé l'évolution du score avec différents prétraitements : Aucun, lemmatisation, suppression des liens, ou les deux.

Le meilleur modèle est celui qui utilise la lemmatisation à termes.

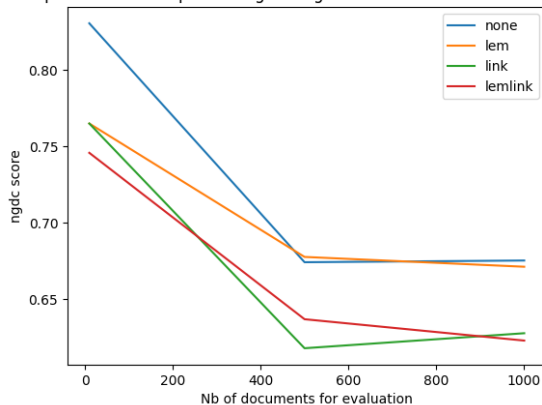
Impact of different processings on ngdc score with FastText Embeddings



```
mean none 0.40813284970481095
mean lem 0.4321129410440689
mean link 0.4259877528641878
mean lemlink 0.42991765517121705
```

****Pour Sbert :**

Impact of different processings on ngdc score with SBERT Embeddings



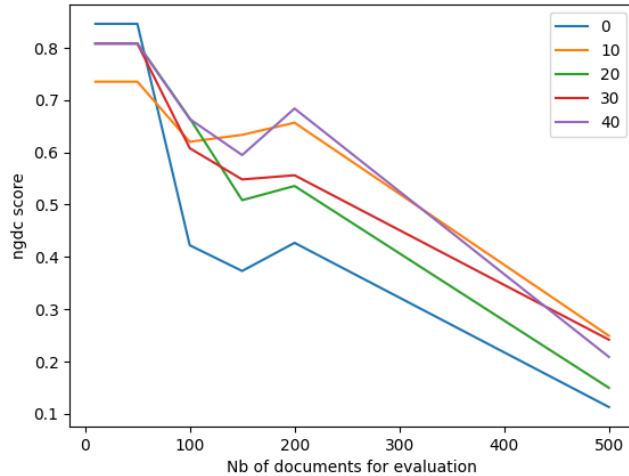
```
mean none 0.726519901682877
mean lem 0.704448669682436
mean link 0.6700167087913602
mean lemlink 0.668346033542138
```

Sbert donne des résultats bien plus convaincant, et finalement nous montre que lui, contrairement à FastText, a des meilleurs résultats sans du preprocessing rajouté.

Maintenant, visualisons si on supprime les mots les plus fréquents :

****Pour FastText :**

Impact on removing a list of n most frequent words in documents

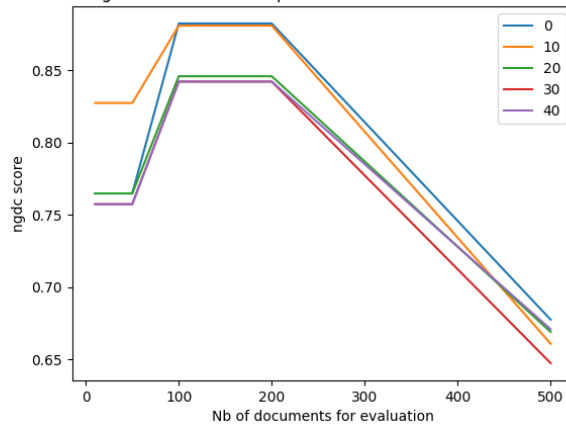


```
mean 0 : 0.5044388977961296
mean 10 : 0.6049570803837552
mean 20 : 0.5791769079220858
mean 30 : 0.595031154571167
mean 40 : 0.6278902959262239
```

Supprimer les 10 mots les plus fréquents semble le meilleur choix selon le graphique pour FastText.

****Pour sbert :**

Impact on removing a list of n most frequent words in documents with SBERT embeddings

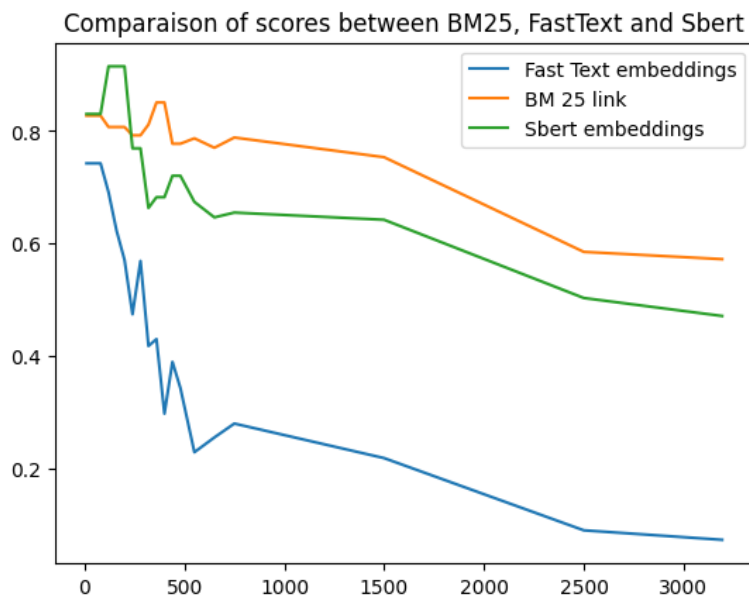


```
mean 0 : 0.8090559791867412
mean 10 : 0.8263811092674738
mean 20 : 0.7893840752560316
mean 30 : 0.7814485267502344
mean 40 : 0.7853515869214444
```

Pour Sbert, les résultats sont bien meilleurs et c'est également avec les 10 mots les plus fréquents. Nous avons fait de même pour les vulgarisations et les résultats sont similaires (voir notebook).

- 6.2 Evaluation finale selon différents embeddings

Enfin, nous avons donc tester pour l'ensemble des documents (de 1 à 3000 avec un pas de 40 par 40). Voici la comparaison entre le BM25 classique, FastText et Sbert :



Comme on peut le constater, FastText est bon seulement au début puis le score chute rapidement. A l'inverse, Sbert s'en sort très bien en étant même meilleur que BM25 au départ de 0 à 200 documents.

```
scoreSbert = evaluateSimilarity(0,200,method='sbert',lem=True,link=False)
scoreFastText = evaluateSimilarity(0,200,method='fasttext',lem=True,link=False)
scoreBM25 = ndgcScore_bm25(0,200)

print("Score on 0 -200 documents - Sbert : ", scoreSbert)
print("Score on 0 -200 documents - FastText : ",scoreFastText)
print("Score on 0 -200 documents - BM25 : ",scoreBM25)

Score on 0 -200 documents - Sbert : 0.8824086793035104
Score on 0 -200 documents - FastText : 0.39329361136569674
Score on 0 -200 documents - BM25 : 0.8072123842662229
```

Puis vers 200 documents il chute et reste légèrement BM25 avec un écart de 0.05 (BM25 est à 0.65 pour 3000 documents, tandis que Sbert est à 0.60 environ). Nous avons donc exploré de nombreuses solutions et réalisé de nombreuses analyses pour en arriver là. Avec du recul, nous pourrions améliorer de nombreuses choses, de part de nouvelles idées de preprocessing ou par l'utilisation de modèles plus performant que le paraphrase-MiniLM-L6-v2 par exemple.

7. User trial

Nous avons créé une petite section dans laquelle l'utilisateur peut taper une vulgarisation et on retrouve le document associé. Exemple testé avec 'deep fried foods may cause cancer...'

```
result

{'MED-4070': 0.6866002, 'MED-1151': 0.6112455, 'MED-3557': 0.60011595}
```