Rapport TP : Recherche d'informations textuelles

BIZEUL Solal et POMMIER Aurélie

Objectifs :

Nous avons pour objectif de créer un algorithme de moteur de recherche textuel qui a pour but de retourner à l'utilisateur le document le plus pertinent pour sa requête en langage naturel. Pour cela, il y a deux étapes à réaliser :

- L'indexation des documents d'un corpus donnée
- La recherche du ou des documents les plus similaires à la requête

Nous allons implémenter plusieurs systèmes d'indexation et de recherche afin de comparer les différents systèmes et de déterminer le plus performant.

Modèles évalués :

Tout d'abord, notons qu'un pré-traitement est réalisé sur chaque document pour obtenir le sac de mots de celui-ci avec la fonction `SAC(doc,stoplist,Separators)`. Cette fonction utilise trois méthodes : la tokenisation pour séparer les mots, la stoplist pour supprimer les termes non pertinents (pas de sens apporté) et le stemming pour supprimer les terminaisons et ainsi regrouper des termes qui ont le même sens.

Modèle ensembliste avec une mesure de similarité de Dice :

Avec ce modèle, le descripteur d'un document correspond à l'ensemble des « mots », préalablement traités (Tokenisation, Stoplist et Stemming), apparaissant dans le document. On ne s'intéresse alors pas au nombre d'occurrences de ces mots mais seulement s'il apparaît ou non dans un document.

On mesure ici la similarité entre deux documents (ou un document du corpus et une requête) avec le coefficient de Dice. D'autres mesures de similarité existent et ont des performances très similaires.

$$\delta_{Dice}(D, Q) = 2 \times \frac{|D \cap Q|}{|D| + |Q|}$$

Modèle vectoriel avec pondération Tf et similarité du cosinus

Pour le modèle vectoriel, nous prenons cette fois en compte la fréquence d'apparition des termes dans le document pour apporter des informations supplémentaires pour le calcul de similarité. Tous les descripteurs ont tous la même taille qui est la taille du vocabulaire complet du corpus et chaque coordonnée correspond à la fréquence du i-ème terme du vocabulaire.

$$v_j = (\mathrm{tf}(t_1, d_j), \mathrm{tf}(t_2, d_j), \ldots, \mathrm{tf}(t_{|V|}, d_j))$$

Nous utilisons la similarité du cosinus pour calculer la similarité de deux documents à partir de leur vecteur Tf.

$$\sigma(v_j, v_q) = \frac{< v_j, v_q >}{||v_j||_2 . ||v_q||_2} = \frac{\sum_{i=1}^{|V|} v_j^{(i)} v_q^{(i)}}{||v_i||_2 . ||v_q||_2}$$

Modèle vectoriel avec pondération Tf-Idf et similarité du cosinus

Ce modèle est très comparable au précédent. Cependant, nous multiplions les coefficients Tf par les Idf afin d'augmenter les coefficients des termes qui apparaissent peu dans le corpus en général et donc qui devrait être plus pertinent puisqu'ils apportent une information plus précise.

$$v_j = (\text{tf}(t_1, d_j).\text{idf}(t_1), \text{tf}(t_2, d_j).\text{idf}(t_2), \ldots, \text{tf}(t_{|V|}, d_j).\text{idf}(t_{|V|}))$$

Nous utilisons, ici encore, la similarité du cosinus.

Indexation :

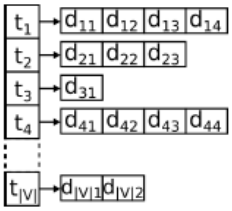
Pour chaque modèle, nous avons implémenté deux types d'indexation : l'index linéaire et l'index inversé.

Index linéaire :

Ce type d'index consiste à avoir un index qui associé chaque id de document à son descripteur pour ensuite faire une recherche qui parcourt tous les descripteurs et calcul la similarité de chaque document avec la requête pour ensuite les trier par ordre décroissant et obtenir les documents les plus pertinents.

Index inversé :

Pour ce type d'indexation, on associe chaque terme du vocabulaire à une liste des descripteurs qui le contiennent.



Ainsi, lors de la recherche, on retourne l'union des sous-listes de descripteurs qui correspondent aux termes de la requête. Une fois cette short-list obtenue, nous calculons uniquement les similarités entre les descripteurs de celle-ci et le descripteur de la requête ce qui permet de gagner du temps de recherche et donc augmenter les performances de l'algorithme.

Protocole d'évaluation :

Pour chaque jeu de données que nous avons à notre disposition, nous allons comparer les courbes rappel-précision calculées ainsi que les scores de mAP à partir de la classe `Evaluator` du fichier python `ir_evaluation.py` pour les trois modèles (ensemblistes, vectoriel Tf et vectoriel Tf-Idf). Pour cela, nous utilisons les vérités-terrain que nous avons.

De plus, nous avons calculé les temps moyens de construction des index et les temps d'exécution des requêtes selon le type d'index (linéaire et inversé) pour comparer leurs performances.

Pour les courbes rappel-précision qui suivent, la courbe bleue correspond au descripteur ensembliste, la courbe orange au descripteur TF et la courbe verte au descripteur TF-IDF. Pour le

descripteur ensembliste, nous avons utilisé la mesure de Jaccard, car c'est celle qui nous permettait d'obtenir les meilleurs résultats en général.

Nous avons choisi d'utiliser la base `med` pour évaluer notre implémentation, car elle a un nombre de documents moyen (1031) et un nombre de requêtes assez faible (30) mais avec un grand nombre de documents pertinents par requête selon le groundtruth. Ceci nous permet d'avoir une exécution assez rapide de tous les algorithmes utilisés ici. Nous avons tout de même voulu tester les autres corpus et comparer leur courbe Rappel-Précision. Néanmoins, tous les tests de temps d'exécution ainsi que les tests sans Stemming et Stoplist seront effectués sur la base `med`.

Résultats :

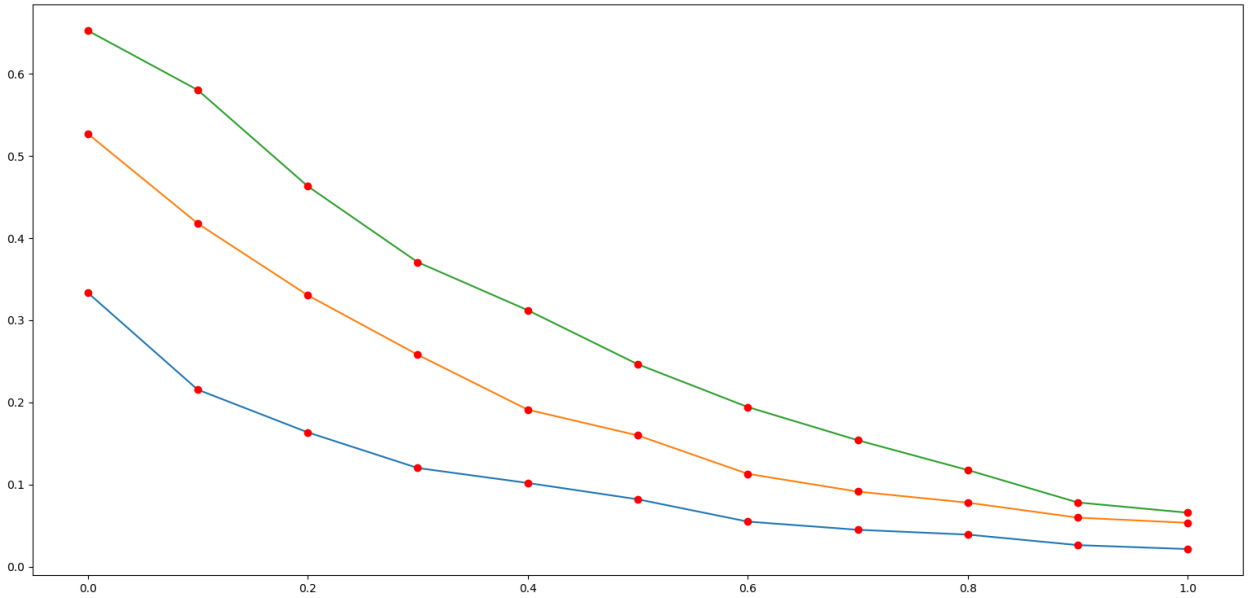


Figure 1 : Courbe Rappel-Précision pour les trois modèles pour le jeu de données `cacm`

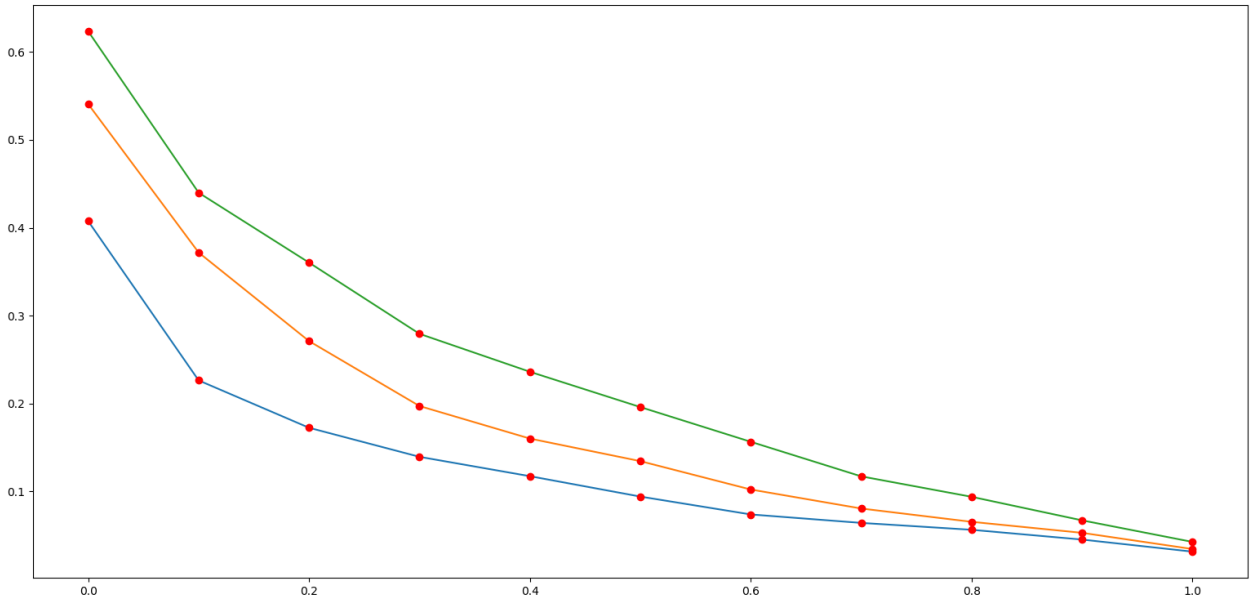


Figure 2 : Courbe Rappel-Précision pour les trois modèles pour le jeu de données `cisi`

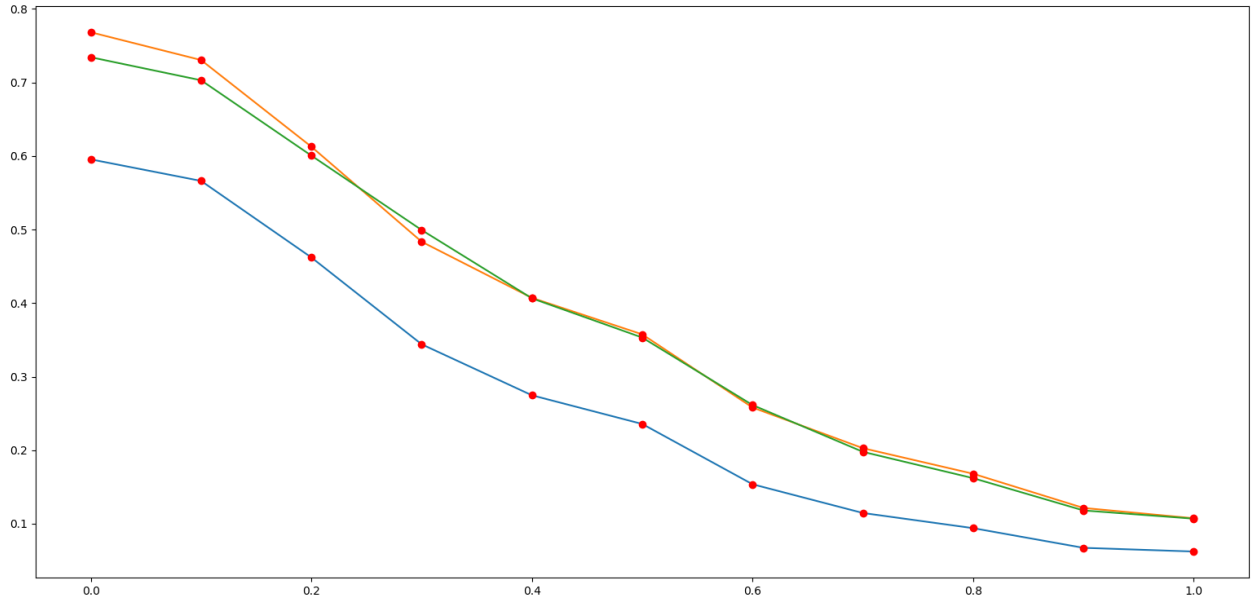


Figure 3 : Courbe Rappel-Précision pour les trois modèles pour le jeu de données `cran`

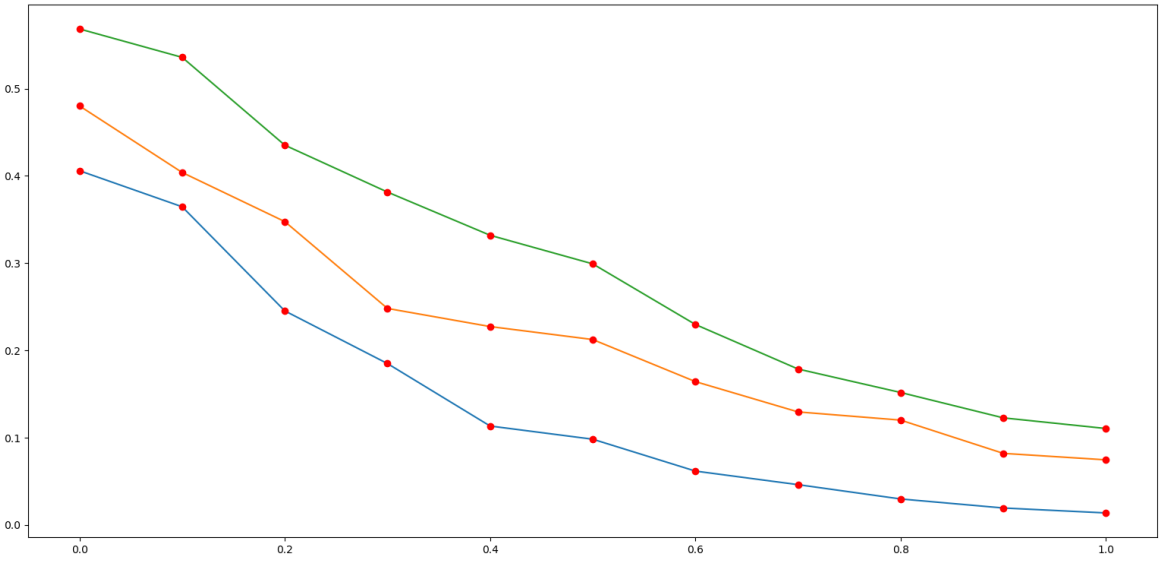


Figure 4 : Courbe Rappel-Précision pour les trois modèles pour le jeu de données `lisa`

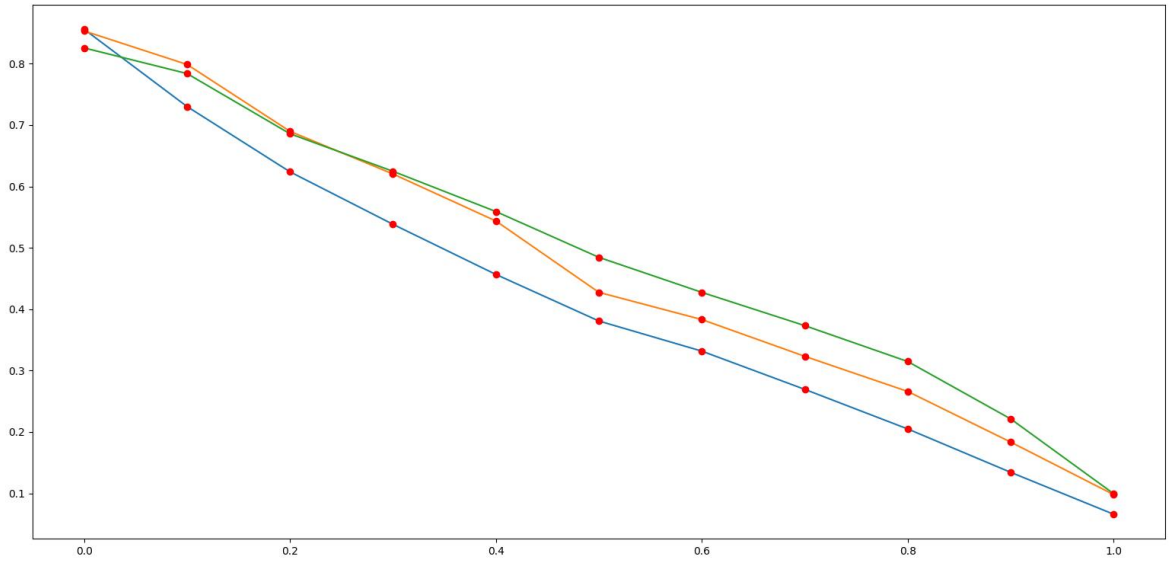


Figure 5 : Courbe Rappel-Précision pour les trois modèles pour le jeu de données `med`

	Ensembliste	TF	TF-IDF
Temps de construction index linéaire	4.30	11.47	24.71
Temps de construction index inversé	10.58	16.65	36.53
Temps moyen de recherche index linéaire	0.08	2.13	2.36
Temps moyen de recherche index inversé	0.02	0.35	0.50
mAP	0.40	0.46	0.47

Figure 6 : Comparaison des temps d'exécution et des mesures mAP

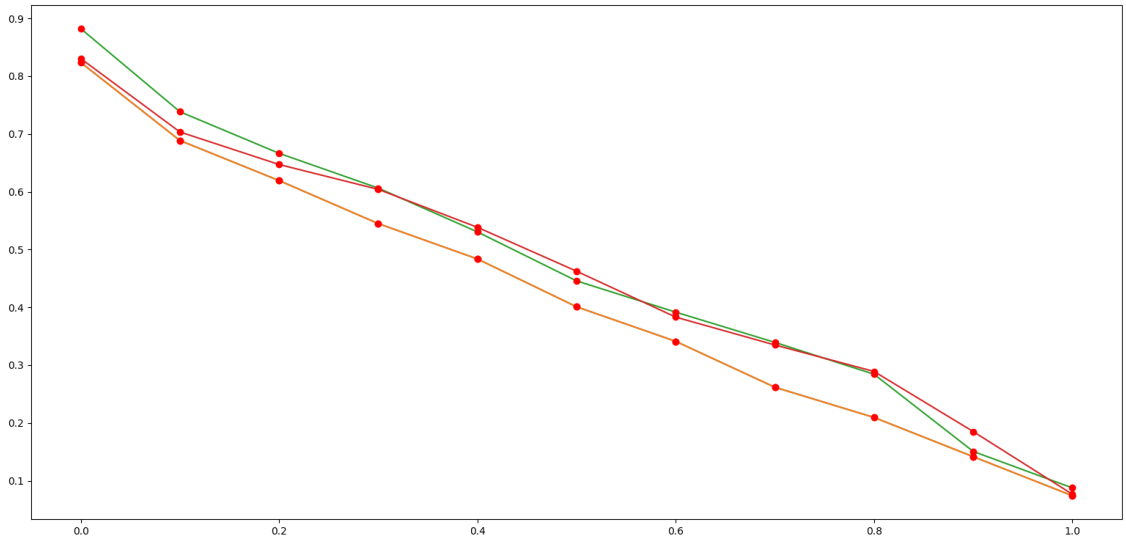


Figure 7 : Courbe Rappel-Précision pour les trois modèles sans Stemmer pour le jeu de données `med`

	Ensembliste	TF	TF-IDF
Temps de construction index linéaire	2.80	14.72	26.07
Temps de construction index inversé	6.34	25.96	32.38
Temps moyen de recherche index linéaire	0.14	4.41	2.83
Temps moyen de recherche index inversé	0.01	0.33	0.18
mAP	0.40	0.45	0.44

Figure 8 : Comparaison des temps d'exécution et des mesures mAP sans Stemmer

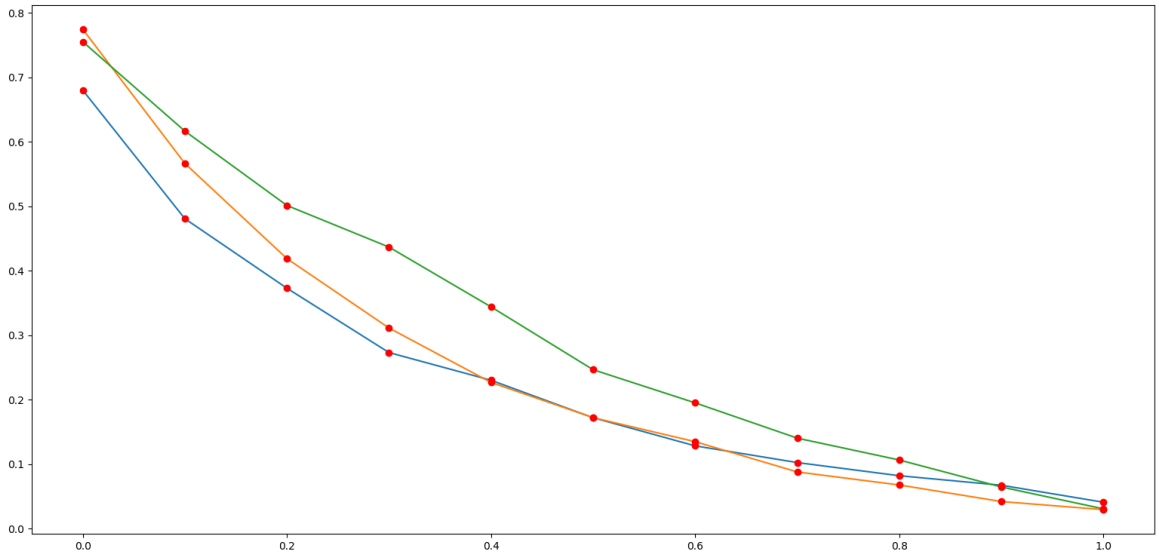


Figure 9 : Courbe Rappel-Précision pour les trois modèles sans Stoplist pour le jeu de données `med`

Temps de construction index linéaire	2.94	7.87	46.33
Temps de construction index inversé	4.29	9.81	49.92
Temps moyen de recherche index linéaire	0.15	1.85	2.13
Temps moyen de recherche index inversé	0.09	1.78	2.36
mAP	0.21	0.23	0.30

Figure 10 : Comparaison des temps d'exécution et des mesures mAP sans Stoplist

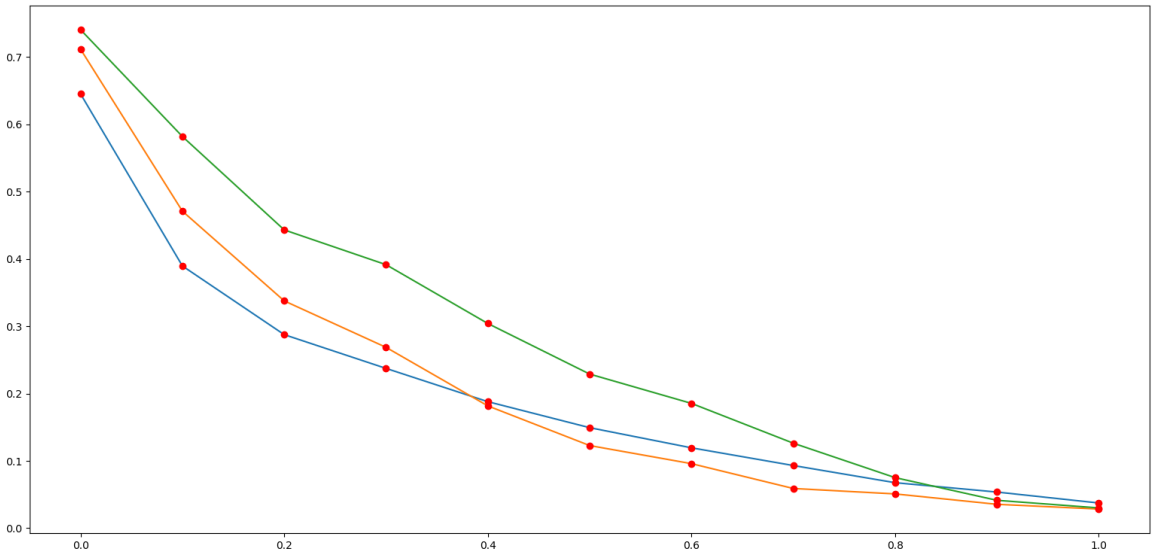


Figure 11 : Courbe Rappel-Précision pour les trois modèles sans Stemmer et sans Stoplist pour le jeu de données `med`

	Ensembliste	TF	TF-IDF
Temps de construction index linéaire	0.63	12.00	30.05
Temps de construction index inversé	1.70	9.97	31.29
Temps moyen de recherche index linéaire	0.13	3.08	2.61
Temps moyen de recherche index inversé	0.16	2.29	2.67
mAP	0.19	0.19	0.27

Figure 12 : Comparaison des temps d'exécution et des mesure mAP sans Stoplist et sans Stemmer

Analyse des résultats :

Tout d'abord, comparons les scores obtenus. On remarque que le descripteur ensembliste résulte en un score inférieur que le descripteur TF, qui donne lui-même un score inférieur que le descripteur TF-IDF (cf. Figure 6). Ceci reste vrai pour la plupart des jeux de données, et est logique par rapport à la théorie.

Ensuite, nous pouvons voir que les scores obtenus dépendent énormément du jeu utilisé. Tandis que `med` atteint 0.87 au maximum de la courbe Rappel-Précision (cf. Figure 5), `lisa` n'atteint que 0.55 (cf. Figure 4). D'ailleurs, plus les résultats d'un corpus sont mauvais et plus les courbes correspondant à chaque descripteur sont éloignées les unes des autres (comparer Figure 1,2 et 4 aux Figures 3 et 5). Cela signifie que les descripteurs vectoriels sont le plus utile lorsque le jeu de donnée est complexe.

Nous pouvons également remarquer que les scores de mAP correspondent bien aux courbes observées, c'est-à-dire que le mAP du descripteur ensembliste est plus petit que celui du descripteur TF, qui est plus petit que celui du descripteur TF-IDF.

Nous pouvons maintenant observer les temps d'exécution de l'algorithme (cf. Figure 6). Tout d'abord, tous les temps de recherche et de construction sont moins longs pour le descripteur ensembliste que pour le descripteur TF. Ceci semble logique, car les algorithmes TF doivent calculer le vecteur TF de chaque terme. De même, les algorithmes utilisant le descripteur TF sont plus rapides que ceux utilisant le descripteur TF-IDF. Ceci semble également logique, car les algorithmes TF-IDF doivent évaluer l'IDF de chaque terme, ce qui rajoute du temps de calcul.

Ensuite, nous pouvons voir que l'index linaire est plus rapidement construit que l'index inversé, mais les recherches prennent moins de temps en utilisant l'index inversé. Ceci suit parfaitement la théorie : le but d'un index inversé est justement de faire des recherches très rapide, au coût d'un temps de construction un peu plus long.

Finalement, nous avons analysé la contribution de la stoplist et du stemming sur nos résultats pour le jeu de données `med`.

En enlevant le stemming, contrairement à ce que l'on pensait, le score n'a que très peu diminué (cf. Figure 7). Ceci est peut-être dû au fait que `med` contient beaucoup de termes scientifiques pointus qui ont donc très peu de racines en commun, ce qui réduit l'utilité du stemming.

En retirant la stoplist (en laissant donc les mots comme « the », « is », « a », etc. dans les documents), nous nous sommes rendu compte de sa grande importance. Les index se sont construits plus vite, mais les recherches prenaient beaucoup plus de temps et les résultats étaient bien moins bons. L'index inversé prend même plus de temps en recherche que l'index linéaire pour le descripteur TF-IDF. Ceci est dû au fait qu'il faille calculer la valeur d'IDF de beaucoup plus de mots qui sont présents dans un grand nombre de documents, ce qui augmente le nombre de valeurs contenues dans les dictionnaires du descripteur total (cf. fonction desc_total dans le code).

En retirant la stoplist et le stemmer, les courbes de Rappel-Précision sont encore plus mauvaises et les scores encore plus bas. On remarque plus l'impact du stemmer ici que lorsque l'on enlève uniquement ce dernier.

En conclusion, nous avons pu, grâce à ce TP, analyser les performances de différents systèmes d'indexation, ainsi qu'appréhender l'implémentation de différents algorithmes de calcul de descripteurs.