CES Data Scientist

Inverted Index

16 décembre 2014

Problem

How to index Web content so as to answer (keyword-based) queries efficiently?

Context: set of text documents

d_1	The jaguar is a New World mammal of the Felidae family.
d_2	Jaguar has designed four new engines.
d_3	For Jaguar, Atari was keen to use a 68K family device.
d_4	The Jacksonville Jaguars are a professional US football team.
d_5	Mac OS X Jaguar is available at a price of US $199 for Apple's new "family pack".
d_6	One such ruling family to incorporate the jaguar into their name is Jaguar Paw.
d_7	It is a big cat.

Text Preprocessing

Initial text preprocessing steps

- Number of optional steps
- Highly depends on the application
- Highly depends on the document language (illustrated with English)

Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- Unambiguous scripts or letters: not very common!

한글
カタカナ
ދިވެހި
Għarbi
þorn

TELECOM ParisTech

Language Identification

How to find the language used in a document?

- Meta-information about the document: often not reliable!
- Unambiguous scripts or letters: not very common!

  한글
  カタカナ
  ދިވެހި
  Għarbi
  þorn

  Respectively: Korean Hangul, Japanese Katakana, Maldivian Dhivehi, Maltese, Icelandic
- Extension of this: frequent characters, or, better, frequent *k*-grams
- Use standard machine learning techniques (classifiers)

Tokenization

Principle

Separate text into tokens (words)

Not so easy!

- In some languages (Chinese, Japanese), words not separated by whitespace
- Deal consistently with acronyms, elisions, numbers, units, URLs, emails, etc.
- Compound words: *hostname*, *host-name* and *host name*. Break into two tokens or regroup them as one token? In any case, lexicon and linguistic analysis needed! Even more so in other languages as German.

Punctuation may be removed and case normalized at this point

Tokenization: Example

d_1	the$_1$ jaguar$_2$ is$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_{11}$
d_2	jaguar$_1$ has$_2$ designed$_3$ four$_4$ new$_5$ engines$_6$
d_3	for$_1$ jaguar$_2$ atari$_3$ was$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$
d_4	the$_1$ jacksonville$_2$ jaguars$_3$ are$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$
d_5	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$_{17}$
d_6	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$_{14}$
d_7	it$_1$ is$_2$ a$_3$ big$_4$ cat$_5$

Stemming

Principle

Merge different forms of the same word, or of closely related words, into a single stem

- Not in all applications!
- Useful for retrieving documents containing *geese* when searching for *goose*
- Various degrees of stemming
- Possibility of building different indexes, with different stemming

Stemming schemes (1/2)

Morphological stemming (lemmatization).

- Remove bound morphemes from words:
  - plural markers
  - gender markers
  - tense or mood inflections
  - etc.
- Can be linguistically very complex, cf:
  Les poules du couvent couvent.
  [The hens of the monastery brood.]
- In English, somewhat easy:
  - Remove final -s, -'s, -ed, -ing, -er, -est
  - Take care of semiregular forms (e.g., -y/-ies)
  - Take care of irregular forms (mouse/mice)
- But still some ambiguities: cf rose

Stemming schemes (2/2)

Lexical stemming.

- Merge lexically related terms of various parts of speech, such as *policy*, *politics*, *political* or *politician*
- For English, Porter's stemming [Porter, 1980]; stem *university* and *universal* to *univers*: not perfect!
- Possibility of coupling this with lexicons to merge (near-)synonyms

Phonetic stemming.

- Merge phonetically related words: search proper names with different spellings!
- For English, Soundex [US National Archives and Records Administration, 2007] stems *Robert* and *Rupert* to *R163*. Very coarse!

Stemming Example

d_1 the$_1$ jaguar$_2$ be$_3$ a$_4$ new$_5$ world$_6$ mammal$_7$ of$_8$ the$_9$ felidae$_{10}$ family$_1$

d_2 jaguar$_1$ have$_2$ design$_3$ four$_4$ new$_5$ engine$_6$

d_3 for$_1$ jaguar$_2$ atari$_3$ be$_4$ keen$_5$ to$_6$ use$_7$ a$_8$ 68k$_9$ family$_{10}$ device$_{11}$

d_4 the$_1$ jacksonville$_2$ jaguar$_3$ be$_4$ a$_5$ professional$_6$ us$_7$ football$_8$ team$_9$

d_5 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$_{17}$

d_6 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$_{14}$

d_7 it$_1$ be$_2$ a$_3$ big$_4$ cat$_5$

Stop Word Removal

Principle

Remove uninformative words from documents, in particular to lower the cost of storing the index

- determiners: *a*, *the*, *this*, etc.
- function verbs: *be*, *have*, *make*, etc.
- conjunctions: *that*, *and*, etc.
- etc.

Stop Word Removal Example

d_1 jaguar$_2$ new$_5$ world$_6$ mammal$_7$ felidae$_{10}$ family$_{11}$
d_2 jaguar$_1$ design$_3$ four$_4$ new$_5$ engine$_6$
d_3 jaguar$_2$ atari$_3$ keen$_5$ 68k$_9$ family$_{10}$ device$_{11}$
d_4 jacksonville$_2$ jaguar$_3$ professional$_6$ us$_7$ football$_8$ team$_9$
d_5 mac$_1$ os$_2$ x$_3$ jaguar$_4$ available$_6$ price$_9$ us$_{11}$ \$199$_{12}$ apple$_{14}$ new$_{15}$ family$_{16}$ pack$_{17}$
d_6 one$_1$ such$_2$ rule$_3$ family$_4$ incorporate$_6$ jaguar$_8$ their$_{10}$ name$_{11}$ jaguar$_{13}$ paw$_{14}$
d_7 big$_4$ cat$_5$

Structure of an inverted index

Assume D a collection of (text) documents. Create a matrix M with one row for each document, one column for each token. Initialize the cells at 0.

Create the content of M: scan D, and extract for each document d the tokens t that can be found in d (preprocessing); put 1 in $M[d][t]$

Invert M: one obtains the inverted index. Term appear as rows, with the list of document ids or *posting list*.

Problem: storage of the whole matrix is not feasible.

Inverted Index

After all preprocessing, construction of an inverted index:

- Index of all terms, with the list of documents where this term occurs
- Small scale: disk storage, with memory mapping (cf. `mmap`) techniques; secondary index for offset of each term in main index
- Large scale: distributed on a cluster of machines
- Updating the index costly, so only batch operations (not one-by-one addition of term occurrences)

Inverted Index Example

family	d_1, d_3, d_5, d_6
football	d_4
jaguar	$d_1, d_2, d_3, d_4, d_5, d_6$
new	d_1, d_2, d_5
rule	d_6
us	d_4, d_5
world	d_1
...	

Note:

- the length of an inverted (posting) list is highly variable – scanning short lists first is an important optimization.
- *entries* are homogeneous: this gives much room for compression.

Storing positions in the index

- phrase queries, NEAR operator: need to keep position information in the index
- just add it in the document list!

family	$d_1/11$, $d_3/10$, $d_5/16$, $d_6/4$
football	$d_4/8$
jaguar	$d_1/2$, $d_2/1$, $d_3/2$, $d_4/3$, $d_5/4$, $d_6/8+13$
new	$d_1/5$, $d_2/5$, $d_5/15$
rule	$d_6/3$
us	$d_4/7$, $d_5/11$
world	$d_1/6$
...	

$\Rightarrow$ so far, ok for Boolean queries: find the documents that contain a set of keywords; reject the other.

Ranked search

Boolean search does not give an accurate result because it does not take account of the relevance of a document to a query.

If the search retrieves dozen or hundreds of documents, the most relevant must be shown in top position!

Weighting terms occurrences

Relevance can be computed by giving a weight to term occurrences.

- Terms occurring frequently in a given document: more relevant.
  The *term frequency* is the number of occurrences of a term t in a document d, divided by the total number of terms in d

  $$\mathrm{tf}(t,d) = \frac{n_{t,d}}{\sum_{t'} n_{t',d}}$$

  where $n_{t',d}$ is the number of occurrences of t' in d.
- Terms occurring rarely in the document collection as a whole: more informative
  The *inverse document frequency* (idf) is obtained from the division of the total number of documents by the number of documents where t occurs, as follows:

  $$\mathrm{idf}(t) = \log \frac{|D|}{\left|\left\{d' \in D \mid n_{t,d'} > 0\right\}\right|}.$$

TF-IDF Weighting

- Some term occurrences have more weight than others:
  - Terms occurring frequently in a given document: more relevant
  - Terms occurring rarely in the document collection as a whole: more informative
- Add Term Frequency—Inverse Document Frequency weighting to occurrences;

$$\text{tfidf}(t,d) = \frac{n_{t,d}}{\sum_{t'} n_{t',d}} \cdot \log \frac{|D|}{\left|\left\{d' \in D \,\middle|\, n_{t,d'} > 0\right\}\right|}$$

$n_{t,d}$	number of occurrences of t in d
D	set of all documents

- Store documents (along with weight) in decreasing weight order in the index

TF-IDF Weighting Example

family	$d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$
football	$d_4/8/.47$
jaguar	$d_1/2/.04$, $d_2/1/.04$, $d_3/2/.04$, $d_4/3/.04$, $d_6/8+13/.04$, $d_5/4/.02$
new	$d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$
rule	$d_6/3/.28$
us	$d_4/7/.30$, $d_5/11/.15$
world	$d_1/6/.47$
...	

Answering Boolean Queries

- Single keyword query: just consult the index and return the documents in index order.
- Boolean multi-keyword query

  (*jaguar* AND *new* AND NOT *family*) OR *cat*

  Same way! Retrieve document lists from all keywords and apply adequate set operations:

AND	intersection
OR	union
AND NOT	difference

- Global score: some function of the individual weight (e.g., addition for conjunctive queries)
- Position queries: consult the index, and filter by appropriate condition

Exercise

Consider the following documents:

1. d_1 = I like to watch the sun set with my friend.
2. d_2 = The Best Places To Watch The Sunset.
3. d_3 = My friend watches the sun come up.

Construct an inverted index with tf/idf weights for terms ‘best’ and ‘sun’. What would be the ranked result of the query ‘best OR sun’?

Answering Top-k Queries

$$t_1 \text{ AND } \ldots \text{ AND } t_n$$

$$t_1 \text{ OR } \ldots \text{ OR } t_n$$

Problem

Find the top-k results (for some given k) to the query, without retrieving all documents matching it.

Notations:

$s(t, d)$ weight of t in d (e.g., tfidf)

$g(s_1, \ldots, s_n)$ monotonous function that computes the global score (e.g., addition)

Basic algorithm

First version of the top-k algorithm: the inverted file contains entries sorted on the document id. The query is

$$t_1 \text{ AND } \ldots \text{ AND } t_n$$

1. Take the first entry of each list; one obtains a tuple $T = [e_1, \ldots e_n]$;
2. Let d_1 be the minimal doc id in the entries of T: compute the global score of d_1;
3. For each entry e_i featuring d_1: advance on the inverted list L_i.

When *all* lists have been scanned: sort the documents on the global scores.

Not very efficient; cannot give the ranked result before a full scan on the lists.

Fagin's Threshold Algorithm [Fagin et al., 2001]

(entries are sorted according to score, with an additional direct index giving $s(t, d)$)

1. Let R be the empty list and $m = +\infty$.
2. For each $1 \leq i \leq n$:
   2.1 Retrieve the document $d^{(i)}$ containing term t_i that has the next largest $s(t_i, d^{(i)})$.
   2.2 Compute its global score $g_{d^{(i)}} = g(s(t_1, d^{(i)}), \ldots, s(t_n, d^{(i)}))$ by retrieving all $s(t_j, d^{(i)})$ with $j \neq i$.
   2.3 If R contains less than k documents, or if $g_{d^{(i)}}$ is greater than the minimum of the score of documents in R, add $d^{(i)}$ to R (and remove the worst element in R if it is full).
3. Let $m = g(s(t_1, d^{(1)}), s(t_2, d^{(2)}), \ldots, s(t_n, d^{(n)}))$.
4. If R contains k documents, and the minimum of the score of the documents in R is greater than or equal to m, return R.
5. Redo step 2.

The TA, by example

q = "*new* OR *family*", and $k = 3$.

family	$d_1/11/.13$, $d_3/10/.13$, $d_6/4/.08$, $d_5/16/.07$
new	$d_2/5/.24$, $d_1/5/.20$, $d_5/15/.10$
...	

Initially, $R = \varnothing$ and $\tau = +\infty$.

1. $d^{(1)}$ is the first entry in L_{family}, one finds $s(new, d_1) = .20$; the global score for d_1 is $.13 + .20 = .33$.
2. Next, $i = 2$, and one finds that the global score for d_2 is .24.
3. The algorithm quits the loop on i with $R = \langle [d_1, .33], [d_2, .24] \rangle$ and $\tau = .13 + .24 = .37$.
4. We proceed with the loop again, taking d_3 with score .13 and d_5 with score .17. $[d_5, .17]$ is added to R (at the end) and τ is now $.10 + .13 = .23$.
   A last loop concludes that the next candidate is d_6, with a global score of .08. Then we are done.

TELECOM ParisTech

Fagin's No Random Access Algorithm [Fagin et al., 2001]

(no additional direct index needed)

1. Let R be the empty list and $m = +\infty$.
2. For each document d, maintain $W(d)$ as its worst possible score, and $B(d)$ as its best possible score.
3. At the beginning, $W(d) = 0$ and $B(d) = g(s(t_1, d^(1)) \dots s(t_n, d^(n))$.
4. Then, access the next best document for each token, in a round-robin way (t_1, $t_2 \dots t_n$, then t_1 again, etc.)
5. Update the $W(d)$ and $B(d)$ lists each time, and maintain R as the list of k documents with best $W(d)$ scores (solve ties with $B(d)$), and m as the minimum value for $W(d)$ in R.
6. Stop when R contains at least k documents, and all documents outside of R verify $B(d) \leq m$.

Bibliography I

Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proc. PODS*, Santa Barbara, USA, May 2001.

Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3): 130–137, July 1980.

US National Archives and Records Administration. The Soundex indexing system. `http://www.archives.gov/genealogy/census/soundex.html`, May 2007.

Licence de droits d'usage

Contexte public } avec modifications

Par le téléchargement ou la consultation de ce document, l'utilisateur accepte la licence d'utilisation qui y est attachée, telle que détaillée dans les dispositions suivantes, et s'engage à la respecter intégralement.

La licence confère à l'utilisateur un droit d'usage sur le document consulté ou téléchargé, totalement ou en partie, dans les conditions définies ci-après et à l'exclusion expresse de toute utilisation commerciale.
Le droit d'usage défini par la licence autorise un usage à destination de tout public qui comprend :
– le droit de reproduire tout ou partie du document sur support informatique ou papier,
– le droit de diffuser tout ou partie du document au public sur support papier ou informatique, y compris par la mise à la disposition du public sur un réseau numérique,
– le droit de modifier la forme ou la présentation du document,
– le droit d'intégrer tout ou partie du document dans un document composite et de le diffuser dans ce nouveau document, à condition que :
  – L'auteur soit informé.

Les mentions relatives à la source du document et/ou à son auteur doivent être conservées dans leur intégralité.

Le droit d'usage défini par la licence est personnel et non exclusif.
Tout autre usage que ceux prévus par la licence est soumis à autorisation préalable et expresse de l'auteur : sitepedago@telecom-paristech.fr