

|  |
| --- |
| *Rapport TP*  **Recherche d’information** |

KHELFA Celia

MEDDAD Farid

BOURAHLA AHMED ZINE EDDINE

Année universitaire : 2020/2021

**Sommaire :**

[Introduction 4](#_Toc66611603)

[Objectifs du TP 4](#_Toc66611604)

[1-phase d’indexation 5](#_Toc66611607)

[a-Construction de structures de documents avec fréquences 5](#_Toc66611608)

[b- Fichier inverse de fréquence : 6](#_Toc66611624)

[c- Construction de fichier inverse de poids : 7](#_Toc66611628)

[d-fonction d’accès 9](#_Toc66611632)

[Critiques de la phase d’indexation 9](#_Toc66611633)

[2- Modèle booléen 9](#_Toc66611634)

[3- Modèle vectoriel 11](#_Toc66611635)

[Evaluation des quatre formules afin de déterminer quelle est la meilleure 12](#_Toc66611636)

[Calcul du rappel et précision pour chaque requête 16](#_Toc66611637)

[4- Expérimentation (Par rapport à la taille) 16](#_Toc66611645)

[5-Comparaison des résultats du modèle vectoriel avec ceux du modèle booléen 17](#_Toc66611646)

[**Points de comparaison** 17](#_Toc66611647)

[Analyse et discussion des résultats 18](#_Toc66611659)

[Conclusion 18](#_Toc66611660)

# Introduction

Le processus de la recherche d’information peut être résumé en trois étapes qui sont la représentation de la collection des documents (corpus), l’interrogation initiée par l’utilisateur (compréhension et traitement des requêtes) et enfin la récupération des documents pertinents.

De ce fait, il s’agit de rendre le volume d’information contenu dans des corpus de documents numériques accessible à l’utilisateur d’une manière rapide, et concise. Ce qui permettra de rendre sa quête vers l’information bien simple tout en remplissant ses besoins exprimés.

# Objectifs du TP

* Créer une collection de documents.
* Implémenter les différents fichiers d’indexation.
* Implémenter les fonctions d’accès.
* Réaliser les deux modèles booléen et vectoriel
* Evaluer les résultats des requêtes selon le rappel et la précision
* Comparaison entre les modèles SRI implémentés, ainsi qu’entre les quatre fonctions du modèle vectoriel.

# Structure des fichiers et algorithmes implémentés

Pour ce faire, nous allons proposer une solution fonctionnelle où chaque module aura un rôle bien défini. Nous allons présenter les modules plus bas. Les index, une fois construits, seront sauvegardés dans des fichiers afin d’éviter la perte de temps à chaque démarrage du programme.

# 1-phase d’indexation

Pour la phase d’indexation nous allons présenter dans le rapport les informations suivantes :

### a-Construction de structures de documents avec fréquences

C’est un dictionnaire avec la structure ci-dessous pour chaque élément :

**Def Struct : IndexElement**

**NumDoc** : Clé .

**ListeMots :** listede motsavec chaque mot associé à sa fréquence.

|  |  |
| --- | --- |
| NumDoc1 | <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence>.  <Mot, fréquence> |
| NumDoc2 | <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence> |
| ……… |  |
| NumDocN | <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence>  <Mot, fréquence>.  <Mot, fréquence> |

* Taille de l’index en nombre d’entrée et en octet

|  |  |
| --- | --- |
| Termes | Freq. |
| subtractions | 2 |
| ………. |  |
| program | 4 |

Max 20 Octets

4 Octets

Donc 24 octets/terme pour 87486 termes trouver dans 3204 documents

|  |
| --- |
| **1135533**  bytes |

* durée d’indexation (temps)

|  |
| --- |
| **=0,50667 s** |

**L’index est conservé dans <Index.pickle>**

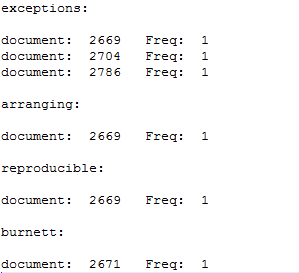
### b- Fichier inverse de fréquence :

En ce qui concerne le fichier inverse du document de fréquence, il contient les mots un par un avec tous les documents où ils figurent. Comme le montre la capture ci-dessous :

**Def Struct : IndexFrequence**

**Clé** : Mot .

**ListeDoc\_Freq :** liste(document,frequence(Mot)).



* Taille de l’index en nombre d’entrée et en octet

Donc pour on a 87486 (doc,freq)listes

|  |
| --- |
| **939569**  bytes |

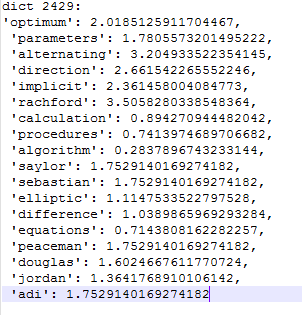
* durée d’indexation (temps)

|  |
| --- |
| **=** **0.17366s** |

**L’index est conservé dans <IndexInvFrequence.pickle>**

### c- Construction de fichier inverse de poids :

La construction d’une structure de donnée avec poids revient au même travail déjà fait lors de la construction de la structure de documents avec fréquences. Pour chaque document, ses mots sont affichés suivis de leurs poids respectifs. Voici ci-dessous un exemple d’exécution de cette fonction :



La formule de calcul du poids étant :

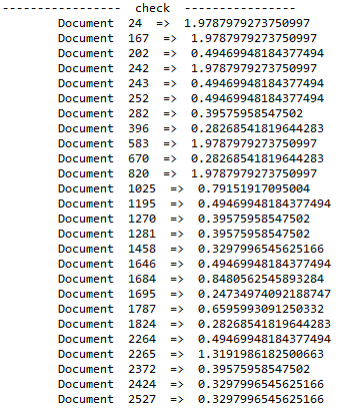
poids(ti, dj)=(freq(ti,dj)/Max(freq(dj))\*Log((N/ni) +1)

Le fichier inverse de poids a la même utilité que le fichier inverse de fréquence. Il affiche pour chaque mot, les documents où il figure ainsi que le poids. La capture ci-dessous le montre :

**Def Struct : IndexInvPond**

**Clé** : Mot .

**ListeDoc\_Freq :** liste(document,Poids(Mot))



* Taille de l’index

|  |
| --- |
| **1119989**  bytes |

* durée d’indexation (temps)

|  |
| --- |
| **=** **72.81650** |

**L’index est conservé dans <** **fichier\_structure\_poid.pickle> and <fichier\_inverse\_poid.pickle>**

## d-fonction d’accès

Afin de connaitre quels sont les mots appartenant à un document ainsi que leurs fréquences et leurs poids, et les documents comportant un mot donné avec ses fréquences set poids dans chacun deux, nous avons implémenté deux fonctions :

|  |
| --- |
| def focntionacces\_doc(i,index): retourne liste de mots avec leurs fréquences   def focntionacces\_mot(mot, index\_inv): retourne liste de mots avec leurs fréquences |

//accès rapide par l’utilisation de la structure <disct>

# Critiques de la phase d’indexation

\*On peut pas traiter les fautes d’autographes (on les reconnaît pas dans l’indexation)

Donc on peut ajouter un module pour traiter ce cas.

\*la difficulté de traiter les mots composer et faire la différence entre « computer science »

And « computer-science ».

\*on peut utiliser des metaheuristics pour améliorer l’indexation.

\*perte d’information (prénom d’auteur,….)

# 2- Modèle booléen

Le modèle booléen reçoit une requête composée de plusieurs mots en plus des connecteurs « et », « ou ».. etc. Tout d’abord, chaque mot de la requête reçue en entrée sera transformé en 1 ou 0 selon son existence dans les documents ou pas. Ensuite, nous avons utilisé la fonction prédéfinie eval() qui prend en entrée les 0 et 1 séparés par les connecteurs logiques « and, or, not » et nous retourne 0 ou 1 pour la liste des documents.

|  |
| --- |
| Variables :  *Entrée : Index, requête*  *Sortie : Docs*  Début :   * On initialise ListeDoc avec une liste vide. * Pour chaque Doc dans Index : * On récupère les mots du document à partir de l’index. Ensuite, on transforme notre requête sous format booléen comme suit. * On tokenize la requête en liste de mots à l’aide de la fonction word-tokenize de nltk. * Pour chaque Mot dans la requête : * On initialise Req à vide. * Si le mot n’est pas dans l’ensemble [‘(‘,’)’,’and’, ‘or’,’not’,’&’,’|’] * Si le mot n’est pas dans le document on met à sa place dans la liste un 0 et un 1 s’il y est. * Puis, on regroupe la liste résultante dans une chaine de caractères avec un espace comme séparateur dans Req. * On évalue cette chaîne grâce à la fonction eval(Req), Si elle retourne 1 on prend le document courant. |

**Exemple de requête dans le modelé booléen**

**Requête 1 : (Avec parenthèses)**

requete = '((computation or exclusion) and collection) or synchronization';

**Liste documents :**

**[1747, 1828, 1844, 1854, 2023, 2321, 2342, 2376, 2578, 2597, 2723, 2738, 2777, 2865, 2895, 2912, 2938, 2946, 3039, 3082, 3128]**

**Temps**

**0.34652963299999984 s**

**Requête 2 : (sans parenthèses)**

R2 = 'life and test or not open and books';

**Liste documents :**

**[397]**

**Temps**

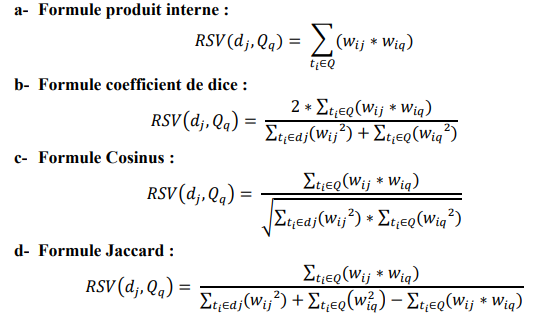
**0.32757 s**

**Remarque et critiques**

|  |  |
| --- | --- |
| **\* inconvénient de ce modèle**  Nous ne pouvons pas ordonner les documents afin de définir lequel est plus important. | **\* Avantage**  Ce modèle est simple et rapide |

# 3- Modèle vectoriel

Le modèle vectoriel reçoit la requête sous forme d’ensemble de mots pondérés avec 1. Il nous retourne la liste des documents ordonnée avec les similarités qui se différencient selon la formule utilisée. Nous avons testé quarte formules :

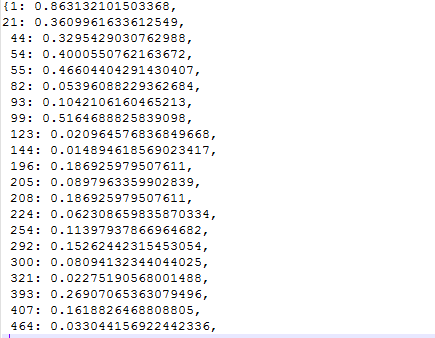


Exemple d’exécution du modèle vectoriel (avec formule cosinus par exemple):

Requete="comunication language software algebraic preliminary"

Num

Document



Valeur de similarités

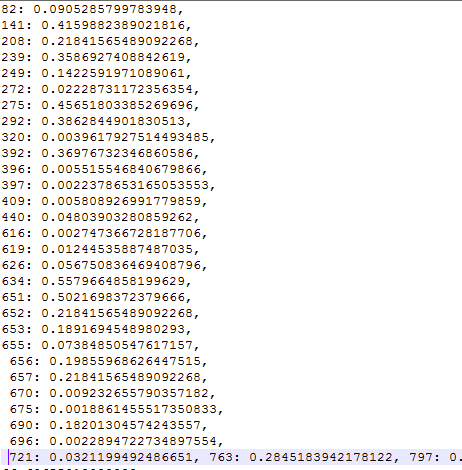
Temps=0.08103008 s

## Evaluation des quatre formules afin de déterminer quelle est la meilleure

Nous avons effectué une évaluation sur chacune des formules afin de déterminer quelle est la meilleure et l’utiliser pour l’expérimentation.

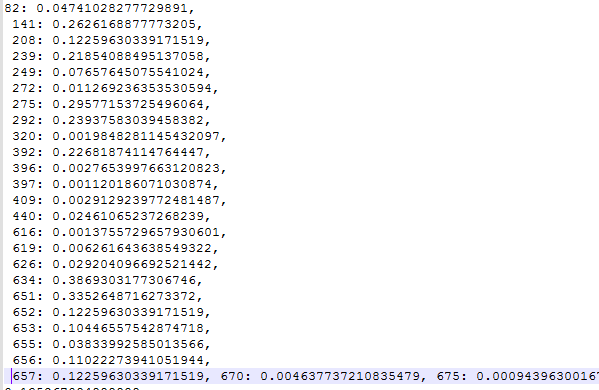
Nous allons utiliser la requête numéro 50 comme exemple

**La formule du coef de dice :**



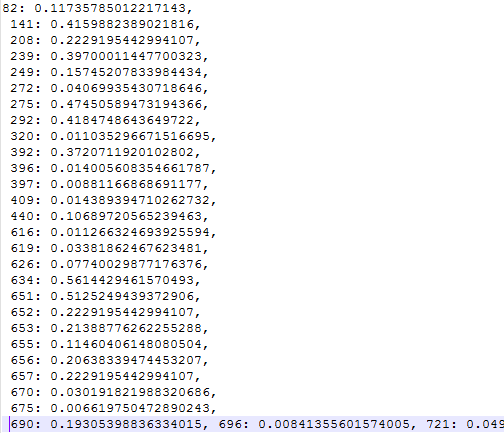
Temps=66.6065531 s

**La formule Jaccard :**



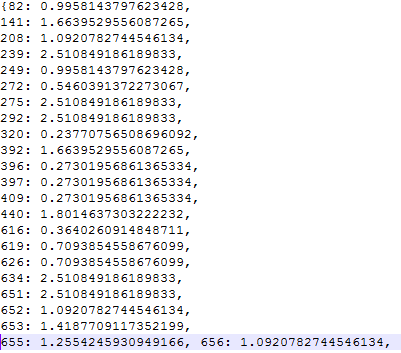
Temps : 0.13526709 s

**La formule cosinus :**



Temps=0.1324958

**La formule produit interne**



Temps=0.08107174 s

Pour notre corpus, nous avons remarqué que la fonction « Inner product » retournait souvent les plus mauvais résultats. Les documents les plus pertinents peuvent avoir un taux de similarité inférieur à celui d’autres documents moins pertinents. En revanche la fonction « mesure de cosinus », d’après les tests que nous avons effectués, retourne le meilleur taux de similarité (les documents les plus pertinents ont la plus grande similarité)

## Calcul du rappel et précision pour chaque requête

## On a calculé le rappel et précisons pour chaque requête du fichier « query.txt »

Avec les 4 mesures du model vectoriel.

Les résultats sont enregistrés dans les 4 fichiers ci déçus :

<< execution\_formule\_cosinus.pickle>>

<< execution\_formule\_jaccord.pickle>>

<< execution\_formule\_coeif\_de\_dice.pickle>>

<< execution\_produit\_interne.pickle>>

**Def Struct : RapPres**

**NumDoc** : num requete .

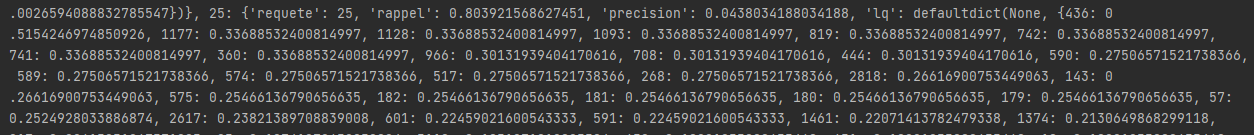
**Listeinfo :{rappel,precision,sorted(**réponse\_mesure**)}**.

lq

rappel

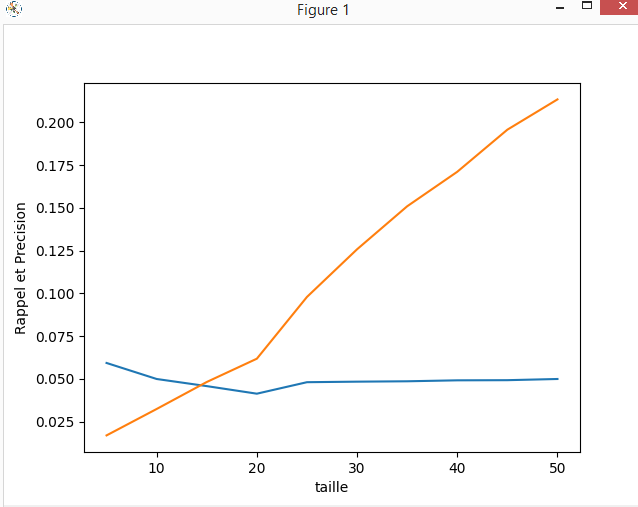
Num requête

precision



# 4- Expérimentation (Par rapport à la taille)

Nous avons calculé le lq de chaque requête avec la formule cosinus. Ensuite, nous avons calculé le rappel et la précision pour chaque requête en utilisant lq et dq. Nous avons remarqué que les résultats ne sont pas équilibrés, l’un est meilleur que l’autre. Donc, nous avons fait une expérimentation afin de déterminer la taille pour lesquels les résultats sont les meilleurs. Pour faire l’expérimentation pour une requête Q, supposons que la taille de lq est de 50 documents. Alors nous commençons par calculer le rappel et la précision de cette requête avec une taille égale à 10. Ensuite, nous recalculons pour une taille égale à 20 puis à 30 et enfin à 50. Le graphe ci-dessou illustre l’expérimentation de taille pour le rappel et la précision.



Du graphe ci-dessus, nous avons fixé la taille à 12.

Les résultats sont enregistrés dans les fichiers ci déçus :

**< evaluation\_mean\_prec.pickle>**

**< evaluation\_mean\_rappel.pickle>**

# 

# 5-Comparaison des résultats du modèle vectoriel avec ceux du modèle booléen

|  |  |  |
| --- | --- | --- |
| **Points de comparaison** | **Booléen** | **Vectoriel** |
| Requête | Nécessite un format formel. | Elle peut être non-précise. |
| Documents | La correspondance est stricte entre le document et la requête ,il n’y pas de distinction entre les documents pertinents. | Les documents sont très distincts. Et la correspondance est à un certain taux de similarité.. |
| Format de la requête | Le format est complexe et nécessite certaines connaissances. | Le format est simple pour l’utilisateur et utilisable même s’il n’a pas une idée fixe sur ses besoins les documents pertinents peuvent être trouvés. |

# Analyse et discussion des résultats

**Avantage :**

* L’indexation des documents permet de réduire la complexité.
* Le prétraitement des données ne garder que les mots reflétant un contenu informationnel. Ceci rend la recherche rapide et précise.
* Les deux points précédents rendent le temps de réponse instantané pour les requêtes et l’évaluation.

**-** Pour le modèle vectoriel, et en utilisant TF-IDF pour le calcul des poids, l’analyse des résultats de l’évaluation nous a menés à observer que la mesure cosinus retourne le plus généralement des documents assez pertinents. Mais il existe pour certaines requêtes ou le rappel du produit interne fut plus élevé que celui du cosinus.

**-** Pour le modèle booléen, les résultats étaient assez exacts et la correspondance entre la requête et le document était très stricte ce qui donne des documents assez ressemblants.

**Améliorations** **possibles :**

**-** On pourrait aussi utiliser d’autres méthodes de mesure.

- procéder à indexation plus exhaustive pour améliorer le résultat du rappel et précision.

-on peut aller plus loin dans l’étape de l’indexation et ne pas s’arrêter sur la stopliste et suppression des ponctuations.

# Conclusion

Le projet nous a permis d’approfondir nos connaissances dans le domaine de la recherche d’informations car c’est un pilier du domaine de l’intelligence artificielle. Il est donc nécessaire d’améliorer ses résultats en s’intéressant aux besoins non exprimés de l’utilisateur pour augmenter la pertinence utilisateur/ machine.