Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique
Département d'informatique

Rapport de projet

Module : **Recherche d'information**

Master 1 SII

Implémentation de différentes méthodes de recherche d'information

- Réalisé par :

  BENHADDAD Wissam

  BOURAHLA Yasser

29-11-2018

Table des matières

Table des figures

Liste des tableaux

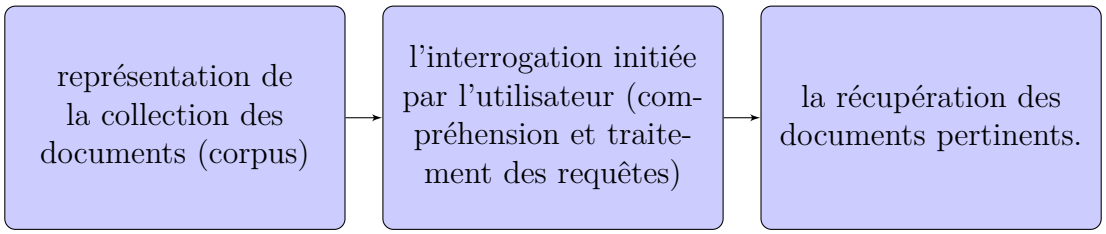
CHAPITRE 1

INTRODUCTION

1.1 Problématique et objectifs

Au fil des années, et depuis l'explosion du volume de données dans disponible sur Internet, il était devenu de plus en plus difficile aux utilisateur d'accéder rapidement et efficacement aux sources d'informations, cette augmentation du niveau de complexité d'accès aux données a motivé les chercheurs a développer de nouvelles méthodes pour faciliter la recherche d'information.

Généralement, le processus recherche d'information peut être divisé en trois étapes comme le montre le schéma suivant :



L'une des applications les plus courantes et les plus connues de la recherche d'information est peut-être la récupération de documents textuels sur Internet. Avec sa croissance récente, Internet devient rapidement le principal média de communication pour les entreprises et l'information académique. Il est donc essentiel de pouvoir exploiter le bon document à partir de ce vaste océan d'informations. C'est en fait l'une des principales forces motrices pour le développement de la recherche documentaire. À ce jour, de nombreux systèmes relativement efficaces ont été développés.

L'une des premières remarques qui viennent à l'esprit est : "comment rendre ce processus aussi rapide et intuitif que possible, sans se soucier du volume de données dans le quel l'information est recherchée ?", le but de ce projet est donc d'étudier les méthode proposées pour remédier à ce problème, de les implémenter sur machine, et de comparer leurs résultats et performance sur une corpus de test préalablement construit, tout cela dans l'optique de mieux comprendre pourquoi cette tâche(la RI) est si complexe et ardue.

1.2 Définitions

Pour mieux comprendre le contexte dans le quel nous nous trouvons, nous allons définir quelques notions en rapport avec la **RI** :

1.2.1 La recherche d'information

La RI [1] peut être définie comme suit :

> "La signification du terme « Recherche d'information » peut être très large. Cependant, en tant que domaine d'étude académique il peut être défini comme le domaine qui étudie les méthodes permettant de trouver des informations pertinentes dans un corpus, composé d'une collection de documents non-structurés. Il s'agit essentiellement de faciliter d'accès de l'utilisateur à un grand volume de données"

1.2.2 L'indexation de documents

L'indexation de documents est un l'opération qui consiste à organiser un ensemble de documents et faciliter ultérieurement la recherche de contenu dans cette collection. Pour y arriver le système essaye de construire un index(une liste de descripteurs à chacun desquels est associée une liste des documents et/ou parties de documents auxquels ce descripteur renvoie), dans ce projet nous allons utiliser deux méthodes d'indexations :

1.2.3 Indexation par fichier inverse

Ce modèle d'indexation construit une représentation de correspondance entre un document **d** et une entité (généralement un terme) **t** contenu dans le document, construisant ainsi un dictionnaire **Dico** dont les entrés sont de la forme suivante :

$$Dico[d, t] \rightarrow V$$

où V peut être une mesure qualifiant l'appartenance du terme **t** dans le document **d** (fréquence d'apparition, poids, position ...).dans ce projet nous utiliserons

1.2.4 La mesure TF :

TF pour **Term-Frequency** est une mesure qui donne simplement combien un terme est important dans un document **d**, sa formule la plus simple est la suivante :

$$TF[w, d] = nombre\_occurence\_w\_dans\_d$$

La forme normalisé est la suivante :

$$TF[w, d] = \frac{nombre\_occurence\_w\_dans\_d}{nombre\_total\_termes\_dans\_d}$$

1. Recherche d'Information

1.2.5 La mesure IDF

IDF pour **Inverse Document Frequency** est une mesure quant à elle qui nous renseigne sur combien de gain d'information un terme peut fournir par rapport à **tout** les documents, sa formule est la suivante :

Soient :

- w le terme au quel on s'intéresse.
- d_w le nombre de document où le terme w apparaît (i.e $1 + |\{d \in D, TF[w, d] \neq 0\}|$)
- D l'ensemble de tout les documents.

$$IDF[w, D] = \log \frac{|D|}{d_w}$$

1.2.6 TF-IDF comme poids d'un document

La mesure **TF-IDF** est le produit entre les deux mesures qui la composent, ainsi :

$$TF - IDF[w, d, D] = TF[w, d] * IDF[w, D]$$

Analyse : cette mesure représente le poids d'un terme w dans un document d par rapport à un corpus (ensemble de documents) D, ce poids va donc indiquer à quel point un terme est présent dans un petit nombre de documents, en effet TF-IDF donne la priorité (attribut un poids élevé) aux termes qui apparaissent le plus souvent dans le document visé, et qui sont peu fréquent dans tout le corpus en général, cela facilite donc le choix du(ou des) document(s) pertinent(s) à renvoyer.

1.2.7 Méthodes de recherche

Maintenant que nous avons notre modèle d'indexation des documents de notre collection, nous devons ensuite définir les méthodes d'interrogation de cette dernière, plusieurs modèles ont vu le jour à travers le développement de la RI, dans ce TP nous avons choisis de nous intéresser à 3 d'entre eux :

1.2.7.1 Modèle booléen

Le modèle booléen est un est tout premier modèle à avoir vu le jour, et est encore à ce jour très utilisé, son principe est simple, ayant :

- une collection de documents D tel que :

$$D = \{D_1, D_2, ..., D_n\}$$

- une collection de termes T tel que :

$$T = \{T_1, T_2, ..., T_m\}$$

- un index de terme/document I tel que :

$$T[t_i, d_j] = \begin{cases} 1, & \text{si } t_j \in d_i \\ 0, & \text{sinon} \end{cases}$$

- une requête en forme normale conjonctive : (une conjonction de clauses)

$$Q = (W_1 \vee W_2 \vee ...) \wedge ... \wedge (W_i \vee W_{i+1}...) \tag{1.1}$$

Le but du modèle est, ayant reçu une requête Q, lo modèle va parcourir chaque termes dans une clause et retourner l'ensemble S_i des unions des sous ensembles $S_{i,j}$des documents qui contiennent le terme W en question, ce processus est répété autant de fois qu'il y a de clauses dans Q. À la fin de ce parcours un ensemble S final sera construit à partir de l'intersection des ensembles S_i construits auparavant. L'algorithme suivant résume ce traitement :

Algorithme 1 : Évaluation modèle booléen

Entrée : (Q : requête en FNC , I index terme/document , D : l'ensemble des documents , T l'ensemble des termes)
Sortie : (S : Ensemble des documents pertinents)
début
  $S \leftarrow \{\}$ **pour chaque** ($c_i \in$`Clauses(`Q`)`) **faire**
    $S_i \leftarrow \{\}$ **pour chaque** $t_j \in c_i$ **faire**
      $S_i \leftarrow S_i \bigcup \{d \in D, I[w, d] \neq 0 | w \in T\}$
    fin
    $S \leftarrow S \bigcap S_i$
  fin
fin
retourner S

1.2.7.2 Modèle Vectoriel

Le modèle vectoriel se veut plus détaillé que son prédécesseur, en effet l'idée est de modéliser chaque document d_i de la collection D comme étant un vecteur v_{d_i},dans l'espace vectoriel EV dont les vecteurs :

$$B = \{W_1 = (1, 0, ..., 0), W_2 = (0, 1, ..., 0), ..., W_n = (0, 0, ..., 1)\}$$

ont forment la **base**

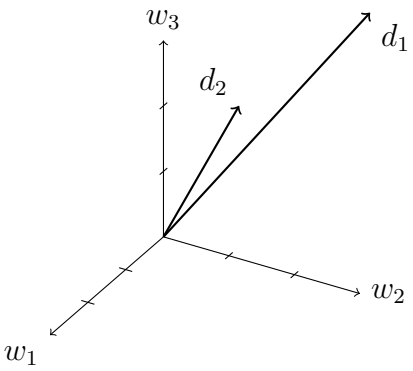
Autrement dit, considérant un document d_i contenant les termes $[w_1, w_2, w_3]$ et l'ensemble des termes $T = \{w_1...w_n\}$, ce dit vecteur d_i sera représenté comme combinaison linéaire de B (la base de EV) tel que :

$$\vec{d_i} = \alpha_1 * \vec{W_1} + \alpha_2 * \vec{W_2} + ... + \alpha_n * \vec{W_n}$$

Tel que :

$$\alpha_i = \begin{cases} poids[w_i, d_i], & \text{si } w_i \in d_i \\ 0, & \text{sinon} \end{cases}$$

Exemple :



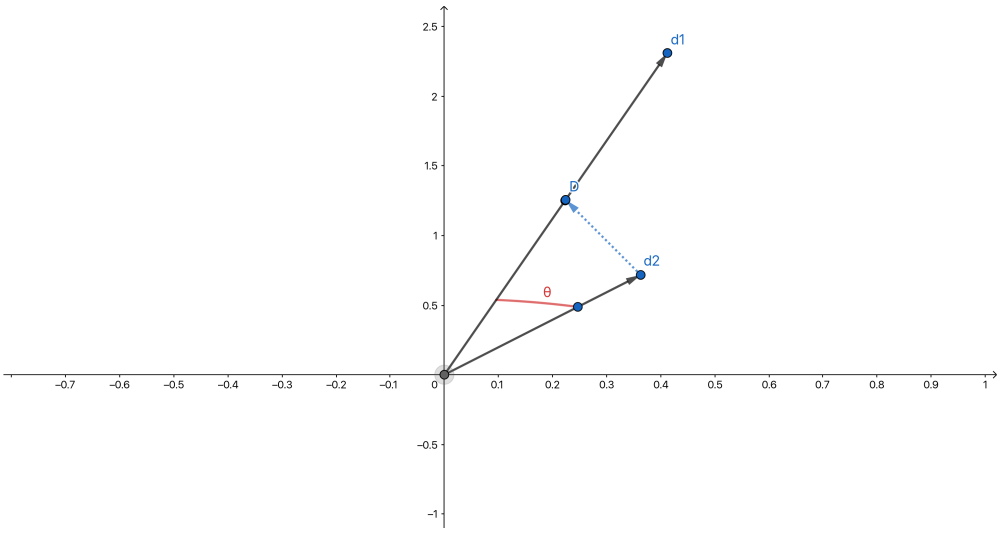
Évaluation d'un requête Q Tout comme un document d, une requête q est une combinaison linéaire de B, la requête est donc un vecteur dans l'espace vectoriel EV, le modèle procède au calcul d'une fonction de similarité Sim entre le vecteur $\vec{q}$ et l'ensemble des vecteur $\{\vec{d_1}, \vec{d_2}, ..., \vec{d_n}\}$ et de sélectionner les **k** vecteurs document qui maximisent Sim, plus formellement , ayant :

- D l'ensemble des vecteurs documents
- q le vecteur requête
- r le vecteur réponse

$$\vec{r} = k\_argmax_{d_i}(Sim(\vec{q}, \vec{d_i}))$$

Pour calculer la similarité entre deux vecteurs $\vec{v_1}$ et $\vec{v_2}$, nous avons testé quatre méthodes :

Produit interne : Du point de vue théorique, le produit interne(qui est une généralisation du produit scalaire) donne une quantification du degré de similarité entre deux vecteurs(en terme de magnitude(force,intensité..) et de directions(opposés,quasi similaires ...), ainsi cette mesure tend à favoriser deux vecteurs qui **pointent vers la même direction**. Cette vision est très abstraite, mais s'applique très bien dans le cadre de la représentation par modèle vectoriel, en effet si nous disposons de deux vecteurs documents $\vec{d_1}$ et $\vec{d_2}$ comme représentés dans le plan orthonormé suivant :



Le produit scalaire grandit quand le nombre de composantes similaires augmente aussi, ainsi pour une requête $\vec{q} = \sum_{i=1}^{n} \alpha_i * w_i$ et un ensemble de documents $D = \{\vec{d_i}, ..., \vec{d_n}\}$, la similarité Sim entre q et un document d_i est définie comme le produit scalaire :

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

Plus cette quantité est grande plus, un document d_i est pertinent par rapport à la requête q.

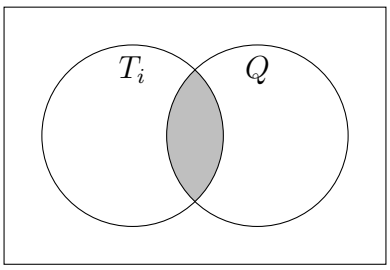
Similarité cosinus : Très liée à la similarité par produit interne, cette mesure essaye de normaliser le degré de similarité entre deux vecteurs(document avec document ou document requête), cela en calculant le cosinus de l'angle entre ces deux vecteurs, par définition si deux vecteurs $\vec{v_1}$ et $\vec{v_2}$ sont perpendiculaires, alors $cos(\overrightarrow{v_1, v_2}) = 0$, ainsi deux documents seront non similaires si leurs vecteurs associés ne points pas vers la même direction, par contraste, deux documents sont totalement similaires si l'un des leurs vecteurs représentant est un un scalaire de l'autre, c.à.d : $\vec{v_1} = \alpha * \vec{v_2}$, ainsi : $cos(\overrightarrow{v_1, v_2}) = 1$, de ce fait la mesure de similarité est une quantité qui prend ses valeurs dans l'intervalle $[0, 1]$.

Pour implémenter son calcul, nous pouvons utiliser la formule du produit scalaire

$$\vec{v_1}.\vec{v_2} = \| \vec{v_1} \| * \| \vec{v_2} \| * cos(\overrightarrow{v_1, v_2})$$

$$cos(\overrightarrow{v_1, v_2}) = \frac{\sum_{i=1}^{n} v_{1,i} * v_{2,i}}{\sqrt{\sum_{i=1}^{n} v_{1,i}^2} * \sqrt{\sum_{i=1}^{n} v_{2,i}^2}}$$

Coefficient de Jaccard : Nous pouvons aussi tacler ce calcul de similarité d'un point de vue ensembliste, en utilisant diverses mesures comme l'indice de **Jaccard**, cette mesure essaye de dénombrer les éléments communs entre deux ensembles, en effet soit l'ensemble des documents des termes T_i apparaissant dans le document d_i, et Q l'ensemble des termes apparaissant dans la requête q(nous verrons comment cette interprétation peut être traduite en opération linéaire incluant des vecteurs), et soit le schéma suivant :



La zone grisée(l'intersection) représente l'information pertinente, soit les documents qui sont à la fois dans T_i et dans la requête, pour quantifier cette amas d'information, l'indexe de **Jaccard** calcule le quotient

$$Sim(Q, T_i) = \frac{|T_i \bigcap Q|}{|T_i \bigcup Q|} = \frac{|T_i \bigcap Q|}{|T_i| + |Q| - |T_i \bigcap Q|}$$

en terme de vecteurs, la cardinalité de l'intersection $|T_i \bigcap Q|$ est remplacée par le produit scalaire $\vec{d_i}.\vec{q}$ et la cardinalité d'un ensemble $|T_i|$(resp. $|Q|$) par le carré de sa magnitude $|\vec{d_i}|^2 = \sum_{j=1}^{n} d_{i,j}^2$ (resp. $|\vec{q}|^2 = \sum_{j=1}^{n} q_j^2$), la formule devient donc :

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

Coefficient de Dice : Très semblable à l'indexe de **Jaccard**, le coefficient de **Dice** essaye de calculer la même quantité, mais en changeant la formule, en effet au lieu de soustraire le cardinal de l'intersection de Q et T_i, il suffit de compter leurs éléments communs deux fois, la formule devient donc :

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

Dans sa version vectorielle :

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

1.2.7.3 Modèle Probabiliste

Ce modèle attaque la problématique d'un point de vu probabiliste, en effet,ayant a disposition un assez bon échantillonnage d'apprentissage, le modèle essaye d'estimer la probabilité qu'un document d_i soit pénitent par rapport à une requête q en , cette estimation se base sur le fait que cette probabilité ne dépend que de la requête et du document choisi(plus précisément sa représentation), plus formellement :

Soient :

- d un document de la collection
- R l'ensemble des document pertinents
- NR l'ensemble des document non-pertinents

Si

$$P(R|d) > P(NR|d)$$

Alors le document d est jugé pertinent, toute la difficulté sera donc de pouvoir calculer ses probabilités.

Hypothèse d'indépendance : Nous supposons que la pertinence d'un document ne dépend pas de celle des autres documents, ainsi comme réponse à la requête q seront renvoyés les documents triés selon leur degré de pertinence, à savoir : $P(R|d)/P(NR|d)$ (voir cours pour la preuve).

Pour estimer les paramètres du modèle, nous avons choisis la modélisation **BIR** (**B**inary **R**etrieval **M**odel), nous supposons que le document d est un ensemble d'événements(présence ou non d'un terme t_i dans d), autrement dit :

d un document de la collection sous sa forme vectorielle binaire $\vec{d} = (t_1, ..., t_n)$ avec

$$d_k = \begin{cases} 1, & \text{si } t_k \in d \\ 0, & \text{sinon} \end{cases}$$

Nous supposons qu'il y a une indépendance entre les apparitions des termes à travers la collection.

Soient les quantité suivantes :

Document	**Pertinent**	**Non-Pertinent**	**Total**
$t_i = 1$	r	$n - r$	n
$t_i = 1$	$R - r$	$N - n - R + r$	$N - n$
Total	R	$N - R$	N

Avec :

- r : Nombre de documents pertinents contenant t_i
- n : Nombre de documents contenant t_i
- R : Nombre total de documents pertinents
- N : Nombre de documents dans la collection

Nous cherchons à calculer le score d'un document d_i par rapport à la requête q, nous utiliserons la formule (extraite du théorème de Bayes) suivante :

$$\begin{gathered} Sim(q, d_i) = \sum \log \frac{p_i * (1 - q_i)}{q_i * (1 - p_i)} \\ \text{En prenant :} \\ p_i = \frac{r}{R} \text{ et } q_i = \frac{n - r}{N - R} \\ \text{Nous obtenons :} \\ Sim(q, d_i) = \sum \log \frac{\frac{r+0.5}{R-r+0.5}}{\frac{n-r+0.5}{N-n-R+r+0.5}} \end{gathered} \tag{1.2}$$

Dans le cas où les documents sont pondérés :

$$Sim(\vec{q}, \vec{d_j}) = \sum_{t_i \in Q} W_{ij} * qtf_i * \log \frac{p_i * (1 - q_i)}{q_i * (1 - p_i)}$$

1.3 Conclusion

Après avoir vu le coté théorique de la RI, nous allons maintenant passer à l'implémentation des différentes méthodes.

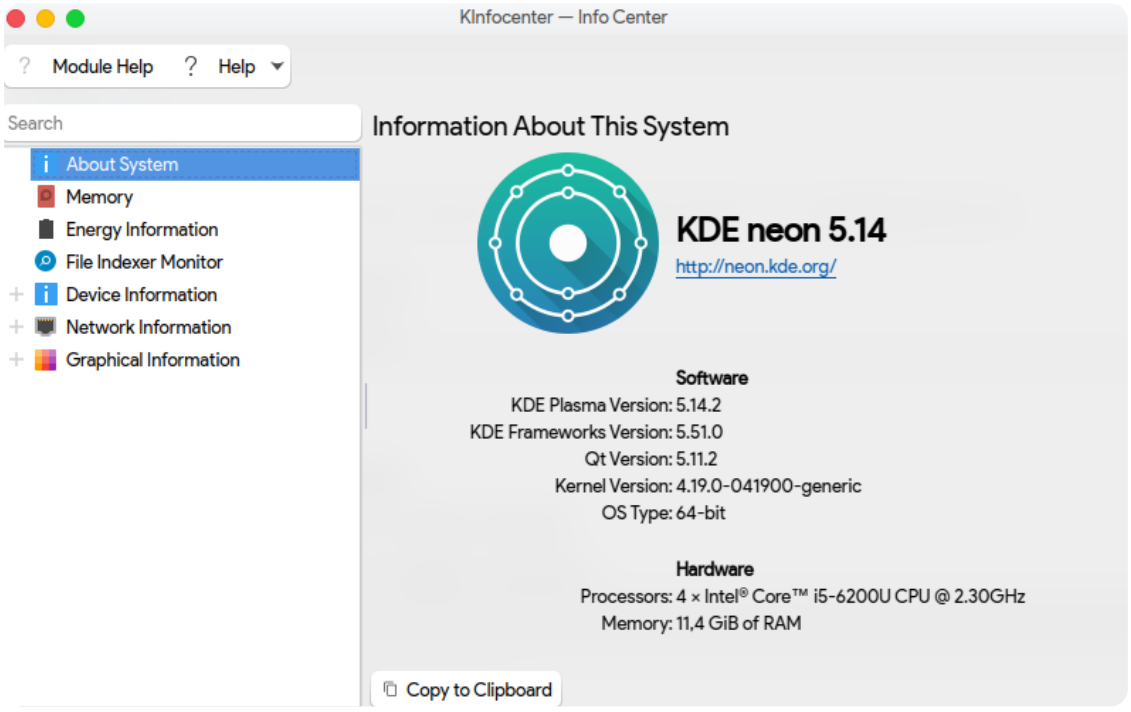
CHAPITRE 2

SOLUTION PROPOSÉES ET IMPLÉMENTATION

2.1 Outils utilisés

2.1.1 Environnement de travail

Les machines utilisées sont les suivantes :



2.1.2 Langage : Python

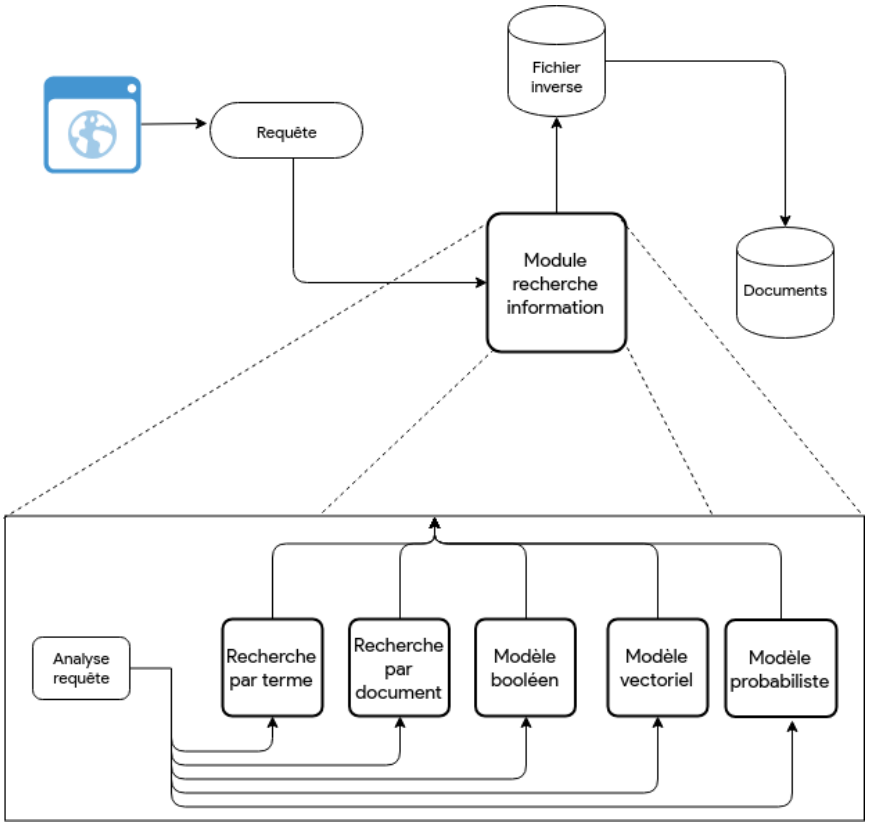
Nous allons principalement utiliser le langage de programmation interprété **Python**, sa flexibilité et la grande quantité de librairies(packages) disponibles en font un excellent outil pour l'implémentation des méthodes cité dans le chapitre précèdent.

2.1.3 PyQt framework

Le framework **Qt** sous C++ est une immense bibliothèque d'outils graphiques qui s'avère être doté d'un portage sur **Python**, **PyQt** sera utilisé pour la mise en place d'une interface graphique pour faciliter l'utilisation de notre application.

2.2 Schéma global du système

Le schémas récapitulatif suivant résume les principales composantes du système développé



Il s'agit maintenant de décrire les différents composants du système, nous commencerons par la construction de l'indexe(fichier inverse) à partir des documents, puis l'analyse de la requête, enfin nous verrons l'implémentation de chaque modèle.

2.3 Construction de l'indexe

La construction de l'indexe est très simple, nous faisons un parcours linéaire sur tout les documents pour construire la table d'indexation **Dico[d,t]**, chaque entrée est la fréquence d'apparition du terme dans le document s'il existe.

Le bout de code suivant montre cette partie :

```python
def generateReversedFile(path="TPRI/D", N=4):
    k = 1
    freq = {}  # dict vide

    while k <= N:
        f = open(path + str(k) + '.txt', 'r')
        t = f.read()
        t = t.lower()
        i = 0
        while i < len(t):
            if t[i] in listcar:
                t = t.replace(t[i], " ")
            i += 1
        a = t.split()
        for w in a:
            if w not in stoplist and len(w) > 1:
                if (w, k) not in freq:
                    freq[w, k] = a.count(w)
                    # print(w, freq[w, k])
        k += 1
        f.close()
    return freq
```

Nous aurons plus tard besoin de calculer le poids d'un terme dans un document, la fonction suivante permet cela :

```python
def getWeights(freq, N):
    ni = getNi(freq)
    maxes = maxFreq(freq, N)
    poids = {}
    for (w, d) in freq:
        poids[w, d] = (float(freq[w, d]) /
        float(maxes[d])) *
        log10(float(N) / float(ni[w]) + 1)
    return poids
```

2.4 Analyse de la requête

Nous avons tout et en tout 3 types de requêtes :

- **Requête de recherche par terme simple** : constituée d'un mot clé simple pour la recherche à travers les documents
- **Requête de recherche par document** : numéro du document, affiche les termes qui sont dans le document avec leurs poids
- **Requête de recherche par mot clés multiple** : permet de rechercher un ensemble de mot clé dans un ensemble de documents(Modèle booléen,vectoriel et probabiliste)

2.5 Implémentation et évaluation du modèle booléen

La fonction de similarité définie dans le chapitre 1 a été implémenté comme le montre la capture suivante :

```python
def getDocScores(freq,query,N):
    # freq = bm.generateReversedFile(path,N)


    tab = query.split()
    stopWords = ['and', 'or', '(', ')', 'not']

    docList = []
    for d in range(1, N+1):
        indexDoc = bm.indexdoc(freq, d)
        newQuery = ""
        for w in tab:
            toAdd = w
            if not (w in stopWords):
                if w in indexDoc.keys() and indexDoc[w] > 0:
                    toAdd = "1"
                else:
                    toAdd = "0"
            newQuery += toAdd + " "
        try:
            docList.append(eval(newQuery))
        except Exception:
            print("wrong query format")
            break
    return docList
```

2.6 Implémentation et évaluation du modèle vectoriel

2.6.1 Score d'un document

:

Par raison de simplicité, nous avons utilisé des **comprehension lists** pour mieux traduire l'aspect mathématique des formules d'origine, voici le code implémentant le calcul de ces mesure :

```python
def getDocScores(freq,query,
                 N,computeFunction=scoreInnerProduct):
    # freq = bm.generateReversedFile(path,N)
    weights = bm.getWeights(freq,N)
    fquery = bm.generateFreqOfQuery(query)
    words = set([w for (w,d) in freq])
    docList = []
    for d in range(1, N+1):
        weightd = bm.indexdoc(weights,d)

        score = computeFunction(weightd,fquery,words)
        docList.append(score)
    return docList
```

FIGURE 2.1 – Score d'un document

```python
def scoreInnerProduct(freq,fquery,words):
    return sum([f(fquery,w)*f(freq,w) for w in fquery])

def scoreCoefDice(freq,fquery,words):
    up = 2*scoreInnerProduct(freq,fquery,words)
    print(freq)
    # words = set([w for w in freq]+[w for w in fquery])
    down = sum([f(fquery,w)*f(fquery,w)+f(freq,w)*f(freq,w) for w in words])
    print("total ",down)
    return up/down

def scoreCosin(freq,fquery,words):
    up = scoreInnerProduct(freq,fquery,words)
    # words = set([w for w in freq] + [w for w in fquery])
    s1 = sum([f(fquery,w)*f(fquery,w) for w in words])
    s2 = sum([f(freq,w)*f(freq,w) for w in words])
    down = math.sqrt(s1*s2)
    return up/down

def scoreJaccard(freq,fquery,words):
    up = scoreInnerProduct(freq,fquery,words)
    # words = set([w for w in freq] + [w for w in fquery])
    down = sum([f(fquery,w)*f(fquery,w)+f(freq,w)*f(freq,w) for w in words]) - up
    return up / down
```

FIGURE 2.2 – Les quatres mesures de similarité

2.7 Implémentation et évaluation du modèle probabiliste

Le modèle probabiliste se résumant au calcul d'une seule mesure de similarité, le code suivant contient son implémentation :

```python
def getDocScores(freq,query,
                 N,pertinent):
    # freq = bm.generateReversedFile(path,N)
    weights = bm.getWeights(freq,N)
    fquery = bm.generateFreqOfQuery(query)
    print(pertinent)
    docList = []
    R = len(pertinent)
    for d in range(1, N+1):
        weightd = bm.indexdoc(weights,d)
        print("for doc " + str(d))
        s = 0
        for w in fquery:
            ri = pertinentContainWord(freq,pertinent,w)
            ni = documentsContainWord(freq,w)
            s = s + (
                f(weightd,w)*math.log10(((ri + 0.5) / (R - ri + 0.5)) / ((ni - ri + 0.5) /
                                                                         (N - ni - R + ri + 0.5))))
            r = math.log10(((ri + 0.5) / (R - ri + 0.5)) / ((ni - ri + 0.5) /
                                                            (N - ni - R + ri + 0.5)))
            print("result for %d %d is %f"%(ri,ni,r))
        score = s
        docList.append(score)
    # print(docList)
    return docList
```

FIGURE 2.3 – Calcul probabilité

CHAPITRE 3

PRÉSENTATION DE L'APPLICATION

3.1 Schéma d'utilisation

Nous passon maintenant à la présentation de notre application pour l'exploitation des différents modules du système, l'application est divisée en 5 onglets, qui contiennent chacun une interface dédié à l'approche qui lui est dédiée, voici une capture d'écran annotée pour plus de clarté :

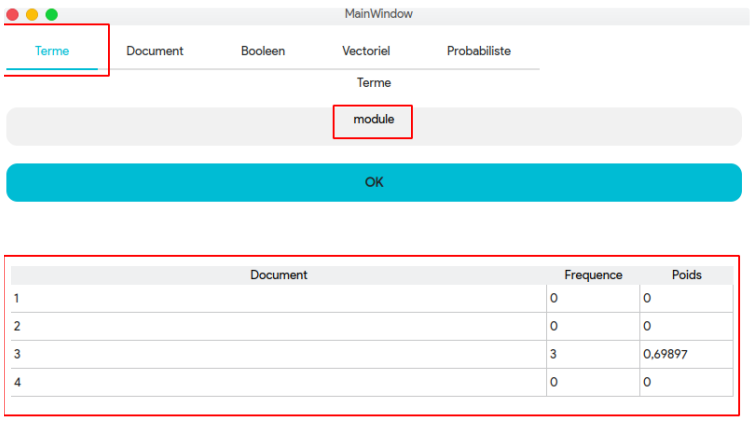


FIGURE 3.1 – Interface génerale

3.2 Exemples d'utilisation

3.2.1 Recherche dans document

Il suffit de donner le numero(nom) du document dans le champ dédié et de lancer la recherche, le résultat est une liste de termes avec leurs poids associés, la capture suivante montre un exemple :

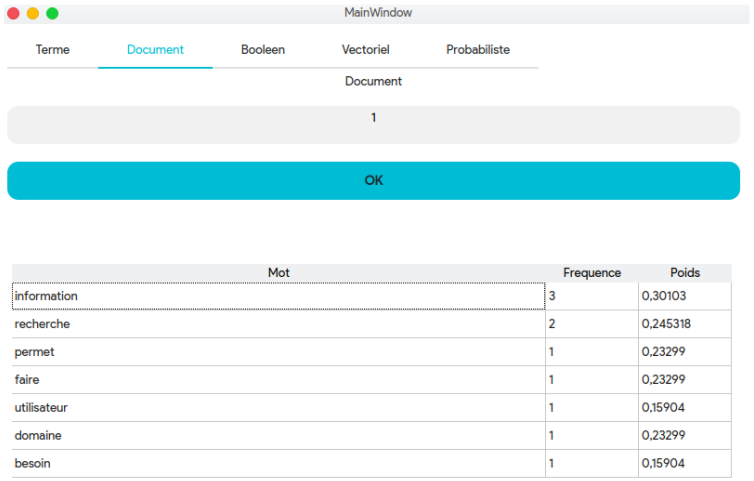


FIGURE 3.2 – Recherche dans un document

3.2.2 Recherche par modèle booléen

De manière semblable ç la recherche par terme, on introduit ici la requête dans le champ approprié, le résultat affiche les documents où le terme se trouve :

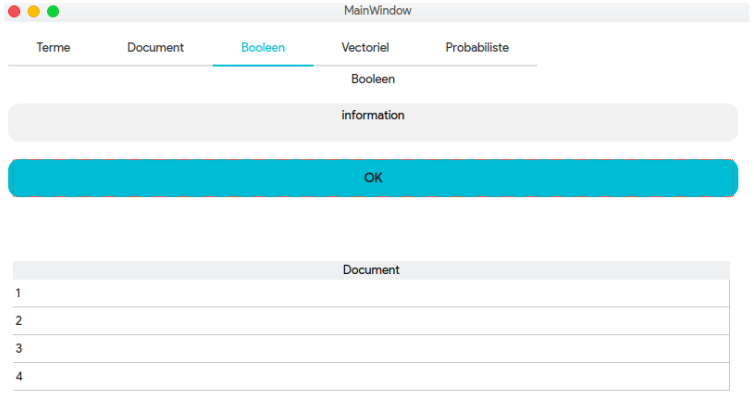


FIGURE 3.3 – Recherche booléenne

3.2.3 Recherche par modèle vectoriel

De la même façon, nous introduisant la liste de termes à rechercher dans leur champs, puis nous choisissons la mesure de similarité désirée, le résultat indique les documents résultants avec leurs scores :

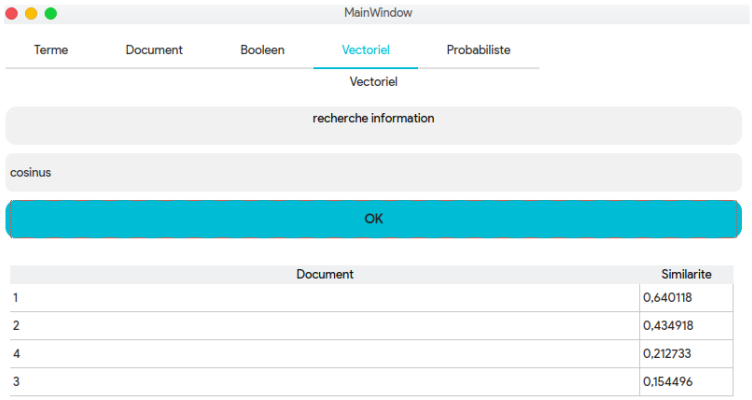


FIGURE 3.4 – Recherche booléenne avec similarité cosinus

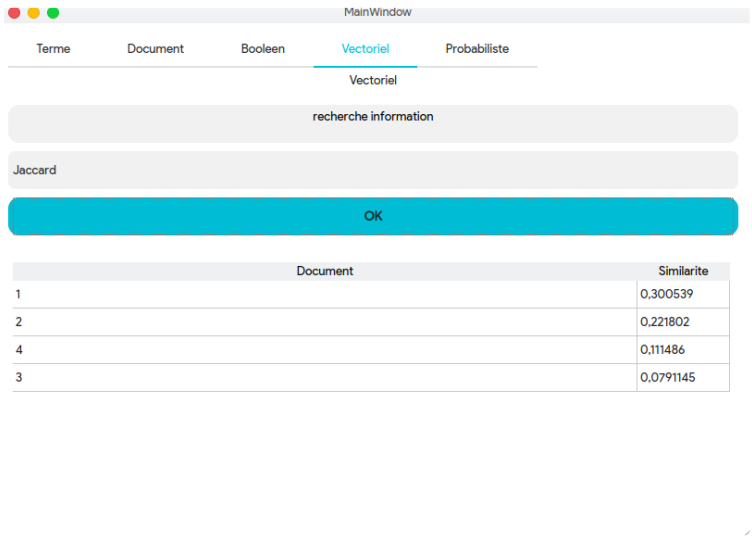


FIGURE 3.5 – Recherche booléenne avec similarité de Jaccard

3.2.4 Recherche par modèle probabiliste

De manière analogue, nous introduisons les termes à rechercher dans le champ de recherche, puis nous donnons la main à l'utilisateur de sélectionner les documents pertinents pour influencer la prochaine recherche, comme le montrent les deux captures suivantes :(une avant et l'autre après sélections des documents jugés **pertinents**) :

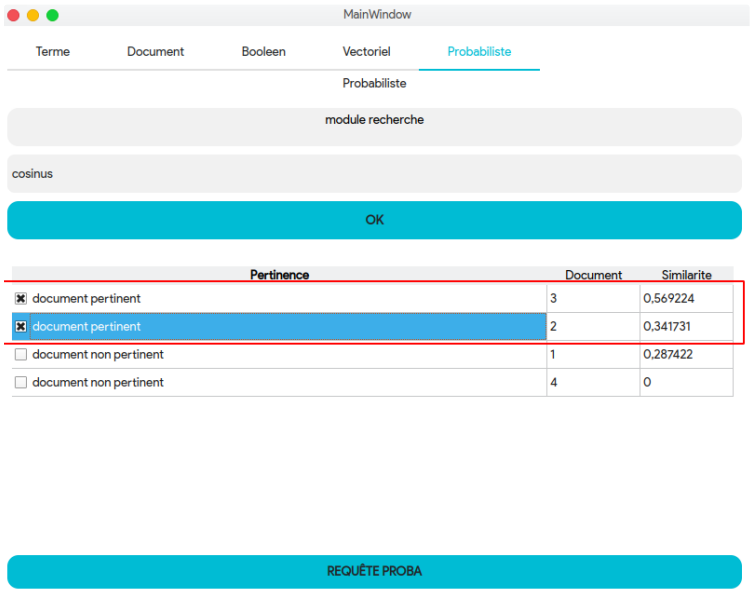


FIGURE 3.6 – Recherche probabiliste avec similarité par cosinus avant sélections

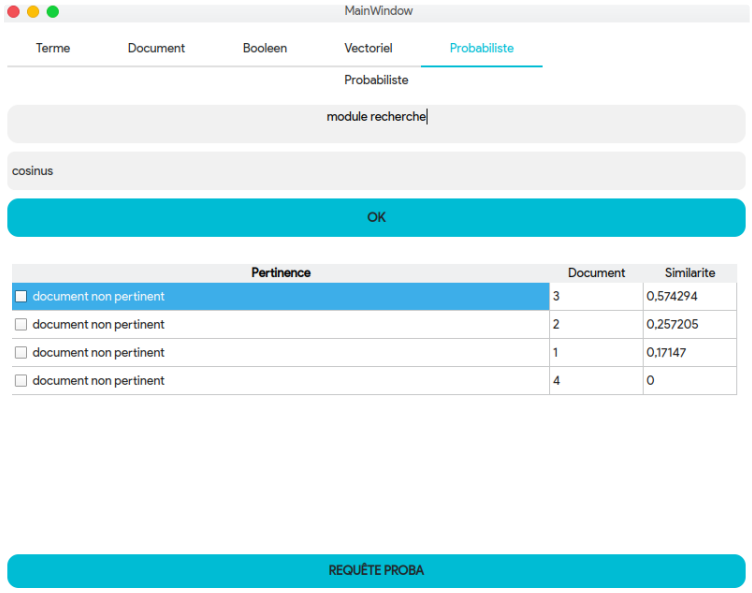


FIGURE 3.7 – Recherche probabiliste avec similarité par cosinus après sélections

CHAPITRE 4

CONCLUSION GÉNÉRALE

4.1 Comparaison des différentes méthodes

BIBLIOGRAPHIE