



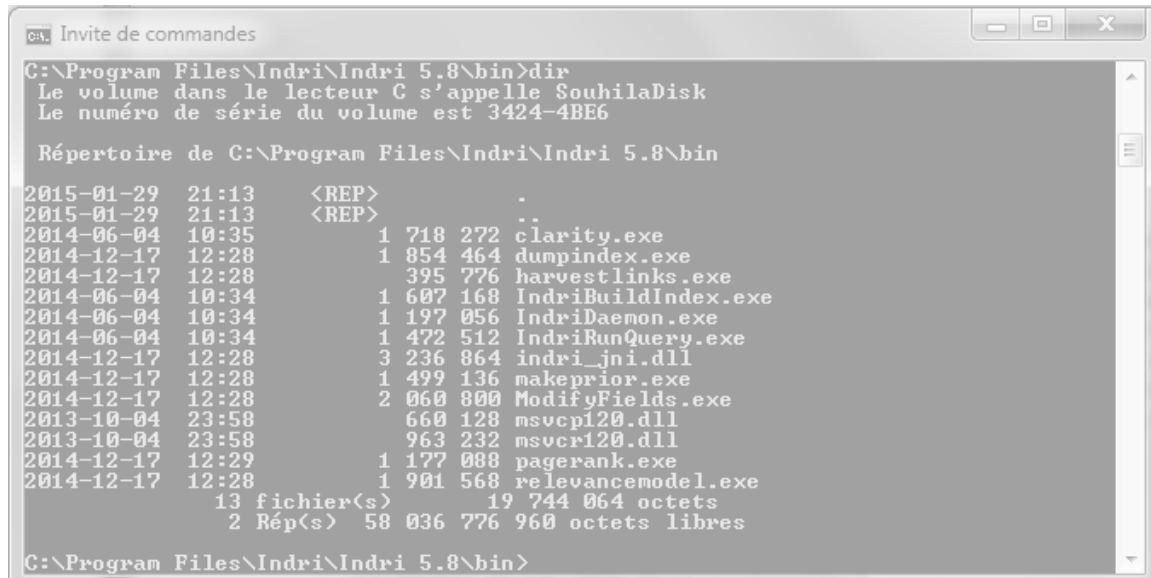
Mouna Mokaddem
Souhila Benbetka
IFT6255

DEVOIR 1

Hiver 2015

Présentation du problème

- **Installation** (Indri , TRECEVAL) : Elle est réalisée sur 2 systèmes ; Lunix et windows7
- **Liste des exécutables de Indri**



```
CA Invite de commandes
C:\Program Files\Indri\Indri 5.8\bin>dir
Le volume dans le lecteur C s'appelle SouhilaDisk
Le numéro de série du volume est 3424-4BE6

Répertoire de C:\Program Files\Indri\Indri 5.8\bin

2015-01-29 21:13      <REP>          .
2015-01-29 21:13      <REP>          ..
2014-06-04 10:35          1 718 272 clarity.exe
2014-12-17 12:28          1 854 464 dumpindex.exe
2014-12-17 12:28          395 776 harvestlinks.exe
2014-06-04 10:34          1 607 168 IndriBuildIndex.exe
2014-06-04 10:34          1 197 056 IndriDaemon.exe
2014-06-04 10:34          1 472 512 IndriRunQuery.exe
2014-12-17 12:28          3 236 864 indri_jni.dll
2014-12-17 12:28          1 499 136 makeprior.exe
2014-12-17 12:28          2 060 800 ModifyFields.exe
2013-10-04 23:58          660 128 msvcpr120.dll
2013-10-04 23:58          963 232 msucr120.dll
2014-12-17 12:29          1 177 088 pagerank.exe
2014-12-17 12:28          1 901 568 relevancemodel.exe
                13 fichier(s)          19 744 064 octets
                2 Rép(s)  58 036 776 960 octets libres

C:\Program Files\Indri\Indri 5.8\bin>
```

- **Les fichiers de travail :**
 - La collection TREC AP88-90
 - Les requêtes Topics (1-50), (51-100), (101-150)
 - Les fichiers d'évaluation (QRELAP8890)
 - Commande TREC-EVAL : [11]

Partie 1 : Méthodes et résultats

1.1 Indexation :

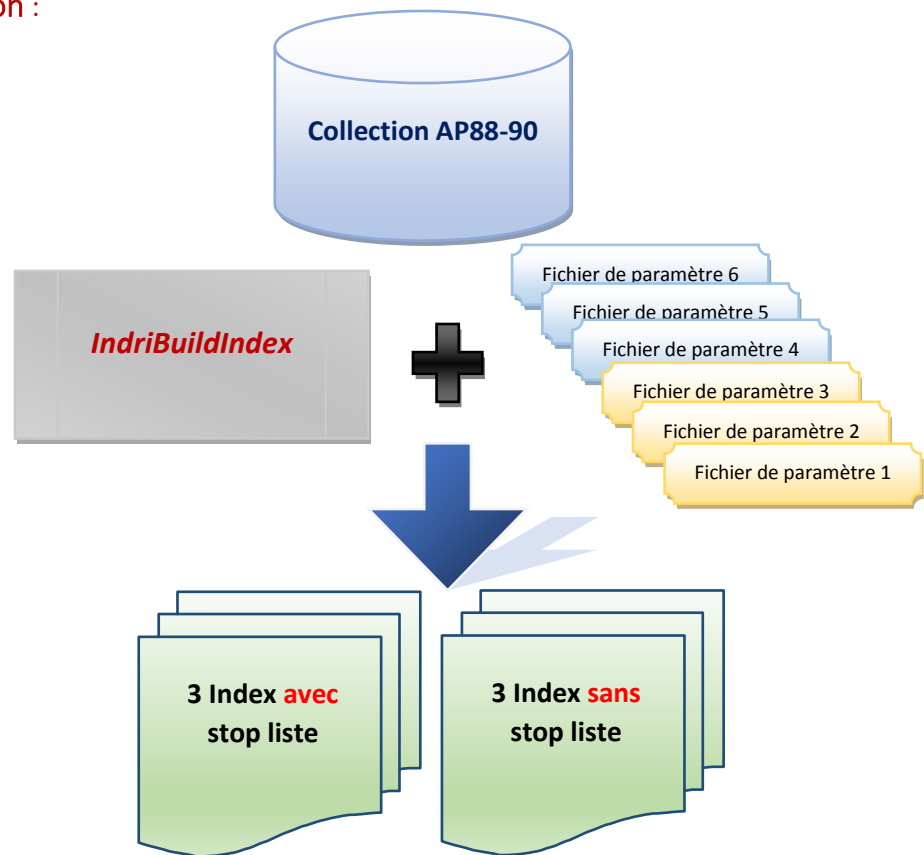


Figure 1: Procédure schématisant l'indexation sous Indri

L'indexation sous Indri est réalisée en exécutant la commande *IndriBuildIndex*. La procédure a pour but construire des index (inverted list).

La commande *IndriBuildIndex* utilise un fichier de paramètres où l'on doit indiquer la source de la collection AP8890, l'emplacement de la sortie, notre index, des options à faire varier. La figure 2 montre la structure d'un fichier paramètres d'indexation *IndriBuildIndex* *<parameter_file>* ,

```
<parameters>
<corpus>
<path>/home/mouna/Desktop/IFT_6255/collection_TREC/AP</path>
<class>trectext</class>
</corpus>
<index>/home/mouna/Desktop/IFT_6255/devoir_1/index_output_5</index>
<memory>512M</memory>
  <stopper>
    <word>a</word>
    <word>about</word>
    <word>above</word>
    <word>according</word>
  </stopper>
<stemmer>
<name>Porter</name>
</stemmer>
</parameters>
```

Figure 2 : Fichier paramètres de la procédure d'indexation

Résultats de l'indexation :

Tableau des résultats des 6 index de la procédure d'indexation:

	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
	Sans Stop liste			Avec Stop liste		
	Sans Stemming	Stemming Porters	Stemming Krovetz	Sans Stemming	Stemming Porters	Stemming Krovetz
Taille Mbit	1100	1000	1000	960.5	936.2	941.5

Les résultats nous montrent que l'utilisation de stoplist et de stemming Porter (index 5) donne un index de taille plus petit que les autres index. Nous avons aussi constaté que les tailles des index 5 et 6 sont proches, avec un léger avantage du stemming Porter par rapport à celui de Krovetz.

Ceci peut s'expliquer par le fait que l'algorithme de Porter, comme nous l'avons vu en cours, transforme les tokens en coupant leurs affixes : ce qui va essentiellement réduire la taille du fichier indexé alors que l'algorithme de Krovetz ne modifie pas les tokens, sauf dans le cas où il ne trouve pas le token en question dans le dictionnaire.

Conclusion sur l'indexation:

Nous pouvons conclure que l'utilisation des options, notamment le stemming et la stoplist, permet de réduire la taille des documents et surtout l'index avec stoplist et stemming Porter. Mais la question qui se pose est la suivante. Lorsqu'on exécute une requête pour la recherche, est-ce qu'on obtient de meilleurs résultats? Les documents résultants sont-ils les plus pertinents pour l'utilisateur, par rapport aux différents index générés ?

Dans la prochaine section, nous allons essayer de répondre à cette question.

1.2 Recherche :

La partie recherche est la deuxième partie qui permet l'accomplissement de la recherche de l'information. Elle a pour but de trouver les documents pertinents pour une requête donnée (dans notre cas les requêtes sont dans les Topics : 1-50, 51-100, 101-150).

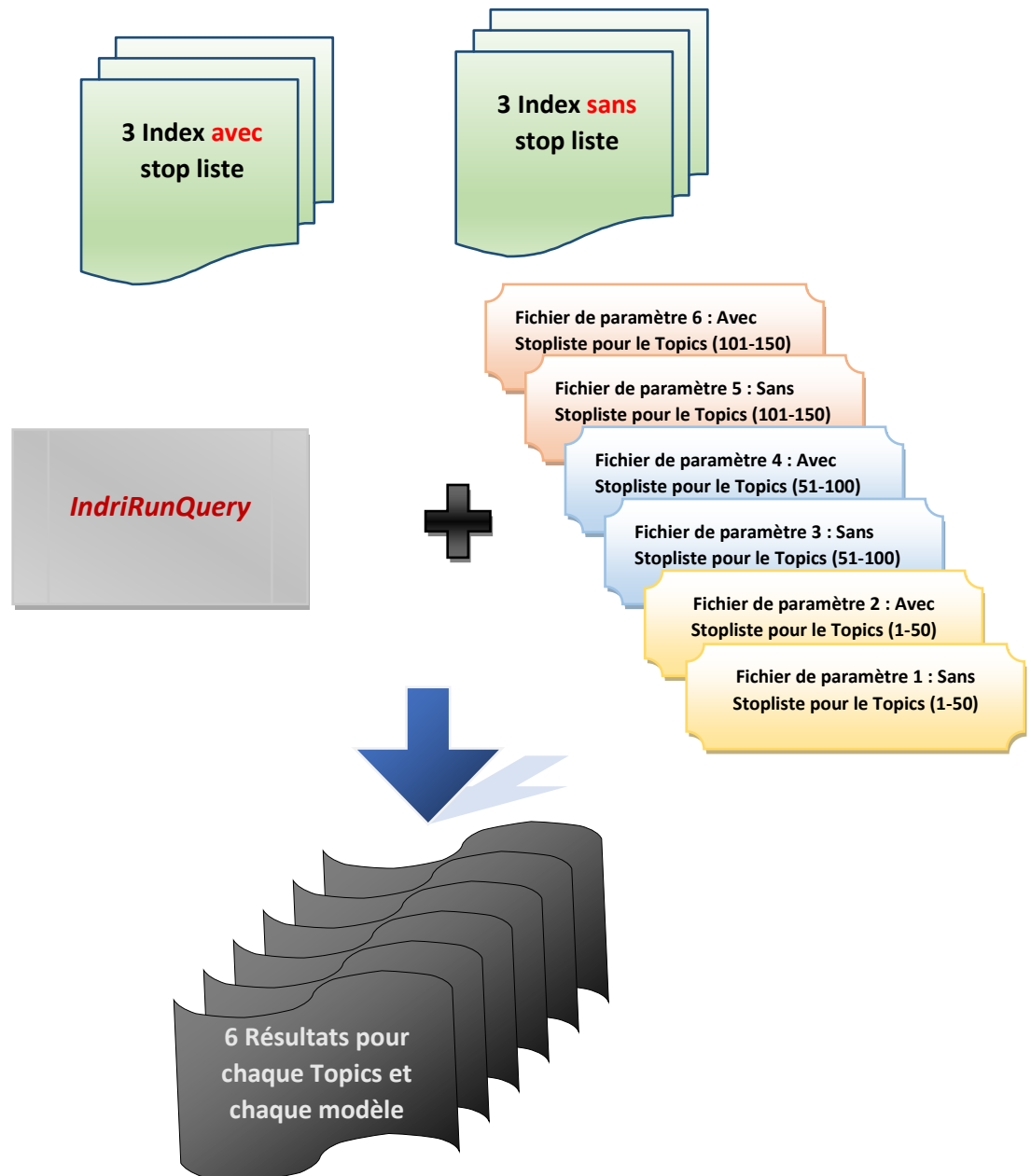


Figure 3: Procédure schématisant la recherche sous Indri

La commande *IndriRunQuery* nous permet d'effectuer la recherche sur les index construits lors de l'étape de l'indexation. La commande utilise aussi un fichier de paramètres. La figure 4 montre sa structure. Les requêtes se trouvent dans les fichiers Topics (figure 5). Nous avons utilisé en premier lieu la recherche sur le titre des 150 requêtes.

```
<parameters>
<index>/home/mouna/Desktop/IFT_6255/devo
ir_1/index_output_1</index>
<count>1000</count>
<query>
<number>1</number>
<text>Antitrust Cases Pending</text>
</query>
<rule>method:dirichlet,mu=1000</rule>
<!--<baseline>okapi</baseline> -->
<runID>runName</runID>
<trecFormat>true</trecFormat>
</parameters>
```

Figure 4: Exemple du fichier paramètres requête

```
<top>
<head> Tipster Topic Description
<num> Number: 007|
<dom> Domain: U.S. Economics
<title> Topic: U.S. Budget Deficit
<desc> Description:
Document will mention a proposal to decrease the U.S. budget deficit.
```

Figure 5: Exemple du fichier Topics requête

1.3 Procédure de filtrage des fichiers Topics

Avant de commencer cette étape, nous avons constaté que les fichiers des Topics contiennent des caractères spéciaux. Ceci nous a conduits à faire une procédure de filtrage afin de les enlever. (Cette procédure n'est pas venue d'emblé. On avait constaté lors de l'étape succédant l'étape de recherche que le nombre de requêtes retourné n'était pas le même. On a cherché d'où le problème venait et on a parcouru les exceptions. Puis nous l'avons solutionné par une procédure de filtrage).

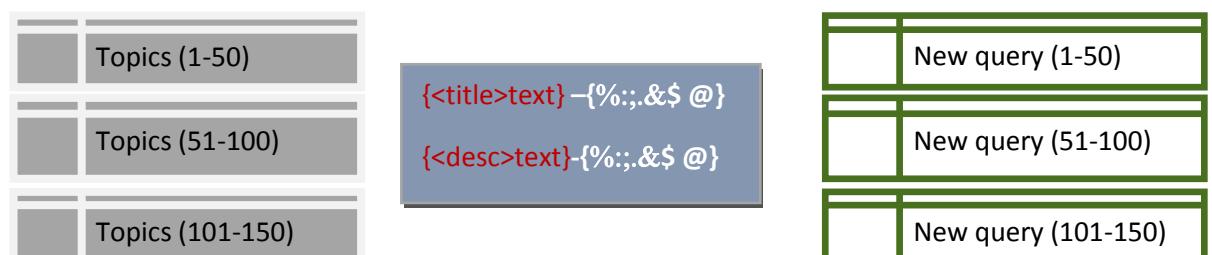


Figure 6: Procédure de filtrage des fichiers Topics

Pour chaque index obtenu on teste les 3 modèles de recherche à savoir : Tf*idf, BM25, Modèle de langue. Pour chaque index on parcourra les 3 ensembles de requêtes modifiées. De ce fait on obtiendra 6 résultats pour chaque modèle et chaque ensemble de Topics au total 18 résultats pour chaque modèle.

1.4 Évaluation :

Cette étape est la partie finale de la recherche de l'information. Elle consiste à comparer les résultats des expérimentations avec les modèles testés et les jugements standards TREC. La comparaison est faite par le biais de la commande **TREC-EVAL**. La figure 6 illustre la procédure.

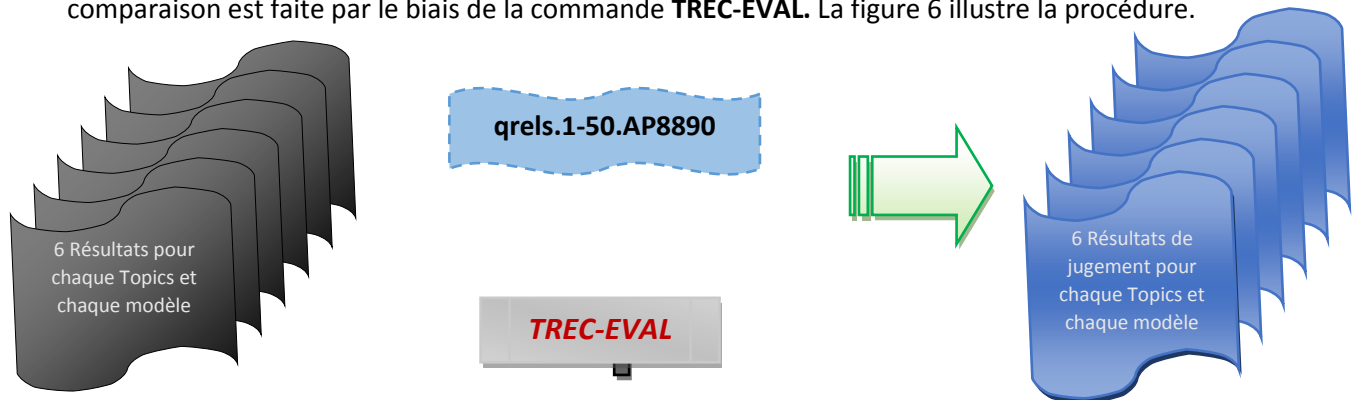


Figure 7: Procédure d'évaluation des résultats des différents modèles

Pour chaque index, chaque modèle et chaque Topics on récupère la valeur de la précision moyenne (**MAP**) ainsi que pour les 10 premiers documents (**P@10**). La précision au point x, P@x, est le ratio des documents pertinents parmi les x premiers documents trouvés. La figure 8 affiche un fichier résultat exemple de l'exécution de la commande :

trec_eval <fichier jugement> <fichier résultat> > <fichier contenant la comparaison>

num_q	all	50
num_ret	all	50000
num_rel	all	9873
num_rel_ret	all	4122
map	all	0.1626
gm_ap	all	0.0810
R-prec	all	0.2251
bpref	all	0.2556
recip_rank	all	0.4726
ircl_prn.0.00	all	0.5560
ircl_prn.0.10	all	0.3421
ircl_prn.0.20	all	0.2803
ircl_prn.0.30	all	0.2393
ircl_prn.0.40	all	0.1939
ircl_prn.0.50	all	0.1528
ircl_prn.0.60	all	0.1037
ircl_prn.0.70	all	0.0786
ircl_prn.0.80	all	0.0498
ircl_prn.0.90	all	0.0309
ircl_prn.1.00	all	0.0105

Figure 8: Fichier résultat de la comparaison

1.4.1 Résultats de l'évaluation du modèle tf*idf:

Tableau 2 : Modèle tf*idf : Topics 1-50

Les variables	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
MAP	0.1063	0.1313	0.1290	0.1082	0.1195	0.1171
P@10	0.1180	0.1380	0.1380	0.1360	0.1420	0.1420

Tableau 3 : Modèle tf*idf Topics 51-100

Les variables	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
MAP	0.1784	0.2193	0.2041	0.1785	0.2195	0.2034
P@10	0.3520	0.3820	0.3680	0.3540	0.3920	0.3680

Tableau 4 : Modèle tf*idf Topics 101-150

Les variables	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
MAP	0.1550	0.1622	0.1659	0.1547	0.1651	0.1655
P@10	0.3340	0.3080	0.3080	0.3320	0.3180	0.3060

Observations :

Nous remarquons que pour le tableau des requêtes 1-50 la meilleure valeur pour la MAP est de 0.1313 pour l'index 2 avec stemming (Porter) et sans stoplist et celle pour la P@10 est de 0.1420 pour l'index 5 et 6 les deux avec stoplist et stemming Porter et krovetz respectivement. Pour le tableau de requêtes 51-100, nous observons les meilleures valeurs pour le même index 5; avec 0.2195 pour MAP et 0.3920 pour P@10. Pour le tableau 101-150, nous remarquons une légère différence par rapport aux deux premiers tableaux avec les meilleures valeurs associées, à l'index 3 pour MAP avec 0.1659 qui est sans stoplist et avec stemming Krovetz et à l'index 1 pour P@10 avec 0.3340 qui est sans stoplist et sans stemming Porter.

1.4.2 Résultats de l'évaluation du modèle BM25:

Tableau 6 : Modèle BM25 : Topics 1-50

	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
Les variables						
MAP	0.1063	0.1313	0.1290	0.1178	0.1475	0.1438
P@10	0.1180	0.1380	0.1380	0.1380	0.1580	0.1620

Tableau 7 : Modèle BM25 Topics 51-100

	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
Les variables						
MAP	0.1714	0.2093	0.1979	0.1844	0.2265	0.2139
P@10	0.3360	0.3500	0.3460	0.3720	0.4100	0.3980

Tableau 8 : Modèle BM25 Topics 101-150

	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
Les variables						
MAP	0.1154	0.1200	0.1257	0.1591	0.1732	0.1740
P@10	0.2440	0.2100	0.2200	0.3620	0.3280	0.3320

Observations :

Nous remarquons que pour le tableau des requêtes 1-50 la meilleure valeur pour la MAP est de 0.1475 pour l'index 5 avec stemming (Porter) et avec stoplist et celle pour la P@10 est de 0.1620 pour l'index 6 avec stoplist et stemming krovetz. Pour le tableau de requêtes 51-100, nous observons les meilleures valeurs pour MAP et P@10 dans l'index 5 qui est avec stoplist et stemming Porter avec 0.2265 et 0.4100 respectivement. Pour le tableau 101-150, nous remarquons une différence par rapport aux deux premiers tableaux pour la meilleure valeur de P@10 qui est associée à l'index 4 qui est sans stemming et avec stoplist avec 0.3620. Quant à la meilleure valeur du MAP, elle est associée à l'index 6 qui est avec stoplist et stemming Krovetz.

1.4.3 Résultats de l'évaluation du modèle de langue:

Tableau 9 : Modèle Langue $\mu=2000$ Topics 1-50

Les variables	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
MAP	0.1145	0.1514	0.1524	0.1176	0.1554	0.1552
P@10	0.1620	0.1760	0.1820	0.1660	0.1760	0.1860

Tableau 10 : Modèle Langue $\mu=2000$ Topics 51-100

Les variables	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
MAP	0.1821	0.2249	0.2093	0.1818	0.2262	0.2080
P@10	0.3760	0.4120	0.3980	0.3800	0.4220	0.3940

Tableau 11: Modèle Langue $\mu=2000$ Topics 101-150

Les variables	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6
MAP	0.1513	0.1611	0.1631	0.1503	0.1630	0.1626
P@10	0.3500	0.3340	0.3480	0.3480	0.3300	0.3400

Observations :

Nous remarquons que pour le tableau des requêtes 1-50 la meilleure valeur pour la MAP est de 0.1554 pour l'index 5 avec stemming (Porter) et avec stoplist et celle pour la P@10 est de 0.1860 pour l'index 6 avec stoplist et stemming krovetz. Pour le tableau de requêtes 51-100, nous observons les meilleures valeurs pour MAP et P@10 dans l'index 5 qui est avec stoplist et stemming Porter avec 0.2262 et 0.4220 respectivement. Pour le tableau 101-150, nous remarquons une différence par rapport aux deux premiers tableaux pour la meilleure valeur de P@10 qui est associée à l'index 1 qui est sans stemming et sans stoplist avec 0.3500. Quant à la meilleure valeur du MAP, elle est associée à l'index 3 qui est sans stoplist et avec stemming Krovetz.

Partie 2 : Comparaisons et discussions

Dans un premier temps nous allons comparer les résultats au sein d'un même modèle pour savoir si l'utilisation de stemming a amélioré la recherche des documents. Cette comparaison va être faite pour chaque modèle. Ensuite, le même travail va être refait mais pour analyser l'apport de l'utilisation d'une stoplist. Enfin, nous allons comparer les trois modèles de recherche d'information pour savoir lequel donne de meilleurs résultats avec la collection TREC que nous utilisons.

2.1 Stemming versus Not Stemming :

D'après les résultats que nous avons enregistrés, les valeurs les plus élevées du MAP sont associées aux index qui utilisent un stemming pour la phase d'indexation des documents. Nous remarquons aussi que le stemming Porter l'emporte sur le stemming Krovetz avec les requêtes 1-50 et 51-100 pour tous les modèles : le modèle BM25 et le modèle de langue avec l'index 5, quant au modèle tf*idf c'est avec l'index 2 et 5. Par contre, pour les requêtes 101-150, c'est avec le stemming Krovetz que nous obtenons les meilleures valeurs; l'index 3 pour les modèles tf*idf, de langue et l'index 6 pour le modèle BM25.

En regardant les différents ensemble de requêtes, nous avons constaté que les deux premiers ensembles contiennent des mots non racinés; des adjectifs, des participes passés, des mots au pluriel comme "taking", "tracking", "motherhood", ...etc. Et c'est pour cela qu'avec l'algorithme de Porter nous avons eu de meilleurs résultats puisque l'algorithme en question transforme les mots en leurs racines et il s'avère que ces mots se sont retrouvés dans les documents indexés. Ceci a augmenté la précision. Par contre, pour le troisième ensemble, nous remarquons qu'il y a utilisation des mots existants dans le dictionnaire comme "international", "medicine", ...etc. Et ces mots là se trouvent dans les documents indexés tels qu'ils le sont. Donc c'est pour cette raison que Krovetz a donné de bons résultats avec le troisième ensemble.

D'un autre côté, les valeurs les plus élevées du P@10 sont alternées entre stemming Porter et Krovetz pour les requêtes 1-50 et 51-100. Par contre, pour le troisième type de requête (c'est à dire 101-150) nous remarquons que les index n'ayant utilisé aucun stemming (index 1 pour tf*idf et modèle de langue et index 4 pour BM 25) ont les meilleures valeurs.

Malgré les différences observées au niveau de la mesure P@10 sur les résultats des requêtes 101-150, nous pouvons conclure qu'en général l'utilisation d'un stemming augmente la performance dans la recherche d'information. Nous pouvons donc affirmer que le stemming augmente la précision.

2.2 Stoplist versus Not Stoplist :

Selon les résultats obtenus, pour les requêtes 1-50, les valeurs les plus élevées de la précision moyenne (MAP) sont associées aux index n'ayant pas utilisé une stoplist : pour le modèle tf*idf la valeur de l'index 2 le montre bien. Par contre pour les autres modèles, nous remarquons que les valeurs les plus élevées sont celles des index avec stoplist. Mais si on compare ces valeurs avec les valeurs des index 4 qui sont sans stemming et avec stoplist, nous observons une chute majeure de la valeur (0.1475 contre 0.1178 pour le modèle BM25 et 0.1554 contre 0.1176 pour le modèle de langue). Par contre, en comparant avec les valeurs des index qui sont avec stemming et sans stoplist, nous trouvons que les valeurs se rapprochent (0.1475 contre 0.1313 pour le modèle BM25 et 0.1554 contre 0.1524 pour le modèle de langue). Nous constatons alors que la hausse de la valeur avec l'index utilisant une stoplist et un stemming n'est pas dûe au fait d'utiliser une stoplist mais plutôt au stemming. Nous pouvons dire alors, qu'avec l'ensemble de requêtes 1-50, le fait d'utiliser une stoplist n'améliore pas la performance (du moins pas autant que nous nous y attendions). Pour la mesure P@10, nous remarquons généralement une légère amélioration quand une stoplist est utilisée.

Comme nous l'avons susmentionné, la MAP est à sa meilleure valeur avec l'index qui n'utilise pas de stoplist pour l'ensemble 1-50. Ceci est dû au fait d'omettre des mots significatifs pour la requête mais existants dans la stoplist. Prenons comme exemple la requête n°42 "What is End User Computing and Who s Doing It" qui va être transformée, en prenant en compte les stop words, en "End User Computing". Dans ce cas cela va engendrer du bruit du fait qu'un terme générique a été pris en compte.

Passons maintenant à l'ensemble de requête 51-100. Pour les modèles tf*idf et de langue, nous remarquons que l'ensemble de requêtes 1-50 l'utilisation de stoplist n'améliore pas vraiment la performance (une légère amélioration), même si nous observons les meilleurs résultats avec la stoplist. Ceci peut s'expliquer par l'effet du stemming. Nous constatons cependant une légère augmentation dans le cas du modèle BM25 quand une stoplist est utilisée (0.2265 avec l'index 5 avec stemming et stoplist contre 0.2093 avec l'index 2 avec stemming et sans stoplist). Pour le pourcentage de documents pertinents (dans les dix premiers récupérés) mesuré avec P@10, nous remarquons en général une amélioration en utilisant une stoplist.

Nous avons constaté que pour l'ensemble 51-100, la non-utilisation d'une stoplist donne de meilleurs résultats. En effet, si nous prenons l'exemple de la requête n°87 "Criminal Actions Against Officers of Failed Financial Institutions", en enlevant le stopword "against", le sens de la requête va totalement changer car les officiers étaient des victimes dans le cas de la requête originale et ils deviennent des coupables.

Pour l'ensemble de requêtes 101-150, nous observons que l'emploi d'une stoplist a amélioré les performances seulement dans le cas du modèle BM25 avec l'index 6. Cependant, pour les modèles de langue et tf*idf, ne pas employer de stoplist donne de meilleurs résultats (index 3 pour modèle de langue et index 3). Pour le P@10, nous constatons, comme pour le cas de MAP, une amélioration lorsqu'une stoplist est utilisée seulement avec le modèle BM25.

D'après les expériences que nous avons réalisées, nous observons que l'option stemming améliore la performance lors de la recherche des documents pertinents car elle améliore la

précision. Cependant, l'option stoplist dans la majorité des ensembles de requêtes nuit à la précision.

2.3 Comparaison des trois modèles :

Tableau 12: Comparaison des valeurs MAP et P@10 pour les 3 modèles :

	MAP			P @10		
	Tf*idf	BM25	Langue	Tf*idf	BM25	Langue
1 -50	0.1313	0.1475	0.1554	0.1420	0.1620	0.1860
51-100	0.2195	0.2265	0.2262	0.3920	0.4100	0.4220
101-150	0.1659	0.1740	0.1631	0.3340	0.3620	0.3500

Pour la première série de requêtes (1-50), nous remarquons que le modèle de langue donne les meilleures valeurs pour le MAP et P@10 ce qui veut dire que le modèle de langue est plus performant du point de vue précision moyenne sur tous les documents mais aussi sur les 10 premiers documents. Pour l'ensemble de requête 51-100, nous constatons que le BM25 l'emporte sur le modèle de langue bien qu'il y ait une légère différence. Mais pour la précision sur les 10 premiers documents, c'est le modèle de langue qui l'emporte toujours. Pour la troisième série de requêtes, c'est le modèle BM25 qui donne de meilleures performances que ce soit pour le MAP ou le P@10. Généralement, nous pouvons conclure que le modèle de langue est très performant surtout si nous partons d'un point de vue réel du fait que l'utilisateur ne s'intéresse jamais aux 1000 documents mais plutôt aux dix premiers documents retrouvés. Nous notons que le modèle tf*idf est très peu performant comparé aux autres modèles.

Étant donné que le modèle de langue a montré de meilleures performances, nous voulons essayer de l'améliorer encore plus en faisant varier les valeurs de la variable de lissage μ . Nous allons le tester avec les valeurs de $\mu = 1000, 1500, 2500, 3000, 3500$ est cela pour les index **(5,6) (Stemming Porter et stop-liste; Stemming Krovetzs et stop-liste)**. Les résultats sont rapportés dans le tableau

2.4 Variation du paramètre μ dans le modèle langue :

Tableau 13: Variation de la valeur du paramètre μ :

		MAP						P@10					
		$\mu=2000$	$\mu=1000$	$\mu=1500$	$\mu=2500$	$\mu=3000$	$\mu=3500$	$\mu=2000$	$\mu=1000$	$\mu=1500$	$\mu=2500$	$\mu=3000$	$\mu=3500$
Topics 1-50	Index 5	0.1554	0.1554	0.1554	0.1554	0.1554	0.1554	0.1760	0.1760	0.1760	0.1760	0.1760	0.1760
	Index 6	0.1552	0.1552	0.1552	0.1552	0.1552	0.1552	0.1860	0.1860	0.1860	0.1860	0.1860	0.1860
Topics 51-100	Index 5	0.2262	0.2262	0.2262	0.2262	0.2262	0.2262	0.4220	0.4220	0.4220	0.4220	0.4220	0.4220
	Index 6	0.2080	0.2080	0.2080	0.2080	0.2080	0.2080	0.3940	0.3940	0.3940	0.3940	0.3940	0.3940
Topics 101-150	Index 5	0.1630	0.1630	0.1630	0.1630	0.1630	0.1630	0.3300	0.3300	0.3300	0.3300	0.3300	0.3300
	Index 6	0.1626	0.1626	0.1626	0.1626	0.1626	0.1626	0.3400	0.3400	0.3400	0.3400	0.3400	0.3400

On remarque, malgré la variation de la valeur du paramètre μ que ni la moyenne de la précision ni le ratio pour les dix premiers documents n'ont changé. Ceci nous permet de conclure que la valeur μ (qui est le paramètre appelé pseudo-fréquence) n'a pas influencé la précision. Elle reste stable et cela est dû à la taille élevée du document. Si on double quasiment la valeur de μ , elle n'influence pas la probabilité de la précision. Cette observation est déduite d'après la formule de la probabilité avec le lissage Dirichlet du cours

$$P_{Dir}(m_i | D) = \frac{tf(m_i, D) + \mu P_{ML}(m_i | C)}{|D| + \mu}$$

où $|D|$ est la taille du document (le nombre total d'occurrences de mots), et $tf(m_i, D)$ est la fréquence du mot m_i dans D .

Partie 3: Compréhension du système

3.1 Le programme d'indexation : IndriBuildIndex

La commande IndriBuildIndex commence par récupérer les paramètres parmi lesquels nous pouvons citer le stemmer, le chemin de la collection, le chemin où le fichier indexé va être placé, ...etc à travers le parsing du fichier paramètre. Puis, à travers les méthodes de la classe `indri::api::IndexEnvironment`, la figure 9 illustre l'ensemble des méthodes utilisées pour la construction de l'index.

Définition de la quantité de mémoire à utiliser pour les structures internes

`indri::api::IndexEnvironment::setMemory`

Normalisation du texte durant la phase d'indexation en normalisant les majuscules, minuscules et quelques ponctuations

`indri::api::IndexEnvironment::setNormalization`

Définition de la stratégie (ordered ou unordered) à utiliser par l'indexer pour l'optimisation de la vitesse d'indexation des offset annotations

`indri::api::IndexEnvironment::setOffsetAnnotationIndexHint`
(`indri::parse::OffsetAnnotationIndexHint hintType`)

Définition du stemmer qui va être utilisé

`indri::api::IndexEnvironment::setStemmer(const std::string & stemmer)`

Définition de la liste des stopwords

`indri::api::IndexEnvironment::setStopwords`
(`const std::vector< std::string > & stopwords`)

Vérification du fichier, où l'index va être placé, s'il est corrompu à cause d'un crash d'indexation alors il faut le récupérer et continuer la construction de l'index. Sinon il faut créer le fichier en question

`indri::api::IndexEnvironment::open(const std::string & repositoryPath,`
`IndexStatus * callback = 0)`

`indri::api::IndexEnvironment::create(const std::string &`
`repositoryPath, IndexStatus * callback = 0)`

Définition du chemin du document où se trouve la collection

`indri::api::IndexEnvironment::setDocumentRoot(const`
`std::string & documentRoot)`

Le choix des tokenizers, parsers, ...etc

`indri::api::IndexEnvironment::addFileClass(const`
`indri::parse::FileClassEnvironmentFactory::Specification & spec)`

Itération sur les fichiers de la collection pour construire l'index au fur et à mesure

`indri::api::IndexEnvironment::addFile`
(`const std::string & fileName`).

Figure 9: Illustration des méthodes du programme IndriBuildIndex

3.2 Le programme de recherche: IndriRunQuery

L'exécution de la commande **IndriRunQuery** nécessite l'exécution de 8 étapes que nous allons décrire dans la figure 10 : Nous allons nous arrêter pour détailler que l'étape numéro 7



Figure 10: Illustration des méthodes du programme IndriRunQuery

Étape 7 : L'évaluation des documents de l'index

- a) les statistiques qui ont été collectées lors de l'étape 5 sont mis dans des noeuds qui sont des objets de type **indri::lang::ScoreAccumulatorNode** à travers la méthode **ScoreAccumulatorNode (ScoredExtentNode *scoredNode)** qui prend en paramètre un noeud de type **ScoredExtentNode** et qui retourne un noeud de type **indri::lang::ScoreAccumulatorNode**.
- b) Appliquer un "copier" de type **indri::lang::UnnecessaryNodeRemoverCopier** à travers la classe **indri::lang::UnnecessaryNodeRemoverCopier::SingleChildWalker** qui va réunir les noeuds enfants d'un même parent qui sont reliés par un OU ou un ET et retourne un nouvel arbre parsé après toutes les modifications.
- c) **Récupération du dénombrement** : La classe **indri::lang::ContextSimpleCountCollectorCopier** qui est appliquée afin d'extraire le nombre de contexte à des fins statistiques pour l'évaluation du meilleur noeud. La classe **indri::lang::FrequencyListCopier** va remplir le vecteur qui contiendra les fréquences des termes déjà récupérés dans l'étape 5.
- d) **L'emboîtement des poids**: Parcourir l'arbre des noeuds avec la classe **indri::lang::WeightFoldingCopier**. Une fois qu'on est dans un noeud candidat qui est un objet de type **Indri::lang::WeightNode** on récupère le poids des noeuds enfants et on calculera à la suite le nouveau poids du noeud père.
- e) **Création du graphe acyclique dirigé (DAG)** : Après avoir eu tous les poids optimaux un graphe acyclique sera créé avec la classe: **Indri::lang::DagCopier** qui parcourt l'arbre et cherche les noeuds équivalents qui seront emboîtés en un seul noeud afin d'avoir un arbre simple et non cyclique ou récursif.
- f) **Construction du réseau d'inférence** : Une fois l'arbre optimisé la classe **indri::infnet::InferenceNetworkBuilder** s'occupe de construire le réseau d'inférence. Durant cette étape, tous les stopwords vont être omis du graphe et en plus le stemming des termes des noeuds de type **index**. Ensuite, pour chaque noeud qui contient un terme indexé, un objet de type **indri::infnet::DocListIteratorNode** est créé pour parcourir dans la prochaine étape l'inverted list des documents indexés associée au terme en question.
- g) **Évaluation des documents**: C'est l'étape cruciale de toute la recherche. On a le réseau inféré on l'utilise pour évaluer les documents candidats pour chaque terme. C'est la méthode **indri::infnet::InferenceNetwork::_evaluateDocument(indri::index::Index & index, de la classe indri::infnet::InferenceNetwork** qui fait l'évaluation des documents. Les différents noeuds qui peuvent être évalués sont :
 - **indri::infnet::Annotator**
 - **indri::infnet::ContextCountAccumulator**
 - **indri::infnet::ContextSimpleCountAccumulator**
 - **indri::infnet::ListAccumulator**
 - **indri::infnet::ScoredExtentAccumulator**

Une fois que tous les documents ont leurs scores, l'étape suivante serait de les ordonner et donner le résultat.

Références :

- 1 Indri's standard stopwords list : <http://www.lemurproject.org/stopwords/stoplist.dft>
- 2 lemur project software wiki : <http://sourceforge.net/p/lemur/wiki/Home/Indri>
- 3 documentation :
http://lemur.sourceforge.net/indri/classindri_1_1api_1_1IndexEnvironment.html#ad42747e233dab4a2ef145816aa52d0e6
- 4 <http://lemur.sourceforge.net/indri/IndriParameters.html>
- 5 http://www.lemurproject.org/doxygen/lemur/html/classindri_1_1api_1_1Parameters.html
- 6 http://lemur.sourceforge.net/indri/classindri_1_1index_1_1Index.html
- 7 http://lemur.sourceforge.net/indri/IndriBuildIndex_8cpp.html
- 8 http://lemur.sourceforge.net/indri/namespaceindri_1_1query.html
- 9 http://lemur.sourceforge.net/indri/IndriRunQuery_8cpp.html
- 10 http://lemur.sourceforge.net/indri/classindri_1_1infnet_1_1InferenceNetwork.html
- 11 http://trec.nist.gov/trec_eval/