

INF344: Données du Web

Information Retrieval



The Inverted Index Model

- Text Preprocessing
- Inverted Index
- Answering Keyword Queries
- Building inverted files
- Spelling correction

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





Information Retrieval, Search

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.

12 May 2015



The Inverted Index Model

- Text Preprocessing

- Inverted Index

- Answering Keyword Queries

- Building inverted files

- Spelling correction

- Clustering

- Indexing Other Media

- Measuring the quality of results

- Conclusion





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

عربى

Gharbi

ḡom





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
pörn

Respectively: Korean Hanguk, 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

12 May 2015





Tokenization: Example

- d_1 the₁ jaguar₂ is₃ a₄ new₅ world₆ mammal₇ of₈ the₉ felidae₁₀ family₁₁
- d_2 jaguar₁ has₂ designed₃ four₄ new₅ engines₆
- d_3 for₁ jaguar₂ atari₃ was₄ keen₅ to₆ use₇ a₈ 68k₉ family₁₀ device₁₁
- d_4 the₁ jacksonville₂ jaguars₃ are₄ a₅ professional₆ us₇ football₈ team₉
- d_5 mac₁ os₂ x₃ jaguar₄ is₅ available₆ at₇ a₈ price₉ of₁₀ us₁₁ \$199₁₂
for₁₃ apple's₁₄ new₁₅ family₁₆ pack₁₇
- d_6 one₁ such₂ ruling₃ family₄ to₅ incorporate₆ the₇ jaguar₈ into₉
their₁₀ name₁₁ is₁₂ jaguar₁₃ paw₁₄
- d_7 it₁ is₂ a₃ big₄ cat₅





Normalization (slide from [Manning et al., 2008])

- Need to “normalize” terms in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define **equivalence classes** of terms.
- Alternatively: do asymmetric expansion
 - window → window, windows
 - windows → Windows, windows
 - Windows (no expansion)
- More powerful, but less efficient

Exercise

Why don't you want to put window, Window, windows, and Windows in the same equivalence class?





Normalization: Other Languages

(slide from [Manning et al., 2008])

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)
- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT \neq mit





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 the₁ jaguar₂ be₃ a₄ new₅ world₆ mammal₇ of₈ the₉ felidae₁₀ family₁₁
- d_2 jaguar₁ have₂ design₃ four₄ new₅ engine₆
- d_3 for₁ jaguar₂ atari₃ be₄ keen₅ to₆ use₇ a₈ 68k₉ family₁₀ device₁₁
- d_4 the₁ jacksonville₂ jaguar₃ be₄ a₅ professional₆ us₇ football₈ team₉
- d_5 mac₁ os₂ x₃ jaguar₄ be₅ available₆ at₇ a₈ price₉ of₁₀ us₁₁ \$199₁₂
for₁₃ apple₁₄ new₁₅ family₁₆ pack₁₇
- d_6 one₁ such₂ rule₃ family₄ to₅ incorporate₆ the₇ jaguar₈ into₉
their₁₀ name₁₁ be₁₂ jaguar₁₃ paw₁₄
- d_7 it₁ be₂ a₃ big₄ cat₅





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 jaguar₂ new₅ world₆ mammal₇ felidae₁₀ family₁₁
- d_2 jaguar₁ design₃ four₄ new₅ engine₆
- d_3 jaguar₂ atari₃ keen₅ 68k₉ family₁₀ device₁₁
- d_4 jacksonville₂ jaguar₃ professional₆ us₇ football₈ team₉
- d_5 mac₁ os₂ x₃ jaguar₄ available₆ price₉ us₁₁ \$199₁₂ apple₁₄
new₁₅ family₁₆ pack₁₇
- d_6 one₁ such₂ rule₃ family₄ incorporate₆ jaguar₈ their₁₀ name₁₁
jaguar₁₃ paw₁₄
- d_7 big₄ cat₅



The Inverted Index Model

Text Preprocessing

Inverted Index

Answering Keyword Queries

Building inverted files

Spelling correction

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





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**; hashing gives the machine responsible for a given term
- 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.



Index size matters

We want to index a collection of 1M emails. The average size of an email is 1,000 bytes and each email contains an average of 100 words. The number of distinct terms is 200,000.

1. size of the collection; number of words?
2. how many lists in the index?
3. we make the (rough) assumption that 20% of the terms in a document appear twice; a document appears in how many lists on average ?
4. how many entries in a list?
5. we represent document ids as 4-bytes unsigned integers, what is the index size ?





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	$d_1/6$

...

⇒ 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

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

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:

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





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;

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

$n_{t,d}$ 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	$d_1/6/.47$
...	



The Inverted Index Model

Text Preprocessing

Inverted Index

Answering Keyword Queries

Building inverted files

Spelling correction

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





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

12 May 2015





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 AND ... AND t_n

t_1 OR ... 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)

$g(s_1, \dots, s_n)$ 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

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

1. Take the first entry of each list; one obtains a tuple $T = [e_1, \dots, e_n]$;
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 $g_{d^{(i)}} = g(s(t_1, d^{(i)}), \dots, s(t_n, d^{(i)}))$ 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 $m = g(s(t_1, d^{(1)}), s(t_2, d^{(2)}), \dots, s(t_n, d^{(n)}))$.
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

$q = \text{"new OR family"}$, 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, $R = \emptyset$ and $\tau = +\infty$.

1. $d^{(1)}$ is the first entry in L_{family} , one finds $s(\text{new}, d_1) = .20$; 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.



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 $B(d) \leq m$** .



The Inverted Index Model

Text Preprocessing

Inverted Index

Answering Keyword Queries

Building inverted files

Spelling correction

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





External Sort/Merge

Building an inverted index from a document collection requires a sort/merge of the index entries.

- first extracts triplets $[d, t, f]$ from the collection;
- then sort the set of triplets on the term-docid pair $[t, d]$.
- contiguous inverted lists can be created from the sorted entries.

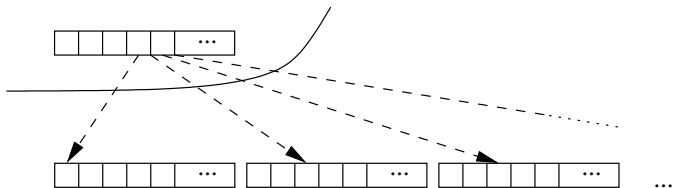
Note: ubiquitous operation; used in RDBMS for ORDER BY, GROUP BY, DISTINCT, and non-indexed joins.





First phase: sort

Repeat: fill the memory with entries; sort in memory (with quicksort); flush the memory in a “run”.



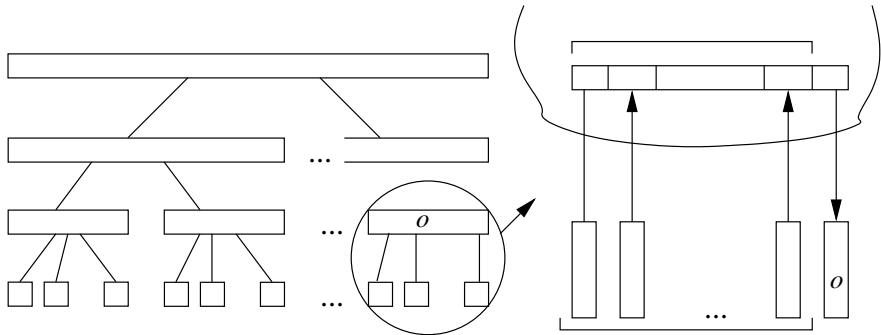
One obtains a list of sorted runs.

Cost: documents are read once; entries are written once.



Second phase: merge

Group the runs and merge.



Cost: one read/write of all entries for each level of the merge tree.



Compression of inverted lists

Without compression, an inverted index with positions and weights may be large than the documents collection!

Compression is essential. The gain must be higher than the time spent to compress.

Key to compression in inverted lists: documents are ordered by id:

[87; 273; 365; 576; 810].

First step: use *delta-coding*:

[87; 186; 92; 211; 234].

Exercise: what is the minimal number of bytes for the first list? for the second?





Variable bytes encoding

Idea: encode integers on 7 bits ($2^7 = 128$); use the leading bit for termination.

Let $v = 9$, encoded on one byte as 10000101 (note first bit set to 1).

Let $v = 137$.

1. the first byte encodes $v' = v \bmod 128 = 9$, thus $b = 10000101$ just as before;
2. next we encode $v/128 = 1$, in a byte $b' = 00000001$ (note the first bit set to 0).

137 is therefore encoded on two bytes:

00000001 10000101.

Compression ratio: typically 1/4 to 1/2 of the fixed-length representation.

12 May 2015





Exercise

The inverted list of a term t consists of the following document ids:

[345; 476; 698; 703].

Apply the VByte compression technique to this sequence. What is the amount of space gained by the method?



The Inverted Index Model

Text Preprocessing

Inverted Index

Answering Keyword Queries

Building inverted files

Spelling correction

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





k -gram indexes for spelling correction (slide

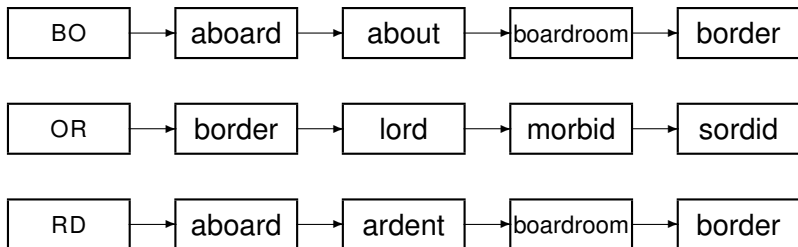
from [Manning et al., 2008])

- **Problem:** able to deal with incorrectly spelled terms in documents, or variants in spelling
- Enumerate all k -grams in the query term
- Use the k -gram index to retrieve “correct” words that match query term k -grams
- Threshold by number of matching k -grams
- E.g., only vocabulary terms that differ by at most 3 k -grams
- Example: bigram index, misspelled word *bordroom*
- Bigrams: *bo, or, rd, dr, ro, oo, om*





k-gram indexes for spelling correction: *bordroom*





Example with trigrams

(slide from [Manning et al., 2008])

- Issue: Fixed number of k -grams that differ does not work for words of differing length.
- Suppose the correct word is NOVEMBER
- Trigrams: *nov, ove, vem, emb, mbe, ber*
- And the query term is DECEMBER
- Trigrams: *dec, ece, cem, emb, mbe, ber*
- So **3 trigrams** overlap (out of 6 in each term)
- How can we turn this into a normalized measure of overlap?





Example with trigrams

(slide from [Manning et al., 2008])

- Issue: Fixed number of k -grams that differ does not work for words of differing length.
- Suppose the correct word is NOVEMBER
- Trigrams: *nov, ove, vem, emb, mbe, ber*
- And the query term is DECEMBER
- Trigrams: *dec, ece, cem, emb, mbe, ber*
- So **3 trigrams** overlap (out of 6 in each term)
- How can we turn this into a normalized measure of overlap?
→ Use **Jaccard coefficient!**





- *an asteroid that fell **form** the sky*
- How can we correct *form* here?
- One idea: **hit-based** spelling correction
 - Retrieve “correct” terms close to each query term
 - for *flew form munich*: *flea* for *flew*, *from* for *form*, *munch* for *munich*
 - Now try all possible resulting phrases as queries with one word “fixed” at a time
 - Try query “*flea form munich*”
 - Try query “*flew from munich*”
 - Try query “*flew form munch*”
 - The correct query “*flew from munich*” has the most hits.
- The “hit-based” algorithm we just outlined is not very efficient.
- More efficient alternative: look at “collection” of queries, not documents



The Inverted Index Model

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion



Clustering Example

web news images wikipedia blogs jobs more »

clusters sources sites

All Results (232)

- Jaguar Cars** (33)
- Parts** (39)
- Club** (33)
- Photos** (28)
- Panthera onca** (15)
- Land Rover** (16)
- Jacksonville Jaguars** (12)
- Defensive, Falcons** (7)
- Atari, Game** (10)
- Classic Jaguar** (6)

Top 232 results of at least 13,030,000 retrieved for the query **jaguar** ([definition](#)) ([details](#))

Search Results

1. [jaguars.com – The official web site of the NFL's Jacksonville Jaguars](#)

The official team site with scores, news items, game schedule, and roster.
[www.jaguars.com](#) • [cache] • Live, Open Directory, Ask

2. [Jaguar](#)



The **jaguar** (*Panthera onca*) is a large member of the cat family native to warm regions of the [Americas](#). It is closely related to the [lion](#), [tiger](#), and [leopard](#) of the [Old World](#), and is the largest species of the cat family found in the Americas.
[en.wikipedia.org/wiki/Jaguar](#) • [cache] • Wikipedia, Ask, Live

3. [Jaguar Enthusiasts' Club](#)

World's largest **Jaguar** / Daimler Club ... Largest **Jaguar** Club in the World, serving over 20,000 members ...
[www.jec.org.uk](#) • [cache] • Ask, Open Directory

4. [US abandons bid for jaguar recovery plan](#)

Jan 18, 2008 - The Interior Department has abandoned attempts to craft a recovery plan for the endangered **jaguar** because too few of the rare cats have been spotted along the Southwest region of New Mexico and Arizona to warrant such action. Some critics of the decision said Thursday the **jaguar** is being sacrificed for the government's new border fence, which is going up along many of the same areas where the ... has crossed into the United States from Mexico. If the U.S. border areas

12 May 2015





Cosine Similarity of Documents

- Document Vector Space model:

terms dimensions

documents vectors

coordinates weights

(The projection of document d along coordinate t is the weight of t in d , say $\text{tfidf}(t, d)$)

- Similarity between documents d and d' : cosine of these two vectors

$$\cos(d, d') = \frac{d \cdot d'}{\|d\| \times \|d'\|}$$

$d \cdot d'$ scalar product of d and d'

$\|d\|$ norm of vector d

- $\cos(d, d) = 1$

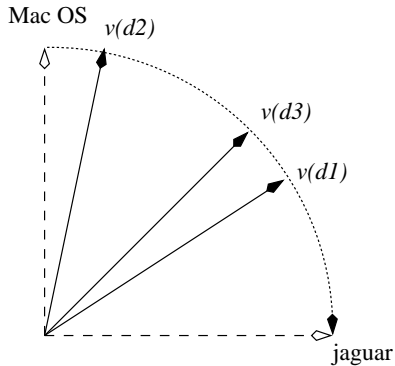
- $\cos(d, d') = 0$ if d and d' are orthogonal (no common term)

12 May 2015





Similarity in a vector space





Agglomerative Clustering of Documents

1. Initially, each document forms **its own cluster**.
2. The similarity between two clusters is defined as the **maximal similarity** between elements of each cluster.
3. Find the two clusters whose mutual similarity is **highest**. If it is **lower than a given threshold**, end the clustering. Otherwise, regroup these clusters. Repeat.

Remark

Many other more refined algorithms for clustering exist.



The Inverted Index Model

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





Indexing HTML

- HTML: text + meta-information + structure
- Possibly: separate index for meta-information (title, keywords)
- Increase weight of structurally emphasized content in index
- Tree structure can also be queried with XPath or XQuery, but not very useful on the Web as a whole, because of tag soup and lack of consistency.





Indexing Multimedia Content

- Basic approach: index **text from context** of the media
 - surrounding text
 - text in or around the links pointing to the content
 - filenames
 - associated subtitles (hearing-impaired track on TV)
- Elaborate approach: index and search the media itself, with the help of **speech recognition** and **sound, image, and video analysis**. Mostly still experimental!
 - TrackID, Shazam: identify a song played on the radio
 - Musipedia: look for music by whistling a tune, <http://www.musipedia.org/>, <http://www.midomi.com/>
 - Image search from a similar image, <http://images.google.com/>, Google Goggles
 - Voxalead, <http://voxaleadnews.labs.exalead.com/>



The Inverted Index Model

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





Precision and Recall

- Quality of search engines results evaluated with **precision** and **recall**

$$\text{precision} = \frac{\text{nb of correct results returned}}{\text{total nb of results}}$$

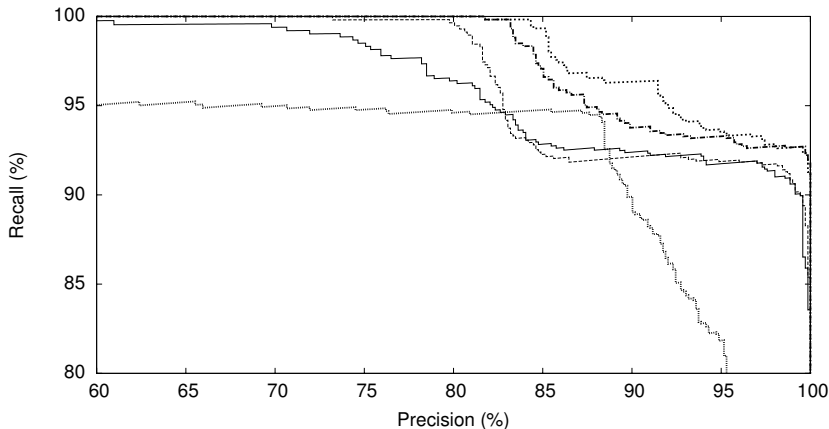
$$\text{recall} = \frac{\text{nb of correct results returned}}{\text{total nb of correct results}}$$

- “Correctness” usually given by **human assessment**
- Precision can be evaluated relatively reliably, much more difficult for recall! (Why?)
- Human judgment quite subjective! Agreement between human evaluators rarely go over 80%





Precision-Recall Curve



- Computed with the **precision-at- k** , **recall-at- k** for the k top results
- **Area under the curve**: quality of a method
- Usually, **interpolate** to force the decreasing of the curve

02 May 2015



The Inverted Index Model

Clustering

Indexing Other Media

Measuring the quality of results

Conclusion





Conclusion

What you should remember

- The inverted index model, associated tools and techniques
- Main ideas behind Fagin's TA and NRA
- The document vector space model

Software

- Most DBMS have text indexing capabilities (e.g., MySQL's FULLTEXT indexes)
- Apache Lucene, free software library to build inverted indexes + Apache Solr, free software for building a keyword search engine (or Elasticsearch that integrates both)

To go further

- A good textbook [Manning et al., 2008]. Available online, along with slides!
- A very influential paper on top- k algorithms: [Fagin et al., 2001]



Bibliography I

- Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proc. PODS*, Santa Barbara, USA, May 2001.
- Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, United Kingdom, 2008. Available online at <http://informationretrieval.org>.
- 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 : sitopedago@telecom-paristech.fr

12 May 2015

