République Algérienne Démocratique et Populaire
Ministère de L'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou
Faculté du Génie Electrique et d'Informatique
Département Informatique

Mémoire

de fin d'études

En vue de l'obtention du diplôme de Master en Informatique

Option : Systèmes Informatiques

Thème :

Indexation sémantique latente de documents textuels

Dirigé par :

Mme Amirouche Fatiha

Réalisé par :

Melle Aliouane DYHIA

Promotion 2011-2012

Remerciements

Avant toute personne, je dois remercier Dieu qui ma donné la fois, la santé, la chance, le courage et la volonté pour terminer ce modeste travail.

Je tiens à présenter mes plus vifs remerciements et mes respects à ma promotrice madame Amirouche pour avoir accepté de m'encadrer, pour ses appréciations et ses remarques précieuses et tout le temps qu'elle m'a consacrée.

Je tiens aussi à présenter mes vifs remerciements et mes respects aux membres de jury qui me feront honneur de juger ce modeste travail.

Je ne remercierai jamais assez mes parents pour m'avoir toujours encouragée et m'avoir inculpée le goût du savoir et de l'ambition.

En fin, je remercie tous ceux qui ont contribué, de prés ou de loin, à la réalisation de ce modeste travail.

Je dédie ce modeste travail,

A mes très chers parents, à toute ma famille et à tous ceux que j'aime.

Sommaire

Chapitre3 : Conception

Chapitre4 : Implémentation et Tests

Liste des figures

Introduction générale

Depuis l'essor de l'informatique, le volume d'information stockée électroniquement ne cesse de s'accroitre. Cependant les milliards de documents accessibles sur le web qui composent cette masse d'information ne sont actuellement qu'imparfaitement utilisables. Le problème qui se pose actuellement n'est plus tant la disponibilité de l'information mais la capacité d'accéder et de sélectionner l'information répondant aux besoins précis d'un utilisateur. En effet, pour un individu, rechercher une information précise dans l'amas des données en croissance exponentielle sur le net serait comme chercher une aiguille dans une botte de foin.

Le partage, la gestion et l'exploitation de ces informations nécessitent la mise en place de solutions adaptées. Le développement de modèles, méthodes et outils de recherche efficaces permettant de mettre en relation ces informations, capables d'assister un utilisateur dans sa recherche d'information pour qu'il accède juste à l'information qu'il juge pertinente est devenu un challenge important pour faire du web un réel outil de partage d'informations.

L'objet d'un système de recherche d'information est de faciliter l'accès à un ensemble de documents, afin de permettre à l'utilisateur de retrouver ceux qui sont pertinents, c'est-à-dire ceux dont le contenu correspond le mieux à son besoin en information. Les systèmes de recherche d'information classiques se basent sur une recherche par mots clés, les documents sont représentés comme des sacs de mots et la pertinence d'un document vis-à-vis d'une requête est souvent estimée en s'appuyant sur les fréquences d'apparition des mots de la requête dans ces mêmes documents. Mais très vite les problèmes inhérents à la richesse des langues se sont imposés.

Pour remédier à ce problème, de nouvelles approches d'indexation ont été développées se basant sur la sémantique. L'exploitation de la sémantique permet de tenir compte des relations entre termes.

Parmi ces approches, l'indexation par la sémantique latente résout les sens des mots par un clustering des mots sémantiquement proches via une technique de réduction de la dimensionnalité de la matrice termes-documents.

Notre travail de master s'inscrit principalement dans ce contexte. Notre objectif est alors d'une part d'implémenter l'approche d'indexation sémantique latente et d'autre part d'intégrer le module résultant dans la plate forme de RI Terrier-3.5.

Ce mémoire s'articule autour de quatre chapitres comme suit :

Le premier chapitre présente un état de l'art sur la recherche d'information à travers la diversité des modèles de RI (booléen, algébrique, probabiliste, etc…) et les différents mécanismes sous-jacents à la conception des SRI (indexation, appariement, reformulation, etc…).

Le second chapitre est dédié à la présentation détaillée de l'indexation sémantique latente à travers ses concepts fondamentaux, et des exemples explicatifs. Enfin, quelques approches liées à LSI y sont présentées.

Le troisième chapitre est dédié à la conception de notre système (approche LSI et son intégration dans la plate forme de RI Terrier).

Les résultats et expérimentations de notre approche dans notre plateforme LSI-Terrier seront présentés dans le quatrième chapitre

Nous concluons notre travail et présentons quelques perspectives en fin de mémoire.

Chapitre1:

Recherche d'Information

Concepts de base et principaux modèles existants

1.1 Introduction :

Le monde assiste depuis ces dernières décennies, à une production massive d'informations dans tous les domaines d'intérêt. De multiples directions de recherche ont tenté de mettre en œuvre des processus automatiques d'accès à l'information.

L'objectif est d'exploiter au mieux les bases volumineuses de ces informations.

Les systèmes de recherche d'information(SRI) servent d'interface entre une source (collection d'informations) contenant des quantités considérables de documents et des utilisateurs cherchant, via des requêtes, des informations susceptibles de se trouver dans cette collection.

Les SRI intègrent un ensemble de techniques permettant de sélectionner ces informations. Ces techniques peuvent être résumées en quatre fonctions, qui sont le stockage et l'organisation de l'information, la recherche d'informations en réponse à des requêtes utilisateurs et la restitution des informations pertinentes pour ces requêtes, la dernière fonction est celle qui est visible pour l'utilisateur.

Durant ce chapitre, notre intérêt se porte sur les principes de la recherche d'information(RI), en commençant par un bref historique de la RI, puis nous présenterons les concepts de base de la RI, ensuite nous décrivons les étapes d'un processus de RI. Nous passerons ensuite aux différent modèles de recherche d'informations classique et enfin nous présenterons les plus importantes mesures utilisées pour évaluer un SRI.

1.2 Bref historique de la recherche d'information :

Le nom Recherche d'Information RI (Information Retrieval IR) fut utilisé pour la première fois par Calvin N Mooers en 1948 dans son mémoire de maîtrise. Le premier problème de la RI a été celui de l'indexation des documents [Salton, 71]. Les techniques initiales, comme l'abstraction, l'indexation et l'utilisation des catégories de classification ont marqué la naissance de la Recherche d'Information comme discipline de recherche. La croissance du

volume de données textuelles comme les livres et les articles dans les bibliothèques a imposé, durant des siècles, de définir des mécanismes efficaces pour les localiser.

D'énormes efforts ont été déployés depuis, pour développer des approches et des techniques permettant de retrouver l'information voulue effectivement et efficacement à partir de vastes collections de données textuelles. Depuis les années 1990, notamment avec l'avènement d'Internet, la recherche d'information est devenue plus d'actualité et plus exigeante que jamais [Baziz, 2005].

1.3 Concepts de base :

Définition de recherche d'information :

La recherche d'information (RI) [Rijsbergen, 79], [Grossman & al., 98], [Salton, 71] et [Baeza-Yates & al., 99] est traditionnellement définie comme l'ensemble des techniques permettant de sélectionner à partir d'une collection de documents, ceux qui sont susceptibles de répondre aux besoins de l'utilisateur exprimé via une requête. Nous citons dans ce sens, la définition donnée par [Rijsbergen, 79] dans sa forme originelle :

« The user expresses his information need in the form of a request for information. Information retrieval is concerned with retrieving those documents that are likely to be relevant to his information need as expressed by his request. »

Plusieurs concepts clés ressortent de cette définition:

a. Collection de documents :

La collection de documents (ou le fond documentaire, corpus,…) constitue l'ensemble des informations exploitables et accessibles par le SRI. Elle est constituée d'un ensemble de documents.

b. Document :

Le document constitue l'information élémentaire d'une collection de documents.

Il représente dans un SRI l'élément objectif. Dans le cadre de la RI traditionnelle, c'est du texte libre, qui peut être caractérisé selon trois vues :

- La vue présentation : c'est la mise en forme d'un document texte (entêtes, paragraphes, alignement…) ;
- La vue logique : qui présente la structure logique d'un document, elle porte des informations sur la structure ;
- La vue de contenu : qui se focalise sur le sens ou la sémantique d'un document le plus souvent par un ensemble de mots.

Dans les SRI traditionnels, la vue de contenu est l'unique intérêt, puisque les utilisateurs forment leurs requêtes en se fixant comme objectif le contenu textuel des documents : c'est d'ailleurs la raison même de l'utilisation de tels systèmes.

c. Besoin en informations :

Le besoin en informations est souvent assimilé au besoin de l'utilisateur. Il est exprimé par une requête.

Trois types de besoins utilisateurs ont été définis par [Ingwersen, 92] :

- Besoin vérification : l'utilisateur cherche à vérifier le texte avec les données connues. Il recherche une donnée particulière, et sait souvent comment y accéder (par exemple chercher un document ayant une adresse connue sur le web).
- Besoin thématique connu : l'utilisateur cherche à clarifier, à revoir ou à trouver de nouvelles informations dans un sujet et un domaine connus (il est possible que le besoin s'affine au cours de la recherche).
- Besoin thématique inconnu : l'utilisateur cherche de nouveaux concepts ou de nouvelles relations hors des sujets ou domaines qui lui sont familiers.

d. Requête :

La requête constitue l'expression du besoin en information de l'utilisateur. Elle représente l'interface entre le SRI et l'utilisateur.

Divers types de langages d'interrogation sont proposés dans la littérature. Une requête est un ensemble de mots clés, mais elle peut être exprimée en langage naturel, booléen ou graphique.

e. Pertinence :

La pertinence est une notion fondamentale et cruciale dans le domaine de la RI. Elle est l'objet de tout système de recherche d'information.

Définir cette notion complexe n'est pas simple car elle est une notion vague, à voir le nombre de définitions autour d'elle.

[Saracevic, 70] les a regroupées dans un ensemble, nous citons quelques unes :

- La correspondance entre un document et une requête, une mesure d'informativité ;
- Un degré de relation (chevauchement) entre le document et la requête ;
- Un degré de la surprise qu'apporte un document, en rapport avec le besoin de l'utilisateur.
- …etc.

Analyser ces définitions nous amène à remarquer que la pertinence est difficile à automatiser, car elle est fortement subjective, c'est-à-dire dépendante de l'utilisateur.

Les travaux menés autour de la notion de pertinence ont montré qu'elle n'est pas une relation isolée entre un document et une requête, elle fait appel aussi au contexte de jugement, ainsi qu'à l'utilisateur lui même. [Denos, 97] fait remarquer qu'il existe une distance plus au moins grande entre les résultats d'un système de RI et les jugements de pertinence de l'utilisateur.

On distingue la pertinence système de la pertinence utilisateur :

- La pertinence système : c'est l'ensemble des principes qui sous-tendent la fonction de correspondance dans un système de recherche d'information. c.-à-d. c'est l'évaluation par le SRI, de l'équivalence entre des documents et une requête.

- La pertinence utilisateur : quant à elle, correspond à l'ensemble des jugements de pertinence que produit l'utilisateur du système de RI.

Le but de SRI est alors de faire correspondre au mieux la pertinence système avec la pertinence utilisateur.

1.4 Processus de recherche d'information :

Pour répondre aux besoins en information de l'utilisateur, un SRI met en œuvre un processus pour réaliser la mise en correspondance des informations contenues dans un fonds documentaire d'une part, et les besoins en information des utilisateurs d'autre part. Un SRI est défini par ses modèles de représentation des documents, des besoins de l'utilisateur et sa fonction d'appariement.

Ce processus est composé de trois fonctions principales:

- L'indexation des documents et des requêtes:
  L'indexation a pour rôle d'extraire à partir d'un document ou d'une requête, une représentation paramétrée qui couvre au mieux son contenu sémantique.
- L'appariement document-requête ou l'interrogation :
  La comparaison entre le document et la requête revient à calculer un score, supposé représenter la pertinence du document vis-à-vis de la requête.
- La reformulation de la requête :
  Qui intervient suite à une itération de recherche et consiste à modifier les requêtes en fonction des résultats présentés et le jugement de l'utilisateur.

Le processus général d'un SRI consiste à représenter le besoin en information, et en parallèle à collecter des documents et les représenter, à déterminer l'appariement entre chaque représentation interne d'un document et celle de la requête puis à décider si le document est pertinent. Les documents pertinents sont alors retournés à l'utilisateur dans une liste ordonnée par ordre décroissant de degré de pertinence. 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.

Le fonctionnement générale d'un SRI est donné à travers le processus de recherche communément appelé processus en U, présenté en figure1.1.

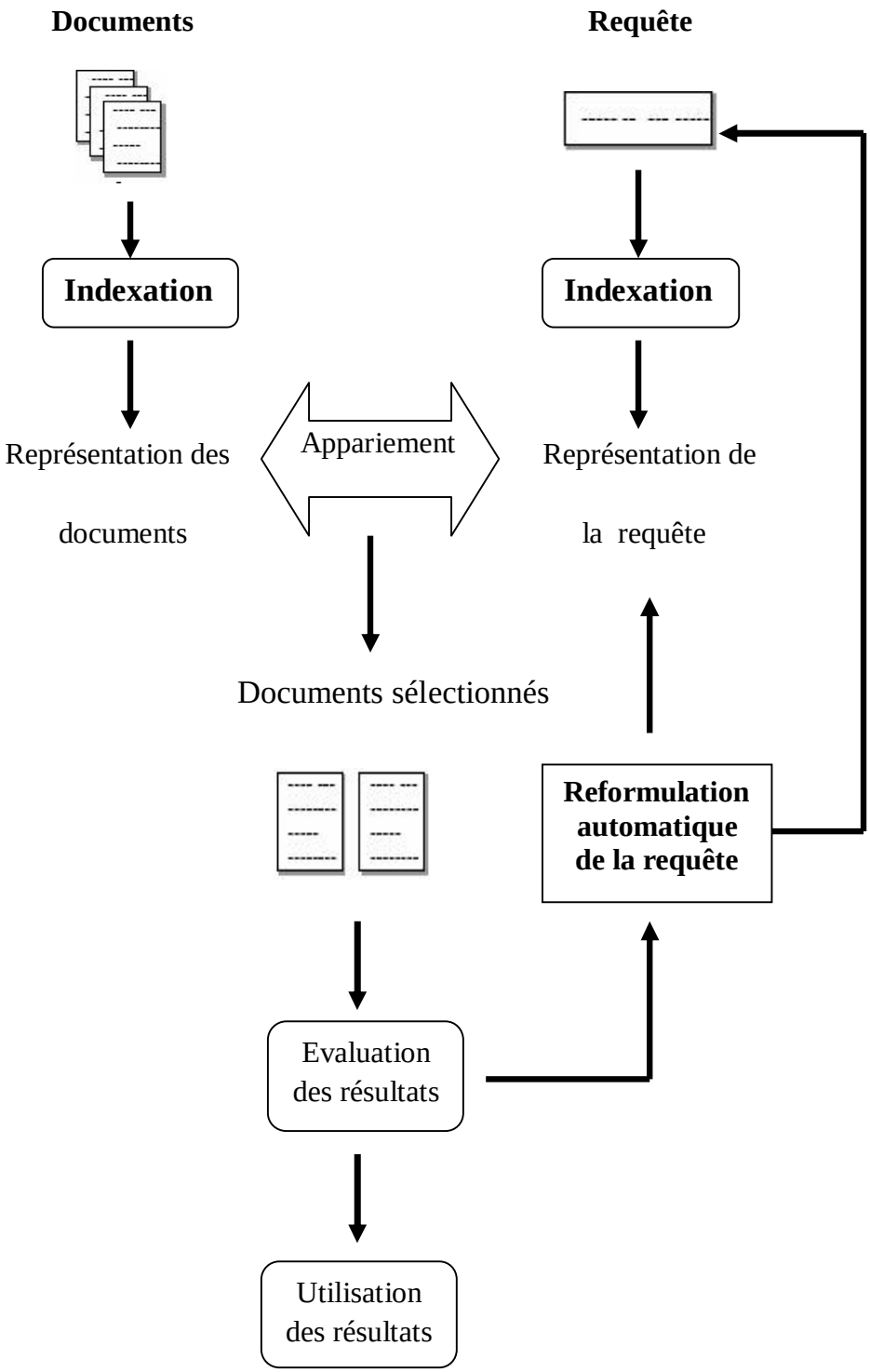


Figure 1.1 : Processus en U de la RI

1.4.1 Indexation :

1.4.1.1 Définition de l'indexation :

L'indexation est une étape très importante dans le processus de RI. Elle consiste à déterminer et à extraire les termes représentatifs du contenu d'un document ou d'une requête. La qualité de la recherche dépend en grande partie de la qualité de l'indexation. Le résultat de l'indexation constitue, ce que l'on nomme le descripteur du document ou de la requête.

L'indexation consiste donc à représenter le document par un ensemble de mots clés qui résument son contenu d'une manière intelligente, permettant ainsi de le retrouver facilement et rapidement. Les mots clés choisis pour indexer les documents sont dits termes d'indexation. Les termes d'indexation doivent en théorie permettre de séparer les documents pertinents des documents non pertinents pour une requête donnée.

Le résultat de l'indexation est un ensemble de termes définissant ce qui est appelé le langage d'indexation.

1.4.1.2 Définition d'un langage d'indexation :

L'ensemble de tous les termes d'indexation constitue le langage d'indexation. Le langage d'indexation exprime le contenu sémantique des documents. Il doit offrir un compromis entre la compacité de la représentation, pour qu'elle puisse être traitée efficacement par un système informatique, et l'expressivité afin d'exprimer aussi fidèlement que possible le contenu des documents.

On peut distinguer deux types de langage d'indexation : le langage contrôlé et le langage libre.

- **Langage contrôlé** : il s'agit d'un lexique figé de descripteurs. L'indexation est alors le plus souvent manuelle, parfois semi-automatique. Un professionnel choisit un ensemble de descripteurs, définis et organisés généralement dans des thésaurus[1] ou des bases terminologiques, pour représenter le document.
- **Langage libre** : Les descripteurs sont extraits automatiquement des documents, ou de la requête de l'utilisateur.

[Gaussier & al., 2003] soulignent que ces deux types de langage d'indexation peuvent coexister dans certains SRI.

1.4.1.3 Types d'indexation :

Techniquement, l'indexation peut être manuelle, automatique ou semi-automatique :

- **Indexation manuelle :** Chaque document est analysé par un spécialiste du domaine ou un documentaliste. Le spécialiste se charge de recenser selon ses connaissances propres, les concepts dont traite un document et à les représenter à l'aide d'un langage documentaire libre ou contrôlé. [Maniez, 2002 ; Lefèvre, 2000]

[1] C'est un vocabulaire contrôlé et dynamique de termes, obéissant à des règles terminologiques propres et reliés entre eux par des relations sémantiques.

Ce mode à l'avantage d'assurer une meilleure correspondance entre les documents et les termes d'indexation choisis, ce qui garantit une bonne précision mais exige un gros effort intellectuel. L'indexation manuelle est caractérisée par un haut degré de subjectivité du fait qu'elle soit réalisée par un être humain. En conséquence, pour un même document, des termes différents peuvent être affectés par des indexeurs différents.

- **Indexation automatique:** C'est un processus complètement automatisé qui se charge d'extraire les termes caractéristiques du document. L'intérêt d'une telle approche réside dans sa capacité à traiter les textes nettement plus rapidement que l'approche précédente, et de ce fait, elle est particulièrement adaptée aux corpus volumineux. **[**Luhn, 57; Maron, 60; Salton, 68]

  L'indexation automatique est l'approche la plus étudiée en RI, nous la détaillons en section suivante.

- **Indexation semi-automatique:** Dans ce mode, un premier processus automatique permet d'extraire les termes du document. Cependant, le choix final reste au spécialiste du domaine ou au documentaliste pour établir les relations entre les mots clés et choisir les termes significatifs à partir de thésaurus ou d'ontologie.

Dans la section suivante, nous nous intéressons particulièrement à l'approche d'indexation automatique, plus répandue, puisque c'est celle qui nous intéresse dans le cadre de notre travail.

1.4.1.4 Indexation automatique :

L'indexation automatique classique est fondée sur l'analyse des documents en vue de l'extraction des termes (mots-clés simples ou composés) représentatifs de leur contenu informationnel.

Elle repose sur les étapes suivantes : l'identification des termes d'indexation, la normalisation des termes d'indexation et la pondération des termes d'indexation.

Ø Identification des termes d'indexation :

L'objectif est de trouver les mots qui représentent au mieux le contenu d'un document. Cette étape comprend :

v L'extraction des termes d'index :

C'est une étape qui peut sembler triviale au premier abord, et qui pourtant constituera la base de tout le reste du processus d'indexation. Il faut donc que cette phase soit d'une qualité maximale. Elle se base sur une analyse morphologique de texte à l'issus de laquelle, les mots du document sont identifiés.

v La réduction du langage d'indexation :

Elle vise à réduire le nombre de termes d'indexation en éliminant tous les mots non importants (mots rares ou mots trop fréquents) du langage d'indexation.

Parmi ces mots, les mots vides qui sont des mots peu significatifs et porteurs de peu de sens, augmentant ainsi la taille de l'index et rendant la recherche plus lente. De ce fait, leur élimination est impérative. Les mots vides sont éliminés en se référant à une

liste prédéfinie de mots dite *stoplist* (ou anti-dictionnaire). Lorsqu'on rencontre un mot dans un texte, on vérifie s'il appartient à la liste stoplist. Si c'est le cas le terme est éliminé de l'index, sinon on le considère comme un terme d'index.

Pour mesurer l'importance d'un mot dans un document, l'indexation s'appuie sur la fréquence d'occurrence de ce mot dans le document. Les mots de fréquences quasi nulles et les mots à fréquences trop élevées peuvent être éliminés de l'index. Cette hypothèse tire ses origines de la conjecture de Luhn [Luhn, 58] qui, pratiquement, définit un seuil de fréquence minimal Smin et un seuil de fréquence maximal Smax tels que, tout terme d'indexation t de fréquence intermédiaire (Smin ≤ freq(t) ≤ Smax), est considéré comme significatif et appartient donc au langage d'indexation.

Ø **Normalisation des termes d'indexation :**

Ce traitement consiste à retrouver pour un mot sa forme normalisée (le masculin pour les noms, l'infinitif pour les verbes, etc…). Ainsi, dans l'index ne sont conservées que les formes normalisées des mots, ce qui offre un gain de place appréciable, mais surtout, une recherche efficace.

Le traitement associé à la normalisation repose sur deux procédures : la lemmatisation et la troncature.

v **La troncature :**

La troncature est utilisée pour éliminer les variations morphologiques ou les variantes orthographiques de mots clés de l'index. Elle consiste à couper le mot à partir d'un rang précis, afin d'obtenir son radical. Pour la langue française, la troncature à 7 caractères est adoptée.

v **La lemmatisation :**

La lemmatisation est l'opération qui consiste à réduire les formes fléchies des mots à leurs racines grammaticales. Cette technique, permet de regrouper les termes ayant des formes légèrement différentes et des sens similaires, notamment les mots conjugués.

Les approches de lemmatisation se basent sur l'élimination des terminaisons des mots, soit en examinant simplement la forme du mot, ou en se basant sur un dictionnaire.

Des expériences ont montré que la troncature et la lemmatisation améliorent significativement les performances pour les langues riches morphologiquement, (ex. le français, l'italien, etc.) [Gaussier & al., 97] et [Gaussier & al., 2000].

Ø **Pondération des termes d'indexation**:

La pondération consiste à associer un poids d'importance (ou valeur de représentativité) w_{ij} à chaque terme t_j d'un document d_i. De manière générale, les formules de pondération utilisées sont basées sur la combinaison d'un facteur de pondération local quantifiant la représentativité locale du terme dans le document, et d'un facteur de pondération global quantifiant la représentativité globale du terme vis-à-vis de la collection de documents. Plusieurs formules existent, dont :

$$w_{ij} = \frac{\mathrm{tf_{ij}}}{\mathrm{df_j}} = tf_{ij} \times \frac{1}{df_j} = tf_{ij} \times idf_j \qquad \text{[Salton \& al., 73]}$$

Où :

- $\mathrm{tf_{ij}}$ est la fréquence d'occurrences du terme t_j dans le document d_i.
- $\mathrm{df_j}$ est la fréquence documentaire du terme t_j (i.e. la proportion de documents de la collection qui contiennent t_j) et idf_j sa fréquence documentaire inverse.

La mesure $tf* idf$ est une bonne approximation de l'importance d'un terme dans un document, particulièrement dans des corpus de documents de tailles intermédiaires. Pour des documents plus longs des normalisations ont été proposées, dont :

- La normalisation pivotée de Singhal [Singhal & al., 96] :

$$\mathrm{w_{ij}} = \frac{\mathrm{tf_{ij}} * \mathrm{idf_j}}{1 + \frac{\mathrm{slope}}{(1-\mathrm{slope}) * \mathrm{pivot}} * \sqrt{\sum_j (\mathrm{tf_{ij}} * \mathrm{idf_j})^2}}$$

Où :

- $\mathrm{tf_{ij}}$ est le nombre d'occurrences du terme $\mathrm{t_j}$ dans l'unité documentaire $\mathrm{d_i}$.
- $\mathrm{idf_j}$ est la fréquence documentaire inverse définie classiquement par : $\log(\mathrm{n}/\mathrm{N_j})$ tel que n est le nombre de documents de la collection et $\mathrm{N_j}$ le nombre de documents indexés par le terme $\mathrm{t_j}$.
- pivot est une constante qui représente l'écart nul entre la probabilité de pertinence et la probabilité de sélection des documents.
- slope est un facteur de normalisation fixé empiriquement, de sorte à minimiser l'écart entre la pertinence et la sélection.

- La formule de Robertson [Robertson & al., 97]

$$\mathrm{w_{ij}} = \frac{\mathrm{tf_{ij}} * (\mathrm{K_1} + 1)}{\mathrm{K_1}\left[(1-\mathrm{b}) + \mathrm{b} * \frac{\mathrm{dl_i}}{\Delta \mathrm{l}}\right] + \mathrm{tf_{ij}}}$$

Où :

- $\mathrm{w_{ij}}$ est le poids du terme $\mathrm{t_j}$ dans le document $\mathrm{d_i}$.
- $\mathrm{K_1}$ constante qui permet de contrôler l'influence de la fréquence du terme $\mathrm{t_j}$ dans le document $\mathrm{d_i}$. Sa valeur dépend de la longueur des documents dans la collection. Le plus souvent, sa valeur est fixée à 1,2.

- ü b constante qui permet de contrôler l'effet de la longueur du document. Sa valeur la plus souvent utilisée est : 0,75.
- ü dl_i est la longueur du document d_i.
- ü Δl est la longueur moyenne des documents dans la collection entière.

1.4.2 Reformulation de requête :

Parfois l'utilisateur est confronté à une situation difficile, il est incapable de trouver les mots précis pour exprimer son besoin en information. Alors, certains pour ne pas dire la majorité des documents qui lui sont retournés l'intéressent moins que d'autres.

Outre cela, certaines requêtes sont courtes, du coup, elles ne sont pas sémantiquement riches, pour que le SRI puisse retourner des documents pertinents [Bai & al., 2006]. Afin de pallier ces problèmes, les chercheurs en RI se sont orientés vers l'intégration d'une étape supplémentaire dans le processus de recherche : la reformulation ou l'expansion de requêtes.

Elle consiste à modifier la requête initiale de l'utilisateur par l'ajout de termes significatifs et/ou la ré-estimation de leur poids.

Si les termes rajoutés proviennent des documents de la collection, on parle de réinjection de pertinence (relevance feedback). En revanche, s'ils sont issus d'une ressource conceptuelle externe (ontologie, thésaurus ou dictionnaire) on parle, dans ce cas de reformulation de requêtes directe. [Khan, 2000] [Baziz & al., 2003] [Voorhees, 93] [Järvelin & al., 2004] et [Bhogal & al., 2006] donnent un état de l'art complet de l'expansion des requêtes ainsi que des différentes approches suivies dans cet axe.

1.4.3 Appariement document-requête :

Le processus d'appariement document-requête permet de mesurer la pertinence d'un document vis-à-vis d'une requête. De manière générale, pour chaque requête, le SRI calcule un score de pertinence (similarité vectorielle, probabiliste, etc.). Ce score de pertinence est calculé à partir d'une fonction ou d'une mesure de similitude, notée RSV (Q;D) (Retrieval Status Value) où Q est une requête et D un document de la collection. Le processus d'appariement est étroitement lié au processus d'indexation et de pondération des termes. Il existe deux méthodes d'appariement :

- **Appariement exact** (« exact match retrieval ») : Le résultat est une liste de documents respectant exactement la requête spécifiée avec des critères précis. Les documents retournés ne sont pas triés [Salton & al., 71].
- **Appariement approché** («best match retrieval ») : Le résultat est une liste de documents sensés être pertinents pour la requête. Les documents retournés sont triés selon leur score de pertinence vis-à-vis de la requête [Robertson & al., 76]

1.5 Taxonomie des modèles de recherche d'information :

Les travaux de recherche dans le domaine de la RI ont conduit à la proposition de nombreux modèles [Baez-Yates & al., 99]. Un modèle de RI a pour rôle de fournir une formalisation du processus de RI et un cadre théorique pour la modélisation de la mesure de pertinence. Il modélise la fonction d'appariement qui joue un rôle central dans la recherche d'information. C'est donc le modèle qui détermine le comportement clé d'un SRI car la première fonction d'un SRI est de mesurer la pertinence d'un document vis-à-vis une requête.

Les modèles de RI se déclinent en trois grandes catégories: les modèles reposant sur la théorie des ensembles (les modèles booléens), les modèles basés sur l'algèbre (modèles vectoriels) et les modèles probabilistes.

La figure 1.2 présente une classification des différents modèles de RI:

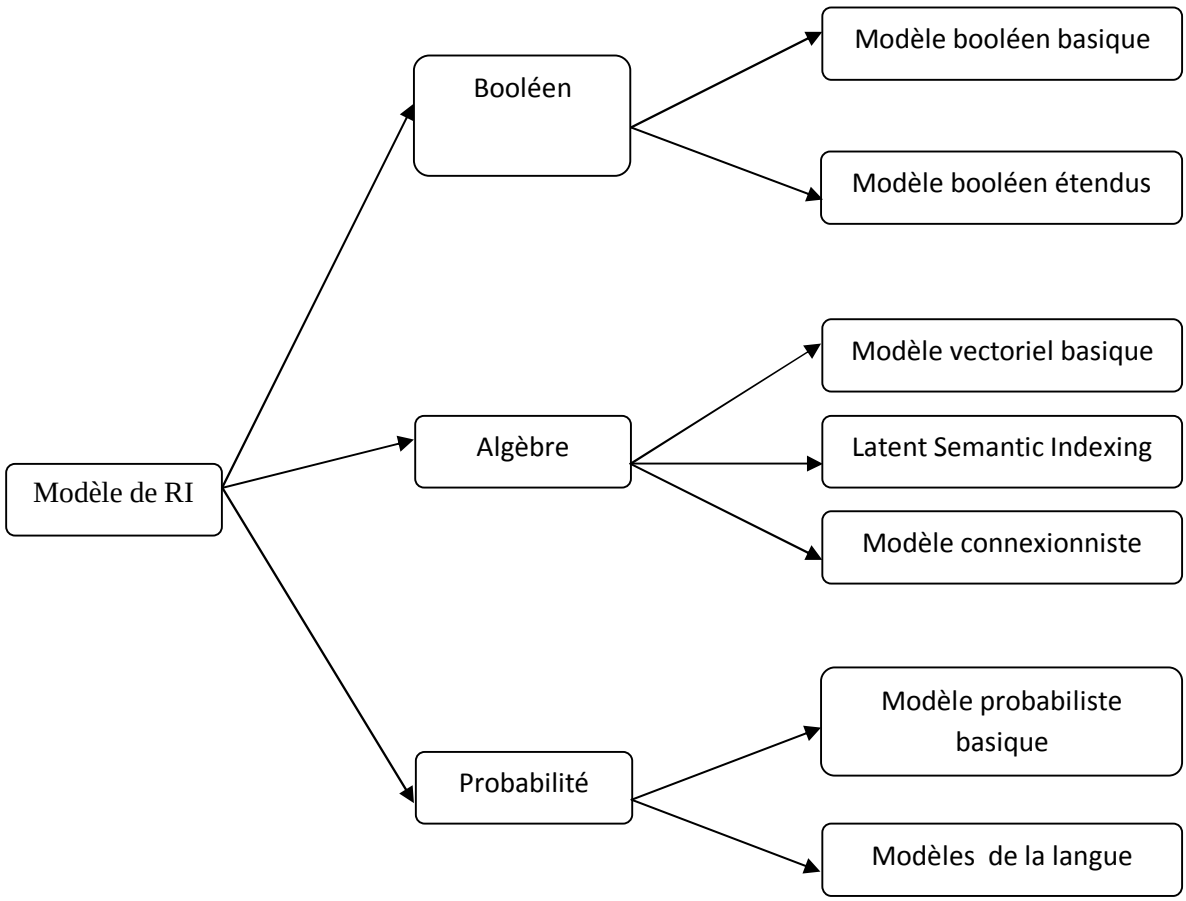


Figure 1.2 : Taxonomie des modèles en RI.

Dans ce qui suit, nous décrivons pour chacune ces catégories, le modèle de base et quelques modèles associés :

1.5.1 Le modèle booléen [Salton & al., 71] :

Ce modèle propose la représentation d'une requête sous forme d'une expression logique. Les termes d'indexation sont reliés par les connecteurs logiques *ET*, *OU* et *NON*. Le processus de recherche mis en œuvre, consiste à effectuer des opérations sur les ensembles de documents définis par la présence et l'absence de termes d'indexation, afin de réaliser un appariement exact avec l'équation de la requête. Une extension de ce modèle a été effectuée par [Salton & al., 83]: le modèle booléen étendu. Ce dernier intègre des poids dans l'expression de la requête et des documents. La sélection des documents s'effectuera donc sur la base d'un appariement rapproché et non plus exact.

a. Le modèle booléen de base :

Le modèle booléen est le plus ancien de tous les modèles de RI conventionnels. Les premiers systèmes de gestion de bibliothèques sont des exemples classiques utilisant ce modèle.

Le modèle booléen de base propose la représentation d'une requête sous forme d'une expression logique. Les termes d'indexation sont reliés par les connecteurs logiques ET ($\wedge$), OU($\vee$) et NON($\neg$).

La correspondance RSV (d_j,q_k), entre une requête $q_k$et un document d_j est déterminée comme suit :

RSV (d_j ,q_i)=1 si $q_i \in d_j$; 0 sinon,

RSV (d_j, $q_i \wedge q_j$)=1 si RSV (d_j, q_i)=1 et RSV(d_j, q_j)=1 ; 0 sinon,

RSV (d_j, $q_i \vee q_j$)=1 si RSV (d_j, q_i)=1 ou RSV(d_j, q_j)=1 ; 0 sinon,

RSV (d_j, $\neg q_i$)=1 si RSV (d_j, q_i)=0 ; 0 sinon,

Sachant que $q_i$est un terme de la requête q_k.

Ce modèle se caractérise par sa simplicité et son caractère intuitif. Il est reconnu pour sa force à faire une recherche très restrictive et obtenir, pour un utilisateur expérimenté, une information exacte et spécifique.

Son inconvénient majeur est que les documents pertinents dont la représentation ne correspond qu'approximativement à la requête ne sont pas sélectionnés. En plus, il n'y a pas de pondération des termes, ainsi, les résultats de la recherche ne peuvent pas être ordonnés. Pour remédier à ces inconvénients, le modèle booléen étendu a été développé.

b. Le modèle booléen étendu : (modèle P-norme)

Le modèle booléen étendu a été introduit par Salton [Salton & al., 83]. C'est une extension du modèle précédent qui vise à tenir compte d'une pondération des termes dans le corpus. Cela permet de pallier aux problèmes du modèle de base en ordonnant les documents retrouvés par le SRI. La requête demeure une expression booléenne classique.

Tandis que les termes d'un document sont maintenant pondérés. En général le poids d'un terme dans un document est fonction du nombre d'occurrences de ce terme dans le document. L'appariement requête-document est le plus souvent déterminé par les relations introduites dans le modèle p-norm basées sur les p-distances, avec $1 \leq p \leq \infty$. La valeur de p est indiquée au moment de la requête. Si m est le nombre de termes dans la requête, les fonctions de similarité se calculent comme suit :

- RSV (d,q_{ou}) = $\left[\frac{x_1^p + x_2^p + \ldots + x_m^p}{m}\right]^{\frac{1}{p}}$

- RSV (d,q_{et}) = 1 – $\left[\frac{(1-x_1)^p + (1-x_2)^p + \ldots + (1-x_m)^p}{m}\right]^{\frac{1}{p}}$

Si p = 1, on se ramène au modèle booléen de base.

1.5.2 Les modèles algébriques :

Les modèles algébriques proposent une représentation vectorielle pour le document et la requête. La mise en correspondance entre le document et la requête consiste à calculer la similarité entre les vecteurs les représentant.

Le modèle vectoriel est l'ancêtre de tous les modèles algébriques. Les premiers travaux de Salton [Salton, 91] avaient pour finalité de concevoir la fonction d'appariement selon les propriétés et les opérations associées au concept d'espace vectoriel. Bien que simpliste, ce modèle reste un des plus utilisés et des plus efficaces.

a. Le modèle vectoriel de base :

Dans ce modèle, un document est représenté sous forme d'un vecteur dans l'espace vectoriel composé de tous les termes d'indexation. 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,

$$d_i = (w_{i1}, w_{i2}, \ldots, w_{in}) \quad \text{pour i = 1, 2, …, m.}$$

Où :

- **ü** w_{ij} est le poids du terme t_j dans le document ld_i.
- **ü** m est le nombre de documents dans la collection.
- **ü** n est le nombre de termes d'indexation.

Une requête q est aussi représentée par un vecteur de mots-clés défini dans le même espace vectoriel que le document.

$$q = (w_{q1}, w_{q2}, \ldots, w_{qn})$$

Où :

- **ü** w_{qj} est le poids de terme t_j dans la requête q. Ce poids peut être soit une forme de $tf * idf$, soit un poids attribué manuellement par l'utilisateur.

La pertinence du document d_i pour la requête q est mesurée comme le degré de corrélation des vecteurs correspondants. Cette corrélation peut être exprimée par l'une des mesures suivantes :

- Le produit scalaire :

$$Sim(d_i, q) = \sum_{j=1}^{n} w_{qj} * w_{ij}$$

- La mesure du cosinus:

$$Sim(d_i, q) = \frac{\sum_{j=1}^{n} w_{qj} * w_{ij}}{\left(\sum_{j=1}^{n} w_{qj}^2\right)^{\frac{1}{2}} * \left(\sum_{j=1}^{n} w_{ij}^2\right)^{\frac{1}{2}}}$$

- La mesure de Dice :

$$Sim(d_i, q) = \frac{2 * \sum_{j=1}^{n} w_{ij} * w_{qj}}{\sum_{j=1}^{n} w_{qj}^2 + \sum_{j=1}^{n} w_{ij}^2}$$

- La mesure de Jacard :

$$Sim(d_i, q) = \frac{\sum_{j=1}^{n} w_{ij} * w_{qj}}{\sum_{j=1}^{n} w_{qj}^2 + \sum_{j=1}^{n} w_{ij}^2 - \sum_{j=1}^{n} w_{ij} * w_{qj}}$$

- Le coefficient de superposition :

$$Sim(d_i, q) = \frac{\sum_{j=1}^{n} w_{ij} * w_{qj}}{\min_i\left(\sum_{j=1}^{n} w_{qj}^2,\ \sum_{j=1}^{n} w_{ij}^2\right)}$$

L'un des avantages du modèle vectoriel réside dans sa simplicité conceptuelle et de mise en œuvre. En outre, il permet de trier les résultats d'une recherche à travers une mesure de similarité document/requête, en plaçant en tête les documents jugés les plus similaires à la requête. Cependant, ce modèle ne permet pas de modéliser les associations entre les termes d'indexation. Chacun des termes est considéré comme indépendant des autres. Le modèle vectoriel généralisé (Generalized Vector Space Model) [Wong & al., 85] permet cependant de résoudre le problème d'indépendance des termes.

b. Latent Semantic Indexing :

L'objectif du modèle LSI est de construire des index conceptuels portant sur la sémantique des mots dans les documents. Ces index sont tirés à partir de la structure sémantique latente des textes des documents. Pour ce faire, partant de l'espace vectoriel de tous les termes d'indexation, le modèle LSI construit un espace d'indexation de taille réduite *k*, par application de la décomposition en valeurs singulières (SVD) de la matrice termes-documents [Deerwester & al., 90]. Ces *k* dimensions capturent une partie importante de la structure sémantique des documents [Berry & al., 94] portée par les associations des termes et documents, et éliminent le bruit dû à la variabilité dans l'usage des mots.

Chaque vecteur document est au final représenté dans l'espace *k*-dimensionnel réduit des termes non bruités. Les documents qui partagent des termes co-occurrents ont des représentations proches. La requête utilisateur est aussi représentée par un vecteur dans l'espace *k*-dimensionnel. Une mesure de similarité est ensuite calculée entre le *k*-vecteur requête et chacun des *k*-vecteurs documents de la collection. A l'issue de la recherche, le système sélectionne les documents pertinents même s'ils ne contiennent aucun mot de la requête.

c. Le modèle connexionniste :

Les systèmes de recherche d'informations basés sur le modèle connexionniste [Kwok, 89], [Boughanem, 92], [Crestani, 95] utilisent les fondements des réseaux de neurones tant pour la modélisation des unités textuelles que pour la mise en œuvre du processus de recherche d'information. Ce modèle peut être vu comme un modèle vectoriel récurrent non linéaire, les neurones formels représentent des objets de la recherche d'information.

L'idée de base est que la recherche d'information est un processus associatif bien représenté par les mécanismes de propagation et d'activation des réseaux de neurones.

Par ailleurs, les capacités d'apprentissage de ces modèles peuvent permettre d'obtenir des systèmes de recherche d'informations adaptatifs.

Sur la base de l'architecture du réseau qu'ils utilisent, les systèmes de recherche d'informations connexionnistes peuvent être repartis en deux catégories :

- les modèles à auto-organisation.
- les modèles à couches.

Dans ce qui suit nous allons définir chaque catégorie.

Ø **Les modèles à auto-organisation** : les systèmes de cette catégorie sont basés sur le modèle des cartes topologiques de Kohonen [Kohonen, 89]. L'apprentissage du réseau permet d'effectuer la classification des documents à partir de leurs descriptions initiales.

Ø **Les modèles à couches** : ces modèles sont largement utilisés en recherche d'information. Certains travaux sont basés sur le modèle probabiliste [Kwok, 89] [Belew, 89], [Crestani & al., 94] et [Kwok, 95]. D'autres utilisent le modèle vectoriel. Les SRI basés sur un modèle connexionniste à couches sont représentés par un minimum de trois couches de neurones interconnectées : la couche requête (q), la couche termes (t) et la couche documents (d) comme le montre la figure1.3. Le mécanisme de recherche est basé sur une activation initiale des neurones termes induite par une requête, et qui se propagent vers les documents à travers les connexions du réseau.

Les valeurs de sortie des différents documents correspondent à leurs degrés de pertinence pour la requête donnée.

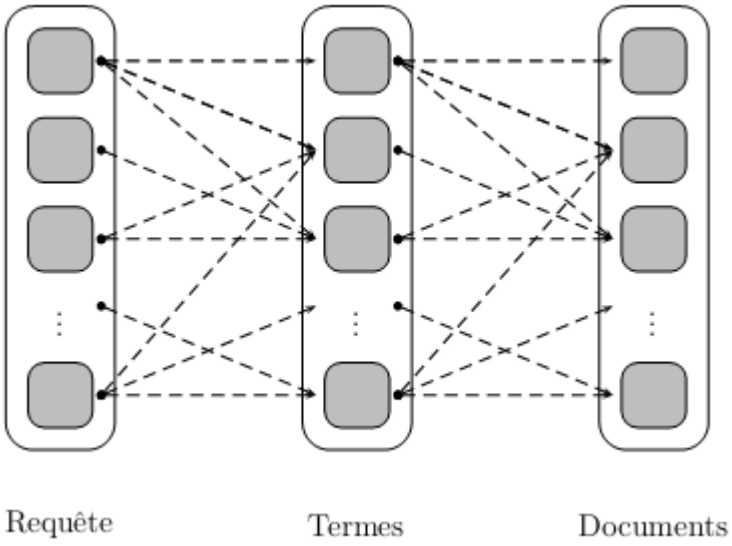


Figure 1.3 : Un modèle de réseau de neurones pour RI

« PIRCS » [Kwok & al., 99] est un exemple de système de recherche d'information basé sur l'approche connexionniste. La pertinence d'un document est RSV (q, d), est calculée par combinaison de deux valeurs de pertinence. La première est produite par le document. La deuxième est produite par la requête. Ces valeurs sont obtenues par un processus de propagation de signaux dans un réseau de neurones.

Formellement, étant donnée une requête q et un document d, nous avons alors :

$$\text{RSV (q, d)} = \alpha RSV_d + (1 - \alpha) RSV_q$$

Où:

- RSV_d : représente la pertinence focalisée sur le document.
- RSV_q : représente la pertinence focalisée sur la requête.

1.5.3 Les modèles probabilistes : [Robertson & al., 76], [Robertson & al., 95]

Ce modèle aborde le problème de la recherche d'information dans un cadre probabiliste. La pertinence document-requête est traduite par le calcul de la probabilité de pertinence d'un document par rapport à une requête. La pertinence entre un document et une requête est mesurée par le rapport entre la probabilité qu'un document d donné soit pertinent pour une requête q, notée p(R=d), et la probabilité qu'il soit non pertinent, notée p (¬R=d), où R est l'événement de pertinence et ¬R de non pertinence. Ces probabilités sont estimées par les probabilités conditionnelles selon qu'un terme de la requête est présent dans un document pertinent ou dans un document non pertinent.

a. le modèle probabiliste de base :

Le premier modèle probabiliste a été proposé par Maron et Kuhns [Maron & al., 60] au début des années 60. Ayant vu l'imprécision et l'incertitude du processus de RI, ces auteurs proposent de modéliser le processus de sélection des documents dans un SRI en utilisant la théorie des probabilités. Le principe de base du modèle probabiliste consiste à présenter les résultats de recherche d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'une requête.

[Robertson, 76] résume ce critère d'ordre par le PRP (Probability Ranking Principal).

Le modèle probabiliste se base sur la distribution des termes, en supposant que :

- La distribution des termes dans les documents pertinents est la même que leur distribution par rapport à la totalité des documents.
- Les variables « documents pertinents », « documents non pertinents » sont indépendantes.
- Deux probabilités conditionnelles sont utilisées dans le processus de décision :

  - P (R/d) : probabilité du document d sachant qu'il est pertinent pour la requête.
  - P (¬R/d) : probabilité du document d sachant qu'in n'est pas pertinent pour la requête.

Etant donné une requête utilisateur notée q et un document d, le modèle probabiliste tente d'estimer la probabilité que le document d appartienne à la classe des documents pertinents (non pertinents). Un document est alors sélectionné si la probabilité qu'il soit pertinent à q est supérieure à la probabilité qu'il soit non pertinent à q.

D'après [Robertson, 94] : le score d'appariement entre le document d et la requête q est donné par :

$$RSV\,(d, q) = \frac{P\,(R/d)}{P\,(\neg R/d)}$$

En appliquant le théorème de Bayes et après simplification on a :

$$RSV\,(d, q) = \frac{P\,(R/d)}{P\,(\neg R/d)} \approx \frac{P\,(d/R)}{P\,(d/\neg R)}$$

Où :

P (d/R) (Respectivement P (d/¬R)) : est la probabilité que le document appartienne à l'ensemble R des documents pertinents (respectivement à l'ensemble ¬R des documents non pertinents).

Si on suppose que les ensembles R (documents pertinents) et ¬R (documents non pertinents) soient connus, alors on peut aisément estimer les probabilités p_i et q_i, en utilisant les proportions définies dans le tableau 1.1 :

Nombre de documents pertinents contenant ti : ri	Nombre de documents pertinents ne contenant pas ti : n-ri	Nombre de documents pertinents : n
Nombre de documents non pertinents contenant ti : Ri-ri	Nombre de documents non pertinents ne contenant pas ti : N-Ri-n+ri	Nombre de documents non Pertinents : N-n
Nombre de documents contenant ti : Ri	Nombre de documents ne contenant pas ti : N-Ri	Nombre d'échantillons : N

Tableau 1.1: Distribution de probabilités de pertinence des termes d'un corpus d'apprentissage.

D'où la RSV se réduit à :

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

b. Le modèle de langue :

Dans les modèles de recherche classiques, on cherche à mesurer la similarité entre un document d et une requête q ou à estimer la probabilité que le document réponde à la requête (p (d_i|q)). L'hypothèse de base dans ces modèles consiste à dire qu'un document n'est pertinent que s'il "ressemble" à la requête. Les modèles de langage, comme décrits par leurs initiateurs Ponte et Croft [Ponte & al., 98], sont eux basés sur une hypothèse différente. Cette hypothèse admet qu'un utilisateur en interaction avec un système de recherche d'information fournit une requête en pensant à un ou plusieurs documents qu'il souhaite retrouver. Le modèle de langue considère que la pertinence d'un document pour une requête est en rapport avec la probabilité que la requête puisse être générée par le document p ($q|d_i$).

En se basant sur le principe d'indépendance des termes (l'apparition d'un terme n'influe pas sur la probabilité d'appartenance d'un autre terme dans le document ou dans la requête), p ($q|d_i$) peut être réécrite comme suit :

$$p\,(q|d_i\,) = \prod_{k=1}^{n} p(q_k|d_i)$$

Où :

- n est le nombre de termes dans la requête.
- q_k est un terme de la requête.
- $(1 \leq k \leq n)$.

Cette équation, introduite par Ponte et Croft [Ponte & al., 98] a été reprise par Berger et Lafferty [Berger & al., 99] pour formaliser l'effet de dérivation des requêtes à partir des documents. Un des modèles proposés [Berger & al., 99] introduit le paramètre de Markov Kernel t $(q_k|\, w_j)$ au modèle. Une version simplifiée de ce modèle peut être écrite sous cette forme :

$$\neg p\,(q|d_i\,) = \psi(n) \prod_{k=1}^{n} \sum_{j=1}^{m} t\,(q_k|\, w_j) p\big(w_j|d_i\big)$$

Où

$\psi(n)$ est une fonction qui exprime la probabilité de générer une requête de longueur n m est le nombre de termes dans le document.

Afin de mettre en évidence les modèles de langage pour la recherche d'information, la formule de base suivante est utilisée :

$$p(d_i, t_1, t_2, \ldots., t_n) = p(d_i) \prod_{j=1}^{n} \Big((1-\lambda) p\big(t_j\big) + \lambda p\big(t_j|d_i\big)\Big).$$

Où:

$\{t_i\}$ sont les termes de la requête.

Hiemstra [Hiemstra 2001] donne à la même équation une justification un peu différente. L'utilisateur est supposé affecter une valeur d'importance binaire à chaque position d'un terme dans la requête.

Si on définit λ_j = p(le terme de position j est important), alors:

$$p(d_i, t_1, t_2, \ldots., t_n) = p(d) \prod_{j=1}^{n} \Big(\big(1-\lambda_j\big) p\big(t_j\big) + \lambda_j p\big(t_j|d_i\big)\Big).$$

1.6 Evaluation des systèmes de recherche d'information :

L'aspect évaluation occupe une place très particulière en RI. L'évaluation des systèmes de recherche d'information [Rijsbergen, 79], [Baeza-Yates & al., 99] constitue une étape primordiale dans l'élaboration d'un modèle de recherche d'information. Elle permet de caractériser le modèle et de fournir des éléments de comparaison entre les modèles existants.

D'une façon générale, un système de recherche d'information idéal a deux objectifs :

- Retrouver tous les documents pertinents,
- Rejeter tous les documents non pertinents.

Ces deux objectifs sont évalués par des mesures de précision et de rappel que nous définissons ci-après.

1.6.1 Les mesures de Précision/Rappel :

Les mesures de précision/rappel sont obtenues en partitionnant l'ensemble des documents restitués par le SRI en deux catégories : les documents pertinents et les documents non pertinents. Ces deux catégories se définissent comme suit:

- **Taux de précision** : La précision mesure la capacité du système à rejeter tous les documents non pertinents à une requête. Il est donné par le rapport entre l'ensemble des documents sélectionnés pertinents et l'ensemble des documents sélectionnés.
- **Taux de rappel** : Le rappel mesure la capacité du système à retrouver tous les documents pertinents répondants à une requête. Il est donné par le rapport entre les documents retrouvés pertinents et l'ensemble des documents pertinents de la base.

Les taux de précision et de rappel sont donnés par les formulations suivantes :

$$\text{Précision} = \frac{R_+}{M}$$

$$\text{Rappel} = \frac{R_+}{R}$$

Où :

- **ü** R: le nombre total de documents pertinents dans la collection,
- **ü** M : le nombre de documents sélectionnés,
- **ü** R+ : le nombre de documents pertinents sélectionnés.

1.6.2 Autres mesures de performance :

Il existe aussi d'autres mesures de performance des SRI telles que :

– **Le temps de réponse** : un SRI doit pouvoir fournir à l'utilisateur les documents correspondants à sa demande dans des temps très courts.
– **La présentation des résultats claire avec facilité d'utilisation** : capacité du système à comprendre les besoins de l'utilisateur et à mettre en valeur les documents correspondants à ceux-ci. Ceci est lié à l'interface avec l'utilisateur.
– **Le nombre total de documents pertinents retournés**, ou le rappel à 1000 documents : ces mesures permettant d'évaluer la performance globale du système au final, en fonction ou non du nombre de documents pertinents total.
– **Le rang du premier document pertinent** : cette mesure a été proposée pour prendre en compte la satisfaction de l'utilisateur qui chercherait un seul document pertinent (comme c'est éventuellement le cas pour les moteurs de recherche sur Internet).
– **La longueur de recherche** : elle est égale au nombre de documents non pertinents que doit lire l'utilisateur pour avoir un certain nombre n de documents pertinents.

1.6.3 Les collections de test :

Une collection de test est constituée d'une collection de documents, d'un ensemble de questions-tests avec leurs « réponses idéales » associées. Ces réponses idéales vont permettre d'évaluer la qualité des réponses fournies par les systèmes à évaluer. Pour évaluer un système de recherche d'information, il suffira alors de lui soumettre les questions-tests, et de comparer les réponses qu'il fournit aux réponses-types. En mesurant l'écart entre la réponse du système et la réponse-type, on obtiendra une mesure de qualité sur les performances du système documentaire.

1.6.4 Les compagnes d'évaluation :

L'histoire de l'évaluation des SRI a connu la naissance de grandes campagnes d'évaluation d'envergure destinées à stimuler et favoriser l'émergence de nouveaux SRI. Ces compagnes sont menées très régulièrement depuis plusieurs années. Les plus connues sont : le programme américain TREC (Text Retrieval Evaluation Conference) entamé en 1992 et le programme CLEF (Cross Language Evaluation Forum). CLEF était à l'origine, la tâche multilingue de TREC avant de devenir depuis 2000, un programme à part entière supportant la majorité des langues européennes.

Ces programmes fournissent l'infrastructure nécessaire pour permettre une procédure d'évaluation uniforme des différentes techniques et stratégies de recherche. TREC par exemple fournit une plate-forme comportant des collections de tests, des tâches spécifiques, des questions-type (topics) et les jugements de pertinence (réponses idéales) correspondants à chaque tâche.

1.7 Conclusion :

Dans ce premier chapitre, nous nous sommes essentiellement intéressés à l'étude des concepts de base et des principaux modèles classiques existants de la recherche d'information. Les systèmes de recherche d'information sont conçus à l'origine pour retrouver les documents pertinents d'un sujet donné. La finalité de chaque système de recherche d'information est de satisfaire les besoins des utilisateurs. Ces derniers sont préoccupés par un seul problème : celui de pouvoir récupérer tous les documents dont ils ont besoin d'une façon rapide et efficace.

Or, le problème majeur des SRI classiques, est qu'ils se contentent de chercher les documents qui contiennent les mêmes mots que ceux de la requête. Ceci est évidement insuffisant dans la mesure où la performance de ces systèmes se voit détériorée.

Pour remédier à ce problème, de nouvelles approches d'indexation ont été développées se basant sur la sémantique. L'exploitation de la sémantique permet de tenir compte des relations entre termes, il est, ainsi, possible d'améliorer les performances d'un SRI en utilisant un espace sémantique plutôt qu'un espace induit par la morphologie des termes.

Plusieurs approches d'indexation se basant sur la sémantique ont été apparues, nous nous intéressons précisément à l'approche d'indexation par la sémantique latente qui sera l'objectif du prochain chapitre.

Chapitre2 :

« Indexation par la Sémantique Latente »

LSI

2.1 Introduction:

L'Indexation par la Sémantique Latente « LSI » est une approche d'indexation qui regroupe les termes co-occurrents en concepts, d'où la réduction de l'espace vectoriel des termes d'indexation. Les documents et les requêtes sont alors représentés dans le nouvel espace composé de concepts de haut niveau, ce que permet de sélectionner des documents pertinents, même s'ils ne contiennent aucun terme de la requête. [Deerwester & al., 90].

Le modèle LSI se base sur la décomposition en valeurs singulières, ou SVD (Singular Valeur Décomposition), de la matrice terme-document.

Le présent chapitre est consacré à l'étude de l'approche LSI et il est divisé en deux parties :

- La première présente l'apport de la sémantique en recherche d'information et décrit la problématique de la RI classique basée mots-clés.
- La deuxième est dédiée à l'indexation par la sémantique latente. Dans cette partie, on décrit les différentes approches de LSI et on donnera les avantages et inconvénients de cette approche.

2.2 De la RI lexicale à la RI sémantique :

Une caractéristique commune à tous les modèles classiques de RI est qu'un document, pour être sélectionné par le système, doit contenir les mêmes mots que la requête. Chaque mot étant considéré comme une suite de caractères.

Nous constatons alors une contradiction entre le but de la RI et la méthode qui la réalise :

RI a pour objectif de retrouver des documents ayant une certaine signification pour la requête, alors qu'elle est implémentée de façon à ce qu'elle cherche des documents contenant les mêmes mots que cette dernière.

Il convient donc de s'interroger sur la possibilité de munir la recherche d'information, en plus d'un index lexical, de ressources sémantiques qui permettront de définir des relations entre les termes de sorte à expliciter le lien sémantique entre le document et la requête.

Plusieurs travaux tentant d'incorporer l'information sémantique dans le processus de la RI sont alors apparus, qui peuvent être classifiés en :

- Approches basées sur l'Indexation par la Sémantique Latente :
  L'Indexation par la Sémantique Latente résout les sens des mots par un clustering des mots sémantiquement proches via une technique de réduction de la dimensionnalité de la matrice termes-documents.

- Approches basées sur l'Indexation Sémantique :
  L'Indexation Sémantique tente de retrouver, parmi les différents sens possibles d'un mot tels que définis dans les dictionnaires, le sens correct du mot dans le texte à indexer. Elle se base historiquement sur les techniques de désambiguïsation des sens des mots (Word Sense Disambiguation WSD) pour résoudre le problème de polysémie en affectant le sens adéquat à un mot dans son contexte d'utilisation dans le document ou la requête. [Amirouche, 2008; Mihalcea, 2004]

- Approches basées sur l'Indexation Conceptuelle :
  L'Indexation Conceptuelle tente à partir d'une taxonomie conceptuelle extraite du texte, de construire sa sémantique. Les liens entre les différents concepts d'une telle taxonomie sont des liens fonctionnels entre entités lexicales.
  L'Indexation Conceptuelle peut être vue comme une généralisation de l'indexation sémantique, dans la mesure où les concepts aussi véhiculent des sens.
  [Baziz, 2005; Woods, 97]

2.3 L'Indexation par la Sémantique Latente :

L'indexation par la sémantique latente « LSI » est une technique d'indexation développée dans les années 90 par [Deerwester & al., 90]. Son but est d'améliorer les techniques d'indexation textuelle et de permettre le rapprochement sémantique de documents au travers des mots les composant.

Techniquement, LSI repose sur la définition suivante :
« Deux mots sont similaires s'ils apparaissent dans des contextes similaires. Deux contextes sont similaires s'ils comportent des mots similaires. Cette récursivité croisée exige un mécanisme particulier, bien plus complexe qu'un simple compactage d'occurrences. »

En effet deux mots peuvent être considérés sémantiquement proches s'ils sont utilisés dans des contextes similaires. Le contexte d'un mot est défini ici comme l'ensemble des mots qui appariassent conjointement avec lui. Par exemple, le mot « vélo » apparait dans le contexte de mots comme « pédaler », « randonnée » et « guidon ». Le mot « bicyclette » apparait aussi dans des contextes similaires. Ces deux mots sont alors considérés sémantiquement proches. LSI regroupe les mots qui sont sémantiquement proches (similaires) en concepts.

Dans la LSI les liens entre les termes et les documents sont calculés et exploités dans le processus de recherche. L'idée principale est que les idées dans un texte sont plus reliées aux concepts qu'elles décrivent que les termes de l'index utilisés pour leur description. Ainsi, la correspondance entre un document et une requête donnée devrait être basée sur la correspondance des concepts plutôt que sur la correspondance des termes de l'index. L'objectif fondamental est d'aboutir à une représentation conceptuelle des documents. Ainsi, les requêtes peuvent rechercher des documents même si elles n'ont aucun mot en commun.

Le but de LSI est de transformer une représentation par des mots-clés en une autre représentation qui est « meilleure ». Le mot « meilleure » est compris dans le sens suivant: les documents et les requêtes sémantiquement similaires seront plus proches avec la représentation transformée qu'avec les mots-clés.

2.3.1 Les concepts fondamentaux de LSI :

2.3.1.1 Matrice d'occurrence :

Le modèle LSI utilise une matrice « terme-document » qui contient les termes sur les lignes et les documents sur les colonnes.

Pour générer la matrice terme-document, on utilise le modèle de l'espace vectorielle (VSM) pour construire les vecteurs représentant des documents, puis on met ces vecteurs dans une matrice dont les colonnes sont les vecteurs représentant des documents et où chaque ligne contient le poids de chaque terme dans les documents. Le modèle vectorielle utilise la mesure tf*idf pour le calcule de ce poids quand à déjà vu dans le chapitre précédent.

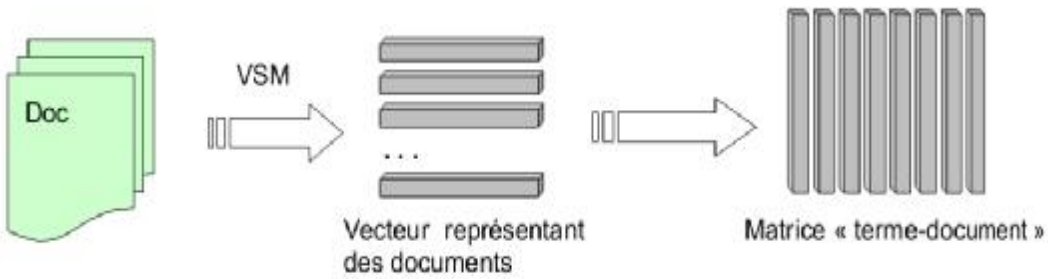


Figure 2.1 : Génération de la matrice termes-documents.

On appliquant une réduction des dimensions basée sur la décomposition en valeurs singulières « SVD » sur cette matrice, les dimensions les moins pertinentes (bruit) sont éliminées et un modèle optimisé est obtenu. Le principe de la décomposition SVD de la matrice terme-document est décrit dans ce qui suit.

2.3.1.2 Décomposition en valeurs singulières « SVD » :

SVD est une méthode algébrique, qui permet la décomposition en valeurs singulières de la matrice terme-document. L'objectif étant de construire une matrice de dimension plus faible

qui donne une approximation de cette matrice des occurrences. On peut justifier cette approximation par plusieurs aspects :

- La matrice d'origine pourrait être trop grande pour les capacités de calcul de la machine. On rend ainsi le procédé de calcul réalisable.
- La matrice d'origine peut être « bruitée », des termes n'apparaissant que de manière anecdotique. On « nettoie » ainsi la matrice en là réduisant, c'est une opération qui améliore les résultats.
- La matrice d'origine peut être présumée « trop creuse » : elle contient plutôt les mots propres à chaque documents que les termes liés à plusieurs documents.

Techniquement, si W est la matrice terme-document de dimension t × d, où t est le nombre de termes distinct de la collection et d le nombre de documents dans la collection, alors SVD la décompose en :

$$W = T \times S \times D \quad (1)$$

Où :

- Ø T est une matrice terme de taille t× r, qui représente les termes des documents;
- Ø S est une matrice diagonale de valeurs singulières de taille r× r diagonale (seul les éléments en diagonal sont non-nuls);
- Ø D est une matrice document de taille r× d. chaque colonne représente les documents dans l'espace vectoriel;
- Ø La valeur r est choisie comme une valeur ≤ min(t, d).

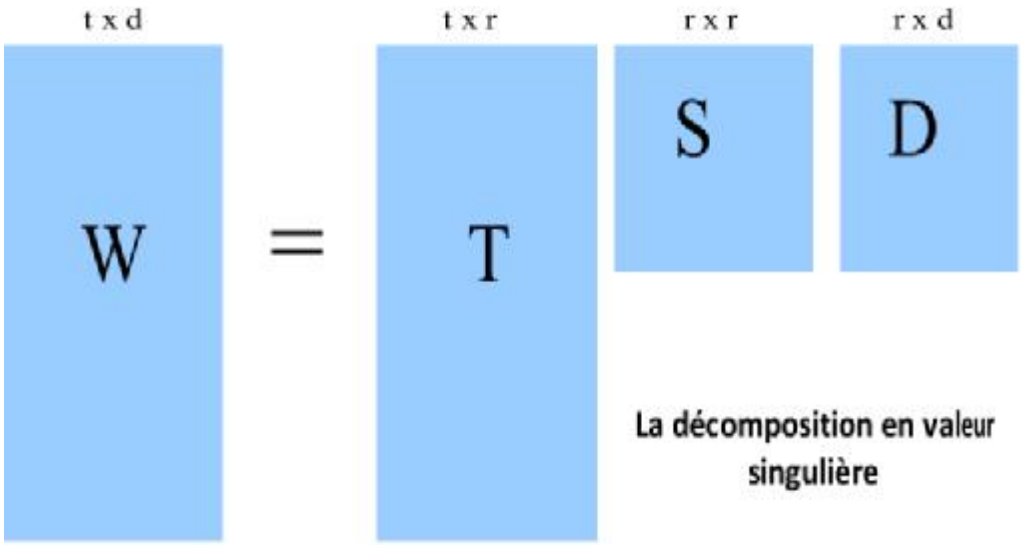


Figure 2.2 : la décomposition en valeur singulière

En suite il faut ranger les éléments de S par ordre décroissant et réarranger les colonnes correspondantes de T et D de façon à garder l'égalité (1).

Une fois la SVD de la matrice W est calculée, il s'agit de sélectionner les K première valeurs singulières de la matrice S. La matrice réduite de S est notée par S1.

Le nombre de dimensions optimal (à environs 300) a été estimé empiriquement pour la langue anglaise [Landauer & Dumais, 97] sans que l'on puisse le justifier théoriquement.

La nouvelle matrice concepts-documents W_k est obtenue en gardant les colonnes correspondantes dans les matrices T et D dans cette espace à K dimension. Les matrices T et D nettoyées deviennent T1 et D1.

$$W_k = T1 \times S1 \times D1$$

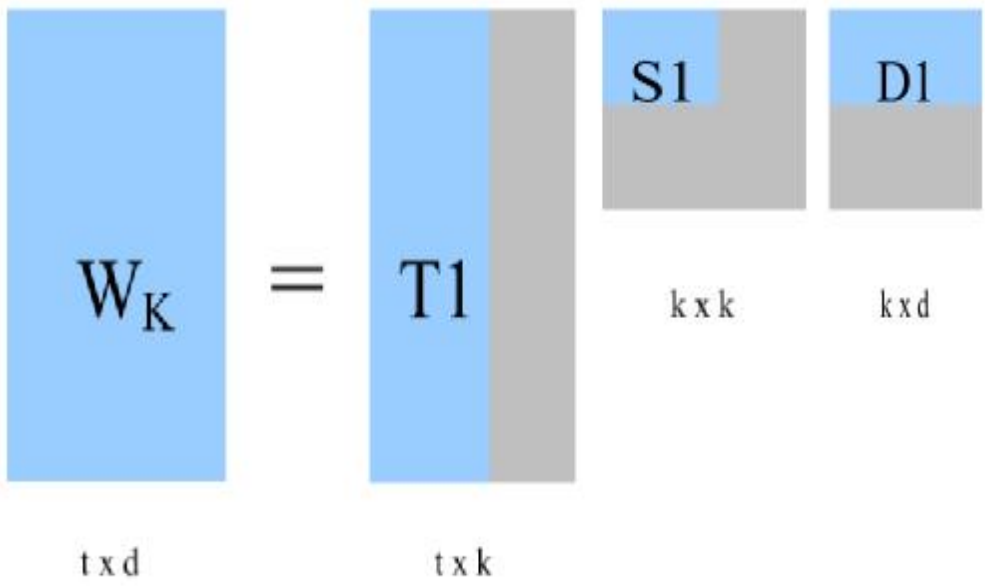


Figure 2.3 : Réduction de l'espace

Il n'y a pas de règle établie pour la détermination de k. Instinctivement nous pouvons supposer que plus k est faible plus nous éliminons du bruit dans les données, mais que parallèlement nous perdons de l'information : il faut prendre grand soin lors du choix de ce nombre.

Cette réduction va nous permettre de ne garder que les k termes les plus significatifs, et afin de calculer la similarité documents-documents, les documents sont représentés dans un espace vectoriel de dimensions k. Les colonnes sont les coordonnées des vecteurs documents. Par la suite, la LSI s'appuie sur une mesure basée sur le calcul de cosinus de l'angle formé par deux vecteurs, celui-ci représentant la similarité.

2.3.1.3 Requête dans LSI :

Comme pour un document, une requête est représentée par un vecteur de mots-clés dans le nouvel espace de K dimension.

D'après [Deerwester & al., 90] une requête est transformée d'abord en un pseudo-document comme suit :

$$d_q = X_q T1 S1^{-1}$$

Où :

X_q est le vecteur de mots-clés de la requête (dans l'espace de mots-clés).

Ensuite, ce pseudo-document est ajouté dans la matrice D1 comme un nouveau document.

Une mesure de similarité est appliquée entre chaque paire de documents. Ainsi, après ce calcul, on peut connaitre la similarité de ce pseudo-document (la requête) avec tous les autres documents.

Typiquement, le z le plus étroit des documents (les documents les plus pertinents ayant le plus grand score) est retourné aux utilisateurs.

2.3.1.4 Mesure de similarité dans LSI :

Une fonction de similarité est appliquée pour ordonnancer les documents par degré de similarité aux requêtes. LSI se base généralement sur la mesure de cosinus pour le calcul de similarité comme dans le modèle vectorielle.

2.3.2 Exemple applicatif de LSI : [Garcia, 2006]

- Soit la matrice terme-document A se composant de 3 documents (colonnes) et 11 termes (lignes) et la matrice requête q :

Terms	d1	d2	d3	q
a	1	1	1	0
arrived	0	1	1	0
damaged	1	0	0	0
delivery	0	1	0	0
fire	1	0	0	0
gold	1	0	1	1
in	1	1	1	0
of	1	1	1	0
shipment	1	0	1	0
silver	0	2	0	1
truck	0	1	1	1

A = ... q = ...

- Décomposer la matrice A par SVD pour construire les matrices U, S et V :

$$A = U \times S \times V^T$$

$$U = \begin{bmatrix} -0.4201 & 0.0748 & -0.0460 \\ -0.2995 & -0.2001 & 0.4078 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.1576 & -0.3046 & -0.2006 \\ -0.1206 & 0.2749 & -0.4538 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.4201 & 0.0748 & -0.0460 \\ -0.2626 & 0.3794 & 0.1547 \\ -0.3151 & -0.6093 & -0.4013 \\ -0.2995 & -0.2001 & 0.4078 \end{bmatrix} \qquad S = \begin{bmatrix} 4.0989 & 0.0000 & 0.0000 \\ 0.0000 & 2.3616 & 0.0000 \\ 0.0000 & 0.0000 & 1.2737 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & 0.2469 \\ -0.5780 & -0.2556 & 0.7750 \end{bmatrix}$$

- Réduire l'espace à 2 dimensions : (K=2)

Les nouvelles matrices nettoyées sont :

$$U1 = \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \qquad S1 = \begin{bmatrix} 4.0989 & 0.0000 \\ 0.0000 & 2.3616 \end{bmatrix}$$

$$V1^T = \begin{bmatrix} -0.4945 & -0.6458 & -0.5817 \\ 0.6492 & -0.7194 & 0.2469 \end{bmatrix}$$

Les nouvelles coordonnées des vecteurs documents dans le nouvel espace sont :

$d1\ (-0.4945, 0.6492)$;

$d2\ (-0.6458, -0.7194)$;

$d3\ (-0.5817, 0.2469)$.

- Calculer les coordonnées du vecteur requête dans le nouvel espace :

$$q = q^T \times U1 \times S1^{-1}$$

$$q = [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1] \begin{bmatrix} -0.4201 & 0.0748 \\ -0.2995 & -0.2001 \\ -0.1206 & 0.2749 \\ -0.1576 & -0.3046 \\ -0.1206 & 0.2749 \\ -0.2626 & 0.3794 \\ -0.4201 & 0.0748 \\ -0.4201 & 0.0748 \\ -0.2626 & 0.3794 \\ -0.3151 & -0.6093 \\ -0.2995 & -0.2001 \end{bmatrix} \begin{bmatrix} \frac{1}{4.0989} & 0.0000 \\ 0.0000 & \frac{1}{2.3616} \end{bmatrix} = [-0.2140\ \ -0.1821]$$

- Calculer la similarité de la requête avec tous les autres documents :

$$sim\ (q, d1) = \frac{(-0.2140)(-0.4945) + (-0.1821)(0.6492)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\ \sqrt{(-0.4945)^2 + (0.6492)^2}} = -0.0541$$

$$sim\ (q, d2) = \frac{(-0.2140)(-0.6458) + (-0.1821)(-0.7194)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\ \sqrt{(-0.6458)^2 + (-0.7194)^2}} = 0.9910$$

$$sim\ (q, d3) = \frac{(-0.2140)(-0.5817) + (-0.1821)(0.2469)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\ \sqrt{(-0.5817)^2 + (0.2469)^2}} = 0.4478$$

L'ordre de pertinence des documents est : d2 > d3 > d1.

2.3.3 Les variantes de LSI :

2.3.3.1 Approche CL-LSI :

La méthode CL-LSI (Croisement de Langue-LSI) [Littman & al., 98] est fondée sur le même principe que la LSI. Elle est appliquée à la recherche d'information par croisement de langues. L'idée de base consiste à considérer un ensemble de documents dans une langue, les traduire soit manuellement par un expert ou automatiquement par un traducteur dans le but de construire un ensemble de documents duaux.

Un document dual ou virtuel est la concaténation d'un document exprimé dans une langue et sa traduction intégrale dans une autre langue. La figure 2.4 représente un document dual (Anglais-Français) issu de la collection HANSARD. Lors de la phase d'analyse, le document dual est considéré comme un seul document indépendamment de la langue. L'ensemble de documents duaux est analysé en utilisant la LSI. Le résultat est représenté par l'espace sémantique réduit ou les termes reliés sont regroupés dans le même concept comme le montre la figure 2.5. L'espace de la figure 2.5 représente les documents duaux (Vdoc1 jusqu'à Vdoc3) et les termes apparaissant dans ces documents. Du fait que le document dual contient des termes en Français et en Anglais, l'espace LSI va automatiquement contenir les termes dans les deux langues par exemple (Aword 1 jusqu'a Aword 4 en Anglais et Fterme 1 jusqu'à Fterme 4 en Français).

Mr. Speaker, we are in constant touch with our consular officials in Bosnia. We are advised the situation there is stabilizing now. There is no immediate threat to Canadians. Therefore my responses yesterday, which no doubt the Hon. Member has seen, have not altered.

Monsieur le président, nous sommes en communication constante avec nos représentants consulaire en Bosnie. D'après nos informations, la situation est en train de se stabiliser, et les Canadiens ne sont pas immédiatement menacés. Par conséquent, mes réponses d'hier, dont le représentant a dû prendre connaissance, n'ont pas changé.

Figure 2.4 : Exemple d'un document dual (Anglais-Français)

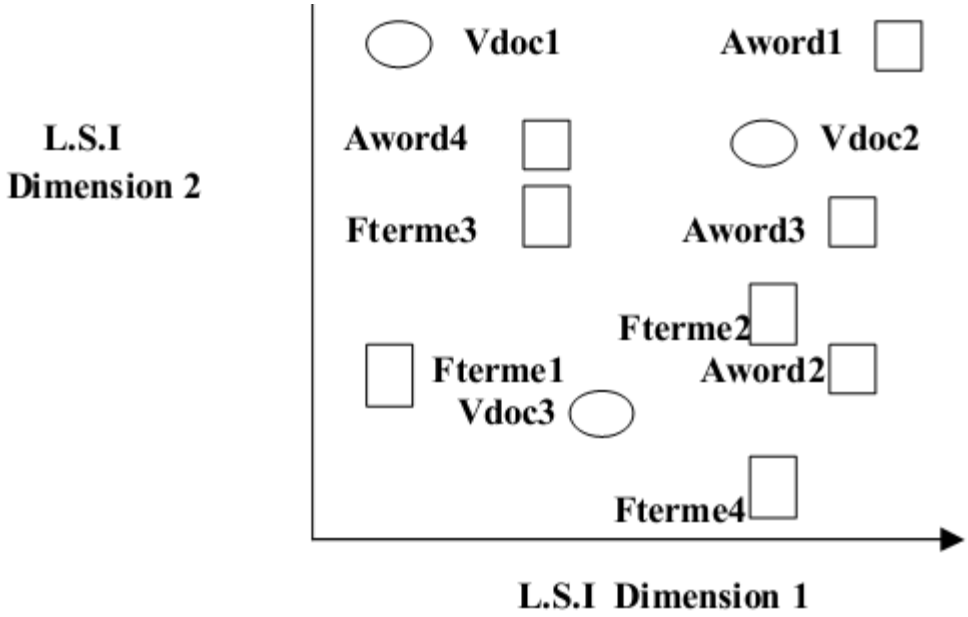


Figure 2.5 : Représentation des documents duaux avec les termes reliés

L'étape suivante dans la méthode CL-LSI consiste à représenter les documents dans chaque langue. La figure 2.6 représente les documents exprimés en Français et en Anglais dans l'espace des termes Anglais et Français. Dans ce cas, l'utilisateur peut poser sa requête soit en Français ou en Anglais et récupérer les documents les plus semblables dans ces deux langues.

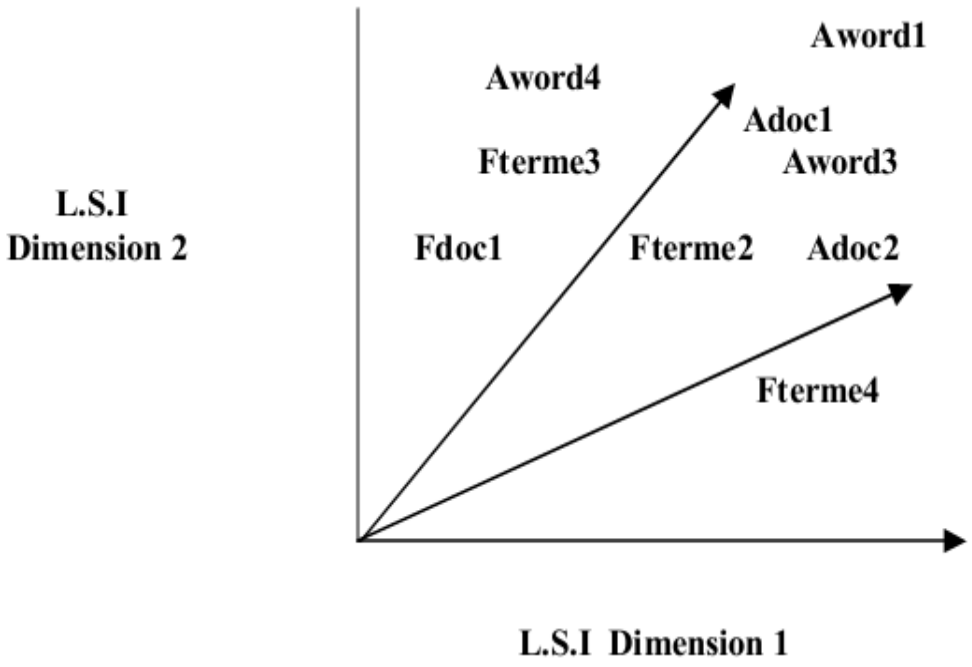


Figure 2.6 : Représentation des documents Français/Anglais avec les termes reliés

Landauer et littman [Landauer & al., 98] ont appliqué cette technique en utilisant un corpus parallèle contenant des documents en Anglais et leur traduction en Français. Ils ont montré que leur méthode est efficace lors de la traduction de requête.

Cette technique a été également utilisée par Oard [Oard & al., 98]. Il a utilisé des collections contenant des résumés d'articles exprimés en Espagnol et en Anglais.

Tous ces travaux ont démontré que la LSI est viable pour la recherche d'information par croisement de langues. Ils ont également montré que le nombre et la longueur des documents à utiliser pendant la phase de recherche influent considérablement sur les résultats obtenus.

2.3.3.2 Approche PLSI « Probabilistic latent Semantic Indexing » :

Le modèle PLSI [Hofmann & al., 99; 2000; 2001] offre une approche d'indexation de documents fondée sur des modèles probabilistes de catégories sémantiques latentes.

Dans PLSI, les documents sont représentés comme des occurrences successives de paires d'indices (d, w) pour une catégorie sémantique z ϵ Z = {z_1, z_2,…, z_k} donné, (c.à.d. z est une variable latente « non observée », on suppose l'existence d'un certain nombre de valeurs de z « k valeurs », mais on ne sait rien sur elles), d étant l'indice d'un document et w celui d'un terme.

Autrement dit, PLSI modélise les documents comme des réalisations de tirages aléatoires successifs de couples document-termes (d, w) : itérativement, une catégorie sémantique z ϵ Z est d'abord choisie, avec une probabilité p(z), puis un terme w et un document d sont choisis avec respectivement des probabilités p(w|z) et p(d| z).

De plus, w et d sont supposés indépendants sachant z, de sorte que la probabilité d'une paire (d, w) s'écrit :

$$P(d, w) = \sum_{z \epsilon Z} p(z)p(w|z)p(d|z).$$

Les paramètres de PLSI sont $p(z)$, $p(w|z)$ et $p(d|z)$, pour tous les z, w et d possibles dans le modèle.

Ces paramètres s'estiment pour une collection de documents donnés en utilisant une variante de l'algorithme Maximisation de l'Espérance (EM) d'après [Hofmann & al., 99 ; 2001].

Cet algorithme a pour objectif l'évaluation de probabilité maximale où les données sont inachevées ou la fonction de probabilité implique des variables latentes. Noter que la notion des données inachevées et les variables latentes sont connexes (quand nous avons une variable latente, nous pouvons considérer nos données comme étant inachevés puisque nous n'observons pas des valeurs des variables latentes ; pareillement, quand nos données sont inachevées, nous pouvons associer une certaine variable latente aux données absentes).

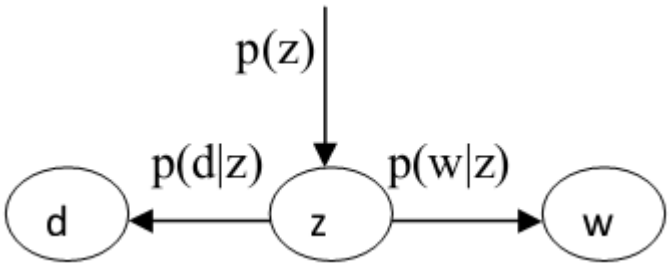


Figure2.7 : représentation graphique du model PLSI symétrique

Dans le contexte PLSI, « la similarité cosinus » employée à l'origine, sans justification théorique pour évaluer la proximité entre documents, a laissé la place à des similarités à base des noyaux de Fisher, mieux justifié théoriquement [Hofmann & al., 2000].

Cette approche a ensuite été étendue à plusieurs autres modèles de similarité.

Le noyau de Fisher pour PLSI dérivé par [Hofmann & al., 2000], mesurant la distance entre un document d et une requête q est défini par:

$$K^H(d,q) = \sum_z \frac{p(z|d)p(z|q)}{p(z)} + \sum_w p'(w|d)p'(w|q) \sum_z \frac{p(z|d,w)p(z|q,w)}{p(w|z)}$$

Avec :

$p'(w|d) = \frac{n(d,w)}{|d|}$;

$n(d,w)$: Le nombre d'occurrences du terme w dans le document d ;

$|d| = \sum_w n(d,w)$, la taille du document en nombre de termes.

Cette mesure améliore notablement les performances par rapport aux formulations originales qui utilisent la mesure cosinus. [Chappelier & al.]

2.3.3.3 Approche LSI Normalisé « NLSI »:

[Husbands & al., 2000] ont noté que l'effet du facteur IDF était atténué par l'algorithme de décomposition SVD. Après étude du problème sous l'angle de la théorie de perturbation des valeurs propres, ils ont proposé le modèle NLSI où la matrice des valeurs propres est recalculée à partir du produit des deux premières matrices T1 et S1 d'abord factorisées par l'algorithme SVD, puis normalisées.

Les étapes de NLSI sont décrites comme suit :

- Factoriser et réduire la matrice terme-document W : W_k = T1 x S1 x D1
- Normaliser les rangées des termes projetés dans le nouvel espace T1 x S1 :
  T = norm (T1 x S1)

- Recalculer les valeurs propres en utilisant la normalisation calculée dans l'étape précédente :

  S = (d' x T) (T' x q)

  Avec d : document et q : requête
- Utiliser T comme opérateur de projection dans la conversion des documents et des requêtes.
- Calculer le score en utilisant la mesure de similarité cosinus.

Selon les auteurs, cet ajustement aurait un effet significatif sur la qualité de recherche dans les collections volumineuses.

2.3.4 Avantages et inconvénients de LSI :

Avantages : les avantages principaux de LSI sont :

- La résolution de la Synonymie, Polysémie et Dépendance de termes [Deerwester & al., 90].
- LSI à une capacité d'établir la proximité sémantique entre des mots et entre des documents.
- LSI est souple et simple.
- Méthode purement mathématique.
- Réduction de l'espace.
- Augmentation de la précision.

Inconvénients :

- Prétraitements lourds.
- Traitement plus important de la requête (projection de la requête dans l'espace conceptuel).
- Sensibilité à la quantité et à la qualité des données traitées.
- Les concepts extraits à partir de la transformation SVD sont des concepts artificiels, à savoir statistiques. Il est, ainsi, très difficile de les interpréter**.**
- La méthode SVD est une méthode très complexe. Elle a besoin de beaucoup de calculs et cela peut prendre énormément de temps pour une grande collection de documents.

2.4 Conclusion :

Nous avons vu dans ce chapitre la présentation détaillée de l'approche LSI à travers ses concepts fondamentaux, et un exemple explicatif. Enfin, quelques approches liées à LSI y sont présentées.

L'approche LSI se base sur deux principaux facteurs: Le premier est l'extraction des relations liant les termes. Ces relations permettent de définir un espace sémantique plus adapté aux traitements de documents notamment en tenant compte de la synonymie. Le second est la réduction de dimension obtenue en limitant la définition de l'espace sémantique aux relations les plus significatives. Ainsi, le bruit contenu dans les données d'origine se trouve éliminé dans l'espace sémantique.

L'objectif de notre travail est d'implémenter un module d'indexation sémantique LSI, (puis éventuellement de l'intégrer à la plate forme de RI Terrier[1]).

Le chapitre suivant est dédié à la conception de notre application.

[1] www.terrier.org

Chapitre3:

Conception

3.1 Introduction :

Après avoir présenté dans les chapitres précédents, les différents concepts nécessaires à l'accomplissement de notre travail, nous passons maintenant à sa partie principale à savoir la conception.

Ce chapitre est divisé en deux parties, la première est dédiée à la description de l'approche LSI implémentée et la seconde est consacrée au détail de l'intégration de l'approche LSI dans la plate-forme Terrier. L'intégration de LSI dans la plate-forme Terrier constitue une extension de cette plate-forme à la prise en charge de la RI sémantique. Le système résultant est nommé **LSI-Terrier**.

3.2 Description de l'approche LSI :

3.2.1 Conception globale:

Nous présentons dans ce qui suit les différentes étapes d'indexation par LSI appliquées pour l'indexation des documents et de la requête :

a. Indexation des documents :

Entrée : Chemin vers la collection des documents ;
Sorties : La matrice des termes, la matrice diagonale et la matrice des documents représentées dans l'espace réduit de concepts ;

Début

Pour chaque document de la collection **faire**
Ouvrir le document à indexer ;
Appliquer l'indexation classique à ce document pour récupérer les termes d'index ;
Fait

Pour chaque terme des termes d'index **faire**
Pour chaque document de la collection **faire**
Construire la matrice terme-document pondérée « matrixPoid » en utilisant la mesure de pondération TF-IDF ;
Fait
Fait

Effectuer la décomposition en valeurs singulières sur la matrice pondérée

SVD (matrixPoid) = T x S x D ;

// *T : la matrice des termes, D : la matrice des documents et S : la matrice diagonale ;*

Choisir la valeur de la dimension réduite : K ;

Ramener la dimension de S à k, la matrice réduite de S est notée par S_k, de même réduire les matrices T et D dans le nouvel espace pour avoir les nouvelles matrices réduites T_k, D_k;

Fin.

b. Indexation de la requête :

Entrée : Chemin vers la requête;
Sortie : Vecteur de la requête dans l'espace réduit;

Début

Appliquer l'indexation classique à la requête pour récupérer ses index ;
Pour chaque terme des termes d'index **faire**
Construire le vecteur d'occurrence de la requête de terme d'index Q ;
Fait
Calculer la matrice transposée de Q : Q^T ;
Calculer la matrice inverse de S_K : S_K^{-1} ;
Calculer les coordonnées du vecteur Q dans le nouvel espace réduit
$Q_K = Q^T \times T_K \times S_K^{-1}$;

Fin.

c. Appariement Document-requête

Dans LSI, l'appariement entre un document et une requête est effectué en utilisant la mesure de similarité cosinus:

Pour chaque document de la collection des documents **faire**

$$\text{Sim}(Q_k, D_k[i]) = \frac{Q_k * D_k[i]}{|Q_k| * |D_k[i]|}$$

Fait

Où:

Q_k: le vecteur de la requête de dimension réduite;

$D_k[i]$: le vecteur du document i de dimension réduite.

3.2.2 Conception détaillée:

Pour réaliser l'approche d'indexation sémantique latente, nous avons mis en œuvre un ensemble de classes définit dans ce qui suit :

- **La classe WeightedMatrix** : elle permet de construire la matrice terme-document pondérée en utilisant la mesure TF-IDF ;
- **La classe SVD** : qui permet la décomposition en valeurs singulières de la matrice pondérée en un produit de 3 matrices : la matrice des termes, la matrice diagonale et la matrice des documents, puis réduit la dimension de ces trois matrices après que la valeur de la dimension réduite est choisie.
- **La classe Request** : qui permet de construire le vecteur de la requête dans un espace de K-dimensions et de calculer la similarité entre la requête et chaque document représentés dans l'espace réduit.

3.3 Intégration de LSI dans la plate-forme Terrier :

3.3.1 Présentation de Terrier :

3.3.1.1 Présentation générale :

Terrier, TERabyte RetrIEveR est un moteur de recherche robuste et efficace qui offre la plate-forme idéale pour travailler avec des grandes quantités de données « jusqu'à 25 millions de documents ». Développé par le département informatique de l'université Glasgow de Scotland. Il est utilisé avec succès pour la recherche Ad-hoc, la recherche web et la recherche inter-langages dans des environnements centralisés et distribués.

Terrier est un logiciel open sources entièrement écrit en java, selon d'études comparatives sur les SRI en open sources effectuée par Ricardo Baeza-Yates, Terrier fait partie des trois meilleurs systèmes dans un environnement java, les deux autres sont MG4J et Lucene.

Comme tous les moteurs de recherche, Terrier possède les principales facettes suivantes :

- Indexation : permet l'extraction des termes des différents documents du corpus.
- Recherche : permet de générer des résultats aux requêtes formulées par les utilisateurs.
- Evaluation des résultats de la recherche.

(Voir Annexe1 pour plus de détails sur le système Terrier)

Dans ce qui suit, nous allons nous intéresser aux processus d'indexation et de recherche que nous allons modifier afin d'intégrer notre module LSI.

3.3.1.2 Le processus d'indexation de Terrier :

L'indexation dans Terrier est divisée en quatre procédures et à chaque procédure des classes java peuvent être ajoutés pour la personnalisation du système.

Les quatre procédures sont :

1. Splitter la collection de document : consiste à parcourir l'ensemble du corpus reçu en entrée par Terrier et envoyer chaque document à l'étape suivante.

2. Extraction des termes (Tokenize Document) : qui consiste à parser chaque document reçu et en extraire les différents termes.

3. Traitement des termes extrait avec TermPipelines : consiste en l'élimination des mots vides et la lemmatisation des termes.

4. La construction de l'index.

(Toutes ces étapes, les modules en charge de leur exécution ainsi que les fichiers résultants de cette indexation sont présentés de façon plus détaillée en Annexe1.)

Les différentes classes associées au processus d'indexation sont organisées dans un ensemble de packages dont on trouve :

Org.terrier.indexing : ce package contient les différentes classes permettant de réaliser un ensemble d'opérations sur la collection des documents, dans le but d'extraire les termes de tous les documents de la collection.

Org.terrier.terms : les classes qui se trouve dans ce package permettent d'effectuer un ensemble de traitement sur les termes extraits. Parmi ces traitement l'élimination des mots vides, lemmatisation des termes, …etc.

Org.terrier.structures : les classes de se packages permettent la construction d'un ensemble de structures où un ensemble de données est stocké, parmi ces structure on a :

- Lexicon : qui stocke les informations de chaque terme de la collection ;
- Inverted Index : où le fichier inverse est stocké ;
- Document Index : qui contient des informations sur les différents documents de la collection ;
- …etc.

La figure ci-dessous donne un aperçu de l'interaction des principaux composants impliqués dans le processus d'indexation.

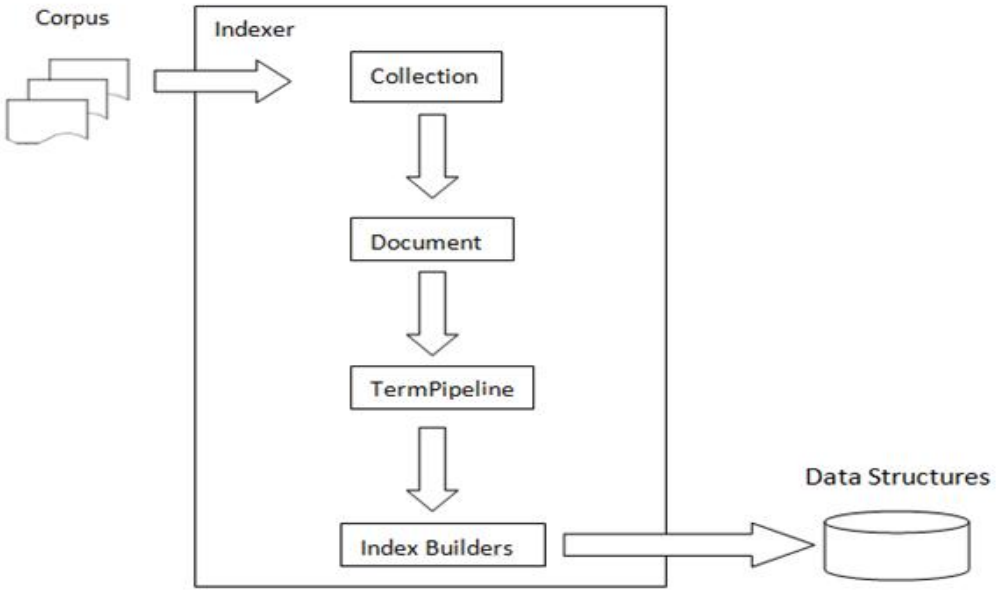


Figure 3.1 : processus d'indexation de Terrier

3.3.1.3 Le processus de recherche de Terrier :

Durant le processus de recherche, chaque requête doit passer par les étapes suivantes :

1. Parsing : qui se charge de tokenizer la requête.
2. Pré-processing : qui applique le TermPipeline à la requête. Elimine les mots vides et les lemmatise.
3. Matching : qui est responsable de l'initialisation du WeightingModel et du calcul des scores entre la requête et les documents.
4. post-filtrage : va filtrer les résultats.
5. post-traitement : peut modifier le ResultSet, par exemple, par un procédé QueryExpansion afin de générer un meilleur classement de documents.

(Toutes ces étapes et les modules en charge de leur exécution seront détaillés en Annexe1.)

Parmi les packages où se trouve les classes associées au processus de recherche on a :

Org.terrier.Matching : contient les classes qui permettent de déterminer les documents répondant à la requête.

Org.terrier.matching.models : on trouve dans ce package les différents modèles de pondération dont TF-IDF, BM25, …etc ;

La figure suivante donne un aperçu de l'interaction des composants Terrier dans la phase de recherche :

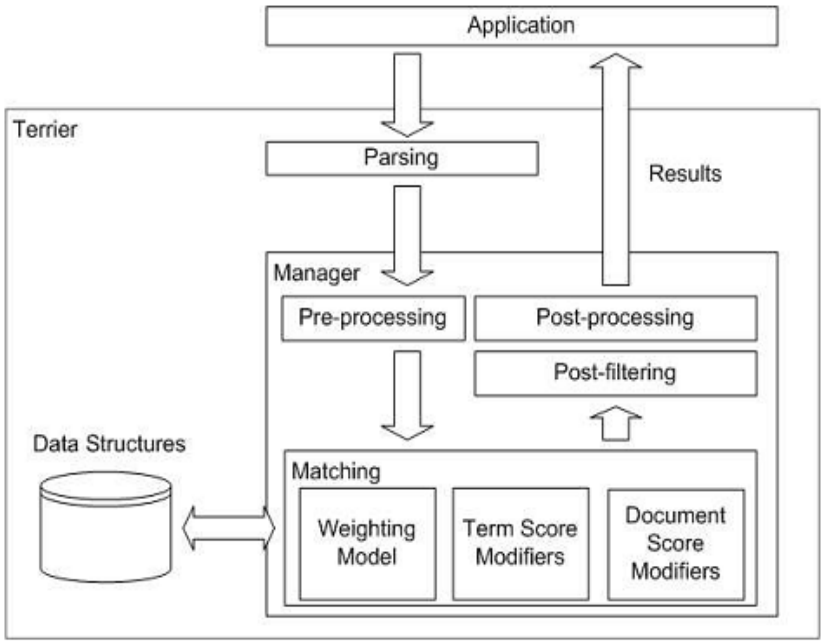


Figure3.2 : processus de recherche de Terrier

3.3.2 Présentation de LSI-Terrier :

3.3.2.1 Conception globale :

Notre objectif à travers cette proposition est d'étendre la plate-forme Terrier avec un module d'indexation sémantique latente. En pratique, il s'agit d'intégrer notre module implémentant l'approche d'indexation LSI dans Terrier 3.5. Le système résultant est nommé LSI-Terrier.

Comme nous avons dit précédemment, l'indexation par LSI consiste à construire une matrice terme-document qu'on décompose par SVD en 3 matrices (matrice diagonale, matrice terme et matrice document) ; ces matrices seront ensuite représentées dans un espace de dimension réduite (K dimensions, K est une valeur donnée).

Pour construire la matrice terme-document, nous avons besoin des termes d'index. Cette matrice est construite à l'issue de l'indexation classique par Terrier. Le même procédé est appliqué à la requête pour la construction de son vecteur de termes. De ce fait, les étapes d'indexation par LSI seront rajoutées à la fin de la phase d'indexation de Terrier.

Pour le processus de recherche de Terrier, on a apporté des modifications dans la phase Matching, en intégrant un nouveau module d'appariement « LSIMatching » qui permet d'une part, la représentation vectorielle réduite de la requête et d'autre part, le calcule de similarité entre un document et une requête représentés dans l'espace réduit ,en utilisant la mesure de cosinus.

3.3.2.2 Conception détaillée :

Dans cette section, nous présentons le détail des modifications apportées aux processus d'indexation et de recherche de Terrier à l'issue de l'intégration du module d'indexation sémantique latente.

a. Processus d'indexation sémantique de LSI-Terrier :

Nous avons étendu le module d'indexation de Terrier par l'intégration du module d'indexation sémantique latente, dans sa dernière phase « Indexer ». Ce module va permettre de construire une nouvelle structure d'index « Matrix Index » qui contient les 3 matrices résultantes de l'indexation par LSI.

La figure suivante illustre le processus d'indexation de LSI-Terrier :

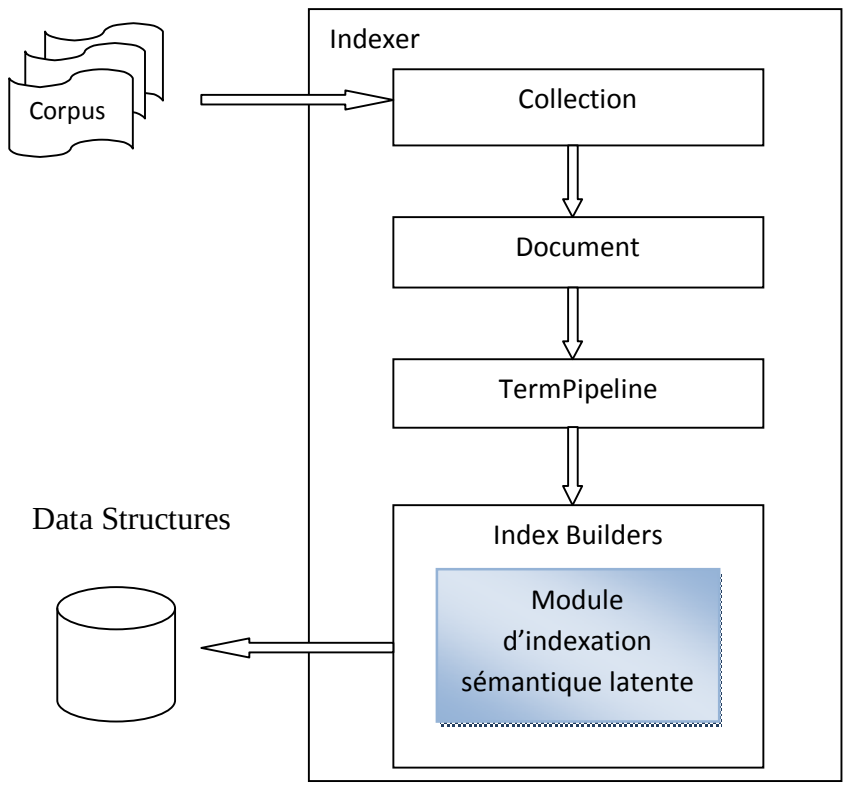


Figure 3.3 : processus d'indexation sémantique de LSI-Terrier

Pour construire cette structure, nous avons étendu la phase « Indexer » en ajoutant un ensemble de classes dans la package « Org.terrier.Structures» qui contient les différentes classes nécessaire pour la construction des différentes structures d'index.

Dans ce qui suit nous donnons un aperçu de l'organisation des classes ajoutées au schéma général du package d'indexation de la plate-forme Terrier, ainsi que leurs descriptions respectives.

Le module d'indexation sémantique latente est organisé comme suit :

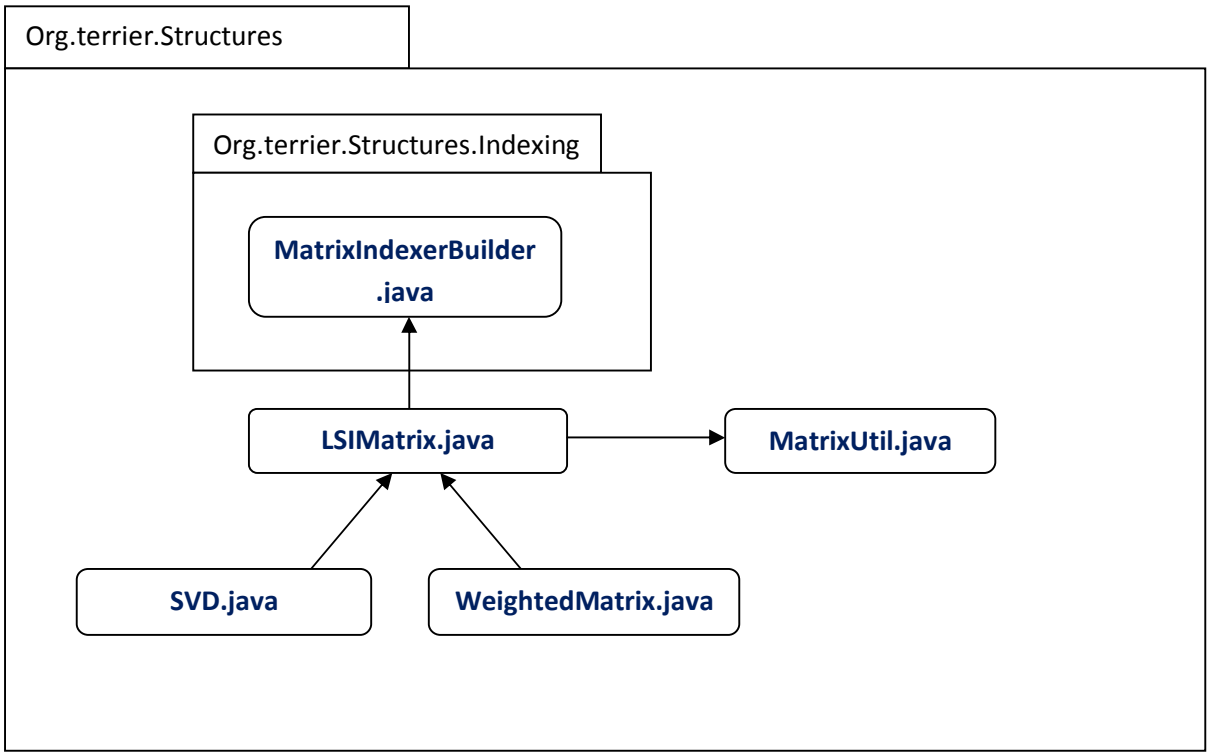


Figure3.4: l'emplacement des classes ajoutées pour l'indexation dans le code source de Terrier

Description des différentes classes :

- *MatrixIndexerBuilder* : cette classe permet de construire la structure *Matrix Index* en instanciant la classe LSIMatrix puis en écrivant l'instance dans le fichier *data.matrix.matrix.*

- *LSIMatrix* : cette classe instanciée les classes SVD et WeightedMatrix et récupère les matrices réduites de T, S et D.

- *WeightedMatrix* : cette classe permet la construction de la matrice pondérée terme-document en utilisant les structures lexicon, document et inverted.

- *SVD* : cette classe permet de décomposer la matrice construite par la classe WeightedMatrix en 3 matrices T, S et D (vus précédemment) et de réduire leurs dimensions (tous cela en utilisant les classes de la librairie JAMA).

- *MatrixUtil* : Comme pour les autres structures, on a ajouté une classe permettant d'afficher le contenu de la structure *Matrix Index* en tapant dans l'invite de commande :

..\LSI-Terrier\bin>trec_terrier --printmatrix

Pour que la construction de la structure *Matrix Index* réussisse, nous avons procédé à la modification d'un certain nombre de classes existantes dans Terrier dont TRECIndexing, Indexer, TrecTerrier, …etc.

b. Processus de recherche sémantique de LSI-Terrier :

Le processus de recherche de Terrier sera étendu par l'intégration d'un module d'appariement dans la phase Matching (nommé « LSIMatching »). Ce module permet, en faisant appel aux matrices : diagonale et document se trouvant dans la structure « Matrix Index », de construire la représentation vectorielle réduite de la requête. Une fois la requête représentée par son vecteur réduit de termes, le module « LSIMatching » permet ensuite d'effectuer l'appariement entre les documents et la requête représentés dans l'espace réduit, en utilisant la mesure de cosinus.

La figure suivante illustre le processus de recherche de LSI-Terrier :

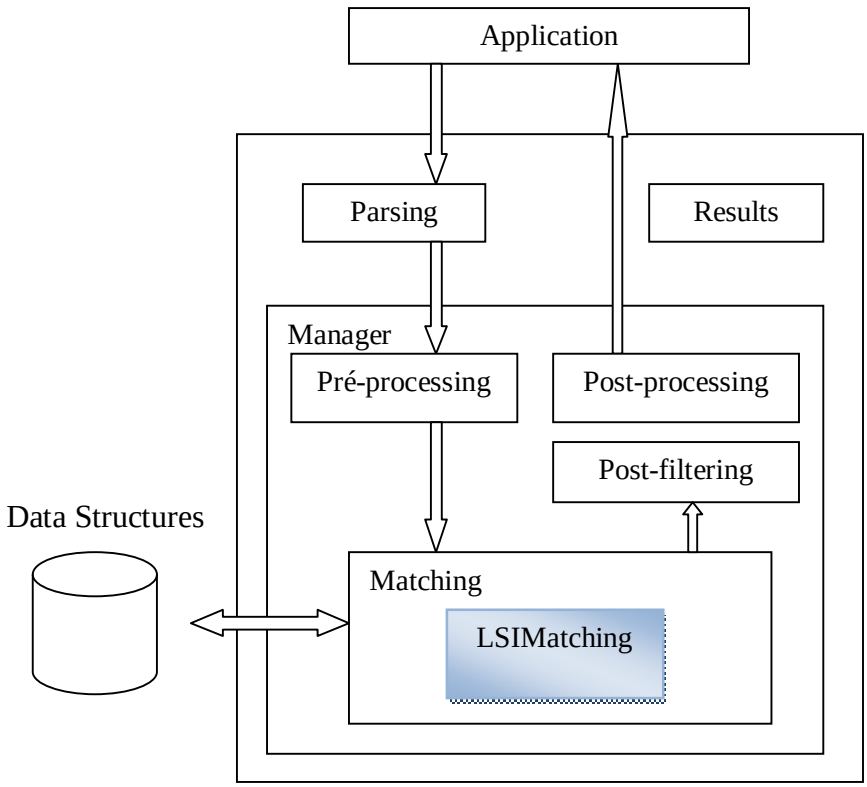


Figure 3.5 : processus de recherche sémantique de LSI-Terrier

Le package « LSIMatching » étendant le module Matching de terrier est composé de deux classes comme suit :

- *LSIRequest*: cette classe permet de construire le vecteur de fréquence de la requête, et de calculer les coordonnées du vecteur de la requête dans l'espace réduit, puis de calculer la similarité entre chaque document et la requête, les résultats sont enregistrés dans un vecteur.

- *Full* : cette classe permet de récupérer le vecteur de similarité de la classe LSIRequest pour pouvoir affecter à chaque score la valeur de similarité lui correspondant, comme ça le résultat d'appariement est prêt pour être traiter par les phases du processus de recherche suivantes.

La figure suivante donne un aperçu du package de recherche étendu :

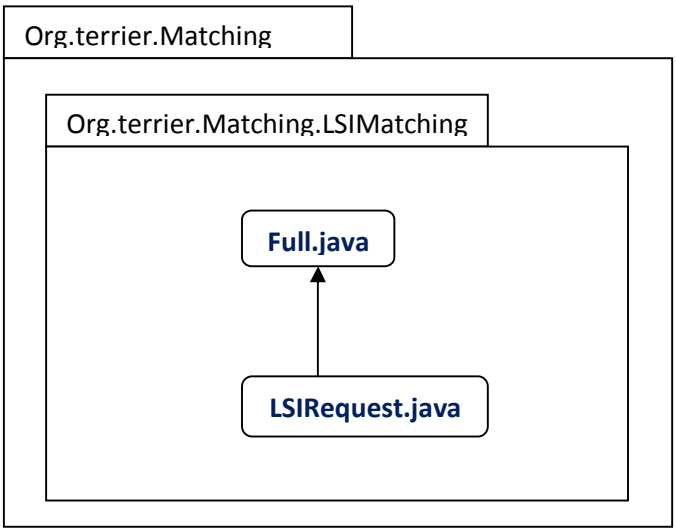


Figure3.6: l'emplacement des classes ajoutées pour la recherche dans le code source de Terrier

Pour pouvoir utiliser LSIMatching pour la recherche, il faut préalablement spécifier le chemin vers ce module d'appariement.

3.4 Conclusion :

Nous avons présenté dans ce chapitre la description détaillée de l'approche d'indexation LSI, puis une architecture d'intégration de cette approche sémantique dans la plate-forme Terrier. Nous avons en particulier décrit les différentes modifications apportées aux modules d'indexation et de recherche de Terrier, le résultat de cette intégration est notre système LSI-Terrier.

Nous consacrons le chapitre suivant à l'implémentation et à l'évaluation de l'approche d'indexation LSI en utilisant la plate-forme LSI-Terrier.

Chapitre 4 :

Implémentation et Tests

4.1 Introduction :

Nous avons décrit dans le chapitre précédent l'approche LSI et son intégration dans la plate-forme Terrier-3.5.

Dans ce chapitre, nous présentons l'environnement technologique de développement de notre approche, puis nous donnons quelques aspects de son implémentation, ainsi que le détail des tests et évaluations que nous avons entrepris sur cette approche à travers le système LSI.

4.2 Présentation de l'environnement du travail :

Le travail que nous avons réalisé a été développé sur un microordinateur ayant les principales caractéristiques suivantes:

- Un microprocesseur Intel ;
- Microprocesseur de fréquence : 2.16 GHZ ;
- RAM de capacité : 2 Go ;
- Disque dur de capacité : 160 Go.

Nous avons travaillé sous le système d'exploitation Microsoft Windows Vista « Édition Familiale Basic », et le langage de programmation orienté objet Java.

4.3 L'environnement de développement :

4.3.1 Présentation du langage de programmation « Java » :

LSI-Terrier est développé entièrement en Java. Le choix de Java s'est imposé vu que l'Open Source de Terrier est entièrement écrit en Java.

Java a été mit au point par Sun Microsystems en 1991. C'est un langage de programmation à usage général, il est multi plate-forme grâce à sa machine virtuelle appelée Java Virtual Machine (JVM), évolué et orienté objet de très haut niveau capable de s'exécuter sur n'importe quelle machine. Parmi les caractéristiques les plus intéressantes de Java, on peut citer:

- Un langage orienté objet ;
- Indépendance vis-à-vis de la plate-forme ;
- Avoir la capacité d'exécuter du code source extérieur de façon sécurisée permettant de compiler le code Java, c'est-à-dire de produire un exécutable capable de fonctionner hors de l'environnement Java.
- La programmation peut se faire pour des exemples simples avec le compilateur Javac, mais pour avoir plus de confort il est préférable d'utiliser un environnement de développement intégré ou IDE, celui que nous avons utilisé est l'environnement Eclipse.

Il existe trois types de programmes en Java : les applications, les applets et les servlets :

- Une application autonome (standalone program) est une application qui s'exécute sous le contrôle direct du système d'exploitation, elle est indépendante de tout logiciel client ou serveur web.
- Une applet est une application qui est chargée par un navigateur et qui est exécutée sous le contrôle de celui-ci.
- Les servlets sont des applications qui s'exécutent au sein d'un serveur d'application.

4.3.2 Présentation de l'environnement Eclipse :

Eclipse IDE est un environnement de développement libre permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation (C++, PHP, …).

Eclipse IDE est principalement écrit en Java. Il permet de développer des programmes Java, par exemple des Applets, des applications. Il permet de compiler tout programme écrit en Java.

Pour utiliser Eclipse, il faut disposer d'un environnement Java ou un JRE (Java Runtime Environnement), qui sert à lire les programmes codés en Java.

Grace aux fonctionnalités dont il dispose, Eclipse se situe comme étant le plus utilisé dans le développement et la compilation de programmes Java aux cotés de Net Beans.

4.3.3 JAMA :

Pour effectuer toutes les opérations sur les matrices nécessaires à la programmation de LSI, dont l'inversion d'une matrice ou sa décomposition en valeurs singulières, nous avons eu recours à une bibliothèque mathématique en Java, en l'occurrence la bibliothèque « JAMA ». La librairie java « JAMA » est une librairie du domaine public, libre d'utilisation et gratuite, qui permet de faire ces opérations et plusieurs autres.

4.4 Implémentation :

Nous présentons dans cette section quelques extraits du code implémentant les différentes classes proposées à l'issue de notre conception.

```
Public class WeightedMatrix {

        ……

        for(int i=0;i<documentindex.getNumberOfDocuments(); i++){

          for(int E=0;E<postings[0].length; E++){

            if(    postings[0][E]==i){

           poid[numterm][i]=wm.LsiScore(postings[1][E],documentindex.getDocumentEntry
           (postings[0][E]).getDocumentLength());
           trouve=true;
           }

            if (!trouve){

                poid[numterm][i]= 0;
           }
          }
        }

        ……
}
```

```
Public class SVD {

        ……

        //transformer la matrice en objet matrix de jama
        Matrix m=new Matrix(Matrice);

        //procéder à SVD de ma matrice
        SingularValueDecomposition svd=new SingularValueDecomposition(m);

        //obtenir les matrices
        Matrix t=svd.getU();
        Matrix s=svd.getS();
        Matrix d=svd.getV();
```

```
        //choisir la valeur de K
        int k=45;

        //les matrices réduites

        sk = s.getMatrix(0, k - 1, 0, k - 1);
        tk = t.getMatrix(0, t.getRowDimension() - 1, 0, k - 1);
        dk = d.getMatrix(0, d.getRowDimension() - 1, 0, k - 1);

        ……
}

Public class LSIRequest {

        ……

        //coordonnées de la requête dans le nouvel k-espace
        Matrix qer=new Matrix(vqf);
        Matrix q=((qer.times(tk)).times(sk.inverse()));

        //calcule de pertinence
        double som=0;
        double[] somcardk=new double [dk.getRowDimension()]; //RACINE somme des carrées des composantes de vk ;
        double somcarq=0;  //RACINE somme des carrées des éléments de q

        double[] sim=new double [dk.getRowDimension()];

        //calcule racine car doc
        for( int i= 0; i < dk.getRowDimension(); i++){

            for( int j = 0; j < dk.getColumnDimension(); j++){

                som=som+(dk.get(i, j)*dk.get(i, j))  ;
            }
            somcardk[i]=Math.sqrt(som);
            som=0;
        }

        som=0;

        //calcule racine car q
        for(int j = 0; j < q.getColumnDimension(); j++){

            som=som+(q.get(0, j)*q.get(0, j))  ;
        }
        somcarq=Math.sqrt(som);
```

```
//calcule de similarité

som=0;

for(int i = 0; i < dk.getRowDimension(); i++){

    for( int j = 0; j < dk.getColumnDimension(); j++){

       som=som+(q.get(0,j )*dk.get(i, j))  ;

     }
     sim[i]=som/((somcarq*somcardk[i]));
     Double num = new Double( sim[i]);

     if (num.isNaN() ){
          sim[i]=0;
     }
     som=0;
}

……

}
```

4.5 Evaluation expérimentale de notre approche :

L'objectif de cette évaluation et de mesurer les performances et la viabilité de notre approche de recherche d'information sémantique « LSI ». Nous présentons dans ce qui suit le cadre d'évaluation (collection de tests et protocole d'évaluation) ainsi que les résultats expérimentaux préliminaires.

4.5.1 Collection de tests :

Les collections de tests ont été traditionnellement utilisées en RI pour évaluer les stratégies de recherche. Plus particulièrement, une collection de référence doit traduire la subjectivité de pertinence des utilisateurs. D'autre part, elle doit contenir une masse d'informations assez importante et variée pour constituer un environnement standard d'interrogation.

Différentes collections de tests sont utilisées en RI. Parmi elles, la collection TIME que nous avons utilisé pour nos expérimentations. Nous allons dans ce qui suit décrire cette collection en détail.

Collection TIME:

Nom :	TIME magazine collection
Collection portant sur :	Divers domaines
Nombre de documents :	423
Nombre de requêtes :	83
Volume de la collection :	1.48 Mo

Tableau4.1: description de la collection TIME.

La TIME est composée de documents au nombre de 423 tirés du magazine TIME de l'année 1963 et d'un nombre de requêtes raisonnable qui est de 83 accompagné d'un fichier de jugement de pertinence.

La collection TIME est une collection portant sur des domaines différents. Elle se caractérise par sa haute efficacité de recherche par rapport aux autres collections typiques.

Un document est identifié comme suit :

/ Numéro du document/

Par exemple : « 032.txt ».

Les documents et les requêtes sont composés de textes simples. En voici un exemple:

Exemple de document (032.txt):

« RUSSIA PARTY TIME THE NEW YEAR'S EVE PARTY WAS GOING FULL BLAST IN THE BANQUET HALL ATOP THE KREMLIN'S PALACE OF CONGRESSES. COMMUNIST BIGWIGS MINGLED WITH DIPLOMATS, MILITARY LEADERS AND STARS OF THE SOVIET CULTURAL ELITE. EVERYONE WAS IN HIGH SPIRITS, INCLUDING SOVIET EX-PRESIDENT KLIMENT VOROSHILOV, 82, WHO BROKE INTO AN IMPROMPTU JIG WHEN THE BAND PLAYED A SNAPPY RUSSIAN MELODY. GENIAL HOST NIKITA KHRUSHCHEV ROARED HIS HEARTY APPROVAL. IT WAS NO OCCASION FOR DISHARMONIOUS WORDS, SO WHEN THE TIME CAME FOR SPEECHES, NIKITA WAVED AWAY THE LATEST 20,000-WORD ATTACK ON HIS POLICIES BY RED CHINA. EVERY FAMILY HAS TROUBLES, KHRUSHCHEV DECLARED, GESTURING AMIABLY AT THE WESTERN DIPLOMATS IN THE CROWD. "YOU JUST GET MARRIED AND YOU WILL SOON NOTICE THAT DIFFERENCES DEVELOP."S FOR HIS DIFFERENCES WITH THE WEST, KHRUSHCHEV HOPED THAT 1963 WOULD SOLVE "URGENT PROBLEMS FRAUGHT WITH NEW CRISES," A BIT OF DOUBLETALK ABOUT

BERLIN THAT COULD FIT ANY EVENTUALITY. THE FIRST WOULD PROBABLY COME AT NEXT WEEK'S CONGRESS OF THE EAST GERMAN COMMUNIST PARTY, WHICH KHRUSHCHEV WILL ATTEND. LAST WEEK, WHEN THE US MOVED 1,500 INFANTRYMEN BY HIGHWAY INTO THE DIVIDED CITY IN A ROUTINE SHIFT OF REGIMENTS, THERE WAS NOT A MOMENT OF OBSTRUCTIONIST DELAY AT THE RUSSIAN CHECKPOINT. READY TO GREET THE FRESH TROOPS WAS A NEW US WEST BERLIN COMMANDANT, MAJOR GENERAL JAMES H .POLK, 51. SAID POLK, IN A MESSAGE TO WEST BERLINERS: "WE ARE HERE TO STAY.»

Exemple de requête :

<NUM : requête>

8: UN TEAM SURVEY OF PUBLIC OPINION IN NORTH BORNEO AND SARAWAK ON THE QUESTION OF JOINING THE FEDERATION OF MALAYSIA.
13: PRECARIOUS TRUCE IN LAOS WHICH WAS BROUGHT UP BY BRITAIN BEFORE THE NATIONS THAT AGREED ON THE TRUCE IN GENEVA LAST YEAR.

Des jugements de pertinence sont associés aux requêtes selon le format suivant :

« Numéro de la requête / Numéro du document / 0 / 0 »

Exemples de jugement de pertinence :

1 268.txt 0 0
1 288.txt 0 0
1 304.txt 0 0
4 370.txt 0 0
4 378.txt 0 0
32 279.txt 0 0
68 007.txt 0 0
68 008.txt 0 0
68 031.txt 0 0

4.5.2 Protocole d'évaluation :

Nous avons effectué notre évaluation en utilisant le système de RI LSI-Terrier selon le Protocol d'évaluation TREC :

Pour chaque requête, les documents restitués par le système sont examinés et des précisions sont calculées à différents points (à 1, 2, 3, 4, 5, 10, … 200 premiers documents restitués). La précision exacte découle de ces précisions. La précision à x (exemple précision à 5) définie le taux de documents pertinents parmi les x premiers documents retrouvés.

Une précision moyenne MAP est ensuite calculée pour chaque requête. Il s'agit de la moyenne des précisions de chaque document pertinent pour cette requête. La précision d'un document est la précision à x, tel que x est le rang de ce document dans l'ensemble des documents pertinents retrouvés. Finalement, la précision moyenne pour l'ensemble des requêtes est calculée permettant d'obtenir une mesure de la performance globale du système.

4.5.3 Résultats expérimentaux :

Pour évaluer la performance de l'approche d'indexation sémantique latente, nous avons réalisé une série d'expérimentations dans le but de comparer l'indexation par LSI par rapport à l'indexation classique.

Sachant que pour avoir de bons résultats en indexant par LSI, il faut choisir la valeur optimale K de la dimension réduite. Pour cela, nous avons préalablement effectué un ensemble de tests sur la collection TIME en utilisant le système LSI-Terrier. Ces tests ont pour objectif de déterminer la valeur optimale de k.

Les résultats obtenus montrent que la valeur de k égale à 180 donne les meilleurs résultats. C'est donc cette valeur que nous retiendrons pour les tests comparatifs suivants.

Après avoir trouvé la meilleure valeur de la dimension réduite, on compare les résultats de l'indexation de la collection TIME par LSI-Terrier aux résultats de l'indexation de cette même collection par le système Terrier standard avec un schéma de pondération classique TF_IDF.

4.5.3.1 Evaluation de l'approche d'indexation sémantique latente:

Le tableau suivant récapitule les résultats de l'évaluation obtenus pour les deux systèmes LSI-Terrier et Terrier classique:

Information :	Modèle classique Terrier 3.5 (pondération TF_IDF)	Modèle sémantique LSI-Terrier (pondération TF_IDF)
Number of queries Retrieved Relevant Relevant retrieved	83 18604 323 125	83 16663 323 111
Average Precision: R Precision :	**0.0090** **0.0025**	**0.0105** **0.0034**
Precision at 1 : Precision at 2 : Precision at 3 : Precision at 4 : Precision at 5 : Precision at 10 : Precision at 15 : Precision at 20 : Precision at 30 : Precision at 50 : Precision at 100 : Precision at 200 :	0.0000 0.0000 0.0041 0.0030 0.0024 0.0061 0.0049 0.0043 0.0033 0.0056 0.0057 0.0056	0.0000 0.0061 0.0041 0.0030 0.0049 0.0049 0.0073 0.0067 0.0061 0.0076 0.0070 0.0067

Precision at 0%:	0.0250	0.0304
Precision at 10%:	0.0232	0.0292
Precision at 20%:	0.0172	0.0202
Precision at 30%:	0.0166	0.0155
Precision at 40%:	0.0147	0.0143
Precision at 50%:	0.0116	0.0115
Precision at 60%:	0.0042	0.0044
Precision at 70%:	0.0033	0.0034
Precision at 80%:	0.0032	0.0034
Precision at 90%:	0.0029	0.0034
Precision at 100%:	0.0029	0.0034
Average Precision:	**0.0090**	**0.0105**

Tableau 4.2: contenus des fichiers d'évaluation (fichiers .res).

Ces résultats montrent que l'approche d'indexation sémantique latente offre une précision moyenne et une précision réelle plus importante que l'indexation classique. En effet, la précision moyenne est passée de 0.0090 pour le modèle classique à 0.0105 pour le modèle sémantique. La R-Precision est passée de 0.0025 à 0.0034 dans le modèle LSI-Terrier.

Voici deux graphes récapitulatifs des résultats présentés dans ce tableau pour les deux modèles étudiés :

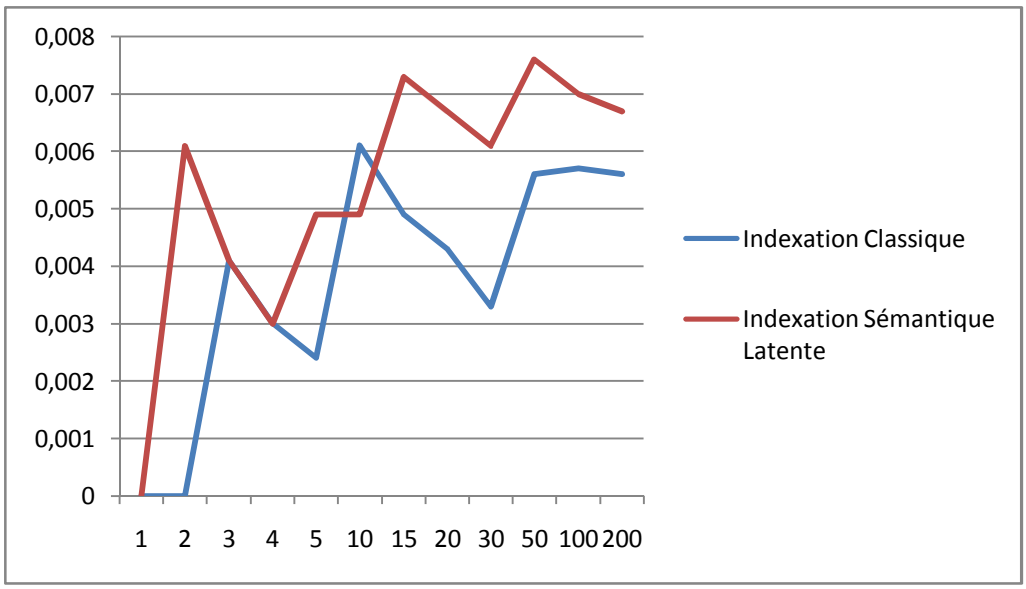


Figure4.1: Précision at x

La comparaison des courbes montre une amélioration certaine de la précision du modèle sémantique par rapport au modèle classique pour les précisions : précision at 2, précision at 5 et précisions comprises entre précision at 15 et précision at 200.

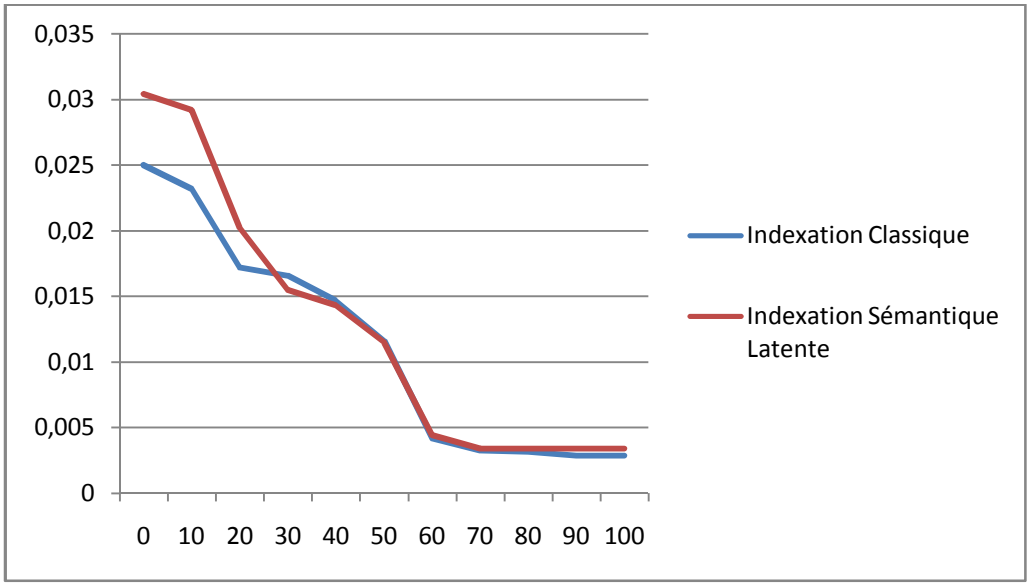


Figure4.2: Précision at x%

Voici une figure illustrant l'augmentation de la précision moyenne et de la R-precision :

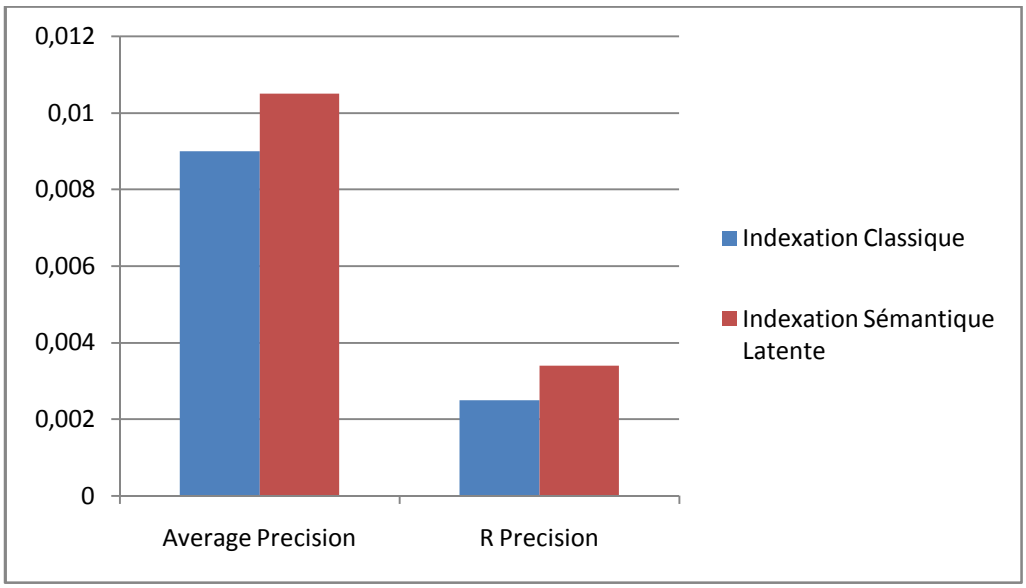


Figure4.3: Augmentation de la précision moyenne et de la R-precision

4.6 Conclusion :

Dans ce chapitre, nous avons présenté les détails de l'implémentation ainsi que les expérimentations et résultats associés à notre approche d'indexation sémantique latente en général et au système LSI-Terrier en particulier. Selon les résultats obtenus, il résulte une amélioration de la précision moyenne et de la R-précision lorsqu'on utilise une indexation sémantique par LSI. Pour pouvoir conclure de façon certaine sur l'efficacité de cette approche, il faut la tester sur des collections de tailles réelles.

Conclusion générale

Les travaux présentés dans ce mémoire se situent dans le contexte général de l'utilisation de la sémantique pour la représentation de l'information dans les systèmes de RI. Nous nous positionnons plus particulièrement dans le cadre de l'indexation sémantique latente.

L'objectif de l'approche d'indexation par LSI est de transformer une représentation standard par mots-clés, en une représentation par concepts qui permet de « meilleurs» résultats. La technique LSI cherche à traiter la dépendance entre des termes afin que des termes « similaires » soient représentés par un représentant commun « concept ». Pour ce faire, partant de l'espace vectoriel de tous les termes d'indexation, le modèle LSI construit un espace d'indexation de taille réduite *k*, par application de la décomposition en valeurs singulières (SVD) de la matrice termes-documents ayant en ligne les termes et en colonne les documents.

Chaque vecteur document est au final représenté dans l'espace *k*-dimensionnel réduit des termes non bruités. La requête utilisateur est aussi représentée par un vecteur dans l'espace *k*-dimensionnel. Une mesure de similarité est ensuite calculée entre le *k*-vecteur requête et chacun des *k*-vecteurs documents de la collection.

Une fois les documents et la requête sont représentés dans l'espace réduit de concepts, une valeur de similarité est calculée entre la requête et chaque document. A l'issue de la recherche, le système sélectionne les documents pertinents même s'ils ne contiennent aucun mot de la requête.

Le principal apport de ce mémoire est l'intégration de l'approche LSI dans la plate forme de RI Terrier. Ainsi, nous avons testé notre approche sur la collection TIME en utilisant le système LSI-Terrier résultant de l'intégration de LSI dans Terrier. L'évaluation des résultats montre une amélioration de la précision moyenne et la R-Précision par rapport aux résultats obtenus en utilisant la plate-forme Terrier.

Cette approche à monter une amélioration pour un corpus de petite ou moyenne taille. Néanmoins, des perspectives d'amélioration restent à étudier, notamment :

1. Il serait judicieux d'effectuer de nouvelles évaluations sur un corpus plus important pour pouvoir conclure d'une façon certaine sur l'efficacité de cette approche.

2. Le choix de la valeur de K (la valeur de la dimension réduite) est déterminé expérimentalement, et dépend donc de la collection utilisée. Il serait alors intéressant de réfléchir à une fonction générale qui permettrait de calculer cette valeur optimale, en amont des expérimentations.

3. Enfin, vu que la méthode SVD est trop complexe, (nécessitant beaucoup de calculs et donc de temps et d'espace mémoire pour de grandes collections de documents), il serait intéressant de réfléchir à une optimisation des calculs qui rendrait LSI, de ce point de vue, plus efficace.

Bibliographie

- **[Amirouche, 2008]:** Fatiha AMIROUCHE, Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets, Thèse de Doctorat en Informatique de l'université de Toulouse, 2008.

- **[Baeza-Yates & al., 99]** Ricardo A. Baeza-Yates, Berthier A. Ribeiro-Neto: Modern Information Retrieval ACM Press / Addison-Wesley 1999.

- **[Bai & al., 2006]** Jing Bai, Jian-Yun Nie, Guihong Cao. "Context Dependent Term Relations for IR". EMNLP 06. 2006.

- **[Baziz & al., 2003]** Mustapha Baziz, Nathalie Aussenac-Gilles, Mohand Boughanem, "Exploitation des Liens Sémantiques pour l'Expansion de Requêtes dans un Système de Recherche d'Information " Dans : XXIème Congrès INFORSID. Pages : 121-134, 2003.

- **[Baziz, 2005]** M. Baziz, "INDEX ATION CONCEPTUELLE GUIDEE PAR ONTOLOGIE POUR LA RECHERCHE D'INFORMATION " thèse de doctorat de l'université de Paul Sabatier. 2005.

- **[Belew, 89]** R. K. Belew, Adaptative Information Retrieval: using a connectionist representation to retrieve and learn about document In proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Boston (USA), pages: 11-20, 1989.

- **[Berger & al., 99]** A. Berger, J. Lafferty Information retrieval as statistical translation in proceedings of ACM-SIGIR 99, pages: 222-229, 1999.

- **[Berry & al., 94]** M.W. Berry, S. T. Dumais, G. W. O' Brien, Using linear algebra for Intelligent Information Retrieval, 1994.

- **[Bhogal & al., 2006]** J. Bhogal, A. Macfarlane, P. Smith. "A review of ontology based query expansion" Information Processing and Management 43 pages: 866–886. 2006.

- **[Boughanem, 92]** M. Boughanem Systèmes de recherche d'information d'un modèle classique à un modèle connexionniste Thèse de l'université Pau l. Sabatier de Toulouse. 1992.

- **[Chappelier & al]** Jean-Cédric Chappelier, Emmanuel Eckard « Rôle de la matrice d'information et pondération des composantes dans les noyaux de Fisher pour PLSI », Laboratoire d'Intelligence Artificielle, École Polytechnique Fédérale de Lausanne.

- **[Crestani & al., 94]** F. Crestani, C. J VanRijsberge Probability Kinematics in information retrieval in proceedings of the 18th Annual International AC M-SIGIR Conference on Research and Development in Information Retrieval, SEATLE Washington, pages: 291-299, 1994.

- **[Crestani, 95]** F. Crestani, Implementation and evaluation of relevance feed-back device based on neural networks, from natural to artificial neural computation. Proceedings of International workshop on artificial neural net works, Ed. Springer Verlag, pages: 597-604, 1995.

- **[Deerwester & al., 90]** Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas and Richard A. Harshman, 1990. "Indexing by Latent Semantic Analysis". In Journal of the American Society of Information Science, Vol. 41:6, pages: 391-407.

- **[Denos, 97]** Denos N. Modélisation de la pertinence en recherche d'information: modèle conceptuel, formalisation et application. Thèse de Doctorat de l'Université Joseph Fourier-Grenoble I, 1997

- **[Gaussier & al., 97]** E. Gaussier, G. Grefenstette, et M. Schulze. Traitement du langage naturel et recherche d'informations : quelques expériences sur le français. In Premières Journées Scientifiques et Techniques du Réseau Francophone de l'Ingénierie de la Langue de l'AUPELF-UREF, 1997.

- **[Gaussier & al., 2000]** E. Gaussier, G. Grefenstette, D. Hull, et C. Roux. Recherche d'information en français et traitement automatique des langues. revue Traitement Automatique des Langues (TAL), 41(2) :473–494, 2000.

- **[Gaussier & al., 2003]** Eric Gaussier, Christain Jacquemin, Pierre Zweigenbaum "Traitement automatique des langues en recherche d'information" Chapitre 2 dans Assistance intelligente à la recherche d'information parru aux éditions Lavoisier.2003

- **[Grossman & al., 98]** David Grossman and Ophir Frieder, Ad Hoc Information Retrieval: Algorithms and Heuristics, Kluwer Academic Publishers, 1998.

- **[Hiemstra 2001]** D. Hiemstra, Using language models for information retrieval. PhD Thesis, UNIVERSITY OF TWENTE, 2001

- **[Hofmann & al., 99]** Hofmann T., « Probabilistic Latent Semantic Indexing », Proc. of 22th Int. Conf. on Research and Development in Information Retrieval, pages: 50-57, 1999.

- **[Hofmann & al., 2000]** Hofmann T., « Learning the Similarity of Documents: An Information-Geometric Approach to Document Retrieval and Categorization », Advances in Neural Information Processing Systems, vol. 12, pages: 914-920, 2000.

- **[Hofmann & al., 2001]** Hofmann T., « Unsupervised learning by probabilistic latent semantic analysis », Machine Learning, vol. 42, n° 1, pages: 177-196, 2001.

- **[Husbands & al., 2000]** Husbands, P, Simon, H. et Ding, C. (2000) "On the Use of the Singular Value Decomposition for Large Scale Information Retrieval", Proceedings of CIR'00.

- **[Ingwersen., 92]** P. Ingwersen. Information retrieval interaction. London, Taylor Graham, 1992.

- **[Järvelin & al., 2004]** Airio, E, Järvelin, K, Saatsi, P, Kekäläinen, J, & Suomela, "CIRI - An Ontology-based Query Interface for Text Retrieval". In Hyvönen, E, Kauppinen, T, Salminen, M, Viljanen, K & Ala-Siuru, P, eds. Web Intelligence. Helsinki: Finnish Artificial Intelligence Society. Pages: 73-82. 2004.

- **[Khan, 2000]** Latifur R. Khan, "Ontology -based Information Selection, Phd Thesis", Faculty of the Graduate School, University of Southern California. August 2000.

- **[Kohonen 89]** T. Kohonen, Self-Organization and Associative Memory 3RD Edition, Springer Verlag, Berlin, pages: 80-89, 1989.

- **[Kwok 89]** K. L. Kwok A neural network for probabilistic information retrieval in proceeding of ACM-SIGIR' 89, pages: 21-30, 1989.

- **[Kwok 95]** K. L. Kwok, A network approach to probabilistic information retrieval. ACM Transactions on Information Systems, pages: 324-353, 1995.

- **[Kwok & al., 99]** K. L. Kwok, L. Grunfeld, M. Chan, TREC-8 adhoc, query and filtering track experiments using PIRCS. Proceedings of TREC-8, pages: 129-137, 2000.

- **[Landauer & Dumais., 97]** Landauer T. K., Dumais S. T. (1997). A solution to Plato's problem: the Latent Semantic Analysis theory of acquisition, induction and representation of knowledge. Psychological Review. Pages: 104, 211-240.

- **[Landauer & al., 98]** Landauer, M. Littman Fully Automatic cross-language document retrieval using latent semantic indexing. In Proceedings of the Sixth Annual Conference of UW Center for the New OED and Text Research, pages: 31-38, 1998.

- **[Lefèvre, 2000]** Lefèvre A. la recherche d'information, du texte intégrale au thesaurus, Paris, Hermès, 253p.

- **[Littman & al., 98]** M. Littman, S. Dumais, K. Landauer Automatic cross-language information retrieval using latent semantic indexing. In Cross-Language information retrieval, edited by G. Grefenstette, pages: 51-62, 1998.

- **[Luhn, 57]** Luhn, H. A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development 4, 1(1957),pages: 309–317.

- **[Luhn, 58]** Luhn, H. The automatic creation of literature abstracts. IBM Journal of Research and Development 24, 2 (1958), 159–165.

- **[Maniez, 2002]** Maniez J. actualité des langages documentaires ; fondements théoriques de la recherche d'information, ABDS, Paris, 2002.

- **[Maron & al., 60]** Maron, M., and Kuhns, J. On relevance, probabilistic indexing and information retrieval. Journal of the Association for Computing Machinery 7 (1960), pages: 216–244.

- **[Mihalcea, 2004]**: Mihalcea, R.: Co-training and Self-training for Word Sense Disambiguation. In: Proc. CoNLL, pp. 33–40 (2004).

- **[Oard & al., 98]** D. Oard, G. Marchioni, A conceptual framework for text filtering, Technical Report CS-TR-3643, university of Marylan, Institute for Advanced computer Studies, April 1998.

- **[Ponte & al., 98]** J. M. Ponte, W. B. Croft, A language modeling approach to information retrieval. Proceedings of ACM SIGIR 98, pages: 40-48, 1998.

- **[Rijsbergen, 79]** Van Rijsbergen C.J. Information Retrieval, Butterworths, londres, 1979.

- **[Robertson & al., 76]** Robertson, S. E., & Sparck Jones, K. (1976). Relevance weighting of search terms. Journal of the American Society for Information Science, 27, pages: 129–146.

- **[Robertson, 94]** ROBERTSON S. E., WALKER S., « Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval », Proceedings of SIGIR 1994, pages: 232-241, 1994.

- **[Robertson & al., 95]** S. Robertson, S. Walker, M. Sparck Jones, and al. Okapi at trec-3. In Second Text Retrieval Conf (TREC-3), pages: 109-26, 1995.

- **[Robertson & al., 97]** S. E. Robertson and S. Walker. On relevance weights with little relevance information. In Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval, pages: 16–24. ACM Press, 1997.

- **[Salton, 68]** Salton, G. Automatic Information Organization and Retrieval. New York: McGraw.Hill Book Company, 1968.

- **[Salton, 71]** G. Salton, "A Comparison between manual and automatic indexing methods". Journal of the American Documentation, 20(1), pp. 6171, 1971.

- **[Salton & al., 71]** G. Salton. The SMART Retrieval System Experiments in Automatic Document Processing. Prentice-Hall Inc, NJ, 1971.

- **[Salton & al., 83]** Salton, G., E.A. Fox, H. Wu. Extended Boolean information retrieval system. CACM 26(11), pp. 1022-1036, 1983.

- **[Salton, 91]** G. Salton. The smart information retrieval system after 30 years - panel. SIGIR, pages: 356–358, 1991

- **[Saracevic, 70]** Saracevic T. "the concept of "relevance" in information science: a historical review", dans T Saracevic (dir), Introduction to Information science. R.R. Bowker, New York, pages: 111-151, 1970.

- **[Singhal & al., 96]** A. Singhal, C. Buckley, M. Mitra. Pivoted document length normalization. In Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. Zurich, Switzerland .Pages: 21 - 29. 1996

- **[Voorhees, 93]** E. Voorhees, “Using WordNet to Disambiguate Word Senses for Text Retrieval” Proceedings of the 16thAnnual Conference on Research and Development in Information Retrieval, SIGIR'93, Pittsburgh, PA, 1993

- **[Wong & al., 85]** Wong, S., Ziarko, W. et Wong, P. (1985). Generalized vector spaces model in information retrieval. In Proc. of the 8th ACM-SIGIR conference, pages: 18 25. Montreal, Quebec.

- **[Woods, 97]:** William A. Woods. 1997. Conceptual indexing: A better way to organize knowledge. Technical Report SMLI TR-97-61, Sun Microsystems Laboratories, Mountain View, CA, April. ww.sun.com/research/techrep/1997/abstract-61.html.

Annexe 1 :

La plateforme de RI « Terrier »

1.1 Présentation de la plate-forme du SRI Terrier :

Terrier, TERabyte RetrIEveR est un moteur de recherche robuste et efficace qui offre la plate-forme idéale pour travailler avec des grandes quantités de données « jusqu'à 25 millions de documents ». Développé par le département informatique de l'université Glasgow de Scotland. Il est utilisé avec succès pour la recherche Ad-hoc, la recherche web et la recherche inter-langages dans des environnements centralisés et distribués.

Terrier est un logiciel open sources entièrement écrit en java, selon j'étude comparative sur les SRI en open sources effectuée par Ricardo Baeza-Yates, Terrier fait partie des trois meilleurs systèmes dans un environnement java, les deux autres sont MG4J et Lucene.

Comme tous les moteurs de recherche, Terrier possède les principales facettes suivantes :

- Indexation : permet l'extraction des termes des différents documents du corpus.
- Recherche : permet de générer des résultats aux requêtes formulées par les utilisateurs.
- Evaluation des résultats de la recherche: permet de mesurer le degré de pertinence des résultats de la recherche aux requêtes formulées par l'utilisateur.

1.2 Installation de Terrier sur Windows :

Pour pouvoir utiliser Terrier, il est nécessaire d'installer une JRE (1.6.0 au plus). La JRE ou la JDK peuvent être téléchargés sur le site de Java : http://www.oracle.com/technetwork/java/javase/downloads/index.html

Après avoir téléchargé une copie de terrier version 3.5 sur la page d'accueil du projet Terrier, créé un dossier dans l'une des partitions du disque dur et extrait le contenu de cette copie dans ce dernier. Le site de Terrier est le suivant :
http://www.terrier.org

1.3 La structure des répertoires de Terrier :

Terrier contient un ensemble de répertoires et ils sont structurés comme suit :

- ü bin/ : contient les scripts nécessaires pour démarrer Terrier.
- ü doc/ : contient la documentation relative à terrier.
- ü ect/ : contient les fichiers de configurations de Terrier, le fichier terrier.properties.sample contient la plupart des propriétés de configuration de terrier.
- ü lib/ : contient les classes compilées de Terrier et les différentes librairies externes utilisées par Terrier.
- ü share/ : contient la liste des mots vides (stop word list) et des exemples des documents à tester dans Terrier.
- ü scr/: contient le code source de Terrier.
- ü var/index : contient les structures de données: fichier inverse, fichier lexicon, index direct et document index.
- ü var/results : contient les résultats de la recherche.
- ü license/ : contient les informations sur la licence des différents composants inclus dans Terrier.

Ces différents répertoires sont illustrés dans la figure suivante:

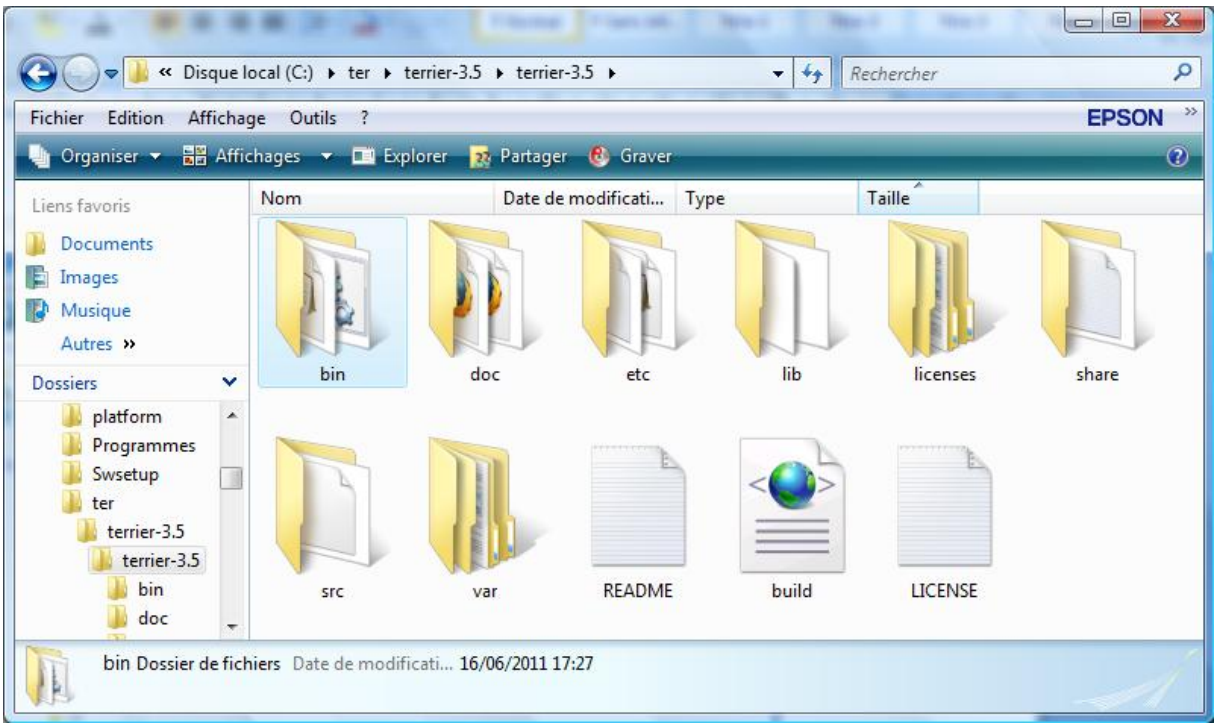


Figure 1.1: Structure de Terrier 3.5.

1.4 Fonctionnalités de Terrier :

Ci-dessous, nous donnons une liste succincte des fonctionnalités offertes par Terrier :

Fonctionnalités générales :

- Indexation pour les formats de fichier de bureau, et pour les collections de recherche couramment utilisés TREC (par exemple des CD TREC 1-5, WT2G, WT10G, GOV, GOV2, Blogs06, Blogs08, ClueWeb09…etc).
- De nombreux modèles de pondération de document, tels que la divergence des paramètres, ainsi que beaucoup de modèles de pondération aléatoire, BM25 Okapi et la modélisation du langage.
- Langage de requête conventionnel pris en charge, y compris les phrases et termes utilisés dans les balises.
- Traitement d'indexation de collections de documents de texte intégral à grande échelle, dans une architecture centralisée à au moins 50 millions de documents.
- Utilise des API d'interrogation Modulaire et d'indexation ouverte, pour permettre une extension faciles pour des applications utilisateurs spécifiques.

Fonctionnalités d'Indexation :

- Indexation de collections de documents diverses, telles que les collections d'essai TREC.
- Indexation de documents de différents formats, tels que HTML, PDF ou Microsoft Word, Excel et PowerPoint.
- Prise en charge pour les encodages différents types de documents(UTF), pour faciliter la recherche multilingue.
- Soutien à la recherche des fichiers à indexer par HTTP, avec intranet permettant d'effectuer facilement une recherche.

Fonctionnalités de Recherche :

- Fournit au bureau, une ligne de commande et des interfaces d'interrogation.
- Fournit des installations d'interrogation standard, ainsi que d'extension des requêtes (feedback pseudo-pertinence).
- Peut être appliqué dans des applications interactives, telles que le « Desktop Search » inclus.
- Fournit de nombreux modèles standards de pondération, tels qu'Okapi BM25, DFR, ou le nouveau modèle de pondération du langage et TF-IDF…
- Le traitement flexible des termes dans un pipeline de composants, tels que les stop-words et la réduction de formes dérivées.

Fonctionnalités d'Expérimentation :

- Gère toutes les collections actuellement disponibles test TREC.
- Facilité d'évaluer de nombreux paramètres, ou de nombreux modèles de pondération en traitement batch.
- Des outils d'évaluation intégré pour une utilisation avec TREC ad-hoc et connus du point de recherche des résultats de recherche, de produire de précision et du rappel des différentes mesures.

1.5 Architecture de Terrier :

Nous donnons ci-après les principaux processus de Terrier :

1.5.1 Le processus d'indexation dans Terrier :

L'indexation dans Terrier est divisée en quatre procédures et à chaque procédure des classes java peuvent être ajoutés pour la personnalisation du système.

Les quatre procédures sont :

1. Splitter la collection de document : consiste à parcourir l'ensemble du corpus reçu en entrée par Terrier et envoyer chaque document à l'étape suivante.
2. Extraction des termes (Tokenize Document) : qui consiste à parser chaque document reçu et en extraire les différents termes.
3. Traitement des termes extrait avec TermPipelines : consiste en l'élimination des mots vides et la lemmatisation des termes.
4. La construction de l'index.

La figure ci-dessous donne un aperçu de l'interaction des principaux composants impliqués dans le processus d'indexation :

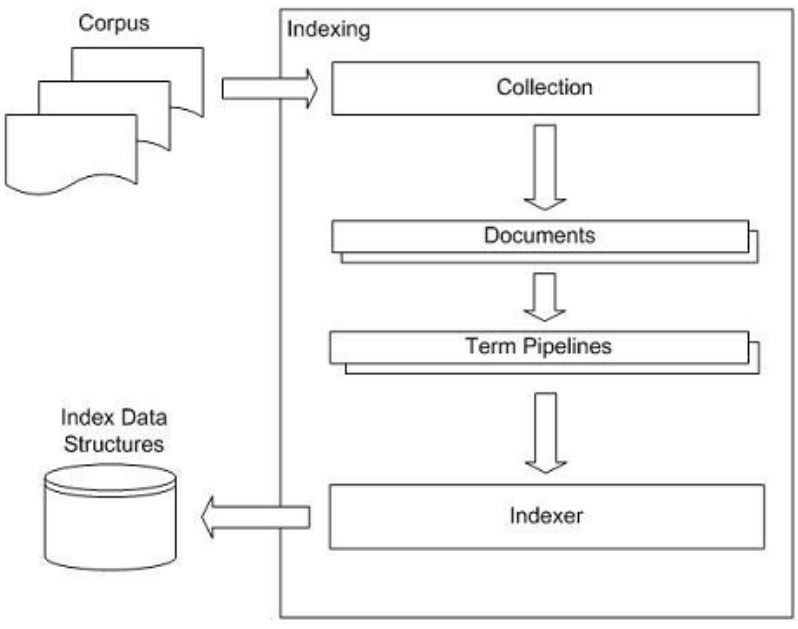


Figure 3.1 : processus d'indexation de Terrier

a. Collection:

Cette composante est une Interface dans le package org.terrier.indexing, elle représente un concept fondamental de l'indexation avec terrier. Son rôle est de Splitter une collection (préalablement construite à partir de corpus) en documents. Pour cela elle fait appel à plusieurs de ses méthodes telles que :

- public Document getDocument ();
- public boolean nextDocument();
- public String getDocid();
- public boolean endOfCollection ();

Plusieurs Classes implémentent l'interface collection selon le format du document :

- SimpleFileCollection: PDF, TXT, HTML,…..
- TrecCollection
- SimpleXMLCollection : XML
- SimpleMedlineXMLCollection
- TRECUTFCollection
- WARC018Collection
- WARC09Collection.

b. Document :

Ce composant est une interface qui se trouve dans le package org.terrier.indexing. cette interface englobe une concept important, celui de Document. Son rôle est de parcourir les documents et d'en extraire les *Termes* en utilisant *Tokeniser*.

Plusieurs méthodes sont utilisées :

- public String getNextTerm();
- public boolean endOfDocument();

Plusieurs Parseurs sont disponibles selon le format du document

- HTMLDocument
- FileDocument
- MSExcelDocument

c. Term Pipeline :

C'est une interface qui se trouve dans le package org.terrier.terms. Son rôle consiste en le traitement des termes extraits. Il peut transformer les termes ou supprimer des termes qui ne devraient pas être indexés. En définitive il :

- Elimine les mots vides (Stopwords)

- Lemmatise les termes selon la langue : pour l'anglais l'algorithme de lemmatisation utilisé est celui de Porter (PorterStemmer).

d. Indexer :

Ce composant est responsable de la gestion du processus d'indexation et entre autre de la construction de l'index. Il instancie les termes pipelines et initialise les *Builders* qui sont en charge de l'écriture de l'index sur le disque dans la structure de données appropriée.

Terrier offre deux types d'indexeur : *BasicIndexer, BlockIndexer*.

e. Data Structures (Structures d'Index):

- **Direct Index:**
  - Id document
  - Longueur document
  - Byte offset dans Direct Index

- **Document Index:**Index
  - Id Terme
  - Fréquence Terme
  - #Filds (#of fields bits)

- **Inverted Index :** Fichier inverse:
  - Id Terme
  - Id document
  - Fréquence terme dans le document
  - #Filds (# of fields bits)

- **Lexicon :** Informations sur chaque terme de la collection:
  - Terme
  - Id terme
  - Nombre documents qui contiennent le terme
  - Fréquence terme dans la collection
  - Offset terme dans le fichier inverse

- **Meta Index :** informations additionnelles (méta informations) sur chaque document comme son *docno* ou son URL.

Un corpus est représenté sous la forme d'un objet de Collection. Les données de texte brut seront représentées sous la forme d'un objet Document.

L'indexeur est chargé de gérer le processus d'indexation. Il parcourt les documents de la collection et envoie chaque terme trouvé par un composant TermPipeline.

Un TermPipeline peut transformer ou supprimer des termes qui ne doivent pas être indexés. Un exemple d'une chaine TermPipeline est : termpipelines=mots vides, PorterStemmer, qui supprime les termes du document en utilisant les objets mot vides, et applique ensuite un algorithme de PorterStemmer en anglais qui radicalise les termes (PorterStemmer).

Une fois que les termes ont été traités par le TermPipeline, ils sont regroupés et les structures de données suivantes sont créées par leurs DocumentBuilders correspondant: DirectIndex, DocumentIndex, Lexicon, et InvertedIndex.

L'indexation en Terrier peut être configurée on changeant les propriétés approprié dans le fichier etc\terrier.properties. Chaque procédure citée auparavant possède ces propres propriétés.

1.5.2 Le processus de recherche de Terrier :

Un des principaux objectifs de Terrier est de faciliter la recherche d'information. Terrier implémente pour cela un certain nombre de fonctionnalité de recherche qu'on a cité auparavant, qui offre un large choix pour le développement de nouvelles applications et pour les testes en RI.

En effet, Terrier offre un grand choix de modèles de pondération, il propose aussi un langage de requête avancée. Une autre fonction de recherche très importante intégrée dans Terrier est l'automatisation de l'expansion de requête.

La figure suivante donne un aperçu de l'interaction des composants Terrier dans la phase de recherche :

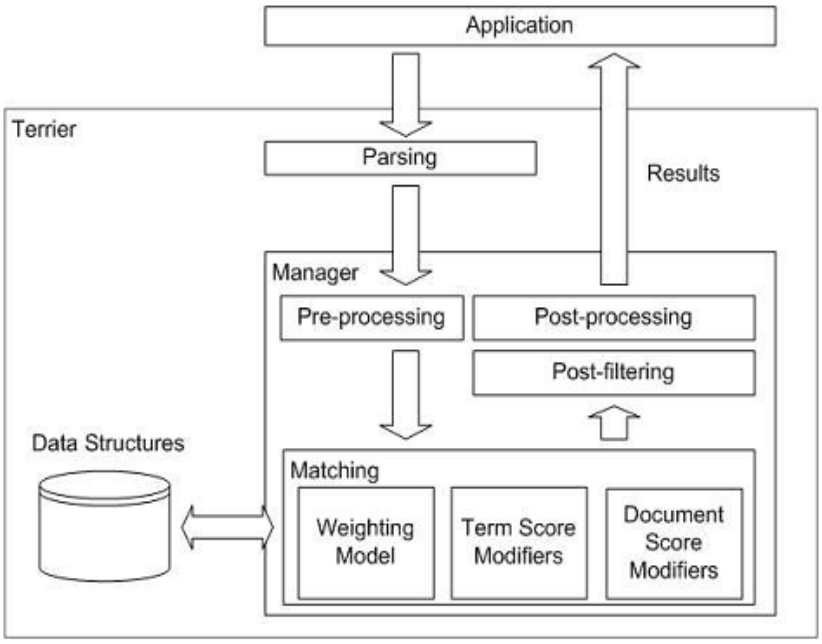


Figure3.2 : processus de recherche de Terrier

a. Query:

C'est une Classe abstraite dans le package org.terrier.querying.parser qui modélise la requête. Un objet Query est crée pour chaque requête.

Terrier supporte Trois modèles de requête:

- *SingleTermQuery*: modèle de requête avec un seul terme.
- *MultiTermQuery* : modèle de requête avec plusieurs termes.
- *FieldQuery* : Terme qualifié par un champ (Exemple: dans le titre du document)

b. Manager :

Ce module est chargé de la gestion de la recherche il fait appel à quatre sous modules :

1. **Pre-processing**: Appliquer Tokeniser et TermPipeline
2. **Matching**: Déterminer les documents qui répondent à la requête en initialisant:
   a. **WeightingModels :** Assigner un score pour chaque terme de la requête dans le document (Pondération). Plusieurs Modèles de pondération sont disponibles dans org.terrier.matching.models dont : TF_IDF, BM25.
   b. **DocumentScoreModifiers :** Modifier le score d'un document en fonction du langage de la requête (ou expansion de la requête)
3. **Post-filtring**: Filtrer les documents pertinents pour la requête.
4. **Post-processing**: Reclasser les documents pertinents s'il y a une expansion de la recherche.

c. Set-Results :

Se charge de récupérer les résultats de Post-processing et de retourner les documents selon leur degré de pertinence.

Une application, comme par exemple le Desktop de Terrier ou l'application TrecTerrier, émet une requête au système Terrier.

Dans une première étape, la requête sera analysée et une instanciation d'un objet de requête aura lieu.

En suite, la requête sera remise à la composante Manager. Le gestionnaire d'une part prétraite la requête, en l'appliquant au TermPipeline.

Après le prétraitement, la requête sera remise à la composante responsable de son traitement.

Cette dernière est responsable de l'initialisation du « WeightingModel » (modèle de pondération) approprié et « DocumentScoreModifiers ». Une fois tous ces éléments instanciés le calcule de score de documents a l'égard de la requête commencera.

Ensuite, le post-traitement est post-compilation ont lieu.

Enfin, le Résultat sera retourné à la composante d'application.

1.6 Utilisation de Terrier : « TREC Terrier »

Dans cette section nous décrivant l'utilisation de TREC Terrier pour l'indexation, la recherche et l'évaluation sur le système d'exploitation Windows XP.

Indexation :

Pour indexer des documents avec Terrier on peut utiliser deux méthodes :

Ø Avec l'invite de commande :
- Spécifier le chemin vers Terrier : cd <le path vers Terrier>\bin
- Spécifier la collection de documents à indexer : trec_setup <le path de la collection à indexer>. Si on ouvre le fichier collection.spec on s'aperçoit que Terrier a mis dans ce dernier tous les pathes vers les fichiers contenus dans le répertoire donné automatiquement.
- L'indexation est lancée par la commande :
  - Ø « trec_terrier –i » Pour une indexation classique two_pass ;
  - Ø « trec_terrier -i –j » Pour une indexation classique Single_pass (Direct Index non construit : Gain d'espaces mémoire). Utilisé pour les collections complexes et volumineuses.

Remarque : pour indexer une collection autre qu'une Collection de TREC, il est nécessaire d'apporter quelques modifications au fichier terrier.properties.

S'il n'y a pas eu d'erreurs, on trouvera dans var\index les fichiers résultants de l'indexation.

Ø Avec desktop_terrier (qui se trouve dans le répertoire bin). L'interface associée permet de sélectionner les documents à indexer puis à lancer l'indexation effective.

Recherche :

Ø Pour faire une recherche sur une collection de documents déjà indexés on utilise la commande « trec_terrier –r ». mais avons, on doit :

- Donner à Terrier les requêtes et ce modifiant etc\terrier.properties comme suit : trec.topics=<le path vers les requêtes>.
- Spécifier le modèle de pondération (ex : TF_IDF) à utiliser dans le fichier etc\trec.models.

Ø Les résultats de la recherche sont stockés dans :
« var\result\nom_fonction_pondération.res »

Ø Notons que l'on peut aussi faire une recherche avec l'interface de Terrier (desktop_terrier).

Evaluation :

Ø Pour effectuer une évaluation, il faut spécifier à Terrier le chemin vers les jugements de pertinence (ou Qrels) de la collection, en spécifiant trec.qrels=<le path vers le fichier qrels de la collection> dans le fichier terrier.propreties.

Ø La commande d'évaluation effective est : trec_terrier –e.

Ø Les résultats de l'évaluation se trouvent dans :
« var\result\nom_fonction_pondération.eval ».

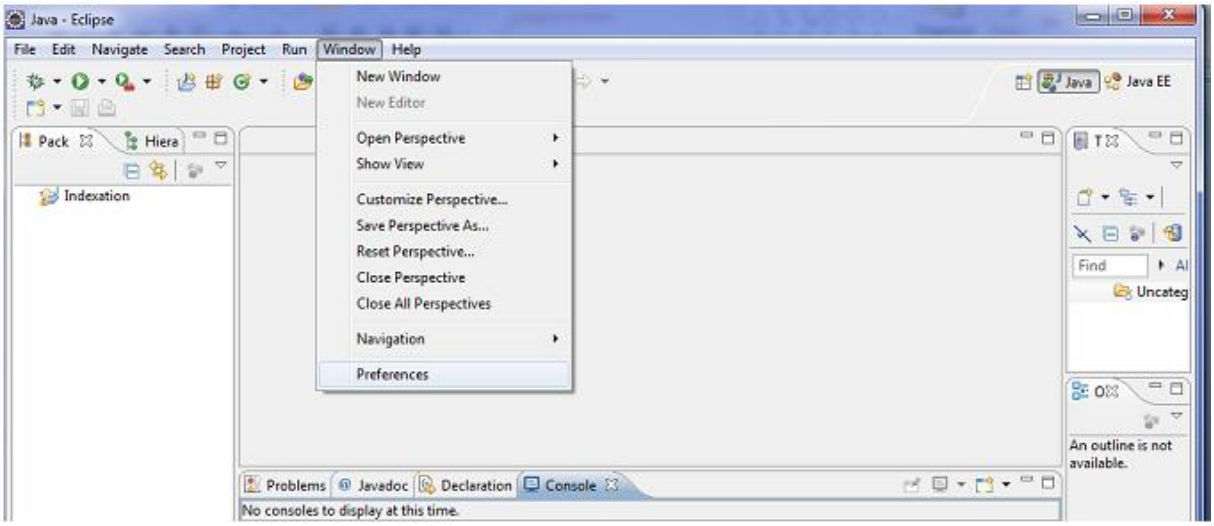
1.7 Extension de terrier :

L'une des principales caractéristiques de Terrier est son extensibilité. Il permet la modification du code sources pour intégrer tout changement nécessaire. La modification peut être soit dans le processus d'indexation ou dans celui de la recherche. Pour que toutes modification soit prise en compte il est nécessaire de recompiler tout le code source de Terrier.

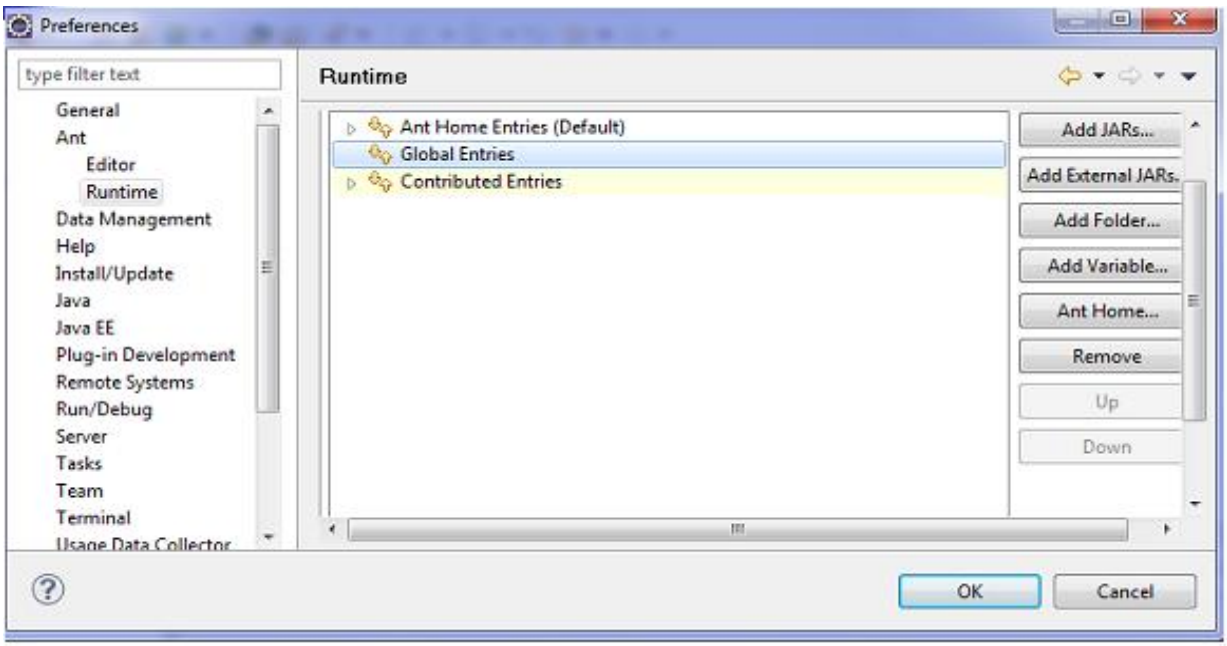
1.8 Compilation de Terrier :

Pour compiler Terrier, utiliser l'Ant de l'environnement IDE Eclipse et le fichier build.xml de terrier 3.5.

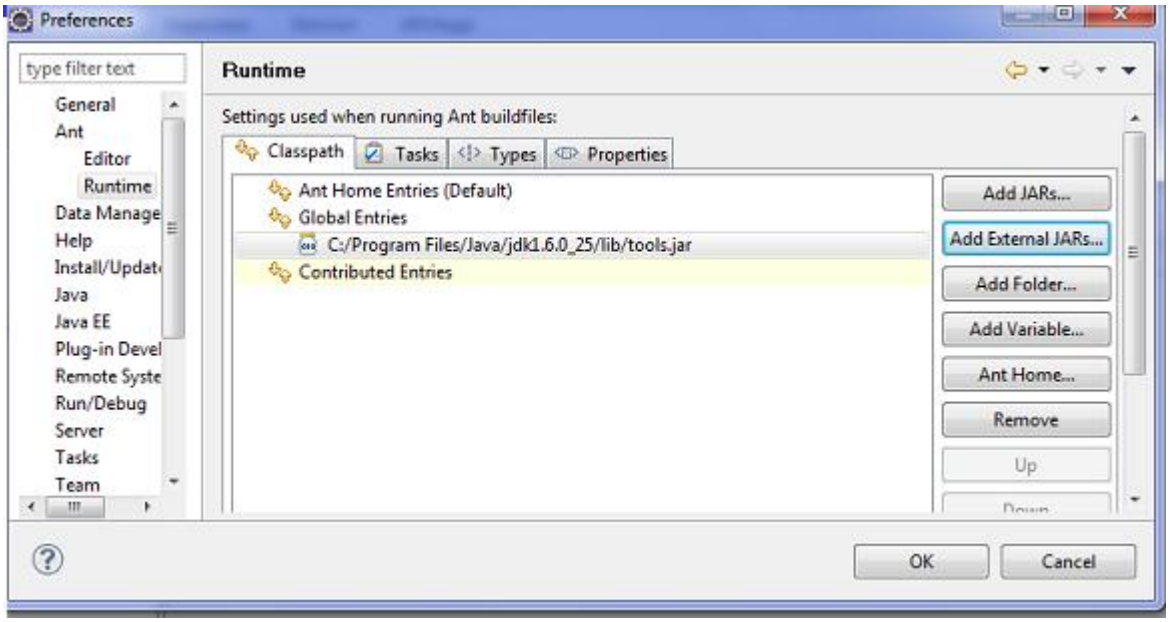
Configuration Ant de l'IDE Eclipse :



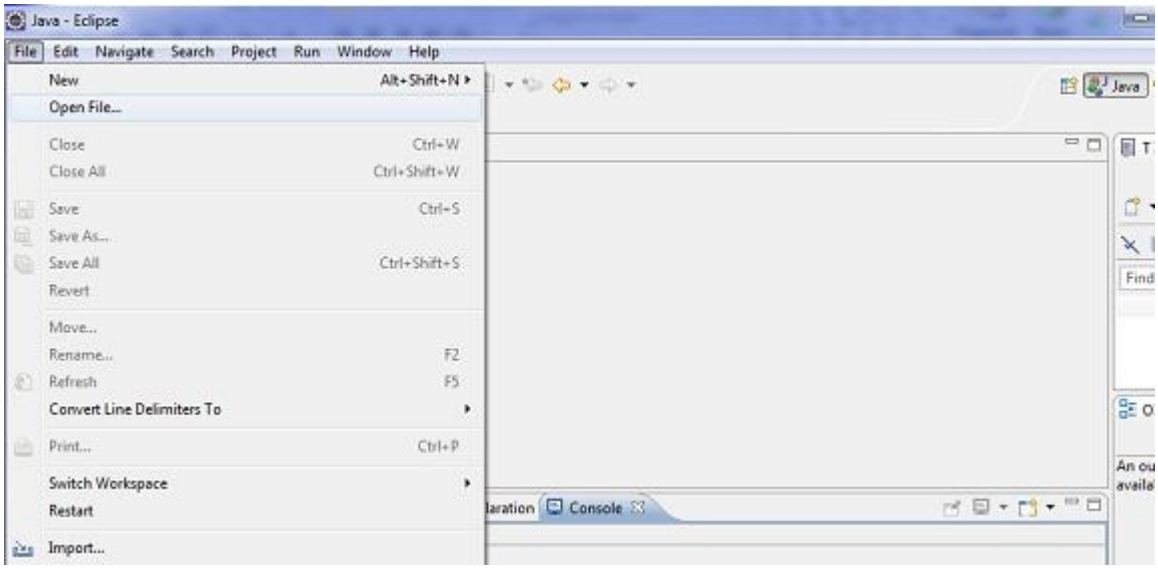
Configuration Ant : 1- Window->Preferences



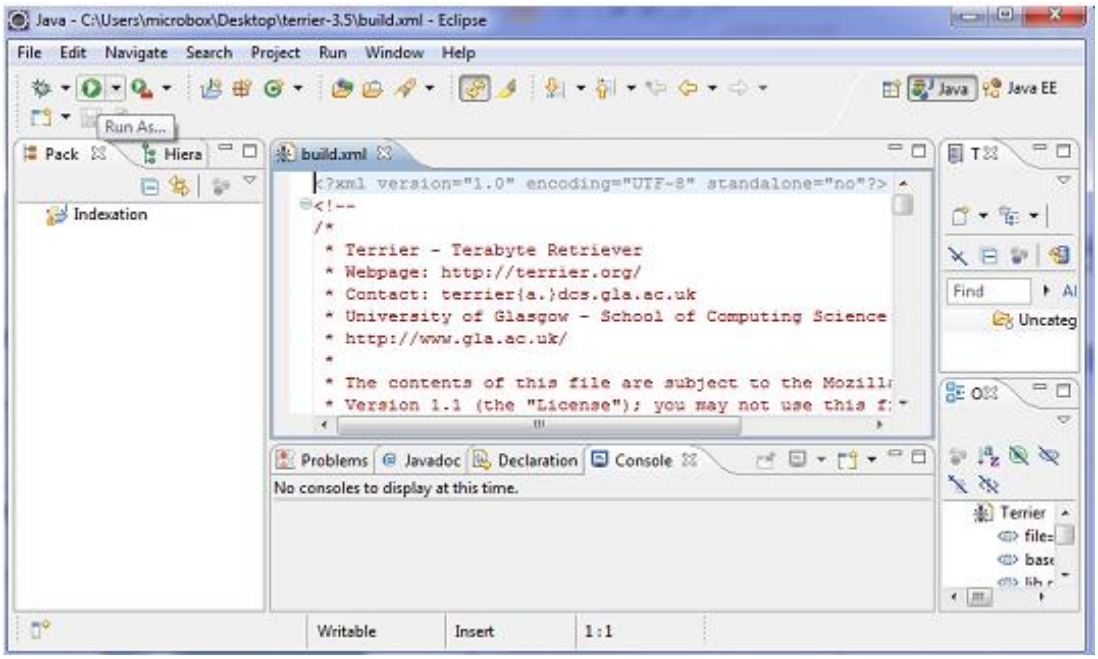
Configuration Ant : 2- Ant ->Runtime->Global Entries



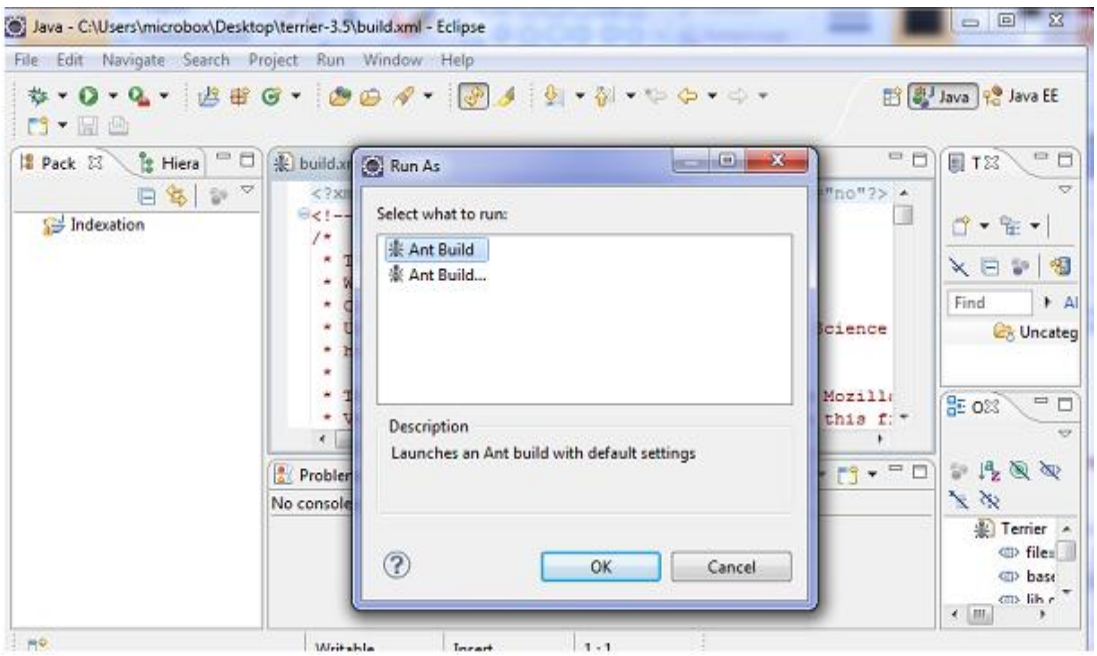
Configuration Ant : 3- Ajouter avec add Externals JARS le fichier ../Java/ jdk1.6.0_25/ lib/tools.jar



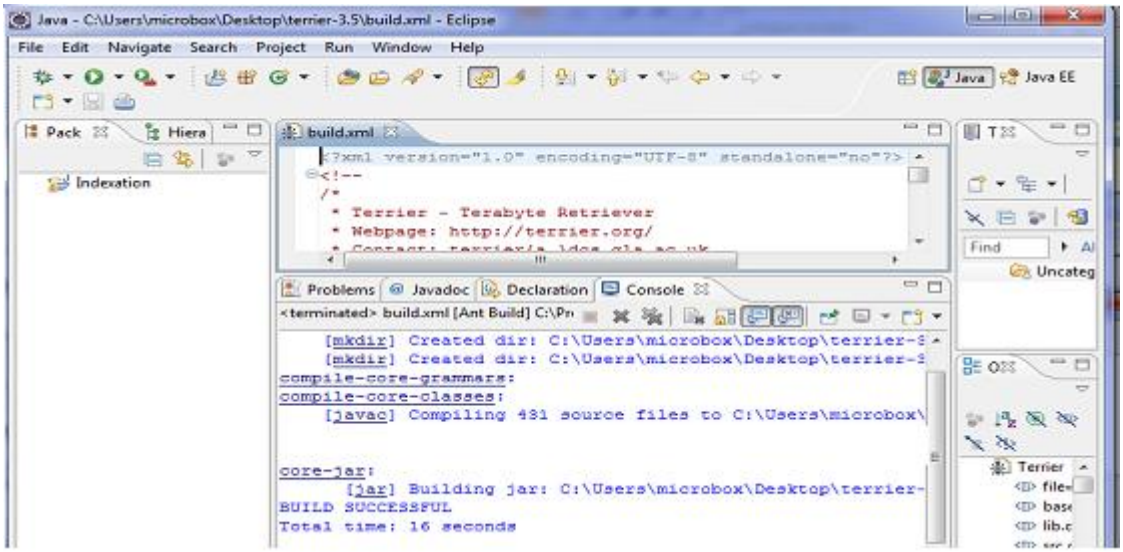
Ouvrir le fichier build.xml de terrier



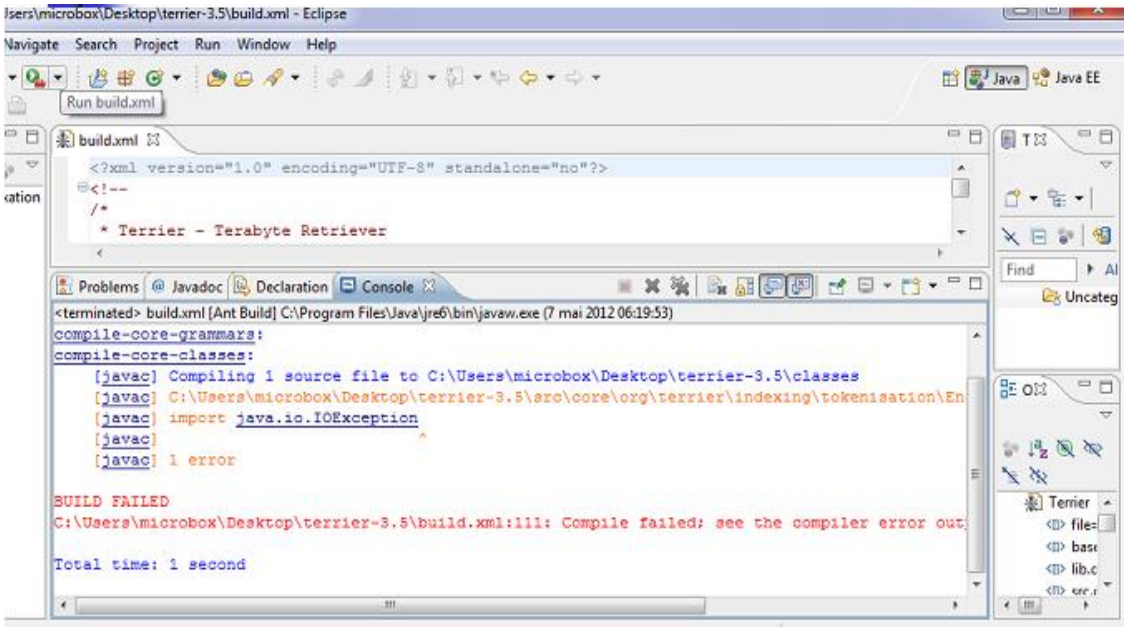
Exécuter le script: Bouton Run



Exécuter Ant Build



BUILD SUCCESSFUL: Compilation réussie



BUILD FAILED: Compilation échouée