Recherche d'information dynamique pour domaines complexes

Mémoire

Robin Joganah

Maîtrise en Informatique

Maître ès sciences (M.sc)

Québec, Canada

Recherche d'information dynamique pour domaines complexes

Mémoire

Robin Joganah

Sous la direction de :

Luc Lamontagne, directeur de recherche

Richard Khoury, codirecteur de recherche

Résumé

Dans ce mémoire, nous traitons du sujet de la recherche d'information dynamique en milieu complexe. Celle-ci a pour but d'inclure l'utilisateur dans la boucle. Ainsi, l'utilisateur a la possibilité d'interagir avec le système en surlignant les passages pertinents et en indiquant le degré d'importance selon ses intérêts.

Dans le domaine de la recherche d'information, les milieux complexes peuvent être définis comme des corpus de textes au sein desquels il est difficile de trouver une information à partir d'une requête générale. Par exemple, si l'utilisateur effectuait une recherche sur les impacts du virus Ebola durant la crise en Afrique en 2014-2015, il pourrait être intéressé par différents aspects liés à ce virus (économiques, de santé publique, etc.). Notre objectif est de modéliser ces différents aspects et de diversifier les documents présentés, afin de couvrir le maximum de ses intérêts. Dans ce mémoire, nous explorons différentes méthodes de diversification des résultats.

Nous réalisons une étude de l'impact des entités nommées et des mots-clés contenus dans les passages issus du retour de l'utilisateur afin de créer une nouvelle requête qui affine la recherche initiale de l'utilisateur en trouvant les mots les plus pertinents par rapport à ce qu'il aura surligné. Comme l'interaction se base uniquement sur la connaissance acquise durant la recherche et celle-ci étant courte, puisque l'utilisateur ne souhaite pas une longue phase d'annotation, nous avons choisi de modéliser le corpus en amont, via les « *word embeddings* » ou plongements lexicaux, ce qui permet de contextualiser les mots et d'étendre les recherches à des mots similaires à notre requête initiale.

Une approche de recherche dynamique doit, en outre, être capable de trouver un point d'arrêt. Ce point d'arrêt doit amener un équilibre entre trop peu et trop plein d'information, afin de trouver un bon compromis entre pertinence et couverture des intérêts.

Table des matières

Liste des figures

Liste des tables

Remerciements

Mes premiers remerciements vont à mon directeur de recherche Luc Lamontagne pour son aide précieuse tout au long de ma maîtrise ainsi que dans la rédaction de ce mémoire.

Je souhaite également remercier mon codirecteur Richard Khoury pour ses précieux conseils tout au long de mon projet de recherche ainsi que ses nombreuses suggestions et corrections durant l'élaboration de ce mémoire.

Je souhaite également remercier les autres élèves du laboratoire du GRAAL, tout particulièrement Jian Mo, Nicolas Garneau et Jonathan Bergeron, pour les échanges qu'ils ont pu permettre tout au long de ma maîtrise et pour l'enrichissement qui a découlé de les avoir côtoyés.

En outre, je souhaite remercier toute ma famille, en particulier mes parents, Nad et Élisabeth Joganah, pour leur soutient tout au long de mes études et pour leur soutient dans ce projet de double-diplôme au Québec. Leur exemple a toujours été un élément moteur dans mon travail.

Je remercie mes frères et sœurs David, Elsa et Nelly, qui m'ont toujours épaulé durant mon parcours scolaire et montré l'exemple de la réussite.

Enfin, je réserve mes derniers remerciements à mon épouse Julie, sans qui ce projet et ce mémoire n'auraient pas pu aboutir, son soutien inconditionnel et ses relectures ont été précieux tout au long de mes études.

Je dédie ce mémoire à ma fille Laure.

Chapitre 1 - Introduction

1.1 Motivations

La recherche d'information dynamique a pour objectif d'inclure l'utilisateur dans le processus de recherche d'information, qui se rapporte aux domaines dits « complexes ». Cette notion se distingue des domaines traditionnels, qui sont déjà traités par les moteurs de recherche du grand public tels que Google, Yahoo ou Bing. Les domaines complexes se réfèrent aux activités professionnelles, qui ne sont généralement pas indexées par ces moteurs de recherche ou mal indexés de sorte que les évènements récents rendent difficile l'exploration de sujets complexes. Nous pouvons établir certains sujets qui illustrent des domaines complexes : comme l'internet profond ou « *Deep Web* », les évènements récents (par ex. la crise africaine du virus Ebola) ou encore des données professionnelles pointues (par ex. les données scientifiques polaires).

Le premier intérêt de ce type de système est de pouvoir trouver des informations sur les zones non indexées d'internet. Celles-ci peuvent contenir des trafics illégaux (drogues, trafic d'êtres humains) ou encore des comportements à risque. L'objectif est de guider un professionnel dans sa recherche en lui permettant de naviguer dans le flot des documents non pertinents, qui constituent l'immense majorité de l'internet non indexé.

Pour matérialiser ce système automatique, il faut inclure l'utilisateur dans la phase de traitement de l'information en lui permettant de surligner les passages qui l'intéressent dans les documents retournés par le système et de leur attribuer à chacun une note sur une échelle numérique, par ex. de 1 à 4.

1.2 Objectifs

Notre premier objectif est de réaliser un système capable de travailler dans un environnement avec une quantité importante de documents non pertinents et non jugés au sein duquel on doit retrouver les documents jugés pertinents par un utilisateur, ainsi que d'être capable de trouver des documents suffisamment diversifiés pour combler les différents intérêts de l'utilisateur par rapport au sujet initial. Contrairement à un système de recherche d'information classique, la seule recherche par mots-clés ne suffit pas à afficher suffisamment de diversité pour atteindre ce type d'objectif.

Notre deuxième objectif est d'utiliser le retour de l'utilisateur de façon optimale pour améliorer les performances de notre système de recherche d'information.

Enfin, notre dernier objectif est de déterminer le moment optimal pour stopper la recherche de documents, dès que l'on juge que l'utilisateur a couvert suffisamment d'information pertinente vis-à-vis de ses intérêts.

1.3 Problématique

Notre problématique est principalement centrée sur l'utilisateur et la capacité du système de recherche d'information à couvrir l'ensemble de ses intérêts.

Considérons par exemple qu'un utilisateur est intéressé par une recherche sur des forums de l'internet profond et désire obtenir un compte Facebook illicite qui a déjà un certain nombre de milliers de contacts, mais ne sait pas comment s'y prendre (exemple tiré du jeu de données « *Illicit goods* » de la compétition TREC 15, qui sera présentée formellement à la section 3.1). Sa recherche sera la suivante : « *Facebook accounts* ». En effectuant cette recherche, le système soumet des documents

divers à l'utilisateur : qui sont les vendeurs de tels comptes, quelles sont les méthodes de paiement disponibles ou encore quels sont les prix fixés pour acquérir de tels comptes. Parmi ces sujets découverts par le système, l'utilisateur pourra annoter ce qui l'intéresse et pourra ainsi approfondir ses connaissances sur les sujets qui lui semblent les plus intéressants. Notre système n'a pas connaissance à l'origine des sujets du domaine de recherche, ni desquels d'entre eux seront intéressants pour notre utilisateur.

Il faut donc réaliser d'une part, une phase d'exploration pour déterminer les intérêts de ce dernier et d'autre part, une phase d'exploitation pour trouver les meilleurs documents qui y correspondent.

1.4 Méthodologie

La méthodologie employée suit les standards établis par la compétition et conférence TREC (« *Text Retrieval Conference* »)[1] pour la catégorie « *Dynamic Domain* »[2] ou « Domaine dynamique ». Nous avons participé à deux éditions de cette compétition (TREC 15 et TREC 16). Nos systèmes ont donc été évalués par l'organisation de la conférence et nos expérimentations ont utilisé les mêmes standards afin de valider nos travaux. Pour ce faire, il a donc été nécessaire de développer un système complet de recherche d'information capable d'indexer les fichiers, de traiter le retour de l'utilisateur ainsi que de lui proposer des nouveaux documents à chaque étape.

Ces évaluations sont basées notamment sur une nouvelle mesure en recherche d'information appelée « *CubeTest* ». Il s'agit d'une mesure qui permet de différencier l'information apportée selon

[1] http://trec.nist.gov/

[2] http://trec-dd.org/

différents sous-intérêts par rapport à une requête initiale. Cette mesure diffère donc des traditionnelles mesures de recherche d'information, que sont le rappel et la précision. Ces deux mesures jugent de façon binaire les documents (pertinents ou non pertinents). Cependant l'objectif de ce mémoire est également d'explorer de nouvelles méthodes d'évaluation. Ainsi nous proposons de modifier cette métrique afin qu'elle prenne en compte d'autres aspects propre à la recherche d'information dynamique.

En résumé, la méthodologie comporte les étapes suivantes :

1. Indexation du corpus d'expérimentation
2. Recherche à l'aide d'une requête initiale
3. Présentation des résultats à l'usager (N meilleurs documents correspondant à la requête)
4. Prise en compte du retour de l'utilisateur (jugement de pertinence des documents présentés)
5. Mise à jour de la requête et nouvelle recherche de documents
6. Itération sur les étapes 3 à 5 jusqu'à qu'un critère d'arrêt soit satisfait
7. Évaluation des résultats sur plusieurs requêtes à l'aide de diverses métriques.

Chacune de ces étapes de la méthodologie seront présentées dans les chapitres subséquents de ce mémoire.

1.5 Plan du mémoire

Ce mémoire est organisé de la manière suivante. Dans le Chapitre 2, nous étudions les notions de base du traitement automatique du langage naturel (TALN) et de l'apprentissage automatique qui seront utilisées tout au long de ce mémoire. Au Chapitre 3, nous étudions en profondeur le thème principal de ce mémoire, à savoir la recherche d'information. Nous faisons ainsi la revue de littérature de la recherche d'information dynamique et mettons particulièrement l'accent sur le format proposé par la compétition TREC, pierre angulaire de ce travail et référence dans le domaine de la recherche d'information pour son format d'évaluation et les jeux de données créés à partir du travail d'assesseurs reconnus. Le Chapitre 4 est consacré à la méthode d'indexation et aux prétraitements effectués sur le

corpus. Le Chapitre 5 traite du problème de la recherche d'information statique et dynamique, il s'agit d'identifier les méthodes permettant de réaliser notamment de la diversification de résultats. Le Chapitre 6 est consacré au retour de l'utilisateur et à la manière dont on peut l'utiliser pour trouver des documents similaires. Le Chapitre 7 est à la fois le dernier chapitre de présentation des méthodes élaborées dans ce mémoire et s'attarde sur le dernier aspect du système dynamique : trouver un critère pour que le système sache s'arrêter automatiquement dès qu'il estime avoir fourni suffisamment d'information à l'utilisateur. Dans le Chapitre 8 nous étudierons les expérimentations réalisées dans le cadre des compétitions TREC 15 et 16, liées aux méthodes développées dans les chapitres précédents et nous présenterons enfin les pipelines qui ont été construits sur les expérimentations passées afin de se comparer à l'état de l'art.

Chapitre 2 - Définitions fondamentales

La recherche d'information est un sujet qui aujourd'hui s'est généralisé par l'accès à la technologie et l'évolution au premier plan d'entreprises dont c'est le cœur de métier, telles que Google et Yahoo. Manning définit ce champ de recherche comme le fait de trouver des documents de nature non structurée pour satisfaire le besoin d'information de l'utilisateur parmi une large collection de documents [13]. Cette définition est plus que jamais d'actualité avec l'émergence des données massives et le besoin de traiter des collections de plus en plus larges et des données non structurées, telles que les systèmes de grandes entreprises qui cherchent à indexer et rendre possible la recherche parmi tous les document stockés [4]. Ce constat est notamment partagé par [21] qui voient en la recherche d'information de nombreux défis moderne avec les systèmes de recommandations de produits divers (musiques, vidéos, livres) qui constituent une application de la recherche d'information dynamique.

La recherche d'information est également étroitement liée au traitement automatique du langage naturel (TALN) par sa problématique intrinsèque. L'information est majoritairement composée de textes écrits par l'être humain, en opposition aux données binaires / numériques fournies par l'intelligence artificielle. Le TALN a donc pour but de permettre de transformer un texte brut en une donnée exploitable par un algorithme. Cela nécessite de passer par un certain nombre d'étapes dans le prétraitement des données, telles que : la transformation en unités lexicales élémentaires ou tokénization, la racinisation et la lemmatisation.

2.1 Tokénization

La transformation en unités lexicales élémentaires a pour objectif de découper un groupe de mots/ponctuations en une liste d'unités lexicales (des jetons ou « *token »* en anglais).

Si l'on prend par exemple une requête initiale telle que :

« Victoria Stafford - murder trial of Michael Thomas Rafferty »

On obtient la liste de jetons suivante :

['Victoria', 'Stafford', '-', 'murder', 'trial', 'of', 'Michael', 'Thomas', 'Rafferty']

2.2 Racinisation et lemmatisation

La racinisation permet de garder uniquement le radical d'un mot [8]. Les terminaisons de mots sont tronquées à l'aide de règles. Ainsi, deux mots ayant le même radical seront comptabilisés comme similaires, ce qui permet de réduire la taille de notre index de mots, qui peut très vite atteindre une taille trop grande pour être utilisable de façon intelligente. Par exemple, si les mots « *tell* » et « *telling* » sont considérés comme différents alors qu'il s'agit du même verbe, cela peut complexifier notre index de mots inutilement, les deux mots ayant sensiblement le même sens.

La lemmatisation permet de ramener les mots à une forme normalisée. La lemmatisation de « *had* » nous donnera donc « *have* », ce qui permettra d'identifier l'usage du mot « *have »* sous ses 2 formes au lieu d'avoir à analyser séparément deux mots identiques (dans leur sens).

La racinisation et la lemmatisation se différencient ainsi : dans le premier cas, on cherchera la racine à partir d'une règle (par ex. « s » à la fin d'un mot) alors que dans le second cas on normalisera par la forme la plus basique présente dans un dictionnaire (infinitif, singulier). Ainsi pour le mot *« has »* la racinisation donnera « ha » alors que la lemmatisation donnera *« have »*.

En recherche d'information, on utilise plus souvent la racinisation.

2.3 Reconnaissance d'entités nommées

Les entités nommées peuvent être définies comme des informations particulières fréquemment désignées à l'aide de noms propres, ce qui inclut toute entité que l'on nomme pour la définir. Dans le cas de l'algorithme issu de la librairie NLTK[3] que nous utilisons, on peut identifier trois grandes familles d'entités :

- Les entités géopolitiques ;
- Les organisations ;
- Les personnes.

Ces entités serviront à identifier des éléments clés d'une requête ou d'un document pertinent. Par exemple, si l'on considère la requête suivante : un homme politique, dont on veut savoir quelles sont les personnes qu'il a rencontrées durant sa campagne aux élections locales. L'utilisation de l'algorithme de reconnaissance d'entités nommées nous permettra d'extraire les noms de ces personnes pour effectuer ensuite d'autres recherches les incluant et trouver des documents satisfaisants pour l'utilisateur.

[3] http://www.nltk.org/

2.4 Expansion de requête

L'expansion de requête est souvent utilisée en recherche d'information pour « augmenter » la qualité de la requête [21] en ajoutant des mots qui pourraient détailler celle-ci. Celle-ci peut prendre la forme de « pseudo-retours » ou bien d'un retour réel de l'utilisateur. Dans le premier cas, on considère les premiers résultats retournés par le moteur de recherche comme pertinents et on extrait les mots communs pour les ajouter à la requête initiale. Dans le second, les résultats sont présentés à l'utilisateur et celui-ci nous indique s'ils sont pertinents. Les mots communs des documents jugés pertinents par l'utilisateur sont alors ajoutés à notre requête. Cette méthode est plus particulièrement détaillée au Chapitre 6 de ce mémoire.

2.5 Apprentissage automatique

L'apprentissage automatique peut être défini comme un algorithme permettant à la machine d'apprendre des mécanismes de façon autonome grâce aux éléments statistiques qui sous-tendent le phénomène.

Il existe différents types d'apprentissage automatique, divisés en deux grandes catégories : « supervisé » et « non supervisé ». La première catégorie concerne les algorithmes qui apprennent sur des données étiquetées, la seconde concerne ceux capables de trouver les patrons sur des données non étiquetées. Durant l'élaboration de ce mémoire, nous avons notamment utilisé deux grandes techniques d'apprentissage non supervisé : la modélisation de sujets ou « *topic modeling* » d'une part – cette technique dite générative est utilisée pour faire émerger différents sujets grâce à la distribution des mots – et d'autre part, le partitionnement (« *clustering* » ou regroupement), qui permet de regrouper les documents les plus semblables au sein d'un même ensemble.

2.5.1 Modélisation de sujets à l'aide de l'allocation de Dirichlet latente (LDA)

L'allocation de Dirichlet latente [6] est une technique de modélisation de sujet qui peut être utilisée en recherche d'information dans un but de diversification. Cette technique permet d'utiliser un corpus existant et d'en faire émerger un nombre de sujets préalablement déterminé. Ceci grâce à un système génératif faisant appel aux variables latentes [3]. Chaque sujet est caractérisé par une distribution de probabilités pour les mots du corpus d'appartenir à ce sujet.

Voici les grandes étapes de l'algorithme LDA :

1. Choisir une distribution $\theta_i \sim Dir(\alpha)$, où $i \in \{1, \dots, M\}$. $Dir(\alpha)$ est la distribution de Dirichlet qui indique la probabilité qu'un document corresponde à un thème. Le paramètre α $(\alpha < 1)$ représente l'apriori de Dirichlet sur la distribution des documents. M est le nombre de documents.
2. Choisir $\varphi_k \sim Dir(\beta)$, où $k \in \{1, \dots K\}$ représente le sujet et β représente le prior de Dirichlet sur la distribution des mots par sujet.
3. Pour chacune des positions du mot i, j , où $i \in \{1, \dots, M\}$ correspond au document et $j \in \{1, \dots, N\}$ correspond à la position du mot dans ce document :
   (a) Choisir un sujet $z_{i,j} \sim Multinomial(\theta_i)$.
   (b) Choisir un mot $w_{i,j} \sim Multinomial(\varphi_{z_{i,j}})$.

Notre jeu de données, que l'on présentera plus en détail dans la partie « Expérimentations et résultats » de ce mémoire est notamment composé de nombreux articles de presse relatifs aux évènements de la crise du virus Ebola en Afrique entre 2014 et 2015. Il est également composé de requêtes auxquelles sont associés les documents pertinents pour différents sous-sujets.

Ainsi, voyons ci-dessous un exemple d'une de ces requêtes et des résultats que l'algorithme LDA peut nous donner en appliquant cette technique sur les 200 premiers résultats (les 200 premiers

articles de presse) associés aux mots-clés recherchés. Pour la recherche issue de la requête « *US Military Crisis response* » :

- Sujet 1: *obama ebola africa outbreak health west liberia military disease*
- Sujet 2: *ebola sierra outbreak leone west liberia world health virus people*
- Sujet 3: *ebola africa military liberia west response army article troops*
- Sujet 4: *ebola response military workers million medical said international support*
- Sujet 5: *military health ebola liberia public crisis human government american years*

Dans ce premier exemple, on voit différents sujets qui se recoupent plus ou moins autour du mot « *Liberia »* et autour du concept des « États-Unis » (« *Obama », « American »*). On a donc des sujets plus centrés sur des questions militaires (« *military », « army », « troops »* dans le même sujet), alors que d'autres semblent plus politiques (« *obama », « africa », « health », « west »*). Bien qu'il y ait une certaine redondance dans les sujets, on peut y voir un intérêt pour spécifier des sous-sujets autour d'un même thème.

Ci-dessous, un autre exemple de modélisation sur des sujets plus controversés, tels que les hypothèses de conspiration autour de la crise d'Ebola :

- Sujet 1: *ebola evidence false flag conspiracy exposed youtube*
- Sujet 2: *ebola conspiracy government health outbreak control people cdc africa theories*
- Sujet 3: *ebola documentary outbreak deadly inside 2014 news dr richard hd*
- Sujet 4: *ebola people virus conspiracy disease theories africa spread world theory*
- Sujet 5: *obama ebola conspiracy theory just caused people stupid 2014 ingraham*

Dans ce deuxième exemple, on voit clairement des regroupements, notamment entre des documents qui vont critiquer les conspirations et d'autres qui vont les considérer comme sérieuses et donc employer des termes proches (*« false flag », « exposed »,* etc.).

Pour générer ces résultats, nous utilisons un nombre d'itération fixé à 100 avec un nombre de sujets égal à 5. Ce paramètre est important car un nombre trop court d'itération ne permet pas de converger vers une distribution probante, tandis qu'un nombre d'itération trop élevé demande trop de temps de calcul pour une amélioration des résultats qui peut être variable. Un bon compromis est donc en général de situer le nombre d'itérations entre 100 et 1000.

2.5.2 Regroupement de documents avec l'algorithme des K-Moyennes (« K-Means »)

L'algorithme des *K*-Moyennes ou *« K-Means »* est un algorithme dit de regroupement ou *« clustering »*. Il s'agit d'un algorithme non supervisé, capable de séparer des données non étiquetées et de les regrouper par distance avec des centroïdes (un centroïde est le poids moyen de chacun des mots d'un groupe de documents). Le nombre de centroïdes *K* est déterminé en amont. Le regroupement peut être utilisé en recherche d'information pour considérer différents aspects d'un sujet [13].

L'algorithme des *K*-Moyennes a pour objectif de minimiser la moyenne des distances euclidiennes entre les documents – représentés par des vecteurs qui contiennent le poids de chacun des mots du vocabulaire – et leurs centroïdes, où chaque centroïde est défini comme la moyenne $\vec{\mu}$ des documents d'un *« cluster »* ω [13] :

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x} \qquad (1.)$$

Pour minimiser cette moyenne, l'algorithme fonctionne de manière itérative en utilisant une initialisation aléatoire. L'algorithme des *K*-Moyennes (Manning et al., 2008) se décrit comme suit :

$K - MEANS(\{\vec{x}_1, \dots, \vec{x}_N\}, K)$

1. $(\vec{s}_1, \vec{s}_2, \dots, \vec{s}_K) \leftarrow SelectionAleatoire(\{\vec{x}_1, \dots, \vec{x}_N\}, K)$
2. Pour $k \leftarrow 1$ à K
3. $\quad \vec{\mu}_k \leftarrow \vec{s}_k$
4. Tant que le critère d'arrêt n'a pas été rencontré.
5. Pour $k \leftarrow 1$ à K
6. $\quad$ Faire $\omega_k \leftarrow \{\}$
7. $\quad$ Pour $n \leftarrow 1$ à N
8. $\quad\quad$ Faire $j \leftarrow \arg min_{j'} \left|\vec{\mu}_{j'} - \vec{x}_n\right|$
9. $\quad\quad\quad \omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$ $(Réassigner\ les\ vecteurs)$
10. $\quad$ Pour $k \leftarrow 1$ à K
11. $\quad$ Faire $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$ $(Recalculer\ les\ centroïdes)$
12. Retourner $\{\vec{\mu}_1, \dots, \vec{\mu}_K\}$

L'intuition derrière son utilisation en recherche d'information est que, s'il existe différents documents parlant d'un même sujet, les documents parlant d'un même sous-sujet devraient avoir des mots en commun et ainsi se regrouper au sein d'un même *« cluster »*.

Chapitre 3 - État de l'art de la recherche d'information dynamique

La recherche d'information dynamique est un sujet récent et complexe. La compétition TREC « *Dynamic Domain* » est supportée par le programme MEMEX de la DARPA, qui a pour objectif d'indexer l'internet profond afin d'être capable de naviguer au travers de jeux de données ayant une grande quantité de bruit. Il s'agit donc d'un problème présent et du futur. C'est un sujet complexe, étant donné le nombre d'éléments qu'il faut prendre en compte tout au long du processus. Ainsi, la recherche dynamique est à la croisée de multiples sous-domaines de l'intelligence artificielle comprenant notamment : la recherche d'information, l'extraction d'information, la reformulation de requête et l'apprentissage automatique.

3.1 Compétition TREC « Domaine dynamique »

La recherche dynamique telle que nous la présentons dans ce mémoire est le reflet de la compétition internationale TREC organisée par le *National Institute of Standards and Technology* (NIST). La conférence TREC est spécialisée dans la recherche d'information depuis 25 ans ; dans ce domaine, elle fait office de référence en termes d'évaluation et de méthodologie pour la création de jeux de données permettant d'attribuer des degrés de pertinence aux documents d'une collection vis-à-vis de certaines requêtes préétablies, via des évaluations réalisées par des assesseurs volontaires.

C'est ainsi que, dans le cadre de la catégorie « *Dynamic Domain* » (« Domaine dynamique »), la compétition TREC a permis, par le biais des assesseurs, de fournir une collection unique de sujets

relatifs à différents domaines permettant ainsi d'avoir un jugement sur les intérêts d'un potentiel utilisateur.

La recherche interactive est le fruit d'interactions entre un utilisateur simulé et un système qui réagit à ses retours. Cet utilisateur est supposé expert dans un domaine et a de multiples intérêts autour d'un sujet complexe. Les domaines choisis pour notre étude correspondent aux jeux de données fournies lors des compétitions TREC 15 et TREC 16 avec des sujets divers, tels que :

- Des bulletins de nouvelles sur la politique locale aux États-Unis ;
- Des articles de journaux en ligne concernant la crise sanitaire du virus Ebola entre janvier et mars 2014 ;
- Des discussions sur des forums parlant de produits illicites ;
- Des données scientifiques polaires.

Chaque domaine contient plusieurs sujets, qui constituent les requêtes initiales. En outre, chaque sujet contient les intérêts de l'utilisateur – entre 2 et 12 sous-sujets : l'objectif est de couvrir ces sous-sujets pour le besoin d'information de l'utilisateur. Par exemple, on imagine que notre utilisateur est à la recherche d'information sur le maire de New York (le nom du maire est ici la requête initiale). Notre système ignore par contre ce qui intéresse notre utilisateur à propos de ce sujet. En outre, notre système ne sait pas à combien s'élève le nombre des intérêts de notre utilisateur. C'est donc là l'une des difficultés posées au système, qui doit à la fois trouver un maximum de documents pertinents, mais aussi couvrir une variété de sujets.

L'utilisateur interagit avec le système en annotant des passages dans les documents retournés par le système dynamique. L'interaction se présente sous la forme d'une liste contenant les paramètres suivants :

- *Topic_id : n*uméro d'identification (ID) du sujet principal de la requête (ex : *« US Military Crisis Response » a* pour ID 1)
- *On_topic : e*st-ce que le document contient un passage pertinent ?

- *Doc_id :* numéro d'identification du document
- *Subtopics : l*iste des sous-sujets couverts lors de la précédente recherche.
- *Passage_text : p*assage pertinent
- *Rating : d*egré de pertinence du passage
- *Subtopic_id : n*uméro du sous-sujet relatif au passage

Avec ces différents paramètres, il est possible de regrouper l'information pour chaque sous-sujet découvert et ainsi tenter de découvrir les intérêts de notre utilisateur.

Cette annotation contient des informations utiles à notre système, telles que la note attribuée au passage par rapport à sa pertinence sur une échelle de 1 à 4 et le sous-sujet vis-à-vis duquel le document est pertinent. L'échelle de 1 à 4 est documentée par la compétition TREC comme suit :

- < 3 résultat peu pertinent
- = 3 résultat intéressant
- = 4 résultat clé

Ainsi, l'on peut avoir un même document qui contient plusieurs passages correspondant aux intérêts sous-jacents de notre utilisateur. Il est également possible qu'un document soit pertinent par rapport à l'un des centres d'intérêts de l'utilisateur, mais qu'il ne soit pas étiqueté comme étant pertinent. Cette dernière information est importante, car lors de la prise en compte du retour de l'utilisateur nous ne pouvons pas affirmer qu'un retour « *on_topic : 0* » indique que le document est non pertinent, mais uniquement qu'il n'est pas jugé. En effet, les évaluations sont réalisées par des assesseurs : ce sont donc des critères réels et il est possible qu'un certain nombre de documents n'ait pas été évalué.

3.2 Méthodologies et techniques utilisées en recherche d'information dynamique

La recherche d'information dynamique telle que décrite par [2] est très proche de la définition donnée dans le cadre de la compétition TREC [20], puisque les auteurs associent également la recherche dynamique à un besoin de recherche professionnelle sur une base de données dont on a peu d'information (dans leur cas, la recherche gouvernementale). Ils décrivent notamment le besoin de guider l'utilisateur qui n'a pas connaissance des mots-clés qui vont l'aider à retrouver les documents pertinents, le sujet étant inconnu ou nouveau pour lui.

La recherche d'information dynamique est au carrefour de toutes les techniques présentées précédemment.

Nous présentons donc dans cette section les différentes techniques qui ont été utilisées lors de ces deux éditions de la conférence, permettant ainsi de voir l'éventail des propositions de systèmes qui ont émergé et les orientations qui semblent être les plus prometteuses.

Un pipeline classique de recherche d'information dynamique contient 4 phases principales dans la quasi-totalité des systèmes existants : une recherche d'information initiale, une recherche d'information itérative comprenant un traitement du retour de l'utilisateur ainsi qu'une diversification des résultats et enfin un critère d'arrêt. Nous illustrons ce système à la Figure 3-1.

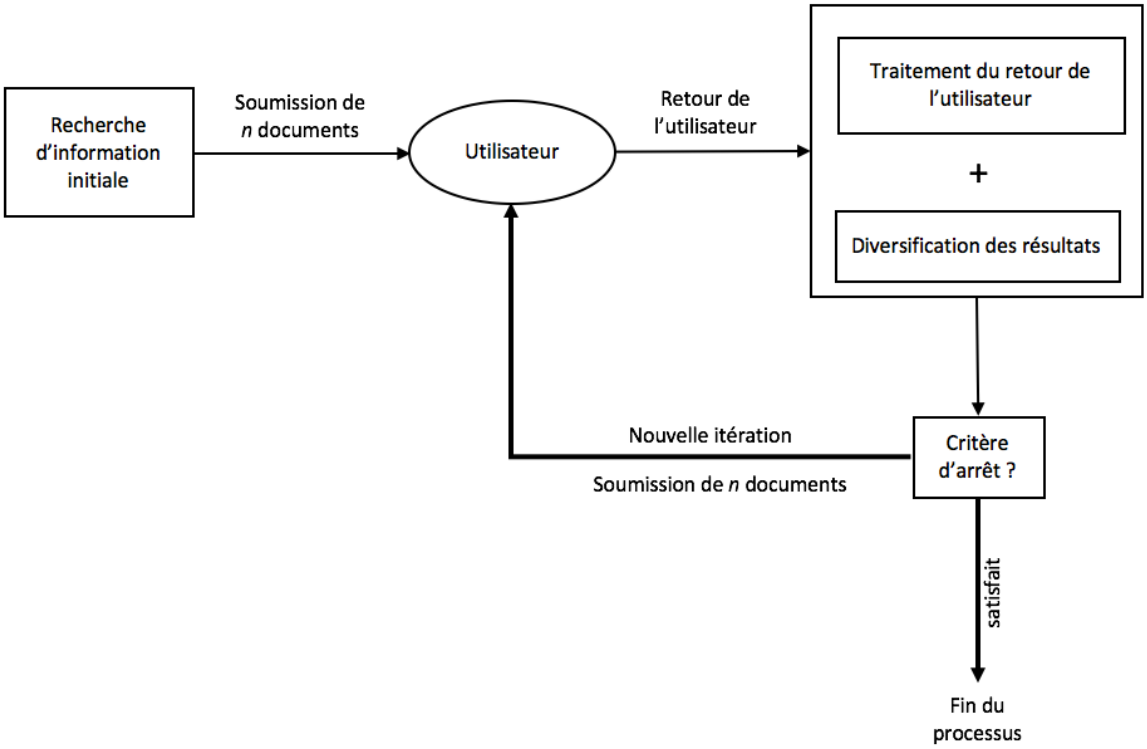

Figure 3-1. Pipeline de recherche d'information dynamique

Pour avoir une vue d'ensemble, prenons un exemple de requête initiale d'un utilisateur et étudions son parcours « idéal » au sein des différentes phases :

- Étape de recherche initiale. - L'utilisateur effectue une recherche initiale : « *US military crisis response ».* Le système retourne les documents en fonction de leur score de similarité avec la requête
- Étape de diversification des résultats. - Le système peut appliquer des techniques de diversification des résultats en cherchant des documents différents parmi les meilleurs documents retournés.
- Étape du traitement du retour de l'utilisateur. - L'utilisateur indique au système quels sont les documents pertinents avec leur degré de pertinence. Imaginons dès lors que deux des

documents parlent d'une intervention militaire des États-Unis à propos d'Ebola, alors que les trois autres parlent d'une intervention militaire d'un autre pays, ou même des États-Unis, mais sur une autre zone d'intervention. En conséquence, le système utilise le passage surligné ou bien l'ensemble des documents positifs pour extraire de nouveaux mots-clés et pénaliser les mots apparaissant dans les articles négatifs. On va ainsi reformuler notre requête « *US military troops, humanitarian crisis, response »*, les termes « *humanitarian »* et « *troops »* se sont ajoutés à la requête par ce processus en fonction de leur proximité avec les termes existants de la requête ou leur présence dans les documents positifs retrouvés durant la recherche.

- Étape du critère d'arrêt. - On peut réitérer les étapes précédentes autant de fois qu'on le souhaite, c'est ici qu'intervient la notion de critère d'arrêt. Dans la recherche d'information dynamique idéale, c'est le système qui décide du moment où s'arrêter et non l'utilisateur.

Les quatre phases ont été étudiées par différentes équipes au cours des compétitions TREC 15 et 16 ce qui a permis de faire émerger un certain nombre de solutions qui sont plus ou moins bonnes pour chaque étape. Nous récapitulons au

Tableau 3-1 les travaux des différentes équipes et les résultats obtenus. Ce tableau n'est pas exhaustif, mais présente uniquement les travaux des équipes que nous étudions plus en détail dans cette revue de littérature. Nous nous comparons à ces différentes équipes dans la partie « Expérimentations et résultats ».

Équipe	TREC	Technique de recherche d'information	Moteur de Recherche	Algorithme de diversification	Retour de l'utilisateur	Indexation de passages	Critère d'arrêt	Manuel ou automatique
Padua	16	BM25	ElasticSearch		Rocchio		Fixe	Automatique
Laval	15 et 16	LM BM25	Solr	LDA - Kmeans			Fixe	Automatique
RMIT	16	LM	Solr		Rocchio	oui	Fixe	Manuel Automatique
Georgetown 2016	15 et 16	LM BM25	Indri	LDA	Relevance feedback		Fixe	Automatique
UFMG	16	LM	ElasticSearch	Hierarchical Flat Multi-dimensional			Fixe Cumul Fenêtre	Automatique

Tableau 3-1. Comparaison des systèmes testés sur le jeu de données TREC 16

3.2.1 Présentation de la recherche d'information

La recherche d'information constitue la base de la recherche dynamique. Les expérimentations pour obtenir les meilleurs résultats ont donc été réalisées avec des travaux portant sur les différentes méthodes de recherche de documents. Il existe en effet plusieurs mesures de similarité permettant de classer les documents en fonction de cette mesure afin de soumettre les documents à l'utilisateur dans l'ordre le plus pertinent pour lui.

Les deux phases de recherche d'information initiale et de recherche d'information itérative peuvent faire appel à des moteurs de recherche. Dans un premier temps, avant de classer les documents, il s'agit de trouver une partie des documents pertinents via une recherche de base. Il est dès lors possible d'utiliser un moteur de recherche uniquement durant la phase initiale pour ensuite chercher à reclasser les documents lors de la phase itérative, ou alors d'effectuer des recherches par le biais du moteur de recherche lors des deux phases avec différents mots-clés afin de retrouver des groupes de documents différents.

Ces deux approches ont différentes vertus. La première permet de fixer un cadre dans lequel les mots-clés de la recherche initiale couvrent une grande majorité des documents pertinents, au risque de passer à côté de documents pertinents ayant une formulation différente du sujet initial. Ainsi, si l'on prend par exemple un sujet de la compétition TREC 16, « *US military crisis response »*, un document peut très bien contenir des mots tels que « *America »*, *« army »*, *« deployment »*, sans pour autant parler de réponse ou de crise, mais d'un synonyme. Ce document serait donc pertinent, mais ne serait pas présent dans l'ensemble de base. Cependant, cette hypothèse reste peu probable, il s'agit donc de mesurer l'impact de tels choix. Nous avons ainsi vu l'un des aspects négatifs du choix d'utiliser le même groupe de documents lors de la recherche initiale et lors des recherches itératives. Mais l'on peut également y voir des éléments positifs, tels que le fait de rester dans le cadre de la recherche initiale : ainsi la présence des mots-clés initiaux est obligatoire, mais l'ajout de mots-clés peut permettre de reclasser ce groupe de documents et faire remonter des documents qui étaient initialement très loin.

Cependant, un certain nombre de systèmes n'utilisent pas l'ensemble des documents pour effectuer une nouvelle recherche, mais ne font que reclasser les n premiers documents (souvent 1000) à partir de la recherche initiale. Dans ce cas si un article parle uniquement de « *America's army deployement to fight ebola* » les mot-clés « *US military crisis response »*, étant tous absents, même en reclassant les 1000 documents initiaux, ne feront pas apparaître un tel article, qui est pourtant pertinent. Mais cette approche a comme avantage de ne pas laisser la recherche diverger trop loin du sujet initial.

3.2.2 Approches de recherche d'information initiale

Lors de la compétition TREC 15, les participants ont utilisé principalement trois moteurs de recherches comme outil de base de recherche d'information : Lemur (Indri), Solr (Lucene) et Terrier. Ces trois moteurs de recherche intègrent les outils de base (indexation, prétraitement, recherche),

cependant ils présentent des différences dans les modèles plus complexes qu'ils intègrent. En effet Lemur et Terrier étant le fruit d'équipes académiques, ils ont tendance à intégrer directement des méthodes développées par leurs laboratoires ou bien les méthodes les plus reconnues par la communauté scientifique. Solr quant à lui a les avantages d'un moteur destiné à un public plus large et permet d'accéder à un outil performant tout en ayant la possibilité d'implémenter les techniques de l'état de l'art.

Plus que le choix du moteur de recherche, c'est surtout le choix de la méthode de recherche utilisée qui apparaît important. Ainsi, l'Université de Konan [20] a comparé la similarité par cosinus de vecteurs TF*IDF avec la similarité probabiliste « *Language Model »* ou modélisation du langage (LM). La seconde méthode a montré des résultats supérieurs sur un même pipeline : comme l'indique [13] la similarité par modélisation du langage a été, dans plusieurs cas, démontrée comme supérieure à des mesures comme BM25 ou par comparaison de vecteur TF*IDF.

Bien qu'il en existe un grand nombre, nous parlerons ici uniquement des principales mesures employées dans le monde de la recherche d'information à ce jour : cosinus de vecteurs TF*IDF, OKAPI (BM25) ou encore plus récemment « *Language Model »* (LM). Lors des compétitions TREC 15 et 16, toutes les équipes ont obtenu leur meilleur résultat avec BM25 ou LM.

La mesure TF*IDF est une mesure de pondération d'un texte par rapport à la collection entière. Cette mesure fait appel à la fréquence TF de chaque terme au sein du document, multipliée par sa valeur IDF qui représente la valeur de fréquence inverse du mot par rapport au corpus total. Ainsi, un mot très fréquent dans la langue de manière globale - comme les mots-outils 'et', 'de', 'tu' - aura une valeur IDF très faible, car on les retrouve partout. Les mots discriminants qui ne se retrouvent que dans quelques documents auront quant à eux une valeur IDF significativement plus élevée. Associés à la fréquence, ce sont donc les mots discriminants (qui apparaissent un nombre de fois important dans une partie des documents) qui auront la valeur TF*IDF la plus élevée. Pour effectuer la comparaison entre document et requête, les deux sont transformés en vecteurs et sont comparés grâce au cosinus. Cette similarité s'exprime de la manière suivante :

$$\omega_{t,d} = tf_{t,d} \,.\, log \frac{|D|}{|\{d' \in D | t \in d'\}|} \quad (2.)$$

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

$tf_{t,d}$ étant la fréquence du terme t dans le document d.

$$idf = log \frac{|D|}{|\{d' \in D | t \in d'\}|} \quad (3.)$$

idf étant la fréquence inverse du document par rapport à la collection.

Pour chaque terme, on calcule sa valeur TF*IDF en fonction de sa fréquence et du nombre d'occurrences dans les documents de la collection. On pourra ensuite avec ce vecteur calculer sa similarité avec la requête avec la mesure de cosinus suivante :

$$\mathrm{sim(d, q)} = \frac{d \cdot q}{\|d\| \|q\|} \quad (4.)$$

en considérant q le vecteur de la requête, d le vecteur du document.

La mesure OKAPI (BM25) est une variante de la mesure TF*IDF qui prend en compte notamment la longueur du texte.

Celle-ci s'exprime de la manière suivante :

$$Score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \quad (5.)$$

Pour chaque terme q_i dans chaque document D on calcule la fréquence $f(q_i, D)$ qui représente donc la fréquence du mot dans le document, on utilise également la longueur moyenne d'un document de la collection $avgdl$. Les paramètres k_1 et b sont des paramètres libres pour l'optimisation.

Cette mesure a notamment comme avantage par rapport à une mesure classique TF*IDF de faire saturer l'impact de la fréquence d'un terme. En effet, le terme k_1 permet de choisir si la saturation doit être rapide ou lente. L'autre nouveauté qu'apporte la mesure BM25 en regard de la mesure TF*IDF est le fait de normaliser en fonction de la longueur moyenne des documents de la collection. Elle permet d'éviter ainsi de donner trop d'importance à des documents beaucoup plus longs (fréquence des mots plus importante) ou beaucoup plus courts que la moyenne (poids plus important en proportion). Cette mesure a été testée à de nombreuses reprises [13] et a fait l'objet notamment de réussites au cours des compétitions TREC, en s'imposant peu à peu comme l'état de l'art dans de nombreuses catégories.

Enfin, parmi les mesures les plus utilisées, il y a la modélisation du langage ou « *Language Model* », qui fait appel notamment au raisonnement bayésien qui cherche à classer les documents non par une approche uniquement fréquentiste, mais en fonction de la probabilité de générer la requête en fonction des mots que contient le document. Il s'exprime de la manière suivante dans sa forme lissée par la distribution de Dirichlet [22]:

$$\left(\mu p(w_1|C), \mu p(w_2|C), \dots, \mu p(w_n|C)\right) \tag{6.}$$

avec C la collection de documents et μ le prior de Dirichlet.

Le modèle prend la forme suivante après lissage :

$$p_\mu(w|d) = \frac{c(w;d) + \mu p(w|C)}{\Sigma_w c(w;d) + \mu} \quad (7.)$$

avec $c(w;d)$ le compte du mot w dans le document d. Et $p(w|C)$ le modèle de langage de la collection.

Il s'agit ainsi d'avoir un modèle de langue, tel qu'on peut l'avoir dans un classificateur. Pour ce faire, on utilise en général un modèle N-gramme. Un modèle N-gramme de type unigramme se matérialise par la probabilité de chaque terme d'apparaître indépendamment dans le document. Autrement dit, si l'on considère l'exemple du

Tableau *3-2* : deux documents et une requête à deux mots. Si le terme 1 est doté d'une probabilité à 0.1 dans le document 1, le terme 2 d'une probabilité à 0.2 dans le document 1 et respectivement 0.2 dans le document 2 et 0.01 dans le document 2, alors le premier document sera mieux classé, car le produit des deux probabilités indépendantes sera supérieur. En général, on appliquera un lissage pour éviter les probabilités égales à 0. On pourrait également appliquer une modélisation complexe de type bigramme ou trigramme, mais la complexité de la modélisation ne représente pas forcément la proximité avec une requête contenant peu de mots. De plus, la représentation sous forme d'unigrammes présente comme atout le fait d'être plus facilement applicable à grande échelle, ce qui est très important en recherche d'information.

Document	Probabilité terme 1	Probabilité terme 2	Probabilité combinée
Document 1	0.1	0.2	0.02
Document 2	0.2	0.01	0.002

Tableau 3-2. Modèle unigramme

Si la manière dont est présentée la problématique de recherche dynamique est nouvelle dans la forme proposée par la compétition TREC avec la catégorie « Domaine dynamique », d'autres formes existent déjà avec notamment des recherches interactives cherchant à maximiser le rappel et la précision en utilisant le retour de l'utilisateur. On peut également mentionner la recherche par session, qui cherche à modéliser les recherches d'un utilisateur au cours d'une session grâce à un processus itératif qui se raffine au fur et à mesure des différentes requêtes effectuées au cours de chaque session. Ces différentes approches de la recherche dynamique, bien qu'elles n'intègrent pas toutes la notion de satisfaire les besoins de l'utilisateur du point de vue de la diversité des concepts et dans une limite de temps, partagent un certain nombre de points communs, qui sont autant de pistes à explorer afin d'établir l'état de l'art dans le domaine de la recherche d'information dynamique.

Outre ces techniques de recherche d'information, nous explorons d'autres algorithmes qui font intervenir une modification de la représentation de la matrice terme-document.

L'indexation sémantique latente est une technique utilisant la décomposition en valeurs singulières. Elle permet de diminuer notamment la dimension de notre matrice terme-document ce qui permet notamment de lutter contre le problème de la polysémie et de la synonymie qui sont les deux faiblesses de la recherche booléenne traditionnelle (ex. les termes « *USA* » et « *America* » seront considérés comme deux mots différents alors qu'ils partagent des caractéristiques de contexte communes)

Enfin l'utilisation des plongements lexicaux ou *« word embeddings »* est à l'origine de nouvelles techniques de recherche d'information. Tout comme l'indexation sémantique latente, ils

permettent de lutter contre la synonymie. En effet, les plongements lexicaux sont une représentation, sous forme d'un vecteur d'une taille choisie (entre 50 et 300 le plus souvent), de l'espace des mots. Dans le cas de la technique la plus répandue *Word2Vec*, on obtient ces vecteurs en utilisant un réseau de neurones qui apprend pour chaque mot à prédire un vecteur de la taille choisie précédemment, en lui donnant le contexte du mot avec comme objectif de prédire le mot au milieu de ce contexte. Les vecteurs résultants de cet apprentissage peuvent ensuite être utilisés pour des relations sémantiques et syntaxiques. Ainsi, on retrouvera les différentes formes d'un verbe souvent proches ou, sémantiquement, ses synonymes.

Les motivations pour mener des expérimentations autour des plongements lexicaux sont nombreuses. Il s'agit d'une part d'une représentation permettant de diminuer la complexité des documents ou phrases, en ayant des vecteurs de taille acceptable permettant d'effectuer des comparaisons entre des groupes de mots de plus ou moins grande taille. Et par ailleurs, cette modélisation nous apporte une connaissance directe du corpus en nous fournissant des informations sur le contexte des mots. En outre, des expérimentations préliminaires sur certains mots nous ont permis de constater un certain nombre de résultats intéressants pour l'utilisation de *Word2Vec* [15]. Nous détaillons ces expérimentations dans la section consacrée aux plongements lexicaux.

3.2.3 Système manuel ou automatique

Par « système automatique », on entend que le moteur de recherche ne va à aucun moment chercher de l'information par un humain situé dans la boucle du système. C'est-à-dire que le système ne « triche » pas en allant chercher des informations du « *ground truth* ». Quasiment tous les systèmes présentés sont donc automatiques. Néanmoins, les règles de la compétition permettant aux équipes de présenter plusieurs systèmes différents, le RMIT [1] a proposé un système qui va chercher l'information du « *ground truth* ». Durant la phase de recherche d'information, l'équipe utilise donc la note de pertinence du document avant de le soumettre à l'utilisateur. Ils écartent ainsi les documents

non pertinents et obtiennent une forme de système idéal, dont le score serait à atteindre si nous étions capables de modéliser de façon précise les centres d'intérêts de notre utilisateur. Les résultats montrent que malgré cette information, la métrique du *CubeTest* (métrique officielle de la compétition TREC définie dans la section 3.3.2.1 de cette revue de littérature) reste assez faible et que le système, bien que meilleur que les autres, demeure dans un ordre de grandeur similaire. Cette expérimentation montre donc que le challenge est complexe, car même en ayant uniquement des documents pertinents, il reste difficile d'alimenter l'utilisateur de manière suffisamment diversifiée et rapide.

3.2.4 Diversification des résultats

L'un des objectifs de la recherche dynamique est de couvrir plusieurs sous-sujets qui pourraient intéresser notre utilisateur. Dans ce cadre, il est important de ne pas lui fournir des documents trop similaires, qui répondraient à un même aspect de la requête initiale. Pour ce faire, on a recours à des techniques permettant de diversifier les résultats. Dans le cas de [12] et [17], cette diversification peut prendre la forme d'un nouveau classement en fonction de la diversité qu'apporte un document. Ils vont dans ce cas sélectionner les *n* premiers documents retrouvés par le moteur de recherche et leur appliquer une nouvelle cote en fonction de la diversité. Il en ressortira un nouveau classement parmi les *n* documents relatifs à la requête initiale.

Pour [12] ce reclassement prend la forme du *« relevance feedback »* (RM3) qui s'exprime de la manière suivante :

$$P(d|q) = (1 - \lambda)P_{old}(d|q) + \lambda * SimScore(d, Feedback) \qquad (8.)$$

où q est la requête, d le document et $P_{old}(d|q)$ la probabilité avant le reclassement, calculée par le « *language model* ». λ un paramètre libre et $SimScore$ une mesure de similarité entre le document et le passage surligné par l'utilisateur.

[17] définissent plusieurs méthodes de diversification des résultats, tout d'abord ils proposent une *« flat diversification »* (ou diversification à plat), avec s qui représente un sous-sujet. Leur but est donc de mesurer l'intérêt de la requête vis-à-vis de cet aspect, puis de mesurer comment chaque document couvre cet aspect.

$$div_x(q, d, \mathcal{D}) = \sum_{s \in S} P(s|q) \cdot P(d|q, s) \prod_{d_j \in \mathcal{D}} \left(1 - P\left(d_j|q, s\right)\right) \quad (9.)$$

où $P(s|q)$ est l'importance relative du sujet s par rapport à la requête q et $P\left(d_j|q, s\right)$ la couverture du document d d'un des sujets s par rapport à la requête q.

Leur seconde méthode est appelée *« hierarchical diversification »* (ou diversification hiérarchique) : ici les aspects sont classés de façon hiérarchique et par la suite chaque document est noté en fonction de sa couverture d'aspect sur ces différents niveaux hiérarchiques. Il en résulte :

$$div_i(q, d, \mathcal{D}) = \sum_{s \in S_i} P(s|q) \cdot P(d|q, s) \prod_{d_j \in \mathcal{D}} \left(1 - P\left(d_j|q, s\right)\right) \quad (10.)$$

Enfin leur dernière méthode est une extension des précédentes, ramenant l'équation à l'exploitation de plusieurs sources K alors que les précédentes méthodes font l'hypothèse que les aspects proviennent d'une seule source :

$$div_M(q, d, \mathcal{D}) = \sum_{k \in K} \theta_k div_k(q, d, \mathcal{D}) \quad (11.)$$

où k est une source et θ_k le poids correspondant à la source.

D'autres équipes ont essayé des méthodes de diversification basées sur des algorithmes tels que l'allocation de Dirichlet latente (LDA). Sur un jeu de données de nouvelles de sites d'informations, [2] nous proposent, pour faire de la recherche d'information dynamique, d'utiliser l'allocation de Dirichlet latente de manière à faire de l'expansion de requête. Ils prennent notamment les 10 meilleurs mots pour chaque sujet déterminé par l'algorithme et effectuent une nouvelle requête grâce à ces mots.

Une autre manière d'utiliser l'allocation de Dirichlet latente a été étudiée par [18]. Dans leur cas, l'algorithme n'est pas utilisé pour faire de l'expansion de requête, mais pour regrouper les documents retrouvés par affinité avec les différents sujets de l'algorithme. C'est une forme de regroupement souple *(« soft clustering »*). Les documents sont donc divisés en cinq groupes – comme le nombre de documents que l'on peut retourner au système à chaque étape – et le meilleur document de chaque groupe est recommandé. Cette démarche suit celle que nous recommandions dans l'édition TREC 15, à la différence que nous proposons un regroupement dur à l'aide de l'algorithme des *K*-Moyennes.

Parmi les méthodes existantes pour parvenir à diversifier les résultats il existe notamment la possibilité d'exploiter l'information des plongements lexicaux ou *« word embeddings ».* Les travaux autour des *« word embedding »* ont été nombreux dans le domaine du traitement automatique du langage naturel depuis les travaux de *Word2vec* [16]. Ainsi, les plongements lexicaux ont notamment été utilisés avec succès dans les systèmes de recommandation, dans l'analyse sémantique et dans la recherche d'information.

Dans ce dernier domaine, différentes approches ont été tentées, notamment l'expansion de requête [19]. Pour ce faire, les auteurs ont notamment comparé l'expansion de requête à deux moments distincts. Soit en amont (« *pre-retrieval* ») : il s'agit alors d'utiliser les mots de la requête et de prendre les mots les plus semblables par rapport à leur vecteur. Soit en aval (« *post-retrieval* ») : en utilisant un « *pseudo-feedback* » – tel que nous l'utilisons avec notre système lorsque nous

appliquons l'algorithme LDA sur les 20 premiers résultats. Cette approche considère ici les premiers résultats comme pertinents et effectue l'expansion à partir des termes retrouvés dans ces documents uniquement. Enfin, ils ont également essayé une technique incrémentale qui consiste à choisir à chaque itération le meilleur terme pour l'ajouter ensuite à la requête et ainsi refaire une expansion jusqu'à atteindre le nombre de termes désiré. Cette dernière technique permet d'éviter à leur algorithme de dériver, car chaque terme est proche de la requête.

3.2.5 Retour d'utilisateur et expansion de requête

Parmi les techniques régulièrement utilisées en recherche dynamique, il y a notamment l'expansion de requête. Cette technique a pour objectif d'améliorer notre capacité à cibler des documents relatifs au sujet principal de la requête en utilisant les termes retrouvés au sein des documents notés comme pertinents par l'utilisateur. Ainsi, si l'on prend comme exemple une requête très vague comme « *Apple* » et que l'on soumet cette requête à un moteur de recherche, celui-ci fera apparaître aussi bien des documents relatifs à la marque informatique qu'au fruit. Cependant, si l'utilisateur indique que parmi les cinq premiers documents, trois traitant de la marque informatique sont non-pertinents et que deux contenant des informations sur le fruit sont pertinents, alors logiquement les documents parlant d'une pomme devraient contenir des mots tels que « *fruit* » ou d'autres termes décrivant le fruit plutôt que la marque. De même, les documents relatifs à la marque, considérés comme non pertinents, contiennent des mots qui ne sont pas dans les documents relatifs au fruit, ce qui nous permet d'établir que ces mots-là doivent être négativement pondérés.

C'est ainsi que l'on peut définir le « *relevance feedback* », qui dans notre cas est appliqué au niveau vectoriel. Dans le cas du modèle vectoriel, il est représenté par l'algorithme de Rocchio qui considère la requête, les documents pertinents et les documents non pertinents. Chacun sera vectorisé et la multiplication des vecteurs par des facteurs permettra de donner un poids positif aux mots

contenus dans les documents pertinents et qui ne sont pas présents dans les documents non pertinents.

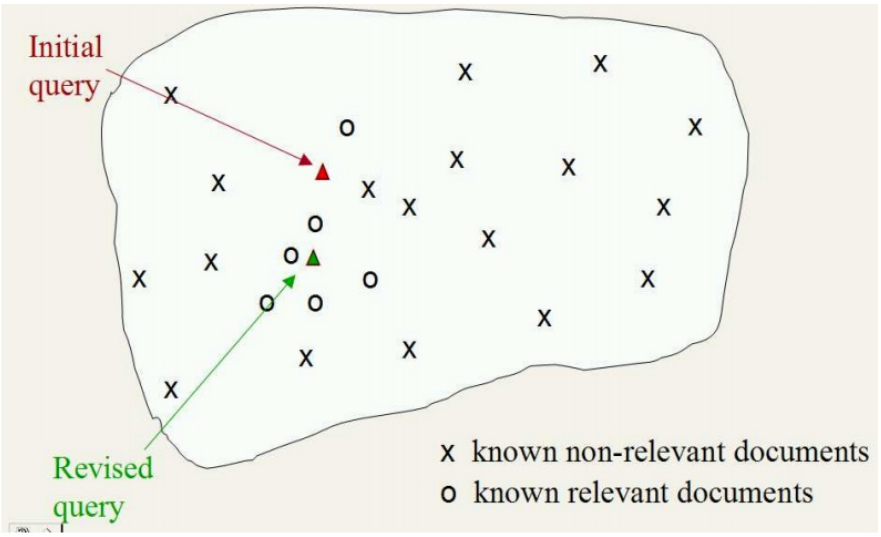

Figure 3-2. Algorithme de Rocchio (Manning et al., 2008)

La technique du « *relevance feedback* » a été notamment utilisée dans le but de reclasser les documents de la première itération en fonction du modèle de langue utilisé dans les passages relevés comme pertinents.

L'algorithme de Rocchio illustré à la Figure 3-2 a été utilisé par plusieurs équipes lors des éditions TREC 15 et TREC 16 [20], [1]. *Beijing University of Posts and Telecommunications* [20] a pour sa part utilisée la similarité en calculant des vecteurs de sacs de mots en fonction des valeurs TF*IDF pour déterminer après la première itération les documents les plus similaires aux documents relevés comme pertinents par l'utilisateur. Le système supprime également les documents les plus similaires aux documents notés comme non pertinents.

3.2.6 Critère d'arrêt

Le critère d'arrêt est l'une des composantes principales du principe de système dynamique, dans le sens où le système doit être capable de savoir quand s'arrêter, afin que l'utilisateur n'ait pas à intervenir dans le processus de recherche d'autre manière qu'en fournissant son avis sur les documents reçus. Cependant, ce critère d'arrêt, comme souligne [20] n'est accompli que dans très peu de cas par les systèmes, qui vont plutôt utiliser soit une limite brute, soit des critères qui ne sont pas « intelligents », mais plutôt prédéterminés, tels qu'une fenêtre d'itérations sans trouver de document pertinent. Nous traiterons de ce problème dans le Chapitre 7 consacré à cette question qui nous a semblé importante. Néanmoins, durant nos travaux soumis pour les éditions TREC 15 et 16, nous avons principalement travaillé autour des itérations 1 et 2 (soit la recherche initiale et une recherche itérative avec le retour de l'utilisateur) pour pouvoir nous comparer aux autres systèmes, nous proposons également des pistes pour évaluer un système en termes de cumul de l'information. Ceci afin de déterminer le total d'information capté durant une recherche afin de voir si un système s'arrête au bon moment.

Nous avons parlé précédemment de limites prédéterminées pour décider quand s'arrête un tel système. C'est ainsi qu'a procédé par exemple [12], en décidant qu'un utilisateur après cinq pages sans voir de résultat abandonnerait la recherche. Une approche du même type, mais un peu différente a été testée par [17]. Dans leur travail, ils comptabilisent le nombre de documents non pertinents puis cumulent le nombre de documents sur la session et se donnent une limite du nombre de documents non pertinents par session. Dans leurs expérimentations, les membres de cette équipe ont comparé cette approche avec une approche fixe (fixer le nombre d'itérations à 10, quel que soit le déroulement de la session). Ils ont également fait varier la manière de prendre en compte les documents non pertinents. En effet, ils ont deux manières de réaliser cette approche : comptabiliser le nombre de documents non pertinents sur la totalité de la session ou comptabiliser le nombre de documents non pertinents contigus.

Cette deuxième approche permet d'éviter de s'arrêter tant qu'on trouve des documents pertinents et de ne s'arrêter que si l'on semble tourner en rond. Dans leur comparaison, ils ont cherché à évaluer, sur une évaluation du *CubeTest* à dix itérations, quelle méthode donnait le meilleur score. Ils ont ainsi comparé tous les algorithmes avec les 3 méthodes d'arrêt et c'est la comptabilisation de dix documents sur l'ensemble de la session non pertinents qui offrait le meilleur résultat. Cependant, il nous faut ici rappeler que le *CubeTest* est une mesure dégressive et que dès lors, le critère qui s'arrête le plus tôt est forcément celui qui va maximiser le score. Le critère consistant à considérer dix documents sur l'ensemble de la session est plus restrictif qu'un critère prenant en compte dix documents non pertinents à la suite, le système va donc avoir tendance à s'arrêter plus tôt. On peut constater cet effet sur la mesure *n*-DCG qui à l'inverse est une mesure de quantité d'information qui est croissante. Dans ce cas, c'est le critère fixe à 10 itérations qui donne le meilleur score.

Tous ces critères sont bruts et ne considèrent finalement à aucun moment l'émergence d'un nouveau sujet capté, ou bien le fait que l'information, bien que pertinente, soit déjà saturée vis-à-vis d'un sujet.

3.3 Évaluation des systèmes

L'évaluation d'un système dynamique présente des similarités avec l'évaluation d'un système de recherche d'information classique, mais comprend également un certain nombre de spécificités. En effet, les indicateurs de classement des documents et la précision sont deux éléments importants pour tout type de système de recherche d'information, puisqu'avoir un système qui retourne une majorité de documents non pertinents n'est pas souhaitable. Cependant, la précision dans un système dynamique n'a pas forcément vocation à être l'intérêt principal de l'utilisateur. Si l'on se réfère aux fondements de cette compétition, il s'agit notamment de répondre aux besoins d'utilisateurs professionnels de la recherche. Contrairement au grand public, ces utilisateurs ont la patience et la

volonté de guider le système pour lui permettre de s'ajuster grâce à leur retour, afin de maximiser la recherche d'information sur l'ensemble de la session et non pas uniquement sur les premiers documents. On pourrait donc tout à fait imaginer un système qui effectue une recherche générale, fournit les cinq meilleurs documents – dont on peut penser qu'une majorité est pertinente, mais pas forcément spécifique aux différents aspects de la recherche – et qui s'arrêterait relativement tôt en fournissant une précision élevée. À l'inverse, un système cherchant à diversifier les résultats et proposant des documents parlant de divers sujets autour de la requête pourrait proposer moins de documents pertinents, mais couvrir plus d'éléments reliés au thème de la requête.

3.3.1 Les méthodes d'évaluation de recherche d'information traditionnelle

3.3.1.1 Précision

La précision est la mesure qui consiste à refléter le nombre de documents pertinents parmi l'ensemble des documents retournés ; elle se définit comme suit :

$$precision = \frac{|D \cap P|}{|D|} \tag{12.}$$

Nous notons D le groupe de documents retournés par le système et P le groupe de documents pertinents.

3.3.1.2 Rappel

Le rappel est la mesure qui cherche à calculer le nombre de documents pertinents trouvés parmi le nombre total des documents pertinents existants. Ainsi, si par exemple on retourne à l'utilisateur l'ensemble des documents, on disposera d'un rappel maximal.

$$rappel = \frac{|D \cap P|}{|P|} \tag{13.}$$

3.3.1.3 Précision à k documents

Comme nous venons de le voir, on peut avoir une valeur de rappel très importante en retournant l'ensemble des documents ou avoir une précision très importante en ne retournant qu'un seul document, celui-ci étant le premier du classement. Cependant, à l'instar de la mesure F1 pour combiner précision et rappel en apprentissage statistique, nous utilisons une sorte de compromis en recherche d'information en utilisant la précision à *k* documents, puisque contrairement à un problème de classification, nous ne pouvons pas considérer qu'un système est bon s'il retourne l'ensemble des documents. Pour une recherche par requête il y a effectivement un nombre important de documents, mais aucun utilisateur n'ira lire 200 documents même s'ils sont tous pertinents à un certain degré.

$$precision@k = \frac{|D[1..k] \cap P|}{k} \tag{14.}$$

Afin de mesurer l'efficacité d'un moteur de recherche, en général on estime qu'il faut que les résultats soient pertinents dans les premières pages. Parmi ces valeurs, sont en général utilisées [4] $k = 5$, 10 ou 20.

3.3.1.4 Position réciproque (« Reciprocal Rank »)

Dans le cas de la précision à k documents, un problème apparaît concernant l'ordre d'apparition des documents puisque seule la pertinence est considérée. Ainsi, en fixant par exemple, k = 20, on pourrait se retrouver avec deux systèmes :

- L'un retournant les meilleurs documents en premiers ;
- L'autre retournant les mêmes documents, mais dans un ordre moins bon.

Bien que ces deux systèmes soient équivalents d'après cette mesure, il serait souhaitable de faire une distinction. C'est pourquoi l'on introduit la notion de position réciproque. Cette nouvelle mesure considère que le besoin de l'utilisateur est satisfait dès que le ou les documents les plus pertinents sont trouvés.

La mesure de la position réciproque est définie comme suit :

$$RR = \frac{1}{\min\{k \mid D[k] \epsilon P|\}} \qquad (15.)$$

Ainsi un système qui trouve les cinq documents pertinents parmi 20 dans les 10 premières positions aura une mesure supérieure à un système qui les trouve entre la 15e et la 20e place.

3.3.2 Mesures d'évaluation propres à la recherche dynamique

3.3.2.1 CubeTest

C'est dans cette idée qu'a été créée la mesure du *CubeTest* par [11]. Cette mesure consiste en une analogie avec un cube d'information à propos d'un sujet (une requête) divisé en sous-cubes représentant les sous-sujets qui intéressent notre utilisateur simulé. Chaque document comporte une note de contribution à chaque sous-sujet. Par exemple, un document A peut contribuer avec une note de 5 au sous-sujet 1 de la requête 2 et avoir une note de 2 au sous-sujet 2 et ne pas contribuer aux sous-sujets 3 et 4.

Cette mesure fonctionne avec une hauteur qui consiste à exprimer que l'information est complète, ainsi il sera inutile de fournir cinq documents ayant une note de 4 pour un même sous-sujet. À l'inverse du système qui aura fourni cinq documents avec une note de 3, mais chacun traitant d'un sous-sujet différent sera récompensé par la diversité qu'il aura su amener. Il y a donc une réelle prise en compte de la couverture des intérêts par les documents, là où les mesures traditionnelles de recherche d'information ne tiennent compte que de la pertinence du document.

Cette mesure prend également en compte le temps écoulé durant la recherche, afin de pénaliser un système qui chercherait à effectuer un nombre infini d'itérations pour maximiser le gain. Cette pénalité de temps est la raison pour laquelle le *CubeTest* est une métrique décroissante. Le but est donc de minimiser la pente. Elle diffère des mesures, telles que DNCG (*Normalized Discounted Cumulative Gain*), qui sont des mesures croissantes et qui cherchent à maximiser le gain d'information.

Elle est définie comme suit :

$$CT(Q, D) = \frac{Gain(Q, D)}{Time(D)} \tag{16.}$$

où Q est l'information nécessaire et D la liste du document. Le gain (*« Gain »*) est l'information cumulée en fonction de la liste des documents et le temps (*« Time »*) est le temps écoulé au cours de la liste de documents D.

Les mesures fonctionnent comme la mesure du volume changeant pendant une période de temps. Plus la mesure *CubeTest* est élevée, plus le système est efficace. Le fait que le *CubeTest* tienne compte de la période de temps signifie que, dans le cas où le système a besoin de plusieurs « tours » pour trouver des documents, le score sera pénalisé (au lieu de trouver des documents dès le début du processus et d'arrêter la recherche dans les premiers tours).

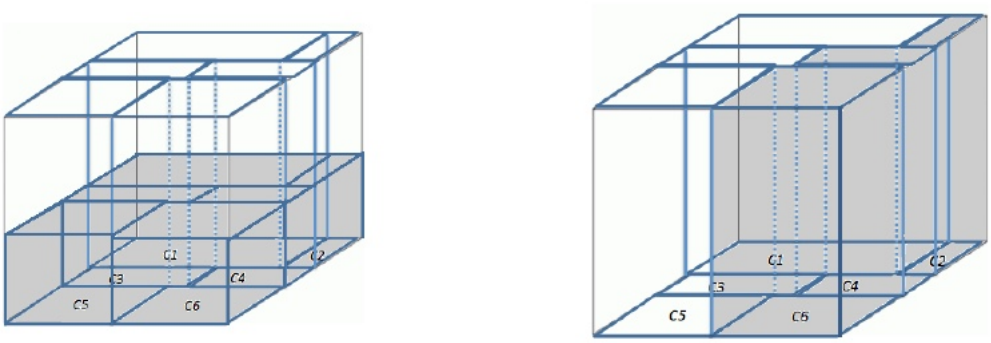

Figure 3-3.CubeTest (Luo et al., 2013)

On peut ainsi voir sur la Figure 3-3 la comparaison entre deux systèmes : celui de gauche qui aura trouvé de l'information sur tous les sujets même si leur quantité d'information est moyenne, et celui de droite qui aura trouvé beaucoup d'information sur un petit nombre de sujets. C'est le premier type de système qui aura le meilleur score, car il offre une meilleure couverture des intérêts de l'utilisateur.

3.3.2.2 µERR (« µ-Expected Reciprocal Rank »)

Cette redondance de l'information a également été utilisée dans la recherche d'information à travers la mesure *µERR* [5]. Cette mesure a pour objectif de sanctionner les documents redondants en donnant une pénalité à un document fournissant moins d'information que le précédent (là où le *CubeTest* ne fait que saturer la valeur). Ainsi, pour adapter la mesure à la recherche dynamique, la méthode est appliquée pour chaque sous-sujet. Il s'agit de pénaliser un document apportant une note de 1 sur un sous-sujet, alors qu'on avait déjà fourni à l'utilisateur un document avec une note de 4. Cette technique permet d'éviter que les systèmes ne cherchent à rapporter à tout prix des documents pertinents, même s'ils sont peu informatifs. Cette mesure est une variante de la position réciproque évoquée dans la précédente section.

Elle est définie comme suit :

$$ERR := \sum_{r=1}^{n} \frac{1}{r} P(\text{l}'\text{utilisateur s}'\text{arrête à la position r}) \qquad (17.)$$

où *n* est le nombre de documents dans le classement et P la précision jusqu'au rang r.

Chapitre 4 - Indexation et prétraitements

Un système de recherche d'information, qu'il soit dynamique ou non, contient plusieurs étapes obligatoires. Parmi celles-ci l'indexation et le prétraitement sont des étapes qui permettent de rendre l'information à la fois disponible, mais aussi de la transformer pour rendre certains algorithmes plus performants.

En effet, un document texte tel qu'un article de site internet contient différentes informations, telles que la date, le titre de la page, le nom du site internet, l'article en lui-même extrait en HTML.

La première question lors de la création d'un système de recherche d'information est donc de savoir ce qu'on souhaite ajouter à notre index. La plupart des algorithmes de recherche traditionnels ont pour but d'effectuer des recherches sur le texte dans le but de faire correspondre des mots-clés avec le contenu de l'article. Ainsi, les champs importants pour un moteur de recherche sont principalement le titre et le contenu de l'article. En effet bien que la date puisse être utile pour classer les éléments en fonction de leur date d'apparition ou bien établir la fiabilité d'une source internet, ces problématiques sont spécifiques à certains thèmes. Pour certaines requêtes, il pourrait être intéressant de reclasser les documents en fonction de leur date (par ex. regrouper les articles parlant d'un sujet aux environs d'une même date).

L'indexation n'est pas non plus figée au contenu déjà présent, mais peut également faire l'objet de nouveaux champs intégrés par l'utilisation d'un prétraitement. On peut par exemple utiliser des algorithmes permettant d'extraire les mots-clés d'un article ou bien d'effectuer un résumé de l'article et proposer ainsi d'indexer un article d'une page internet avec 4 champs : son titre, son résumé, ses mots-clés et son contenu.

Dans notre cas, nous avons fait le choix d'utiliser uniquement l'information textuelle, à savoir le titre et le contenu HTML, pour effectuer nos analyses. En outre, nous proposons une analyse de l'intégration de contenu supplémentaire comme le résumé et les mots-clés des articles.

La seconde question lors de l'indexation est de savoir sous quelle forme l'on souhaite stocker l'information. On peut en effet conserver l'information sous sa forme brute, il s'agit donc de stocker le code HTML, ou bien effectuer un prétraitement en utilisant diverses techniques permettant d'extraire le contenu textuel d'une page HTML.

En outre, le texte lui-même une fois extrait peut faire l'objet de différents traitements.

4.1 Extraction du contenu de l'article

L'extraction de l'article entend supprimer les éléments non relatifs à l'article de la nouvelle qu'on cherche à analyser. Il ne s'agit donc pas simplement de supprimer les balises HTML et le code JavaScript d'une page, mais également de supprimer tout ce qui apparaît dans les colonnes ou menus d'un site internet ainsi que les pieds de pages, etc. qui contiennent de l'information que l'on peut assimiler à du bruit (voir Figure 4-1).

Figure 4-1. Corps de l'article (exemple).

Le prétraitement constitue une étape importante de tout travail en recherche d'information et traitement du langage naturel. Il s'agit d'une étape qui permet d'éliminer les caractères non conventionnels et de normaliser le texte dans le but d'obtenir un corpus fidèle au corpus initial, mais

qui nous permettra d'appliquer les algorithmes dans un environnement leur permettant de converger vers la meilleure solution.

Prenons par exemple le cas d'un corpus contenant des pages internet d'un journal en ligne. Si nous n'effectuons pas de prétraitement et prenons le contenu HTML des pages en retirant uniquement les tags, ce qui constitue un prétraitement minimal : dans ce cas, l'ensemble des articles apparaissant dans un même « cadre » de type actualités ou encore les informations pour se connecter vont apporter du bruit par rapport aux mots de l'article ciblé. Un algorithme qui vise à regrouper les éléments textuels va faire le lien entre ces parties qui apparaissent dans chaque document et considérer qu'il s'agit d'un concept.

Suite à ce constat, nous avons décidé de nous orienter vers la solution *BoilerPipe*[4], qui est un extracteur d'articles entraîné sur des pages internet. Cet outil permet donc, avec une page HTML en entrée, de retourner uniquement le contenu de l'article principal de la page. Cette extraction a été testée en comparaison avec un corpus sans prétraitement (Tableau 8-2).

4.2 Résumé d'articles et extraction de mots-clés

L'un des problèmes évoqués dans le prétraitement pour extraire l'information d'un article est la présence de texte hors contexte, de publicités, de pollution par des scripts côté client (JavaScript). Bien que l'outil *BoilerPipe* permette de traiter une partie de ces problématiques, il reste qu'il ne permet pas de gérer le contenu par rapport à sa cohérence avec le reste du texte. L'idée est donc qu'un algorithme de résumé de texte permet de faire émerger les principales phrases d'un article et de laisser de côté des éléments qui ne sont pas liés à l'idée principale.

[4] https://github.com/kohlschutter/boilerpipe

Pour ce faire, nous utilisons un algorithme d'extraction des phrases importantes, [14], avec la librairie *Gensim*. Cet algorithme permet par un système de graphes, d'établir les phrases ou les mots-clés qui ont le plus de poids dans le graphe. Pour les mots-clés, ce poids est déterminé en fonction de fenêtres de cooccurrences, en fonction de la fréquence des cooccurrences de certains termes, ils vont avoir un poids plus important et être connectés à un grand nombre de termes. D'autre part, il y a un système de filtrage en fonction de la forme syntaxique, de sorte que les termes classiquement fréquents tels que les pronoms, les mots-outils ne soient pas représentés dans le graphe.

Pour ce qui est de classer les phrases et les relier par un système de graphe, il s'agit d'une fonction de similarité entre les phrases par recoupement des mots apparaissant dans les phrases qui permet de créer un graphe de relation avec un poids plus ou moins important en fonction de cette similarité.

Les phrases et les termes ayant le plus de poids dans leurs graphes respectifs sont ensuite sélectionnés. Dans notre cas, nous avons utilisé les 10 premiers termes pour les mots-clés, et 10% des phrases.

Pour illustrer l'utilisation de cet algorithme, prenons par exemple la page Wikipédia anglaise du virus Ébola. Nous sélectionnons le paragraphe de présentation du virus et le passons dans l'extracteur de mots-clés. Nous obtenons en sortie les 10 mots suivants : *« including », « include », « includes », « outbreaks », « outbreak », « fluid », « fluids », « fevers », « infected », « services ».* Nous pouvons constater deux choses dans cet exemple. Premièrement il est important d'utiliser la lemmatisation ou le *stemming* en amont pour éviter les répétition de mêmes termes sous différentes formes tels que « *fluid* » ou « *fluids* ». D'autre part, même si les termes trouvés ne couvrent pas la totalité du sujet, ils permettent effectivement d'avoir une idée du contenu de l'article.

4.3 Plongements lexicaux ou « *word embeddings* »

Le prétraitement fait principalement intervenir des techniques dites non supervisées, parmi celles-ci la technique des *« word embeddings »* est une technique souvent utilisée en traitement du langage naturel afin de modéliser un corpus. Cette modélisation permet de transformer les mots en vecteurs. Ces vecteurs permettent donc par similarité de cosinus de trouver des synonymes, des analogies ou encore des paragraphes similaires en effectuant la somme des vecteurs des mots d'un paragraphe.

Les plongements lexicaux sont obtenus par l'utilisation d'un réseau de neurones composé de trois couches : la couche d'entrée composée des mots, la couche cachée qui comporte les poids du réseau et la couche de sortie dont la taille est choisie au préalable. Ce dernier paramètre est particulièrement important, car la taille de la couche de sortie détermine également la taille du réseau et donc sa capacité à apprendre. Empiriquement, un réseau étroit est relié à une meilleure capacité à rapprocher des mots syntaxiquement proches (souvent un verbe sous ses différentes formes, des pluriels, etc.), alors qu'un réseau plus large est associé en général à des vecteurs pouvant capter des similarités sémantiques (synonymes, analogies). Ce réseau de neurones est entraîné selon l'un des deux modèles existants : CBOW (ou « sac de mots continu ») ou SKIP-GRAM. Dans le premier cas, le réseau construit les vecteurs en apprenant à prédire un mot en fonction de son contexte, alors que dans le second modèle, le réseau apprend à prédire le contexte en fonction du mot d'entrée (tel qu'illustré à la Figure 4-2).

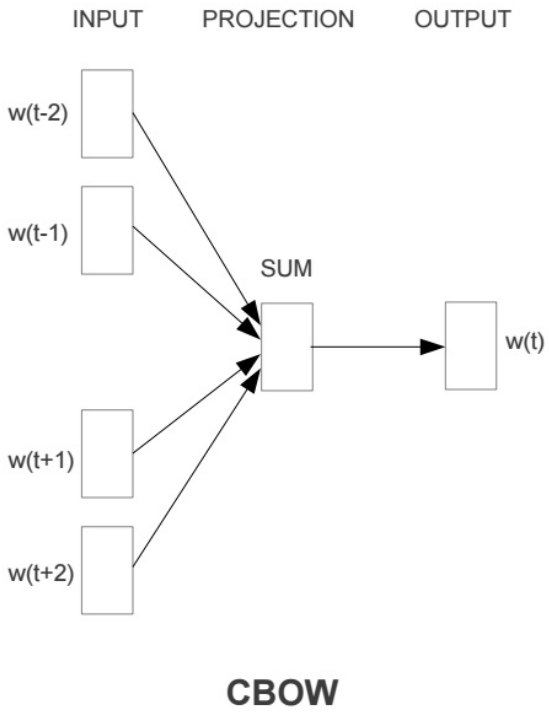

Figure 4-2. CBOW (Mikolov, Chen, Corrado, & Dean, 2013)

Ainsi si l'on prend un exemple simple tel que la phrase suivante : *Le jeune garçon mange une pomme*. Dans le cas de l'algorithme CBOW, on fixe une fenêtre d'analyse (par ex. 2 mots avant, 2 mots après). Ainsi on aura à partir de cette seule phrase des exemples d'entraînement pour notre réseau de neurones (Tableau 4-1) :

Le réseau de neurones est entraîné avec les colonnes X en couche d'entrée et la colonne Y en couche de sortie. En pratique, on fixe habituellement la taille de la couche de neurones cachés entre 50 et 300. Cette couche de neurones cachés sera notre matrice de poids, qui va permettre de multiplier un mot en entrée par cette matrice et d'avoir en sortie un vecteur de la taille du nombre de neurones cachés.

X				Y
		jeune	garçon	le
	le	garçon	mange	jeune
le	jeune	mange	une	garçon
jeune	garçon	une	pomme	mange
garçon	mange	pomme		une
mange	une			pomme

Tableau 4-1.Exemples d'entrainement

Cette représentation a donc pour objectif de transformer un mot par les poids d'une matrice de contexte. Ceci permet non seulement d'avoir des termes synonymes, mais également de faire des analogies, dont voici un exemple : *« Kaci Hickox - Pauline Cafferkey + Glasgow = Maine »*. Pour mieux comprendre cette analogie il faut savoir que Kaci Hickox et Pauline Cafferkey sont deux infirmières qui ont été mises en quarantaine suite à leur retour d'Afrique. La première est américaine, la seconde écossaise. En appliquant notre analogie, on constate qu'on peut retrouver le lieu d'origine de Kaci Hickox en utilisant les informations concernant Pauline Cafferkey qui vient de Glasgow. On peut constater une petite différence dans le sens où l'on part d'une ville et que l'analogie se fait avec une région, cependant le résultat reste pertinent.

Ces plongements lexicaux peuvent donc être entraînés de plusieurs manières. Soit par CBOW: il permet ainsi de générer le mot en fonction de son contexte. Soit à l'inverse pour le modèle SKIP-GRAM, on cherchera à partir d'un mot à prédire son contexte. Ce dernier modèle est reconnu comme plus performant, par expérimentations sur des jeux de données importants. Le modèle CBOW, effectuant plus d'exemples d'entraînement à partir d'un échantillon, peut être meilleur sur des jeux de données de taille modeste.

Ces modèles peuvent également avoir différentes tailles de vecteurs. Nous choisissons de mener l'expérimentation avec des tailles dites « standards » de 50 et 300 qui sont en générale les bornes hautes et basses des expérimentations autour des distributions de mots vectorielles.

Cette modélisation permet d'utiliser notamment des techniques de recherche vectorielle à partir de comparaisons des vecteurs de mots, ou de vecteurs de phrases.

Une fois le corpus modélisé, chaque mot apparaît comme un vecteur et il est donc possible de rechercher ses *k* plus proches voisins. Par exemple, pour le mot « *military* » la liste des plus proches voisins sera la suivante : « *army* », « *soldiers* », « *troops* ». On a également la possibilité de combiner les vecteurs, en ajoutant à « *military* » le mot « *response* ». Cette association aura comme plus proches voisins les termes suivants : « *deployment* », « *deploy* », « *fight* ». Cet exemple nous montre qu'avec un ensemble de termes, nous obtenons des termes synonymes du concept de « *military response* » alors que dans le cas du terme seul « *military* » nous obtenions des synonymes correspondant à ce terme en particulier.

Cependant, cette modélisation n'est pas parfaite et peut faire apparaître un certain nombre de dérives. Si l'on prend l'exemple du terme « *Obama* » qui est associé au Président des États-Unis durant la période de la crise d'Ebola. On retrouve, parmi les trois plus proches voisins, le terme « *Obola* ». Ce terme n'est autre qu'une utilisation des internautes complotistes, qui étaient persuadés que le président B. Obama était à l'origine de la crise d'Ebola. D'un point de vue de modélisation, ce résultat est tout à fait juste puisque le mot « *Obola »* est utilisé dans les mêmes contextes que le mot « *Obama* » sur des sujets parfois similaires, ce qui crée une proximité entre les deux termes. Cependant, ce genre de résultat pose des problèmes en termes d'expansion de requête, car il réfère à un groupe particulier d'articles traitant l'information de manière très spécifique. Si ce genre de résultat est très intéressant pour des requêtes sur le complotisme durant la crise d'Ebola, il risque aussi d'amener du bruit dans le cas d'une requête sur l'action du Président Obama durant cette crise.

4.4 Recherche par passage, découpage en phrases

Le prétraitement effectué nous permettant d'avoir uniquement le vrai contenu textuel de la page, nous pouvons envisager de réaliser une segmentation du corpus en phrases. La motivation de cette démarche est le fruit d'une réflexion autour des requêtes concernant les personnes ou les organisations que nous considérons comme des requêtes à entités nommées.

Voici l'exemple d'une requête portant sur une personne : « *Daniel Berehulak* ».

- *"Daniel Berehulak is a freelance photographer represented by Reportage by Getty Images"*
- *"For the past six weeks photographer Daniel Beherulak has been covering the virus' deadly spread for the New York Times"*
- *"When Berehulak first arrived in Monrovia on Aug. 22 – armed with 300 pairs of gloves 35 Personal Protective Equipment suits goggles surgical face masks hand sanitizers and countless rolls of tape – he met with Getty Images photographer John Moore who had been covering the Ebola crisis for a week."*

Dans cet exemple, nous pouvons distinguer trois passages qui ont été désignés comme pertinents. Chacun d'entre eux est pertinent vis-à-vis d'un sous-concept différent. La première réponse concerne la personnalité de Daniel Berehulak, la seconde ses activités et la troisième ses contacts. Le dénominateur commun de ces passages est le fait qu'une partie de la requête apparaisse au sein de la phrase et que chaque type de sous-sujet ait un marqueur linguistique. Les contacts auront souvent comme lien le verbe « *meet* » et, dans les passages décrivant les activités du photographe, on trouvera des verbes d'action et des marqueurs temporels. Ces différences sont donc notables principalement à un niveau de granularité aussi fin que celui de la phrase. Dans les phrases évoquées précédemment, on peut aisément remarquer qu'un article faisant référence au travail du photographe ne parlera pas forcément du journaliste lui-même, mais plutôt de son récit et de la situation qu'il dévoile. Ainsi le contenu vraiment important par rapport à notre requête se retrouve noyé dans le document et n'occupe

qu'une phrase ou deux. Ce constat justifie ainsi le fait d'explorer la possibilité de traiter les phrases individuelles comme des documents à part et d'effectuer une modélisation des sujets sur les phrases au lieu des documents. Nous proposons une analyse de cette expérimentation au chapitre 8.

4.5 Conclusion sur l'indexation et le prétraitement

Dans ce chapitre, nous avons pu voir différentes techniques permettant de prendre en compte des particularités des documents et comment normaliser ceux-ci afin de maximiser la probabilité de les faire retrouver par des algorithmes de recherche d'information. Ces différentes techniques sont importantes pour permettre notamment de combattre les problèmes tels que la synonymie, ou la polysémie. Cependant, elles restent des outils permettant uniquement aux algorithmes de recherche d'information d'élargir ou de restreindre leur spectre afin de trouver les documents les plus pertinents. Des techniques telles que la racinisation, peuvent aussi bien permettre de retrouver un document qui aura un mot dont la forme est rare, mais peut aussi empêcher de retrouver un document précis, car il utilise justement une forme rare. C'est donc un travail d'équilibre entre intérêt pour l'utilisateur en général, et certains besoins spécifiques.

Chapitre 5 - Recherche d'information statique et dynamique

La recherche d'information dans le cas d'un système interactif et dynamique – tel que nous souhaitons concevoir dans nos travaux – contient deux phases distinctes. Pour la phase d'initialisation, lors de la première requête de l'utilisateur, l'information dont on dispose est la même qu'un système classique de recherche d'information requête-index. Une fois cette première étape réalisée, vient la phase de recherche d'information dynamique, qui consiste à prendre en compte le retour de l'utilisateur afin de trouver des documents qui collent au mieux avec son intérêt de recherche.

Nous retrouvons ici la manière dont nous procédons pour l'étape de recherche d'information, qui consiste à retrouver un certain nombre n de documents pertinents par itération (par ex. 5). Pour ce faire, nous avons utilisé les trois méthodes suivantes.

5.1 Modèle de langage

Nous utilisons le moteur de recherche Solr comme base de référence en recherche d'information avec comme mesure de similarité le modèle de langage (« *Language Model* »), pour récupérer les documents ayant donné une requête : soit la requête initiale de l'utilisateur, soit les requêtes reformulées générées lors de l'étape de raffinement dynamique. Les cinq principaux documents retournés sont conservés sous la forme de la recommandation Solr, les cinq documents les plus pertinents étant donné la requête. Dans le même temps, les n meilleurs documents retournés par Solr sont utilisés comme corpus pour les algorithmes de modélisation de sujet LDA et *K*-Moyennes. Ces deux algorithmes mettent en œuvre la diversification complémentaire des résultats.

5.2 Allocation de Dirichlet latente (LDA)

L'allocation de Dirichlet latente (LDA) est un algorithme qui prend une distribution de mots et découvre les regroupements de sujets qu'il implique. Autrement dit, cet algorithme est capable de faire émerger d'un corpus de documents des concepts clés qui ne sont pas propre à un document unique, mais qui peuvent se retrouver dans plusieurs paragraphes de différents documents. De cette manière, l'algorithme permet de dégager les grands axes d'une requête assez générale pour orienter les recherches futures dans un système dynamique. Dans notre système, nous avons utilisé cet algorithme pour regrouper en cinq groupes de sujets différents les n principaux documents retournés par Solr à partir de la requête en cours. Le système crée alors cinq requêtes étendues, une pour chacun des cinq groupes de sujets, en ajoutant les cinq mots les plus probables de la distribution de chaque groupe par rapport à la requête en cours. Il exécute ensuite une nouvelle recherche Solr pour chacune des cinq nouvelles requêtes et conserve le document le plus pertinent pour chaque requête.

On peut résumer notre procédure par la suite d'opérations suivante :

1. Effectuer une requête par *« Language Model ».*
2. Garder les n premiers documents.
3. Appliquer l'algorithme LDA en fixant à k le nombre de sujets.
4. Garder les mots les plus représentatifs pour chaque topic.
5. Effectuer une nouvelle requête en utilisant les mots de chaque sujet (soit une requête par sujet).
6. Recommander le meilleur document de chacune de ces requêtes.

5.3 *K*-Moyennes (« *K-Means* »)

K-Means est un algorithme de regroupement (ou *« clustering »*) non supervisé (il est présenté à la Section 2.5.2). Contrairement à LDA, K-Means ne permet pas de faire émerger des concepts présents au sein d'un document, mais vise plutôt à identifier les documents qui ont des contenus similaires. Ainsi un document parlant majoritairement d'économie sera jugé connexe à d'autres documents traitant également de ce sujet. La détection de ces similarités permet d'éviter de proposer deux documents redondants tel que des nouvelles qui reprennent la même information. Dans notre système, nous l'utilisons pour créer cinq groupes de documents à partir du corpus de n documents retournés par la méthode de recherche *« Language Model »* (ou modélisation du langage). Le système conserve le document le plus proche du centroïde de chaque *« cluster »* pour créer l'ensemble de cinq recommandations de *K*-Moyennes.

On peut ainsi résumer notre procédure par la suite d'opérations suivante :

1. Effectuer une requête par *« Language Model ».*
2. Garder les n premiers documents selon leur score, obtenus à la précédente requête.
3. Appliquer l'algorithme des *K*-Moyennes sur cet ensemble de n documents.
4. Récupérer les vecteurs des k centroïdes calculés par l'algorithme.
5. On recommande le document le plus proche de chacun de ces centroïdes, donc k documents, en calculant pour chaque centroïde sa proximité avec les n premiers documents.

5.4 Combinaison

Après ces étapes, notre système a généré trois ensembles de cinq documents recommandés au moyen d'une requête. En outre, ces ensembles peuvent contenir certains documents en commun. Nous avons donc décidé de créer une combinaison pondérée des documents retrouvés parmi ces trois ensembles. Nous présentons l'architecture de cette combinaison à la Figure 5-1.

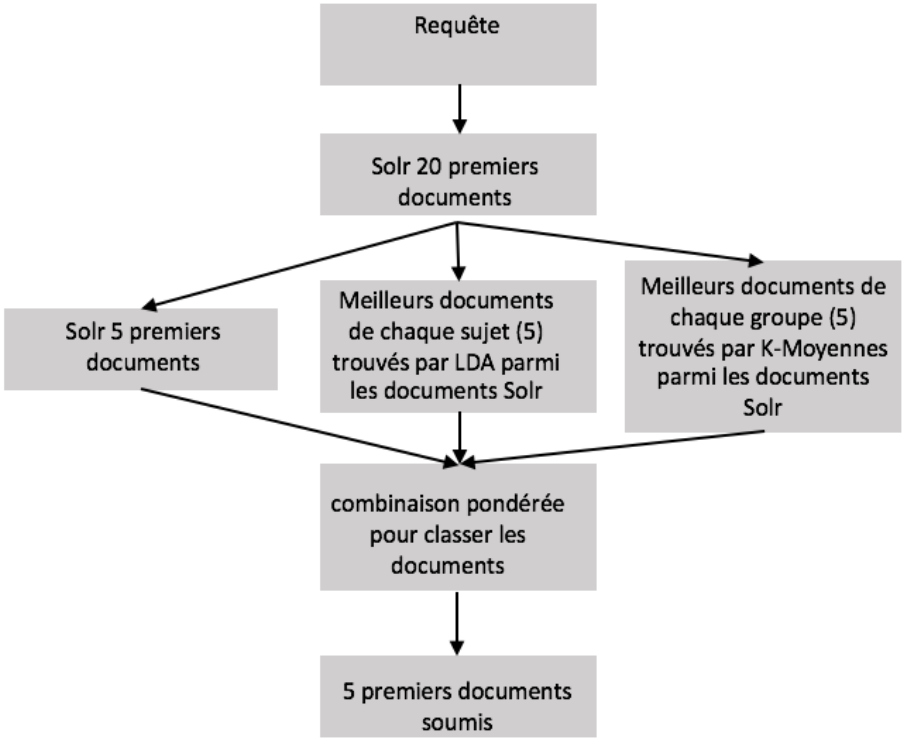

Figure 5-1. Processus de recherche d'information

Ces pondérations ont fait l'objet d'expérimentations que nous présentons dans le chapitre « Expérimentations et résultats ». L'intuition derrière cette pondération est le fait que certains

algorithmes sont meilleurs pour éviter les redondances, tandis que d'autres font émerger des concepts cachés. Les algorithmes de base sont en général meilleur pour faire correspondre les documents avec les mots-clés de la requête initiale. Nous voulons évaluer si la combinaison de ces trois algorithmes, chacun donnant leur « vote » sur le classement du document, permet de mieux couvrir l'information sur une majorité de critères.

5.5 Méthodes de recherche à partir des *« word embeddings »*

Nous proposons ici deux méthodes pour exploiter les *« word embeddings »* (ou plongements lexicaux).

La première approche est une approche que l'on pourrait qualifier de naïve, qui consiste à utiliser la modélisation en utilisant notre modèle *Word2Vec* entraîné sur les données concernant le sujet « Ebola » à partir du jeu de données TREC uniquement. Ainsi, nous allons utiliser pour chaque mot de notre requête initiale les trois mots les plus proches d'après notre modèle dans le corpus. Ainsi notre requête concernant l'armée américaine contiendra aussi la notion de « troupes », « army » et « humanitarian crisis ».

La seconde approche est plus « mathématique » : elle consiste à comparer les vecteurs des mots de notre requête aux vecteurs des mots des titres des documents. Ainsi, nous allons utiliser une technique similaire au *« word moving distance »* [9]. Pour ce faire, nous prenons pour chaque mot de la requête, le mot qui a le vecteur le plus proche de lui parmi tous les mots du titre et nous calculons leur similarité. Enfin, nous cumulons la meilleure similarité pour chaque mot de la requête pour attribuer une note de proximité avec le document. Ce qui va nous permettre ensuite de classer ces documents selon cette distance.

Enfin, la troisième approche est celle faite au niveau des phrases (comme présenté dans la section 4.4). Un document va être découpé en phrases et on va moyenner les vecteurs de mots de

chaque phrase et mesurer leur similarité avec la moyenne de la requête. Ceci nous permet d'éviter des calculs trop lourds comme la comparaison mot à mot décrite dans le paragraphe précédent qui ne concerne que deux phrases (titre et requête) contre des documents qui contiennent parfois plusieurs dizaines de phrases.

5.5.1 Expansion de requête

Notre première méthode consiste donc à utiliser une requête initiale donnée par l'utilisateur, et de prendre les mots indépendamment les uns des autres. Pour chaque mot de la requête, nous allons le transformer en vecteur via notre modèle *Word2Vec* (soit celui que nous avons entraîné, soit celui de Google News). Pour chaque vecteur, nous pouvons rechercher les mots les plus proches dans notre collection. Il s'agit d'une simple mesure de similarité par le cosinus des deux vecteurs.

Nous proposons dans notre étude deux modèles distincts entraînés avec *Word2Vec*. L'un est déjà entraîné sur les données des nouvelles de Google (qui incluent la crise d'Ebola notamment). De l'autre côté, nous avons entraîné notre propre modèle à partir de zéro, sur les données TREC 16 avec environ 200 000 pages internet parlant du sujet Ebola. Nous avons décidé de nous concentrer sur ce sujet, car nous avons un modèle Google News, qui permet de nous confronter à un modèle qui connaît a priori le sujet, alors que sur des sujets tels que les produits illicites ou bien les données polaires les résultats pourraient être plus compliqués aussi bien à comparer qu'à créer, le jeu de données étant moins exhaustif.

Notre modèle est entraîné à partir de la méthode CBOW, avec une fenêtre de 8 mots et un minimum d'occurrence de 3 afin d'éviter les mots présents dans un seul document, qui pourraient venir fausser le modèle.

5.5.2 Reclassement des documents en fonction de leurs vecteurs de mots

Notre seconde application des *« word embeddings »* dans le contexte de la recherche d'information est l'application d'une mesure de similarité du cosinus entre chaque mot de la requête et chaque phrase d'un document. Cependant ce processus étant coûteux, nous proposons différentes solutions :

- Appliquer la similarité uniquement au titre, réduisant ainsi les documents à une seule phrase ;
- Appliquer le reclassement uniquement sur un échantillon de 200 à 500 documents retrouvés grâce à une autre méthode de recherche d'information *(« Language Model »* ici). Et le calcul d'une moyenne de vecteur pour chaque phrase en fonction des différents vecteurs de mots.

Ces deux solutions permettent de voir l'impact que peut avoir ce type de comparaison entre requête et document sous forme de vecteurs.

La première solution s'effectue en utilisant chaque mot de notre requête initiale sous forme de vecteur et chaque mot de notre titre de document en face. Nous mesurons la similarité par le cosinus entre chaque mot et effectuons la somme des similarités les plus fortes entre chaque mot de la requête avec chaque mot du titre.

La seconde solution se matérialise par une somme des vecteurs des mots de chaque phrase ou de chaque requête. Cette somme nous donne un nouveau vecteur qui va nous permettre de comparer rapidement un document à une requête.

5.6 Conclusion sur les méthodes de recherche d'information

Les méthodes de recherche d'information contribuent aussi bien à la recherche d'information initiale qu'aux recherches itératives d'un système dynamique. Elles permettent d'assurer le fait de retrouver des documents pertinents par rapport à la requête, mais peuvent également permettre de diversifier les résultats. Nous proposons ici une variété de méthodes prenant aussi bien en compte les techniques de regroupement que les méthodes permettant d'extraire des sujets de façon non supervisée, ainsi que des méthodes à partir de plongements lexicaux. Cependant, ces méthodes de recherche et de diversification ont pour but de découvrir les intérêts de l'utilisateur, mais le traitement du retour de l'utilisateur permet quant à lui d'exploiter ces intérêts afin de trouver des documents toujours plus pertinents.

Chapitre 6 - Exploitation du retour de l'utilisateur

6.1 Expansion de requête à partir du retour de l'utilisateur

Le processus de raffinement dynamique consiste, pour notre système, à prendre en compte les retours de l'utilisateur sur l'ensemble des cinq documents récupérés et utiliser ces retours pour améliorer les résultats. Il y a deux situations possibles dans lesquelles le système pourrait être utilisé. La première est celle où l'utilisateur a marqué au moins un document comme pertinent. Nous pouvons considérer que nous sommes dans la bonne zone du domaine à rechercher. La deuxième situation est celle dite du « faux départ » (Figure 6-2), lorsqu'il n'y a aucun document pertinent dans l'ensemble des cinq résultats.

Dans la première situation, notre algorithme de raffinement dynamique profite des informations de rétroaction positive pour générer une nouvelle requête. Notre algorithme a accès à différentes sources d'information, notamment le titre du sujet et le passage du document marqué comme pertinent par l'utilisateur.

Nous avons utilisé l'extraction des mots de titre, considérant qu'il s'agissait d'une partie très instructive de la rétroaction. En raison de l'importante variation de longueur entre les passages surlignés, il ne nous a pas paru judicieux de prendre en compte l'entièreté des passages. Ainsi, afin de nous concentrer sur les parties les plus informatives, nous avons choisi d'utiliser les fonctions NER (« *Named Entity Recognition* ») de NLTK[5] pour capturer les entités nommées, en considérant que la présence d'une entité nommée au sein d'un passage signifie que cette entité peut avoir une importance ou un sens relatifs au sous-sujet connexe.

[5] http://www.nltk.org/

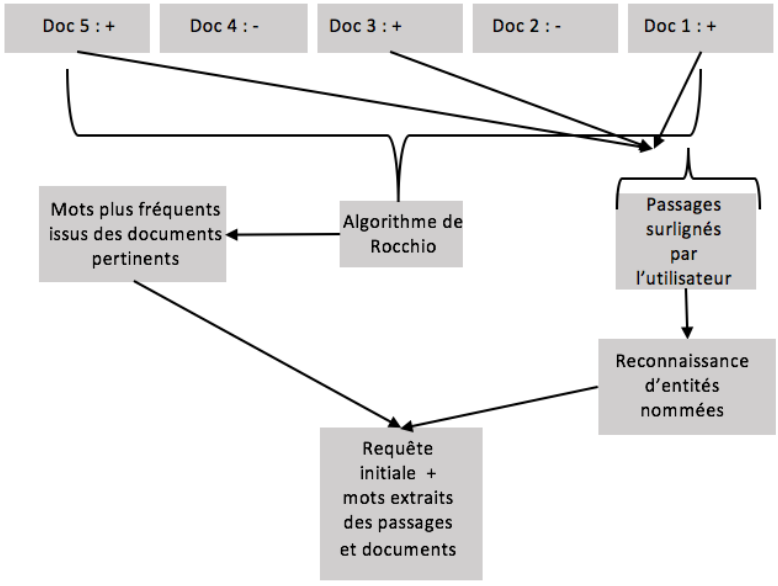

Figure 6-1.Traitement des retours de l'utilisateur

Nous avons également utilisé l'extraction de mots via l'algorithme de Rocchio appliqué sur nos cinq documents. Cet algorithme fonctionne en appliquant un poids négatif pour les termes présents dans les documents négatifs et en stimulant les mots présents dans les documents positifs. Avec un modèle vectoriel TF*IDF, on peut extraire des mots communs et informatifs à partir de documents positifs. Avec ces nouveaux mots, on crée une nouvelle requête, qui sera utilisée sur notre processus statique en tant que requête initiale pour fournir cinq nouveaux documents à l'utilisateur.

Nous résumons ce procédé de traitement du retour de l'utilisateur dans la Figure 6-1.

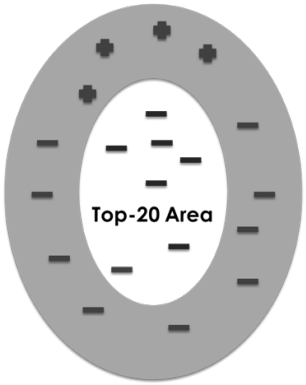

Figure 6-2. Situation du faux départ

Dans la deuxième situation, dite du « faux départ », l'utilisateur n'a noté aucun des cinq documents précédemment retournés comme pertinents. Cette situation est difficile, et ce pour deux raisons. Premièrement, nous sommes dans une mauvaise zone et il pourrait être difficile de trouver un document pertinent. Deuxièmement, aucun document positif n'a été trouvé et le système a donc perdu du temps dans le processus de récupération de documents. La métrique *CubeTest* prend en compte le temps et par conséquent, il s'agit de trouver plus rapidement les documents pertinents pour améliorer les performances du système.

La situation du « faux départ » signifie que l'algorithme s'est retrouvé dans une mauvaise zone de l'espace des documents (Figure 6-2). Cette zone correspond au fait que notre requête n'est proche que de documents non-pertinents, les mots-clés utilisés lors de la requête ne sont donc pas suffisants, car associés à des documents non relatifs à l'intérêt de notre utilisateur. Il est alors essentiel de passer à une zone différente. Nous utilisons pour cela l'algorithme de retour de pertinence de Rocchio sur les cinq documents, afin d'en extraire les mots-clés les plus courants. Ces mots-clés reçoivent un poids négatif dans une nouvelle requête Solr, afin de récupérer des documents très différents dans le prochain processus de recherche d'information. Nous avons constaté que cette approche donne de meilleurs résultats que celle qui consiste à prendre les documents les mieux classés de la requête précédente.

Le résultat de cette approche peut s'illustrer de la manière suivante : si l'on considère une requête telle que *« Apple »* si notre algorithme ne tombe que sur des documents parlant de pommes, alors que l'on cherchait des articles sur la marque, les mots communs vont bien souvent être fruit, jus, etc. En mettant un poids négatif sur ces mots on a de grandes chances de faire ressortir des articles relatifs à la marque.

Cependant, le problème de notre jeu de données est que l'on ne peut pas considérer les documents non jugés comme non pertinents : ils peuvent en effet contenir de l'information relative à notre sujet, mais n'ont tout simplement pas été annotés.

L'utilisation d'une telle technique peut donc mener à tout simplement s'éloigner d'une zone qui était bonne, mais qui contenait beaucoup de documents non jugés.

6.2 Exploitation du retour de l'utilisateur par similarité de vecteurs de paragraphes

Comme discuté précédemment, les plongements lexicaux (*« word embeddings »*) peuvent également être utilisés au niveau des paragraphes. La démarche est similaire, mais l'algorithme est entraîné à prédire un paragraphe ou un document en lieu et place d'un mot [10]. Nous la nommerons de la manière suivante : *Doc2Vec*.

L'objectif en utilisant cet algorithme est de l'entraîner à un niveau de granularité différent pour voir comment nous pouvons prendre en compte le retour de l'utilisateur. En effet, on sait que l'utilisateur nous fournit un degré de pertinence par rapport à un passage dans le document. On a donc le choix d'utiliser l'ensemble du document ou bien uniquement le passage surligné.

Pour ce faire, on peut donc découper notre corpus de deux façons différentes et entraîner un modèle *Doc2Vec* de la librairie *Gensim* de deux manières :

- Pour chaque document, on effectue un découpage en phrases et on considère chaque phrase comme un paragraphe et donc comme un document à part ;
- On entraîne l'algorithme sur le document au complet.

Le principal problème de cette dernière approche dans la théorie est le fait que les documents sont très différents en termes de taille et peuvent donc contenir différents sujets. Un document traitant de trois sujets ne sera donc peut-être pas aussi proche de certains documents par la variété de son contenu. Alors que l'approche par phrase qui est une des granularités les plus fines pour comparer des groupes de mots devrait permettre de retrouver des documents parlant d'un passage précis.

Le traitement du retour de l'utilisateur est donc un élément capital de la recherche d'information dynamique, il est également un terrain de recherche riche et comportant une grande difficulté du fait de la rareté des retours et du besoin d'un système efficace rapidement. Cependant, si l'on souhaite traiter le retour de l'utilisateur, il faut également savoir à quel moment y mettre un terme par un critère d'arrêt.

Chapitre 7 - Critère d'arrêt, exploitation et méthode d'évaluation

Comme présenté dans la vue d'ensemble d'un système de recherche d'information dynamique (Figure 3-1), la seconde phase de notre système est une boucle qui se clôt uniquement sur un critère d'arrêt. Le système doit donc être capable de décider quand arrêter la recherche et sortir de cette boucle lorsqu'une itération supplémentaire ne produira plus un raffinement utile des résultats de recherche.

7.1 Introduction

Durant nos deux participations à TREC 15 et TREC 16 nous avons essentiellement travaillé sur des systèmes qui n'utilisaient pas de critère de recherche « intelligent ». Nous avons utilisé des limites codées en dur de 2 à 10 itérations pour voir comment le système se comportait.

Afin de déterminer un critère de recherche, il faut s'attarder sur le but de la tâche. L'objectif de la recherche dynamique est de couvrir l'intérêt de l'utilisateur sur des sous-sujets spécifiques.

Si nous voulons construire une manière d'établir que le système a recueilli suffisamment d'information, nous devons donc prendre en compte la quantité d'information pour chaque sous-concept si l'ensemble des sous-concepts ont été découverts de manière satisfaisante.

7.2 Algorithme

Notre méthode consiste à utiliser à chaque itération de la recherche la connaissance dont nous disposons par rapport au retour de notre utilisateur. Comme on l'a vu précédemment, à chaque étape l'utilisateur nous fournit à la fois la pertinence du document (1 à 4) et le concept auquel il est rattaché. On peut donc déterminer à partir de cette information combien au minimum il existe de concepts et le degré de pertinence relatif à chaque concept.

De cette manière, on peut mesurer un gain relatif pour chaque concept et en faire la moyenne. De cette notion de gain, découle un gain par itération, ce qui nous permet par une méthode de recuit-simulé en établissant l'intérêt de l'utilisateur à une température de base, qui est stimulée par le gain ou non d'information à chaque étape. Cette notion de température permet à la fois de satisfaire le besoin du système de recueillir de l'information en début de recherche et de voir la notion d'intérêt décroître si après n itérations l'utilisateur n'a vu aucun document pertinent.

Nous modélisons ainsi la baisse de l'intérêt de l'utilisateur par une exponentielle négative, ainsi la valeur de notre température est multipliée par cette exponentielle négative.

Dans notre cas nous utilisons une température initiale de 1000 avec une exponentielle négative avec un coefficient de 2, afin de correspondre environ à 4 itérations d'initialisations qui devraient permettre dans une grande partie des cas de découvrir au moins un sujet afin de stimuler notre mesure de l'intérêt.

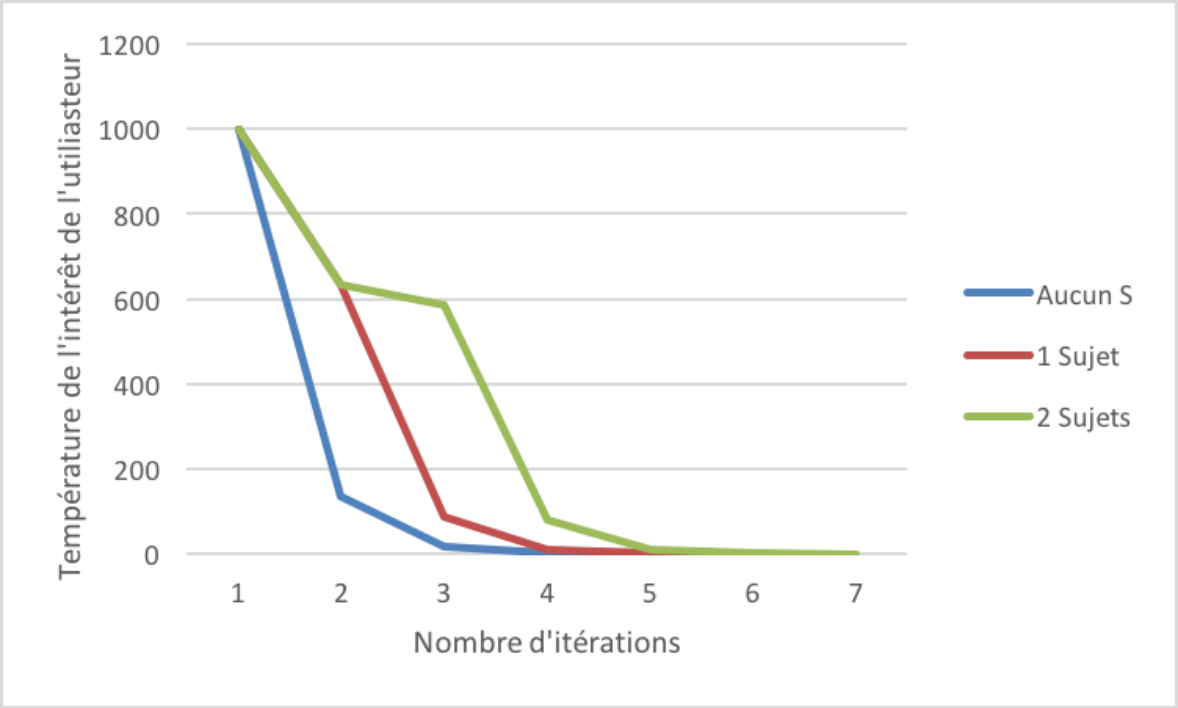

Figure 7-1. Évolution de la température en fonction du temps

Nous pouvons voir à la Figure 7-1 qu'une requête pour laquelle nous ne trouvons aucun document pertinent et donc aucun nouveau sujet, se termine après 4 itérations, en ayant fixé le seuil de la température minimale à 1 pour continuer. Dans notre figure, nous pouvons constater que si l'on trouve un sujet après la 2e itération, nous allons freiner notre baisse de température qui va permettre d'avoir une itération supplémentaire à la fin. De même pour un 2e sujet découvert lors de l'itération 4 pour notre dernière courbe.

Cette technique de recuit-simulé, qui consiste donc à stimuler la température lors de nouvelles découvertes de sujets intéressants, permet donc d'interrompre des recherches qui ne semblent donner aucun résultat, et permet de continuer la recherche lors de la découverte de nouveaux sujets intéressants.

7.3 Évaluation

L'évaluation d'un tel critère est complexe, car il fait appel à des notions qui peuvent être subjectives. Ainsi, l'abandon du système n'est pas forcément en accord avec l'intérêt de l'utilisateur qui aurait été prêt à observer quelques itérations supplémentaires. Le choix de la température initiale peut donc être un facteur important.

Ainsi, dans notre cas l'évaluation cherche principalement à s'accorder avec la dégressivité du *CubeTest*. Cependant, cette mesure ne suffit pas seule pour interpréter le résultat d'une accumulation d'information, car sa dégressivité est très rapide (division par le nombre d'itérations), l'ajout d'information est donc divisé par 2 lors de la seconde itération, 3 lors de la troisième et ainsi de suite. En effet, nous rappelons que le *CubeTest* se définit comme suit :

$$CT(Q,D) = \frac{Gain(Q,D)}{Time(D)}$$

le temps (*« Time »*) étant équivalent au numéro de l'itération dans notre cas.

Nous proposons donc deux variantes additionnelles liées au *CubeTest* pour aider à mieux apprécier l'apport d'information sur la durée.

Nous proposons tout d'abord un contrepoids plus progressif :

$$PCT = \alpha \cdot CT + (1-\alpha) \cdot Itération \qquad (18.)$$

Il s'agit d'un *CubeTest* positif, nous l'appellerons *Positive CubeTest* (PCT). α est un paramètre libre, que nous fixons empiriquement à 0.5 dans nos expérimentations. Ce second terme a pour objectif d'encourager l'algorithme à prolonger la recherche malgré la dégressivité du *CubeTest.*

Cette mesure a pour avantage de garder la propriété du *CubeTest* en termes de gain d'information tout en apportant une dégressivité du gain en fonction des itérations.

En outre, nous proposons une seconde alternative qui consiste simplement au cumul du *CubeTest* ou *Cumulative CubeTest* (CCT) avec une pénalité de temps nulle. Il s'agit donc d'une mesure purement de gain d'information qui permet d'apprécier le gain total d'un système, cela permet de mieux interpréter notamment si 10 itérations vont apporter une information réellement bien supérieure ou qu'à partir d'un certain nombre d'itérations finalement le gain total est très faible.

7.4 Résultats et analyses

Nous pouvons analyser les courbes de *CubeTest* (Figure 7-2) sur le nombre d'itérations en fonction de la méthode du « coude » : celle-ci consiste à voir les principales cassures en termes de progression d'une courbe. Cette cassure permet de repérer les seuils qui engendrent une progression plus lente et donc de potentiels points d'arrêts de la recherche. On peut notamment constater que notre métrique PCT semble effectivement tenir son rôle de contrepoids positif à la mesure CT tout en ayant une progressivité plus lissée que la dégressivité du CT. Notre métrique CCT permet de voir la quantité totale d'information accumulée et l'on peut noter que la cassure correspond parfaitement au PCT, ces deux mesures offrent donc une corrélation intéressante pour l'analyse des systèmes dynamiques. En revanche, la mesure ACT donne une courbe très lisse, ou il semble difficile de prendre une décision quant au choix d'un moment ou arrêter la recherche.

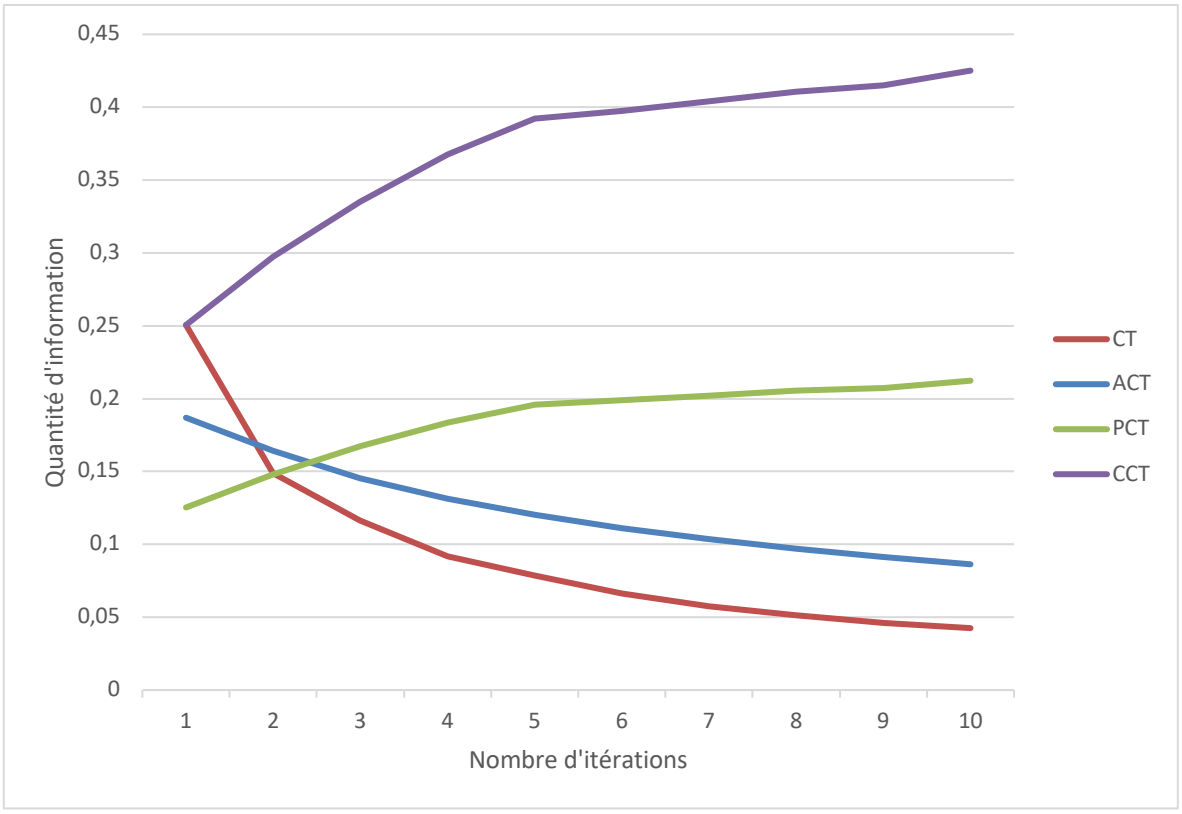

Figure 7-2. Progression des mesures d'information en fonction du nombre d'itérations

En utilisant notre algorithme nous pouvons observer à la Figure 7-1 que le système s'arrête automatiquement après quatre itérations s'il n'a trouvé aucun document de par la dégressivité qu'implique notre intérêt décroissant modélisé par une exponentielle négative. D'autre part si durant les deux premières itérations un document est trouvé, le regain d'intérêt de l'utilisateur va permettre de chercher une ou deux pages supplémentaires, mais on observe que dans très peu de cas nous atteignons l'itération 6 ou plus. Il est effectivement très rare qu'un nouveau sujet intéressant vis-à-vis des intérêts de notre utilisateur simulé apparaisse après la 4e itération. Nous pouvons globalement séparer les sujets en « simples » et « difficiles ». C'est ainsi que dans de nombreux cas la plupart des sujets sont découverts relativement rapidement, ou bien ne le sont pas du tout.

7.5 Conclusion des travaux sur le critère d'arrêt

En conclusion, nous pouvons observer que notre méthode par recuit-simulé semble offrir un critère d'arrêt raisonnable entre exploration et exploitation. Elle permet de continuer à explorer si l'utilisateur a trouvé de nouveaux documents pertinents par rapport à un nouveau sujet. Élément qui, dans les autres systèmes, n'était jusqu'ici pas pris en compte puisque les autres systèmes n'étudiaient que la pertinence ou non d'un document pour estimer s'il fallait continuer la recherche, alors que cette pertinence peut déjà être saturée vis-à-vis d'un sous-sujet déjà couvert.

Nous offrons ici une nouvelle manière d'aborder un critère intelligent, il reste cependant complexe d'évaluer la qualité d'un critère d'arrêt par rapport à une mesure dégressive, car plus le système s'arrête tôt, meilleure est sa note s'il ne découvre pas de nouvelle information. Ainsi, un système s'arrêtant de manière fixe à deux itérations obtiendrait de meilleurs résultats que nous, sans pour autant être intelligent, ou même satisfaire le besoin réel de l'utilisateur. Ainsi il pourrait être intéressant de mettre en œuvre des métriques permettant à la fois de combiner la notion de *CubeTest* avec une notion de gain d'information sur le long terme pour pouvoir mieux cerner la qualité d'un critère d'arrêt, qui mériterait une métrique spécifique à cette problématique. C'est pour cette raison que nous avons introduit le *Positive CubeTest* et le *Cumulative CubeTest* qui permettent d'étudier le gain d'information total sur un nombre d'itérations choisi. Nous utiliserons ces métriques dans notre évaluation finale permettant une analyse plus fine des systèmes proposés.

Chapitre 8 - Expérimentations et résultats

Nos expérimentations ont été menées dans un premier temps lors de l'édition TREC 15, puis ont été poursuivies par la suite avec l'édition TREC 16. Nous ne présentons que les résultats sur le jeu de données TREC 16 par souci de cohérence, cependant nos résultats de TREC 15 ont également été publiés [7]. Lors de ces deux éditions, nous avons pu étudier des aspects spécifiques de la recherche d'information dynamique. Nous avons également effectué des expérimentations en dehors de ce cadre sur les plongements lexicaux appliqués à la recherche d'information dynamique, car ce sujet est très peu représenté dans la littérature récente. Nous proposons dans ce chapitre les conclusions des différentes expérimentations réalisées, ainsi qu'une expérimentation de synthèse présentant les grandes approches combinées en un petit nombre de pipelines finaux.

8.1 Résultats expérimentaux

8.1.1 Analyse du jeu de données TREC 16

Afin de comprendre un peu mieux le contexte dans lequel nous allons effectuer nos recherches, nous proposons ici une analyse du jeu de données fourni par la compétition TREC 16. Ce jeu de données a été élaboré par des assesseurs qui ont annoté les passages qu'ils estimaient pertinents en utilisant des moteurs de recherche afin de retrouver les documents. Dans un but d'impartialité, ils ont utilisé les résultats de Lemur, Solr et Terrier pour diversifier les sources et ne pas se coller à un moteur de recherche ce qui aurait induit un biais.

Le jeu de données TREC 16 contient au total 53 requêtes initiales différentes comme :

- *« US Military Crisis Response »*
- *« Economic Impact Ebola »*
- *« Healthcare Impact »*

Il est constitué d'un corpus d'environ 400 000 documents, répartis entre les domaines Ebola (à propos des données de la crise) et Polar (données scientifiques sur les problématiques des pôles Nord et Sud).

Pour réaliser notre analyse, nous allons utiliser le moteur de recherche Lucene par son interface Solr pour rechercher les 125 premiers documents pour chaque requête et par la suite analyser combien de documents pertinents sont inclus dans chaque « groupe ». Cela permet d'avoir une idée de la complexité de la tâche et de savoir si une majorité de documents est facile à trouver ou bien si la tâche se révèle très complexe. L'intérêt de cette analyse est de pouvoir mieux discuter de nos résultats dans les chapitres suivants. Car un jeu de données « fabriqué » présente des difficultés plus complexes pour l'analyse et les biais à plusieurs niveaux peuvent être importants et jouer sur l'impact des algorithmes que nous mettons en œuvre. Une telle analyse peut également offrir des pistes pour améliorer à l'avenir la création de tels corpus afin de coller au plus près de la réalité de la tâche que l'on veut accomplir : une recherche d'information dynamique en milieu complexe.

Nombre de documents pertinents parmi les 125 premiers résultats	<10	<20	<30	<40	<50	50 et plus
Nombre de sujets parmi les 53 sujets du jeu de données TREC 16	27	5	5	4	3	9

Tableau 8-1. Étude du jeu de données

On peut constater (Tableau 8-1) qu'une grande majorité des requêtes ont moins de 10 documents pertinents parmi les premiers 125 documents retournés par notre moteur de recherche, cela montre la difficulté de la tâche qui peut parfois ressembler à retrouver une exception parmi un flot de documents non évalués ou non pertinents. Au contraire, un certain nombre de requêtes ont plus d'un tiers des documents qui sont notés pertinents et seront des requêtes relativement faciles pour le système, car capable à la fois de fournir de la pertinence dès les premiers tours, et en plus de donner de l'information pour l'exploitation du retour de l'utilisateur. La grande disparité en fonction des sujets implique également des conséquences sur l'évaluation des systèmes qui rend assez difficile la lecture d'une comparaison entre deux systèmes. En effet, le bénéfice entre exploitation et exploration peut être compliqué à établir avec la grande polarité qu'on observe entre sujets très difficiles et sujets très simples. Une meilleure exploration ne sera pas forcément récompensée, car les sujets « moyennement difficiles » sont très peu représentés et un système effectuant beaucoup d'exploration pour espérer capter les quelques documents ne sera pas forcément récompensé par une métrique d'évaluation dégressive qui pénalise fortement le temps passé sans trouver de document satisfaisant.

Cette analyse doit donc permettre pour les analyses à venir de mieux appréhender ce qui peut être à l'origine de la disparité des résultats en fonction des méthodes et mieux comprendre l'impact que peuvent avoir certains choix en termes d'exploration et d'exploitation.

8.1.2 Impact du prétraitement

Nous avons parlé à la section 4.1 du prétraitement consistant à supprimer les éléments perturbateurs des pages internet, nous avons ainsi effectué des expérimentations dans le but d'établir l'impact sur le corpus Ebola d'un tel prétraitement en effectuant une recherche dynamique sur deux itérations avec une recherche par modèle de langage.

	CubeTest @ 2	AVG Cubetest @ 2
BoilerPipe	0.181	0.195
Full Page	0.147	0.151

Tableau 8-2. Impact du prétraitement sur le corpus Ebola

Comme nous pouvons le voir dans le Tableau 8-2, l'utilisation d'un prétraitement permettant de nettoyer le contenu de la page HTML a un impact significatif sur un corpus tel que celui du sujet *« Ebola »,* qui comporte uniquement des pages internet. L'impact absolu est de 0.034 et de 0.044 respectivement pour le *CubeTest* (CT) et *l'AVG CubeTest* (ACT). Cependant, il est à noter que la meilleure performance avec la métrique de *CubeTest* – obtenue sur l'ensemble du jeu de donnée TREC 15, avec un système utilisant uniquement des documents pertinents (donc un biais très important, qui avait pour but de voir vers quel score pouvait tendre un système idéal n'ayant pas un flot de documents non pertinents à gérer) – était de 0.3. Si l'on estime donc que notre maximum est de 0.3, une progression de 0.147 à 0.181 peut être vue comme un impact significatif.

8.1.3 Processus de recherche d'information initial

Le processus de recherche d'information consiste essentiellement en un paramétrage du logiciel de recherche, en l'occurrence Solr.

Pour ce faire, différentes possibilités s'offrent à nous : notamment le choix de la méthode de similarité et le choix des paramètres propres à chaque similarité. Dans le cadre de notre participation à TREC 15, nous avions travaillé uniquement avec la similarité par défaut offerte dans Solr, qui est une mesure de similarité utilisant TF*IDF avec une mesure de similarité par cosinus sans normalisation. Les travaux de certains participants à TREC 15 ont suggéré que d'autres techniques, telles que BM25 et la modélisation de langage (« *Language Model* ») pouvaient offrir des résultats supérieurs pour les premières itérations de la deuxième phase de recherche de documents. Nous avons donc procédé à la comparaison expérimentale de ces trois mesures, comparaison présentée au Tableau 8-3.

	CubeTest@2	AVG CubeTest@2
TF*IDF	0.039	0.038
BM25	0.100	0.097
LM	0.156	0.164

Tableau 8-3. Comparaison des techniques de recherche d'information

Ces comparaisons ont été effectuées avec les paramètres par défaut pour chaque mesure à partir du moteur de recherche Solr. Cependant, les résultats sont suffisamment significativement différents pour ne pas envisager de faire une étude poussée du paramétrage de TF*IDF et BM25 en comparaison de la mesure « *Language Model Dirichlet* » qui utilise la modélisation de langage

associée à un prior permettant le lissage du modèle. D'après [22] il est conseillé d'utiliser un prior μ d'une valeur de 2000 pour la plupart des collections, en sachant que des courtes requêtes auront de meilleurs résultats avec un prior faible qui va avoir pour effet de peu lisser le *« Language model »* et va essayer de coller au plus près de la requête. Au contraire de requêtes plus longues, qui bénéficieront d'un prior plus large, car elles contiennent plus de mots, mais n'apparaissent pas dans 100% des cas.

Nous avons comparé expérimentalement l'évolution du système en fonction des valeurs de ce prior.

Prior μ	CubeTest@2	AVG CubeTest@2
750	0.151	0.162
850	0.155	0.164
1250	0.156	0.164
1500	0.151	0.163
2000	0.152	0.161
3000	0.143	0.151

Tableau 8-4. Impact du prior sur les résultats

Nous pouvons constater (Tableau 8-4) que nous obtenons le meilleur résultat autour d'un prior de 1250. Notre collection étant faite d'articles à propos d'un sujet récent et sur des bulletins de nouvelles, les articles sont en général plutôt courts, ce qui explique le besoin d'un prior plus faible que 2000 qui est considéré comme la valeur par défaut pour un contenu textuel.

8.1.4 Découpage en phrase et impact sur la modélisation de sujets

Après notre analyse d'un découpage en phrases d'un article (voir section 4.4) nous avons donc imaginé tester expérimentalement l'impact d'une segmentation des articles et ne retenir qu'une sélection de phrases contenant une partie de la requête, dans le but d'effectuer la modélisation de concept avec l'allocation de Dirichlet latente (LDA). Le Tableau 8-5 présente cette expérimentation.

Algorithme	CubeTest@2	AVG CubeTest@2
LDA sur l'ensemble du document	0.141	0.110
LDA sur des phrases avec une partie de la requête	0.162	0.117

Tableau 8-5. Impact de la segmentation en phrases

Le résultat obtenu est que les scores sur LDA sont bien supérieurs en effectuant ce prétraitement sur les documents sélectionnés au cours de la recherche. Le bruit éliminé permet de construire des concepts plus proches de notre sujet de recherche.

8.1.5 Comparaison des algorithmes de diversification

Le changement de mesure de similarité ayant un impact significatif, nous avons testé comment se comportent nos algorithmes utilisés lors de TREC 15 sur ce nouveau corpus avec cette nouvelle mesure (voir Tableau 8-6).

Algorithme	CubeTest@2	AVG CubeTest@2
LM (Référence)	0.156	0.164
LDA	0.162	0.117
K-Moyennes	0.143	0.157
Algorithme de combinaison de LDA, K-Moyennes et LM	0.159	0.153

Tableau 8-6. Comparaison des algorithmes de recherche d'information

Le premier constat est que le système de base, après deux itérations (donc la prise en compte d'un seul retour de l'utilisateur sur cinq documents), est meilleur que le système utilisant les algorithmes LDA, *K*-Moyennes ainsi que la combinaison des trois.

Cependant, ce résultat est établi en comparant l'utilisation du même algorithme à chaque itération. Les meilleurs documents retournés par Solr, même s'ils ne sont pas diversifiés, ont plus de chance d'être pertinents et donc de permettre l'obtention d'un résultat. On peut donc comprendre que la principale limitation ici est le fait qu'il existe finalement très peu de place à la diversification sur seulement cinq documents par itérations.

On a vu dans cette section que pour améliorer les résultats de notre algorithme de modélisation de sujet, le travail autour des entités nommées était efficace, cependant, le seul usage du *« Langage model »* reste supérieur à l'utilisation de la modélisation de sujets. Ceci peut s'expliquer par le fait que durant les premières itérations la diversification n'apporte pas forcément de l'information, les meilleurs documents étant dans les premiers résultats. De fait, on sait que la mesure de *CubeTest* privilégie notamment les systèmes qui trouvent rapidement des documents pertinents. Ce qui peut fortement influencer les résultats et pénaliser fortement les documents qui cherchent à diversifier les résultats trop tôt.

Il pourrait donc être envisagé de traiter le problème en deux étapes distinctes, exploration et exploitation, fin d'utiliser les algorithmes de diversification et de modélisation au bon moment du processus afin de maximiser leur intérêt. Cependant comme nous l'avons noté dans notre analyse du jeu de données, cette maximisation des intérêts peut être compliquée à analyser étant donné la polarité dans la complexité des requêtes.

8.1.6 Impact du traitement du retour de l'utilisateur

La diversification est importante. Cependant, on peut également se demander quel impact ont certains paramètres de notre procédé du traitement du retour de l'utilisateur. Ainsi nous avons fait varier le nombre de mots à ajouter dans notre reformulation de requête sur l'algorithme Rocchio avec 4-8-10-12 mots. L'analyse de ces variations est présentée au Tableau 8-7.

Algorithme	CubeTest@2	AVG CubeTest@2
LM + Rocchio 4 mots	0.151	0.164
LM + Rocchio 8 mots	0.151	0.166
LM + Rocchio 10 mots	0.154	0.167
LM + Rocchio 12 mots	0.152	0.165

Tableau 8-7. Analyse du nombre de mots dans l'expansion de requête

On peut noter ici que le système tire bénéfice d'une expansion contenant autour d'une dizaine de mots, ceci peut s'expliquer par le fait que les mots les plus importants pour l'algorithme sont parfois

des mots assez généraux par rapport à la recherche, et ne permettent pas forcément de trouver des mots permettant de cliver suffisamment les documents généraux et spécifiques.

8.1.7 Récapitulatif des résultats intermédiaires pour le choix des pipelines finaux

Ces expérimentations résument les approches que l'on pourrait qualifier de classiques en opposition aux méthodes utilisant les « *embeddings* ». Nous constatons que pour la prise en compte du retour de l'utilisateur, l'utilisation de l'algorithme Rocchio avec une limite de 10 mots semble donner les meilleurs résultats. Nous considérons également les méthodes de recherche et diversification, la modélisation du langage est largement supérieure aux autres, nous utiliserons un prior fixé à 1250 étant donné qu'il semble offrir les meilleures performances dans notre cas. Les algorithmes de diversifications n'étant pas meilleurs, nous souhaitons tout de même comparer sur le pipeline final l'algorithme des K-Moyennes qui semble le plus proche de l'état de l'art si l'on considère les deux mesures CT et ACT, nous pourrons ainsi observer son comportement sur une mesure telle que le CCT et sur un nombre d'itérations plus important.

8.2 Résultats expérimentaux sur les plongements lexicaux ou *« word embeddings ».*

8.2.1 Description des expérimentations

L'une des préoccupations majeures que l'on a dans le cas de la recherche dynamique sur un domaine en particulier est d'être capable de s'adapter à n'importe quel domaine. Il n'est donc pas

possible d'utiliser des listes de mots externes, ontologies ou encore thésaurus, qui couvrent souvent trop de domaines divers et qui peuvent nous amener à de mauvaises interprétations.

Prenons par exemple le cas d'un des domaines présentés lors de TREC 15, « *illicit goods* » dont l'un des sujets concernait des comptes Facebook volés. Si l'on utilise une ressource externe pour avoir des informations concernant le sujet « Facebook », nous aurons très peu de chance de trouver des mots relatifs aux comptes volés, aux informations concernant les prix à l'achat de tels comptes ou encore aux moyens de s'en procurer.

De même, dans le cas du sujet du virus Ebola, ce qui nous intéresse est un épisode particulier de l'histoire du virus à un moment précis ; mais une ressource externe pourrait nous donner des informations relatives à des crises précédentes qui ne sont pas pertinentes dans notre cas. La seule information dont nous disposons réellement est l'accès à notre corpus et nous avons donc voulu créer la connaissance à partir de celui-ci. C'est ainsi que nous avons cherché à modéliser des sujets à partir du corpus pour chaque requête avec l'algorithme LDA. Cependant, et bien que cette technique permette de trouver un certain nombre de concepts, les résultats varient fortement et sont relativement sensibles au bruit. Nous avons donc cherché à savoir si une autre modélisation était possible. Pour cela, nous avons envisagé, face à la grande quantité de documents, d'appliquer des représentations de plongement lexical ou *« word embedding »* à notre corpus.

8.2.2 Résultats

Méthode	CubeTest@2	ACT@2
Expansion Word2Vec entraîné sur le corpus TREC Ebola	0.166	0.165
Expansion Word2Vec pré-entraîné sur Google News	0.179	0.188
Distance entre le titre et la requête	0.088	0.095
Distance entre les 200 premiers documents et la requête	0.125	0.138
Distance entre les 500 premiers documents et la requête	0.130	0.136
Language Model (référence)	0.187	0.197

Tableau 8-8. Comparaison des méthodes utilisant les « word embeddings »

Nous comparons au Tableau 8-8 les deux grandes idées exprimées précédemment avec différents paramètres. Il faut tout d'abord noter que ces tests ont été effectués uniquement sur le jeu de données « Ebola » pour lequel les résultats sont en moyenne meilleurs, ce qui explique que notre score de référence avec le *« Language Model »* soit supérieur aux résultats précédents.

Notre première idée était d'effectuer de l'expansion de requête, pour ce faire nous avons essayé deux configurations. L'une où nous avons utilisé un modèle *Word2Vec* entraîné sur les données du corpus (*W2V corps*) et l'autre qui utilise le modèle de Google News (*W2V gnews*). Nous utilisons les mots de notre requête et cherchons le mot le plus similaire que nous ajoutons à notre requête. Dans ce cas, nous observons que l'expansion de requête basée sur le modèle de Google News est meilleure que notre modèle utilisé sur nos données. Plusieurs raisons peuvent expliquer cela. Un certain nombre de mots peuvent être surreprésentés dans notre échantillon et mener à des

mauvais synonymes (comme on l'a vu précédemment avec *« Obama » / « Obola »*). Et certains mots peuvent être sous-représentés dans nos données et n'avoir pas assez d'apparitions pour que nous les gardions dans notre modèle, ou bien que cette sous-représentation fasse apparaître de faux synonymes.

Notre seconde idée était de générer une distance par cosinus entre les vecteurs et les phrases d'un document. Une fois cette distance calculée, nous prenons les scores de similarité des 10 meilleures phrases de chaque document et les additionnons pour obtenir un score de similarité par rapport à notre requête, et ce afin d'avoir un nombre minimum de phrases relatives à notre sujet. Cela permet d'éviter les articles qui mentionneraient la requête sans en parler de manière détaillée. Nous avons également proposé de tester cette distance avec le titre uniquement. Les calculs étant importants de même que la nécessité de découper en phrase chaque document, nous proposons de sélectionner uniquement les 200 et 500 premiers résultats et de les reclasser avec notre méthode. Nous pouvons noter que les résultats ne se dégradent pas en ajoutant des résultats. On peut donc en conclure que la méthode bien qu'étant inférieure au *« Language model »* n'est pas directement corrélée au fait d'avoir les résultats issus du moteur de recherche, mais est capable d'apporter des résultats semblables à la mesure BM25 qui est très utilisée dans les moteurs de recherche actuels.

Enfin, la distance avec le titre uniquement offre les pires résultats, ceci peut s'expliquer par le fait que le titre n'est pas forcément garant d'une qualité de contenu et qu'un titre peut correspondre aux attentes, mais l'article ne pas être pertinent, ou tout du moins ne pas contenir suffisamment d'information sur le sujet pour être retenu comme pertinent.

8.2.3 Comparaison de différents paramétrages et prétraitements pour les plongements lexicaux de Word2Vec

L'un des problèmes que l'on peut observer lorsqu'on essaye de modéliser un corpus par *Word2Vec* est le fait que certains mots vont être présents seuls alors qu'ils apparaissent toujours en

bigramme (par ex. *« New York »*). Il peut donc être intéressant notamment d'intégrer un algorithme de détection de bigrammes et trigrammes.

Ainsi, peut-être pourra-t-on faire émerger des synonymes aux bigrammes/trigrammes. Dans notre cas, il est intéressant de faire ce genre de traitement notamment pour l'expansion de requête. Prenons un exemple de requête initiale : *« Alleged Alternative Treatments for Ebola »*.

Après détection des *N*-grammes, nous obtenons la requête suivante : *« Alleged Alternative_Treatments for Ebola ».*

Après expansion, on trouvera un nouveau mot pour chaque « *token »* (jeton) ainsi : *« Allege Effective_Natural_Cure without_proper disease »*

On peut voir (Tableau 8-9) notamment que sans détection de *N*-gramme nous n'aurions jamais trouvé d'expression comme *« Effective Natural Cure »*.

Méthode	CubeTest@2	ACT@2
Expansion avec N-grammes Dimension 300	0.1519	0.1570
Expansion sans N-grammes dimension 300	0.1441	0.1470

Tableau 8-9. Comparaison utilisation de détection de N-grammes

Nous avons également comparé la dimension des vecteurs (cette comparaison est présentée au Tableau 8-10), pour voir si elle pouvait impacter notre usage.

Méthode	CubeTest@2	ACT@2
Expansion avec N-grammes Dimension 300	0.1519	0.1570
Expansion avec N-grammes dimension 50	0.1512	0.1574

Tableau 8-10. Analyse dimension vecteurs

Les écarts sont trop faibles pour être considérés dans ce cas. Ces modèles ont été entraînés sur les corpus *« polar »* et *« Ebola »* indépendamment. Pour chaque jeu de données, nous appliquons un autre modèle.

8.2.4 Conclusion des travaux effectués autour des vecteurs de mots

En conclusion, nous avons apporté un regard nouveau sur la manière dont on pourrait traiter la problématique de recherche d'information à partir de la modélisation du corpus avec un modèle de type *Word2Vec*. Cependant, nos résultats actuels ne sont pas en mesure de concurrencer la méthode de recherche *« Language Model »,* mais les résultats sont proches de la mesure BM25. De plus, il est envisageable d'atteindre de meilleurs résultats en optimisant le modèle *Word2vec* en essayant différentes configurations pour atteindre des vecteurs peut-être plus fidèles. En outre nous avons utilisé les vecteurs de Google News qui ont montré que des modèles basés sur plus de données pouvaient offrir de meilleurs résultats. Une des pistes pourrait être d'élargir la collection en allant chercher des articles similaires sur internet afin d'obtenir plus d'exemples pour l'apprentissage de notre modèle. Enfin, différentes manières d'aborder la problématique de comparaison de la requête à un document pourraient être envisagées. Dans notre cas, nous avons choisi les titres et le découpage en phrases, il serait également envisageable d'établir un compte par mot et de comparer le nombre d'itérations des

mots les plus proches pour obtenir un score de similarité non pas au niveau du passage, mais au niveau du document entier. Il serait également envisageable de travailler sur un modèle *Word2vec* déjà existant (de type Google News) et qui soit « augmenté » par les données d'un corpus particulier.

8.3 Résultats des pipelines finaux

8.3.1 Choix des systèmes comparatifs

Nos analyses des travaux précédents nous ont permis de dégager les tendances principales pour choisir les éléments des pipelines finaux que nous allons comparer vis-à-vis des meilleurs systèmes soumis à TREC 16 (voir Tableau 8-11).

Soumissions à TREC 16	ACT @ 2	CT @ 2	ACT @ 5	CT @ 5	ACT @ 10	CT @ 10
Rmit_oracle.lm.1000[6]	0.2789	0.2543	0.2330	0.1751	0.2065	0.1489
Rmit_lm_rocchio.Rp.NRd.10	**0.1644**	0.1439	0.1160	0.0636	0.0866	0.0381
Rmit_lm_nqe	0.1614	0.1539	0.1196	0.0758	0.0860	0.0426
ufmgXM2	0.1612	0.1574	0.1305	0.0927	0.1237	0.0852
ufmgHS2	0.1611	**0.1581**	0.1316	0.0910	0.1215	0.0801
ufmgHM3	0.1601	0.1578	0.1317	0.0967	0.1181	0.0808
ufmgHM2	0.1601	0.1578	0.1318	0.0957	0.1219	0.0842
ufmgXS2	0.1587	0.1506	0.1289	0.0898	0.1188	0.0786
FirstIterBaseline[7]	0.1516	0.2174	0.1516	0.2174	0.1516	0.2174
LDA_Indri73	0.1425	0.1242	0.1045	0.0645	0.0985	0.0583
TenthIterBaseline	0.1352	0.1274	0.0944	0.0518	0.0639	0.0259
SecondIterationBaseline	0.1352	0.1274	0.1352	0.1274	0.1352	0.1274
FifthIterBaseline	0.1352	0.1274	0.0944	0.0518	0.0944	0.0518
Rmit_lm_psg.max	0.1178	0.1275	0.0928	0.0688	0.0708	0.0404
UPD_IA_BiQBFi	0.1107	0.1281	0.0925	0.0760	0.0925	0.0760
UPD_IA_BiQBDiJ	0.1107	0.1281	0.0938	0.0774	0.0938	0.0774

Tableau 8-11. TREC 16

[6] La soumission est juste présente à titre de comparaison, mais étant un système qui vérifie la pertinence des documents avant de les soumettre il ne peut pas être comparé.

[7] Nous ne considérons pas le score de cette soumission car s'arrêtant à la première itération, le score étant dégressif, ce n'est pas comparable avec les autres systèmes.

La liste des soumissions étant exhaustive, nous proposons de prendre les plus performants en points de comparaison : *Rmit_oracle.lm.1000* est un système soumis par l'équipe de l'université australienne du RMIT, leurs résultats sont supérieurs aux autres par le fait qu'ils utilisent les données cachées de la pertinence pour savoir s'ils soumettent un document ou non. Leur approche est donc similaire à la *« Judged-only task »* de TREC 15. Ces résultats ne sont pas directement comparables aux autres soumissions, car c'est une forme de « triche », cependant il est intéressant de noter que c'est un score qu'on pourrait qualifier d'idéal, car bien qu'il puisse proposer des résultats peu ou moyennement pertinents, il retourne tout de même des documents pertinents à 100%.

Le second système que nous étudierons est *Rmit_lm_rocchio.Rp.NRd.10*. Il utilise notamment le *« Language Model »* et Rocchio pour exploiter le retour de l'utilisateur, c'est le meilleur système en termes *d'Average CubeTest* (ACT) après deux itérations.

Le dernier système que nous allons étudier est *ufmgHS2* (que nous avons présenté dans la revue de littérature), car il est le meilleur système en termes de *CubeTest* après deux itérations.

8.3.2 Présentation des pipelines

Nous présentons trois pipelines que nous évaluons dans nos expérimentations.

Le premier utilise :

- Le *« Language Model »* pour retrouver les documents ;
- Rocchio avec une limite de dix mots pour l'expansion et les entités nommées du passage surligné par l'utilisateur pour exploiter au mieux son retour ;
- Le critère d'arrêt utilisant notre mesure de température simulant l'intérêt de l'utilisateur.

Le second utilise :

- *K*-Moyennes pour sélectionner cinq documents parmi les 25 meilleurs retrouvés par le *« Language Model »*
- Le critère d'arrêt utilisant la température.

Le troisième et dernier utilise :

- Le *« Language Model »* pour retrouver les documents ;
- Expansion de requête avec les plongements lexicaux, le modèle word2vec est entraîné uniquement à partir des corpus Ebola et Polar, avec une dimension de 300.
- Le critère d'arrêt utilisant la température.

Système	ACT @ 2	CT @ 2	ACT @ 5	CT @ 5	CCT @ 2	CCT @ 5
Rmit_oracle.lm.1000[8]	0.2789	0.2543	0.2330	0.1751	Non évalué	Non évalué
Rmit_lm_rocchio.Rp.NRd.10	0.1644	0.1439	0.1160	0.0636	Non évalué	Non évalué
ufmgHS2	0.1611	**0.1581**	**0.1316**	**0.0910**	Non évalué	Non évalué
LM + Rocchio 10w + NE + Critère d'arrêt	**0.1669**	0.1541	0.1184	0.06918	0.3082	0.3430
K-Moyennes + Critère d'arrêt	0.1431	0.1572	0.1134	0.0773	**0.3143**	**0.3718**
Expansion Word2Vec dimension 300	0.1570	0.1519	0.11808	0.07516	0.3039	0.3314

Tableau 8-12. Résultats pipelines finaux

Les résultats présentés dans le Tableau 8-12 présentent nos trois principaux pipelines qui ont émergé des précédentes expérimentations. Nous observons que logiquement, notre système utilisant Rocchio et l'approche bayésienne (LM) offre des résultats similaires à une configuration similaire du système du RMIT. Cependant, nous avons notamment rajouté les entités nommées et une optimisation du prior de Dirichlet. Ces éléments ont permis d'obtenir un score supérieur à l'état de l'art pour la

[8] La soumission est juste présente à titre de comparaison, mais étant un système qui vérifie la pertinence des documents avant de les soumettre il ne peut pas être comparé en termes de performances.

mesure *Average CubeTest*. Néanmoins pour le *CubeTest* classique, nous n'obtenons pas le score le plus élevé, bien qu'en restant assez proche du système de [17].

Nous introduisons également les mesures présentées au Chapitre 7 pour cette évaluation finale. En effet, les résultats sont particulièrement étonnants pour *K*-Moyennes qui semble, bien qu'ayant des mesures globalement inférieures, accumuler plus d'information globalement que les autres techniques. Cette méthode est donc principalement pénalisée par l'absence de classement à l'issue du regroupement. En effet, nous proposons de prendre le meilleur document de chaque groupe en ne faisant pas de distinction entre ceux-ci.

Enfin, notre méthode concernant l'expansion de requête grâce à *Word2Vec*, bien que proche de l'état de l'art dans la plupart des cas, ne semble que handicaper la requête initiale.

Précédemment, nous avions discuté de la manière dont avait été réalisé le corpus, par des assesseurs utilisant des moteurs de recherche différents. Cependant, la méthode induite dans ce mémoire par l'utilisation de *Word2Vec* et les autres méthodes de plongement lexical sont par nature très différentes. Elles permettent de trouver des synonymes et donc des documents dont les assesseurs sont sans doute passés à côté durant l'évaluation. En effet bien que cette modélisation puisse être moins bonne que les méthodes existantes, nous avons pu essayer de voir comment se comporte le système en regardant de plus près certaines requêtes et il semble évident qu'un certain nombre de documents non jugés n'ont pas été comptabilisés alors qu'ils présentaient des liens avec la requête. On voit donc qu'il est très compliqué de créer un jeu de données représentatif de la réalité et que les résultats présentés doivent être nuancés par ce biais possible.

Chapitre 9 - Conclusion et travaux futurs

9.1 Enjeux et travaux réalisés

La recherche d'information dynamique est un domaine qui va trouver dans l'avenir de plus en plus d'intérêts. En effet, au regard du nombre croissant des données, le besoin en systèmes capables de trouver des documents parmi une collection contenant de nombreux documents non indexés devient de plus en plus important. L'état de l'art de la catégorie « Domaine dynamique » s'est fait tout au long de la rédaction de ce mémoire. Les deux compétitions auxquelles nous avons participé sont également les deux premières éditions dans cette catégorie. Nous avons ainsi contribué à l'établissement de cet état de l'art par nos soumissions aux conférences TREC 15 et TREC 16, au cours desquelles nos travaux ont été présentés.

La recherche d'information dynamique peut être considérée comme une tâche complexe. Ainsi, lors de la compétition TREC 16, une équipe a décidé de se comparer à un système qui aurait accès à l'information « interdite » : savoir si un document est pertinent. Le système ayant accès à cette information, bien qu'étant supérieur aux autres systèmes, présentait des mesures relativement faibles en termes de satisfaction des besoins de l'utilisateur. On peut en conclure que la recherche d'information dynamique est rendue complexe par plusieurs de ses composantes : être capable de fournir des documents hautement pertinents, être capable de fournir des documents diversifiés, être capable de prendre en compte le retour de l'utilisateur sur la base d'un faible volume de documents et enfin trouver un critère d'arrêt.

Ces quatre problèmes constituent eux-mêmes des problématiques de recherche active, leur combinaison est donc sujette à de nombreuses innovations. Il est également compliqué de faire varier certains paramètres tant l'impact sur l'ensemble du processus peut être important. Lors de conférences

internationales telles que TREC, les discussions s'orientent notamment autour de la difficulté de la reproductibilité des résultats dans le cadre de systèmes entiers. Cependant, au cours de ces deux dernières années, le nombre de participants et le nombre de systèmes élaborés sur différentes bases (Lucene, Terrier, Indri) ont permis de recouper certains choix qui semblent être efficaces, quel que soit le système.

En outre, l'utilisation de nouvelles caractéristiques telles que les *« word embeddings »* pourrait trouver une utilité dans la recherche d'information. Nous avons essayé dans ce mémoire d'apporter des éléments de nouveauté de ce point de vue. Les résultats, bien que n'étant pas meilleurs que l'état de l'art, méritent des travaux plus approfondis permettant peut-être de faire émerger avec certaines configurations un système meilleur que l'état de l'art actuel.

9.2 Nos contributions

Nous avons apporté notre contribution concernant les méthodes d'évaluation d'un système dynamique, via notre proposition de prendre en compte le pur gain en information sur une courte session, plutôt que d'appliquer une pénalité de temps trop agressive qui nuit à la lisibilité de la progression des systèmes. Ce choix d'une métrique d'évaluation mène logiquement à notre contribution pour un critère d'arrêt pour les systèmes dynamiques. Nous avons proposé une méthode simple appliquant une température qui va être stimulée en fonction de la découverte de nouveaux sujets qui pourraient intéresser l'utilisateur et donc le rendre attentif à de nouveaux documents qui pourraient enrichir ce sujet.

Nous avons également proposé un certain nombre de techniques utilisant les plongements lexicaux, qui n'avaient pas été appliqués à la recherche dynamique auparavant avec notamment l'utilisation dans l'expansion de requête ou dans la recherche de documents similaires. Les résultats sont inférieurs à l'état de l'art, mais constituent sans doute une approche à approfondir.

Enfin, nous avons proposé l'utilisation de techniques de regroupement et de modélisation par sujet, méthodes qui ont souvent des résultats proches de l'état de l'art. La technique des K-moyenne peut même être à l'état de l'art si on considère une métrique pénalisant moins le facteur de temps telle que la variante du *CubeTest* (*Cumulative CubeTest*) que nous proposons.

9.3 Suggestions de travaux futurs

Dans ce mémoire nous avons pu approfondir de nombreuses notions autour de la recherche dynamique, cela a également permis de pouvoir noter un certain nombre de problématiques aussi bien au niveau de la difficulté de la tâche, que des jeux des données ainsi que de l'évaluation des systèmes.

La tâche qui est définie comme ayant pour objectif de prendre en compte le retour de l'utilisateur offre des difficultés très particulières au problème. Il est à la fois très pauvre en données directes (retour de l'utilisateur rare et très précis) et demande à la fois des résultats très spécifiques. De plus, souvent le fait de trouver un sujet ne va nous donner de l'information que sur ce sujet et va donc compliquer la tâche de trouver des éléments permettant de diversifier les résultats. Les enjeux sont donc très importants, dans le but de trouver des techniques permettant de prendre en compte des données très spécifiques et de produire des résultats hautement spécifiques, mais en partant avec des résultats plus généraux. On pourrait donc notamment s'intéresser à l'apprentissage par transfert. Ceci implique cependant d'avoir un corpus référence, cependant pour utiliser ce type d'algorithme il faut des données de références abondantes, ce qui n'existe pas pour le moment en recherche d'information dynamique de manière ouverte.

En effet, le jeu de données complique de manière importante la tâche d'évaluation des systèmes. Notre analyse du jeu TREC 16 permet de se rendre compte de la grande polarité dans la difficulté entre les requêtes, alors que l'on s'intéresse finalement à un sujet qui permet de traiter le retour de l'utilisateur avec un jeu de données qui n'en contient que très peu. Il en résulte que les

méthodes de recherche d'information traditionnelle sont souvent très performantes, car elles permettent de trouver les quelques documents pertinents lors de la première itération, et les itérations suivantes ne servent qu'à trouver quelques documents. De plus, une forte pénalité de temps est mise en place par des évaluations officielles telles que le *CubeTest.* En conséquence, il faudrait pouvoir construire un jeu de données plus important en termes de quantité et surtout plus fidèle à l'esprit de la recherche d'information dynamique. La majorité des documents étant non jugés ne reflètent pas l'interaction qu'on pourrait avoir avec un utilisateur réel qui nous permettrait de réellement interpréter les documents ayant une note de 0 comme des éléments négatifs.

Enfin, les méthodes d'évaluations, bien que prenant en compte l'information des différents sujets, semblent contradictoires avec l'objectif de la recherche d'information. D'un côté, on a un utilisateur cherchant à obtenir de l'information, mais de l'autre nous observons une métrique qui pénalise fortement l'algorithme qui va continuer à chercher des sous-sujets non exploités. Nous proposons donc d'aller dans le sens de nos métriques proposées au Chapitre 7, en ayant une vision positive de l'algorithme qui va aller creuser l'information et trouver un sujet difficile. On pourrait ainsi notamment attribuer une difficulté aux sujets et offrir une récompense plus grande en termes de gain si un sujet difficile est couvert par les documents retrouvés.

La recherche dynamique est donc un sujet extrêmement vaste qui dispose de nombreux enjeux d'avenir. Le travail à effectuer pour améliorer ce type de système reste très important, nous espérons que nos contributions permettront d'aller plus loin dans ce domaine.

Références

[1] A. ALBAHEM, D. SPINA, L. CAVEDON and F. SCHOLER, *RMIT @ TREC 2016 Dynamic Domain Track: Exploiting Passage Representation for Retrieval and Relevance Feedback*, *TREC*, 2016.

[2] D. ANDRZEJEWSKI and D. BUTTLER, *Latent topic feedback for information retrieval*, *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2011, pp. 600-608.

[3] D. M. BLEI, A. Y. NG and M. I. JORDAN, *Latent Dirichlet Allocation*, Journal of machine Learning research, 3 (2003), pp. 993-1022.

[4] S. BÜTTCHER, C. L. CLARKE and G. V. CORMACK, *Information retrieval: Implementing and evaluating search engines*, Mit Press, 2016.

[5] O. CHAPELLE, D. METLZER, Y. ZHANG and P. GRINSPAN, *Expected reciprocal rank for graded relevance*, *Proceedings of the 18th ACM conference on Information and knowledge management*, ACM, 2009, pp. 621-630.

[6] R. DEVEAUD, E. SANJUAN and P. BELLOT, *Accurate and effective latent concept modeling for ad hoc information retrieval*, Document numérique, 17 (2014), pp. 61-84.

[7] R. JOGANAH, R. KHOURY and L. LAMONTAGNE, *Laval University and Lakehead University at TREC Dynamic Domain 2015: Combination of Techniques for Subtopics Coverage*, *Proceedings TREC 2015*, Gaithersburg, 2016.

[8] D. JURAFSKY and H. JAMES, *Speech and language processing an introduction to natural language processing, computational linguistics, and speech*, (2000).

[9] M. KUSNER, Y. SUN, N. KOLKIN and K. WEINBERGER, *From word embeddings to document distances*, *International Conference on Machine Learning*, 2015, pp. 957-966.

[10] Q. LE and T. MIKOLOV, *Distributed representations of sentences and documents*, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1188-1196.

[11] J. LUO, C. WING, H. YANG and M. HEARST, *The water filling model and the cube test: multi-dimensional evaluation for professional search*, *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, ACM, 2013, pp. 709-714.

[12] J. LUO and H. YANG, *Re-ranking via User Feedback: Georgetown University at TREC 2015 DD Track*, *Proceedings of TREC 2015*, Gaithersburg, 2016.

[13] C. D. MANNING, P. RAGHAVAN and H. SCHÜTZE, *Introduction to information retrieval*, Cambridge university press Cambridge, 2008.

[14] R. MIHALCEA and P. TARAU, *TextRank: Bringing Order into Text*, *EMNLP*, 2004, pp. 404-411.

[15] T. MIKOLOV, K. CHEN, G. CORRADO and J. DEAN, *Efficient estimation of word representations in vector space*, arXiv preprint arXiv:1301.3781 (2013).

[16] T. MIKOLOV, I. SUTSKEVER, K. CHEN, G. S. CORRADO and J. DEAN, *Distributed representations of words and phrases and their compositionality*, *Advances in neural information processing systems*, 2013, pp. 3111-3119.

[17] F. MORAES, R. L. T. SANTOS and N. ZIVIANI, *UFMG at the TREC 2016 Dynamic Domain track*, *TREC*, Gaithersburg, 2016.

[18] R. RAHIMI and G. H. YANG, *An Investigation of Basic Retrieval Models for the Dynamic Domain Task*, (2017).

[19] D. ROY, D. PAUL, M. MITRA and U. GARAIN, *Using word embeddings for automatic query expansion*, arXiv preprint arXiv:1606.07608 (2016).

[20] H. YANG, J. FRANK and I. SOBOROFF, *TREC 2015 Dynamic Domain Track Overview*, (2016).

[21] H. YANG, M. SLOAN and J. WANG, *Dynamic information retrieval modeling*, *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, ACM, 2014, pp. 1290-1290.

[22] C. ZHAI and J. LAFFERTY, *A study of smoothing methods for language models applied to information retrieval*, ACM Transactions on Information Systems (TOIS), 22 (2004), pp. 179-214.