Université des Sciences et de la Technologie Houari Boumediene

Recherche d'information

Rapport projet
Modules de base de la recherche d'information appliqué sur la collection CACM

Realisé par:

HEBBACHE Imad eddine

BELABDI Malak

Pr. Maissa ANANE

December 27, 2021

Introduction

Depuis X temps l'esprit humain tenté de mettre en place des systèmes pour faciliter l'accès aux différents objets (ou informations) dans son environnement, par exemple "l'organisation des livres dans une bibliothèque".
De nos jours, l'avancement de la technologie en général et du domaine **IT** en particulier a donné naissance à une très grande branche pour faciliter l'accès aux objets, cette branche est appelée La Recherche d'information. La Recherche d'information (RI) est le domaine qui consiste à trouver un objet dans tout média pertinent pour répondre à la requête d'un utilisateur.
La RI concerne la représentation, le stockage, l'organisation et l'accès aux sources d'information, pour cela nous avons besoin d'un système de recherche d'information (SRI). Ce dernier est un ou plusieurs programmes informatiques qui ont pour but de sélectionner des informations pertinentes répondant à des besoins utilisateurs (requête).
Dans ce projet nous allons implémenté les modules de base en RI sur la collection CACM en utilisant le langage PYTHON à savoir : l'indexation des documents, la pondération des termes, l'appariement requête-document, la réalisation du modèle Booléen, la réalisation du modèle Vectoriel et l'évaluation.
Le résultat de cette étude est une IHM qui facilite l'utilisation et la visualisation des résultats obtenu par l'interrogation de notre SRI.

1 Phase d'indexation des documents

1.1 Étude de la collection CACM

La collection CACM est un ensemble de documents regroupés dans un seule fichier (cacm.all). Plusieurs délimiteurs sont introduit, on s'intéresse aux champs I, T et W:

- .In : indique le début d'un document, ou n est l'identificateur unique du document correspondant.
- .T : indique le champs contenant le titre du document courant.
- .W : indique le champs contenant le résumé du document courant.

Implémentation

- tout d'abord on récupère le contenu du fichier

```
#Lecture du fichier qui contient les documents
CacmAll = open("F:\RI\Projet\cacm\cacm.all", "r")

#lecture des lignes
lines = CacmAll.readlines()
```

- ensuite on procède a la récupération des champs des documents par la boucle suivante

```
i=0
while(i<len(lines)):
    #on recupere la ligne
    line = lines[i]

    #si la ligne commence par .I on recupere l'identifiant du document
    if(line.startswith('.I')):
        DocIDF = int(line.split()[1])

    #sinon si la liste commence par .T, on recupere les lignes du champ titre
    if(line.startswith('.T')):
        i+=1
        titre = ""

        #on ajoute les lignes tanqu'elles ne commencent pas par un des marqueur en prenant
        # en compte l'identifiant et le saut de ligne
        while((i<len(lines)) and (re.findall('\.([TWBANX]\n|I [0-9]+\n)', lines[i]))==[]):
            titre = titre + " " + lines[i]
            i+=1
        DictDoc[DocIDF] = titre

        #on decremente la ligne afin de retourner au marqueur
        i-=1

    #sinon si la liste commence par .W, on recupere les lignes du champ resume
    if(line.startswith('.W')):
        i+=1
        resume = ""
        while((i<len(lines)) and (re.findall('\.([TWBANX]|I [0-9]+)', lines[i]))==[]):
            resume = resume +" "+lines[i]
            i+=1
        DictDoc[DocIDF] = DictDoc[DocIDF] + resume
        i-=1
    i+=1
```

- A la fin, le résultat est un dictionnaire **DictDoc** sous la forme document : texte du document, ci joint le contenu du 20em document par exemple:

```
(20, ' Accelerating Convergence of Iterative Processes\n A technique is discussed which, when
 applied\n to an iterative procedure for the solution of\n an equation, accelerates the rate
of convergence if\n the iteration converges and induces convergence if\n the iteration diverg
es.  An illustrative example is given.\n')
```

1.2 Tokenisation

Après récupération des champs T et W de chaque document, on doit découper les lignes de chaque document en mot. Les espaces et toutes ponctuations sont considérés comme séparateurs du mot.

Implémentation

Nous avons procédé comme suit (dans la fonction StopwordElimination(text)):

- définir la liste des ponctuations

```
ponctuation_list = ['?', '.', '!', '<', '>', '}', '{', ':', '(', ')', '[', ']', '\"',
',', '-', "»", "«", '\'', '’', '#', '+', '_', '-', '*', '/', '=','\"n', ';','$']
```

- remplacer les ponctuations par des espaces

```
word_list = []
# Eliminer la punctuation
for character in ponctuation_list:
    text = text.replace(character, ' ')
```

- faire une séparation des mots par rapport aux espaces

```
# str -> list
words = text.split()
for word in words:
    if word.lower() not in stopwords_list:
        word_list.append(word.lower())
return word_list
```

- le résultat est tous les mots des documents y compris les mots vides. Ci joint les mots du document 206:

```
['Symbol', 'Manipulation', 'in', 'XTRAN']
```

1.3 Comparaison avec *"commonwords"*

Dans l'étape précédente, on remarque qu'ils existent quelques mots non-significatifs dans les documents. Afin d'avoir une bonne modélisation de nos document on doit éliminer ces mots on utilisant le fichier *"commonwords"*.

Implémentation Ces étapes sont implémentées dans la fonction StopwordElimination(text) également:

- On récupère la liste des mots vide

```
# ouverture du fichier common_words
stopwordsfile = "F:\RI\Projet\cacm\common_words"
```

- On fait une comparaison comme suit : si un mot d'un document existe dans la liste des mots vide, on l'ignore. Voici le code associé

```
# str -> list
words = text.split()
for word in words:
    if word.lower() not in stopwords_list:
        word_list.append(word.lower())
return word_list
```

Voici la liste des mots du document 206 après élimination des mots vides ('in' dans ce cas):

```
{'symbol': 1, 'manipulation': 1, 'xtran': 1}
```

1.4 Création du dictionnaire des fréquence

Le but de cette étape est de construire un dictionnaire *"indexDict"*qui a comme clé les documents de toute la collection et comme valeur les termes appartenant à ce document avec leur fréquence. Alors le dictionnaire a la structure suivante :

$$indexDict = \{document_j : \{terme_i : frequence_{ij}, ...\}, ...\}$$

Implémentation
L'implémentation à nécessiter l'utilisation de deux fonctions :

1. *dicFreq*():Cette fonction, prend en paramètres une liste de termes, et retourne un dictionnaire de la forme :

   $$\{terme_i : frequence\ du\ terme_i\ dans\ la\ liste\ en\ entree, ...\}$$

   Ci joint le code de cette fonction :

```
def dict_freq(word_list):
    frequence_dict = {}
    for word in word_list:
        if word not in frequence_dict:
            frequence_dict[word] = word_list.count(word)
    return frequence_dict
```

2. *CreationIndexDict*(): Nous avons Défini une variable **'allwords'**qui va contenir tout les termes de la collections sans redondances. Par la suite,on parcourt les clés du dictionnaire DictDoc (tous les documents de la collection) et on calcul la fréquence des termes de ce document en utilisant la fonction définie ci-dessus. Le résultat est le dictionnaire *"indexDict"*. Ci joint le code de la fonction:

```python
#la creation du dict indexé
def creation_indexedDict():
    inexedDict={}
    #allwords est une liste qui contient tous
    # les terms de la collection sans redand
    global allwords
    allwords=[]
    for key in DictDoc.keys():
        text=DictDoc[key]
        word_list=Stopword_elimination(text)
        for word in word_list:
            if word not in allwords:
                allwords.append(word)
        if key not in inexedDict.keys():
            inexedDict[key]=dict_freq(word_list)
    return inexedDict
```

Par exemple on vérifie le tuple du document 300 :

```
(300, {'cobol': 3, 'sample': 1, 'problem': 2, 'simplified': 1, 'merchandise':
1, 'control': 1, 'chosen': 1, 'presenting': 1, 'users': 2, 'potential': 1, 'co
mputing': 1, 'systems': 1, 'mythical': 1, 'department': 1, 'store': 1, 'langua
ge': 2, 'bros': 1, 'programming': 1, 'runs': 1, 'computer': 1})
```

1.5 Création d'un fichier inversé par fréquence

Afin d'améliorer l'efficacité de notre SRI, il est préférable de convertir le résultat de l'indexation (indexDict) en un fichier inverse (un dictionnaire). Voici la structure que nous avons utilisé:

$$fichier\ inverse = \{terme_i : \{document_j : frequence_{ij}, ...\}, ...\}$$

Implémentation
L'implémentation à nécessiter l'utilisation de deux fonctions :

1. dictDocFreq(word,inexedDict): La fonction parcourt tout les documents de la collection et vérifie si le document contient le terme, si c'est le cas elle insert le tuple

   $$\{document : frequence\ du\ terme\ dans\ le\ document\}$$

   Ci joint le code de la fonction:

```python
def dict_doc_freq(word,inexedDict):
    dic_doc_freq={}
    for i in inexedDict.keys():
        for j in inexedDict[i].keys():
            if j==word:
                if i not in dic_doc_freq.keys():
                    dic_doc_freq[i]=inexedDict[i].get(word)
    return dic_doc_freq
```

2. creationFichierInverse(): On parcourt la liste *'allwords'* et on fait appel à la fonction précédente à chaque fois. Ci joint le code de la fonction:

```python
def creation_fichierInverse():
    fichier_inverse={}
    for word in allwords:
        if word not in fichier_inverse:
            fichier_inverse[word] = dict_doc_freq(word,inexedDict)
    return fichier_inverse
```

Par exemple on vérifie le tuple du terme 'section' :

```
('section', {202: 1, 319: 1, 533: 1, 1198: 1, 1350: 2, 1434: 1, 1540: 1, 1781:
 3, 2046: 2, 2150: 2, 2181: 1, 2198: 2, 2316: 1, 2366: 1, 2556: 1, 2766: 1, 30
23: 1})
```

1.6 Fonctions d'accès

1. ReturnWords(indiceDocument): Cette fonction prend en entrée un indice de document et retourne la liste des termes de ce document avec leur fréquence. Voici le code :

```
def ReturnWords(indiceDocument):
    indexdoc = creation_indexedDict()
    for i in indexdoc.keys():
        if(i == indiceDocument):
            #print(indexdoc[i])
            return indexdoc[i]
    print("liste de mots-fréquences non trouvée")
```

voici un exemple de mots retournés avec le document 1 :

```
{'preliminary': 1, 'report': 1, 'international': 1, 'algebraic': 1, 'language': 1}
```

2. *ReturnDocs*(*terme*): Cette fonction prend en entrée un terme et retourne l'ensemble de documents contenant ce terme en indiquant la fréquence. Ci joint le code:

```
def ReturnDocs(terme):
    for i in fichier_inverse.keys():
        if (i == str(terme).lower()):
            return fichier_inverse[i]
    print("liste de documents-fréquences non trouvée")
```

Voici un exemple de documents retournés avec le terme 'Algebraic':

```
{1: 1, 21: 1, 44: 1, 54: 1, 55: 1, 93: 1, 99: 1, 284: 1, 393: 1, 964: 2, 1029: 1, 1214: 2, 12
16: 1, 1223: 1, 1253: 2, 1258: 1, 1334: 1, 1365: 1, 1394: 2, 1397: 4, 1453: 2, 1471: 1, 1543:
 1, 1589: 1, 1824: 1, 1975: 1, 2054: 1, 2090: 1, 2164: 1, 2165: 2, 2166: 2, 2167: 4, 2323: 2,
 2547: 1, 2645: 1, 2802: 1, 2809: 1, 2931: 1, 2958: 2, 3031: 3, 3071: 1, 3077: 2, 3078: 1, 31
89: 3, 3199: 3, 3202: 1, 3203: 2}
```

1.7 Technique de sauvegarde utilisées :

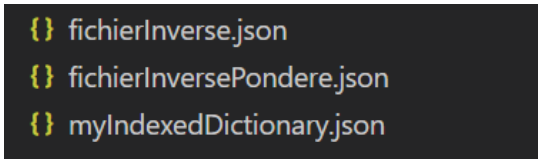
Afin de ne pas faire les calculs des indexes à chaque réception d'une requête, nous avons choisi de stocker tous les indexes sous forme d'un fichier d'extension ***'.json'***.
Voici l'enregistrement et le chargement de l'indexe ”
Enregistrement!

```
tfinexedDict = open("myIndexedDictionary.json", "w")
json.dump(inexedDict,tfinexedDict)
tfinexedDict.close()
```

Chargement:

```
#loading dictionary
tfinexedDict = open("myIndexedDictionary.json", "r")
inexedDict = json.load(tfinexedDict)
print(inexedDict)
```

Voici les structures obtenues :



1.8 tailles et durées d'indexation

Variable	Nb d'entrées	Taille(octets)	T1(s)	T2(s)
DictDoc	3204	81976	1.1776	1.7150
indexedDic	3204	81976	6.4836	1.1540
fichier-inverse	9223	163892	58.1490	1.1430

Table 1: Tailles et temps d'indexation

Où

- T1 est le temps d'exécution des calculs des indexes (fonctions de calcul)
- T2 est le temps d'exécution de chargement des indexes (après enregistrement sous forme de fichier.json)

Remarque

Une grande amélioration dans le temps d'exécution après enregistrement des indexes.

1.9 Critiques

- **Avantages :**
  - L'élimination des champs non importants (étape 1.1) a réduit le temps d'exécution et l'espace mémoire occupé par les indexes.
  - L'élimination des mots vides (étape 1.3) a permet de garder seulement les mots significatifs à un document spécialement, mais pas les mots communs entre l'ensemble de documents (comme les connecteurs, les verbes, les pronoms..., ce qui réduit énormément le temps de réponse à une requête.
  - La création des indexes (étapes 1.4 et 1.5) est la meilleure méthode de stockage d'une collection.

- Les fonction d'accès (étape 1.6) permettent un accès direct aux fréquences, qui est idéal pour une requête indiquant un seul terme ou document.

- **Inconvénients :**
  - Avec ce type d'indexation, on peut avoir beaucoup de redondances, car nous sommes entrain de prendre le terme en entier. Or, il est souhaitable de faire une normalisation et garder que les radicales. Exemple : notre indexe contient les mots : Program, Programs et Programming, la normalisation permet de garder le radical Program seulement.
  - Malgré la réduction des mots, les indexes restent coûteux en terme de temps d'exécution (leur création) et d'espace mémoire.

2 Réalisation du modèle booléen

2.1 Représentation

Données : Pour le stockage des données, nous avons utilisé une structure similaire à la structure du fichier inversé, au lieu de mettre des fréquence on mit 1. Pour un terme donné nous aurons tout les document dans il appartient.

$$DictBool = \{terme_i : \{document_j : 1, ...\}, ...\}$$

Ci-joint le code utilisé :

```python
#Module de Representation des Document
def MRD(fichier_inverse):
    DictBool = {}
    for terme in fichier_inverse.keys():
        DictDocs = {}
        document = 1
        for document in dict(fichier_inverse[terme]).keys():
            DictDocs[document]=1
        DictBool.update({terme : DictDocs})
    return DictBool
```

2.2 Évaluation d'une requête :

Une requête est représentée par un ensemble de termes séparés par des connecteurs logiques. La réponse à une requête booléenne consiste à évaluer la requête, récupérer les termes puis accéder au dictionnaire DictBool et retourner les documents pertinents répondant à la requête.

Implémentation

- ModeleBooleen(Requete,Index,StopwordList): Cette fonction prend en entrée la requête sous forme d'une chaîne de caractère, le dictionnaire DictBool (Index) et la liste des mots vides.
  Tout d'abord, elle récupère les mots de la requête et les connecteurs.
  Pour les mots, on les rend en minuscules et on élimines les mots vides. Pour chaque document on vérifie si les mots de la requête existe dedans, si c'est le cas on remplace le terme par 1 sinon on le remplace par 0,Voici un exemple de requête : 1 and (0 or 1).
  La dernière étape consiste à évaluer la requête par la fonction **'eval'**, si la valeur retournée est 1, on ajoute les documents correspondant à la liste finale retournée.
  Ci joint le code de la fonction :

```python
def ModeleBooleen(Requete,Index,Stopword_list):
    listeDoc=[]
    Req=''
    for doc in Index:
        document=dict()
        for mot in Index[doc]:
            document[mot]= Index[doc][mot]
        MotsReqNoisy=nltk.tokenize.word_tokenize(Requete)
        MotsReq=[]
        for mot in MotsReqNoisy:
            if(mot.lower() not in ['and','or','(',')','not']):
                if mot.lower() not in Stopword_list:
                    MotsReq.append(mot.lower())
            else:
                MotsReq.append(mot.lower())
        for mot in MotsReq:
            Req=''
            if(mot.lower() not in ['and','or','(',')','not']):
                if(mot.lower() not in  document):
                    MotsReq[MotsReq.index(mot)]=0
                else:
                    MotsReq[MotsReq.index(mot)]=1
        for el in MotsReq:
            Req=Req+' '+str(el)
        if(eval(Req)==1):
            listeDoc.append(doc)
    return listeDoc;
```

3 Création d'un fichier inverse par pondération

Une modélisation améliorée pour le fichier inversé consiste à modifier la mesure d'évaluation de similarité par fréquence en évaluation par poids, où un poids d'un terme dans un document est calculé par la formule suivant : $poids(ti, dj) = (freq(ti, dj)/Max(freq(dj)) * Log((N/ni) + 1)$

- $freq(ti, dj)$: Fréquence du terme t_i dans le document d_j
- $Max(freq(dj))$: La fréquence maximale dans le document d_j
- N : le nombre total des documents de la collection.
- n_i : le nombre de document dont le terme t_i apparaît.

Le Résultat serai un dictionnaire de la forme :

$$FichierInvrsePoids = \{terme_i : \{document_j : poids_{ij}, ...\}, ...\}$$

Implémentation Nous avons implémenté cet algorithme par deux fonctions:

1. $dict-doc-poids(word, inexedDict, fichier_inverse, N)$: La fonction parcours l'indexe *'indexedDict'* afin de récupérer la fréquence maximale d'un document et la fréquence du *'Word'* dans un document. On calcul le n_i apartir du fichier inverse.La dernière étape est l'application de la formule.
   Ci joint le code :

```
def dict_doc_poids(word,inexedDict,fichier_inverse,N):
    #N est le nombre total des documents de la collection
    dic_doc_poids={}
    #on parcour tout les numeros de documents de 1 à 3204
    for i in inexedDict.keys():
        #on vérifi si le doc contient le mot word(ti) on parcourant ses mots
        for j in inexedDict[i].keys():
            if j==word:
                if i not in dic_doc_poids.keys():
                    #on recupere la frequence du mot dans le docuement i  = dj
                    freq=inexedDict[i].get(word)
                    #on recupére la freq maximal dans le doc i = dj
                    max=0
                    for y in inexedDict[i].values():
                        if y>max:
                            max=y
                    ni=len(fichier_inverse[word])
                    #on calcul ni avec le fichier inversé pour gagné du temps
                    dic_doc_poids[i]=(freq/max)*math.log10( (N/ni)+1 )
    return dic_doc_poids
```

2. $creation - fichierInversePondere()$: Cette fonction parcourt la liste des terme et fait un appel à la fonction précédente à chaque fois. Ci joint le code :

```
#cette fonction permet la creation du fichier inverse pondéré
def creation_fichierInversePonderé():
    fichier_inverse_pondere={}
    for word in allwords:
        if word not in fichier_inverse_pondere:
            fichier_inverse_pondere[word] = dict_doc_poids(word,indexedDict,fichier_inverse,N)
    return fichier_inverse_pondere
```

Par exemple voici les poids du terme 'Algebraic' dans les différents documents :


```
('algebraic', {1: 1.8399191115604092, 21: 1.8399191115604092, 44: 1.8399191115604092, 54: 1.8399191115604092, 55
: 1.8399191115604092, 93: 0.6133063705201364, 99: 1.8399191115604092, 284: 0.4599797778901023, 393: 1.8399191115
604092, 964: 1.8399191115604092, 1029: 0.3679838223120819, 1214: 1.8399191115604092, 1216: 0.9199595557802046, 1
223: 0.6133063705201364, 1253: 0.7359676446241638, 1258: 0.3679838223120819, 1334: 0.4599797778901023, 1365: 0.4
599797778901023, 1394: 1.8399191115604092, 1397: 1.8399191115604092, 1453: 0.9199595557802046, 1471: 0.613306370
5201364, 1543: 0.26284558736577274, 1589: 0.4599797778901023, 1824: 0.26284558736577274, 1975: 0.613306370520136
4, 2054: 0.4599797778901023, 2090: 1.8399191115604092, 2164: 0.4599797778901023, 2165: 1.8399191115604092, 2166:
 0.9199595557802046, 2167: 1.2266127410402727, 2323: 1.2266127410402727, 2547: 0.4599797778901023, 2645: 0.30665
31852600682, 2802: 0.4599797778901023, 2809: 0.26284558736577274, 2931: 0.4599797778901023, 2958: 0.919959555780
2046, 3031: 1.1039514669362456, 3071: 0.3679838223120819, 3077: 0.3066531852600682, 3078: 0.4599797778901023, 31
89: 1.8399191115604092, 3199: 1.379939333670307, 3202: 0.9199595557802046, 3203: 1.2266127410402727})
```


3.1 taille et temps d'exécution

Variable	b d'entrées	Taille(octets)	Tps de calcul	tps de chargement
FichierInvrsePoids	9223	163892	54.3132	1.31707

Table 2: Taille et temps d'exécution

4 Réalisation du modèle vectoriel

4.1 Données :

Pour nous les données, nous utilisons les indexes déjà stockés. Une requête est représentée par un ensemble de terme. Une interrogation du modèle vectoriel consiste à retourner les documents pertinents par rapport aux termes d'une requête.

4.2 Implémention

4.2.1 Les fonctions d'appariement :

Nous avons utilisé le dictionnaire *'FichierInvrsePoids'* afin de récupérer les poids W_{ij} d'un terme dans un document. Pour le poids W_{iq} , nous avons procédé selon le cours (si un terme i apparaît dans la requête q alors son poids est égale à 1, sinon 0.
Voici les programmes utilisés pour chacune des quatre fonctions :

1. Produit Interne:

```
#fonction 1 du modele vectoriel
def ProduitInterne(dj,requete):
    produit = 0
    for terme in requete:
        produit = produit + FichierInvrsePoids[str(terme).lower()][dj]
    return produit
```

2. Coefficient de Dice:

```
241  #fonction 2 du modele vectoriel
242  def CoefDeDice(dj, NewRequete,requete,documentWords,FichierInvrsePoids):
243      produit = 2*ProduitInterne(dj, NewRequete,FichierInvrsePoids)
244      somme = 0
245      i = 0
246      while(i<len(documentWords)):
247          somme = somme+ math.pow(FichierInvrsePoids[str(documentWords[i]).lower()][dj],2)
248          i+=1
249      somme = somme + len(requete)
250      return produit/somme
```

3. Cosinus:

```
252  #fonction 3 du modele vectoriel
253  def Cosinus(dj, NewRequete,requete,documentWords,FichierInvrsePoids):
254      produit = ProduitInterne(dj, NewRequete,FichierInvrsePoids)
255      somme = 0
256      i = 0
257      while(i < len(documentWords)):
258          somme = somme+ math.pow(FichierInvrsePoids[str(documentWords[i]).lower()][dj],2)
259          i+=1
260      return produit/math.sqrt(somme*len(requete))
261
```

4. Jaccard:

```
263  #fonction 4 du modele vectoriel
264  def Jaccard(dj,NewRequete,requete,documentWords,FichierInvrsePoids):
265      produit = ProduitInterne(dj, NewRequete,FichierInvrsePoids)
266      somme = 0
267      i = 0
268      while(i < len(documentWords)):
269          somme = somme+ math.pow(FichierInvrsePoids[str(documentWords[i]).lower()][dj],2)
270          i+=1
271      somme = somme+len(requete)-produit
272      return produit/somme
```

4.2.2 Interrogation du SRI :

1. Tout d'abord, on traite la requête (récupération des termes, élimination des mots vides et conversion en minuscules):

```
#Module de Representation des Requetes
def MRR_Vec(text,Stopword_list):
    req=text.split()
    requete=[]
    for mot in req:
        if mot.lower() not in Stopword_list:
            requete.append(mot.lower())
    return requete
```

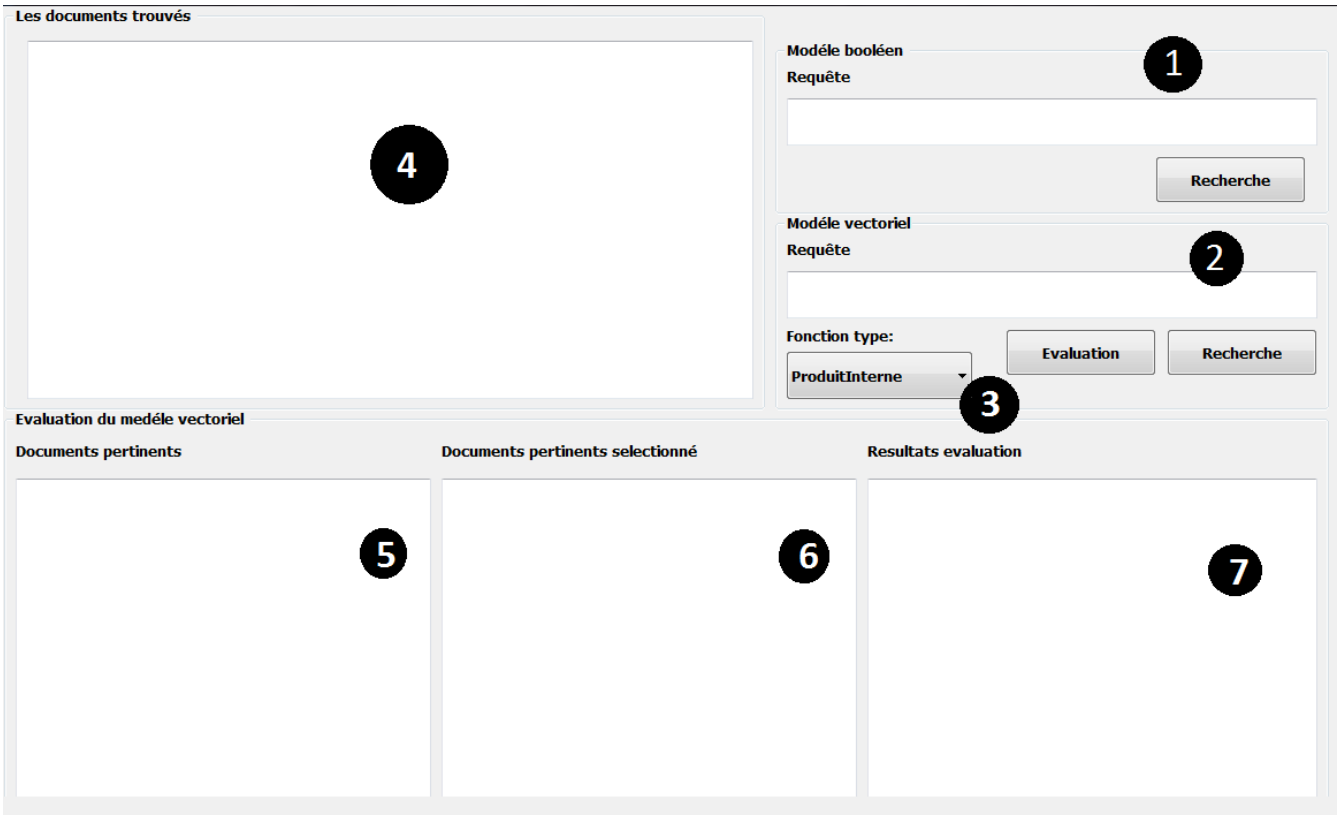
2. Par la suite nous avons utilisée la fonction **'MAVec(fichierInversePondere,indexed,requete,fonction)'**. Cette fonction prend en entrée le fichier inverse des poids, l'index 'indexedDict' (indexed), la requête après traitement et la fonction d'appariement sélectionnée.
   Elle fait appel à la fonction d'appariement correspondante après test et retourne la liste des documents répondants à la requête. Ces derniers sont ordonnés par ordre décroissant selon la valeur de similarité. Voir implantation ci-dessous

```
412  #Module d'Appariemment entre requete document
413  def MA_Vec(fichier_inverse_pondere,indexed,requete,fonction):
414      listDocSim=dict()
415      if fonction == "ProduitInterne":
416          for doc in indexed:
417              documentWords=list()
418              for mot in indexed[doc]:
419                  documentWords.append(mot)
420              NewReq=list()
421              for mot in requete:
422                  if mot.lower() in documentWords:
423                      NewReq.append(mot.lower())
424              if(len(NewReq)>0):
425                 listDocSim[doc]=ProduitInterne(doc,NewReq,fichier_inverse_pondere)
426          listDocSim=sorted(listDocSim.items(), key=lambda t: t[1])
427      elif fonction == "CoefDeDice":
428          for doc in indexed:
429              documentWords=list()
430              for mot in indexed[doc]:
431                  documentWords.append(mot)
432              NewReq=list()
433              for mot in requete:
434                  if mot in documentWords:
435                      NewReq.append(mot)
436              if(len(NewReq)>0):
437                  listDocSim[doc]=CoefDeDice(doc,NewReq,requete,documentWords,fichier_inverse_pondere)
438          listDocSim=sorted(listDocSim.items(), key=lambda t: t[1])
439      elif fonction == "Cosinus":
440          for doc in indexed:
441              documentWords=list()
442              for mot in indexed[doc]:
443                  documentWords.append(mot)
444              NewReq=list()
445              for mot in requete:
446                  if mot in documentWords:
447                      NewReq.append(mot)
448              if(len(NewReq)>0):
449                 listDocSim[doc]=Cosinus(doc,NewReq,requete,documentWords,fichier_inverse_pondere)
450          listDocSim=sorted(listDocSim.items(), key=lambda t: t[1])
451      elif fonction == "Jaccard":
452          for doc in indexed:
453              documentWords=list()
454              for mot in indexed[doc]:
455                  documentWords.append(mot)
456              NewReq=list()
457              for mot in requete:
458                  if mot in documentWords:
459                      NewReq.append(mot)
460              if(len(NewReq)>0):
461                 listDocSim[doc]=Jaccard(doc,NewReq,requete,documentWords,fichier_inverse_pondere)
462          listDocSim=sorted(listDocSim.items(), key=lambda t: t[1])
463      return listDocSim
```

5 Interface et tests:

5.1 Présentation de l'interface :

Nous avons implémenter une interface qui permet à l'utilisateur de faire entrer des requêtes et recevoir des réponses de recherche.
Notre interface contient les éléments suivants :

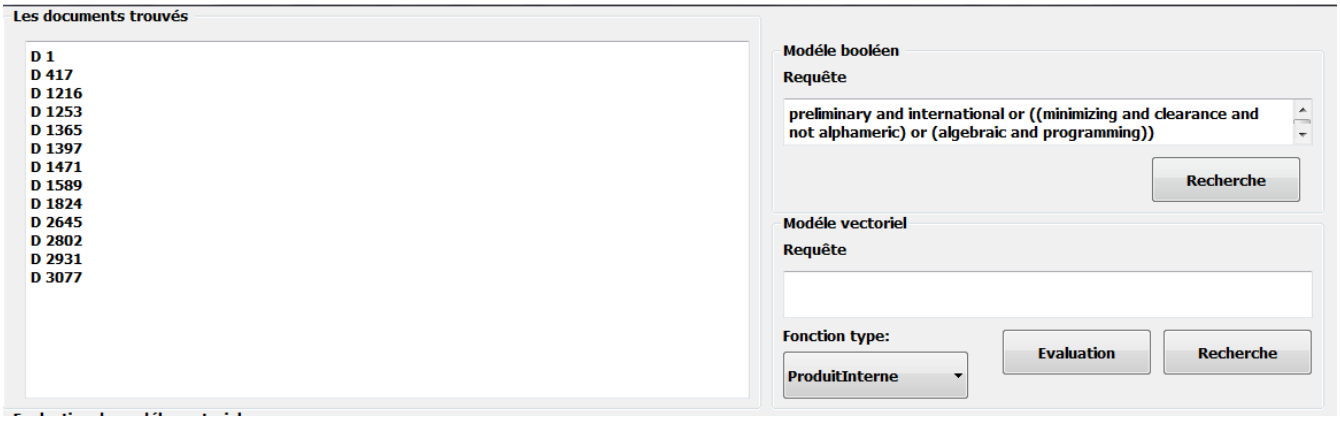


1. Introduire une requête booléenne.
2. Introduire une requête vectorielle ou un numéro de requête (un entier) d'évaluation.
3. Choisir la fonction d'appariement.
4. Affichages des documents retournés par notre SRI.
5. Affichage des documents pertinents à une requête de 'qrels'.
6. Affichage des documents pertinents retournés par notre SRI sur une requête de 'query'.
7. Calcul de rappel et précision de notre SRI.

5.2 Démonstration des tests:

5.2.1 Modèle booléen:

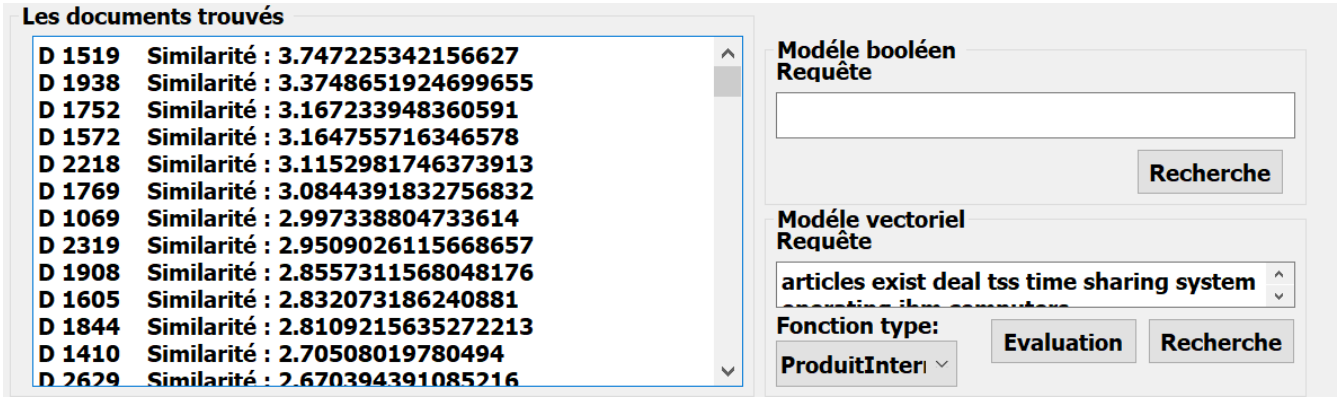
Tout d'abord, l'utilisateur introduit la requête booléenne et clic sur le bouton recherche. Les résultats sont affichés à droite. Voici une illustration :



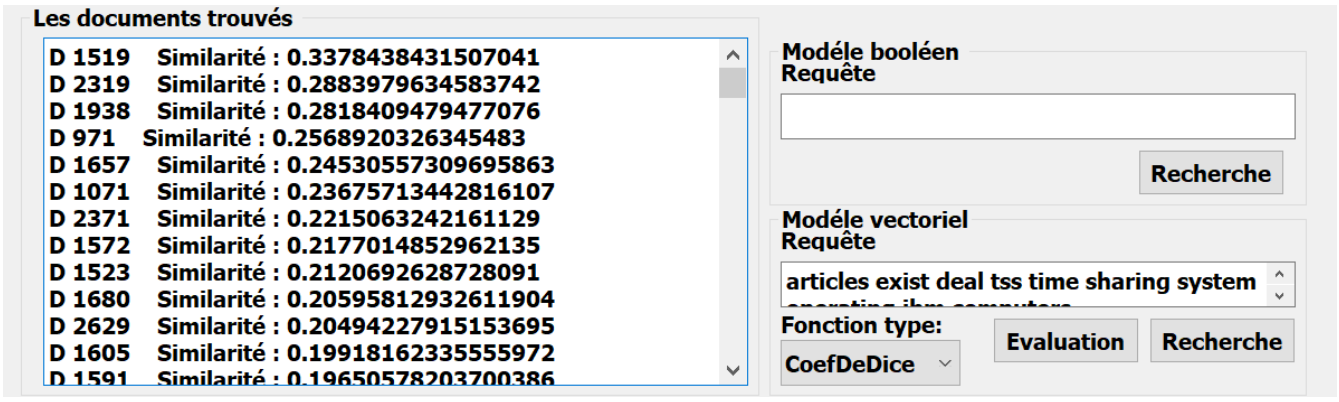
5.2.2 Modèle vectoriel:

L'utilisateur entre la requête, choisi la fonction de similarité et appui sur le bouton recherche. Voici un exemple :

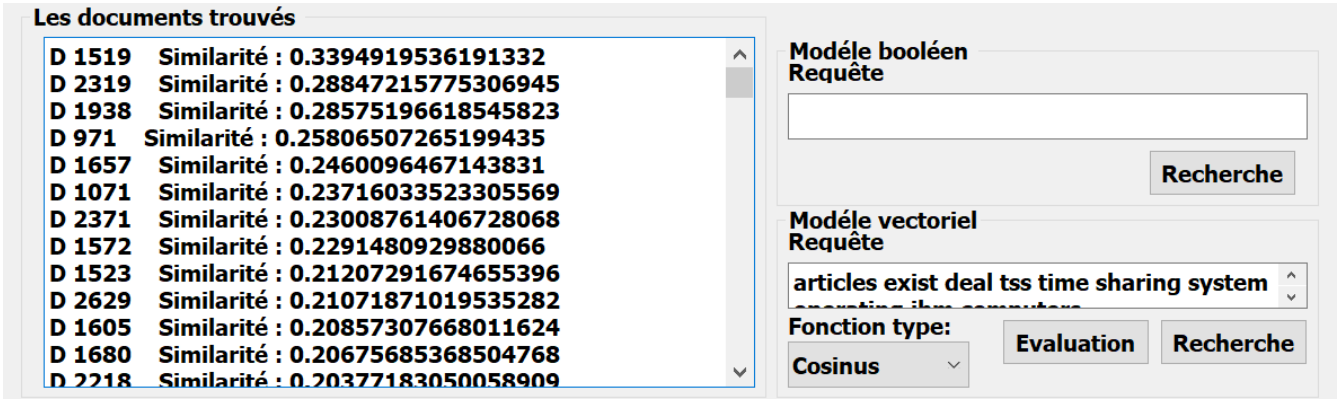
Produit interne :



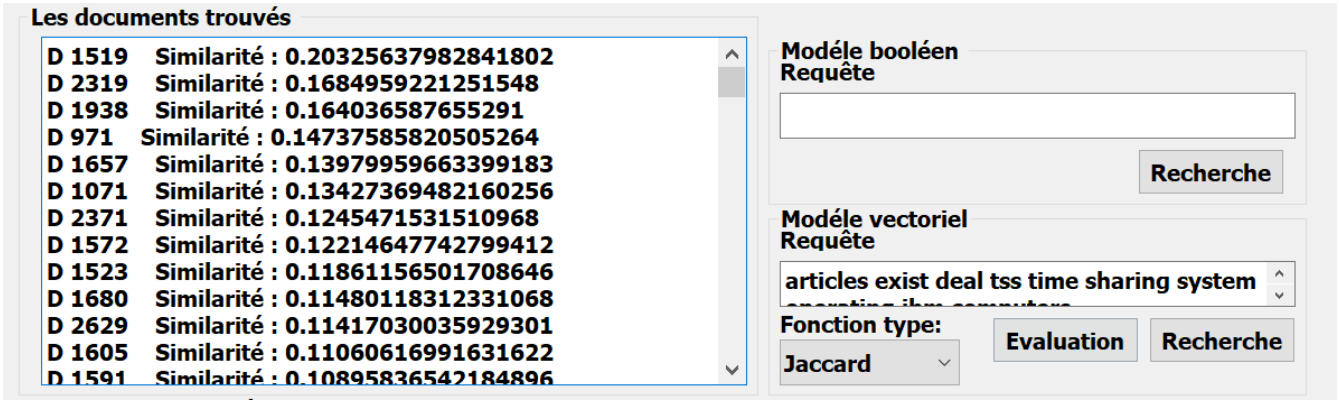
Coefficient de Dice :



Cosinus :



Jaccard :



5.3 Comparaison :

- On peut remarqué facilement que les 4 formules d'appariements du modèle vectoriel nous on généré les mêmes documents.
- L'ordonnancement des documents trouvés diffère d'une formule a une autre

6 Évaluation du modèle vectoriel

Cette partie consiste a l'évaluation des résultats obtenus par notre modèle en utilisant les deux fichiers de la collection CACM : (query.text et qrels.text)

6.1 Chargement des requêtes de test et résultats de test

On doit parcourir les deux fichiers query.text et qrels.text pour extraire leur contenu.

Chargement query.text:

On prend on considération les champs I,W on procède comme pour le fichier cacm.all vu dans la phase d'indexation sauf que le résultat final sera chargé dans un dictionnaire **Query** :

$$Query = \{Numero\ requête : [Liste\ des\ termes], ...\}$$

cette opération est réalise par la fonction ExtractionQuery(path,stopwordsList,ponctuationList) voir son implémentions ci-dessous.

```
78  def ExtractionQuery(path,stopwords_list,ponctuation_list):
79      queryfile = open(path, "r")
80
81      #lecture des lignes
82      lines = queryfile.readlines()
83
84      #dictionnaire des querys {queryIDF:liste des lignes des query}
85      DictQuery ={}
86
87      #parcourt et separation par rapport au .I, .T et .W ignorer les autres champs
88      #c'est plus facile d'utiliser la fonction while pour avancer ligne par ligne et
89      i=0
90      while(i<len(lines)):
91          #on recupere la ligne
92          line = lines[i]
93
94          #si la ligne commence par .I on recupere l'identifiant du document
95          if(line.startswith('.I')):
96              queryIDF = int(line.split()[1])
97              DictQuery[queryIDF]=""
98
```

```
        #sinon si la liste commence par .W, on recupere les lignes du champ resume
        if(line.startswith('.W')):
            i+=1
            resume = ""
            while((i<len(lines)) and (re.findall('\.([TWBANX]|I [0-9]+)', lines[i]))==[]):
                resume = resume +" "+lines[i]
                i+=1
            DictQuery[queryIDF] = DictQuery[queryIDF] + resume
            i-=1
        i+=1
    Query={}
    for key in DictQuery.keys():
        text=DictQuery[key]
        word_list=Stopword_elimination(text,ponctuation_list,stopwords_list)
        cleanWord_list = []
        for word in word_list:
            if word not in cleanWord_list:
                cleanWord_list.append(word)
        Query[key]=cleanWord_list
    #print(Query)
    return Query
```


Chargement qrels.text:

On charge dans un dictionnaire **Qrels** le numéro de la requête associé avec la liste des documents de la reponse pour cette requête, cette opération est réalise par la fonction ExtractionQrels(path),voir son implémention ci-dessous.

```
136  def ExtractionQrels(path):
137      qrelsfile = open(path, "r")
138
139      #lecture des lignes
140      lines = qrelsfile.readlines()
141      Qrels={}
142      i=0
143      while(i<len(lines)):
144          line = lines[i].split(" ")
145          numQuery=int(line[0])
146          if  numQuery not in Qrels.keys():
147              list = []
148              list.append(int(line[1]))
149              Qrels[numQuery] = list
150          else:
151              list.append(int(line[1]))
152              Qrels[numQuery]=list
153          i +=1
154      #print(Qrels)
155      return Qrels
156
```

6.2 Evaluation

Après les chargement des données qu'on vas utilisé pour notre évaluation, on passe a l'évaluation du modèle. Pour cela on a créé une fonction evaluation() qui récupère le contenu du numéro de la requête entré par l'utilisateur du dictionnaire **Query** (la requête) ainsi que ses réponse du dictionnaire **Qrels** (liste des documents pertinent), En suite on fait appel a la fonction MAVec() vu précédemment pour récupéré la liste des documents trouvé par notre modèle, Enfin on calcul le rappel et la précision selon les listes sélectionnés avec les deux fonctions Rappel() et Prescision() qu'on à créé. Voir implantation ci-dessous.

```python
def evaluation(self):
    if(self.textEditRequeteVectoriel.toPlainText()!=""):
        try:
            Numreq = int(self.textEditRequeteVectoriel.toPlainText())
            requete = dictQuery[Numreq]
            print(requete)
            fonction=self.comboBoxTypeFonctionAppa.currentText()
            list_DocsSimilarite=MA_Vec(FichierInvrsePoids,indexedDict,requete,fonction)
            listDocselected = []
            S=0.12
            for element in reversed(list_DocsSimilarite):
                if(element[1]>= S):
                    listDocselected.append(int(element[0]))
            #limitation du T
            T=50
            if(len(listDocselected)>T):
                print(str(len(listDocselected)))
                listDocselected=listDocselected[:T]
            print(listDocselected)
            DocPertinent = dictQrels[Numreq]
            print(DocPertinent)
            DocPertientTrouv=ListeDocPertientTrouv(listDocselected,DocPertinent)
            print(DocPertientTrouv)
            if(len(DocPertinent)>0):
                rappel=Rappel(DocPertientTrouv,DocPertinent)
            if(len(listDocselected)>0):
                precision=Precision(DocPertientTrouv,listDocselected)
            self.textBrowserDocsPertinents.clear()
```



```python
#fonction qui retourne le nombre de document pertinents trouvés
def ListeDocPertientTrouv(ListeDocTrouv, ListeDocPertinent):
    ListeDocPertientTrouv=[]
    for document in ListeDocTrouv:
        if(document in ListeDocPertinent):
            ListeDocPertientTrouv.append(document)
    return ListeDocPertientTrouv


#calcul du rappel d'une requete donnée ==> nombre de documents pertinents trouvés/nombre de document
def Rappel(ListeDocPertientTrouv, ListeDocPertinent):
    return len(ListeDocPertientTrouv)/len(ListeDocPertinent)

#calcul de précision d'un requete donnée ==> nombre de documents pertinents sélécionnés/nombre total
def Precision(ListeDocPertientTrouv, ListeDocTrouv):
    return len(ListeDocPertientTrouv)/len(ListeDocTrouv)

```

6.3 Expérimentations

On a utilisé deux paramètres (S,T) pour trouvé les rappels et les précisions idéal pour notre modèle, voir capture d'écran.

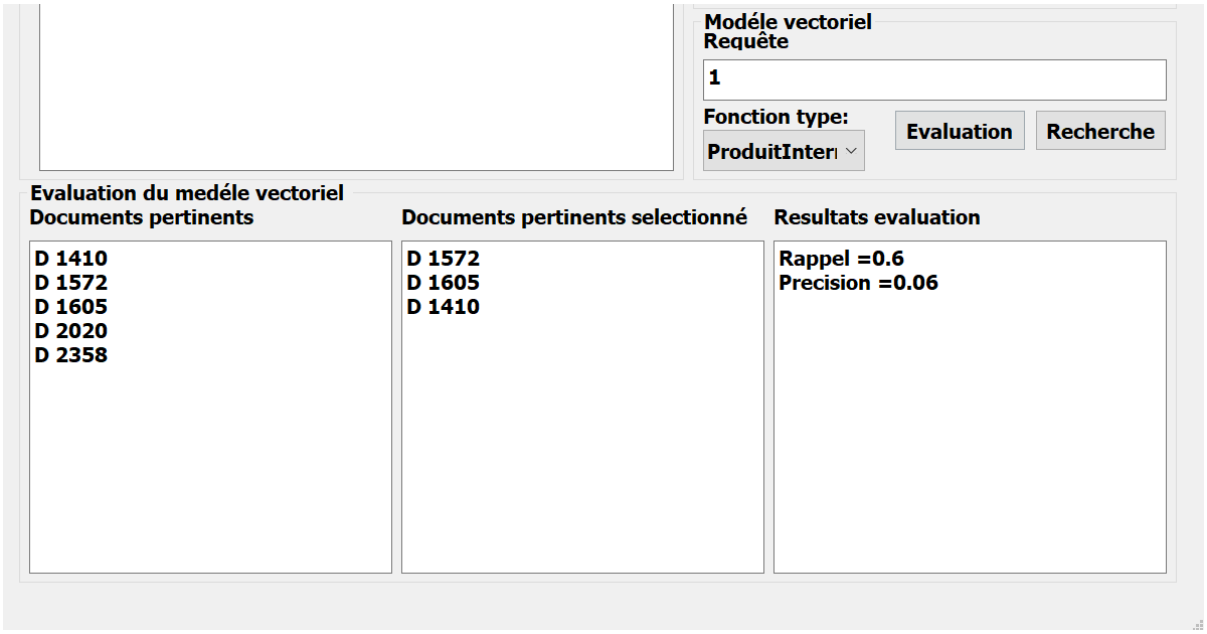
```
294  def evaluation(self):
295      if(self.textEditRequeteVectoriel.toPlainText()!=""):
296          try:
297              Numreq = int(self.textEditRequeteVectoriel.toPlainText())
298              requete = dictQuery[Numreq]
299              print(requete)
300              fonction=self.comboBoxTypeFonctionAppa.currentText()
301              list_DocsSimilarite=MA_Vec(FichierInvrsePoids,indexedDict,requete,fonction)
302              listDocselected = []
303       Seuil  S=0.12
304              for element in reversed(list_DocsSimilarite):
305                  if(element[1]>= S):
306                      listDocselected.append(int(element[0]))
307              #limitation du T
308       Taille  T=50
309              if(len(listDocselected)>T):
310                  print(str(len(listDocselected)))
311                  listDocselected=listDocselected[:T]
312              print(listDocselected)
313              DocPertinent = dictQrels[Numreq]
314              print(DocPertinent)
315              DocPertientTrouv=ListeDocPertientTrouv(listDocselected,DocPertinent)
316              print(DocPertientTrouv)
317              if(len(DocPertinent)>0):
318                  rappel=Rappel(DocPertientTrouv,DocPertinent)
```

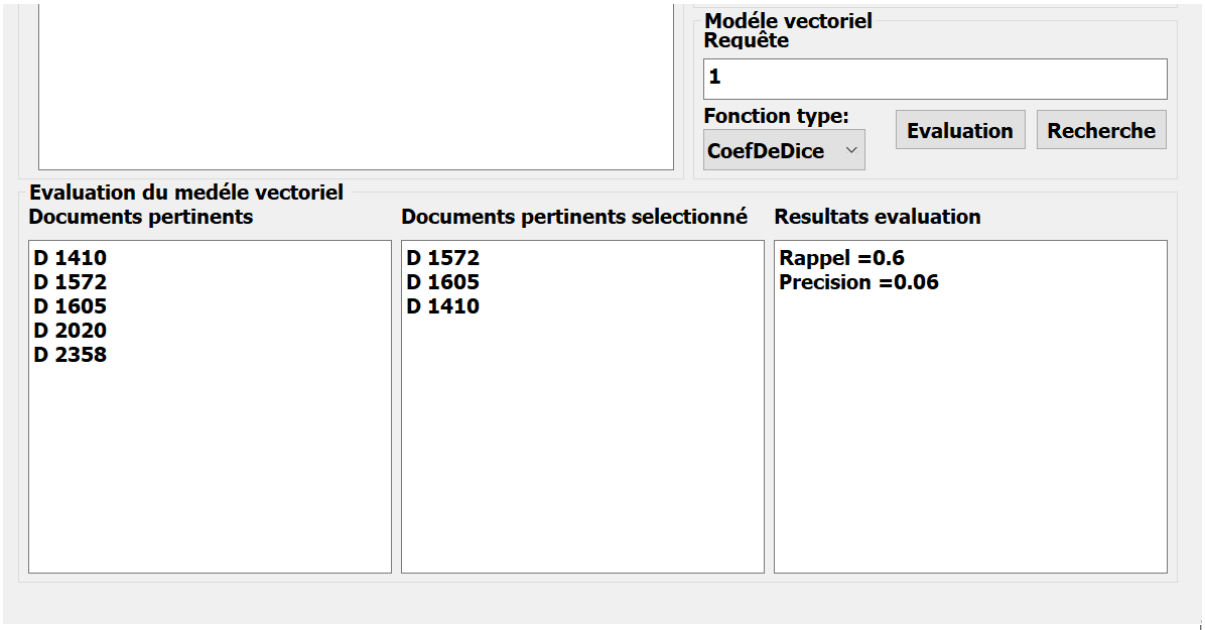
Pour fixer la taille et le seuil de similarité on a fait plusieurs tests avec différentes requêtes de teste appliqué sur la fonction cosinus, selon les résultats des similarités on gros retourné par notre modèle pour les 64 requête on prend le S est le T qui maximise le Rappel est la précision. Nous avons constaté que un S=0.12 et un T=50 maximise notre rappel et notre précision du modèle

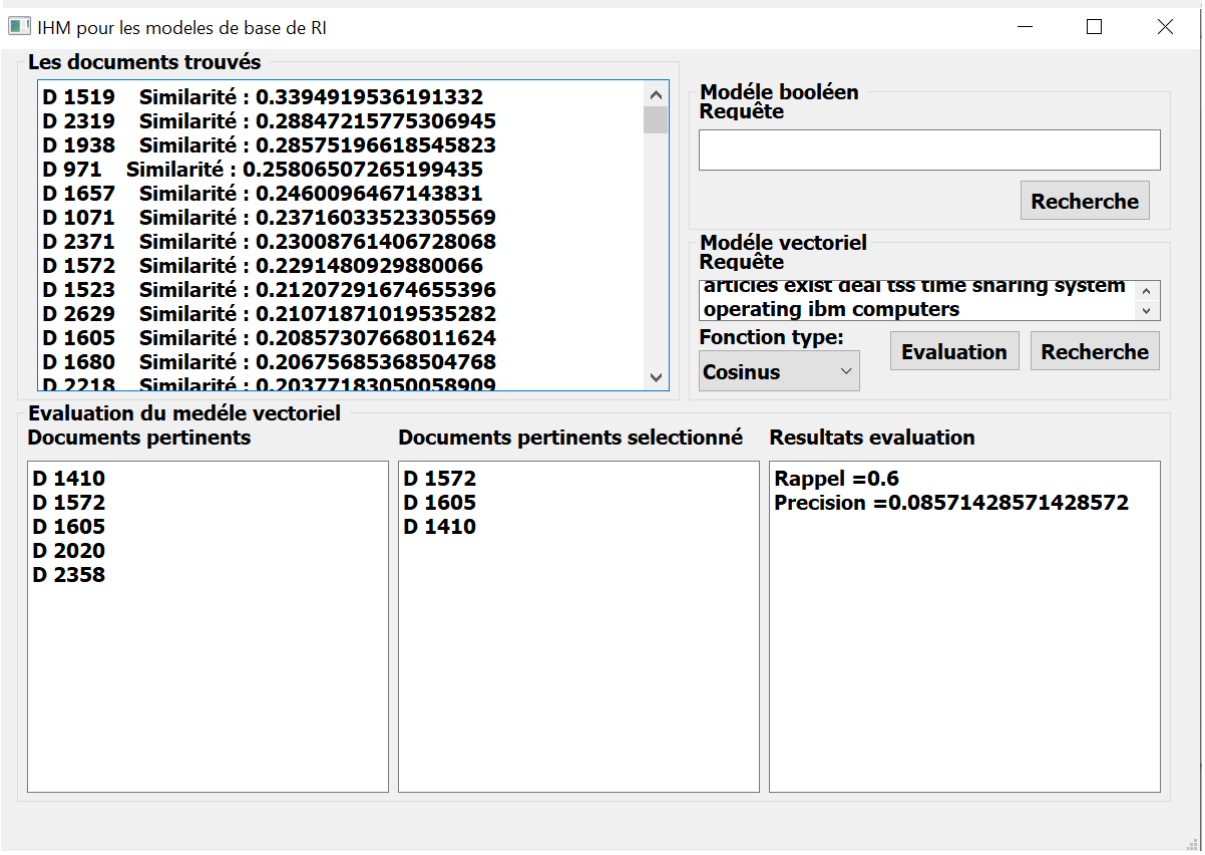
6.4 Résultats

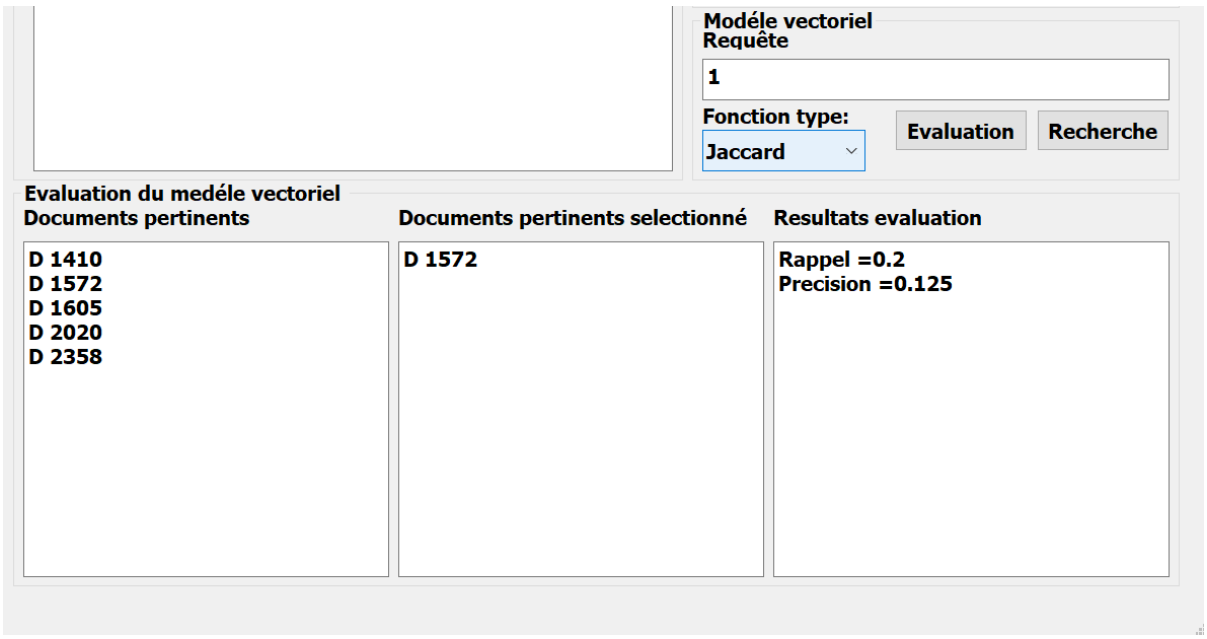
Voici des exemples de tests de requêtes avec les 4 fonctions, après fixation des paramètres :

- Requête 1 :

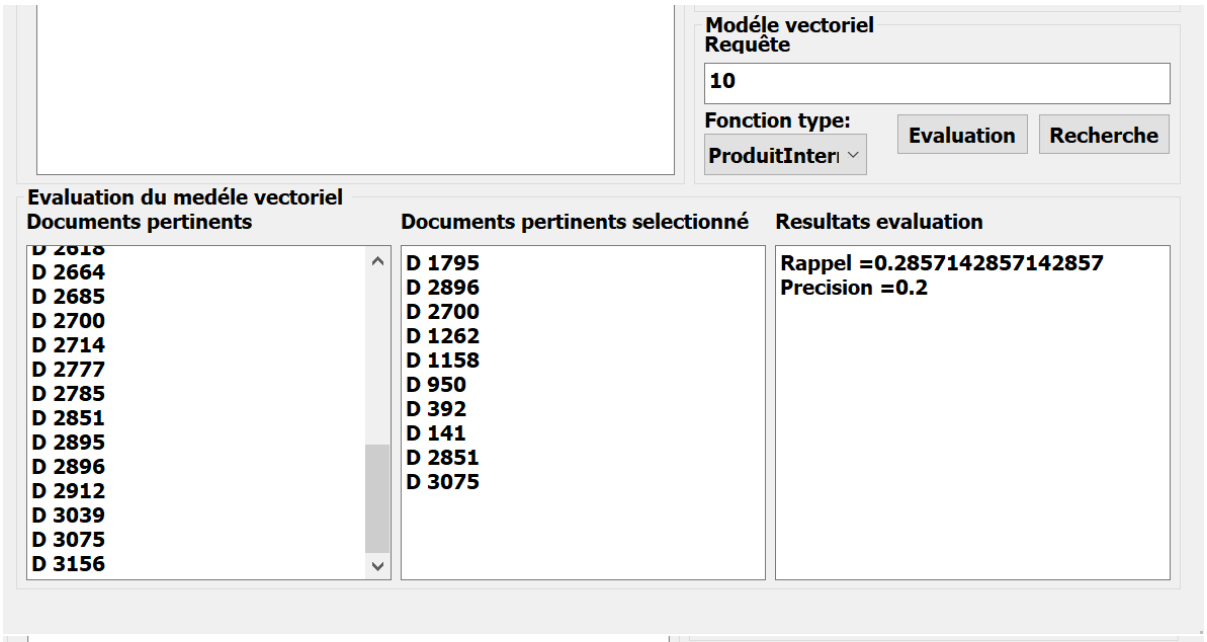


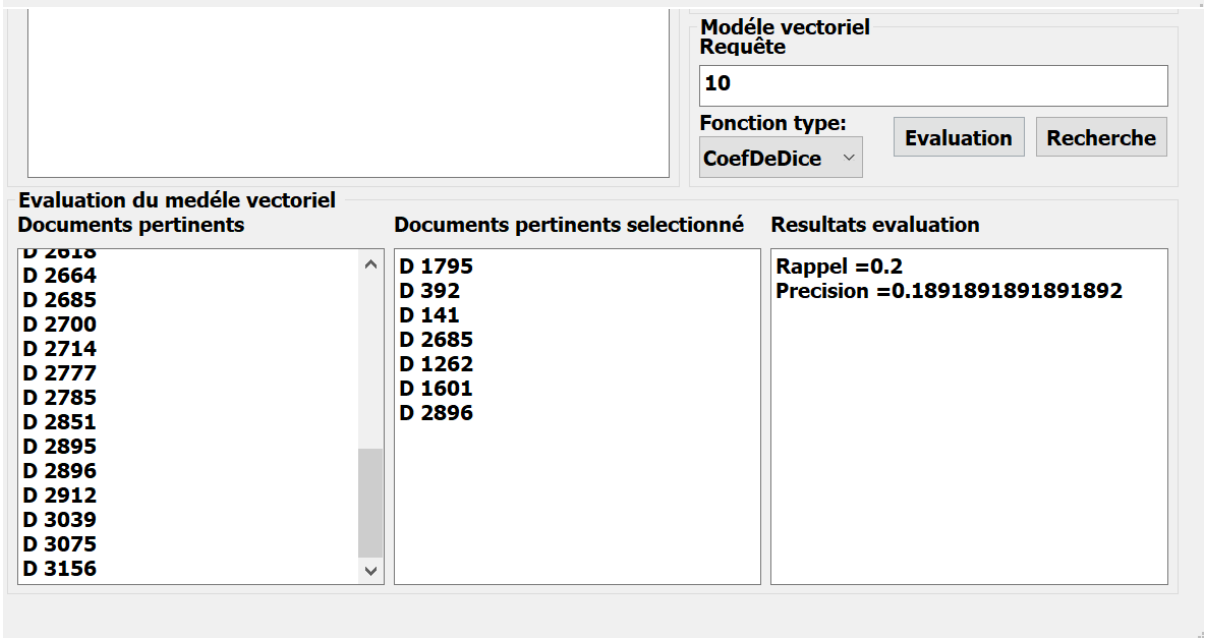

Modéle vectoriel
Requête
1
Fonction type:
CoefDeDice
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 1410
D 1572
D 1605
D 2020
D 2358
D 1572
D 1605
D 1410
Rappel =0.6
Precision =0.06

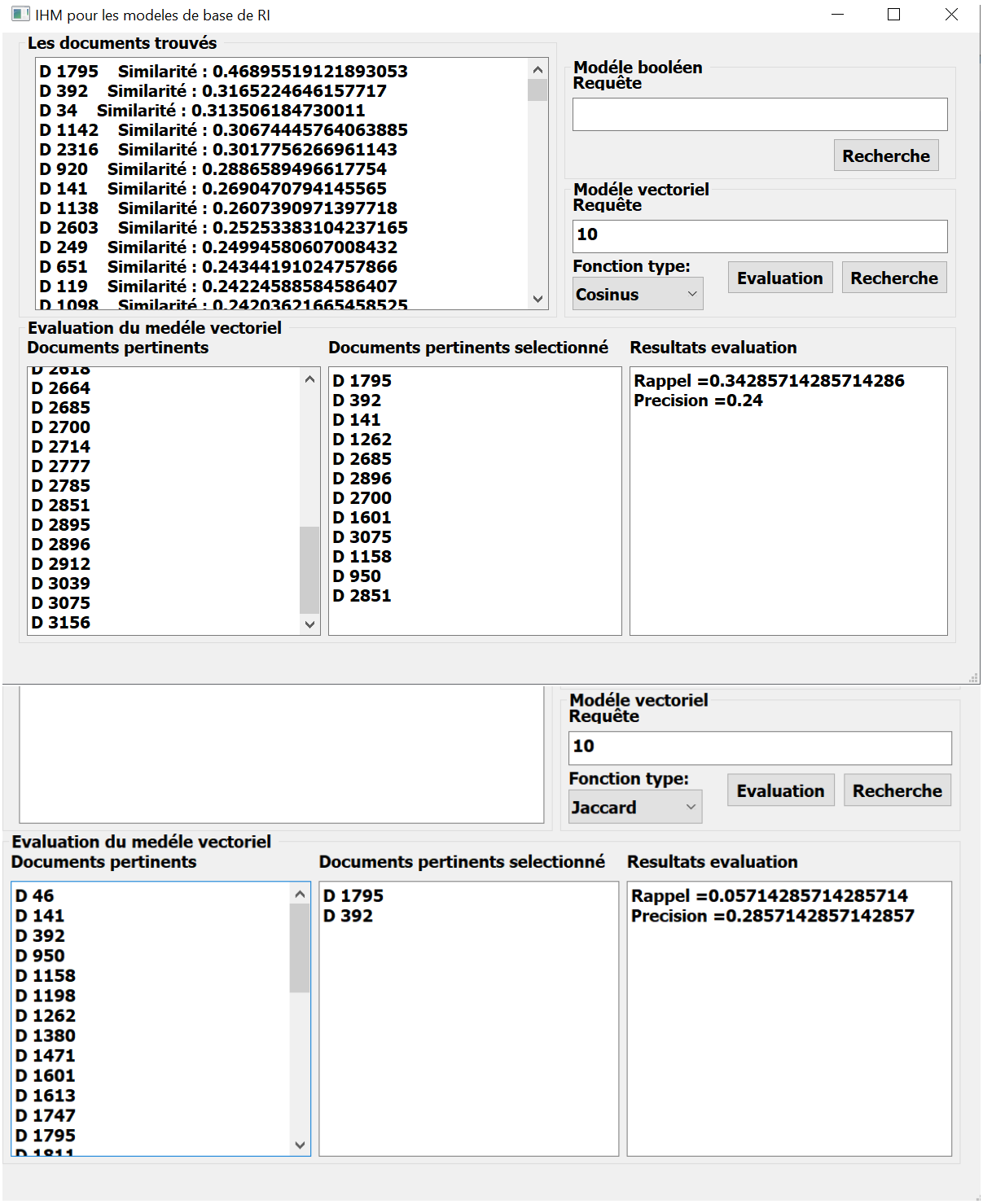

IHM pour les modeles de base de RI
Les documents trouvés
D 1519 Similarité : 0.3394919536191332
D 2319 Similarité : 0.28847215775306945
D 1938 Similarité : 0.28575196618545823
D 971 Similarité : 0.25806507265199435
D 1657 Similarité : 0.2460096467143831
D 1071 Similarité : 0.23716033523305569
D 2371 Similarité : 0.23008761406728068
D 1572 Similarité : 0.2291480929880066
D 1523 Similarité : 0.21207291674655396
D 2629 Similarité : 0.21071871019535282
D 1605 Similarité : 0.20857307668011624
D 1680 Similarité : 0.20675685368504768
Modéle booléen
Requête
Recherche
Modéle vectoriel
Requête
articles exist deal tss time sharing system operating ibm computers
Fonction type:
Cosinus
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 1410
D 1572
D 1605
D 2020
D 2358
D 1572
D 1605
D 1410
Rappel =0.6
Precision =0.08571428571428572


Modéle vectoriel
Requête
1
Fonction type:
Jaccard
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 1410
D 1572
D 1605
D 2020
D 2358
D 1572
Rappel =0.2
Precision =0.125

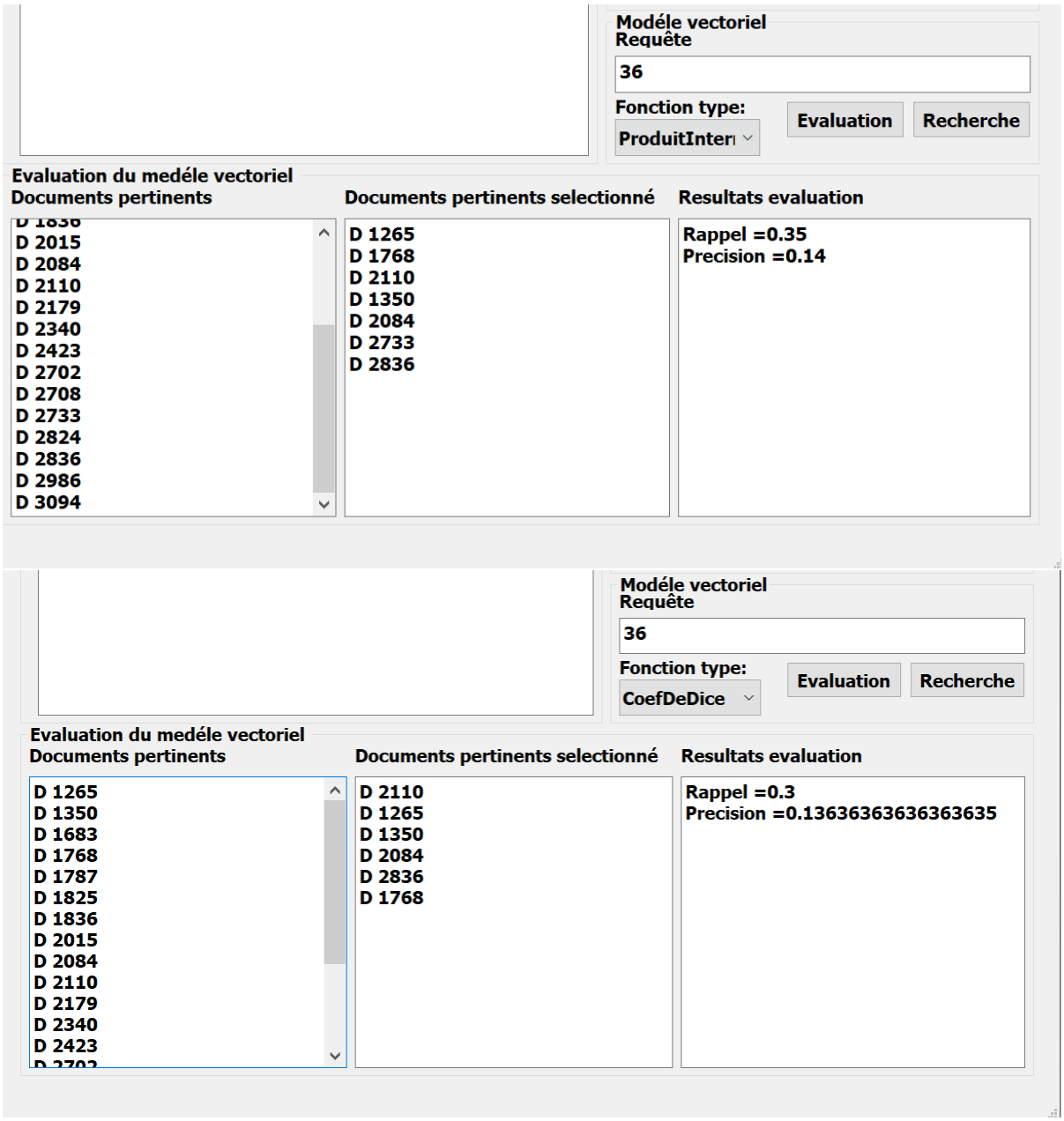
- Requête 10 :

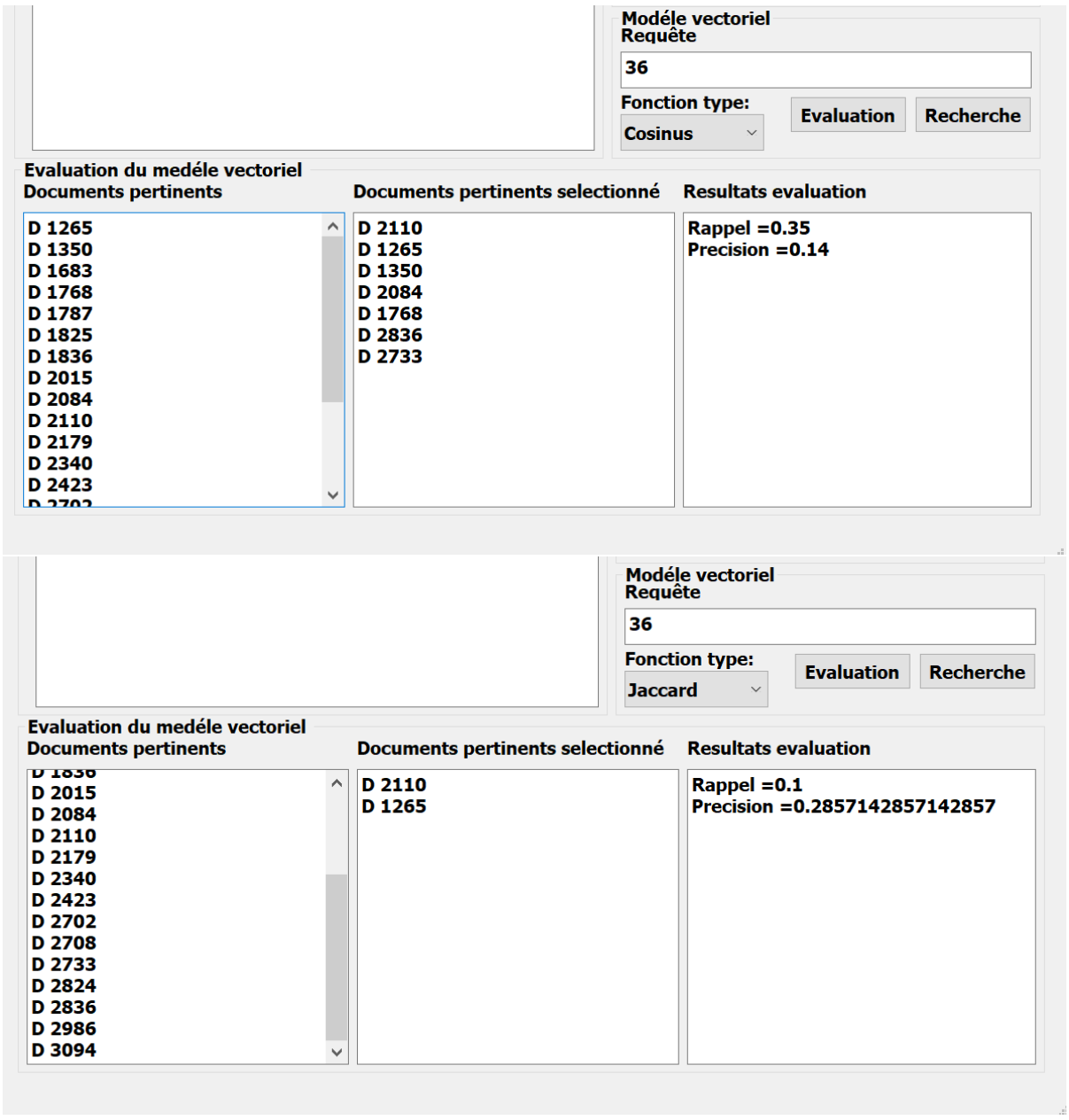

Modéle vectoriel
Requête
10
Fonction type:
ProduitInter
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 2664
D 2685
D 2700
D 2714
D 2777
D 2785
D 2851
D 2895
D 2896
D 2912
D 3039
D 3075
D 3156
D 1795
D 2896
D 2700
D 1262
D 1158
D 950
D 392
D 141
D 2851
D 3075
Rappel =0.2857142857142857
Precision =0.2


Modéle vectoriel
Requête
10
Fonction type:
CoefDeDice
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 2664
D 2685
D 2700
D 2714
D 2777
D 2785
D 2851
D 2895
D 2896
D 2912
D 3039
D 3075
D 3156
D 1795
D 392
D 141
D 2685
D 1262
D 1601
D 2896
Rappel =0.2
Precision =0.1891891891891892

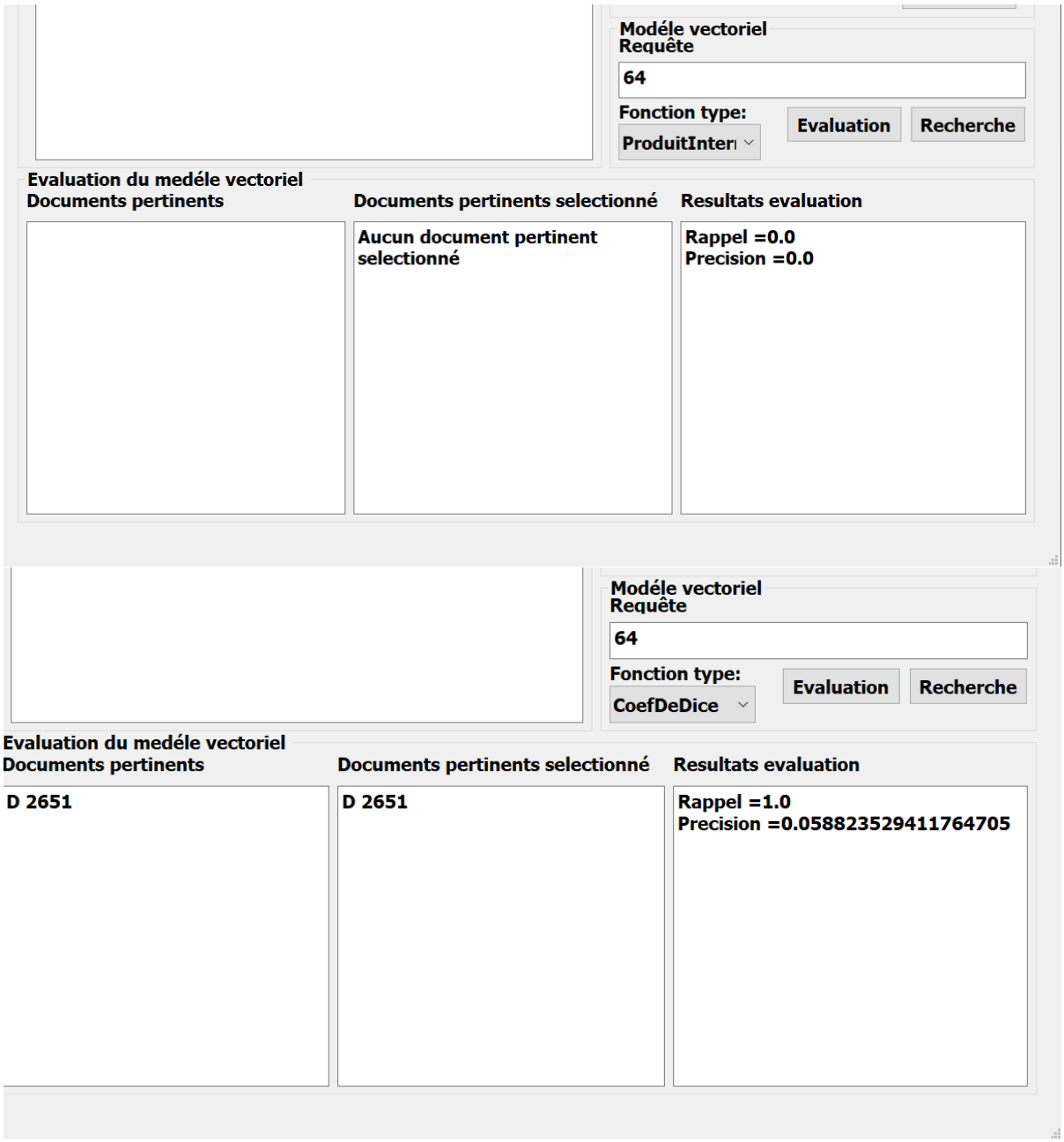

IHM pour les modeles de base de RI
Les documents trouvés
D 1795 Similarité : 0.46895519121893053
D 392 Similarité : 0.3165224646157717
D 34 Similarité : 0.313506184730011
D 1142 Similarité : 0.30674445764063885
D 2316 Similarité : 0.3017756266961143
D 920 Similarité : 0.2886589496617754
D 141 Similarité : 0.2690470794145565
D 1138 Similarité : 0.2607390971397718
D 2603 Similarité : 0.25253383104237165
D 249 Similarité : 0.24994580607008432
D 651 Similarité : 0.24344191024757866
D 119 Similarité : 0.24224588584586407
Modéle booléen
Requête
Recherche
Modéle vectoriel
Requête
10
Fonction type:
Cosinus
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
D 2664
D 2685
D 2700
D 2714
D 2777
D 2785
D 2851
D 2895
D 2896
D 2912
D 3039
D 3075
D 3156
Documents pertinents selectionné
D 1795
D 392
D 141
D 1262
D 2685
D 2896
D 2700
D 1601
D 3075
D 1158
D 950
D 2851
Resultats evaluation
Rappel =0.34285714285714286
Precision =0.24
Modéle vectoriel
Requête
10
Fonction type:
Jaccard
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
D 46
D 141
D 392
D 950
D 1158
D 1198
D 1262
D 1380
D 1471
D 1601
D 1613
D 1747
D 1795
Documents pertinents selectionné
D 1795
D 392
Resultats evaluation
Rappel =0.05714285714285714
Precision =0.2857142857142857

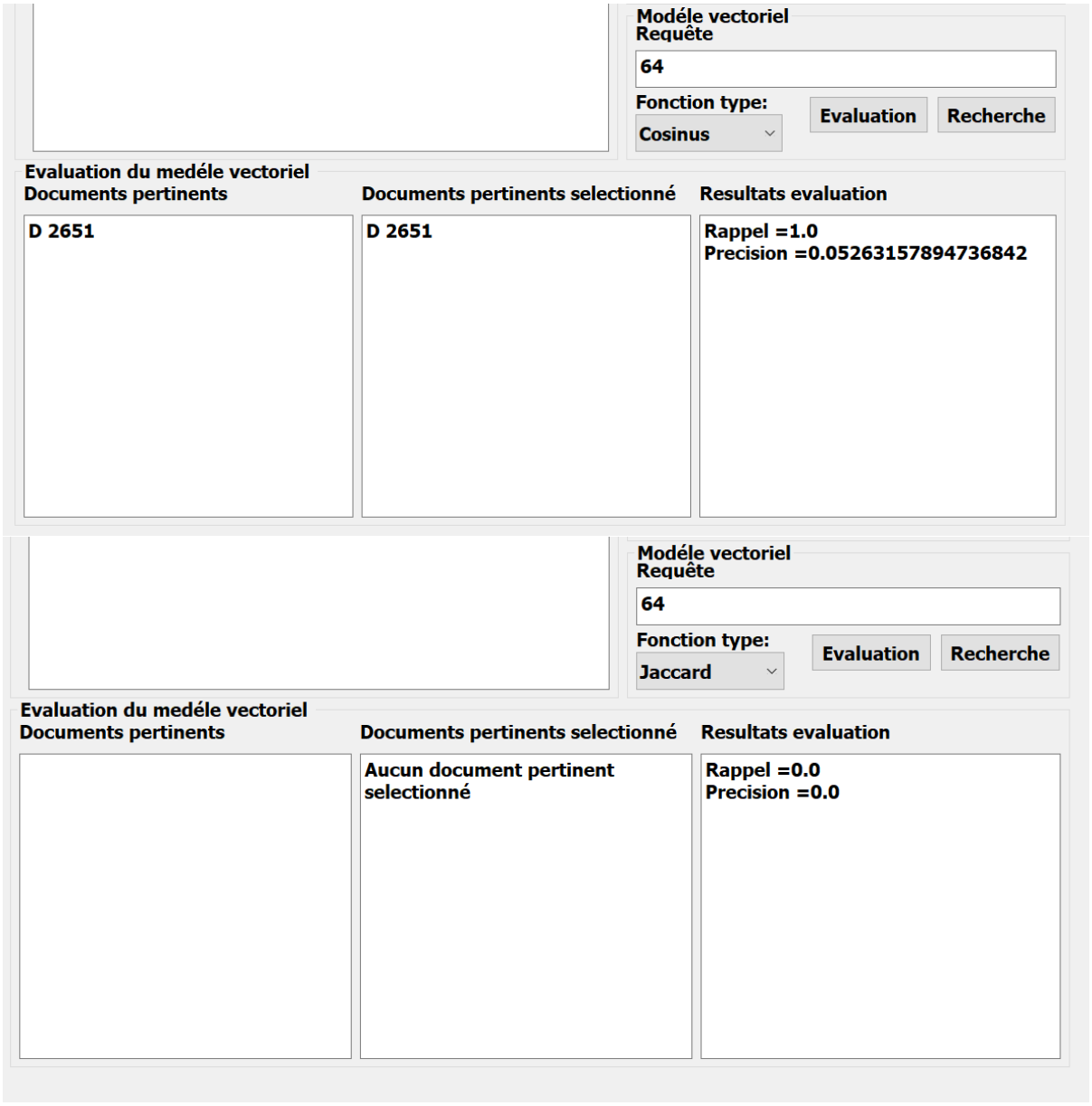
- Requête 36 :


Modéle vectoriel
Requête
36
Fonction type:
ProduitInter
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 2015
D 2084
D 2110
D 2179
D 2340
D 2423
D 2702
D 2708
D 2733
D 2824
D 2836
D 2986
D 3094
D 1265
D 1768
D 2110
D 1350
D 2084
D 2733
D 2836
Rappel =0.35
Precision =0.14
Modéle vectoriel
Requête
36
Fonction type:
CoefDeDice
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 1265
D 1350
D 1683
D 1768
D 1787
D 1825
D 1836
D 2015
D 2084
D 2110
D 2179
D 2340
D 2423
D 2110
D 1265
D 1350
D 2084
D 2836
D 1768
Rappel =0.3
Precision =0.13636363636363635


Modéle vectoriel
Requête
36
Fonction type:
Cosinus
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 1265
D 1350
D 1683
D 1768
D 1787
D 1825
D 1836
D 2015
D 2084
D 2110
D 2179
D 2340
D 2423
D 2110
D 1265
D 1350
D 2084
D 1768
D 2836
D 2733
Rappel =0.35
Precision =0.14
Modéle vectoriel
Requête
36
Fonction type:
Jaccard
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 2015
D 2084
D 2110
D 2179
D 2340
D 2423
D 2702
D 2708
D 2733
D 2824
D 2836
D 2986
D 3094
D 2110
D 1265
Rappel =0.1
Precision =0.2857142857142857

- Requête 64 :


Modéle vectoriel
Requête
64
Fonction type:
ProduitInter
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
Aucun document pertinent selectionné
Rappel =0.0
Precision =0.0
Modéle vectoriel
Requête
64
Fonction type:
CoefDeDice
Evaluation
Recherche
Evaluation du medéle vectoriel
Documents pertinents
Documents pertinents selectionné
Resultats evaluation
D 2651
D 2651
Rappel =1.0
Precision =0.058823529411764705



6.5 Remarques:

On remarque qu'en général, le Produit interne et le coefficient de Dice donnent des valeurs proches. La fonction qui donne une précision améliorée est la fonction de Jaccard. Le rappel optimal est généralement retourné par la fonction Produit interne et par fois cosinus.

Conclusion

Lors de la fin de ce TP nous avons réaliser un mini moteur de recherche spécifié à la collection (CACM).Ce projet nous a permis d'appliquer les notions du RI vu en cours et voir l'utilité et l'importance de ces derniers.
La phase d'implémentation nous a permit de constater que la réalisation nécessite les points suivants :

- Une grande précision de mesures de similarité.
- Un bon choix du modèle.
- une bonne préparation de données
- une structure de stockage adéquate.