République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Mémoire de Fin d'Etudes
De MASTER ACADEMIQUE

Domaine : **Mathématiques et Informatique**
Filière : **Informatique**
Spécialité : **Systèmes Informatiques**

Thème

Nouvelle approche de lissage pour étendre le modèle de langue dans Lucene

Réalisé par :
AMAROUCHE Sylia
MOSTEFAI Sabrina

Mémoire soutenu publiquement le 22/10/ 2020 devant le jury composé de :

Président : *Mme R.AOUDJIT*
Examinateur : *M S.SADI*

Encadré par : *Mme F.AMIROUCHE*

Année universitaire : 2019/2020

Résumé

Notre travail s'inscrit dans le cadre de la Recherche d'Information (RI). Plus précisément, nous nous intéressons à un modèle de recherche, le modèle de langue, et à ses différentes approches de lissage. Notre objectif étant d'étendre l'API de recherche d'information LUCENE, en y implémentant diverses approches de lissage. Lucene étant une API libre écrite entièrement en JAVA, elle permet de créer un moteur d'indexation et de recherche de documents textuels. Notre contribution est double : d'abord, nous avons proposé une approche de lissage que nous avons intégré au modèle de langue de Lucene afin d'améliorer ses capacités de recherche. L'approche proposée, nous est basée sur la combinaison de deux approches de lissages déjà implémentés dans Lucene. Ensuite, nous avons implémenté l'approche de lissage Absolute Discount et l'avons intégrée au modèle de langue de Lucene. Nos approches proposées ont été évaluées en recherche d'information sur une collection de tests. Notre approche combinée a apporté de très bons résultats de recherche, dans certains cas, comparé aux autres approches implémentées dans Lucene.

Mots clés : recherche d'information, modèle de langue, lissage, Lucene.

Abstract

Our work focuses on Information Retrieval (IR), and more precisely we center our intention on a particular search model, which is the unigram language model and its different smoothings, that we used to extend Lucene. Our goal is to extend the Information Retrieval API named Lucene, by implementing various smoothing techniques. Lucene is a free API written entirely in JAVA, it allows you to create an indexing and search engine for textual files. Our contribution has two parts : first, we proposed a smoothing approach that we integrated into Lucene's language model in order to improve its search capacities. The suggested approach is based on a combination of algorithms already implemented in Lucene. Next, we have implemented the Absolute Discount smoothing approach and integrated it into Lucene's language model. Our proposed approaches have been evaluated in information retrieval on test collection. Our contribution yielded very good search results in some cases compared to other approaches implemented in Lucene.

Keywords : information retrieval, language model, smoothing, Lucene.

Remerciements

Après avoir rendu grâce à Dieu le Tout Puissant et le Miséricordieux, nous tenons à exprimer toute notre reconnaissance à notre directrice de mémoire, Madame AMIROUCHE Fatiha. Nous la remercions de nous avoir proposé et dirigé ce modeste travail et pour sa gentillesse et sa disponibilité.

Nous adressons nos sincères remerciements à tous les professeurs, intervenants et toutes les personnes qui par leurs paroles, leurs écrits, leurs conseils et leurs critiques ont guidé nos réflexions et nos recherches.

Nous remercions nos parents qui ont toujours été là pour nous et qui nous ont soutenu.

"Éduquer est un art, vous êtes les artistes de ma vie

Élever un enfant est un poème, vous êtes les plus grands des poètes"

Puis, un remerciement très chaleureux à nos frères, nos grands-parents, nos tantes, nos oncles, nos cousins, nos cousines, nos amis et amies.

Table des figures

Liste des tableaux

Table des matières

Introduction Générale

Contexte

L'information peut être considérée aujourd'hui comme un capital (immatériel) important tant pour les organisations que pour les individus eux-mêmes. Cette information permet de construire ou de valider une connaissance concernant différents objets de la vie quotidienne (concurrents, langages de programmation, sports...). Dans le même temps, on peut souligner que l'information se retrouve partout, dans les intranets, les extranets et particulièrement sur le Web qui n'en finit plus de grandir et qui constitue une source d'information privilégiée. Il n'est donc pas surprenant de constater que le nombre de requêtes sur les moteurs de recherche du Web augmente de façon significative.

Pour identifier les informations répondant à ses besoins, l'usager pratique ce que nous définissons comme une activité de recherche d'information. La recherche d'information (RI) est le domaine par excellence qui s'intéresse à répondre à ce type d'attente. En effet, la RI offre des outils, les systèmes de Recherche d'Information (SRI), pour stocker, organiser et retrouver au besoin, l'information pertinente, dans de grands volumes d'informations préalablement stockées.

La gestion et la recherche de ces vastes volumes d'informations peuvent être très difficiles, c'est pourquoi la création d'applications de recherche efficaces et à hautes performances est devenue une nécessité. Le modèle de recherche, qui constitue le cœur d'un SRI, est une préoccupation principale dans la recherche efficace et performante d'informations. Plusieurs modèles de recherche ont été ainsi proposés, dont l'un des plus performants est le modèle de langue objet de notre étude.

Problématique

Pendant des décennies, de nombreuses recherches se sont concentrées sur la RI. On peut dire que la communauté open source porte désormais les fruits de ce travail acharné car de nombreuses plateformes de gestion de données open source sont développées, parmi lesquelles la plateforme de RI, Apache Lucene, qui a gagné en popularité et est considérée par beaucoup comme le cadre de recherche de texte incontournable. Apache Lucene offre de puissantes fonctionnalités d'indexation et de recherche avec son moteur de recherche.

De nombreux modèles de recherche de la RI sont implémentés dans Lucene, parmi lesquels les modèles de langue avec lissage (deux différentes approches de lissage sont implémentées dans Lucene).

Notre projet d'études consiste à étendre le modèle de langue du moteur de recherche Lucene avec de nouvelles approches de lissage.

Structure du mémoire

Notre présent rapport est structuré en quatre chapitres comme suit :

Chapitre 1 : **Généralités sur la recherche d'information**. Dans ce chapitre nous présentons les concepts généraux liés à la recherche d'information, le processus de recherche, ainsi que les modèles de recherche. Enfin, ce chapitre aborde également l'évaluation des SRI.

Chapitre 2 : **Les Modèles de langue**. Dans ce chapitre nous présentons les modèles de langue et leur utilisation en RI.

Chapitre 3 : **Analyse et conception**. Dans ce chapitre, nous présentons la plateforme Apache Lucene, puis nous décrirons en détail nos différentes approches proposées pour l'extension de Lucene.

Chapitre 4 : **Mise en œuvre et évaluation**. Ce dernier chapitre présente les détails d'implémentation et de mise en œuvre de nos approches, ainsi que leur évaluation.

Nous terminons notre mémoire par une conclusion générale.

Chapitre 1

Généralités sur la recherche d'information

1.1 Introduction :

La Recherche d'Information (RI) peut être définie comme une activité dont la finalité est de retrouver et de retourner un ensemble de documents à un utilisateur en fonction de son besoin en informations. Le défi est de pouvoir, parmi le volume important de documents disponibles, trouver ceux qui correspondent au mieux à l'attente de l'utilisateur. Ainsi, l'objectif principal de la RI est de fournir des modèles, des techniques et des outils pour stocker et organiser des masses d'informations et localiser celles qui seraient pertinentes relativement à un besoin en information d'un utilisateur, souvent, exprimé à travers une requête. Ces outils sont appelés des Systèmes de Recherche d'Information (SRI). De manière générale, le fonctionnement d'un SRI consiste à construire une représentation des documents et de la requête et d'établir une comparaison entre ces deux représentations (documents/requête) pour retourner les documents pertinents. Cette comparaison se base sur modèle de recherche.

Ce chapitre a pour but d'introduire les concepts de base de la RI.

1.2 Définitions :

La Recherche d'Information (RI), a pour objectif de retrouver, dans une collection de documents préalablement traitée et stockée (sur un support informatique), l'ensemble des documents pertinents pour un besoin d'information d'un utilisateur, formellement exprimé par une requête. La RI est mise en œuvre à travers des Systèmes de Recherche d'Information (SRI).

De ce qui suit, nous allons définir les concepts de base de la RI, concepts que nous venons d'évoquer dans la précédente définition :

1.2.1 Document :

Le document constitue l'information élémentaire d'une collection de documents. Cela pourrait être un texte, une vidéo, une image, ... L'information élémentaire, appelée aussi granule de document, peut représenter tout ou une partie d'un document.

1.2.2 Collection de documents :

Aussi appelée fond documentaire ou corpus (recueil de documents), est un ensemble de documents préalablement stockés et organisés en vue de la recherche.

1.2.3 Requête :

La requête exprime le besoin en informations de l'utilisateur. C'est elle, qui initie le processus de recherche d'information. La requête doit contenir des éléments clés suffisamment

discriminants afin d'exprimer au mieux le besoin en informations de l'utilisateur. La requête est écrite dans un langage d'interrogation compréhensible par le SRI. Parmi les langages d'interrogation les plus utilisés nous citons :

Le langage booléen : la requête est exprimée par un ensemble de mots clés (termes de recherche) séparés par des opérateurs logiques (non, ou, et).

Le langage naturel : la requête est exprimée dans un langage libre. SMART [8], SPIRIT [11], OKAPI [18] et Mercure [14]sont des systèmes interrogeables en langage naturel

Le langage graphique : la requête est construite à partir d'une liste de mots clés connus. Pour ce faire, une vue d'ensemble représentant le contenu, notamment sémantique, des documents est offerte à l'utilisateur pour l'assister à formuler sa requête. PROTEUS [20] et NEURODOC [13] sont interrogeables en langage graphique.

1.2.4 Notion de pertinence :

La pertinence est une notion très importante en RI. Elle représente le degré de concordance d'un document avec la requête utilisateur.

On distingue deux types de pertinence :

-Pertinence système :

La pertinence système est définie par un score attribué par le SRI à chaque document, dans le but d'évaluer la correspondance de son contenu avec celui de la requête. Cette pertinence est objective et déterministe.

-Pertinence utilisateur :

La pertinence utilisateur est définie par les jugements de pertinence de l'utilisateur vis à vis des documents que le SRI lui retourne en réponse à sa requête. Ce type de pertinence est subjectif, car un même document retourné en réponse à une même requête, peut être jugé différemment par deux utilisateurs différents. La pertinence utilisateur est dite évolutive car, si à un instant t donné un document est jugé non pertinent, à l'instant $t+1$, il pourrait être jugé pertinent car la connaissance de l'utilisateur sur le sujet aura évolué .

La qualité d'un SRI réside dans sa capacité à calculer une pertinence système qui approche au mieux la pertinence utilisateur.

1.3 Le processus de recherche d'information :

Le processus de recherche d'information est mis en œuvre par un SRI. La figure représente schématiquement ce processus, couramment appelé processus en U.

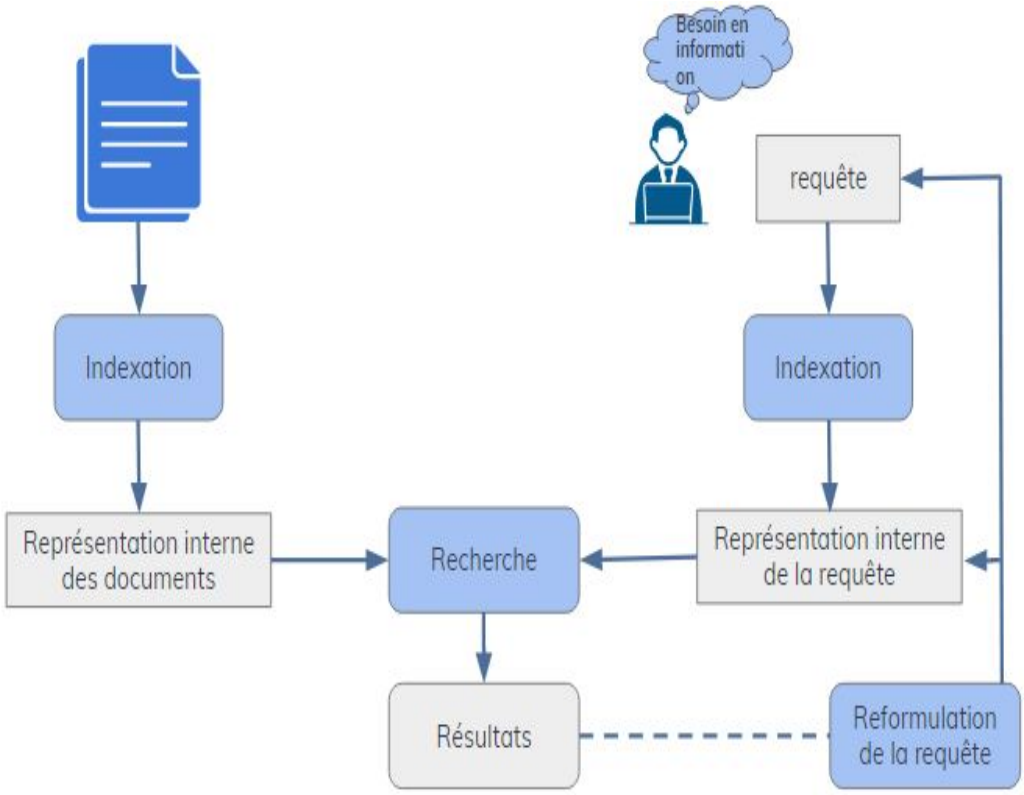


FIGURE 1.3.1 – Processus en U de la recherche d'information

Le processus de recherche consiste à comparer la représentation interne de la requête aux représentations internes des documents de la collection. L'utilisateur exprime ses besoins sous forme d'une requête. La requête est ensuite transformée en une représentation interne équivalente, lors d'un processus d'interprétation. Un processus semblable, dit indexation, par lequel on construit aussi la représentation interne des documents de la base documentaire. Le processus d'appariement, communément appelé processus de recherche, consiste alors à mettre en correspondance et à calculer le degré d'appariement des représentations internes des documents et de la requête. Les documents qui correspondent au mieux à la requête, ou documents pertinents, sont alors retournés à l'utilisateur, dans une liste ordonnée par ordre décroissant de degré de pertinence lorsque le système le permet. Afin d'améliorer les résultats de la recherche, le système peut être doté d'un mécanisme d'amélioration et de raffinement de la requête par reformulation.

1.4 L'indexation :

L'indexation [2] consiste à sélectionner un ensemble de termes représentatifs du contenu d'un document ou d'une requête. Le résultat de l'indexation est une représentation interne, dite descripteur de l'unité textuelle correspondante, composée de l'ensemble des termes sélectionnés. Les descripteurs des documents sont rangés dans une structure de données appelée index. L'ensemble des termes d'index constitue le langage d'indexation.

Il existe trois types d'indexations:manuelle, automatique ou semi-automatique. Nous les définissons ci-après :

1.4.1 Indexation manuelle :

Chaque document est analysé par un expert du domaine, qui identifie les mots clés appelés descripteurs. L'indexation manuelle fournit une terminologie pour indexer et rechercher des documents, assurant ainsi une meilleure qualité des résultats retournés par le système de RI. Cependant, l'indexation manuelle présente un effort trop coûteux en terme de temps et en besoin humain. De plus, un degré de subjectivité lié au facteur humain fait que le même document peut être indexé de différentes façons par des personnes différentes, et même par la même personne mais à des moments différents [9].

1.4.2 Indexation automatique :

Donc ce cas, l'indexation est totalement automatisée et est donc plus adaptée pour les bases documentaires de grandes tailles. Le SRI choisit les termes les plus représentatifs dans chaque document et leur associe des poids.

1.4.3 Indexation semi-automatique :

C'est une indexation hybride qui utilise l'indexation automatique et manuelle. À l'aide d'un processus automatique, une liste de descripteurs initiale est construite, puis un expert documentaliste intervient pour le choix final des termes à partir de cette liste.

L'indexation automatique reste la technique d'indexation la plus utilisée lors d'un processus de RI. Elle réalise les traitements en plusieurs étapes : l'analyse lexicale, l'élimination des mots vides, la normalisation et la pondération.

1.4.3.1 Analyse lexicale :

Comme première phase, l'analyse lexicale permet d'extraire du document un ensemble d'unités lexicales (ou termes). Une unité lexical est définie comme une séquence de caractères délimitée par des séparateurs (blanc,ponctuation...).

1.4.3.2 L'élimination des mots vides :

Elle consiste à extraire les termes significatifs et à éviter les mots vides (pronoms personnels, prépositions,...) .

Les mots vides peuvent aussi être des mots athématiques (les mots qui peuvent se retrouver dans n'importe quel document parce qu'ils exposent le sujet mais ne le traitent pas, comme par exemple *adapter*, *éliminer*,...etc). On distingue deux techniques pour éliminer les mots vides :

— L'utilisation d'une liste, déterminée au préalable, de mots vides (aussi appelée $anti-dictionnaire$ ou *stoplist* [5]).
— Détermination des mots les plus fréquents ou les plus rares dans une collection de documents, qui sont probablement des mots vides, par l'usage de mesures statistiques, ensuite passer à l'étape de l'élimination de ceux qui ont une fréquence qui dépasse un certain seuil dans cette collection (ou n'atteignent pas le seuil dans le cas des mots rares) .

1.4.3.3 La normalisation :

La normalisation consiste à représenter les différentes variantes d'un terme par un format unique appelé lemme ou racine, ce qui a pour effet de réduire la taille de l'index. Pour cela on utilise deux procédures : *la lemmatisation* et *la troncature*.

La lemmatisation prend en compte la forme canonique du mot, pour un verbe par exemple, on va considérer sa forme à l'infinitif, pour un nom, adjectif ...etc, on va prendre en compte sa forme au masculin.

La troncature, quant à elle, consiste à éliminer les suffixes des mots retenus à l'issu de l'étape précédente.

1.4.3.4 La pondération :

La pondération permet d'associer à chaque terme t_i un poids représentant son importance dans un document d_j. En général, ce poids est mesuré sur base de considérations statistiques en utilisant les deux mesures suivantes :

- La fréquence locale ou fréquence d'occurrence tf (*term frequency*) :

Cette mesure est proportionnelle à la fréquence tf du terme dans le document. L'idée est que, plus un terme est fréquent dans un document, plus il est important dans la description de ce document. Le tf est souvent exprimé selon l'une des déclinaisons suivantes :
— tf : utilisation brute.
— $0.5 + 0.5\left(\frac{tf}{\max(tf)}\right)$.

- La fréquence globale ou fréquence documentaire inverse idf (*inverced document frequency*) :

Idf mesure l'importance d'un terme dans toute la collection. L'idée est que les termes qui apparaissent dans peu de documents de la collection sont plus représentatifs du contenu de ces documents que ceux qui apparaissent dans tous les documents de la collection. Pour ce faire, on associe à chaque terme une mesure égale au logarithme de l'inverse de la proportion des documents qui contiennent le terme :

$$idf\,(t_i) = log\frac{|D|}{|\,\{d_j : t_i \in d_j\}\,|}$$

Avec :

t_i : le terme dont on cherche la fréquence.

$|\{d_j : t_i \in d_j\}|$: le nombre de documents où le terme t_i apparait.

$|D|$: le nombre total de documents dans la collection.

$idf\ (t_i)$: la fréquence globale du terme t_i.

Ces deux mesures (tf et idf) sont combinées pour fournir le poids du terme t_i comme suit :

$$w_{ij} = tf_{ij} \times idf_j$$

C'est cette mesure qui est utilisée le plus souvent en RI . Elle concrétise l'idée qu'un terme est important dans un document d'une collection si d'une part, il est très fréquent dans ce document (tf élevé), et peu fréquent dans les autres documents de la collection (idf élevé).

On distingue un autre facteur qui est la taille du document car plus ce dernier est long plus le terme apparait plus fréquemment. D'autres formulations de$tf_{ij} \times idf_j$ ont été proposées dont le but est de normaliser les fréquences en fonction de la taille des documents. La plus connue est la formule BM25 du système Okapi [21]

1.5 Appariement (Recherche) :

Les SRI intègrent un processus de *recherche/décision* qui permet de sélectionner l'information jugée pertinente pour l'utilisateur. Ce processus, dit d'appariement ou de comparaison, consiste a calculer une mesure de similitude (correspondance) entre la requête indexée et les descripteurs des documents de la collection. Seuls les documents dont la similitude dépasse un seuil prédéfini, sont sélectionnés par le SRI.

La fonction d'appariement est un élément clé d'un SRI, car la qualité des résultats dépend de l'aptitude du système à calculer une pertinence des documents la plus proche possible du jugement de pertinence de l'utilisateur. Il existe deux types d'appariements :

- Appariement exact :

Le résultat est une liste de documents respectant exactement la requête spécifiée avec des critères précis. Les documents retournés ne sont pas triés.

- Appariement approché :

Le résultat est une liste de documents censés être pertinents pour la requête. Les documents retournés sont triés selon leur degré de pertinence par rapport à la requête.

1.6 Reformulation de la requête :

Outre les processus d'indexation et d'appariement, un SRI peut aussi inclure un processus de reformulation de la requête. Ce processus a pour rôle d'améliorer la performance du système en offrant un mécanisme de raffinement de la requête utilisateur.

On peut citer deux catégories de techniques qui peuvent être utilisées au cours de ce processus : les méthodes locales et les méthodes globales.

Les méthodes locales :

Ces méthodes se basent sur la technique dite de réinjection de pertinence (*relevance feedback*). En se référant à la requête initiale, une liste de documents sera présentée à l'utilisateur comme résultat, où il choisira les documents qu'il considère comme pertinents, et ceux qu'il juge comme non pertinents. La requête est ensuite reformulée à l'aide des termes sélectionnés dans les documents jugés pertinents par l'utilisateur.

Les méthodes globales :

La requête est réécrite en se basant sur des ressources externes telles que les bases lexicales, les thésaurus et les ontologies. Ainsi, des termes (synonymes, termes sémantiquement liés, ...), qui ne sont pas forcément présents dans la requête initiale, sont proposés à l'utilisateur pour affiner sa requête.

1.7 Les modèles de recherche d'information :

Si c'est l'indexation qui choisit les termes d'index pour représenter le contenu d'un document ou d'une requête, c'est le modèle qui permet de donner une interprétation à ces termes d'index. Étant donné un ensemble de termes pondérés issus de l'indexation, le modèle remplit deux fonctions :

—La première est de créer une représentation interne pour un document ou pour une requête basée sur ces termes.

—La seconde est de définir une méthode de comparaison entre une représentation de document et une représentation de requête afin de déterminer leur degré de correspondance (ou similarité).

Le modèle joue un rôle central dans la RI. C'est lui qui détermine le comportement clé d'un SRI. De nombreux modèles existent, parmi lesquelles nous citons : *les modèles booléen*s, *les modèles vectoriels* et *les modèles probabilistes.*

1.7.1 Modèles booléens :

Le modèle booléen [19] est le plus simple des modèles de RI. C'est aussi le premier qui s'est imposé dans le monde de la *RI*, même de nos jours il est tout autant aussi utilisé. Ce

modèle est basé sur la théorie des ensembles et sur l'algèbre de bool, tels que les documents sont représentés par l'ensemble des termes qu'ils contiennent et la requête est représentée sous forme d'une équation logique contenant des termes reliés par des connecteurs logiques ($AND(\wedge)$, $OR(\vee)$, $NOT\,(\neg)$) .

L'appariement *RSV (Retrieval Status Values)* entre une requête q et un document d est une valeur booléenne : si un document implique (au sens logique) la requête, alors le document est pertinent. Sinon, il est considéré non pertinent. La RSV est formellement définie comme suit :

— RSV(d,t)=1 si le terme t∈d, sinon 0.

— RSV(d, NOT q_i)=1− RSV(d, q_i).

— RSV(d, q_i AND q_j)=RSV(d,q_i) AND RSV(d, q_j).

— RSV(d, q_i OR q_j)=RSV(d,q_i) OR RSV(d, q_j).

Le modèle booléen étant facile à mettre en œuvre présente cependant un certain nombre d'inconvénients tel que l'utilisateur doit avoir une grande connaissance du langage booléen, qui est assez complexe. De plus les documents qui sont retournés aux utilisateurs ne sont pas ordonnés par ordre de pertinence (en effet, la RSV est binaire : un document est soit pertinent ou non).

1.7.2 Modèles vectoriels :

Ce modèle introduit par *Gérard Salton*[7] est basé sur la théorie de l'algèbre et plus précisément sur le calcul vectoriel [10]. Un document est représenté sous forme d'un vecteur dans l'espace vectoriel composé de tous les termes d'index. Les coordonnées d'un vecteur document représentent les poids des termes correspondants. Formellement, un document d_i est représenté par un vecteur de dimension n [3].

$$d_i = (w_{i1}, w_{i2}, ..., w_{in}) \qquad pour\ i = 1, 2, ..., m$$

Avec :

w_{ij} : poids du terme t_j dans le document d_i.

m : nombre de documents dans la collection.

n : nombre de termes d'indexation.

Dans le même espace vectoriel que le document, on représente la requête Q :

$$Q = (w_{Q1}, w_{Q2,...}, w_{Qn})$$

Avec :

w_{Qj}:le poids de terme t_j dans la requête Q. Ce poids peut être soit une forme de $tf \times idf$, soit c'est à l'utilisateur de l'attribuer manuellement.

La pertinence du document d_i pour la requête Q est mesurée comme le degré de similarité entre le vecteur du document et le vecteur de la requête. Pour exprimer cette similarité(RSV) on utilise les mesures de similarités vectorielles, dont les suivantes :

Mesures	Formules
Le produit scalaire	$RSV\,(q,d_i) = \sum_{j=1}^{n} w_{qj}.w_{ij}$
La mesure de cosinus	$RSV\,(q,d_i) = \frac{\sum_{j=1}^{n} w_{qj}.w_{ij}}{\sqrt{\sum_{j=1}^{n} w_{qj}^2 \sum_{j=1}^{n} w_{ij}^2}}$
La mesure de Jaccard	$RSV\,(q,d_i) = \frac{\sum_{j=1}^{n} w_{qj}.w_{ij}}{\sum_{j=1}^{n} w_{qj}^2 \sum_{j=1}^{n} w_{ij}^2 - \sum_{j=1}^{n} w_{qj}.w_{ij}}$
La mesure de Dice	$RSV\,(q,d_i) = \frac{2\times\sum_{j=1}^{n} w_{qj}.w_{ij}}{\sum_{j=1}^{n} w_{qj}^2 \sum_{j=1}^{n} w_{ij}^2}$

TABLE 1.7.1 – Les mesures de similarité utilisées dans le modèle vectoriel

Les mesures les plus utilisées sont le calcul du produit scalaire et la mesure du cosinus. En terme de rapidité, la méthode du calcul scalaire est la plus adaptée par rapport à la méthode de calcul du cosinus qui est elle plus lente, mais son inconvénient est qu'elle renvoie une valeur de similarité non normalisée contrairement à la mesure du cosinus qui retourne une valeur de similarité normalisée (comprise entre 0 et 1)[16] [26] .

Les avantages [22] du modèle vectoriel sont :

- Pouvoir classer les documents par ordre de pertinence par rapport à une requête.
- Limiter le nombre de documents retournés à l'utilisateur en ignorant les documents avec un degré de similarité inférieur à un certain seuil.
- Les SRI se basant sur ce modèle présentent en général des résultats plus satisfaisants que ceux qui se basent sur le modèle booléen .

Néanmoins, ce modèle présente un inconvénient non négligeable, car il considère que les termes d'indexation forment une base, et donc qu'ils sont indépendants. Or, ceci est rare.ment le cas.

1.7.3 Modèles probabilistes :

1.7.3.1 Le modèle probabiliste de base :

Ce modèle est fondé sur le calcul de la probabilité de pertinence d'un document pour une requête. Le principe de base consiste à retrouver des documents qui ont, en même temps, une forte probabilité d'être pertinents, et une faible probabilité d'être non pertinents. Étant donné une requête utilisateur q et un document d, il s'agit de calculer la probabilité de pertinence du document pour cette requête.

Dans ce modèle, documents et requête sont représentés par des vecteurs de poids dans l'espace vectoriel des termes d'index[10].

La pertinence d'un document pour une requête est calculée comme suit :

$$RSV\,(d,q) = \frac{P\,(d|R)}{P\,(d|NR)}$$

Avec :

- $P\,(d|R)$: est la probabilité que le document d appartienne à l'ensemble R des documents pertinents.

— $P\,(d|NR)$:est la probabilité qu'un document d appartienne à l'ensemble NRdes documents non pertinents.

En appliquant le modèle BIR (Binary Independance Retrieval) les deux probabilités ci-dessus sont donnés par[1] :

$P(d|R) = p(t_1 = x_1, t_2 = x_2, ...|R) = \prod_i P(t_i = x_i|R) = \prod_i P(t_i = 1|R)^{x_i} * P(t_i = 0|R)^{1-x_i}$

$P(d|NR) = p(t_1 = x_1, t_2 = x_2, ...|NR) = \prod_i P(t_i = x_i|NR) = \prod_i P(t_i = 1|NR)^{x_i} * P(t_i = 0|NR)^{1-x_i}$

Tel que :

la variable document $d(t_1 = x_1, t_2 = x_2, .., t_n = x_n)$ est représentée par un ensemble d'évènements indépendants qui dénotent la présence $(x_i = 1)$ ou l'absence $(x_i = 0)$ d'un terme dans d.

En posant : $P(t_i = 1|R) = p_i$ et $P(t_i = 1|NR) = q_i$, on obtient :

$$RSV(d, q) = \frac{p_i^{x_i} * (1 - p_i)^{1-x_i}}{q_i^{x_i} * (1 - q_i)^{1-x_i}}$$

En appliquant la fonction *log* et après plusieurs transformations, le calcul du score de correspondance devient comme suit :

$$RSV(d, q) = \sum_{i;x_i=1} log\frac{p_i(1 - q_i)}{q_i(1 - p_i)}$$

À partir de l'ensemble R des documents pertinents et NR des documents non pertinents, on estime les probabilités p_i et $q_i$comme suit :

$$p_i = \frac{r_i}{n} \qquad et \qquad q_i = \frac{R_i - r_i}{N - n}$$

avec :

r_i : c'est les documents pertinents contenant le terme t_i.

n : c'est les documents pertinents.

R_i : c'est les documents contenant t_i.

$R_i - r_i$:c'est les documents non pertinents contenant t_i.

N : c'est le nombre de documents.

$N - n$: c'est les documents non pertinents.

Ce qui donne la formule suivante :

$$RSV(d,q) = \sum_{i,x_i=1} log \frac{r_i(N - R_i - n + r_i)}{(n - r_i)(R_i - r_i)}$$

L'inconvénient du modèle BIR est qu'il est impossible d'estimer ses paramètres si des collections de test ne sont pas disponibles, pour y pallier, S.Roberston a proposé la formule BM25, largement utilisée dans les travaux actuels de RI, la formule est la suivante :

$$Score(d,q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{tf(q_i,d) \cdot (k_1 + 1)}{tf(q_i,d) + k_1 \cdot (1 - b + b \cdot \frac{|d|}{avgdl})}$$

avec :

$tf(q_i, d)$: est la fréquence du terme $q_i$dans d.

$|d|$: est la langueur du document.

$avgdl$: est la langueur moyenne des documents.

$IDF(q_i)$: est la fréquence inverse du document pondérant la requête q_i.

k_1, b : sont des paramètres de contrôle, tel que $k_1 \in [1.2, 2.0]$ et $b = 0.75$.

Le modèle probabiliste de base est un modèle tout à fait fonctionnel qui a démontré son efficacité en recherche d'information. Cependant il présente l'inconvénient de générer des calculs de probabilités relativement complexes par rapport à d'autres modèles.

1.7.3.2 Modèle de langue :

Le modèle de langue est un modèle probabiliste, inspiré des techniques statistiques de modélisation de langue en informatique linguistique. Le principe de ce modèle consiste à construire un modèle de langue pour chaque document, soit M_d, puis de calculer la probabilité qu'une requête q puisse être générée par le modèle de langue du document, soit $P(q|M_d)$ [4] [17].

Le modèle de langue fait l'objet de notre présente étude. Une description détaillée de ce modèle est présentée dans le *Chapitre 02*.

1.8 Évaluation des SRI :

L'évaluation des approches de RI est nécessaire pour mesurer leur efficacité, leur performance et pour pouvoir les comparer en étudiant l'impact des différents facteurs employés dans ces approches. Un système de RI efficace doit répondre de façon satisfaisante aux besoins d'information de l'utilisateur en termes de qualité des résultats retournés, de rapidité du système ainsi que la facilité d'utilisation du système qui représentent les principaux facteurs à évaluer pour un système de RI [15]. Dans notre cas et de manière plus générale

en RI, on s'intéresse particulièrement à : la capacité d'un système à sélectionner des documents pertinents. Le mode d'évaluation généralement utilisé de nos jours est basé sur celui développé dans le projet Cranfield [6] communément appelé le paradigme de Cranfield. Ce paradigme définit la méthodologie d'évaluation des systèmes de RI en se basant sur trois éléments : une collection de documents sur laquelle les recherches sont effectuées, un ensemble de requêtes de test (besoins des utilisateurs) et la liste des jugements de pertinence des documents par rapport aux requêtes. L'idée générale de ce paradigme est de créer un environnement unique afin de pouvoir comparer les systèmes équitablement. Cet environnement est appelé la collection de test.

1.8.1 Collection de test :

La collection ou corpus de test est un aspect fondamental qui constitue le contexte d'évaluation. Généralement, chaque collection de test est caractérisée par une collection de documents, une ensemble de requêtes, et des jugements de pertinence des documents par rapport à ces requêtes. Dans une tâche de construction d'une collection de test, les jugements de pertinence constituent la tâche la plus complexe. Les jugements de pertinence indiquent pour chaque document du corpus s'il est pertinent, et parfois même son degré de pertinence, pour chaque requête. Afin de construire ces listes de jugements des documents pour toutes les requêtes, les utilisateurs (ou un groupe d'évaluateurs) doivent examiner le contenu de chaque document, et juger s'il est pertinent par rapport à la requête. Dans les campagnes d'évaluation tels que TREC, les collections de documents contiennent plusieurs millions de documents, ce qui rend impossible le jugement exhaustif de pertinence. Donc, dans le cas de grandes collections, les jugements de pertinence sont construits en se basant sur la technique de pooling [12], effectuée à partir des 1000 premiers documents retournés par les systèmes participants à l'évaluation. La Figure 1.8.1 illustre le protocole adopté par les campagnes d'évaluation officielles.

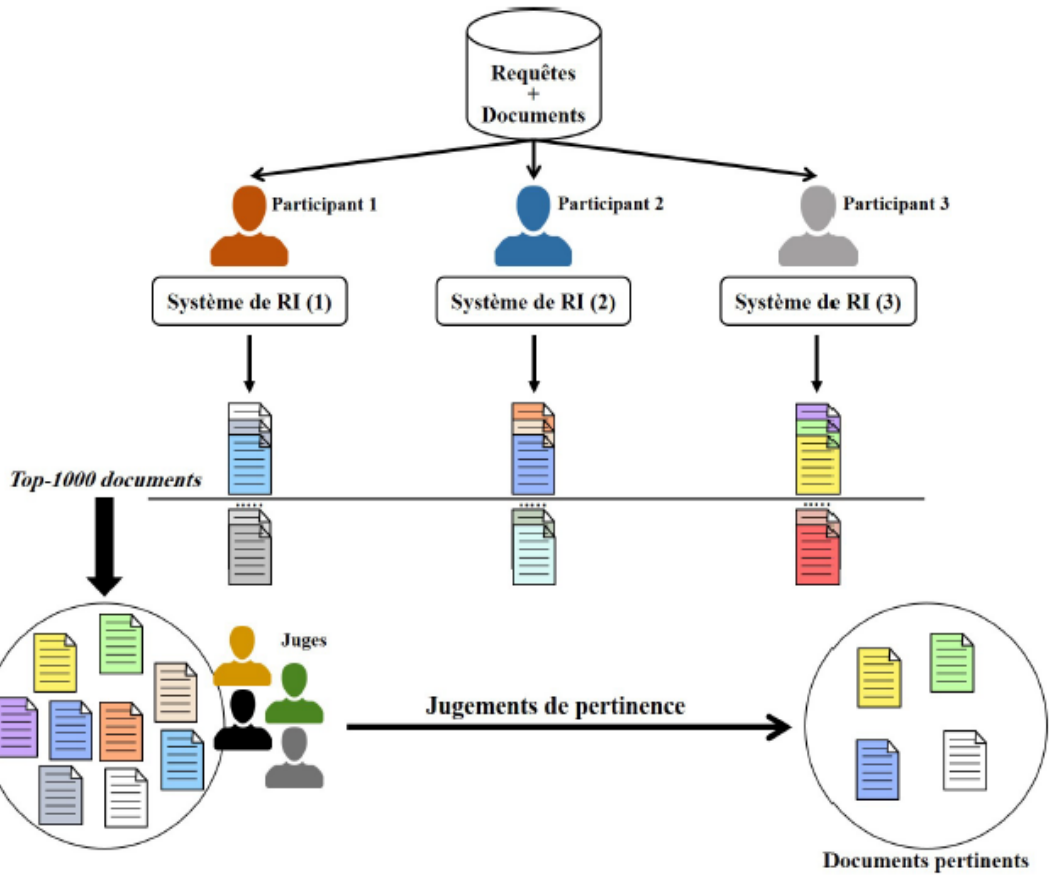


FIGURE 1.8.1 – Protocole adopté par les campagnes d'évaluation

Les collections de test sont le résultat de projets d'évaluation qui se sont multipliés depuis les années 1970, on peut citer la collection CACM [1], la collection CISI [2], la campagne CLEF [3] et la campagne TREC [4].

La taille des collections augmente au fil des années, passant de 2 Go dans TREC 1 à 25 To dans TREC 2011. Chaque collection est composée d'un certain nombre de documents, allant de quelques milliers à plusieurs millions. Les documents sont codés à l'aide de SGML dans un format spécifique TREC.

1.8.2 Campagnes d'évaluation :

De nombreuses campagnes d'évaluation sont apparues parmi elles on trouve la campagne TREC que nous décrivons dans ce qui suit :

TREC(Text REtrieval Conference)

1. http://www.search-engines-book.com/collections/
2. ftp://ftp.cs.cornell.edu/pub/smart/cisi
3. www.clef-campaign.org
4. http://trec.nist.gov/

La campagne TREC constitue à ce jour la campagne de référence dans le cadre de l'évaluation des systèmes de recherche d'information et cela depuis son lancement en 1992 [23][24]. Son objectif est de proposer une plate-forme qui réunit des collections de test, des tâches spécifiques et des protocoles d'évaluation pour chaque tâche ainsi que des mesures d'évaluation afin d'évaluer les différentes stratégies de recherche[14].

Les tâches proposées se sont diversifiées d'une campagne à une autre (à raison d'une par an), parmi les tâches proposées dans TREC on peut citer : recherche d'information sur le web, recherche d'information médical, recherche d'information dans les micros blogs, recherche d'information contextuelle, etc.

Cette campagne, co-organisée par le NIST6 et la DARPA, a pour but d'encourager la recherche documentaire basée sur de grandes collections de test, tout en fournissant l'infrastructure nécessaire pour l'évaluation des méthodologies de recherche et de filtrage d'information. Pour chaque session de TREC, un ensemble de documents et de requêtes (les "topics") est fourni. Les participants exploitent leurs propres systèmes de recherche sur les données et renvoient à NIST une liste ordonnée de documents. NIST évalue ensuite les résultats comme suit. L'ensemble des documents pertinents pour chaque requête est obtenu en prenant les K documents les mieux classées des différents SRI participant à la campagne d'évaluation. Ces documents sont ensuite montrés à des juges qui décident finalement de la pertinence de chaque document. Les participants à TREC disposent de la liste des documents pertinents pour chaque requête , et peuvent ainsi évaluer les performances de leurs SRI respectifs.

1.8.3 Mesures d'évaluation :

L'objectif principal est de quantifier, pour chaque requête la capacité du système à retourner des documents pertinents. La Figure 1.8.2 illustre les différents ensembles manipulés lors de l'évaluation d'un système de RI. Les documents pertinents non retournés par le système représentent l'ensemble de documents silence tandis que les documents non-pertinents retournés par le système génèrent du bruit. Un bon système retourne le maximum de documents pertinents (minimiser le silence) sans augmenter le nombre de documents non pertinents retournés (minimiser le bruit).[2]

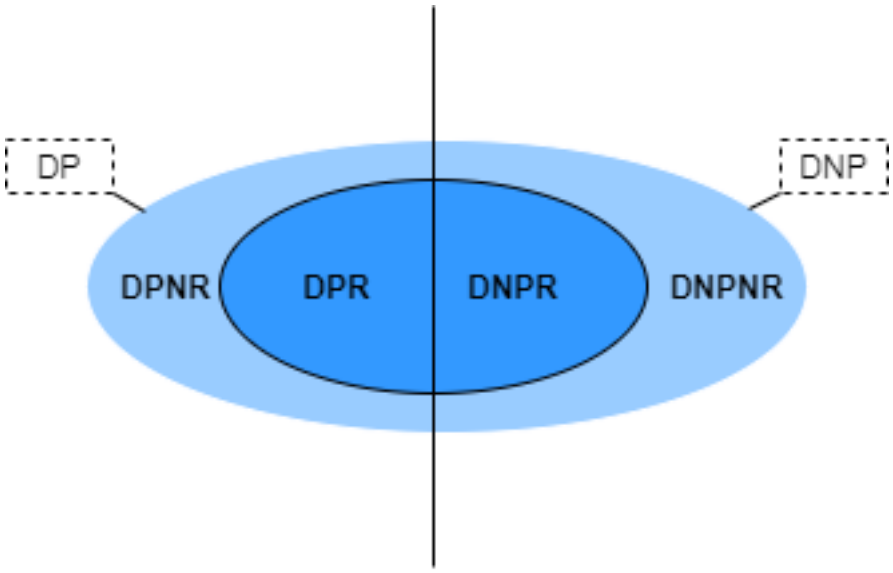


FIGURE 1.8.2 – Ensembles de documents utilisés pour l'évaluation d'un système de RI

avec :

DP : ensembles des documents pertinents.

DNP : ensembles des documents non pertinents.

$DPNR$: ensembles des documents pertinents non retournés.

$DNPNR$: ensembles des documents non pertinents non retournés.

DPR : ensembles des documents pertinents retournés.

$DNPR$: ensembles des documents non pertinents retournés.

Nous détaillons dans ce qui suit les principales mesures d'évaluation selon deux catégories :

1.8.3.1 Mesures d'évaluation non ordonnées :

Ces mesures sont calculées sans prendre en considération le classement des résultats de recherche.

— Rappel :

Le rappel mesure la proportion de documents pertinents retournés parmi tous les documents pertinents de la collection. Il est calculé avec la formule suivante :

$$rappel = \frac{|DPR|}{|DP|}$$

Où :

$|DPR|$: représente le nombre de documents pertinents retournés par le système.

$|DP|$: représente le nombre de documents pertinents dans la collection de documents.

Un taux de rappel = 1, signifie que tous les documents pertinents ont été restitués. Inversement, un taux de rappel = 0, signifie qu'aucun document pertinent n'a été restitué par le système.

Le rappel permet aussi de définir le *silence* documentaire qui représente la proportion des documents pertinents non retournés par le système :

$$silence = 1 - rappel$$

— Précision :

La précision mesure la proportion de documents pertinents restitués parmi tous les documents restitués. Elle est calculée avec la formule suivante :

$$précision = \frac{|DPR|}{|DPR| \cup |DNPR|}$$

Où :

$|DPR|$: représente le nombre de documents pertinents retournés par le système.

$|DPR| \cup |DNPR|$:représente le nombre de documents pertinents retournés par le système ainsi que l'ensemble des documents non pertinents retrouvés.

Un taux de précision = 1 signifie que seulement les documents pertinents ont été restitués. Tandis qu'un taux de précision = 0, signifie qu'aucun des documents retournés n'est pertinent.

La précision permet aussi de définir le *bruit* documentaire qui mesure la proportion de documents non pertinents retournés par le système :

$$bruit = 1 - précision$$

— La F-mesure :

Le rappel et la précision sont les principales mesures utilisées pour l'évaluation d'un SRI. Mais il existe d'autres mesures qui ne sont pas moins significatives, et qui pour la plupart combinent la précision et le rappel, dont la F-mesure (ou moyenne harmonique) qui permet d'agréger le rappel et la précision dans une mesure unique. La F-mesure est calculée comme suit :

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}$$

1.8.3.2 Mesures d'évaluation ordonnées :

Ces mesures sont calculées en prenant en considération le classement des résultats de recherche.

— P@x(Précision à x) :

La précision à x est la précision obtenue en considérant les x premiers résultats retournés par le SRI (en supposant que ceux-ci sont triés suivant leur pertinence).

— La moyenne des précisions moyennes ou MAP(Mean Average Precision) :

C'est la moyenne des précisions moyennes obtenues sur l'ensemble des requêtes à chaque fois qu'un document pertinent est retrouvé. Sa formule est comme suit

$$MAP = \frac{\sum_{q \in Q} AP(q)}{|Q|}$$

avec :

$|Q|$: l'ensemble des requêtes.

$AP(q)$: la précision moyenne aux documents pertinents vus pour une requête q. Elle se calcule comme suit :

$$AP(q) = \sum_{x=1}^{n} \frac{p@x \cdot pert(x)}{tpert}$$

Où :

n : indique le nombre de documents retournés par le système pour la requête q.

$p@x$: indique la précision à x.

$pert(x)$: cette fonction est égale à 1 si à la position x le document est pertinent, sinon elle sera égale à 0.

$tpert$: est le nombre total de documents pertinents retournés.

— Rang réciproque ou MRR :

Calcule l'inverse du rang auquel le premier document pertinent a été récupéré. Il s'agit d'une moyenne entre les requêtes, pour mesurer le rang réciproque moyen (MRR). Sa formule est comme suit :

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{R_i}$$

avec :

$|Q|$: la taille de la requête.

R_i : la position de classement du premier document pertinent pour la ième requête.

— La courbe rappel-précision interpolée :

La plupart des approches d'évaluation des SRI se basent sur les deux mesures de rappel et de précision. En effet, plus ces deux valeurs sont proches de 1 pour un système donné, plus ce système est performant. Ces deux mesures ont toujours une relation inverse, une augmentation de la précision, entraine une diminution du rappel et vice-versa. Par conséquent, elles doivent être adaptées pour une utilisation avec la liste des résultats classés du système RI.

En d'autres termes, plus la courbe rappel-précision interpolée d'un système décroit tardivement, plus ce système est performant.

Grâce à cette représentation, cela devient possible de comparer deux systèmes de recherche d'information. Particulièrement, si les deux courbes ne se croisent pas, on peut déduire que le système dont la courbe se trouve au dessus de l'autre, est plus performant (car il offre un taux de rappel et de précision plus élevé). Le Figure 1.8.3 est un exemple d'une courbe typique de rappel-précision, qui compare la performance globale de deux systèmes :

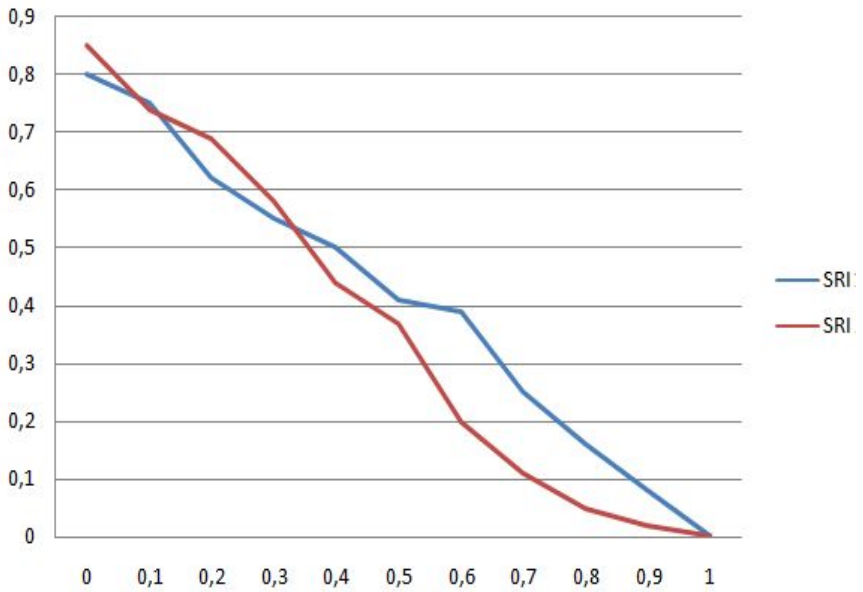


FIGURE 1.8.3 – Courbe rappel-précision

1.9 Conclusion :

Dans ce chapitre, nous avons passé en revue les principaux concepts de la RI. Nous avons, particulièrement, introduit des notions de base, telles que document ,collection de documents ,requête et notion de pertinence.

Nous avons décrit, à partir du processus dit en U, ses différents modules qui sont l'indexation, appariement et reformulation de la requête. Ensuite nous avons étudié les principaux modèles de la RI à savoir les modèles booléen, vectoriel , et probabiliste (avec sa variante : le modèle de langue). Enfin nous avons vu l'évaluation des SRI.

Le but d'un SRI est de rechercher l'information pertinente pour une requête utilisateur, par ailleurs dans le processus de recherche d'information le processus d'appariement joue un rôle primordiale dans le choix des documents pertinents qui seront retournés comme résultat à l'utilisateur. Afin d'obtenir un SRI performant, il est nécessaire de construire une bonne représentation du document et de la requête et de développer un modèle de RI qui supporte ces représentations. Dans le chapitre suivant, nous présentons en détail le modèle de langue, objet de notre étude.

Chapitre 2

Les Modèles De Langue

Introduction

Depuis leur première utilisation en recherche d'information (RI)[17], les modèles statistiques de langue ont acquis une grande popularité, en raison de leur efficacité et performance.

Un des atouts de cette nouvelle classe de modèles de RI, concerne leur fondement théorique, basé sur la théorie des probabilités. Le domaine de recherche d'information s'est beaucoup inspiré du succès des méthodes statistiques en linguistique informatique. Dans une certaine mesure, la RI a des choses en commun avec la linguistique informatique : Les deux domaines possèdent de grandes masses de textes, ce qui permet d'entrainer des modèles statistiques. Nos contributions exploitent largement ces modèles, notre objectif dans ce chapitre est de donner une vue globale de ces modèles et par la suite leur exploitation en RI. La partie qui suit dans la section 2.1 définit le principe général des modèles de langue puis la section 2.2 présente plus en détail les modèles de langue en recherche d'information, ensuite en section 2.3 les différentes techniques de lissage et enfin en section 2.4 lissage et modèle de recherche.

2.1 Les modèles de langue en linguistique informatique :

Les modèles de langue ont pour objectif d'étudier le langage en observant les séquences de termes dans un corpus d'entrainement. Ils définissent une distribution de probabilités sur toutes les séquences possibles "S" Figure2.1.1 ou autres unités linguistiques dans une langue.

Construire un modèle de langue statistique consiste à estimer une distribution de probabilité sur des séquences de termes dans une langue particulière. Sa fonction $P(S|M)$ qui est la probabilité de générer une séquence de mots S à partir du modèle de la langue M.

Supposons que$S = t_1 t_2 ... t_k$est une séquence de k termes, la probabilité de génération de la séquence S est formulée ainsi :

$$P(S|M) = \prod_{i=1}^{k} P(t_i|t_{i-n+1}...t_{i-1})$$

Dans le cas où $n = 1$, on parle du modèle *uni-gramme*. On considère dans ce modèle une indépendance entre tous les termes. La probabilité de chaque terme est estimée indépendamment des autres.

La probabilité $P(S|M)$ est formulée comme suit :

$$P(S|M) = \prod_{i=1}^{k} P(t_i)$$

Quand $n = 2$, le modèle de langue est dit *bi-gramme*. Il estime la probabilité de chaque terme sachant le terme le précédent immédiatement. La probabilité $P(S|M)$ est alors formulée ainsi :

$$P(S|M) = \prod_{i=1}^{k} P(t_i|t_{i-1})$$

Dans le cas où $n = 3$, le modèle de langue est dit *tri-gramme*. Il estime la probabilité de chaque terme sachant les deux termes précédant le terme courant. La probabilité $P(S|M)$ est alors formulée ainsi :

$$P(S|M) = \prod_{i=1}^{k} P(t_i|t_{i-2}t_{i-1})$$

Et il en va ainsi pour tous les modèles de langue d'ordre supérieur.

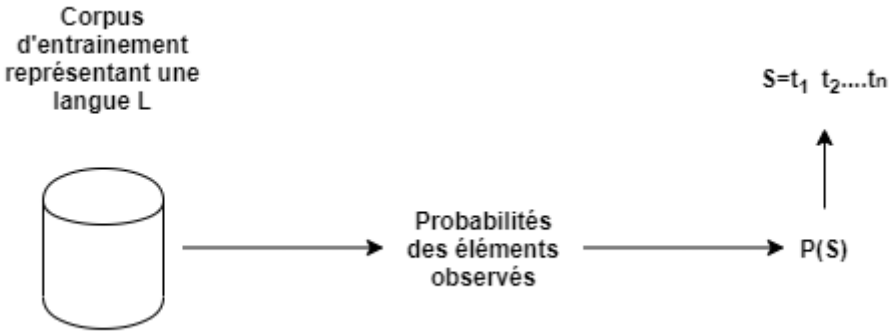


FIGURE 2.1.1 – Principe des modèles de langue en linguistique informatique

Ce que l'on doit estimer sont les probabilités $P(t_i)$ (*un-gramme*), $P(t_{i-1}t_i)$ (*bi-gramme*) et $P(t_{i-2}t_{i-1}t_i)$ (*tri-gramme*) pour la langue. Cependant, il est difficile d'estimer ces probabilités pour une langue dans l'absolue. L'estimation ne peut se faire que par rapport à un corpus de textes C. Si le corpus est suffisamment grand, on peut faire l'hypothèse qu'il reflète la langue en général. Ainsi, le modèle de langue peut être approximativement le modèle de langue pour ce corpus $P(\bullet|C)$. Selon les fréquences d'occurrence d'un n-gramme α, sa probabilité $P(\alpha|C)$ peut être directement estimée comme suit :[14]

$$P_{ML}(\alpha) = \frac{|\alpha|}{\sum_{\alpha_j \epsilon C} |\alpha_j|} = \frac{|\alpha|}{|C|}$$

Avec :

$|\alpha|$: la fréquence d'occurrence du n-gramme α dans ce corpus.

α_j : un n-gramme de la même longueur que α.

$|C|$: la taille du corpus (c'est-à-dire le nombre total d'occurrences de mots).

Ces estimations sont appelées les estimations de vraisemblance maximale (*Maximum Likelihood*, ou ML). On désignera aussi ces estimations par P_{ML}.

Pour illustrer l'estimation de la probabilité , nous donnons ici deux exemples simples :

— Cas uni-gramme :
Supposons un petit corpus contenant 10 mots, avec les fréquences comme montrées dans Tableau2.1.1 :

Mot	le	un	Lucene	est	recherche	bibliothèque	index	une	document	java
Fréquence	3	3	2	3	2	1	2	2	1	1
P(•|C)	0,15	0,15	0,1	0,15	0,1	0,05	0,1	0,1	0,05	0,05

TABLE 2.1.1 – Exemple d'estimation de vraisemblance maximale cas uni-gramme

En utilisant l'estimation de vraisemblance maximale, nous obtenons les probabilités comme illustrées dans la table (note : la fréquence totale de mots dans ce corpus est $|C| = 20$) .

Après estimation des probabilités, nous pouvons calculer la probabilité de construire la séquence $S =$ " *Lucene est une bibliothèque java* "dans cette langue comme suit :

$P(S|M) \approx P(S|C) = P(Lucene|C)*P(est|C)*P(une|C)*P(bibliothèque|C)*P(java|C) = 0,1 * 0,15 * 0,1 * 0,05 * 0,05$

— Cas bi-gramme :

Supposons le corpus du premier exemple(cas uni-gramme) contenant un certain nombre de bi-gramme, nous prenons 9 d'entre eux et leur donnons des fréquences de façon aléatoire comme montrées dans le Tableau2.1.2 :

Bi-gramme	une bibliothèque	une recherche	est une	est le	Lucene est	recherche le	bibliothèque java	un document	recherche index
Fréquence	2	1	1	2	1	2	3	1	2
P(•|C)	0,1	0,05	0,05	0,1	0,05	0,1	0,15	0,05	0,1

TABLE 2.1.2 – Exemple d'estimation de vraisemblance maximale cas bi-gramme

Après estimation des probabilités, nous pouvons calculer la probabilité de construire la séquence $S =$ " *Lucene est une bibliothèque java* " dans cette langue comme suit :

$P(S|M) \approx P(S|C) = P(est|Lucene)*P(une|est)*P(bibliothèque|une)*P(java|bibliothèque) = (\frac{P(Lucene\ est)}{P(Lucene)}) * (\frac{P(est\ une)}{P(est)}) * (\frac{P(une\ bibliothèque)}{P(une)}) * (\frac{P(bibliothèque\ java)}{P(bibliothèque)}) = \frac{0,05}{0,1} * \frac{0,05}{0,15} * \frac{0,1}{0,1} * \frac{0,15}{0,05}$

2.2 Les modèles de langue en recherche d'information :

Comme il a été dit, les modèles de langues en linguistique informatique ont pour but de déterminer si une séquence de mots en une langue est correct, ou de la générer à partir du modèle. La séquence de mots générée doit répondre aux critères linguistiques d'une langue, comme par exemple les contraintes syntaxiques et morphologiques .

Pour la RI, le principe diffère : on vise à retrouver les documents dont la sémantique est pertinente à celle de la requête. Dans la tâche de RI, on s'intéresse moins aux contraintes syntaxiques et morphologiques , mais c'est la notion de pertinence qui devient centrale .

2.2.1 Principe général des modèles de langue en recherche d'information :

La plupart des modèles de langue développés pour la RI se base sur le principe que la pertinence d'un document face à une requête est en rapport avec la probabilité que la requête puisse être générée par le modèle de langue du document. Cependant, il existe d'autres approches. Dans ce qui suit, nous allons étudiés toutes les approches :

3 approches ont été envisagées pour pouvoir évaluer cette probabilité :

$\alpha-$Calculer la probabilité que la requête soit générée par le modèle de langue du document :$P(Q|M_D)$.

$\beta-$Calculer la probabilité que le document soit généré par le modèle de langue de la requête : $P(D|M_Q)$.

$\gamma-$Comparer les modèles de langue de la requête et du document : M_Q et M_D.

2.2.1.1 Probabilité que la requête soit générée par le modèle de langue du document : $P(Q|M_D)$

Un document D est vu comme la représentation d'un langage auquel correspond un modèle de langue M_D. Le score du document face à une requête Q se caractérise par la probabilité que son modèle génère la requête :

$$Score(Q, D) = P(Q|M_D) \qquad (1)$$

Une requête est une suite de termes : $Q = t_1 t_2 \dots t_n$ ce qui permet d'avoir $Score(Q, D) = P(t_1 t_2 ... t_n|M_D)$.

Ce principe, dit *Query Likelihood,* est celui utilisé par la plupart des systèmes de recherche d'information basée sur les modèles de langue, qu'on détaillera par la suite.

2.2.1.2 Probabilité que le document soit généré par le modèle de langue de la requête :$P(D|M_Q)$

Cette approche consiste à construire un modèle de langue pour la requête et à déterminer le score d'un document $D = t_1 t_2 \dots t_m$ par la probabilité que le document puisse être généré par ce modèle :

$$Score(Q, D) = P(t_1 t_2 ... t_m|M_Q)$$

On utilise pas ou peu cette formule car il y a un certain déséquilibre qui se créé entre les documents longs et les documents courts, ainsi les documents longs seront favorisés.

2.2.1.3 Comparaison des modèles de langue de la requête et du document : M_Q et M_D

Une autre possibilité consiste à construire un modèle de langue pour le document $P(\bullet|M_D)$ et un autre pour la requête $P(\bullet|M_Q)$. Pour déterminer le score d'un document face à la requête on compare entre le $M_D$et le M_Q.

2.2.2 Approche de modélisation des systèmes de recherche d'information basé sur les modèles de langue :

2.2.2.1 Modèle de document et fonction de correspondance :

a-Cas uni-gramme :

Soit D_1 = l'ensemble des termes contigües de D de taille 1.

D_1=$\{(t_1), (t_2), ..., (t_i), ..., (t_{Nd})\}$

On appelle $M_{D,1}$le modèle uni-gramme du document D. On notera $M_{D,1}$ par M_D en donnant préalablement $n = 1$.

$M_D = \{(t_i, P(t_i)), \forall i \in [1..N_d]\}$avec $t_i \epsilon D_1$

$M_D = \{(t_1, P(t_1)), (t_2, P(t_2)), ..., (t_i, P(t_i)), ..., (t_{Nd}, P(t_{Nd}))\}$

avec $\forall i, \forall j \ i \neq j, t_i \neq t_j$

$P(t_i)$:estimation de la vraisemblance maximale, c'est à dire la fréquence du terme t_i sur la taille du document D.

$P(t_i) = \frac{tf(t_i,d)}{|d|}$

Sa fonction de correspondance est construite comme suit :

Pour chaque $D \in C$, la formule (1) est développée ainsi: $P(Q|M_D) = \prod_{i=1}^{N_q} P(t_i)$

$P(Q|M_D) = P(t_1) * P(t_2) * \ldots * P(t_i) * \ldots * P(t_{Nq})$

b-Cas n-gramme :

Soit D_n = l'ensemble des termes contigües de D de taille n.

$D_n = \{(t_1, ..., t_n), (t_2, ..., t_{n+1}), ..., (t_{Nd-n+1}, ..., t_{Nd})\}$

On appelle $M_{D,n}$ le modèle n-gramme du document D. On notera $M_{D,n}$ par M_D en donnant préalablement n.

$M_D = \{([t_{i-n+1}, ..., t_i], P([t_{i-n+1}, ..., t_i])), \forall i \in [n...N_d]\}$ *avec* $t_i \epsilon D$

$M_D = \{([t_1, ..., t_n], P([t_1, ..., t_n])), ([t_2, ..., t_{n+1}], P([t_2, ..., t_{n+1}])), ... ([t_{i-n+1}, ..., t_i], P([t_{i-n+1}, ..., t_i])), ..$
$, ([t_{Nd-n+1}, ..., t_{Nd}], P([t_{Nd-n+1}, ..., t_{Nd}]))\}$

$P([t_{i-n+1}, ..., t_i])$: La fréquence du n-gramme sur la taille du document D.

Sa fonction de correspondance est construite comme suit :

Pour chaque $D \in C$ et $t_i \in Q$ la formule (1) est développée ainsi :

$$P(Q|M_D) = \prod_{i=n}^{N_q} P(t_i|t_{i-n+1}...t_{i-1})$$

$P(Q|M_D) = P(t_i|t_{i-n+1} \dots t_{i-1}) * \dots * P(t_{Nq}|t_{Nq-n+1} \dots t_{Nq-1})$

$P(t_i|t_{i-n+1} \dots t_{i-1}) = \frac{P(t_{i-n+1}t_{i-n+2} \dots t_i)}{P(t_{i-n+1} \dots t_{i-1})}$

Dans la pratique, en recherche d'information, les modèles uni-grammes restent les plus utilisés compte tenu de leur performance. Les modèles bi-grammes s'avérant coûteux, peu de systèmes de recherche d'information les utilisent actuellement.

2.3 Lissage :

Pour éviter que l'absence d'un terme de la requête dans le document ne cause une probabilité nulle et ait une trop grande répercussion sur sa valeur de pertinence, le modèle du document est lissé. Cette opération de lissage peut également être interprétée comme une façon de modéliser l'importance des termes de la requête (Zhai et al, 2001). Plusieurs approches de lissage sont représentées, nous citons les suivants :

2.3.1 Lissage de Laplace

Cette méthode consiste à ajouter la fréquence un (1) à tous les *n-grammes*, appelé aussi *ajouter-un*. La probabilité du *n-gramme* est estimée ainsi :

$$P(\alpha) = \frac{|\alpha| + 1}{\sum_{\alpha_j \epsilon C} |\alpha_j| + 1} = \frac{|\alpha| + 1}{|C| + N}$$

Avec :

N : le nombre de n-grammes (distincts)

$|C|$: la taille du corpus.

L'inconvénient de cette méthode est que si le corpus ne contient qu'une petite portion des n-grammes parmi tous les n-grammes possibles (et c'est souvent le cas dans la pratique, même pour un grand corpus), la majeure partie de la masse de probabilité sera distribuée uniformément sur les n-grammes non vus dans le corpus. Les n-grammes vus dans le corpus ne joueront qu'un rôle mineur dans la définition du modèle. On ne peut donc pas s'attendre à une très bonne performance (c'est-à-dire de reconnaitre les phrases autorisées d'une langue correctement).

2.3.2 Lissage Good-Turing :

Cette méthode permet l'ajustement de la fréquence "r"d'un n-gramme (α) en une fréquence dite corrigée " r^*", exprimée ainsi :

$$r^* = (r+1) \times \frac{n_{r+1}}{n_r}$$

Avec :

n_r : le nombre de n-grammes de fréquence "r " dans la collection d'apprentissage. Ainsi, pour tout n-gramme (α) l'estimation de sa probabilité devient alors :

$$P_{GT}(\alpha) = \frac{r^*}{\sum_{\alpha_j \epsilon C} r_j^*}$$

Dans cette méthode, on applique une diminution de fréquence de l'ordre de $\frac{r^*}{r}$ au n-gramme α et cette fréquence est redistribuée sur les n-grammes non vus.

Quand la fréquence r est grande, le nombre n_r de n-grammes de cette fréquence est petit. Le lissage Good-Turing dans ce cas effectue une grande modification de la valeur de r. L'estimation Good-Turing pour les n-grammes de grande fréquence a donc tendance d'être instable. Dans l'autre bout du spectre, l'estimation de Good-Turing pour les n-grammes de faibles fréquences sont plus stables. C'est souvent pour ces derniers n-grammes que le lissage Good-Turing est recommandé.

2.3.3 Lissage Jelinek-Mercer

Le lissage Jelinek-Mercer est un lissage d'interpolation, il consiste à combiner un modèle avec un ou des modèles d'ordre inférieur systématiquement, plutôt que d'utiliser ce dernier seulement dans le cas de fréquence 0. Pour une combinaison de modèle bi-gramme avec le modèle uni-gramme, on a :

$$P_{JM}(t_i|t_{i-1}) = \lambda_{t_{i-1}} P_{ML}(t_i|t_{i-1}) + (1 - \lambda_{t_{i-1}}) P_{JM}(t_i)$$

où

$\lambda_{t_{i-1}}$: un paramètre déterminé de telle manière à maximiser l'espérance des données tel que $\lambda \in [0, 1]$. Ce paramètre peut dépendre du terme t_{i-1} ou il peut être attribué d'une

valeur identique pour tous les termes. Sa valeur est souvent déterminée avec le processus de maximisation de l'espérance (EM).

Dans le contexte d'utilisation des modèles de langue en RI, on se limite souvent aux modèles uni-gramme. Dans ce cas, ce lissage prend un autre sens. Nous avons deux modèles uni-grammes, l'un estimée sur un document D et l'autre sur un grand corpus C. Le modèle de langue pour le document est une interpolation du modèle de langue du document $P(\bullet|d)$ avec le modèle de langue du corpus $P(\bullet|C)$. Dans ce cas, le modèle de document est exprimé ainsi :

$$P_{JM}(t_i|d) = (1-\lambda)P_{ML}(t_i|d) + \lambda P_{ML}(t_i|C)$$

Les modèles $P_{ML}(t_i|d)$ et $P_{ML}(t_i|C)$ sont estimés selon le maximum de vraisemblance.

L'inconvénient de ce lissage est qu'il ne tient pas compte de la taille des échantillons.

2.3.4 Lissage Absolute Discount :

Le lissage Absolute Discount (Ney,Essen,and Kneser,1994) diminue la probabilité des termes observés dans le document en soustrayant une constante δ de leur compte. Les comptes enlevés sont redistribués à l'ensemble des termes de la collection selon leur probabilité dans la collection. On a :

$$P_{AD}(t_i|d) = \frac{max(tf(t_i,d)-\delta, 0)}{|d|} + \frac{\delta|d|_u}{|d|}P_{ML}(t_i|C)$$

avec :

$tf(t_i, d)$: le nombre d'occurrences du terme t_i dans le document d.

$|d|_u$: le nombre de termes uniques (distincts) dans le document.

$\delta \in [0, 1]$: une constante qui diminue l'importance des termes connus par le modèle de langue.

2.3.5 Lissage de Dirichlet :

Le lissage de Dirichlet exploite les valeurs de λ en fonction de la taille de l'échantillon, contrairement au lissage Jelinek-Mercer. Dans ce cas cette formule s'écrit comme suit :

$$\begin{aligned} P_{Dir}(t_i|d) &= \frac{|d|}{|d|+\mu}P_{ML}(t_i|d) + \frac{\mu}{|d|+\mu}P_{ML}(t_i|C) \\ &= \frac{|d| \times P_{ML}(t_i|d) + \mu \times P_{ML}(t_i|C)}{|d|+\mu} \\ &= \frac{tf(t_i,d) + \mu P_{ML}(t_i|C)}{|d|+\mu} \end{aligned}$$

avec : $P_{ML}(t_i|d) = \frac{tf(t_i,d)}{|d|}$

où $|d|$est la taille du document (le nombre d'occurrences de termes), $tf(t_i, d)$ est la fréquence du terme t_i dans d et μ est un paramètre appelé pseudo fréquence tel que $\mu \in [0, +\infty]$.

Les méthodes de lissages les plus répandues sont le lissage de Jelinek-Mercer et le lissage de Dirichlet qui permettent de pénaliser les termes fréquents au profit des termes plus rares, ce qui correspond à l'effet *idf* dans le modèle vectoriel.

2.4 Lissage et modèle de recherche :

En surface, les modèles de langues semblent fondamentalement différents des modèles vectoriels et du schéma de pondération TF-IDF, car le modèle utilise explicitement le TF mais pas le IDF. Cependant, il existe un lien intéressant entre l'approche du modèle de langue et les heuristiques utilisées dans les modèles traditionnels. Cette connexion entre les deux modèles est établie grâce au lissage utilisé dans le modèle de langue.

La formule du modèle de langue après lissage est comme suit :

$$logP(q|d) = \sum_{i=1}^{n} logP(t_i|d)$$

$$= \sum_{i,tf>0} logP(t_i|d) + \sum_{i,tf=0} log\alpha_d P(t_i|C)$$

donc

$$P(t|d) = \begin{cases} P(t|d) \ si \ t \ vu \\ \alpha_d P(t|C) \ si \ non \end{cases}$$

avec :

— α_d :c'est un coefficient attribué selon le lissage afin de contrôler la masse de probabilité attribuée aux termes non-vus.

on pose :

$$\sum_{i,tf=0} log\alpha_d P(t_i|C) = \sum_{i} log\alpha_d P(t_i|C) - \sum_{i,tf>0} log\alpha_d P(t_i|C)$$

on aura :

$$\sum_{i,tf>0} logP(t_i|d) + \sum_{i} log\alpha_d P(t_i|C) - \sum_{i,tf>0} log\alpha_d P(t_i|C)$$

$$= \sum_{i,tf>0} log\frac{P(t_i|d)}{\alpha_d P(t_i|C)} + \sum_{i} log\alpha_d P(t_i|C)$$

$$= \sum_{i,tf>0} log\frac{P(t_i|d)}{\alpha_d P(t_i|C)} + \sum_i (log\alpha_d + logP(t_i|C))$$

On obtient la formule générale de classement :

$$logP(q|d) = \sum_{i,tf>0} log\frac{P(t_i|d)}{\alpha_d P(t_i|C)} + nlog\alpha_d + \sum_i logP(t_i|C)$$

À partir de cette formule, nous pouvons voir que le lissage avec le modèle de langue de collection nous donne un effet de pondération semblable à la pondération TF-IDF plus une normalisation de la longueur tel que :

La première partie $\frac{P(t_i|d)}{\alpha_d P(t_i|C)}$ reflète la pondération TF-IDF avec :

$P(t_i|d)$:est lié à la pondération TF, dans le sens où si le terme t_i apparait fréquemment dans le document alors la probabilité estimée va être grande.

$P(t_i|C)$:joue le rôle du IDF, donc vu qu'il est dans le dénominateur , plus il est grand et plus le poids sera petit.

Dans la deuxième partie $log\alpha_d$, α_d joue le rôle de la normalisation de la longueur des documents qui est une autre technique importante pour améliorer les performances dans les modèles.

Et de ce qui est de $\sum_i logP(t_i|C)$ elle sera ignorée car elle est la même pour tous les documents ce qui permet de ne pas affecter le classement de ces derniers.

2.5 Conclusion

La recherche d'information s'avérant un processus incertain et imprécis, une incertitude dans la représentation des documents et dans l'expression des besoins de l'utilisateur existe. Ainsi les modèles probabilistes tentent d'estimer la probabilité qu'un document donné soit pertinent pour une requête donnée. Une autre façon de modéliser les documents sous forme probabiliste apparait avec les modèles de langue. Ces modèles, initialement utilisés dans les domaines de la reconnaissance de la parole et de la traduction automatique de texte, fournissent une représentation du langage. Les modèles de langue uni-gramme ont montré leur efficacité sur les tâches de recherche d'information. Ces modèles tiennent compte de la probabilité des termes de la requête dans l'ensemble du corpus. Le lissage permet d'éviter les probabilités nulles pour des documents dans le cas où un mot de la requête s'avère absent du document mais que tous les autres mots demeurent présents. Dans le chapitre suivant, nous allons introduire l'API Lucene et par la suite décrire les approches d'extension de celle ci.

Chapitre 3

Analyse et Conception

3.1 Introduction :

Lucene est une API libre, permettant de créer un moteur de recherche afin d'indexer et de rechercher du texte et des documents [1]. Cette API permet d'effectuer des recherches " plein texte ". Elle est entièrement écrite en Java, mais a aussi été portée dans différents autres langages tels que C/C++, Ruby, .NET, Python ou PHP.

Notre objectif est d'étendre cette API avec différentes approches que nous allons proposer et ainsi tenter d'améliorer ses capacités de recherche.

Dans ce qui suit, nous allons présenter l'API Lucene, puis détailler nos approches de lissage proposées pour l'extension de cette API.

3.2 Présentation de Lucene :

Lucene agit comme une couche intermédiaire entre les données à indexer et une application. Pour ce faire, il indexe des documents. Par document, on ne parle pas de fichiers de type Excel, Word, PDF ou HTML, mais d'une structure de données constituée de champs tels que SGML. Un champ est une donnée possédant un nom (titre, auteur, date de publication, contenu, etc.) et à laquelle est associée une valeur et plus généralement du texte. C'est ce texte qui est indexé, recherché et affiché. Les documents indexés sont regroupés au sein d'une collection appelée " index ".

À partir de l'index, Lucene permet ensuite une recherche rapide et efficace dans ces documents. L'utilisation de Lucene implique nécessairement une couche de programmation afin de pouvoir mettre en œuvre une indexation, une IHM de moteur de recherche,etc.

1. http://lucene.apache.org/core/documentation.html, visité le 16 juillet 2012

3.2.1 Architecture de Lucene :

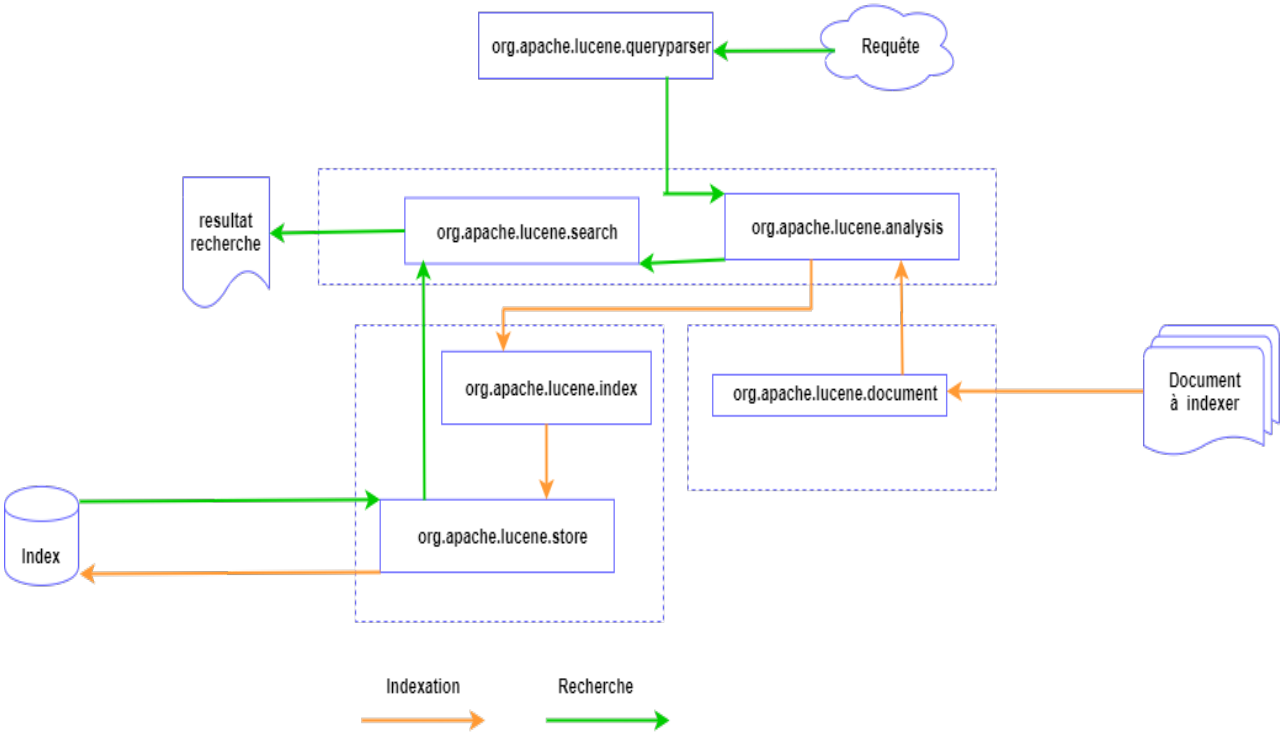


FIGURE 3.2.1 – Architecture de Lucene

Lucene est composé de 7 paquetages principaux :

-Org.apache.lucene.analysis : il contient du code afin de convertir du texte en élément indexable. Il contient la classe :

- **Analyzer** : permet d'extraire les mots importants pour l'index et supprimer le reste. Lucene fournit un certain nombre d'implémentation d'analyseurs qui devrait convenir à la plupart des applications de recherche, nous citons : **StandarAnalyzer**, **SimpleAnalyzer**, **StopAnalyzer**, **KeywordAnalyzer**.

-Org.apache.lucene.document : contient des classes relatives aux documents, tel que les classes :

- **Document** : représente un rassemblement de champs(Fields), ainsi les métadonnées sont indexées et stockées séparément comme des champs d'un document.
- **Field :** Chaque document contient un ou plusieurs champs, inséré dans une classe intitulé Field. Chaque champ (Field) correspond à une portion de donnée qui est interrogée ou récupérée depuis l'index durant la recherche.

-Org.apache.lucene.index : il contient le code pour accéder aux index. On y trouve les classes :

- **IndexWriter** : permet de créer un index ou d'ajouter des documents dans un index existant.
- **IndexReader :**permet d'accéder à l'index lors des opérations de recherche.

-Org.apache.lucene.queryparser : son rôle est d'analyser les requêtes afin de générer la requête sous forme d'objet query qui pourront ensuite être réutilisé par le parseur. On y trouve la classe :

— **QueryParser** : est un analyseur de requête qui permet de chercher à travers l'index.

-Org.apache.lucene.search :il se charge de fournir les objets pour chercher dans les indexes. Il fourni les classes suivantes :

— **IndexSearcher** : la classe IndexSearcher est la classe qui se charge de l'ouverture de l'index en lecture seulement.
— **Query** : c'est la méthode la plus basique d'interrogation de Lucene, elle permet d'extraire les documents qui contiennent des champs avec des valeurs spécifiques.
— **Hits** : est un conteneur d'index pour classer les résultats de recherche de document. Pour des raisons de pertinences, les exemples de classements ne chargent pas tous les documents de l'index pour une requête donnée, mais seulement une partie d'entre eux.
— **TopDocs :** cette classe est un simple conteneur de pointeurs vers les N premiers documents des résultats de recherche, qui correspondent à une requête donnée.
— **Similarities** : Ce module est le noyau de la recherche sur Lucene, il permet de mettre en correspondance le vecteur du document et le vecteur de la requête.

-Org.apache.lucene.store : représente une couche d'abstraction d'entrée sortie. On y trouve les classes :

— **Directory** : représente l'emplacement de l'index de Lucene.
— **FSDirectory** : c'est une classe qui étend de la classe Directory, elle sert à stocker des fichiers d'index dans le système de fichiers.

Lucene intègre un autre paquetage qui est **org.apache.lucene.util**, définit comme suit :

-Org.apache.lucene.util : les classes de ce paquetage sont utilisées par les autres paquetages. Par exemple on y trouve la classe Version qui permet de préciser la version de Lucene utilisée.

D'après l'architecture représentée dans la Figure 3.2.1, on déduit que le moteur de recherche Lucene est caractérisé par deux fonctionnalités ou deux processus principaux qui sont : L'indexation et la recherche, que nous allons détailler ci dessous.

3.2.2 Fonctionnement de Lucene :

— **Processus d'indexation :**

Le processus d'indexation est l'une des fonctionnalités de base fournies par Lucene. Le diagramme suivant illustre le processus d'indexation et l'utilisation des classes. *IndexWriter* est le composant le plus important du processus d'indexation.

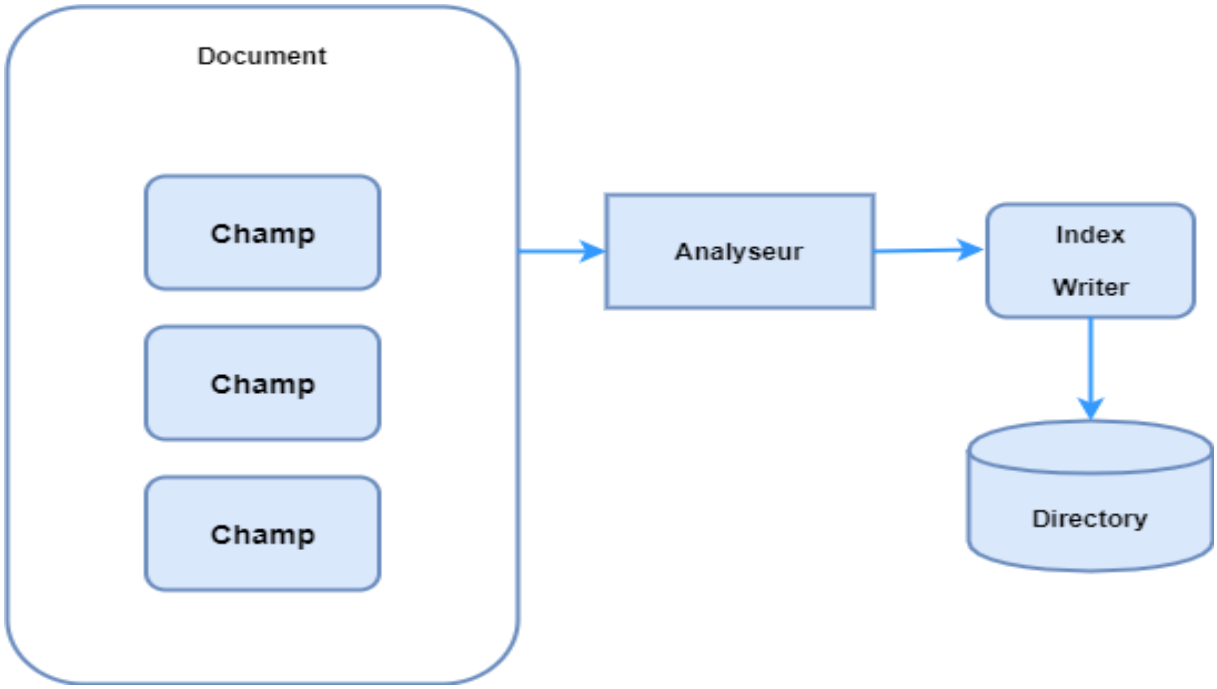


FIGURE 3.2.2 – Processus d'Indexation

Le ou les documents constitués d'un ou de plusieurs champs, où chaque champs a une valeur, sont soumis à l'alyseur qui divise le document ou les documents en une série d'éléments atomiques individuels appelés tokens grâce à *Analyzer*. Chaque token correspond à un "mot" dans la langue. Ensuite *IndexWriter* crée un index, ou le met à jour s'il l'index existe déjà en rajoutant ou supprimant des documents dans cet index. L'index créé est stocké dans un répertoire unique appelé *Directory*.

— **Processus de recherche :**

Le processus de recherche est la deuxième fonctionnalité essentielle fournie par Lucene. Le diagramme suivant illustre le processus et son utilisation. IndexSearcher est l'un des principaux composants du processus de recherche.

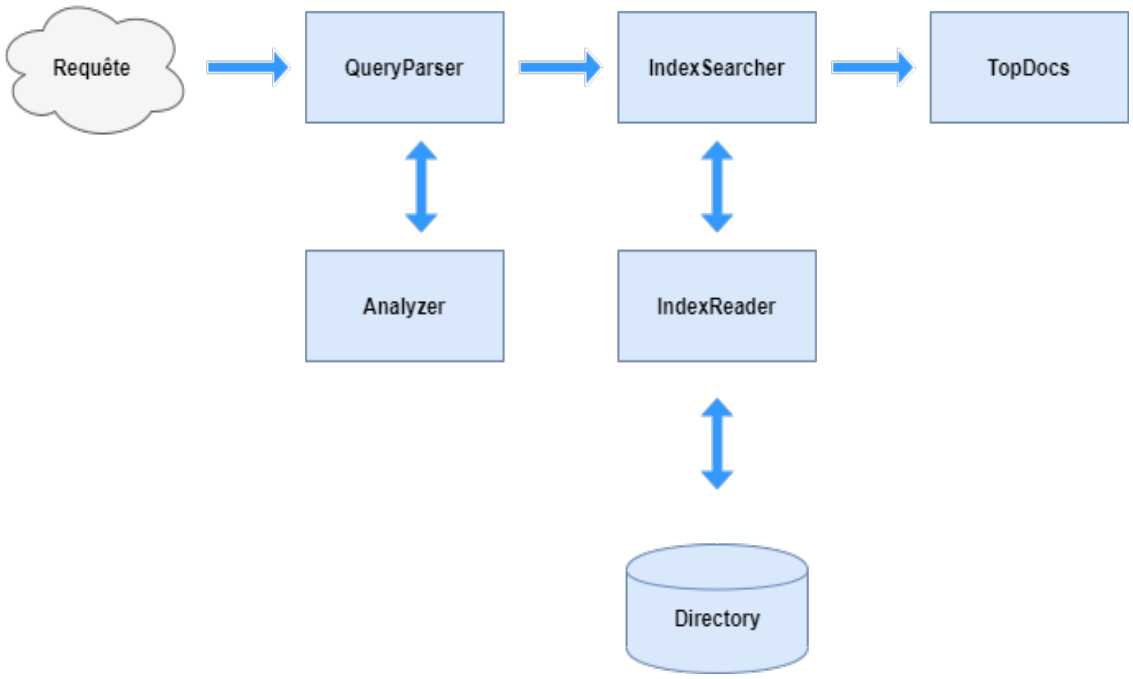


FIGURE 3.2.3 – Processus de recherche

La requête de l'utilisateur est fournie à QueryParser. QueryParser utilise ensuite un analyseur pour traiter la requête afin de produire un ensemble de tokens. Les tokens sont ensuite envoyés à IndexSearcher, qui ouvre le répertoire où se trouve l'index créé lors du processus d'indexation à l'aide de IndexReader, pour exécuter une recherche. Le résultat de la recherche renvoyé par IndexSearcher est un objet TopDocs où il contient des statistiques sur les correspondances totales et les DocIds des documents correspondants.

3.2.3 Les formules de similarité de Lucene :

Au cours de la recherche, Lucene assigne un score à chaque document.

Lucene intègre différentes classes de similarité nous résumons quelques unes d'entre elles dans le Tableau suivant :

Similarités	Formules	Paramètres	Descriptions
Par Défaut	$\sum_i \left(tf\left(t_i, d\right)^{\frac{1}{2}} \cdot idf\left(t_i\right)^2 \right)$	/	Se base sur les modèles booléen et la pondération TF-IDF du modèle vectoriel vectoriel
BM25	$\sum_i idf \cdot \frac{tf(t_i,d)}{k_i((1-b)+b\frac{\|d\|}{avgld})+tf(t_i,d)}$	k_1, b	C'est un modèle probabiliste basé sur une approche TF-IDF
LM Dirichlet	$\sum_i (log(1+\frac{tf(t_i,d)}{\mu P(t_i\|C)}) + log(\frac{\mu}{\|d\|+\mu}))$	μ	Se base sur le modèle de langue utilisant le lissage de Dirichlet
LM Jelinek-Mercer	$\sum_i log(1+\frac{\frac{(1-\lambda)tf(t_i,d)}{\|d\|}}{\lambda P(t_i\|C)})$	λ	Se base sur le modèle de langue utilisant le lissage Jelinek-Mercer

TABLE 3.2.1 – Classes de similarité intégrés dans Lucene

3.3 Les approches de lissage proposées :

Le modèle de langue a été implémenté dans Lucene avec deux lissages différents : le lissage Dirichlet et le lissage Jelinek-Mercer, notre objectif serait d'étendre Lucene avec d'autres lissages du modèle de langue, nous avons opté pour les suivants : Jelinek-Dirichlet, Dirichlet-Jelinek et Absolute Discount.

Ci dessous, nous allons détailler ces approches :

3.3.1 Première approche (approche principale) :

L'étude de Zhai et al [25]suggère que la performance des modèles de langues est influencée par le choix des paramètres (λet μ) et qu'elle est variable suivant le type de requêtes. Ainsi, le lissage de Jelinek-Mercer a pour avantage de donner de meilleurs résultats sur les requêtes longues mais de moins bons résultats sur les requêtes courtes, contrairement au lissage de Dirichlet qui a pour avantage d'obtenir de meilleurs résultats sur les requêtes courtes mais il est moins bons pour les requêtes longues, de plus il prend en considération la taille des documents, contrairement à Jelinek-Mercer, ce qui permet de moins lisser les documents longs.

Nous partons de l'idée que la combinaison de ces deux lissages pourrait apporter une solution intéressante car cella permettra de joindre les avantages de l'une et de l'autre.

Pour combiner ces deux lissages, nous proposons les schémas suivants :

Schéma 1 : Jelinek-Dirichlet

$$P_{JD} = P_{\lambda,\mu}(t|d) = (1-\lambda)\frac{tf(t,d) + \mu P(t|C)}{|d| + \mu} + \lambda P(t|C)$$

Cette formule nous l'avons obtenue en remplaçant le $P_{ML}(t|d)$ de la formule de Jelinek-Mercer $P_{JM}(t|d)$ avec la formule de Dirichlet $P_{Dir}(t|d)$

avec : $P_{JM}(t|d) = (1-\lambda)P_{ML}(t_i|d) + \lambda P_{ML}(t|C)$ et$P_{ML}(t_i|d) = P_{Dir}(t|d) = \frac{tf(t,d)+\mu P(t|C)}{|d|+\mu}$

Cette approche est constituée de deux étapes :

La première étape consiste à lisser le modèle de langue de document $(P_{ML}(t|d))$ avec Dirichlet, ensuite dans la deuxième étape il est encore lissé avec Jelinek-Mercer.

Schéma 2 : Dirichlet-Jelinek

$$P_{DJ} = P_{\lambda,\mu}(t|d) = \frac{(1-\lambda)tf(t,d) + |d|\lambda P(t|C)}{|d| + \mu} + \frac{\mu}{|d| + \mu}P(t|C)$$

Cette formule nous l'avons obtenue en remplaçant le $P_{ML}(t|d)$ de la formule de Dirichlet $P_{Dir}(t|d)$ avec la formule de Jelinek-Mercer $P_{JM}(t|d)$

avec : $P_{Dir}(t|d) = \frac{|d|\times P_{ML}(t_i|d)+\mu\times P_{ML}(t_i|C)}{|d|+\mu}$et $P_{ML}(t_i|d) = (1-\lambda)\frac{tf(t,d)}{|d|} + \lambda P_{ML}(t_i|C)$

Cette approche est constituée de deux étapes :

La première étape consiste à lisser le modèle de langue de document $(P_{ML}(t|d))$ avec Jelinek-Mercer, ensuite, dans la deuxième étape, il est encore lissé avec Dirichlet.

3.3.2 Deuxième approche :

Le lissage Absolute Discount, comme le lissage Jelinek-Mercer, interpole le modèle de langue de document avec celui du corpus, mais contrairement à Jelinek qui diminue le $P_{ML}(t|d)$ en le multipliant par $(1-\lambda)$, Absolute Discount diminue $P_{ML}(t|d)$ en soustrayant une constante δ du compte. Cette approche de lissage n'est pas implémentée dans Lucene, nous préconisons d'étendre ce dernier avec cette approche.

Sa formule a été introduite dans la *Chapitre 2* dans la section Lissage.

3.4 Intégration des approches de lissage dans le modèle de recherche :

a-Description générale :

Une fonction de classement pour les modèles langue est intégrée sur la base des différentes approches. Cette fonction est la formule générale de classement vu dans le *Chapitre 2* :

$$logP(q|d) = \sum_{i:tf(q_i,d)>0} log\frac{P(q_i|d)}{\alpha_d P(q_i|C)} + nlog\alpha_d + \sum_i logP(q_i|C)$$

En appliquant cette formule aux différents lissages vu, nous obtenons les résultats suivants :

Similarités	Formules	α_d	Paramètres
Absolute Discount	$\sum_i(log(1 + \frac{max(tf(t_i-\delta,0)}{\delta\|d\|_u P(t_i\|C)}) + log(\frac{\delta\|d\|_u}{\|d\|}))$	$\frac{\delta\|d\|_u}{\|d\|}$	δ
Jelinek-Dirichlet	$\sum_i(log(1 + \frac{(1-\lambda)tf(t_i,d)}{((1-\lambda)\mu+\lambda(\|d\|+\mu))P(t_i\|C)}) + log(\frac{(1-\lambda)\mu}{\|d\|+\mu} + \lambda))$	$\frac{(1-\lambda)\mu}{\|d\|+\mu} + \lambda$	λ, μ
Dirichlet-Jelinek	$\sum_i(log(1 + \frac{(1-\lambda)tf(t_i,d)}{((\|d\|\lambda+\mu)+\mu)P(t_i\|C)}) + log(\frac{\|d\|\lambda+\mu}{\|d\|+\mu}))$	$\frac{\|d\|\lambda+\mu}{\|d\|+\mu}$	λ, μ

TABLE 3.4.1 – Tableau des formules implémentées dans Lucene

b-Architecture d'intégration :

Le schéma ci-dessous montre une partie de l'architecture de Lucene dans laquelle nous avons implémenter les classes de nos approches (en rouge). Les résultats de cette implémentation seront représentés et évalués au niveau du *Chapitre4*.

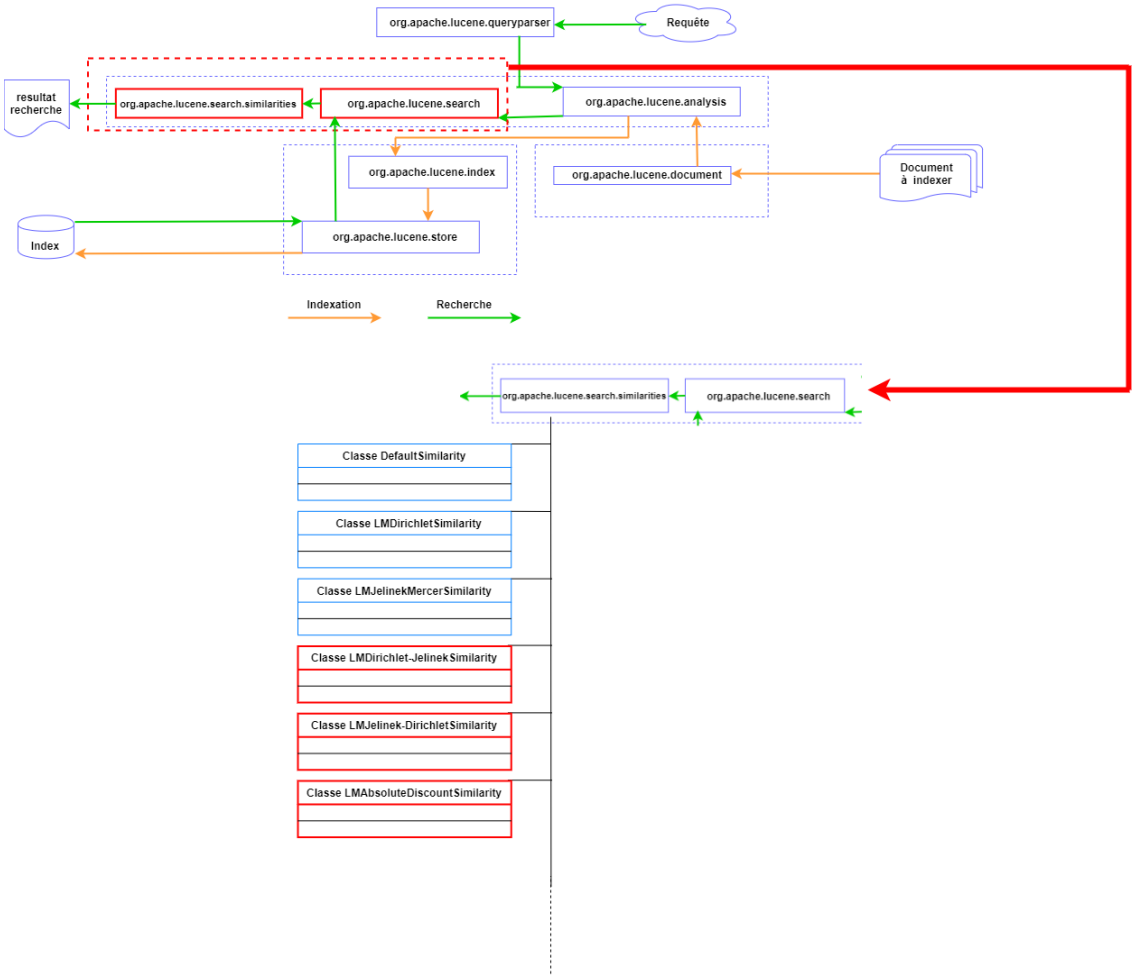


FIGURE 3.4.1 – Schéma de la conception détaillée des approches

Dans L'API Lucene, au niveau du paquetage org.apache.lucene.search on a un sous-paquetage org.apache.lucene.search.similarities qui regroupe toutes les classes de similarité, notre travail consiste à implémenter d'autres classes de similarité qui sont : LMDirichlet_JelinekSimilarity, LMJelinek_DirichletSimilarity et LMAbsoluteDiscountSimilarity, afin d'essayer d'améliorer les résultats de recherche de Lucene et ainsi améliorer leur évaluation.

3.5 Conclusion :

Dans ce chapitre, nous avons pu acquérir de solides connaissance sur Lucene que ça soit sur son architecture, ses API principales et son fonctionnement, par la suite nous avons parlé de nos différentes approches et la façon dont celles-ci vont êtres intégrées dans le modèle de recherche. Dans le chapitre suivant nous allons pouvoir concrètement réaliser leur implémentation et les évaluer à l'aide de l'outil trec-eval.

Chapitre 4

Mise En Œuvre Et Évaluation

4.1 Introduction :

Dans ce chapitre, nous allons tout d'abord présenter les différents outils qui vont être utilisés pour implémenter nos approches proposées précédemment et ensuite passé à l'implémentation de celles-ci, puis à leur évaluation à l'aide d'un ensemble de mesures d'évaluation et enfin comparer ces approches avec d'autres modèles de recherche existants.

4.2 Implémentation :

Ici nous allons détailler la manière dont l'implémentation de nos approches va se faire.

4.2.1 Les outils de développement :

Notre implémentation tourne autour des outils suivants, en plus de l'API Lucene que nous avons précédemment présenté :

4.2.1.1 Le langage JAVA :

C'est un langage de programmation orienté objet, développé par Sun Microsystems. Java est rapide, sécurisé et fiable, il permet de créer des logiciels compatibles avec de nombreux systèmes d'exploitations (Windows, Linux, Macintosh, Solaris).

4.2.1.2 IDE Eclipse :

Eclipse est un environnement de développement (IDE) historiquement destiné au langage Java, même si grâce à un système de plugins il peut également être utilisé avec d'autres langages de programmation, dont le C/C++ et le PHP. Eclipse nécessite une machine virtuelle Java (JRE) pour fonctionner. Mais pour compiler du code Java, un kit de développement (JDK) est indispensable.

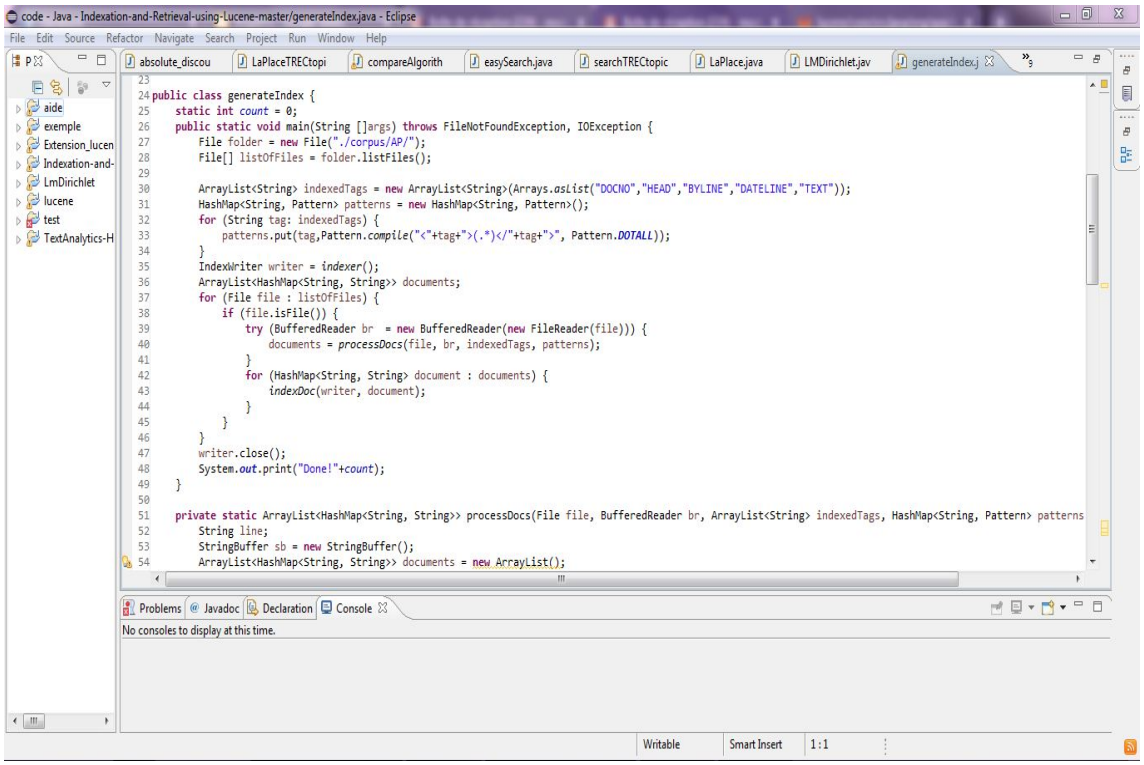


FIGURE 4.2.1 – Interface de l'IDE Eclipse

4.2.2 Implémentation des approches dans Lucene :

Notre objectif étant d'étendre le modèle de langue de Lucene avec d'autres lissages, qui font l'objet de nos approches, et cela en implémentant les classes suivantes :

- La classe LMJelinek_DirichletSimilarity.
- La classe LMDirichlet_JelinekSimilarity.
- la classe LMAbsoluteDiscountSimilarity.

Chacune de ces classes intègre les mêmes méthodes : un constructeur de la classe, une méthode de calcul de similarité (***score()***) à qui Lucene fait appel lors de la phase de recherche, des méthodes d'initialisation des paramètres et une méthode pour retourner le nom de de la similarité (getName()). Chacune d'elle étend la classe LMSimilarity de Lucene.

La Figure 4.2.2 illustre la classe LMJelinek_DirichletSimilarity :

```java
public class LMJelinek_DirichletSimilarity extends LMSimilarity {
private float mu;
private CollectionModel collectionModel ;
private double lamda;
 public void LMJelinek_DirichletSimilarity(CollectionModel collectionModel, float mu , double lamda) {
      this.collectionModel=collectionModel;
      this.mu = mu;
      this.lamda = lamda;
  }
public LMJelinek_DirichletSimilarity(float mu , double lamda) {
       this.mu = mu;
       this.lamda = lamda;
      }
public void LMJelinek_DirichletSimilarity(CollectionModel collectionModel) {
      LMJelinek_DirichletSimilarity(collectionModel,2000,0.7);
  }

 public void LMJelinek_DirichletSimilarity() {
    this.mu=(2000);
    this.lamda = lamda;
  }

  @Override
protected float score(BasicStats stats, float freq, float docLen) {
      float score = stats.getBoost() * (float)(Math.log(1 + ((1-lamda)*freq) /
        (((1-lamda)*mu+lamda*(docLen+mu)) * ((LMStats)stats).getCollectionProbability())) +
             Math.log((((1-lamda)*mu) / (docLen + mu))+lamda));
    return score > 0.0f ? score : 0.0f;
  }
public float getMu() {
    return mu;
  }
public double getLamda() {
       return lamda;
      }

  @Override
  public String getName() {
    return String.format("LMJelinkDirichlet(%f)(%f)", getMu(),getLamda());
  }
```

FIGURE 4.2.2 – Exemple de classe implémentée

Dans ce qui suit nous verrons la méthode score() pour chaque classe utilisant les formules introduites dans le Tableau 3.4.1 du *Chapitre 3* :

— Méthode score pour la classe LMJelinek_DirichletSimilarity :

```java
@Override
protected float score(BasicStats stats, float freq, float docLen) {

  float score = stats.getBoost() * (float)(Math.log(1 + ((1-lamda)*freq) /
      (((1-lamda)*mu+lamda*(docLen+mu)) * ((LMStats)stats).getCollectionProbability())) +
           Math.log((((1-lamda)*mu) / (docLen + mu))+lamda));
  return score > 0.0f ? score : 0.0f;
}
```

FIGURE 4.2.3 – Fonction de score pour la classe LMJelinek_DirichletSimilarity

— Méthode score pour la classe LMDirichlet_JelinekSimilarity :

```java

@Override
protected float score(BasicStats stats, float freq, float docLen) {

  float score = stats.getBoost() * (float)(Math.log(1 + ((1-lamda)*freq) /
      (((docLen*lamda)+mu) * ((LMStats)stats).getCollectionProbability())) +
           Math.log(((docLen*lamda)+mu) / (docLen + mu)));
  return score > 0.0f ? score : 0.0f;
}

```

FIGURE 4.2.4 – Fonction de score pour la classe LMDirichlet_JelinekSimilarity

— Méthode score pour la classe LMAbsoluteDiscountSimilarity :

```
71  @Override
72  protected float score(BasicStats stats, float freq, float docLen) {
73      double max ;
74      if (freq-segma>0)
75        {
76            max = freq-segma;
77
78        }else {
79            max = 0 ;
80        }
81      float score = stats.getBoost() * (float)(Math.log(1 + (max) /
82      ((segma*N) * ((LMStats)stats).getCollectionProbability())) +
83         Math.log((segma*N / (docLen ))));
84
85      return score > 0.0f ? score : 0.0f;
86
87
88
89  }
90
```

FIGURE 4.2.5 – Fonction de score pour la classe LMAbsoluteDiscountSimilarity

Toutes les méthodes scores reçoivent les mêmes paramètres qui sont :

stats : C'est un objet de type BasicStats, qui permet de stocker toutes les statistiques des méthodes de classement couramment utilisés.

freq : la fréquence du terme de la requête dans le document.

docLen : la longueur du document.

Et font appel aux mêmes méthodes qui sont les suivantes :

stats.getBoost() : Permet de récupérer la valeur appliquée (Boost) lors de la recherche du terme t dans une requête.

((LMStats)stats).getCollectionProbability() : Renvoie la probabilité que le terme actuel soit généré par la collection.

4.3 Évaluation :

Après implémentation de nos approches, nous passons à leur évaluation en utilisant l'outil trec-eval.

4.3.1 Protocole :

Pour évaluer nos approches, nous les testons sur une collection de documents TREC AP89, constituée de 84 678 documents, avec 50 requêtes dont des courtes et des longues. À partir des résultats des tests et les jugements de pertinences, nous prenons les 1000 premiers résultats de la recherche selon l'ordre de pertinence et nous les évaluons à l'aide de l'outil trec-eval. Les résultats de leur évaluation seront comparés avec les résultats

d'évaluation des 1000 premiers résultats de la recherche d'autres modèles déjà existants dans Lucene, eux mêmes évalués dans le même environnement(même collection de test) que nos approches.

4.3.2 La collection TREC AP89 :

Nous utilisons dans notre travail une collection qui comprend 84 678 documents, la collection est TREC AP89 (Associated Press newswire, 1989). Les documents sont des articles de journaux de 1989, en anglais. Ils sont structurés en format SGML, avec plusieurs champs (tous les documents ne possèdent pas les mêmes champs). La Figure 4.3.1 illustre un exemple d'un document TREC :

<DOC>
<DOCNO> AP890101-0001 </DOCNO>
<FILEID>AP-NR-01-01-89 2358EST</FILEID>
<FIRST>r a PM-APArts : 60sMovies 01-01 1073</FIRST>
<SECOND>PM-AP Arts : 60s Movies, 1100</SECOND>
<HEAD>You Don't Need a Weatherman To Know '60s Films Are Here</HEAD>
<HEAD>Eds : Also in Monday AMs report.</HEAD>
<BYLINE>By HILLEL ITALIE</BYLINE>
<BYLINE>Associated Press Writer</BYLINE>
<DATELINE>NEW YORK (AP) </DATELINE>
<TEXT>
The celluloid torch has been passed to a new generation : filmmakers who grew up in the 1960s. "Platoon," "Running on Empty," "1969" and "Mississippi Burning" are among the movies released in the past two years from writers and directors who brought their own experiences of that turbulent decade to the screen. "The contemporaries of the '60s are some of the filmmakers of the '80s. It's natural," said Robert Friedman, the senior vice president of worldwide advertising and publicity at Warner Bros. Chris Gerolmo, who wrote the screenplay for "Mississippi Burning," noted that the sheer passage of time has allowed him and others to express their feelings about the decade. "Distance is important," he said. "I believe there's a lot of thinking about that time and America in general."...
</TEXT>
</DOC>

FIGURE 4.3.1 – Exemple d'un document TREC

Dans nos expérimentations, la recherche va s'effectuer sur le champ <TEXT>.

Chaque collection TREC a généralement 50 à 100 requêtes correspondantes, elles aussi sont structurées en format SGML.

Une requête TREC est structurée comme suit : une en-tête, un identifiant de requête unique TREC, un domaine, un titre, une description plus détaillée du besoin en information ...etc. Un exemple d'une requête TREC est montré dans la Figure 4.3.2.

<top>
<head> Tipster Topic Description
<num> Number : 051
<dom> Domain : International Economics
<title> Topic : Airbus Subsidies
<desc> Description :
Document will discuss government assistance to Airbus Industrie, or mention a trade dispute between Airbus and a U.S. aircraft producer over the issue of subsidies.
<smry> Summary :
Document will discuss government assistance to Airbus Industrie, or mention a trade dispute between Airbus and a U.S. aircraft producer over the issue of subsidies.
<narr> Narrative :
A relevant document will cite or discuss assistance to Airbus Industrie by the French, German, British or Spanish government(s), or will discuss a trade dispute between Airbus or the European governments and a U.S. aircraft producer, most likely Boeing Co. or McDonnell Douglas Corp., or the U.S. government, over federal subsidies to Airbus.
<con> Concept(s) :
1. Airbus Industrie
2. European aircraft consortium, Messerschmitt-Boelkow-Blohm GmbH, British Aerospace PLC, Aerospatiale, Construcciones Aeronauticas S.A. 3. federal subsidies, government assistance, aid, loan, financing
...
<def> Definition(s) :
</top>

FIGURE 4.3.2 – Exemple d'une requête TREC

Dans nos expérimentations, nous utilisons 50 requêtes et la recherche est effectuée avec les champs <title> et <desc> qui sont considérées respectivement comme requête courte et requête longue.

Chaque collection a un fichier, appelé "qrels", de jugements de pertinence des documents par rapport aux requêtes. Ce jugement est effectué par des êtres humains qui sélectionnent manuellement les documents qui doivent être récupérés lorsqu'une requête particulière est exécutée. Ce fichier peut être considéré comme la "bonne réponse" et les documents récupérés par le système IR devraient se rapprocher du maximum.

Un jugement de pertinence a le format suivant :

Query-ID 0 Document-ID Pertinence

Query-ID : C'est une séquence alphanumérique pour identifier la requête.

Le champ "0" : N'est pas actuellement utilisé.

Document-ID : C'est une séquence alphanumérique pour identifier le document jugé.

Pertinence : Degré de pertinence entre le document et la requête (0 pour non pertinent et 1 pour pertinent).

La Figure 4.3.3 illustre un fichier "qrels" :

```
51 0 AP891221-0274 0
51 0 AP891225-0071 0
52 0 AP890104-0218 1
52 0 AP890104-0221 1
52 0 AP890105-0245 0
52 0 AP890105-0284 0
52 0 AP890106-0200 1
52 0 AP890109-0116 1
52 0 AP890109-0209 0
52 0 AP890110-0077 1
52 0 AP890113-0170 0
52 0 AP890116-0015 0
52 0 AP890116-0077 1
52 0 AP890116-0127 0
52 0 AP890117-0041 0
52 0 AP890117-0161 1
52 0 AP890118-0251 0
52 0 AP890119-0172 0
52 0 AP890119-0302 0
52 0 AP890119-0349 0
```

FIGURE 4.3.3 – Exemple de jugements de pertinence

4.3.3 L'outil trec-eval :

Le trec_eval est un outil utilisé pour évaluer les classements des documents triés par ordre de pertinence et cela en utilisant des mesures d'évaluation. L'évaluation est basée sur deux fichiers :

- Le premier, appelé "qrels" (jugements de pertinence).
- Le second contient les classements des documents renvoyés par le système *RI*. Ce fichier a le format suivant :

Query-ID Qx Document-ID Rang Score STANDARD

Query-ID : C'est une séquence alphanumérique pour identifier la requête.

Qx : est actuellement ignoré par trec_eval, mais dans notre cas il nous permet de différencier les requêtes courtes des requêtes longues et cela par rapport au x , s'il est pair la requête est considérée comme courte et s'il est impair elle est considérée comme longue.

Document-ID : C'est une séquence alphanumérique pour identifier le document récupéré.

Rang : C'est une valeur entière qui représente la position du document dans le classement, mais ce champ est également ignoré par trec_eval.

Score : Peut être un entier ou une valeur flottante pour indiquer le degré de similitude entre le document et la requête, de sorte que les documents les plus pertinents auront des scores plus élevés.

STANDARD : Est utilisé uniquement pour identifier cette exécution (ce nom est également affiché dans la sortie), on peut utiliser n'importe quelle séquence alphanumérique (dans notre cas on utilise le nom de la similarité).

La commande pour exécuter trec-eval a le format suivant :

```
$ ./trec_eval [-q] [-m mesure] fichier-qrel fichier-résultats
```

trec_eval : est le nom du programme exécutable.

-q : en plus de l'évaluation récapitulative, donne une évaluation pour chaque requête.

-m : affiche uniquement une mesure spécifique ("-m all_trec" montre toutes les mesures, " -m official "est le paramètre par défaut qui n'affiche que les principales mesures).

fichier-qrel : chemin du fichier avec la liste des documents pertinents pour chaque requête.

fichier-résultats : chemin du fichier avec la liste des documents récupérés par le système.

En exécutant trec_eval avec les options par défaut un ensemble de mesures sera affiché. Dans notre travail, les mesures utilisées sont les suivantes :

Resultats TRECEVAL	Ce qu'elle retourne
P@10	Précision des 10 premiers documents
P@30	Précision des 30 premiers documents
P@100	Précision des 100 premiers documents
P@500	Précision des 500 premiers documents
P@1000	Précision des 1000 premiers documents
MAP	La moyenne des précisions moyennes(Mean Average Precision)
MRR	Rang réciproque moyen(Mean Reciprocal Rank)
iprec_at_recall_x	La précision lorsque le rappel = x

TABLE 4.3.1 – Les mesures d'évaluation utilisées

4.3.4 Test et résultats :

Pour cette tâche, nous testons nos algorithmes de recherche LMJelinek_DirichletSimilarity, LMDirichlet_JelinekSimilarity et LMAbsoluteDiscountSimilarity et les comparons avec d'autres modèles déjà existants (LMDirichletSimilarity, LMJelinekSimilarity, DefaultSimilarity(TF-IDF) et BM25Similarity) en suivant le protocole d'évaluation.

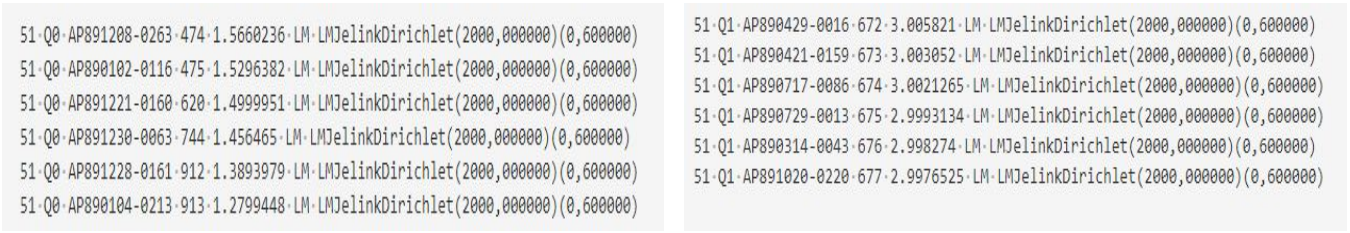
Remarque : au début nous avons implémenté les lissages de façon brute (avec les formules présentées tel-quelles dans le modèle de langue) et l'évaluation, des résultats de classement obtenus, a donné des résultats très mauvais.

Après avoir effectuer plusieurs tentatives avec des valeurs différentes, nous avons constaté que les résultats de l'évaluation sont optimaux pour $\lambda = 0.6$,$\mu = 2000$ et $\delta = 0.6$. Donc durant tout le processus nous utiliseront les valeurs suivantes pour nos approches. Concernant les valeurs de μpour LMDirichletSimilarity , λpour LMJelinekMercerSimilarity et b et $k_1$pour BM25Similarity, nous gardons celles de Lucene par défaut ($\mu = 2000$, $\lambda = 0.7$,$b = 0.75$, $k_1 = 1.2$).

4.3.4.1 Résultats de classement :

Voici quelques résultats obtenus suite à la recherche effectuée sur les requêtes de notre collection :

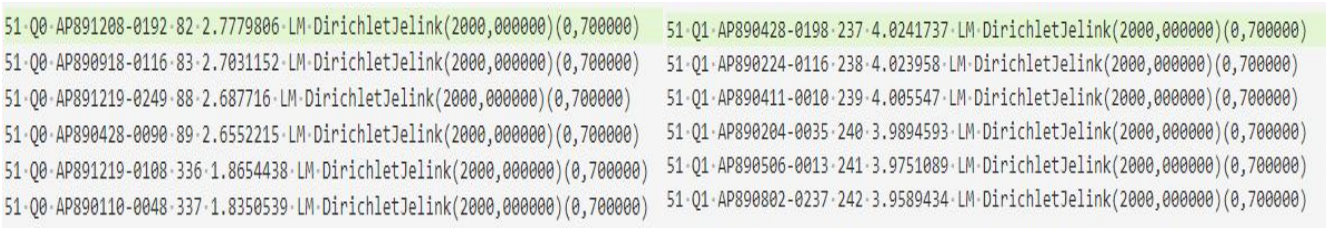
— Pour LMJelinek_DirichletSimilarity :



(a) Pour les requêtes courtes (b) Pour les requêtes longues

FIGURE 4.3.4 – Résultats de classement pour LMJelinek_DirichletSimilarity

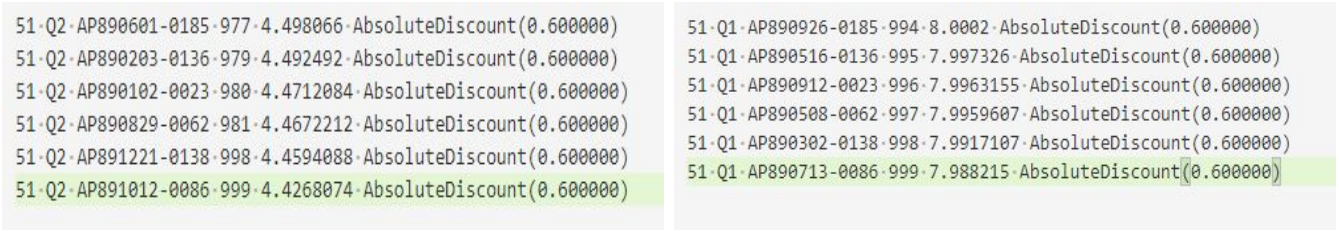
— Résultats de classement pour LMDirichlet_JelinekSimilarity :



(a) Pour les requêtes courtes (b) Pour les requêtes longues

FIGURE 4.3.5 – Résultats de classement pour LMDirichlet_JelinekSimilarity

— Résultats de classement pour LMAbsoluteDiscount :



(a) Pour les requêtes courtes (b) Pour les requêtes longues

FIGURE 4.3.6 – Résultats de classement pour LMAbsoluteDiscount

4.3.4.2 Résultats d'évaluation :

Nous présentons ici les résultats d'évaluation, des classements de documents pour chaque algorithme, pour les requêtes courtes et les requêtes longues avec l'outil trec-eval :

— Pour les requêtes courtes :

Mesures utilisées	Default Similarity	LMDirichlet Similarity	LMJelinek MercerSimilarity	LMJelinek DirichletSimilarity	LMDirichlet JelinekSimilarity	LMAbsolute DiscountSimilarity	BM25 Similarity
P@10	0.2980	0.3300	0.2820	0.3160	0.3160	0.2820	0.3000
P@30	0.2487	0.2680	0.2387	0.2627	0.2627	0.2460	0.2513
P@100	0.1650	0.1704	0.1618	0.1642	0.1642	0.1612	0.1694
P@500	0.0624	0.0642	0.0622	0.0630	0.0630	0.0620	0.0641
P@1000	0.0368	0.0382	0.0365	0.0379	0.0379	0.0363	0.0373
MAP	0.1968	0.2070	0.1944	0.1971	0.1971	0.1879	0.2013
MRR	0.4696	0.4802	0.4637	0.5090	0.5090	0.4474	0.4770
iprec_at_ recall_0.0	0.5051	0.5351	0.5018	0.5469	0.5469	0.4812	0.5171
iprec_at_ recall_0.10	0.3879	0.3977	0.3669	0.3902	0.3902	0.3700	0.3926
iprec_at_ recall_0.20	0.3044	0.3284	0.2955	0.3156	0.3156	0.2839	0.3083
iprec_at_ recall_0.30	0.2510	0.2717	0.2548	0.2535	0.2535	0.2483	0.2609
iprec_at_ recall_0.40	0.2125	0.2312	0.2187	0.2111	0.2111	0.2099	0.2184
iprec_at_ recall_0.50	0.1920	0.2135	0.2030	0.1957	0.1957	0.1933	0.2008
iprec_at_ recall_0.60	0.1609	0.1693	0.1564	0.1555	0.1555	0.1540	0.1672
iprec_at_ recall_0.70	0.1328	0.1344	0.1295	0.1228	0.1228	0.1219	0.1363
iprec_at_ recall_0.80	0.0947	0.0951	0.0919	0.0900	0.0900	0.0881	0.0956
iprec_at_ recall_0.90	0.0616	0.0578	0.0560	0.0567	0.0567	0.0472	0.0563
iprec_at_ recall_1.0	0.0209	0.0244	0.0204	0.0245	0.0245	0.0194	0.0199

TABLE 4.3.2 – Résultats d'évaluation pour requêtes courtes

— Pour les requêtes longues :

Mesures utilisées	Default Similarity	LMDirichlet Similarity	LMJelinek MercerSimilarity	LMJelinek DirichletSimilarity	LMDirichlet JelinekSimilarity	LMAbsolute DiscountSimilarity	BM25 Similarity
P@10	0.2480	0.2420	0.2200	0.2480	0.2480	0.2320	0.3000
P@30	0.2027	0.2173	0.2060	0.2120	0.2120	0.2073	0.2513
P@100	0.1408	0.1454	0.1384	0.1406	0.1406	0.1410	0.1694
P@500	0.0536	0.0560	0.0527	0.0552	0.0552	0.0523	0.0641
P@1000	0.0316	0.0331	0.0317	0.0331	0.0331	0.0316	0.0373
MAP	0.1544	0.1591	0.1524	0.1594	0.1594	0.1535	0.2013
MRR	0.4497	0.3581	0.3661	0.3693	0.3693	0.3776	0.4770
iprec_at_ recall_0.0	0.4667	0.4033	0.4075	0.4188	0.4188	0.4040	0.5171
iprec_at_ recall_0.10	0.3466	0.3067	0.3309	0.3224	0.3224	0.3223	0.3926
iprec_at_ recall_0.20	0.2491	0.2665	0.2594	0.2637	0.2637	0.2560	0.3083
iprec_at_ recall_0.30	0.2134	0.2339	0.2117	0.2348	0.2348	0.2218	0.2609
iprec_at_ recall_0.40	0.1642	0.1813	0.1731	0.1826	0.1826	0.1785	0.2184
iprec_at_ recall_0.50	0.1362	0.1531	0.1417	0.1518	0.1518	0.1424	0.2008
iprec_at_ recall_0.60	0.1101	0.1273	0.1098	0.1285	0.1285	0.1142	0.1672
iprec_at_ recall_0.70	0.0878	0.0932	0.0852	0.0919	0.0919	0.0847	0.1363
iprec_at_ recall_0.80	0.0596	0.0672	0.0618	0.0652	0.0652	0.0591	0.0956
iprec_at_ recall_0.90	0.0361	0.0412	0.0401	0.0426	0.0426	0.0338	0.0563
iprec_at_ recall_1.0	0.0070	0.0138	0.0102	0.0175	0.0175	0.0103	0.0199

TABLE 4.3.3 – Résultats d'évaluation pour requêtes longues

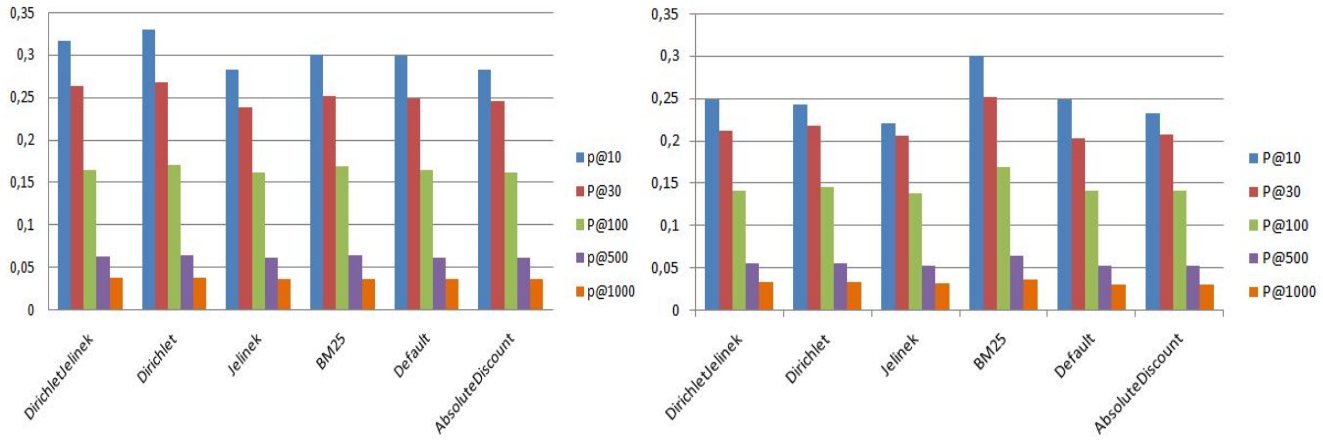
Remarque : Nous constatons que les résultats d'évaluation pour LMDirichlet_JelinekSimilarity et pour LMJelinek_DirichletSimilarity sont les mêmes.

Pour la suite, la comparaison se fera pour un seul algorithme (LMDirichlet_JelinekSimilarity).

4.3.4.3 Comparaison des approches avec les algorithmes existants :

Dans cette partie, nous allons minutieusement analyser les résultats obtenus dans les Tableaux 4.3.2 et 4.3.3.

— Basée sur la précision :



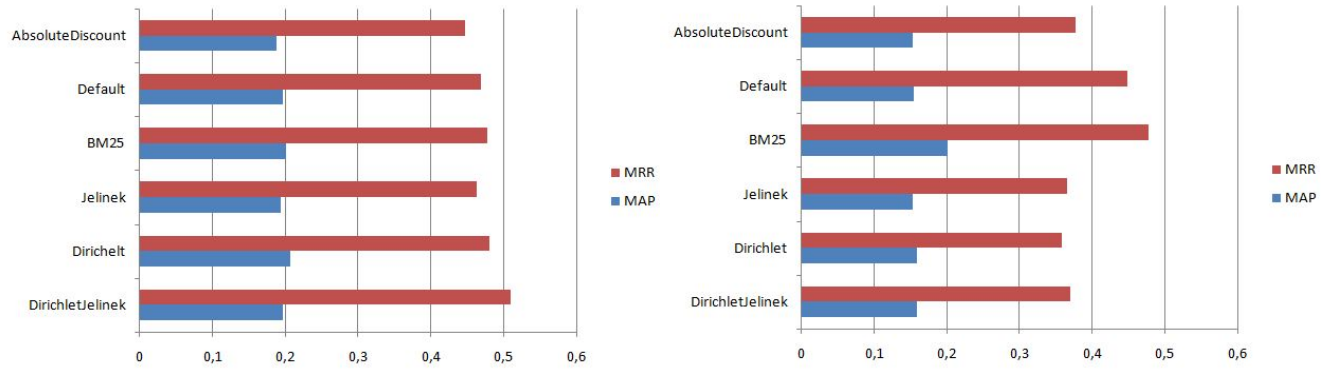
(a) Requêtes courtes (b) Requêtes longues

FIGURE 4.3.7 – Analyse comparative entre les différents algorithmes basée sur la précision

Dirichlet et Dirichlet_Jelinek donnent les meilleurs valeurs de précision pour les requêtes courtes par rapport aux autres algorithmes, tel que Dirichlet est meilleur pour moins de documents et est presque comparable à Dirichlet_Jelinek lorsque nous envisageons plus de documents. Par contre AbsoluteDiscount et Jelinek sont les moins bons.

Le BM25 donne les meilleurs résultats de précision pour les requêtes longues. Pour plus de documents, Dirichlet_Jelinek et Dirichlet donnent d'assez bon résultats.

— Basée sur MAP et MRR :



(a) Requêtes courtes (b) Requêtes longues

FIGURE 4.3.8 – Analyse comparative entre les différent algorithmes basée sur MAP et MRR

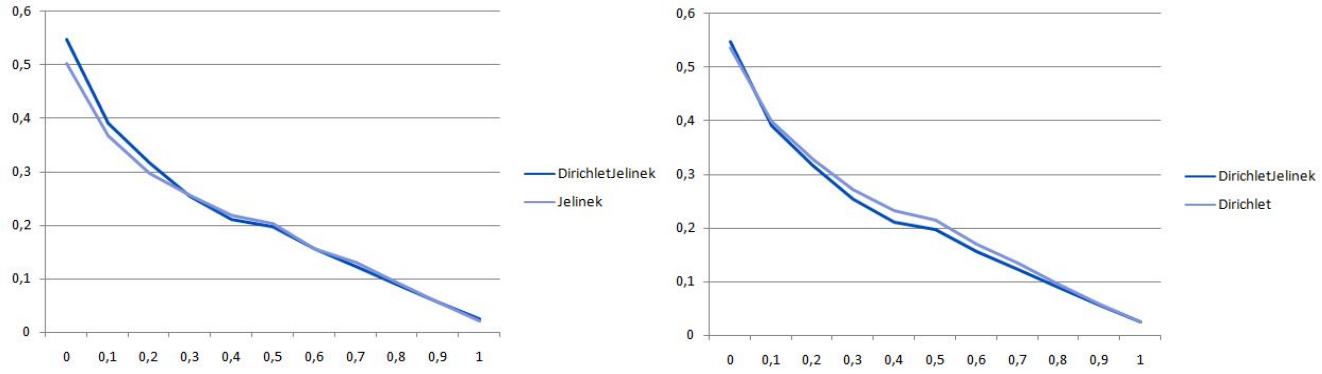
Pour les requêtes courtes, Dirichlet donne un MAP légèrement meilleur, et Dirichlet_Jelinek donne un MRR meilleur comparé aux autres algorithmes.

Pour les requêtes longues, BM25 donne un MAP et un MRR légèrement meilleurs ce qui indique qu'en moyenne, cet algorithme fonctionne mieux que les autres algorithmes.

Donc pour les requêtes courtes, pour Dirichlet, MAP suggère que la précision des résultats de recherche est forte et pour Dirichlet_Jelinek, MRR suggère que le rang auquel le meilleur document est récupéré est fort par rapport aux autres algorithmes, et pour les requêtes longues, pour BM25, MAP suggère que la précision des résultats de recherche est forte et MRR suggère que le rang auquel le meilleur document est récupéré est fort par rapport aux autres algorithmes.

— Basée sur la courbe rappel-précision :

Dans notre approche principale, comme nous l'avons dit, nous avons réaliser une combinaison entre le lissage de Dirichlet et le lissage Jelinek pour avoir au final le lissage Dirichlet_Jelinek. Dans ce qui suit nous allons comparer cette approche avec ces deux lissages à l'aide des résultats observés à partir des courbes rappel-précision :



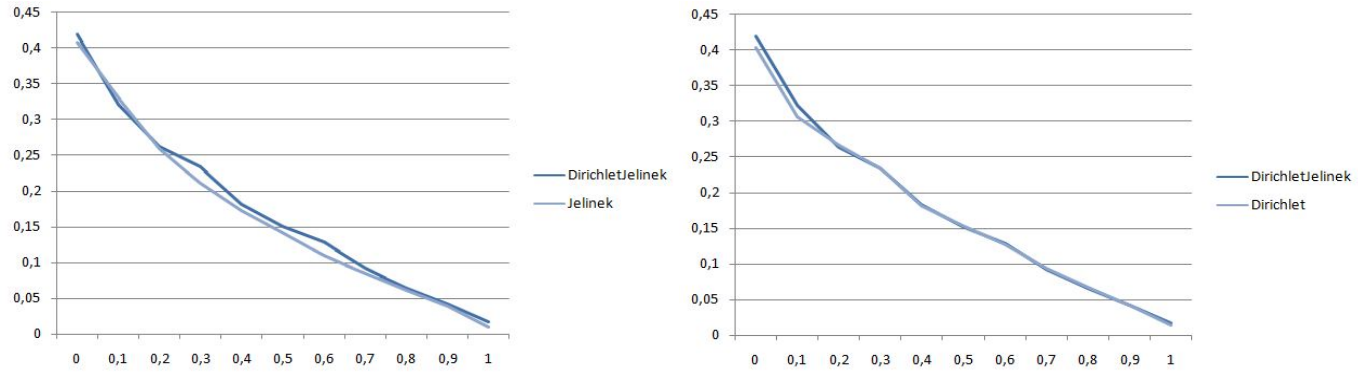
(a) Jelinek et Dirichlet_Jelinek (b) Dirichlet_Jelinek et Dirichlet

FIGURE 4.3.9 – Courbe rappel-précision pour requêtes courtes

Pour les requêtes courtes :

- Nous constatons une légère amélioration du côté de Dirichlet_Jelinek par rapport Jelinek(a).

- Pour le rappel =0 , Dirichlet_Jelinek est meilleur que Dirichlet, et pour la suite des valeurs Dirichlet le surpasse(b).



(a) Jelinek et Dirichlet_Jelinek (b) Dirichlet_Jelinek et Dirichlet

FIGURE 4.3.10 – Courbe rappel-précision pour requêtes longues

Pour les requêtes longues :

- Nous constatons une très légère amélioration de Dirichlet_Jelinek par rapport à Dirichlet(b), cela se voit moins sur la courbe mais plus sur le tableaux des résultats d'évaluation (Tableau 4.3.3).

- Nous observons une amélioration de Dirichlet_Jelinek comparé à Jelinek(a) pour certaines valeurs du rappel.

Au cours de cette évaluation, nous avons analysé les données retournées par trec-eval pour les différents algorithmes, et concernant nos approches, nous avons constaté les choses suivantes :

- Le modèle de langue utilisant le lissage AbsoluteDiscount, qui n'est qu'autre que notre deuxième approche, n'a pas donné de résultats satisfaisants par rapport aux autres algorithmes, par conséquent il n'apporte pas une amélioration .

- Notre algorithme Dirichlet_Jelinek apporte des améliorations pour Dirichlet et Jelinek :
  — Il améliore les performances de Dirichlet par rapport aux requêtes longues, et concernant les requêtes courtes malgré qu'il y ait une légère amélioration, mais Dirichlet reste quand même le meilleur.
  — Pour Jelinek, les résultats sont satisfaisants de telle sorte qu'il y a eu des améliorations pour les requêtes courtes et les requêtes longues. On peut dire que Dirichlet_Jelinek est une bonne solution pour Jelinek.

4.4 Conclusion :

Dans ce chapitre nous avons pu voir les détails d'implémentation de nos approches ainsi que l'ensemble des outils qui ont permis cette implémentation, ensuite nous avons expliqué le protocole utilisé pour l'évaluation et enfin nous avons évalué les résultats des implémentations.

Conclusion Générale

Notre objectif à travers ce projet de fin d'études est d'intégrer de nouvelles approches de lissage au modèle de langue de Lucene. Nous avons ainsi apporté deux principales contributions dans ce sens :

- La première, en proposant une nouvelle approche de lissage basée sur la combinaison de deux approches existantes : le lissage de Dirichlet et le lissage Jelinek-Mercer. Cette nouvelle approche proposée a ensuite été intégrée au modèle de langue de Lucene. Nous avons défini le schéma d'intégration correspondant puis l'avons implémenté dans Lucene.
- La seconde contribution consiste à intégrer une approche de lissage existante (et non implémentée) dans Lucene.

Nos modèles de langue ainsi obtenus ont été testés en recherche d'information, et évalués sur la collection de tests AP89.

Les résultats obtenus par la nouvelle approche que nous avons proposée sont très encourageants, puisque ces derniers sont meilleurs, dans certains cas, que les résultats issus des autres approches implémentées dans Lucene.

En perspective de ce travail, nous envisageons de tester notre approche combinée sur d'autres collections que celle proposée, afin d'avoir une meilleure perception de son fonctionnement et de se prononcer définitivement sur son efficacité.

Bibliographie

[1] Amirouche.F. Modéle de recherche,recherche d'information. *Université UMMTO.*

[2] Ismail Badache.. Recherche d'information sociale : exploitation des signaux sociaux pour améliorer la recherche d'information. 2016.

[3] F. Boubekeur. Contribution à la définition de modèles de recherche d'information flexibles basés sur les cpnets,. 2008.

[4] Kraaij W. Nie J.-Y Boughanem, M. Modèles de langue pour la recherche d'information. les systèmes de recherche d'informations,. *Hermes-Lavoisier,*, pages 163–182.

[5] Fox C. Lexical analysis and stoplists. in : Information retrieval. *Upper Saddle River, NJ : Prentice-Hall ;*, 1992.

[6] Jack Mills Cyril Cleverdon and Michael Keen. Factors determining the performance of indexing systems volume 1. design. cranfield : College of aeronautics,. 1966.

[7] Salton. 83 et al Salton. Introduction to modern information retrieval. 1983.

[8] Salton G. The smart retrieval system : Experiments in automatic document processing. *Englewood Cliffs, NJ : Prentice-Hal*, 1971.

[9] Louis M. Gomez George W. Furnas, Thomas K. Landauer and Susan T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM, 30(11):*, :964–971, 1987.

[10] Arezki Hammache. Recherche d'information : un modèle de langue combinant mots simples et mots composés,. 2013.

[11] Ruas A Sanderson M Sester M Van Kreveld M Weibel R Jones CB, Purves R. Spatial information retrieval and geographical ontologies an overview of the spirit project. in : Proceedings of the 25th annual international acm sigir conference on research and development in information retrieval. *New York, NY : ACM*, p. 387-388., 2002.

[12] C.J. Van Rijsbergen K.S. Jones. Research, and development department. report on the need for and provision of an ideal information retrieval test collection. british library research and development reports. university computer laboratory,. *URL https://books.google.fr/books ?id=cuGnSgAACAAJ.*, 1975.

[13] François C. Lelu A. Information retrieval based on a neural unsupervised extraction of thematic fuzzy clusters. les réseaux neuro-mimétiques et leurs applications (neuro nimes). :93-104., 1992.

[14] Boughanem M. Systèmes de recherche d'information : d'un modèle classique à un modèle connexioniste. *Thèse de doctorat. Toulouse : Université Paul Sabatier,*, 1992.

[15] Thomas Mandl. Recent developments in the evaluation of information retrieval systems : Moving towards diversity and practical relevance. *Informatica.*

[16] Koll MB. Noreault T, McGill M. A performance evaluation of similarity measures, document term weighting schemes and representations in a boolean environment. . *In : Oddy RN, Robertson SE, Van Risjbergen CJ, Williams PW. Proceedings of the 3rd annual ACM conference on Research and development in information retrieval. London : Butterworth Co ;*, p. 57-76., 1981.

[17] Croft W. B Ponte, J.M. . a language modeling approach to information retrieval. proceedings of the 21st annual international acm sigir conference on research and development in information retrieval,. pp. 275- 281, 1998.

[18] Sparck Jones K Robertson SE. Relevance weighting of search terms. *Journal of American Society of Information Sciencec, JASIS*, 129, 1976.

[19] G. Salton. A comparison between manual and automatic indexing methods. journal of american documentation,. pages 61–71„ 1971.

[20] Greco M. Proteus Signore O, Garibald AM. : a concept browsing interface towards conventional information retrieval system database and expert system applications. *In Proceedings of DEXA*, 1992.

[21] Hugo Zaragoza Stephen Robertson. The probabilistic relevance model : Bm25 and beyond. *The 30th Annual International ACM SIGIR Conference*, 23-27, 2007.

[22] Croft WB. Turtle H. Evaluation of an inference network-based retrieval mode. acm transactions on information systems. :187-222., 191.

[23] D.K. Voorhees E.M Voorhees, E.M Harman. Trec : Experiment and evaluation in information retrieval. digital libraries and electronic publishing„. 2005.

[24] E.M. Voorhees. Trec] : Continuing information retrieval's tradition of experimentation. communications of the acm. *50*, pp. 51–54, 2007.

[25] Lafferty J. Zhai, C. A study of smoothing methods for language models applied to ad hoc information retrieval. in w.b. croft, d.j. harper, d.h. kraft, j. zobel (eds.),. *Proceedings of the 24th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval,.*

[26] Rasmussen EM. Zhang J. Developing a new similarity measure from two different perspectives. information processing and management. 279-294, 2001.