Données textuelles

Dan VODISLAV

CY Cergy Paris Université
Master Informatique M1
Cours BDA

Plan

- Traitement des données textuelles
- Indexation du texte
- Requêtes mots-clés: modèles booléen et vecteur

Données textuelles

- Problème: comment gérer les données de type texte, afin de répondre efficacement à des requêtes mots-clés?
- Exemple (en anglais): ensemble de 7 documents

  d1: The jaguar is a New World mammal of the Felidae family.

  d2: Jaguar has designed four new engines.

  d3: For Jaguar, Atari was keen to use a 68K family device.

  d4: The Jacksonville Jaguars are a professional US football team.

  d5: Mac OS X Jaguar is available at a price of US $199 for Apple's new "family pack".

  d6: One such ruling family to incorporate the jaguar into their name is Jaguar Paw.

  d7: It is a big cat.

Pré-traitement du texte

- Étapes de pré-traitement dépendantes de l'application et de la langue des documents
  - Découpage en mots ("tokenization")
  - Lemmatisation ("stemming")
  - Élimination des mots fréquents ("stop words")

Découpage en mots

- Le mot est l'unité de recherche
- Le découpage en mots n'est pas si simple que ça!
  - Dans certaines langues, les mots ne sont pas séparés par des espaces (chinois, japonais)
  - Traitement spécial pour les acronymes, les abréviations, les élisions, les nombres, les URLs, les adresses mail, les unités de mesure, etc.
  - Mots composés (hostname, host-name, host name): les garder ensemble ou les séparer en mots?
    - Parfois très compliqué (Allemand): analyse lexicale et linguistique
- Dans cette étape: on enlève la ponctuation et on passe en minuscules

Exemple découpage en mots

d1: the$_1$ jaguar$_2$ is$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$

d2: jaguar$_1$ has$_2$ designed$_3$ four$_4$ new$_5$ engines$_6$

d3: for$_1$ jaguar$_2$ atari$_3$ was$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$

d4: the$_1$ jacksonville$_2$ jaguars$_3$ are$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$

d5: mac$_1$ os$_2$ x$_3$ jaguar$_4$ is$_5$ available$_6$ at$_7$ a$_8$ price$_9$ of$_{10}$ us$_{11}$ \$199$_{12}$ for$_{13}$ apple's$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

d6: one$_1$ such$_2$ ruling$_3$ family$_4$ to$_5$ incorporate$_6$ the$_7$ jaguar$_8$ into$_9$ their$_{10}$ name$_{11}$ is$_{12}$ jaguar$_{13}$ paw$_{14}$

d7: it$_1$ is$_2$ a$_3$ big$_4$ cat$_5$

Lemmatisation

- Considérer la racine (lemme) des mots, afin de "fusionner" tous les mots de la même famille
  - Pas toutes les applications souhaitent cela!
  - En cherchant *enfant* on retrouve *enfance* ou *enfantin*
  - Différents degrés de lemmatisation
  - Il est possible de construire plusieurs index, avec des techniques de lemmatisation différentes
- Types de lemmatisation
  - Morphologique
  - Lexicale
  - Phonétique

Types de lemmatisation

- Morphologique: enlever les terminaisons
  - Pluriel, genre, temps, mode, …
  - Pas toujours facile: "*Les poules du couvent couvent*."
  - En anglais c'est plus facile (terminaisons régulières, avec quelques exceptions bien répertoriées), mais il reste des ambiguïtés
- Lexicale
  - Termes d'une même famille
  - En anglais: algorithme de Porter, basé sur des critères morphologiques (!)
    - Pas toujours de bons résultats (ex. university, universal → univers)
  - Couplage avec des dictionnaires pour regrouper aussi les synonymes
- Phonétique
  - Regroupe des mots qui se prononcent pareil (ou presque pareil)
  - Objectif: traiter les fautes de frappe/orthographe
  - Assez grossier

Exemple lemmatisation

d1: the$_1$ jaguar$_2$ be$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$

d2: jaguar$_1$ have$_2$ design$_3$ four$_4$ new$_5$ engine$_6$

d3: for$_1$ jaguar$_2$ atari$_3$ be$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$

d4: the$_1$ jacksonville$_2$ jaguar$_3$ be$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$

d5: mac$_1$ os$_2$ x$_3$ jaguar$_4$ be$_5$ available$_6$ at$_7$ a$_8$ price$_9$ of$_{10}$ us$_{11}$ 199_{12}$ for$_{13}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

d6: one$_1$ such$_2$ rule$_3$ family$_4$ to$_5$ incorporate$_6$ the$_7$ jaguar$_8$ into$_9$ their$_{10}$ name$_{11}$ be$_{12}$ jaguar$_{13}$ paw$_{14}$

d7: it$_1$ be$_2$ a$_3$ big$_4$ cat$_5$

Élimination des mots fréquents

- Les mots fréquents se trouvent partout et ne distinguent pas les documents les uns par rapport aux autres
  - Leur élimination ne modifie pas significativement les résultats des requêtes
- Avantage: le nombre de mots du texte et la taille de l'index diminuent sensiblement
- Mots fréquents:
  - Articles: le, la, un, une, des, …
  - Verbes fonctionnels: être, avoir, faire, …
  - Conjonctions: et, ou, que, …
  - etc.

Exemple élimination mots fréquents

d1: jaguar$_2$ new$_5$ world$_6$ mammal$_7$ felidae$_{10}$ family$_{11}$

d2: jaguar$_1$ design$_3$ four$_4$ new$_5$ engine$_6$

d3: jaguar$_2$ atari$_3$ keen$_5$ 68k$_9$ family$_{10}$ device$_{11}$

d4: jacksonville$_2$ jaguar$_3$ professional$_6$ us$_7$ football$_8$ team$_9$

d5: mac$_1$ os$_2$ x$_3$ jaguar$_4$ available$_6$ price$_9$ of$_{10}$ us$_{11}$ \$199$_{12}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$

d6: one$_1$ such$_2$ rule$_3$ family$_4$ to$_5$ incorporate$_6$ jaguar$_8$ their$_{10}$ name$_{11}$ jaguar$_{13}$ paw$_{14}$

d7: big$_4$ cat$_5$

Index inversé

- Index de tous les mots (termes) retenus après le pré-traitement, avec pour chacun la liste de documents où il apparaît
- Stockage
  - Petite échelle: sur disque, avec des mécanismes de mémoire virtuelle
  - Grande échelle: *cluster de machines*, avec hachage pour trouver la machine qui stocke l'entrée pour un mot donné
- Mise à jour
  - Coûteuse, réalisée généralement par lots (batch) et non pas individuellement

Exemple index inversé

family: *d1, d3, d5, d6*
football: *d4*
jaguar: *d1, d2, d3, d4, d5, d6*
new: *d1, d2, d5*
rule: *d6*
us: *d4, d5*
world: *d1*
…

Positions des mots dans le document

- Utiles pour les requêtes de type "phrase" (plusieurs mots qui doivent se suivre dans l'ordre) ou des opérateurs comme NEAR
- Rajoutées à l'index pour chaque document
  - Un document peut avoir plusieurs occurrences du mot

family: *d1/11, d3/10, d5/16, d6/4*
football: *d4/8*
jaguar: *d1/2, d2/1, d3/2, d4/3, d5/4, d6/8+13*
new: *d1/5, d2/5, d5/15*
rule: *d6/3*
us: *d4/7, d5/11*
world: *d1/6*
…

Requêtes booléennes

- Trouver les documents qui contiennent les mots de la requête
- Un seul mot: liste de documents dans l'index inversé
- Plusieurs mots

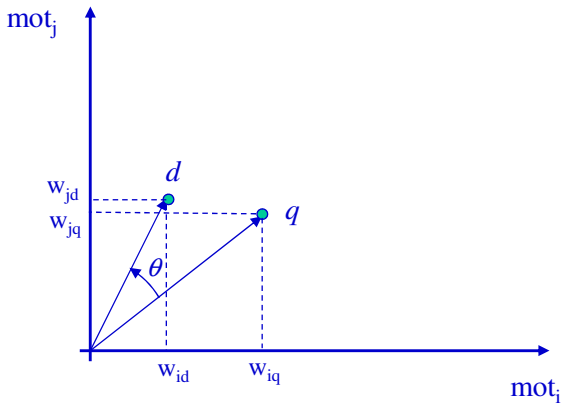
  Ex. *(jaguar AND new AND NOT family) OR cat*
  - On cherche les listes de documents pour les mots de la requête
  - Opérations sur les listes, suivant l'opérateur logique:
    - *AND:* intersection
    - *OR*: union
    - *AND NOT*: différence
- Requêtes impliquant la position dans le texte
  - Pareil, mais on filtre par rapport à la position des mots

Modèle vecteur

- Réponse: documents *similaires* à une requête
- Méthode
  - Dans chaque document une partie des mots sont indexés
    - *n*: nombre total de mots indexés dans le système
  - Un document *d* devient un point (vecteur) dans un espace à *n* dimensions
    - à chacun des mots (*i*) est associé un poids (une coordonnée) $1 \geq w_{id} \geq 0$
  - Requête *q* : point (vecteur): $(w_{1q},\ldots,w_{iq},\ldots,w_{nq})$
  - Réponse: les documents (points) les plus proches du point requête
- Avantage: ordonnancement des réponses par similarité décroissante
- Modèle booléen: les poids sont 0 / 1 et l'appartenance d'un document au résultat est stricte

Modèle vecteur : similarité

- Exemples de mesures de similarité
  - Distance euclidienne
  - On préfère: cosinus de l'angle entre les deux vecteurs
    - $\text{sim}(d, q) = \cos(\theta) = \sum (w_{id} * w_{iq}) / (\text{norme}(d) * \text{norme}(q))$



Calcul des poids: tf/idf

- Fréquence d'un terme (mot) dans un document
  - *tf* : « term frequency »
  - exprime l'importance du terme (i) pour caractériser le document (d)

  $\mathbf{tf_{id} = freq_{id} / max_t(freq_{td})}$
- Inverse de la fréquence d'un terme dans un corpus de documents
  - *idf* : « inverse document frequency »
  - exprime la capacité du terme (i) à filtrer parmi les documents du corpus
    - corpus de N documents, le terme i apparaît dans n_i documents

  $\mathbf{idf_i = log(N/n_i)}$
- Calcul des poids : $\mathbf{w_{id} = tf_{id} * idf_i}$
  - w_{iq} : une formule similaire (avec des variantes), en considérant q comme un document contenant les mots de la requête

Exemple calcul tf/idf

family: *d1/11, d3/10, d5/16, d6/4*
football: *d4/8*
jaguar: *d1/2, d2/1, d3/2, d4/3, d5/4, d6/8+13*
new: *d1/5, d2/5, d5/15*
rule: *d6/3*
us: *d4/7, d5/11*
world: *d1/6*
…

- $tf_{jaguar,d6} = freq_{jaguar,d6} / max_t(freq_{t,d6}) = 2/2 = 1$
  $tf_{rule,d6} = freq_{rule,d6} / max_t(freq_{t,d6}) = 1/2 = 0,5$
  $tf_{new,d6} = freq_{new,d6} / max_t(freq_{t,d6}) = 0/2 = 0$
- $idf_{jaguar} = log(N/n_{jaguar}) = log(7/6) = 0,067$
  $idf_{rule} = log(N/n_{rule}) = log(7/1) = 0,845$
  $idf_{new} = log(N/n_{new}) = log(7/3) = 0,368$
- $w_{jaguar,d6} = tf_{jaguar,d6} * idf_{jaguar} = 0,067$
  $w_{rule,d6} = tf_{rule,d6} * idf_{rule} = 0,422$
  $w_{new,d6} = 0$