Fondement de la Recherche d'Information Web

Romain Pascual et Adam Hotait

13 Mars 2019

Le but de ce projet est de mettre en œuvre les notions fondamentales d'indexation et de modèles de recherche vues en cours par la réalisation d'un petit moteur de recherche ad-hoc sur deux bases statiques de documents textuels.

1 Corpus

Les corpus de travail sont la collection CACM et le corpus issu du cours CS 276 de l'Université de Stanford.

- La collection CACM (Communications of the ACM) est une base assez classique en recherche d'information qui contient les titres, auteurs et résumés d'un ensemble d'articles scientifiques issus des Communications d'ACM entre 1958 et 1979. Un des avantages principaux de cette base est qu'un ensemble de requêtes et de jugements de pertinence est disponible.

- La seconde collection contient un ensemble de pages web du domaine stanford.edu. C'est un corpus de 170 MBs. Il est organisé en 10 sous-répertoires (numérotés de 0 à 9). Chaque fichier correspond au contenu textuel d'une page web individuelle. Chaque nom de fichier est unique dans chaque sous-répertoire mais ce n'est pas le cas globalement.

2 Création d'un index inversé et moteur de recherche booléen et vectoriel

Il s'agit ici de mettre en place l'ensemble des traitements utiles vus en cours pour transformer un document donné en une liste de termes d'index.

2.1 Traitement linguistique

Les étapes de traitements linguistiques sont au nombre de trois : la tokénisation, la comparaison avec une stop-liste et la lemmatisation. Toutefois ces trois étapes sont ici simplifiées. Ainsi, on limite l'étape de tokénisation à considérer tout caractère non alpha-numérique comme un séparateur de mots. Cette étape est donc simplement réalisées en utilisant des expressions régulières. La comparaison avec une stop-liste est effectuée sur la première collection (puisqu'une stop liste est fournie) mais pas sur la seconde. Par ailleurs cette étape est précédé d'une transformation de tous les caractères alphabétique en minuscule. Enfin l'étape de lemmatisation ou troncature est ici négligée dans la collection CACM (la lemmatisation étant déjà effectuée pour la collection CS276).

Collection CACM Dans le cas de cette collection, un fichier unique contient l'ensemble des documents. Les documents sont séparés par un ensemble de marqueurs et contiennent plusieurs champs eux-même identifiés par des marqueurs. Dans le cadre de ce projet, on se limite aux marqueurs des champs correspondant à l'identifiant, le titre, le résumé et les mots clefs relatifs aux documents. Bien que non demandé dans l'énoncé, nous avons choisi d'inclure les noms de familles des champs d'auteurs, car plusieurs des requêtes étudiée plus tard dans l'étape d'évaluation nécessitent les auteurs.

Le traitement linguistique ainsi effectué nous permet de trouver 192129 tokens dans la collection (**Question 1**), pour une taille de vocabulaire de 9496 mots (**Question 2**).

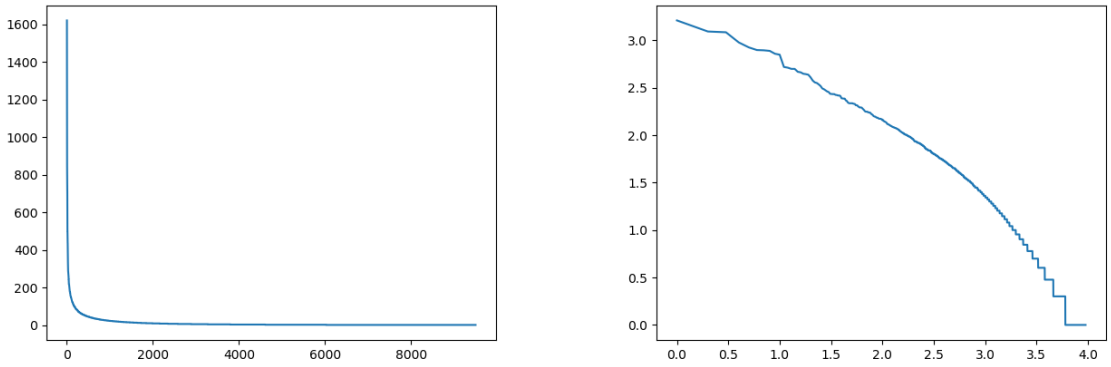
Par régression linéaire, on obtient les paramètres ($k \simeq 28.8$ et $b \simeq 0.48$) de la loi de Heap :

$$M = kT^b$$

où M est le nombre de tokens et T la taille du vocabulaire (**Question 3**).

On peut alors estimer à 62679 la taille du vocabulaire pour une collection de 1 million de tokens (**Question 4**).

On peut, de plus, tracer les graphes fréquence f vs rang r (figure 1a) et $log(f)$ vs $log(r)$ (figure 1b pour tous les tokens de la collection (**Question 5**).



(a) frequence f vs rang r (b) $log(f)$ vs $log(r)$

Figure 1: Graphes fréquences et rang

Collection CS276 Dans cette collection, les documents ont déjà été pré-traités et ne contiennent que des séquences de mots.

On peut effectuer le même traitement linguistique, ce qui permet de trouver 25578547 tokens dans la collection (**Question 1**), pour une taille de vocabulaire de 310162 mots (**Question 2**).

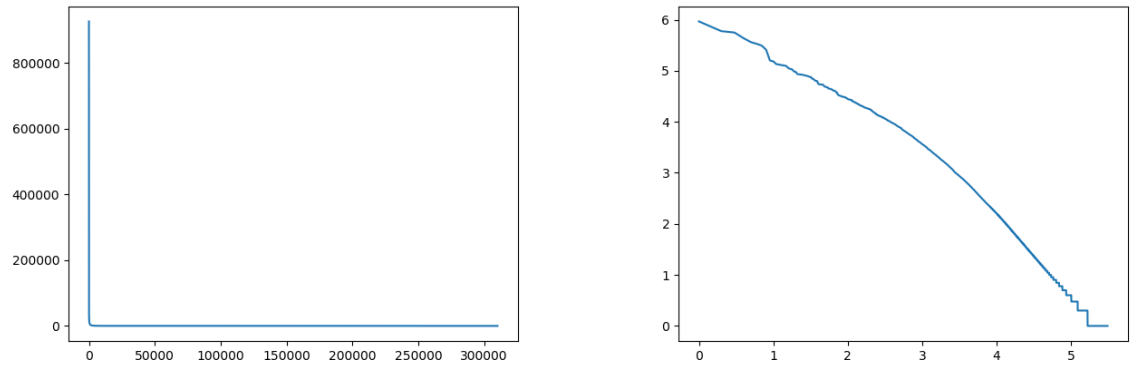
Par régression linéaire, on obtient les paramètres ($k \simeq 0.05$ et $b \simeq 0.91$) de la loi de Heap :

$$M = kT^b$$

où M est le nombre de tokens et T la taille du vocabulaire (**Question 3**).

On peut alors estimer à 142631 la taille du vocabulaire pour une collection de 1 millions de tokens (**Question 4**).

On peut, de plus, tracer les graphes fréquence f vs rang r (figure 2a) et $log(f)$ vs $log(r)$ (figure 2b pour tous les tokens de la collection (**Question 5**).



(a) frequence f vs rang r (b) $log(f)$ vs $log(r)$

Figure 2: Graphes fréquences et rang

2.2 Indexation

Il reste maintenant à créer les indexes. Pour cela nous avons décider d'utiliser des tables de hachage, aussi bien pour le dictionnaire (terme, termeID) que pour l'index en lui-même. Puisque les documents sont déjà référencés à l'aide d'un identifiant unique dans la collection CACM, le dictionnaire (document, documentID) n'est pas nécessaire. Enfin, afin de pouvoir mener à bien la recherche vectorielle, il est nécessaire de garder la fréquence d'un terme dans un document, nous avons majoritairement utilisé un index où la liste de postings contient un couple (documentID, freq) et pas simplement l'identifiant du document. Notons que la collection CACM est petite et l'index peut être construit directement en mémoire. En revanche la collection CS276 est considérée comme 10 blocs indépendants et nous avons utilisé l'algorithme SPIMI (ou Single Pass In Memory Indexing). Les résultats intermédiaires sont stockés dans le dossier `indexes`.

Modèle de recherche booléen Dans ce modèle, la requête est donnée sous la forme d'une expression booléenne. Les opérateurs acceptés sont la conjonction $\wedge$ (et), la disjonction $\vee$ (ou) et la négation $\neg$ (non).

Afin d'éviter les problèmes de parenthésage, les requêtes booléennes sont manipulées sous forme infixe (les opérateurs d'arités 2 sont placés devant les deux éléments auxquels ils s'appliquent). Les connecteurs logique sont en anglais et en majuscule: 'AND', 'OR' et 'NOT'. Les mots doivent eux être renseignés en minuscule. L'opérateur 'AND' se traduit par une intersection des listes de posting. L'opérateur 'OR' se traduit par une union des listes de posting. Enfin l'opérateur 'NOT' correspond à un passage au complémentaire.

Modèle de recherche vectoriel Il s'agit ici du modèle vectoriel avec le calcul de similarité par la mesure du cosinus. Le processus de recherche est basé sur les résultats d'indexation de l'étape précédente. Trois méthodes de pondération ont été implémentées :

- la pondération tf-idf,
- la pondération tf-idf normalisée,
- la fréquence normalisée.

2.3 Evaluation pour la collection CACM

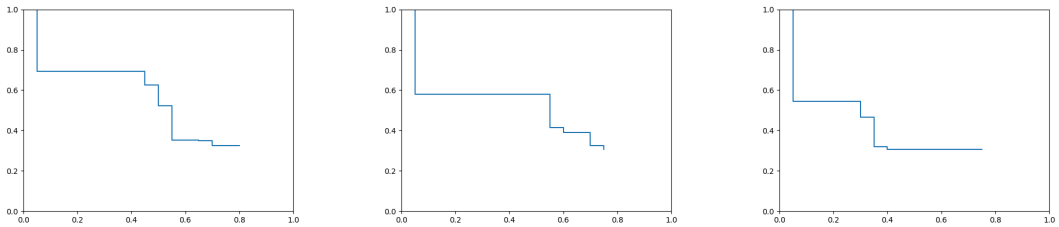
On cherche à évaluer les systèmes de recherche de deux manières différentes :

- les mesures de performances : temps de calcul pour l'indexation, temps de réponse à une requête et occupation de l'espace disque par les différents inde.
- les mesures de pertinence à l'aide des requêtes et jugements de pertinence disponibles.

Mesure de performances L'index a été crée en 0.4490s et est stocké dans un fichier de 975.9ko sur le disque. Pour des recherches booléennes, le temps de réponse est de l'ordre de 10^{-4}s. Pour des recherches vectorielles, le temps de réponse est de l'ordre de 10^{-2}s pour les pondérations 'tf-idf' et 'tf-idf-norm' et de l'ordre de 10^{-1}s pour la pondération 'freq-norm'.

Mesure de pertinence L'évaluation de la pertinence des requêtes est effectuée à l'aide de la série de requête et de documents pertinents proposés dans la collection CACM. On obtient ainsi différentes courbes précision-rappel (figure 3). On calcule aussi la E-mesure et la F-measure moyenne sur l'ensemble des requêtes. On obtient ainsi les résultats suivants :

Pondération	E mesure	F mesure
tf-idf	0.84	0.14
tf-idf normalisé	0.82	0.16
fréquence normalisé	0.84	0.15



(a) Courbe précision-rappel pour la requête 36 et la pondération 'tf-idf'

(b) Courbe précision-rappel pour la requête 36 et la pondération 'tf-idf-norm'

(c) Courbe précision-rappel pour la requête 36 et la pondération 'freq-norm'

Figure 3: Courbes Précision-Rappel pour la requête 36.

Remarque : il est possible d'exécuter l'évaluation de la collection, ce qui créera un fichier `CACM_evaluation` dans le dossier `results` avec des résultats d'évaluation un peu plus détaillés.

3 Création d'un index inversé compressé

Il s'agit ici d'implémenter la méthode de compression Variable Byte Encoding pour la collection CS276.

Tout d'abord, on utilise la méthode Delta Coding qui consiste à stocker la différence entre deux documentID consécutifs dans la liste de posting plutôt que leur valeur. Il convient ensuite d'implémenter la méthode de Variable Byte Encoding qui consiste à coder sur moins de bit les plus petits entiers et sur plus de bits les plus grands. Plus précisément on code sur un octet les restes consécutifs de la division euclidienne par 128 avec le premier bit qui sert à spécifier si l'octet suivant correspond toujours à l'entier en cours. Il devient alors difficile d'utiliser un

délimiteur dans les listes de posting. Ainsi une liste est stockée en spécifiant d'abord le nombre d'éléments puis ces éléments énumérés un par un. Ainsi lorsque l'on a lu suffisamment de valeur on sait que l'on a lu tout la liste.

Par ailleurs, nous avons aussi essayé d'utiliser `pickle` qui est un module de sérialisation Python pour avoir un élément de comparaison avec la compression VBE.

Type de Compression	Nom du fichier	Taille (en Mo)
Delta Coding (enregistrement en string)	CS276_DC.txt	38,9
Delta Coding + Variable Byte Encoding	CS276_VBE.dat	139,7
Delta Coding + Serialisation (pickle)	CS276_DC_Ser.index	31,9

On remarque que la sérialisation diminue légèrement la taille du dossier mais que l'implémentation de VBE n'est pas très satisfaisante. Cela provient du fait que stocker un entier sur un octet n'est en effet pas très pertinent. On peut par exemple noter que l'index comprend 13769488 valeurs stockées dans 310162 liste de posting. Si on moyenne le nombre de bits pour les termeIDs entre 0 et 310162, on obtient ~ 24, ce qui correspond à 3 octets. Avec 13769488 valeurs stockées dans 310162 liste de posting, on obtient 45 éléments en moyenne par liste de posting, ce qui tient sur un octet. Étant donné que la collection comporte 98998 documents, en moyenne un documentID se stocke sur ~ 22 bits, c-à-d 3 octets.

On obtient alors une estimation à 167Mo de la taille du fichier. Le fichier réel a une taille de 139.7Mo ce qui peut signifier que le Delta Code est peu efficace.

Toutefois, cela ne semble pas être le cas lorsque l'on compare les fichiers stockés en string avec et sans Delta Code, on obtient une réduction par 3 de la taille du fichier.

Enfin le fichier avec Delta Coding et Variable Byte Encoding est plus lourd en mémoire que l'index d'origine. Il y a donc probablement une erreur dans notre gestion des valeurs sous forme de bits.

Remarque : il est possible d'obtenir ces trois fichiers dans le dossier `results` en exécutant le code correspondant à la collection CS276.