

UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE



RECHERCHE D'INFORMATION

Rapport du mini projet Recherche d'information

Rédaction:

MOULAI HASSINA SAFAA

Matricule : 201400007564

HOUACINE NAILA AZIZA

Matricule : 201400007594

M2 SII Groupe:3

Professeur

Mr. KECHID Samir

December 20, 2018

Contents

1	Collecte de document	3
1.1	Pseudo-code	3
2	Indexation des documents	4
2.1	Structure de données	4
2.2	Pseudo-code	4
2.3	Explication de l'algorithme	4
2.4	Résultat	5
3	Pondération des termes	6
3.1	Structure de données	6
3.2	Pseudo-code	6
3.3	Explication de l'algorithme	7
3.4	Résultat	7
4	Fonctions d'accès	8
4.1	Recherche les termes(fréquence/poids) d'un Document donné	8
4.2	Recherche les documents(fréquence/poids) contenant un terme donné	9
5	Appariement Requête-Document	11
5.1	Modèle Booléen	11
5.2	Modèle Vectoriel	14
5.3	Modèle Probabiliste avec échantillon	15
6	Bonus	19
6.1	Modèle Probabiliste sans échantillon	19
6.2	Modèle Probabiliste BM25	20
7	Comparaison des trois modèles	21

List of Figures

1	Fichier Inverse de notre collection	5
2	Fichier Inverse de notre collection	7
3	fonction d'accès à travers un document	9
4	Fonction d'accès à travers un terme	10
5	Résultat du modèle booléen.	13
6	Vérification du résultat du modèle booléen.	13
7	Affichage de l'échantillon et sélection de l'utilisateur.	17
8	Résultat du modèle probabiliste.	18
9	Résultat du modèle probabiliste sans échantillon.	19
10	Résultat du modèle BM25.	20

Introduction

La recherche d'information est apparue comme domaine d'intérêt de nombreux informaticien et mathématicien peu après la création des premiers ordinateur,

Mais c'est surtout avec l'explosion de ressource d'information et de contenus textuelle, images, vidéo,... disponibles sur le web principalement, que la recherche d'information est devenu primordiale pour tous système de grande échelle, tel que les données ont toujours besoins d'être filtré ou trié par ordre de pertinence selon le besoin,

Aussi afin de restreindra à l'essentiel les données dont nous avons besoin ou de retrouver une information dans une grande masse de données, tout cela afin d'obtenir de meilleurs résultat et d'optimiser les étapes suivante de certain système.

La recherche d'information est très largement utilisé aussi bien dans le e-learning, les ontologies, construction de dataset, traitement automatique du langage (TALN) ... etc. mais pas que cette technologie est aussi très présente et très demandé dans les domaines tel que le droit, les langues, ...etc.[1]

Objectifs

Ce Projet à pour objectif de nous familiariser avec les procédures de base en recherche d'information (RI) et des différentes étapes et méthode qu'elle couvre, tel que l'indexation des documents, la pondération des termes et l'appariement requête-document principalement en utilisant: le modèle Booléen, le modèle Vectoriel et le modèle Probabiliste.

1 Collecte de document

Comme source de données nous avons choisi le **brown corpus** de **nltk** disponible en open source, et cela pour sa richesse et sa diversité car la recherche d'information est appliquée sur de larges collections de documents de diverse provenance, aussi cette bibliothèque offre déjà le liste des **stopwords** complète.

1.1 Pseudo-code

L'auto-cr ation de notre r pertoire pour la collection de documents    tudier.

Algorithm 1 CR ATION DU R PERTOIRE

Input: localpath : chemin vers le dossier   cr er,
name : nom du dossier   cr er

Output: Le r pertoire est cr e

```
1 dir_name ← Concatenner(localpath+" "+name)
2 try:
    s.mkdir(dirName)
3 except FileExistsError:
    print("Directory ", dirName, " already exists")
```

L'automatisation du remplissage de nos documents   partir du contenu des documents du brown corpus.

Algorithm 2 CREATION DE N DOCUMENTS

Input: directory : chemin vers le dossier ou cr er les documents,
n : nombre de documents   cr er

Output: Les documents cr er et remplis

```
4 liste ← nltk.corpus.gutenberg.fileids()
5 for element ∈ liste do
6     path_element ← Concatenner(directory+" "+element)
        f ← Ouvrire_en_ecriture(path_element)
        Ecrire(f,Convertir_en_text(nltk.corpus.gutenberg.words(element))[:60])
        Fermer(f)
```

2 Indexation des documents

un index inversé est une correspondance entre du contenu, comme des mots ou des nombres, et sa position dans un ensemble de données comme un enregistrement en base de données, un document ou un ensemble de documents ; sur le même principe qu'un index terminologique. Le but de l'index inversé est de permettre une recherche plein texte plus rapide, contre un temps d'insertion de nouvelles données augmenté. [wiki]

2.1 Structure de données

On a utilisé pour indexer nos termes appartenant à un document , on opté pour un dictionnaire de dictionnaire comme suit:

```
Fichier-Inverse : {  
    terme1 : {doc1 : freq , doc2 : freq , ...},  
    terme2 : {doc1 : freq , doc2 : freq , ...},  
    ...  
}
```

- Le premier dictionnaire a pour clé le **terme** et comme valeurs un dictionnaire de **document** : **fréquence** tel que la fréquence est celle du terme dans le document.

exemple

2.2 Pseudo-code

Algorithm 3

Input: listDocs : liste des documents

Output: FichierInverseFreq

```
7 Fichier_Inverse ← New Dictionary(term,New Dictionary(Doc,Frequency)))  
8 for doc ∈ listDocs do  
9   for terme ∈ doc do  
10    Fichier_Inverse[terme][doc]←Fichier_Inverse[terme][doc]+1;  
11 return InverseFichiers
```

2.3 Explication de l'algorithme

La création du fichier inverse se fait comme suit:

- Parcours des documents un par un
- Pour chaque document on parcourt ses termes
- Pour chaque pair de terme document on incrémente la fréquence en accédant à notre dictionnaire avec les deux pair de clés (TERME , DOCUMENT).

2.4 Résultat

ainsi notre fichier inverse peu etre représenté par une matrice dont les lignes sont les termes et les colonnes les documents, puis chaque cellule C_{ij} contient la fréquence du terme t_i de la i^{eme} ligne dans le document d_j de la j^{eme} colonne.

Terme	2	3	4	5	6	7	8	9	10	11	12
1											
2	1	1	1	1	1	1	1	1	0	1	1
3	2	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0
5	1	0	0	0	0	0	1	0	0	0	0
6	1	1	1	1	1	1	1	1	0	1	1
7	1	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	1	0	0	0	0
9	1	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0
13	1	0	0	1	0	0	0	0	0	0	0
14	1	0	0	0	0	2	0	0	0	0	0
15	1	0	0	0	0	0	0	0	0	0	0
16	1	0	0	0	0	0	0	0	0	0	0
17	1	0	0	0	0	0	0	0	0	0	0

Figure 1: Fichier Inverse de notre collection

3 Pondération des termes

Il est souvent utile de donner plus d'importance à certain terme lors d'un processus de RI, pour cela plusieurs approche existent.

Parmi les approches les plus répondu pour la pondération des termes d'une collection de document nous retrouvons **TF*IDF**, dont la formule est la suivante:

$$poids(t_i, d_j) = ((fichierInverse[t_i][d_j])/Max) * \log((N/N_i) + 1)$$

Avec:

t_i le i^{eme} terme.

d_j le j^{eme} document.

Max : fréquence maximal du terme dans le document.

N_i : nombre de Documents ou apparait le terme t_i .

N : le nombre de document de la collection.

3.1 Structure de données

La structure de notre **fichier inverse pondéré** est identique à celui du fichier inverse des fréquence à la différence que chaque cellule contient un **poids** et non pas une fréquence.

```
Fichier-Inverse-pondéré : {  
  terme1 : {doc1 : poids , doc2 : poids , ...},  
  terme2 : {doc1 : poids , doc2 : poids , ...},  
  ...  
}
```

3.2 Pseudo-code

Algorithm 4

Input: listDocs : liste des documents

Output: fichierInversePoids

```
12 Fichier_Inverse_Poids ← New Dictionary(term,New Dictionary(Doc,Poids)))  
13 for doc ∈ listDocs do  
14   for terme ∈ doc do  
15     poids ← TFIDF()  
     Fichier_Inverse_Poids[terme][doc] ← poids;  
16 return InverseFichiersPoids
```

3.3 Explication de l'algorithme

- pour chaque terme de la collection.
- pour chaque document de la collection.
- Calculer le fonction **TF*IDF** et l'affecter au (Terme,Document) correspondant.
- retourner le fichier Inverse des Poids.

3.4 Résultat

Notre fichier inverse des poids est aussi affichable sous forme de matrice [terme,Document]
 ⇒ Poids, comme dans la figure qui suit:

Terme	2	3	4	5	6	7	8	9	10	11	
1	austen-emm...	whitman-lea...	melville-mob...	burgess-bust...	bible-kjv.txt	shakespeare...	austen-sense...	milton-paradi...	hello.txt	chesterton-b...	ches...
2	[0.360273077...	0.360273077...	0.360273077...	0.240182051...	0.360273077...	0.360273077...	0.720546154...	0.360273077...	0	0.720546154...
3	emma	2.995732273...	0	0	0	0	0	0	0	0	0
4	jane	0.996215082...	0	0	0	0	0	1.992430164...	0	0	0
5	austen	0.996215082...	0	0	0	0	0	1.992430164...	0	0	0
6]	0.360273077...	0.360273077...	0.360273077...	0.240182051...	0.360273077...	0.360273077...	0.720546154...	0.360273077...	0	0.720546154...
7	volume	1.497866136...	0	0	0	0	0	0	0	0	0
8	chapter	0.874599927...	0	0	0	0	0	1.749199854...	0	0	0
9	woodhouse	1.497866136...	0	0	0	0	0	0	0	0	0
10	handsome	1.497866136...	0	0	0	0	0	0	0	0	0
11	clever	1.497866136...	0	0	0	0	0	0	0	0	0
12	rich	1.497866136...	0	0	0	0	0	0	0	0	0
13	comfortable	1.175687628...	0	0	0.783791752...	0	0	0	0	0	0
14	home	1.175687628...	0	0	0	2.351375257...	0	0	0	0	0
15	happy	1.497866136...	0	0	0	0	0	0	0	0	0
16	disposition	1.497866136...	0	0	0	0	0	0	0	0	0
17	seemed	1.497866136...	0	0	0	0	0	0	0	0	0

Figure 2: Fichier Inverse de notre collection

4 Fonctions d'accès

Il est nécessaire de définir des fonctions d'accès pour n'importe quelle structure de données que nous définissons, c'est pourquoi les deux fonctions qui suivent nous permettent de manipuler les informations de nos fichiers inverse de fréquence et de poids.

4.1 Recherche les termes(fréquence/poids) d'un Document donné

Pour un document d saisi pas l'utilisateur on retourne la liste de ses termes avec leurs fréquences et leurs poids.

Pseudo-code

Algorithm 5

Input: document, fichier_Inverse

Output: listTermes

```
17 for  $terme \in Fichier\_Inverse$  do
18   for  $doc \in Fichier\_Inverse[terme]$  do
19     listTermes[terme]  $\leftarrow$  Fichier_Inverse[terme][doc]
20 return listTermes
```

Explication de l'algorithme

la fonction prend on paramètre le document dont nous voulons récupérer les termes, puis procède comme suit:

- pour chaque terme du fichier inverse.
- si le document contient ce terme on récupère la fréquence de ce terme.
- on retourne la liste de (terme,freq) du document en entrée.

Résultat

Après que l'utilisateur ne saisisse le nom du document donc il souhaite connaître les terme/fréquence/poids, il valide en cliquant sur "**OK**", les informations recherchées lui sont affichées comme suit:

Modèle Probabiliste			
Document			
austen-sense.txt			Ok
Termes			
	Terme	Fréquence	Poids
5	chapter	1	1.749199854809259
6	lived	1	1.992430164690206
7	long	1	1.749199854809259
8	sense	1	2.995732273553991
9	sensibility	1	2.995732273553991
10	family	1	2.995732273553991
11	dashwood	1	2.995732273553991
12	settled	1	2.995732273553991
13	sussex	1	2.995732273553991
14	estate	1	2.995732273553991

Figure 3: fonction d'accès à travers un document

4.2 Recherche les documents(fréquence/poids) contenant un terme donné

Pour terme donné on retourne la liste des documents dans les quels il apparait avec sa fréquences et son poids pour chaque documents.

Pseudo-code

Algorithm 6

Input: t_i , fichier_Inverse

Output: listDocs

```

21 for  $terme \in Fichier\_Inverse$  do
22   if  $terme == t_i$  then
23     for  $doc \in Fichier\_Inverse[terme]$  do
24       listDocs[doc]  $\leftarrow Fichier\_Inverse[terme][doc]$ 
25 return listDocs

```

Explication de l'algorithme

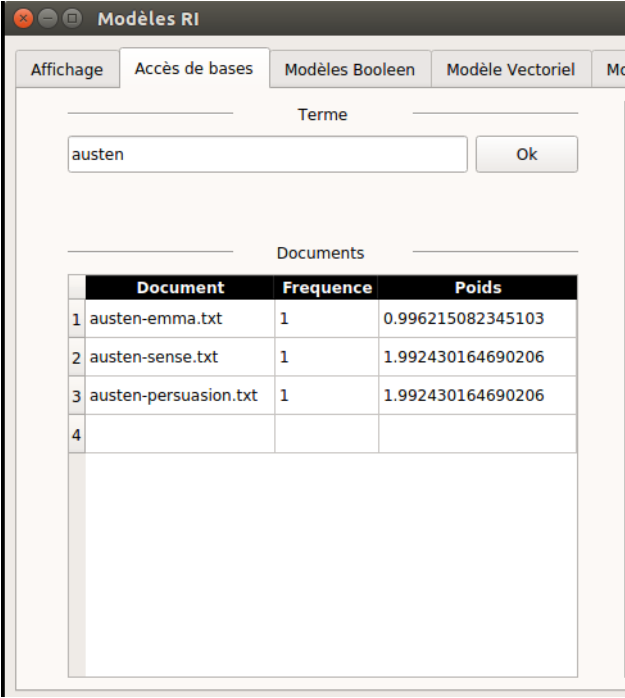
la fonction prend on paramètre le terme recherché t_i dont nous voulons récupérer les documents ou il apparait, pour cela nous procédons comme suit:

- pour chaque terme du fichier inverse.
- s'il s'agit du terme recherché.

- pour chaque document comportant ce terme ajouter le document/fréquence de ce terme dans le document en cours de traitement
- on retourne la liste de (doc,freq) du document en entrée.

Résultat

Un champs de saisi est mis à disposition de l'utilisateur pour y taper le terme dont il veut connaitre les documents oui il apparait et avec quelle fréquence et poids, puis il n'a plus qu'a valider sa recherche via le bouton "**OK**", les informations recherchées lui sont affichées comme suit:



The screenshot shows a window titled "Modèles RI" with several tabs: "Affichage", "Accès de bases", "Modèles Booleen", "Modèle Vectoriel", and "Modèle". The "Accès de bases" tab is selected. Below the tabs, there is a section labeled "Terme" with a text input field containing "austen" and an "Ok" button. Below this, there is a section labeled "Documents" containing a table with three columns: "Document", "Frequence", and "Poids". The table lists three documents: "austen-emma.txt", "austen-sense.txt", and "austen-persuasion.txt", each with a frequency of 1 and a weight of 0.996215082345103. There is also an empty row with index 4.

	Document	Frequence	Poids
1	austen-emma.txt	1	0.996215082345103
2	austen-sense.txt	1	1.992430164690206
3	austen-persuasion.txt	1	1.992430164690206
4			

Figure 4: Fonction d'accès à travers un terme

5 Appariement Requête-Document

Il s'agit de calculer à quelle degrés une requête correspond ou concorde avec le contenu d'un document, pour cela nous allons utiliser comme mesure la **similitude**, celle si peut être calculé selon diverses modèle telle que: **le modèle boolean, le modèle vectoriel, le modèle probabiliste avec ses différentes variantes, ...**

Chacun des modèles cités ci-dessus possède des avantages et inconvénients, c'est pour cela que nous allons les implémenter et tester sur un même ensemble de document et comparer les résultats obtenus.

5.1 Modèle Booléen

Le modèle booléen traite une requête comme une expression logique. tel que pour un document doc1 et un terme t1,nous pouvons évaluer la similitude comme étant vraie (Similaire) si le terme t1 existe dans le document doc1 et comme fausse (non similaire) dans le cas contraire.

Requête

Dans ce modèle nous définissons trois opérateurs AND, OR et NOT pour relier les termes de la requête.

exemple : requête = "t1 AND (t2 OR t3)"

Ainsi cette requête peut etre décomposé comme suit :

requête1 = t1

requête2 = t2

requête3 = t3

Avec :

Appariement(requête, doc) = Appariement(requête1, doc) AND (Appariement(requête2, doc)
OR Appariement(requête3, doc))

ainsi cela revient a évaluer une l'expression boolean composé de "True" et "False" ainsi que des opérateur logique de bases cités précédemment.

Fonction d'appariement : Pseudo-code

Algorithm 7 MODÈLE BOOLEAN

Input: *listDocs* : liste des documents,
 fichierInverse : le fichier inverse de notre ensemble de documents,
 requete

Output: *listDocsPertinents* : la liste des documents pertinents
(c'est-à-dire dont la similitude == 1)

```
26 for doc ∈ listDocs do
27   for terme ∈ requete do
28     if terme ∉ [AND, OR, NOT] then
29       if terme ∈ fichierInverse[doc] then
30         resultat ← 1
31       else
32         resultat ← 0
33     else
34       resultat ← terme
35   resultat = ConvertirEnChaineDeCara(resultat)
36   if Evaluation(resultat) == True then
37     Ineserer(doc, listDocsPertinents)
37 return listDocsPertinents
```

Explication de l'algorithme

notre algorithme effectue alors la séquence d'instruction suivante:

- pour chaque document de la collection.
- pour chaque terme de la requete différent des opérateurs (et, ou, not).
- rechercher le terme dans le fichier inverse pour le document en cours de traitement.
- s'il existe remplacer le terme par la valeur booléenne "*True*" par "*False*" sinon.
- Évaluer la nouvelle requête obtenu (composée de booléen et d'opérateur logique)
- retourner la liste de document dont l'évaluation a donnée "*True*"

Résultat

Pour le requete :

(jane and austen) and not volume

nous aurons comme résultat tous les documents de notre collection qui contiennent les mot : **jane** et **austen** mais qui ne contiennent pas le mot **volume**.

l'exécution de notre modèle booléen nous retourne 2 documents qui sont : **austen-sense.txt** et **austen-persuasion.txt**

comme le montre la figure suivante:

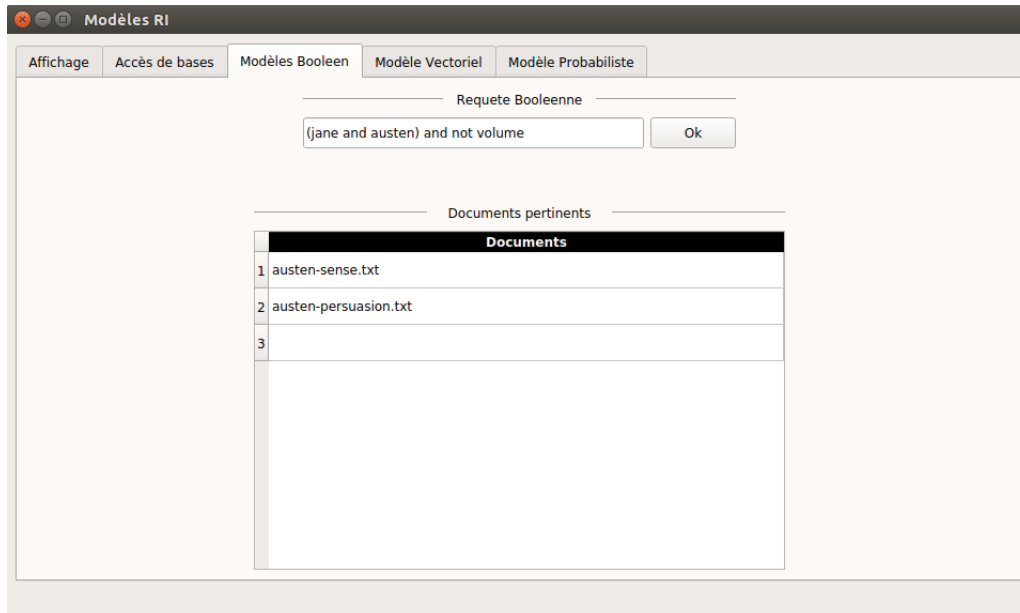


Figure 5: Résultat du modèle booléen.

Nous pouvons vérifier qu'effectivement ce sont les deux seules document vérifiant la requête de notre collection.

The screenshot shows the 'Fichier Inverse' table in the 'Modèles RI' application. The table has columns for terms and document IDs. The terms 'jane' and 'austen' are highlighted in blue, indicating they are part of the query. The table is as follows:

Term	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	austen-emma.txt	whi...	me...	bur...	bi...	sha...	austen-sense.txt	mi...	he...	ch...	ch...	sh...	br...	carr...	austen-persuasion.txt	edge...	cheste...	shakes...
2	[1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
3	emma	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	jane	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
5	austen	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
6]	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	volume	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	chapter	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	0	0
9	woodhouse	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	handsome	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	clever	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	rich	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	comfortable	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14	home	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
15	happy	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	disposition	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	seemed	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	unite	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6: Vérification du résultat du modèle booléen.

5.2 Modèle Vectoriel

Dans le modèle vectoriel les documents sont modélisant à travers un vecteur contenant tous les termes de la collection de tel façon à ce que si le terme apparaît dans le document alors la case du terme en question sera mise à 1 et 0 sinon si on modélise avec un système sans poids .

$$T = \langle t_1, t_2, \dots, t_n \rangle \quad (1)$$

soit un document j les termes qui apparaissent dans le document seront représentées par leur poids

$$Document : d_j = (w_{1j}, w_{2j}, \dots, w_{nj}) \quad (2)$$

w_{ij} : poids du terme t_i dans le document d_j à $tf \cdot idf$

Requête

La requête est également représenté par un vecteur de l'ensemble des termes de la collection pondéré de la manière suivante:

$$Query = (w_{1q}, w_{2q}, \dots, w_{nq}) \quad (3)$$

w_{iq} : poids du terme t_i dans la requête q .

Fonction de similitude

Pour le modèle vectoriel il existe plusieurs fonction de similitude avant de les citer on doit définir la mesure par laquelle se fait la classification des documents selon leurs degré de pertinence pour une requête .

on dira qu'un document est d_1 est d'autant plus pertinent à une requête que le vecteur associé est similaire à celui de la requête .

1- Produit Interne

le produit interne permet de quantifier le degré de similarité entre deux vecteurs , ainsi plus le produit est grand plus le degré de similarité entre les deux est grande .

$$\sum_{i=1}^n q_i * d_i$$

$$Sim(q, d_i) = \vec{q} \cdot \vec{d_i} = \sum_{j=1}^n q_j * d_{i,j}$$

2- Coefficient de Dice

$$Sim(Q, T_i) = \frac{2 * |T_i \cap Q|}{|T_i \cup Q|} = \frac{|T_i \cap Q|}{|T_i| + |Q|}$$

$$Sim(q, d_i) = \frac{2 * \sum_{j=1}^n q_j * d_{i,j}}{\sum_{j=1}^n q_j^2 + \sum_{j=1}^n d_{i,j}^2}$$

3- Cosinus

$$\vec{q} \cdot \vec{d} = \|\vec{q}\| * \|\vec{d}\| * \cos(\overrightarrow{q, d})$$

$$\cos(\overrightarrow{q, d}) = \frac{\sum_{i=1}^n q_{1,i} * d_{2,i}}{\sqrt{\sum_{i=1}^n q_{1,i}^2} * \sqrt{\sum_{i=1}^n d_{2,i}^2}}$$

4- Jaccard

$$Sim(Q, T_i) = \frac{|T_i \cap Q|}{|T_i \cup Q|} = \frac{|T_i \cap Q|}{|T_i| + |Q| - |T_i \cap Q|}$$

$$Sim(q, d_i) = \frac{\sum_{j=1}^n q_j * d_{i,j}}{\sum_{j=1}^n q_j^2 + \sum_{j=1}^n d_{i,j}^2 - \sum_{j=1}^n q_j * d_{i,j}}$$

q_i est le poids du terme t_i dans la requête .

d_i est le poids du terme t_i dans le document .

Fonction d'appariement : Pseudo-code

Algorithm 8 MODÈLE VECTORIEL

Input: fichierInverse : le fichier inverse de notre ensemble de documents,
requete

Output: listDocsPertinents : la liste des documents triés par degré de pertinence

```

38 for  $doc \in listDocs$  do
39    $sim \leftarrow FonctionAppariment(doc, requete)$ 
40   if ( $sim \neq 0$ ) then
41      $listDocsPertinents[doc] \leftarrow sim$ 
42  $listDocsPertinents \leftarrow TrierOrdreDecroissant(listDocsPertinents)$ 
43 return  $listDocsPertinents$ 

```

Algorithm 9 FONCTIONDAPPARIMENT:(JACCARD)

Input: Document,
requete

Output: similarité

```
44 somme1  $\leftarrow$  0
45 somme2  $\leftarrow$  0
46 somme3  $\leftarrow$  0
47 for  $i \in [0, longueur(doc)]$  do
48   | somme1  $\leftarrow$  somme1 + (requete[i] * doc[i] )
49   | somme2  $\leftarrow$  somme2 + (requete[i]2)
50   | somme3  $\leftarrow$  somme3 + (document[i]2)
51 var  $\leftarrow$  somme2 + somme3 - somme1
52 if (var == 0) then
53   | sim=0
54 else
55   | sim  $\leftarrow \frac{somme1}{var}$ 
56 return sim
```

Algorithm 10 FONCTIONDAPPARIMENT:(COSINUS)

Input: Document,
requete

Output: similarité

```
57 somme1  $\leftarrow$  0
58 somme2  $\leftarrow$  0
59 somme3  $\leftarrow$  0
60 for  $i \in [0, longueur(doc)]$  do
61   | somme1  $\leftarrow$  somme1 + (requete[i] * doc[i] )
62   | somme2  $\leftarrow$  somme2 + (requete[i]2)
63   | somme3  $\leftarrow$  somme3 + (document[i]2)
64 var  $\leftarrow$  somme2 + somme3 - somme1
65 if (somme2 ==0 or somme3 ==0) then
66   | sim=0
67 else
68   | sim  $\leftarrow \frac{somme1}{\sqrt{(somme2*somme3)}}$ 
69 return sim
```

Algorithm 11 FONCTIONDAPPARIMENT:(DICE)

Input: Document,
requete

Output: similarité

```
70 somme1 ← 0
71 somme2 ← 0
72 somme3 ← 0
73 for  $i \in [0, longueur(doc)]$  do
74    $somme1 \leftarrow somme1 + (requete[i] * doc[i])$ 
75    $somme2 \leftarrow somme2 + (requete[i]^2)$ 
76    $somme3 \leftarrow somme3 + (document[i]^2)$ 
77  $var \leftarrow somme2 + somme3 - somme1$ 
78 if  $(somme2 == 0 \text{ or } somme3 == 0)$  then
79    $sim = 0$ 
80 else
81    $sim \leftarrow \frac{2 * somme1}{(somme2 + somme3)}$ 
82 return  $sim$ 
```

Algorithm 12 FONCTIONDAPPARIMENT:(PRODUIT INTERNE)

Input: Document,
requete

Output: similarité

```
83  $sim \leftarrow 0$ 
84 for  $i \in [0, longueur(doc)]$  do
85    $sim \leftarrow sim + (requete[i] * document[i])$ 
86 return  $sim$ 
```

Explication de l'algorithme

l'algorithme basé de le modèle vectoriel fonctionne comme suit:

- Pour chaque document de notre collection on calcule la similarité entre notre document et la requête
- On choisi une fonction de similarité parmi celle implémenté (jaccard, dice, produit interne ,cosinus)
- si la similarité est non nulle alors on l'ajoute à une liste de tel sorte que l'indice est celui du document
- on trie la liste des documents pertinents par ordre décroissant
- le premier élément de la liste étant le document le plus pertinent ayant la plus grande similarité

Résultat

on a exécuté la requête $q = ("blue")$.

voici les résultats avec les différentes fonctions d'appariement

	Document	Similarité
1	bryant-stories.txt	1
2	chesterton-ball.txt	1
3		

Figure 7: Résultat du modèle vectoriel avec Produit interne comme Fonction d'appariement.

	Document	Similarité
1	chesterton-ball.txt	0.06451612903225806
2	bryant-stories.txt	0.05
3		

Figure 8: Résultat du modèle vectoriel avec Dice comme Fonction d'appariement.

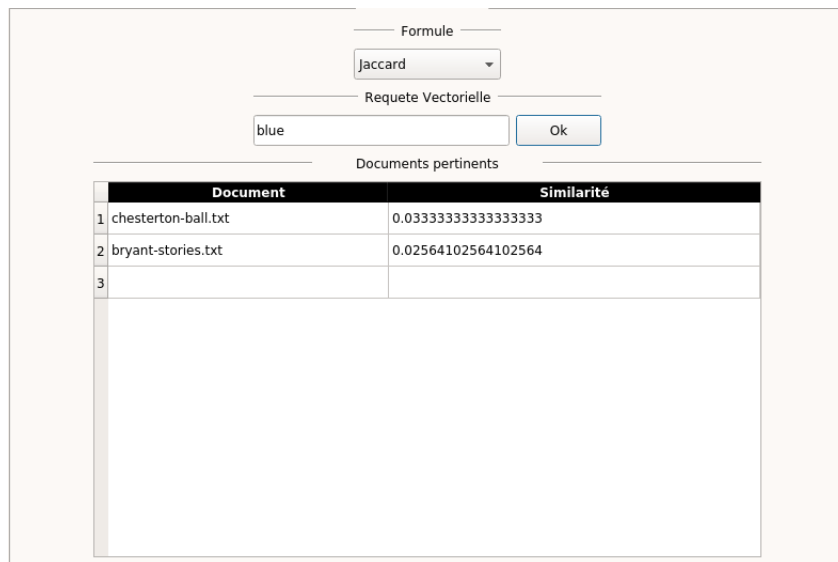


Figure 9: Résultat du modèle vectoriel avec Jaccard comme Fonction d'appariement.

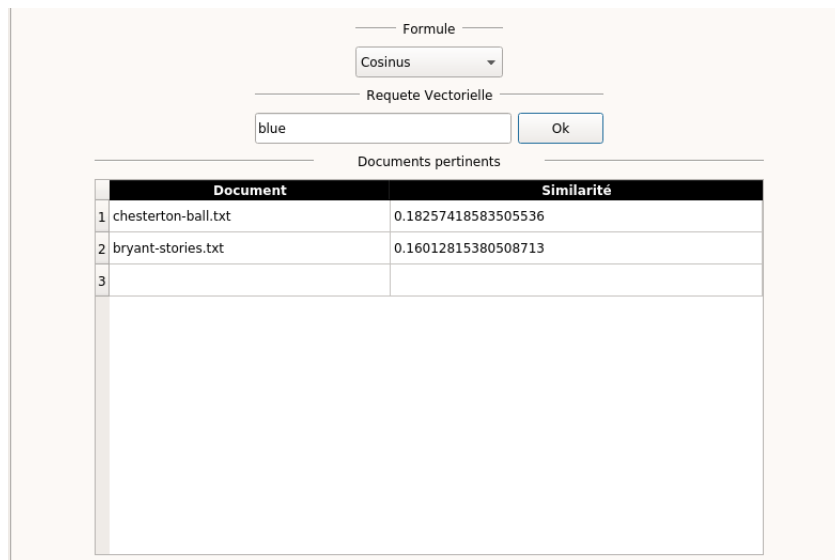


Figure 10: Résultat du modèle vectoriel avec COSINUS comme Fonction d'appariement.

Comparaison

On remarque que avec le produit interne on a eu une similarité identique entre les deux documents contrairement aux trois autres fonctions jaccard ,dice , cosinus qui ont donné des résultats identiques avec le même tri des documents .

On peut conclure que les fonctions jaccard ,cosinus , dice sont plus précises pour le calcul de similarité entre les documents comparé au produit interne ceci se traduit par la complexité de leurs calcul .

5.3 Modèle Probabiliste avec échantillon

Le modèle probabiliste quant à lui, requière un échantillon d'apprentissage. Dans notre cas, cet échantillon sera les documents jugés pertinents par l'utilisateur lui même.

C'est à dire que notre modèle probabiliste répondra en premier lieu à la requête de l'utilisateur avec le modèle vectoriel cité juste au dessus, puis à partir de cet échantillon l'utilisateur choisira parmi les documents retournés par le modèle vectoriel ceux qu'il juge pertinents \Rightarrow échantillon d'apprentissage .

Puis nous procédons à une deuxième recherche avec cette même requête en appliquant le modèle probabiliste dont la formule pondéré qui est comme suit:

$$\text{Sim}(d_j, Q) = \sum_{i=1}^n P_{ij} * P_{iq} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)}$$

avec:

d_j : le document j.

Q : la requête.

n : le nombre de terme dans la requête Q.

r_i : le nombre de documents pertinents contenant le terme t_i .

R : le nombre de documents pertinents.

n_i : le nombre de documents contenant le terme t_i .

N : le nombre de document de notre collection.

P_{ij} : le poids du terme t_i dans le document j. P_{iq} : le poids du terme t_i dans la requête Q.

Requête

La requête à la même forme que celle décrite précédemment pour le modèle Vectoriel.

Fonction d'appariement : Pseudo-code

Algorithm 13 MODÈLE PROBABILISTE

Input: listDocs : liste des documents,
fichierInversePoids : le fichier inverse des poids de notre ensemble de documents,
listDocPertinents : la liste des documents pertinents
requete: requete recherchée

Output: listDocsPertinents : la liste des documents pertinents

```
87  $N \leftarrow \text{longueur}(\text{listDocs})$ 
 $R \leftarrow \text{longueur}(\text{listDocPertinents})$ 
for  $doc \in \text{listDocs}$  do
88    $Sim \leftarrow 0$ 
   for  $terme \in \text{requete}$  do
89      $n_i \leftarrow \text{NbrDocContenantTi}(\text{listDocs}, \text{terme})$ 
      $r_i \leftarrow \text{NbrDocContenantTi}(\text{listDocPertinents}, \text{terme})$ 
      $P_{ij} \leftarrow \text{fichierInversePoids}[\text{dos}][\text{terme}]$ 
      $P_{iq} \leftarrow \text{nbrOccurence}(\text{terme}, \text{requete})$ 
90      $Sim \leftarrow Sim + [P_{ij} * P_{iq} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)}]$ 
91   if  $Sim > 0$  then
92      $\text{Ineserer}(doc, \text{listDocsPertinents})$ 
93 return  $\text{listDocsPertinents}$ 
```

Explication de l'algorithme

notre algorithme effectue alors la séquence d'instruction suivante:

- pour chaque document de la collection.
- pour chaque terme de la requête.
- Calculer le nombre de document contenant le terme courant de la requête.
- Calculer le nombre de documents Pertinents contenant le terme courant de la requête.
- Récupérer le poids du terme dans le document en cours d'appariement.
- Récupérer le poids du terme dans la requête recherchée.
- Calculer la similitude entre le document et la requête via la formule de ce modèle comme donnée précédemment.
- retourner la liste de document dont la similitude obtenue est > 0 .

Résultat

Prenons comme requete :

austen emma

la première étape consiste à générer l'échantillon à partir du modèle vectoriel, pour cela nous appuyons via notre interface sur le bouton **Générer l'échantillon** ce trouvant complètement en bas a droite.

Nous voyons apparaitre une liste de document parmi la quelle l'utilisateur peut sélectionner ceux qu'il juge pertinents, comme dans la figure ci-dessous:

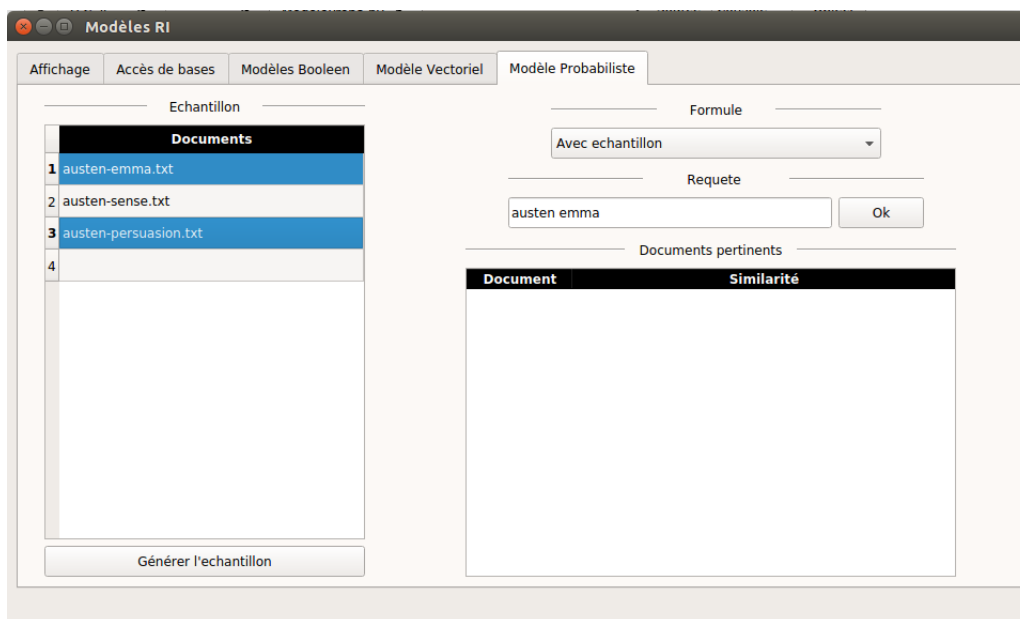


Figure 11: Affichage de l'échantillon et sélection de l'utilisateur.

Maintenant que nous avons notre liste de documents pertinents, nous passons à l'application de la formule du modèle probabiliste, et cela en appuyant sur le bouton **"OK"** à coté de la requête.

Le résultat est donnée dans le tableau juste en dessous, tel que **les documents sont trié par ordre décroissant de similitude**.

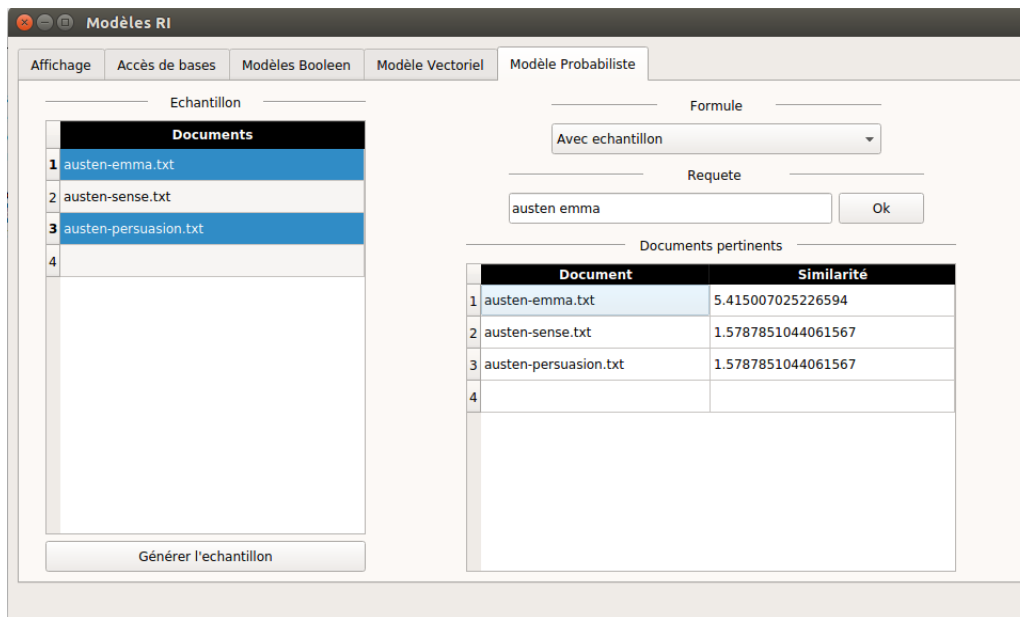


Figure 12: Résultat du modèle probabiliste.

6 Bonus

6.1 Modèle Probabiliste sans échantillon

Le principe étant assez proche du modèle probabiliste avec échantillon, à la différence de l'absence d'échantillon d'apprentissage comme son nom le précise.

De ce fait, la formule ne prend pas en compte les informations relatives à l'échantillon, tel qu'elle se présente comme suit:

$$\text{Sim}(d_j, Q) = \sum_{i=1}^n P_{ij} * P_{iq} \log \frac{N - n_i + 0.5}{n_i + 0.5}$$

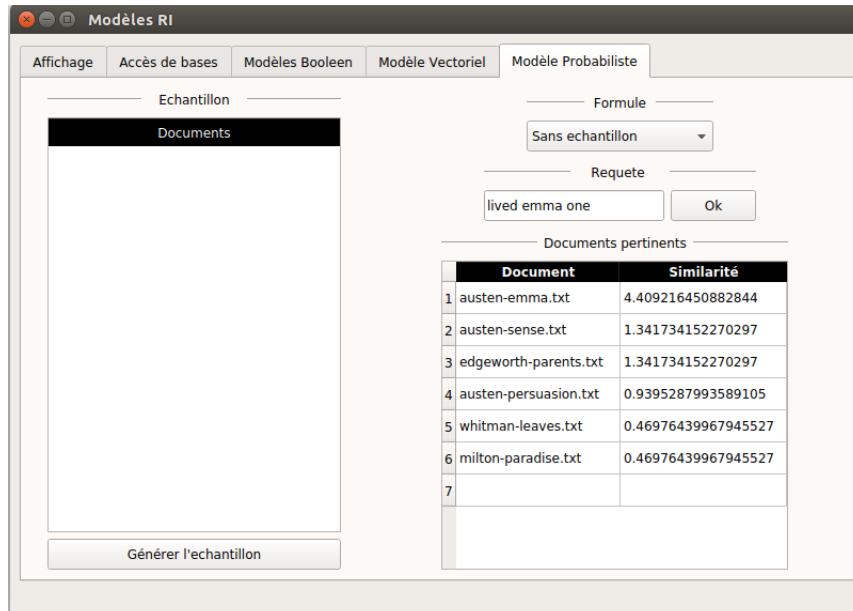
Requête

Identique à celle du modèle vectoriel et probabiliste avec échantillon.

Résultat

Dans l'onglet dédié au modèle probabiliste de notre interface, nous avons une liste déroulante **ComboBox** nous permettant de sélectionner le **modèle Probabiliste sans échantillon**, puis procède normalement à la saisi de la requête et l'exécution de l'appariement.

Ainsi la liste des documents est retournée dans l'ordre décroissant de similitude, comme dans l'exécution suivante:



The screenshot shows a software interface titled 'Modèles RI' with several tabs: 'Affichage', 'Accès de bases', 'Modèles Booleen', 'Modèle Vectoriel', and 'Modèle Probabiliste'. The 'Modèle Probabiliste' tab is active. It contains a section for 'Echantillon' with a 'Documents' list and a 'Générer l'échantillon' button. To the right, there's a 'Formule' dropdown set to 'Sans échantillon' and a 'Requete' field with the text 'lived emma one' and an 'Ok' button. Below these is a table titled 'Documents pertinents' showing a list of documents and their similarity scores.

	Document	Similarité
1	austen-emma.txt	4.409216450882844
2	austen-sense.txt	1.341734152270297
3	edgeworth-parents.txt	1.341734152270297
4	austen-persuasion.txt	0.9395287993589105
5	whitman-leaves.txt	0.46976439967945527
6	milton-paradise.txt	0.46976439967945527
7		

Figure 13: Résultat du modèle probabiliste sans échantillon.

6.2 Modèle Probabiliste BM25

Il existe une variante du modèle probabiliste tout aussi utilisé **BM25**, aussi appelée l'estimation de **S.Walker S.Roberston** qui à la différences des autres modèle prend en compte la longueur des documents ainsi que 2 constantes, la formule est donnée comme suit:

$$\text{Sim}(d_j, Q) = \sum_{i=1}^n \left(\frac{(K + 1) * \text{freq}_{ij}}{\text{freq}_{ij} + K * (1 - B) + B * \frac{dl}{AVGdl}} \right) * \log \frac{N - n_i + 0.5}{n_i + 0.5}$$

Avec:

K : Constante par défaut = 1.2.

B : Constante par défaut = 0.75.

freq_{ij} : fréquence du terme i dans le document j .

dl : Longueur du document j .

$AVGdl$: Longueur moyenne des documents de la collection.

Requête

Identique à celle du modèle vectoriel et probabiliste avec et sans échantillon.

Résultat

Toujours dans l'onglet du modèle probabiliste nous pouvons choisir l'option **BM25** de la liste défilante, insérer une requête et lancer l'évaluation de similitude afin d'obtenir les documents pertinents dans l'ordre décroissant de leur similitude.

The screenshot shows the 'Modèles RI' application window. The 'Modèle Probabiliste' tab is selected. Under 'Formule', 'BM25' is chosen. The 'Requete' field contains 'little austen' and the 'Ok' button is visible. Below, the 'Documents pertinents' section displays a table with the following data:

	Document	Similarité
1	austen-emma.txt	0.8701872827463777
2	austen-persuasion.txt	0.38344616605767184
3	austen-sense.txt	0.38247246250813954
4		

On the left, there is an 'Echantillon' section with a 'Documents' header and a 'Générer l'échantillon' button at the bottom.

Figure 14: Résultat du modèle BM25.

7 Comparaison des trois modèles

Après avoir étudié, implémenté et testé les 3 principaux modèles qui sont : **modèle booléen**, **modèle vectoriel** et **modèle probabiliste avec échantillon**, nous avons pu constater les avantages et inconvénients de chacun, citons :

- le modèle est très rigide tel qu'il suffit qu'un seul terme de la requête corresponde pas au contenu d'un document pour que ce dernier soit jugé non pertinent.

Surtout il se peut que certains problèmes nécessitent un genre de modèle, mais cela reste des cas particuliers, en générale il est préférable d'avoir des documents ordonnés par degrés de pertinence, ainsi il suffira de définir un seuil minimal pour éliminer les documents trop peu pertinents.

- le modèle vectoriel est bien plus souple que le précédent, il prend en compte le degré de similitude et ceci peut être fait via une des 4 fonctions de calcul de similitude qu'il propose, soient : **Produit Interne**, **coefficient de Dice**, **Cosinus** et **Jaccard**.
- le modèle Probabiliste apporte une nouvelle notion "**d'échantillon**" tel que l'utilisateur peut influencer les similitudes des documents en précisant une liste de documents qu'il juge lui-même pertinents, et ce après connaissance de la collection à traiter.

Ainsi ce modèle est encore plus souple et adaptable selon les besoins de l'utilisateur ce qui lui procure les meilleurs résultats en général, mais ce n'est pas toujours possible de donner cette liberté à l'utilisateur, ce n'est pas toujours facile pour l'utilisateur de reconnaître les documents pertinents sur une large collection de documents.

Conclusion

En conclusion, après avoir passé en revue les différents modèles en mettant l'accent sur les avantages et inconvénients de chacun, nous déduisant que tous les modèles peuvent être jugé les plus performant selon le problème à résoudre et les ressources disponibles, car il n'y a pas de super modèle classé comme permettant d'obtenir les meilleurs résultats, et les plus précis pour tous les problèmes données.

Néanmoins nous pensons que lorsque nous ne savons pas quel modèle est le plus approprié il est préférable de se diriger vers le modèle vectoriel et selon l'évolution et les résultats décider s'il faut changer de modèle ou pas.

Aussi la création et manipulation des fichier inverse des fréquences et des poids permet une utilisation simple et structuré de nos données, d'ailleurs celles-ci permette une nette réduction du temps d'exécution et de recherche dans la collection de document.

Références bibliographiques

[1] : <http://www.iro.umontreal.ca/nie/IFT6255/historique-RI.html>