République Algérienne Démocratique et Populaire

Ministère de l’Enseignement Supérieur et de la Recherche Scientifique

**Université des Sciences et de la Technologie Houari Boumediene**

Département Informatique

# Module : Recherche d’information

**Rapport du Projet RI**

**Réalisation par :**

**-CHABANE Nouar 201400007379**

**-BEZGALI Meriem 201300005476**

**Groupe : 01**

1. **Introduction :**

La signification du terme « Recherche d’information » peut être défini comme le domaine qui étudie les méthodes permettant de trouver des informations pertinentes à partir d’un corpus constitué d’un ensemble de collection de documents volumineux et ceci en un lapse de temps jugé assez court.   
 La recherche d’information passe généralement par trois étapes qui sont :

* La représentation de la collection des documents (corpus),
* L’interrogation initiée par l’utilisateur (traitement des requêtes),
* La récupération des documents pertinents.

1. **Problématique :**

Dans ce TP, il est demandé d’implémenter les fonctionnalités suivantes :

* La création des 2 fichiers inverse et inverse pondéré pour le corpus fourni lors du TP,
* Implémentation des méthodes d’accès aux fichiers inverse et inverse pondéré,
* Implémentation des 3 modèles de recherche d’information vu lors des cours et TP et qui sont :
* Booléen
* Vectoriel (avec 4 fonctions de calcule de similarité)
* Probabiliste (avec la formule pondérée pour le calcul de similarité)
* Développer une IHM
* Effectuer une comparaison entre les 3 modèles implémentés

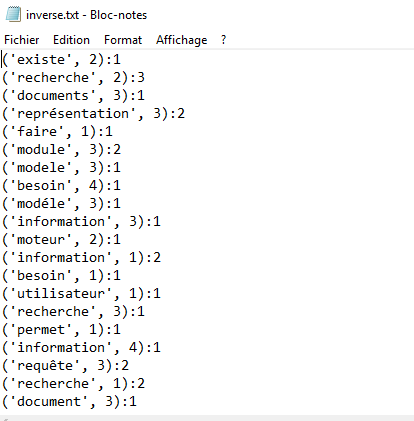
1. **Plan et objectif :**

Dans ce qui suit nous allons répondre aux points présents dans la problématique et pour cela nous allons :

* Expliquer la phase d’indexation et son implémentation (structure des fichiers inverse et inverse pondéré et les algorithmes utilisée pour indexer le corpus),
* Expliquer les algorithmes implémentés pour les méthodes d’accès aux fichiers inverse et inverse pondéré,
* Expliquer les algorithmes implémentés pour les 3 modèles,
* Voir le fonctionnement de l’IHM développé sur un exemple,
* Et enfin, analyser les résultats et comparé les 3 modèles.

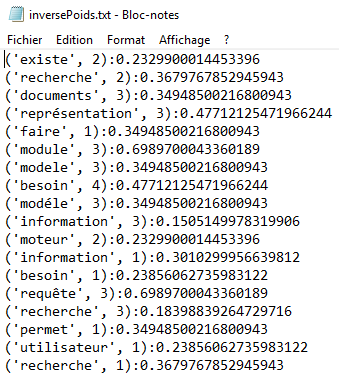
1. **Structure des fichiers :**
2. Structure des fichiers
3. Fichier inverse :

Le fichier inversé des fréquences contient des tuples de la forme, **(Mot,Document) : Frequence**, il est considéré comme un dictionnaire qui indexe chaque mot et cite sa fréquence pout tous document de la collection.

  
**figure1**.Exemple d’indexation du corpus

1. Fichier inverse de poids :

Le fichier inverse de poids contient des tuples de la forme, **(Mot,Document) : Poids**, ainsi chaque mot de la collection est lui y’est attribué un poids, même si c’est le même mot il peut avoir des poids différent selon le document dans le quelle il existe et donc le poids du mot est vu par rapport dans quel document il se trouve.

  
**figure2**.Exemple d’un fichier inverse pondéré du corpus

1. **Algorithmes implémentés :**

Dans cette partie nous allons voir les algorithmes utilisés pour indexer le corpus, accéder aux fichier inverse et inverse pondéré, rechercher dans quel document un terme existe-t-il, les termes d’un document donné, et les algorithmes des modèles booléen, vectoriel et probabiliste.

1. Algorithme d’indexation du corpus et création des fichier inverse/inverse poids :

Fichier inverse : Pour construire le fichier inverse on procède comme suit :

* Extraction des mots du corpus.
* Eliminer les mots vides en utilisant un document contenu tout les mots vide de la langue française fourni lors du TP.
* Enlever les signes de ponctuation et normalisation des mots, c.-à-d. rendre tous les mots en format minuscule.
* Construction d’un index ayant la structure

onstruction

**Procedure Fichier inverse :**

**Variables en entrée :**

**Mots\_vides** (liste des mots vides FR récupéré d’un fichier)

**Signes de ponctuation**(contenues dans une liste)

**Documents du corpus**

**Début :**

Pour chaque ***Document*** du corpus :

* Spliter les mots du document dans une liste
* Pour chaque ***Mot*** dans ***ListeMotDoc*** :

Si le mot n’est pas dans la liste **Mots\_vides** :

Si [Mot, Document] n’a pas été déjà calculé :

* + - * Calculer la ***fréquence*** de **Mot** dans **document**
      * Ajouter [***Mot***, ***Document***] ***:*** ***fréquence*** à une **liste**
* Créer un nouveau fichier nommé **FichierInverse** et écrire le contenu de **liste** dans ce fichier

**Fin**

Fichier inverse poids : il s’agit de faire le même travail que pour l’index de fréquence, sauf qu’il faut calculer le poids du mot au lieu de sa fréquence. Le poids d’un mot est calculé grâce à la formule de pondération comme suit :

*Avec :*

1. Fonction de mise à jour des 2 fichier inverse :

Nous avons créé une procédure pour mettre à jour les 2 fichiers inverse si l’utilisateur souhaite d’ajouter d’autre document à la collection, la fonction est implémentée comme suit :

Onstr

**Procedure Mettre\_à\_jour :**

**Début :**

-Si clique sur bouton **Mettre à jour :**

Alors appeler les deux fonctions **‘’Fichier inverse ‘’** et **‘’Fichier inverse poids ‘’**

**Fin**

1. Procédure d’accès :

* Accès fichier inverse et inverse poids :

Nous avons défini une procédure qui permet d’ouvrir les 2 fichiers inverses comme suit :

Onstruction

**Procedure Accés :**

**Variables en entrée :**

**Path**: Chemin du répertoire des 2 fichiers inverse

**Début :**

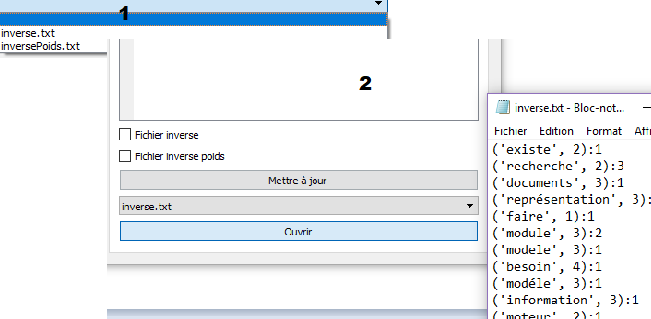
-Remplir un **comboBox** avec 2 options :

**-Fichier inverse.txt**

**-Fichier inverse Poids.txt**

-Si Clique sur bouton **Ouvrir** alors :

* + - Récupéré choix de l’option voulu de **comboBox** et la concaténé à **Path**
    - Affichage du fichier voulu dans l’interface

**figure3**.Ouverture d’un fichier inverse à partir de l’IHM

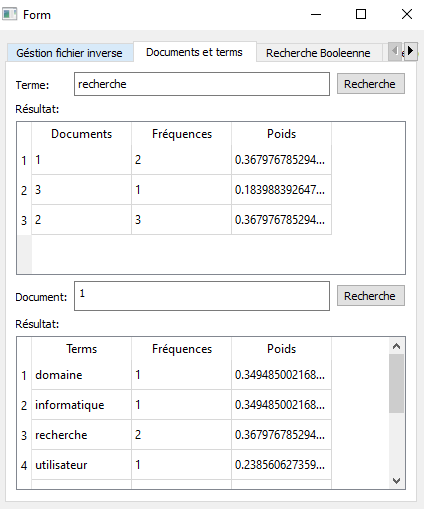
Afin de connaitre quels sont les mots appartenant à un document ainsi que leurs fréquences et leurs poids, et les documents comportant un mot donné, nous avons implémenter deux procédures :

* Recherche par document :

Prend en entrée le numéro du document et affiche dans une tableau la liste de ses mots ainsi que leurs fréquences et leurs poids.

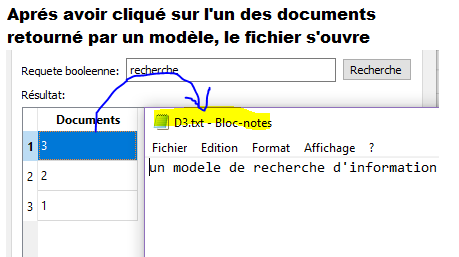
* Recherche par mot :

Prend en entrée un mot et affiche les documents dans lesquels il existe, ainsi que son poids et sa fréquence dans chacun de ces documents.

  
**figure4**.Onglet dans le quel les 2 fonctions précédentes sont implémentés

1. Procédure d’accès à un document :

Nous avons défini une procédure assez simple, qui permet d’accéder à un fichier de la collection si l’utilisateur le souhaite, ou par exemple s’il utilise l’un des modèles il peut ouvrir les documents retournés.

  
**figure5**.Ouverture d’un fichier de la collection

1. Les algorithmes des 3 modèles implémentés :

* Modèle booléen :

Dans l’histoire de la RI, le modèle booléen fut le premier proposé. C’est une méthode ensembliste utilisant l’algèbre de Boole permettant de fouiller de grands corpus et de retourner les documents qui contiennent un ou plusieurs termes de la requête. Ainsi un document est soit pertinent soit non pertinent par rapport à une requête donnée, écrite sous forme d’expression logique, utilisant les trois opérateurs : AND, OR, et NOT. -Définition prise du cours RI-

Dans cette partie du projet il s’agit d’évaluer des requêtes booléennes entré par l’utilisateur et afficher la liste des documents jugés pertinents par le modèle booléen.

**Procedure Modèle\_Booleen :**

**Variables :**

***Entrée : requête***

**Début :**

* Spliter les mots de la requête dans une liste
* Pour chaque **mot** de requête :
* On vérifie s’il n’est pas un opérateur (***And, Or*** et ***Not***)
* Si c’est le cas alors on récupère les documents contenant ce **mot** en utilisant la fonction de **recherche par mot** dans une **liste**
* Pour chaque élément de **liste :**
* On appelle la fonction « eval(**mot splité avec les opérateurs**) » de python
* On récupère les numéros des documents ayant donné un résultat 1 avec la requête dans ***DocsPertinents***
* Affichage de ***DocsPertinents*** *dans l’interface*

**Fin**

* Modèle vectoriel :

« Le modèle vectoriel le plus simpliste possible peut être décrit comme suit. Pour chaque document dans un ensemble, on compte le nombre d’occurrences de chaque mot (fréquence d’occurrence du terme, notée « TF »). Ce décompte ou cette vérification des occurrences forme un vecteur pour chaque document et l’ensemble des vecteurs forme une matrice parfois appelée matrice d’occurrences. Au lieu de compter le nombre d’occurrences, on peut aussi simplement vérifier la présence du terme et attribuer la valeur 1 lorsque le terme est présent, et la valeur 0 quand le terme est absent. »- Définition tirée du site de l’université de téluq Québec*-*

Dans cette partie du projet nous avons utilisé les 4 mesures du modèle vectoriel afin de calculer la similarité pour une requête donnée et afficher par ordre décroissant les documents pertinents.

**Procedure Modele\_Vectoriel :**

**Variables :**

***Entrée : requête, Fonction de similarité choisit***

**Début :**

* On crée un **vecteur requête** de taille « n » qui le nombre de mot de la collection, dont toutes les valeurs sont initialisées à 0
* On crée un vecteur contenant les mots et leurs poids avec l’indice du document dans lesquels ils sont présent
* Pour chaque ***Mot*** dans requête :

Si ***Mot*** existe dans Collection :

* On remplace sa valeur dans le **vecteur requête** par 1 (au lieu de 0)

Pour chaque document on calcule la similarité comme suit :

**mesure=0**

− Si ***choixFonction = « Inner Product »*** :

On ajoute **mesure** (la similarité de ce document) à ***ListeRésultats***

* Sinon, si **choixFonction = « Mesure de Cosinus » :**

On ajoute **mesure** à ***ListeRésultats***

* Sinon, si choixFonction = « Mesure de Jacard »

***SommeXY=***

On ajoute **mesure** à ***ListeRésultats***

* Sinon, si choixFonction = « Coef. De Dice »

On ajoute **mesure** à ***ListeRésultats***

Après avoir calculer la similarité on trie la ***ListeRésultats*** par ordre décroissant et on l’affiche dans l’interface

**Fin**

* Modèle Probabiliste :

Le modèle probabiliste tente d’estimer la probabilité d’observer des événements liés au document et à la requête, et étant donné que la **RI** est un processus incertain et imprécis car on a :

* De l’imprécision dans l’expression des besoins
* Et de l’incertitude dans la représentation des informations

Donc la théorie de la probabilité semble adéquate pour quantifier (mesurer) cette incertitude et imprécision et par conséquent, ce modèle tente d’estimer la probabilité qu’un document donné soit pertinent pour une requête donnée. - Définition prise du cours RI -

Pour implémenter le modèle probabiliste, il nous faut un échantillon d’apprentissage. Dans notre cas, cet échantillon sera les documents jugés pertinents par l’utilisateur. Donc, dans ce TP, ce modèle probabiliste sera implémenté en deux étapes. La 1ère étape consiste à répondre à la requête utilisateur avec le modèle vectoriel déjà développé précédemment, puis l’utilisateur choisira parmi les documents trouvés ceux qui sont pertinents pour lui. La 2ème étape fait une deuxième recherche avec cette même requête en appliquant le modèle probabiliste, en utilisant l’ensemble des documents choisis par l’utilisateur comme un échantillon d’apprentissage. – explication prise de l’énoncé du projet RI –

Nous avons utilisé la formule pondérée du modèle probabiliste qui est définit comme suit :

*Avec*:

* ri: nombre de documents pertinents contenant ieme terme
* n: nombre de documents contenant le ieme terme
* R : nombre total de documents pertinents
* N : nombre de documents dans la collection
* i : allant de 1 jusqu'à T

**Procedure Modèle\_Probabiliste:**

**Variables en *Entrée : requête, document jugé pertinent sélectionné à partir du modèle vectoriel***

**Début :**

* Spliter les mots de la requête dans une liste
* Pour chaque **document** :

Similarité=0

* + Pour chaque **mot** de la requête :

-calculer :

**-**mettre dans un **Dictionnaire** l’indice du document comme clé et sa similarité comme valeur

-Trier le **Dictionnaire** par ordre décroissant des valeurs et l’affiché

**Fin**

1. **Outils utilisés et IHM :**

* Langages de programmation :

1. Python :

Nous avons utilisé le langage de programmation **Python**, qui est un langage de programmation objet interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme. Le langage Python est placé sous une licence libre et fonctionne sur la plupart des plates-formes informatiques et a un langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation aisée aux concepts de base de la programmation. -définition prise de Wikipedia-

1. PyQt :

PyQt est un module libre qui permet de lier le langage Python avec la bibliothèque Qt distribué sous deux licences : une commerciale et la GNU GPL. Il permet ainsi de créer des interfaces graphiques en Python. Une extension de QtDesigner ou QtCreator (utilitaire graphique de création d'interfaces Qt) permet de générer le code Python d'interfaces graphiques. -définition prise de Wikipedia-

* IDE et Tool :

1. Visual Studio Code

Nous avons utilisé **Visual Studio Code** qui est un [éditeur de code](https://fr.wikipedia.org/wiki/%C3%89diteur_de_texte) extensible développé par [Microsoft](https://fr.wikipedia.org/wiki/Microsoft) pour [Windows](https://fr.wikipedia.org/wiki/Windows), [Linux](https://fr.wikipedia.org/wiki/Linux) et [macOS](https://fr.wikipedia.org/wiki/MacOS" \o "MacOS). Le [code source](https://fr.wikipedia.org/wiki/Code_source) est fourni sous la [licence libre](https://fr.wikipedia.org/wiki/Licence_de_logiciel_libre) MIT (plus précisément la [licence Expat](https://fr.wikipedia.org/wiki/Licence_Expat)) sur le site du projet sur [Github](https://fr.wikipedia.org/wiki/Github" \o "Github). En revanche, l'exécutable est proposé sur le site officiel de Microsoft sous une [licence privatrice](https://fr.wikipedia.org/wiki/Logiciel_privateur). -définition prise de Wikipedia-

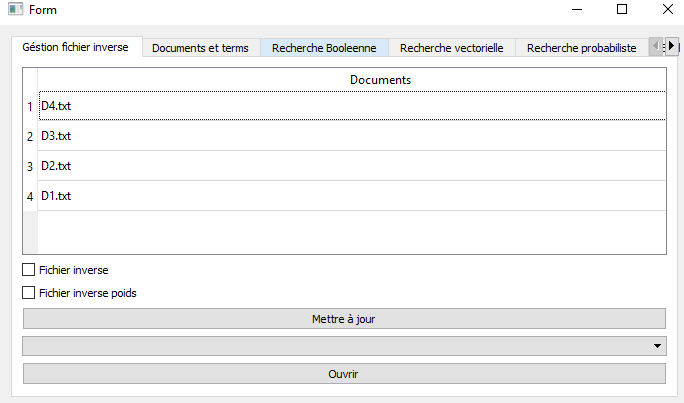
1. Qt Creatot

**Qt Creator** est un [environnement de développement intégré](https://fr.wikipedia.org/wiki/Environnement_de_d%C3%A9veloppement_int%C3%A9gr%C3%A9) [multiplate-forme](https://fr.wikipedia.org/wiki/Logiciel_multiplate-forme" \o "Logiciel multiplate-forme) faisant partie du [framework](https://fr.wikipedia.org/wiki/Framework" \o "Framework) [Qt](https://fr.wikipedia.org/wiki/Qt). Il est donc orienté pour la programmation en C++. Il intègre directement dans l'interface un débogueur, un outil de création d'interfaces graphiques, des outils pour la publication de code sur [Git](https://fr.wikipedia.org/wiki/Git) et [Mercurial](https://fr.wikipedia.org/wiki/Mercurial) ainsi que la documentation [Qt](https://fr.wikipedia.org/wiki/Qt). L'éditeur de texte intégré permet l'[autocomplétion](https://fr.wikipedia.org/wiki/Autocompl%C3%A9tion" \o "Autocomplétion) ainsi que la [coloration syntaxique](https://fr.wikipedia.org/wiki/Coloration_syntaxique). Qt Creator utilise sous [Linux](https://fr.wikipedia.org/wiki/Linux) le compilateur [gcc](https://fr.wikipedia.org/wiki/GNU_Compiler_Collection" \o "GNU Compiler Collection). Il peut utiliser [MinGW](https://fr.wikipedia.org/wiki/MinGW" \o "MinGW) ou le compilateur de [Visual Studio](https://fr.wikipedia.org/wiki/Visual_Studio) sous [Windows](https://fr.wikipedia.org/wiki/Windows). -définition prise de Wikipedia-

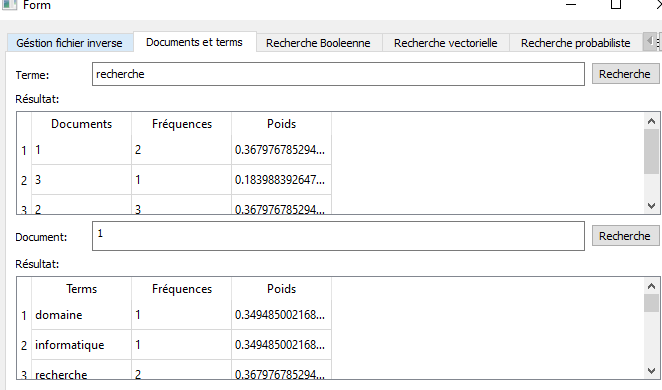
* Présentation de l’IHM

Nous avons implémenté une interface qui permet à l’utilisateur d’exploiter l’application d’une manière simple. Notre interface est constituée de 5 onglets, que nous allons présenter ci-dessous :

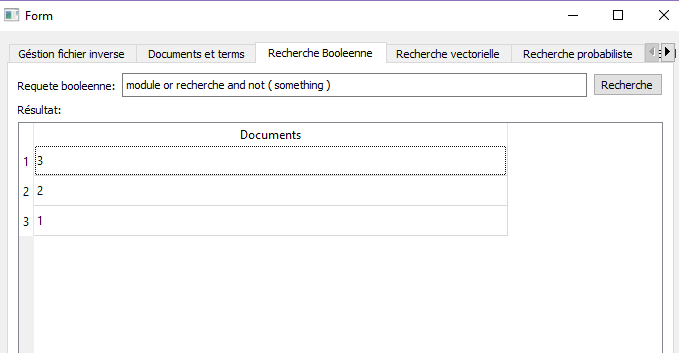
1er onglet : Nommée « Gestion fichier inverse », permet de mettre à jour les fichiers inverses et de les ouvrir, et contient aussi un tableau pour afficher les fichiers de la collection et les ouvrir en cliquant sur la cellule souhaitée du tableau pour l’ouvrir. (Ces 2 fonctions ont été déjà présenté avec un exemple dans la partie algorithmes implémentés)

**figure6**.Onglet gestion fichier

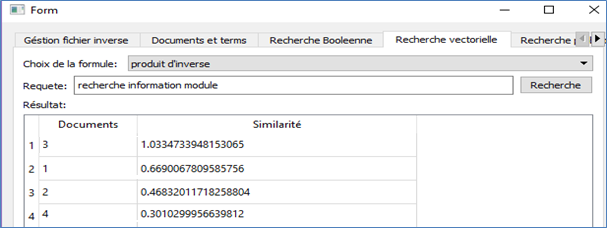
2eme onglé : Dans cet onglet l’utilisateur peut recherche soit un terme et il aura comme résultat les fichiers dans lequel il existe, soit afficher les mots d’un document, leurs fréquences et leurs poids, aussi il peut ouvrir un document en cliquant sur la cellule du tableau.

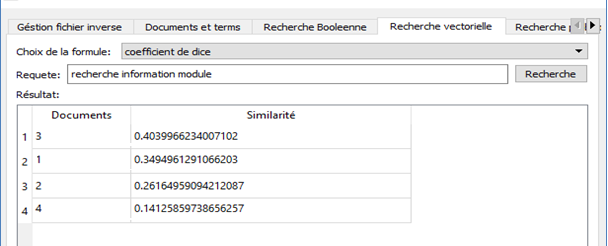
  
**figure7**.Onglet documents et terms

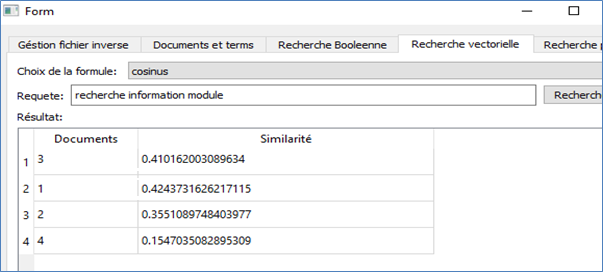
3eme onglet : cet onglet concerne le modèle booléen, il contient une entré pour taper la requête booléenne de l’utilisateur et un tableau pour afficher les documents retournés par la procédure modèle booléen. (Possibilité d’ouvrir les fichiers retournés en cliquant sur la cellule du tableau)

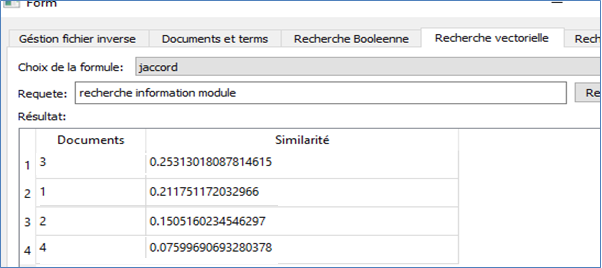
**figure8**.Onglet di modèle booléen

4eme onglet : cet onglet concerne le modèle vectoriel, il contient une entré pour taper la requête de l’utilisateur, un comboBox pour choisir la formule à appliquer, et un tableau pour afficher les documents retournés par la procédure modèle vectoriel. (Possibilité d’ouvrir les fichiers retournés en cliquant sur la cellule du tableau)  
« Pour ce qui suit nous allons rechercher la requête : module recherche information »

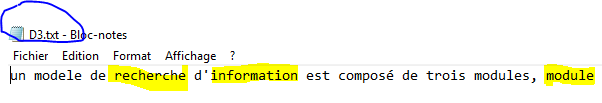
 **figure9**.Onglet du modèle vectoriel, cas formule produit interne

**figure10**.Onglet du modèle vectoriel, cas formule coefficient de dice

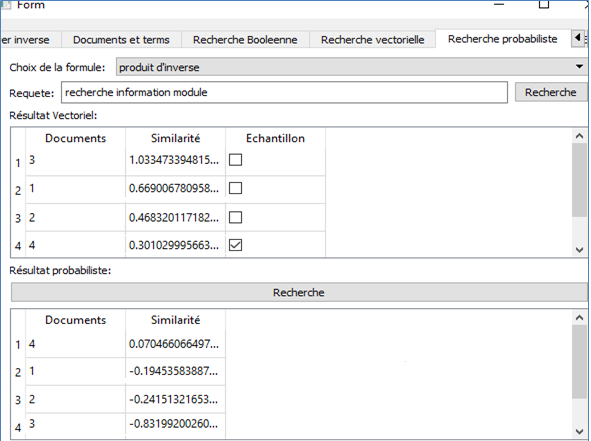
**figure11**.Onglet du modèle vectoriel, cas formule produit interne

**figure12**.Onglet du modèle vectoriel, cas formule jaccord

**Remarque :** On remarque qu’à chaque application d’une formule on obtient l’ordre de pertinence des documents comment suit : 3, 1, 2 et enfin 4, donc on peut s’assuré que les résultats ne sont pas aléatoires et sont parfaitement correcte, car effectivement le document 3 et le plus similaire par rapport à la requête formulé car il contient effectivement les 3 mots recherchés. (Recherche 1 fois, information 1fois, et module 2 fois)

**figure13**.Contenu du document 3 avec les mots de la requête (Le 2eme terme module et un peut plus loin dans le document)

5eme onglet : cet onglet contient les même composant que celui du vectoriel, en plus le tableau de cet onglet contient des checkBox pour que l’utilisateur choisisse les documents pertinents pour ensuite appeler le modèle vectoriel sur la même requête entrée précédemment.

**figure14**.Onglet du modèle probabiliste, cas formule produit interne

1. **Analyse et comparaison**

Le modèle joue un rôle central dans la RI. C'est lui qui détermine le comportement clé d'un SRI mais on générale, chaque modèle utilise une indexation des fichiers qui présente les avantages suivants :

Avantages :

* L’indexation des documents permet de réduire la complexité, les rend accessibles et plus faciles à manipuler.
* Le prétraitement des données comme l’utilisation d’un fichier pour les mots vides permet de ne garder que les mots reflétant un contenu informationnel. Ceci réduit la taille des indexes et rend la recherche rapide et précise.
* Les deux points précédents rendent le temps de réponse instantané pour les requêtes et l’évaluation

Amélioration et perspectives :

Dans ce projet nous avons utilisé 3 modèles classiques, qui sont le booléen, le vectoriel et le probabiliste, mais on peut améliorer les résultats d’un SRI comme suit :

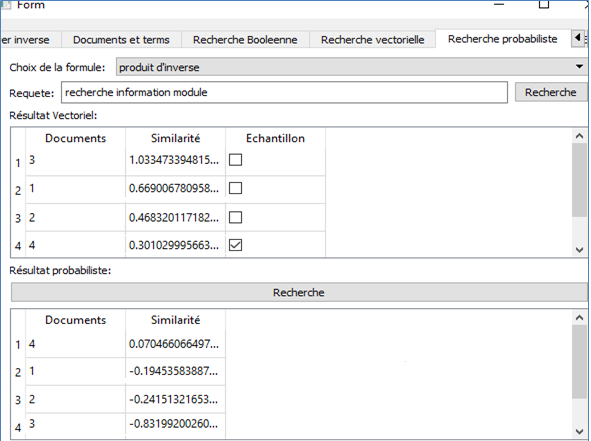
* Utilisation d’autres modèles plus avancés afin de répondent aux requêtes des utilisateurs
* On pourrait aussi utiliser d’autres méthodes de mesure pour le modèle vectoriel et probabiliste.
* Implémenté une procédure pour évaluer le taux de rappel et de précision des résultats, en ayant au préalable un corpus qui contient pour chaque requête le résultat attendu et par la suite rectifier certain résultat erroné à l’aide de ce corpus et d’une fonction d’évaluation des résultats.

Comparaison des modèles vectoriel et booléen :

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Booléen** | **Vectoriel** | **Probabiliste** |
| Documents | * Le modèle effectue une correspondance stricte entre le document et la requête. (Soit le document repend parfaitement à la requête et on le retourne sinon le document et totalement ignoré) * Aucune distinction entre les documents pertinents, pas comme les 2 autres modèles où nous avons une mesure de similarité. | * Les documents sont très distincts. * La correspondance est mesurée avec un certain taux de similarité. * Possibilité d’utilisation de plusieurs formules pour calculer la similarité. | * La correspondance est mesurée avec un certain taux de similarité. * Les documents sont très distincts. * Ce modèle a besoin d’un échantillon pour appliquer la formule de calcule de similarité. * Possibilité d’utilisation de plusieurs formules pour calculer la similarité. (Dans ce projet nous avons choisit la formule avec pondération) |
| Format de la requête | * Il faut avoir une idée fixe et claire de la requête. * Le format est complexe pour un simple utilisateur qui ne connait pas la syntaxe d’une requête booléenne. | * Le format est simple et l’utilisateur peut trouver des documents pertinents même si sa requête n’est pas très précise. * Pas besoin de connaitre une syntaxe pour formuler la requête. | * Le format est simple et l’utilisateur peut trouver des documents pertinents même si sa requête n’est pas très précise. * Pas besoin de connaitre une syntaxe pour formuler la requête. |

Analyse des résultats des mesures du modèle probabiliste et vectoriel :

Nous avons remarqué qu’après avois utilisé le modèle vectoriel et après le modèle probabiliste pour la même requête et en sélectionnant les documents les moins pertinent comme échantillon, ces derniers deviennent les plus pertinents dans le résultat du modèle probabiliste et donc ce modèle est jugé efficace si nous possédant au préalable un échantillon d’apprentissage.

  
**figure15**.Un screen de l’interface affirmant l’analyse entre modèle booléen et probabiliste

# Conclusion :

La recherche d’information présente des modèles assez intéressants qui ont pour fonction de permettre à l'utilisateur d'accéder à des documents qui contribuent à combler son besoin d'information, exprimé sous forme de requête, et le système peut être vu par l’utilisateur comme un instrument de prédiction de la pertinence des documents d’un corpus par rapport à sa requête.

Pour évaluer l’adéquation entre un ensemble de documents et une requête, le système doit posséder d’une part une représentation interne des documents disponibles et de la requête utilisateur et d’autre part de méthodes de comparaison. La représentation interne ainsi que les méthodes de mesure choisies influent directement sur le résultat de la recherche.

L’indexation des documents est une étape primordiale car elle détermine de quelle manière les connaissances contenues dans les documents fournis sont représentées. Elle a lieu à chaque ajout d’un document dans l’ensemble des documents étudiés. Aussi, le temps que prend la recherche dépend des structures utilisées. Par exemple, pour diminuer le temps de réponse, on pourrait garder les indices des mots présents dans le document ou bien la requête et leur affecter les pondérations correspondantes et ignorer les indices qui seront à zéro ce qui diminue le temps de parcours mais aussi le temps de réponse avec.

La recherche d’information est un pilier du domaine de l’intelligence artificielle. Il est donc nécessaire d’améliorer ses résultats en s’intéressant aux besoins non exprimés de l’utilisateur pour augmenter la pertinence utilisateur/ machine.

Dans ce projet nous avons eu la chance d’implémenter les 3 modèles vu en cours et ainsi de mieux les assimilés et de comparé leurs résultats, nous avons aussi appris à indexer un corpus donné, et enfin nous avons apprit a créé une IHM à l’aide du langage PyQt et de son tool QtDesigner.

* Annexe :

Le contenu du corpus :

Le corpus est composé de 4 documents, contenant le texte suivant :

* D1.TXT : la recherche d'information est un domaine qui permet de faire la recherche d'information pour un utilisateur selon son besoin en informatique.
* D2.TXT : il existe plusieurs modéles de recherche d'information, chaque moteur de recherche est developé sur la base d'un ou plusieurs modéles de recherche.
* D3.TXT : un modele de recherche d'information est composé de trois modules, module de représentation de documents, modéle de représentation de requête, module d'appariement entre document et requête
* D4.TXT : une reqûete est une représentation du besoin d'un utilisateur en information

Quelque screen du code :

