

Fouille de textes

Pré-traitements textuels et modèle de base pour représenter des documents textuels

Solen Quiniou

`solen.quiniou@univ-nantes.fr`

Master 2 ATAL – Université de Nantes

Année 2020-2021

Plan du cours

- 1 Pré-traitements du texte
- 2 Représentation vectorielle d'une collection de documents
- 3 Similarité syntaxique
- 4 Implémentation de la représentation vectorielle d'une collection de documents
- 5 Références

Plan du cours

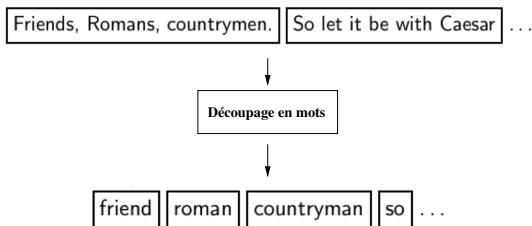
- 1 Pré-traitements du texte
- 2 Représentation vectorielle d'une collection de documents
- 3 Similarité syntaxique
- 4 Implémentation de la représentation vectorielle d'une collection de documents
- 5 Références

Introduction

- Les **pré-traitements textuels** visent à « nettoyer » les textes sélectionnés
 - Les pré-traitements effectués sont à choisir en fonction de la tâche de fouille de texte réalisée
- **Pré-traitements classiques**
 - ▶ Segmentation du texte en phrases et en mots
 - ▶ Normalisation et traitement des accents, des signes diacritiques et de la casse des mots
 - ▶ Lemmatisation et étiquetage grammatical des mots
 - ▶ Racinisation des mots
 - ▶ Suppression des mots vides...
- **Autres traitements plus coûteux**
 - ▶ Chunking
 - ▶ Analyse syntaxique des phrases
 - ▶ Reconnaissance des entités nommées du texte
 - ▶ Analyse sémantique du texte

Découpage en mots (*tokenization*)

- Le **découpage en mots** (*tokenization*) consiste à découper une séquence de caractères en des unités appelées mots (*tokens*)



- On s'appuie généralement sur les espaces et la ponctuation (la ponctuation est également supprimée)
- Chaque mot peut ensuite subir d'autres traitements linguistiques

Découpage du texte en mots – problèmes

- Il n'est pas toujours facile de savoir où couper les mots
 - ▶ Problème des apostrophes
 - ★ *l'ensemble, aujourd'hui* : un ou deux mots ?
 - ▶ Problème des tirets
 - ★ *state-of-the-art, compte-rendu* : combien de mots ?
 - ▶ Problème des espaces
 - ★ *San Francisco* : un ou deux mots ?
 - ▶ Problème des nombres
 - ★ *127.0.0.1, 15/09/2013*
- Selon les langages, le découpage en mots ainsi que les autres pré-traitements peuvent être encore plus difficiles
 - ▶ Problème des langues agglutinantes, alphabets non latin, langues avec différents sens de lecture. . .

Normalisation, accents, signes diacritiques et casse

● Normalisation

- ▶ Il y a des cas dans lesquels les mots ne sont pas exactement les mêmes mais on veut quand même les mettre en correspondance
 - ★ Par exemple, *U.S.A.*, *USA* et *U.S.*
- ▶ Pour cela, nous définissons des **classes d'équivalence**
 - ★ *U.S.A.*, *USA* et *U.S.* appartiendront à la même classe d'équivalence

● Accents et signes diacritiques

- ▶ On peut supprimer les accents et les signes diacritiques dans la plupart des cas
 - ★ Peu d'impact en anglais : *cliché* et *cliche*
 - ★ Plus d'impact en espagnol, par exemple : *peña* et *pena*
 - ★ Substitution de lettres en allemand (traitement du umlaut) : *Universität* → *Universitaet*

● Traitement de la casse

- ▶ On peut mettre tous les mots en minuscule

Lemmatisation et étiquetage grammatical

- La **lemmatisation** réduit les formes fléchies des mots à leur forme de base (lemme)
 - ▶ *suis, es, est, sommes, êtes, sont* → *être*
 - ▶ *nous sommes arrivés* → *nous être arriver*
- Pour faire la lemmatisation, il faut tout d'abord identifier la forme de base du dictionnaire correspondant à chaque mot.
Cela nécessite un **étiqueteur grammatical (ou morpho-syntaxique)** qui dépend de la langue : son rôle est d'identifier la classe grammaticale de chaque mot
 - ▶ *je souris* → *<je, je, PRONOM> <souris, sourire, VERBE>*
 - ▶ *la souris* → *<la, le, DETERMINANT> <souris, souris, NOM>*

Racinisation (*stemming*)

- La **racinisation** (*stemming*) réduit les mots à leur « racine » en coupant généralement la fin des mots ; elle dépend du langage
 - ▶ *automate, automatisme, automatique* → *automat*
- **Algorithme de Porter** [Por80]
 - ▶ C'est l'algorithme le plus utilisé pour l'anglais (mais il existe pour d'autres langues)
 - ▶ Il utilise des conventions et cinq phases de réductions (ensembles de commandes)
 - ★ *Parmi les commandes, utiliser celle qui s'applique au plus long suffixe (exemple de convention)*
 - ★ *Supprimer le « ement » final s'il reste plus d'un caractère après réduction (exemple de commande)*
- D'autres algorithmes existent, comme l'**algorithme de Krovetz** [Kro93] qui utilise un dictionnaire et qui ne coupe que les mots n'y apparaissant pas

Suppression des mots vides (*stop list*)

- Un **mot vide** (*stop word*) est un mot qui n'apporte pas d'information utile
- Une **liste de mots vides** (*stop list*) est une liste de mots vides à supprimer dans les documents
 - ▶ **Prépositions** : *de, sur...*
 - ▶ **Déterminants** : *le, une...*
 - ▶ **Pronoms** : *je, vous...*
 - ▶ **Quelques adverbes et adjectifs** : *déjà, plusieurs...*
 - ▶ **Quelques noms et verbes** : *faire, mois...*
- Il existe plusieurs listes de mots vides standards
 - ▶ La liste SMART contient 571 mots anglais
- Généralement, la suppression des mots vides améliore l'efficacité d'applications de la fouille de textes

Plan du cours

- 1 Pré-traitements du texte
- 2 Représentation vectorielle d'une collection de documents**
- 3 Similarité syntaxique
- 4 Implémentation de la représentation vectorielle d'une collection de documents
- 5 Références

Représentation vectorielle d'une collection de documents

- A l'issue d'éventuels pré-traitements, **chaque document** est représenté par un **sac de mots** (ou plutôt un sac de termes) : on ne considère pas l'ordre des mots dans le document
 - ▶ Un **terme** est un mot normalisé par les pré-traitements vus précédemment
 - ▶ Par exemple, *Jean est plus rapide que Marie* et *Marie est plus rapide que Jean* sont représentés par le même sac de mots
- Une **collection de documents** est représentée par une matrice $n \times m$, avec n le nombre de termes de la collection et m le nombre de documents
 - ▶ Chaque document est ainsi représenté par un vecteur de n composantes et chaque composante correspond au poids du terme, dans le document et par rapport à la collection
 - On veut donner un poids plus faible aux termes communs et un poids plus élevé aux termes spécifiques (qui sont les plus intéressants)

Fréquence des termes : tf

- La **fréquence d'un terme** t dans un document d correspond au nombre d'occurrences de ce terme dans le document :

$$tf(t, d) = freq(t, d)$$

- Mais cette fréquence seule n'est pas suffisante
 - ▶ Un terme apparaissant beaucoup de fois dans beaucoup de documents est moins intéressant qu'un terme apparaissant moins de fois dans peu de documents
- Nous allons pondérer la fréquence des termes par leur fréquence dans la collection de documents

Fréquence des documents : *idf*

- La **fréquence de document** du terme t correspond au nombre de documents dans lequel le terme apparaît :

$$df(t) = \sum_d \delta(t, d)$$

- ▶ $\delta(t, d) = 1$ si le terme t apparaît dans le document d et $\delta(t, d) = 0$ sinon
- $df(t)$ est une mesure inverse de l'informativité du terme car des termes fréquents sont moins informatifs que des termes rares

- On définit alors la **fréquence inverse de document** d'un terme t dans une collection de N documents :

$$idf(t) = \log_{10}\left(\frac{N}{df(t)}\right)$$

- $idf(t)$ est bien une mesure de l'informativité du terme : plus l'*idf* est élevé et plus le terme est intéressant

Pondération des termes dans les documents : *tf.idf*

- La **pondération** d'un terme t dans un document d est alors le produit des deux fréquences précédentes :

$$poids(t, d) = tf(t, d) \times idf(t)$$

- La **pondération** *tf.idf* est une des pondérations les plus couramment utilisées en recherche d'information et en fouille de textes

Exemple de calcul du *tf.idf*

Une collection contient 10 documents.

Dans un document d , 3 termes apparaissent de la façon suivante :

Terme	<i>tf</i>	<i>df</i>	<i>idf</i>	<i>tf.idf</i>
<i>voiture</i>	10	2		
<i>vélo</i>	5	2		
<i>véhicule</i>	20	8		

Complétez le tableau en calculant l'*idf* et le *tf.idf* de chacun des termes du document d

Pondération des termes dans les documents : *tf.idf*

- La **pondération** d'un terme t dans un document d est alors le produit des deux fréquences précédentes :

$$poids(t, d) = tf(t, d) \times idf(t)$$

- La **pondération** *tf.idf* est une des pondérations les plus couramment utilisées en recherche d'information et en fouille de textes

Exemple de calcul du *tf.idf*

Une collection contient 10 documents.

Dans un document d , 3 termes apparaissent de la façon suivante :

Terme	<i>tf</i>	<i>df</i>	<i>idf</i>	<i>tf.idf</i>
<i>voiture</i>	10	2	$\log_{10}(\frac{10}{2}) = 0,7$	$10 \times 0,7 = 7$
<i>vélo</i>	5	2	0,7	3,5
<i>véhicule</i>	20	8	0,1	2

Complétez le tableau en calculant l'*idf* et le *tf.idf* de chacun des termes du document d

Quelques schémas *tf.idf* courants

- Il existe d'autres manières couramment utilisées pour calculer la fréquence d'un terme :
 - ▶ $tf(t, d) = \log_{10}(freq(t, d))$
 - ▶ $tf(t, d) = \log_{10}(freq(t, d)) + 1$
 - ▶ $tf(t, d) = \log_{10}(freq(t, d) + 1)$
 - Ces nouvelles fréquences sont ensuite utilisées dans la pondération *tf.idf* définie précédemment
- D'autres variantes sont données dans le chapitre 6 de [MRS08]

Normalisation des pondérations

- Afin de prendre en compte la **taille des documents** et de ne pas donner un poids trop important aux documents plus longs, on peut normaliser la pondération des termes :

$$poids_{norm}(t, d) = \frac{tf(t, d) \times idf(t)}{\sqrt{\sum_{k=1}^t (tf(k, d))^2 \times (idf(k))^2}}$$

- La normalisation force toutes les valeurs des pondérations normalisées à être dans un intervalle, généralement entre 0 et 1 inclus

Plan du cours

- 1 Pré-traitements du texte
- 2 Représentation vectorielle d'une collection de documents
- 3 Similarité syntaxique**
- 4 Implémentation de la représentation vectorielle d'une collection de documents
- 5 Références

Similarité entre documents

- Les documents d'une collection sont ainsi représentés dans un **espace vectoriel** composé de tous les termes (les termes sont les axes de cet espace) [SWY75]
- Dans les vecteurs, chaque élément (d_i) représente le poids du terme t_i correspondant (généralement calculé par le *tf.idf*) :
 - ▶ **Espace vectoriel** : (t_1, t_2, \dots, t_n)
 - ▶ **Document D_i** : $(d_{i1}, d_{i2}, \dots, d_{in})$

Similarité entre documents

- Les documents d'une collection sont ainsi représentés dans un **espace vectoriel** composé de tous les termes (les termes sont les axes de cet espace) [SWY75]
- Dans les vecteurs, chaque élément (d_i) représente le poids du terme t_i correspondant (généralement calculé par le *tf.idf*) :
 - ▶ **Espace vectoriel** : (t_1, t_2, \dots, t_n)
 - ▶ **Document D_i** : $(d_{i1}, d_{i2}, \dots, d_{in})$
- Le **score de mise en correspondance** entre un document D et chaque document D_i de la collection correspond à une mesure de similarité entre les 2 vecteurs :
$$\text{score}(D, d_i) = \text{sim}(D, D_i)$$
- Les documents sont **ordonnés** selon cette mesure de similarité

Similarité entre documents

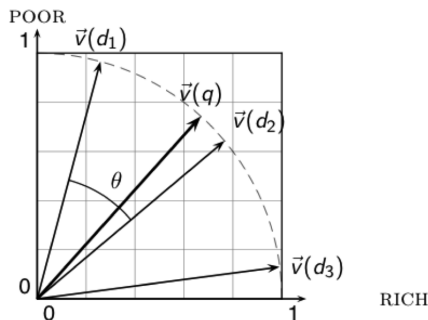
- Les documents d'une collection sont ainsi représentés dans un **espace vectoriel** composé de tous les termes (les termes sont les axes de cet espace) [SWY75]
- Dans les vecteurs, chaque élément (d_i) représente le poids du terme t_i correspondant (généralement calculé par le *tf.idf*) :
 - ▶ **Espace vectoriel** : (t_1, t_2, \dots, t_n)
 - ▶ **Document** D_i : $(d_{i1}, d_{i2}, \dots, d_{in})$
- Le **score de mise en correspondance** entre un document D et chaque document D_i de la collection correspond à une mesure de similarité entre les 2 vecteurs :

$$\text{score}(D, d_i) = \text{sim}(D, D_i)$$

- Les documents sont **ordonnés** selon cette mesure de similarité
- **Remarques**
 - ▶ Le nombre de **dimensions de l'espace vectoriel** (c'est-à-dire le nombre de termes considéré) est généralement très élevé ; dans le cas de la recherche sur le Web, on a plusieurs dizaines de millions de dimensions !
 - ▶ Les vecteurs sont relativement « **creux** » : la plupart de leurs éléments sont égaux à 0

Calcul de la similarité entre documents

- La **distance euclidienne** peut être grande pour des vecteurs n'ayant pas la même taille...
- Elle n'est donc pas adaptée pour calculer la similarité entre 2 vecteurs
- En revanche, **l'angle** entre les vecteurs de deux documents peut être utilisé pour mesurer la similarité entre les documents



Similarité cosinus (1)

- Comme tous les vecteurs contiennent des valeurs positives, ils sont tous dans le même quadrant : les angles entre les vecteurs sont donc compris entre 0° et 90°

Similarité cosinus (1)

- Comme tous les vecteurs contiennent des valeurs positives, ils sont tous dans le même quadrant : les angles entre les vecteurs sont donc compris entre 0° et 90°
- On peut ainsi utiliser le cosinus des angles ; il est plus facile à calculer que l'angle lui-même
 - ▶ Le cosinus est une fonction monotone décroissante en fonction des angles dans l'intervalle $[0, 90]$
 - ★ Plus l'angle est grand, plus le cosinus est petit
 - ★ Plus l'angle est petit, plus le cosinus est grand
 - Trier les documents selon les angles croissants avec le document considéré revient à les trier selon les cosinus décroissants des angles avec le document considéré

Similarité cosinus (2)

- Le score de mise en correspondance entre un document D et un document D_i formant un angle Θ dans l'espace vectoriel est alors :

$$\text{score}(D, D_i) = \cos(\Theta)$$

$$= \frac{\vec{d} \cdot \vec{d}_i}{|\vec{d}| \cdot |\vec{d}_i|}$$

$$= \frac{\sum_{j=1}^n d_j \times d_{ij}}{\sqrt{\sum_{j=1}^n d_j^2} \times \sqrt{\sum_{j=1}^n d_{ij}^2}}$$

- Les vecteurs \vec{d} et \vec{d}_i sont constitués des poids *tf.idf* des termes t_j de l'espace vectoriel
- \cdot représente le produit scalaire entre 2 vecteurs
- Le dénominateur correspond à une normalisation du produit scalaire par rapport à la taille des vecteurs (si les vecteurs ont déjà la même taille, la normalisation n'est pas nécessaire)

Discussions sur le modèle vectoriel

- Avantages

- ▶ Fondements mathématiques (géométrie)
- ▶ Similarité calculée possiblement sur différents éléments
- ▶ Termes pondérés selon leur importance dans les documents
- ▶ Bonnes performances dans les expérimentations en recherche d'information (calcul de similarité entre le vecteur d'une requête et ceux des documents d'une collection)

Discussions sur le modèle vectoriel

- Avantages

- ▶ Fondements mathématiques (géométrie)
- ▶ Similarité calculée possiblement sur différents éléments
- ▶ Termes pondérés selon leur importance dans les documents
- ▶ Bonnes performances dans les expérimentations en recherche d'information (calcul de similarité entre le vecteur d'une requête et ceux des documents d'une collection)

- Inconvénients

- ▶ Pas de relations pouvant être définies entre les termes → un terme avec un poids fort peut dominer la recherche
- ▶ Hypothèse d'indépendance faite sur les termes, qui ne reflète pas forcément la réalité (c'est également le cas dans la plupart des modèles classiques)

Exercice : utilisation du modèle vectoriel

Soit la collection de documents suivante :

D_1 *shipment of phones arrived in a truck*

D_2 *shipment of phones damaged in a fire*

D_3 *delivery of silver arrived in a silver truck*

Questions

- 1 Construisez les vecteurs des documents de la collection (après lemmatisation des termes et suppression des mots vides)
- 2 Calculez la similarité cosinus entre chaque couple de documents
- 3 Quels sont les deux documents les plus similaires, selon le modèle vectoriel, en utilisant la similarité cosinus ?

Plan du cours

- 1 Pré-traitements du texte
- 2 Représentation vectorielle d'une collection de documents
- 3 Similarité syntaxique
- 4 Implémentation de la représentation vectorielle d'une collection de documents**
- 5 Références

Représentation du résultat de l'indexation

- A l'issue du processus d'indexation de la collection de documents, chaque document est représenté par un ensemble de termes pondérés :

$$d_i \mapsto \{(t_1, w_1), (t_2, w_2) \dots\}$$

- Exemple

- ▶ $d_1 \mapsto \{(\text{ordinateur}, 0.1), (\text{architecture}, 0.4) \dots\}$
- ▶ $d_2 \mapsto \{(\text{ordinateur}, 0.3), (\text{souris}, 0.6) \dots\}$

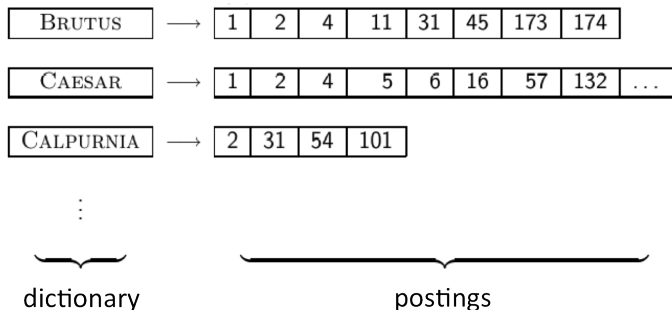
Comment représenter de manière efficace la collection de documents indexés ?

Représentation naïve : matrice d'incidence

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

- Une entrée dans la matrice est à 1 si le terme apparaît dans le document et à 0 sinon
 - ▶ *Calpurnia* apparaît dans *Julius Caesar*
 - ▶ *Calpurnia* n'apparaît pas dans *The Tempest*
- **Inconvénient** : si la matrice contient beaucoup de 0, beaucoup de place mémoire sera utilisée pour indexer peu d'information

Représentation par un index inversé



- Pour chaque terme t de la collection de documents, on stocke la liste des identifiants des documents qui contiennent t
 - ▶ Le **dictionnaire** (*dictionary*) correspond à l'ensemble des termes de la collection
 - ▶ La liste **postings** d'un terme contient la liste des identifiants des documents contenant le terme

Construction de l'index inversé

- Une approche simple pour construire l'index inversé de la collection consiste à suivre les étapes suivantes :
 - 1 Faire une première passe sur la collection pour collecter toutes les paires terme-idDoc (identifiant de document)
 - 2 Ordonner les paires par rapport aux termes puis par rapport aux identifiants des documents
 - 3 Pour chaque terme, organiser les identifiants de documents en une liste chaînée et calculer des statistiques telles que les fréquences de termes (*tf*) et de documents (*idf*)

Exemple de construction d'index inversé[MRS08] (1)

Soit une collection composée des deux documents suivants :

- ① *Doc₁* : I did enact Julius Caesar I was killed I' the Capitol Brutus killed me.
- ② *Doc₂* : So let it be with Caesar the noble Brutus hath told you Caesar was ambitious.

Construire l'index inversé correspondant à la collection donnée

Exemple de construction d'index inversé [MRS08] (2)

Doc 1. i did enact julius caesar i was
killed i' the capitol brutus killed me
Doc 2. so let it be with caesar the
noble brutus hath told you caesar was
ambitious



term	docID	term	docID
i	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
i	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	i	1
killed	1	i	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2
brutus	2	the	1
hath	2	the	2
told	2	told	2
you	2	you	2
caesar	2	was	1
was	2	was	2
ambitious	2	with	2



term doc. freq. → postings lists

Exercice : construction d'un index inversé (1)

Soit les 6 documents suivants :

- **Doc. 1** *Le robot américain Curiosity a détecté près de la surface de Mars des émanations régulières de méthane*
- **Doc. 2** *Les scientifiques de la mission ont indiqué que l'origine de ce gaz n'avaient pu être déterminée*
- **Doc. 3** *Les chercheurs ont constaté que les émissions régulières de méthane dans le cratère de Gale étaient moitié moins importantes que ce qu'ils pensaient trouver*
- **Doc. 4** *Ces émanations très faibles proviennent de la décomposition de la poussière du sol sous l'effet de la lumière du soleil et des matériaux organiques transportés par les météorites*
- **Doc. 5** *Ils ont aussi découvert que les niveaux de méthane dans le cratère près de l'endroit où se trouvait Curiosity connaissaient des pics plus élevés*
- **Doc. 6** *Ces résultats suggèrent que le méthane est produit occasionnellement ou s'échappe du sol près du cratère de Gale*

Exercice : construction d'un index inversé (2)

Questions

- 1 Donnez les documents obtenus après les pré-traitements suivants :
 - 1 Suppression des majuscules
 - 2 Lemmatisation
 - 3 Suppression des mots outils à partir de la liste suivante : {aussi, avoir, ce, dans, de, dont, et, être, il, indiquer, le, moins, moitié, ne, ou, où, par, penser, pouvoir, plus, que, se, sous, très}
- 2 Dessinez l'index inversé de cette collection, en indiquant le df de chaque terme ainsi que son tf dans chaque document ; l'index inversé sera construit à partir des documents pré-traités
- 3 Calculez la pondération $tf.idf$ de chaque terme du document 1

Exercice : construction d'un index inversé (3)

Réponses

1 Donnez les documents obtenus après les pré-traitements

- ▶ **Doc. 1** *robot américain curiosity détecter près surface mars émanation régulier méthane*
- ▶ **Doc. 2** *scientifique mission origine gaz déterminer*
- ▶ **Doc. 3** *chercheur constater émission régulier méthane cratère gale important trouver*
- ▶ **Doc. 4** *émanation faible provenir décomposition poussière sol effet lumière soleil matériau organique transporter météorite*
- ▶ **Doc. 5** *découvrir niveau méthane cratère près endroit trouver curiosity connaître pic élevé*
- ▶ **Doc. 6** *résultat suggérer méthane produire occasionnellement échapper sol près cratère gale*

Exercice : construction d'un index inversé (4)

Réponses

2 Dessinez l'index inversé de cette collection

- ▶ Comme tous les tf sont à 1, on ne les met pas dans l'index inversé
- Comme l'index inversé comporte 45 termes, on ne dessine ici que le début de l'index, en ne prenant en compte que les termes des 2 premiers documents mais en calculant leur fréquence sur toute la collection

américain	1	→	1
curiosity	2	→	1 5
détecter	1	→	2
déterminer	1	→	1
émanation	2	→	1 4
gaz	1	→	2
mars	1	→	1
méthane	4	→	1 3 5 6
mission	1	→	4
origine	1	→	2
près	3	→	1 5 6
régulier	2	→	1 3
robot	1	→	1
scientifique	1	→	2
surface	1	→	1

Exercice : construction d'un index inversé (5)

Réponses

3 Calculez la pondération *tf.idf* de chaque terme du document 1

- ▶ $tf.idf(robot) = tf(robot) \times idf(robot) = 1 \times \log(\frac{6}{1}) = 0,78$
- ▶ $tf.idf(américain) = tf(américain) \times idf(américain) = 1 \times \log(\frac{6}{1}) = 0,78$
- ▶ $tf.idf(curiosity) = tf(curiosity) \times idf(curiosity) = 1 \times \log(\frac{6}{2}) = 0,48$
- ▶ $tf.idf(détecter) = tf(détecter) \times idf(détecter) = 1 \times \log(\frac{6}{1}) = 0,78$
- ▶ $tf.idf(près) = tf(près) \times idf(près) = 1 \times \log(\frac{6}{3}) = 0,30$
- ▶ $tf.idf(surface) = tf(surface) \times idf(surface) = 1 \times \log(\frac{6}{1}) = 0,78$
- ▶ $tf.idf(mars) = tf(mars) \times idf(mars) = 1 \times \log(\frac{6}{1}) = 0,78$
- ▶ $tf.idf(émanation) = tf(émanation) \times idf(émanation) = 1 \times \log(\frac{6}{2}) = 0,48$
- ▶ $tf.idf(régulier) = tf(régulier) \times idf(régulier) = 1 \times \log(\frac{6}{2}) = 0,48$
- ▶ $tf.idf(méthane) = tf(méthane) \times idf(méthane) = 1 \times \log(\frac{6}{4}) = 0,18$

Construction de l'index inversé et problèmes mémoire

- Pour construire l'index, on traite un document à la fois.
 - ▶ La liste des documents, pour chaque terme, n'est donc complète qu'à la fin du processus
 - ▶ Il faut stocker tous les résultats intermédiaires en mémoire
 - ▶ Cela peut nécessiter beaucoup de mémoire pour de grandes collections
- Une solution pour utiliser moins de mémoire : **l'indexation une passe en mémoire** (*Single-Pass In-Memory Indexing*)
 - ▶ On découpe tout d'abord la collection en blocs qui tiennent en mémoire principale
 - ▶ On génère des dictionnaires séparés pour chaque bloc
 - ▶ On génère un index inversé pour chaque bloc
 - ▶ On fusionne enfin ces index inversés séparés en un seul grand index inversé
- D'autres solutions sont présentées dans le chapitre 4 de [MRS08]

Implémentation de l'index inversé

- Pour stocker le dictionnaire, il faut mémoriser deux informations pour chaque terme :
 - ▶ La fréquence document
 - ▶ La liste des identifiants des documents
 - On peut stocker ce couple d'informations dans une entrée de taille fixe et stocker ces entrées dans un [tableau](#)
 - Est-ce qu'on peut utiliser une structure de données permettant une recherche plus rapide des termes ?
- Plus de détails dans le chapitre 3 de [MRS08]

Stockage du dictionnaire dans une table de hachage

- Chaque terme du vocabulaire est transformé en un entier par la fonction de hachage
- Avantages
 - ▶ La recherche dans une table de hachage est plus rapide que dans un arbre
- Inconvénients
 - ▶ Pas de recherche de préfixe possible (par exemple, pour chercher tous les termes commençant par *automat*)
 - ▶ Il faut recalculer la fonction de hachage si le vocabulaire change

Stockage du dictionnaire dans un arbre

- L'utilisation d'arbres permet de résoudre le problème de la recherche de préfixes
- Les arbres les plus simples sont les **arbres binaires**
 - ▶ Seuls les arbres binaires équilibrés permettent une recherche efficace
 - ▶ Rééquilibrer des arbres est coûteux (quand on ajoute des mots au vocabulaire)
- On peut utiliser plutôt des **B-arbres**
 - ▶ Chaque nœud interne d'un B-arbre a un nombre de fils dans un intervalle donné (par exemple, entre 2 et 4)

Compression de l'index inversé

- Deux éléments de l'index inversé peuvent être compressés :
 - ▶ Le dictionnaire
 - ★ Motivation principale : faire tenir le dictionnaire en mémoire principale
 - ▶ Les listes d'identifiants de documents
 - ★ Motivation principale : réduire l'espace disque nécessaire, diminuer le temps nécessaire pour lire les listes à partir du disque dur
 - Présentation de quelques mécanismes de compression
- Plus de détails dans le chapitre 5 de [MRS08]

Compression du dictionnaire

- Étant donnée une collection de documents de taille T (nombre de termes), pouvons-nous estimer la taille du vocabulaire M ?

- ▶ Loi de Heap :

$$M = kT^b$$

- ★ k et b sont des paramètres, dont les valeurs varient généralement dans les intervalles suivants : $30 \leq k \leq 100$ et $b \approx 0,5$
- ★ La taille du vocabulaire augmente avec le nombre de documents
- ★ Le vocabulaire d'une grande collection sera grand lui aussi

- ▶ La loi de Heap a été montrée empiriquement pour de grandes collections

- Une des solutions pour compresser le vocabulaire consiste à utiliser des B-arbres préfixe

- ▶ Seules les feuilles de ces arbres contiennent les termes (+ les fréquences documents et les pointeurs vers les listes *postings*)
- ▶ Chaque terme n'est plus une entrée de taille fixe comme précédemment

Compression des listes d'identifiants de documents

- Les listes des identifiants des documents prennent beaucoup plus de place que le vocabulaire (facteur au moins 10)
- L'objectif de la compression des listes est de compresser les **identifiants des documents**
- Une des solutions pour compresser les identifiants des documents consiste à mémoriser les **trous** entre documents
 - ▶ Chaque liste est ordonnée par ordre croissant des identifiants
 - ★ $[information, 8 : \langle 3, 8, 12, 19, 22, 23, 26, 33 \rangle]$; $[apre, 2 : \langle 4, 32 \rangle]$
 - ▶ Il suffit de stocker les trous au lieu des identifiants de documents

Compression des listes d'identifiants de documents

- Les listes des identifiants des documents prennent beaucoup plus de place que le vocabulaire (facteur au moins 10)
- L'objectif de la compression des listes est de compresser les **identifiants des documents**
- Une des solutions pour compresser les identifiants des documents consiste à mémoriser les **trous** entre documents
 - ▶ Chaque liste est ordonnée par ordre croissant des identifiants
 - ★ $[information, 8 : \langle 3, 8, 12, 19, 22, 23, 26, 33 \rangle]$; $[apre, 2 : \langle 4, 32 \rangle]$
 - ▶ Il suffit de stocker les trous au lieu des identifiants de documents
 - ★ $[information, 8 : \langle 3, 5, 4, 7, 3, 1, 3, 7 \rangle]$; $[apre, 2 : \langle 4, 28 \rangle]$





Compression des listes d'identifiants de documents

- Les listes des identifiants des documents prennent beaucoup plus de place que le vocabulaire (facteur au moins 10)
- L'objectif de la compression des listes est de compresser les **identifiants des documents**
- Une des solutions pour compresser les identifiants des documents consiste à mémoriser les **trous** entre documents
 - ▶ Chaque liste est ordonnée par ordre croissant des identifiants
 - ★ [*information*, 8 : $\langle 3, 8, 12, 19, 22, 23, 26, 33 \rangle$] ; [*apre*, 2 : $\langle 4, 32 \rangle$]
 - ▶ Il suffit de stocker les trous au lieu des identifiants de documents
 - ★ [*information*, 8 : $\langle 3, 5, 4, 7, 3, 1, 3, 7 \rangle$] ; [*apre*, 2 : $\langle 4, 28 \rangle$]
 - ▶ On peut utiliser un **encodage variable** pour stocker les trous
 - ★ **Termes fréquents** : les trous seront petits donc moins de bits utilisés pour encoder les trous
 - ★ **Termes rares** : les trous seront grands donc plus de bits utilisés pour encoder les trous

Plan du cours

- 1 Pré-traitements du texte
- 2 Représentation vectorielle d'une collection de documents
- 3 Similarité syntaxique
- 4 Implémentation de la représentation vectorielle d'une collection de documents
- 5 **Références**

Références I

-  R. Krovetz, *Viewing Morphology as an Inference Process*, Proc. of ACM SIGIR Conference, 1993, pp. 191–202.
-  Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press, 2008.
-  M.F. Porter, *An Algorithm for Suffix Stripping*, Program **14** (1980), no. 3, 130–137.
-  G. Salton, A. Wong, and C.S. Yang, *A vector space model for automatic indexing*, Commun. ACM **18** (1975), no. 11, 613–620.