Document indexing, similarities and retrieval in large scale text collections

Eric Gaussier

Univ. Grenoble Alpes - LIG
Eric.Gaussier@imag.fr

Course objectives

- Introduce the main concepts, models and algorithms for computing indexing text documents and computing similarities in large scale text collections
- We will focus on:
  - Document indexing and representations in large scale collections
  - Standard models for Information Retrieval (IR)
  - PageRank (computing the importance of a page on the Web)
  - Learning to rank models

Application domains

- Information retrieval
  - Query indexing module
  - Documents indexing module
  - Module to match queries and documents
- Classification
  - Binary, multi-class; mono-/multi-label
  - Flat vs hierarchical
- Clustering
  - Hard vs soft clustering
  - Flat vs hierarchical

Part 1: Indexing, similarities, information retrieval

Content

1. Indexing
2. Standard IR models
3. Evaluation

Indexing steps

1. Segmentation
   - Segment a text into words:

     the importance of retrieving the good information
     the, importance, of, retrieving, the, good, information

     7 words but only 6 word types; depending on languages, may require a dictionary
2. Stop-word removal (stop-word list)
3. Normalization
   - Upper/lower-case, inflected forms, lexical families
   - Lemmatization, stemming

→ Bag-of-words: *importance, retriev, inform*

Vector space representation

- The set of all word types constitute the vocabulary of a collection. Let M be the size of the vocabulary and N be the number of documents in the collection $\rightarrow$ M-dimensional vector space (each axis corresponds to a word type)
- Each document is represented by a vector the coordinates of which correspond to:
  - Presence/absence or number of occurrences of the word type in the doc: $w_i^d = \mathrm{tf}_i^d$
  - Normalized number of occurrences: $w_i^d = \frac{\mathrm{tf}_i^d}{\sum_{i=1}^M \mathrm{tf}_i^d}$
  - *tf*idf*:
    $$w_i^d = \frac{\mathrm{tf}_i^d}{\sum_{i=1}^M \mathrm{tf}_i^d} \underbrace{\log \frac{N}{\mathrm{df}_i}}_{\mathrm{idf}_i}$$
    where df_i is the number of docs in which word (type) i occurs

A sparse representation

Most of the words (terms) only occur in few documents and most coordinates of each document are null; storage space is thus saved by considering only words present in documents → sparse representation

Example

$$\text{document } d \begin{cases} \text{int l} & \text{(doc length)} \\ \text{ArrWords int[l]} & \text{(sorted word indices)} \\ \text{ArrWeights float[l]} & \text{(word weights)} \\ \ldots & \end{cases}$$

How to compute a dot product between documents?

Dot product with sparse representations

Inverted file

It is possible, with sparse representations, to speed up the comparison between docs by relying on an *inverted file* that provides, for each term, the list of documents they appear in:

$$\text{word } i \begin{cases} \text{int l} & \text{(number of docs)} \\ \text{ArrDocs int[l]} & \text{(sorted doc indices)} \\ \ldots & \end{cases}$$

Remark Advantageous with measures (distances, similarities) that do not rely on words not present in docs; dot/scalar product?, cosine?, Euclidean distance?

Building an inverted file

With a static collection, 3 main steps:

1. Extraction of id pairs *(term, doc)* (complete pass over the collection)
2. Sorting acc. to term id, then doc id
3. Grouping pairs corresponding to same term

Easy to implement when everything fits into memory

How to proceed with large collections?

Insufficient memory

Intermediate "inverted files" are temporarily stored on disk. As before, 3 main steps:

1. Extraction of id pairs *(term, doc)* (previous algo.) and writing on file F
2. Reading file F by blocks that can fit into memory; inversion of each block (previous algo.) and writing in a series of files
3. Merging all local files to create global inverted file

→ *Blocked sort-based indexing* (BSBI) algorithm

BSBI (1)

1. $n \leftarrow 0$
2. while (some docs have not been processed)
3. do
4. $n \leftarrow n + 1$
5. block $\leftarrow$ ParseBlock()
6. BSBI-Invert(block)
7. WriteBlockToDisk(block,f_n)
8. MergeBlocks(f_1, ..., f_n;f_{merged})

BSBI (2)

The inversion (in BSBI) consists in sorting pairs on two different keys (term and doc ids). Complexity in $O(T \log T)$ where T represents the number of (term,doc) pairs

Example

$t_1 =$ "brutus", $t_2 =$ "caesar", $t_3 =$ "julius", $t_4 =$ "kill", $t_5 =$ "noble"

$t_1 : d_1$	$t_2 : d_4$	$t_2 : d_1$
$t_3 : d_{10}$	$t_1 : d_3$	$t_4 : d_8$
$t_5 : d_5$	$t_2 : d_2$	$t_1 : d_7$

Standard IR models

The different standard models

- Boolean model
- Vector-space model
- Prob. models

Boolean model (1)

Simple model based on set theory and Boole algebra, characterized by:

- Binary weights (presence/absence)
- Queries as boolean expressions
- Binary relevance
- System relevance: satisfaction of the boolean query

Boolean model (2)

Example

q = programming $\wedge$ language $\wedge$ (C $\vee$ java)
(q = [prog. $\wedge$ lang. $\wedge$ C] $\vee$ [prog. $\wedge$ lang. $\wedge$ java])

	programming	language	C	java	$\cdots$
d_1	3 (1)	2 (1)	4 (1)	0 (0)	$\cdots$
d_2	5 (1)	1 (1)	0 (0)	0 (0)	$\cdots$
d_0	0 (0)	0 (0)	0 (0)	3 (1)	$\cdots$

Relevance score

$RSV(d,q) = 1$ iff $\exists\, q_{cc} \in q_{dnf}$ s.t. all terms in q_{cc} are in d; 0 otherwise

Boolean model (3)

Algorithmic considerations

Sparse term-document matrix: inverted file to select all document in conjonctive blocks (can be processed in parallel) - intersection of document lists

	d_1	d_2	d_3	$\cdots$
programming	1	1	0	$\cdots$
langage	1	1	0	$\cdots$
C	1	0	0	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	

Boolean model (4)

Advantages and disadvantages

+ Easy to implement (at the basis of all models with a union operator)

- Binary relevance not adapted to topical overlaps

- From an information need to a boolean query

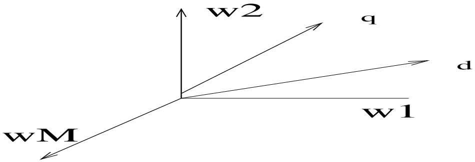
Remark At the basis of many commercial systems

Vector space model (1)

Corrects two drawbacks of the boolean model: binary weights and relevance

It is characterized by:

- Positive weights for each term (in docs and queries)
- A representation of documents and queries as vectors (see before on bag-of-words)



Vector space model (2)

Docs and queries are vectors in an M-dimensional space the axes of which corresponds to word types

Similarity Cosine between two vectors

$$RSV(d_j, q) = \frac{\sum_i w_i^d w_i^q}{\sqrt{\sum_i (w_i^d)^2}\sqrt{\sum_i (w_i^q)^2}}$$

Proprerty The cosine is maximal when the document and the query contain the same words, in the same proportion! It is minimal when they have no term in common (similarity score)

Vector space model (3)

Advantages and disadvantages

+ Total order (on the document set): distinction between documents that completely or partially answer the information need

- Framework relatively simple; not amenable to different extensions

Complexity Similar to the boolean model (dot product only computed on documents that contain at least one query term)

Probabilistic models

- *Binary Independence Model* and BM25 (S. Robertson & K. Sparck Jones)
- *Inference Network Model* (Inquery) - *Belief Network Model* (Turtle & Croft)
- *(Statistical) Language Models*
  - *Query likelihood* (Ponte & Croft)
  - *Probabilistic distance retrieval model* (Zhai & Lafferty)
- *Divergence from Randomness* (Amati & Van Rijsbergen) - *Information-based models* (Clinchant & Gaussier)

Generalities

Boolean model	$\rightarrow$	binary relevance
Vector space model	$\rightarrow$	similarity score
Probabilistic model	$\rightarrow$	probability of relevance

Two points of view: document generation (probability that the document is relevant to the query - BIR, BM25), query generation (probability that the document "generated" the query - LM)

Introduction to language models: two die

Let D_1 and D_2 two (standard) die such that, for small ϵ:

For D_1, $P(1) = P(3) = P(5) = \frac{1}{3} - \epsilon$, $P(2) = P(4) = P(6) = \epsilon$
For D_2, $P(1) = P(3) = P(5) = \epsilon\,$; $P(2) = P(4) = P(6) = \frac{1}{3} - \epsilon$

Imagine you observe the sequence $Q = (1, 3, 3, 2)$. Which dice most likely produced this sequence?

Answer

$P(Q|D_1) = (\frac{1}{3} - \epsilon)^3\epsilon\,$; $P(Q|D_2) = (\frac{1}{3} - \epsilon)\epsilon^3$

Introduction to language models: two die

Let D_1 and D_2 two (standard) die such that, for small ϵ:

For D_1, $P(1) = P(3) = P(5) = \frac{1}{3} - \epsilon$, $P(2) = P(4) = P(6) = \epsilon$
For D_2, $P(1) = P(3) = P(5) = \epsilon\,;\ P(2) = P(4) = P(6) = \frac{1}{3} - \epsilon$

Imagine you observe the sequence $Q = (1, 3, 3, 2)$. Which dice most likely produced this sequence?

Answer

$P(Q|D_1) = (\frac{1}{3} - \epsilon)^3\epsilon\,;\ P(Q|D_2) = (\frac{1}{3} - \epsilon)\epsilon^3$

Language model - QL (1)

Documents are die; a query is a sequence → What is the probability that a document (dice) generated the query (sequence)?

$$RSV(q,d) =_{\mathrm{r}} P(q|d) = P(q_1 ... q_l|d) = \prod_{j=1}^{l} P(q_j|d) = \prod_{i} P(i|d)^{occ(i;q)}$$

where occ_i^q denotes number of occurrences of word i in q

How to estimate the quantities $P(i|d)$?
Maximum Likelihood principle ⇒ $p(i|d) = \frac{occ(i;d)}{\sum_i occ(i;d)}$

Problem with query words not present in docs

Language model - QL (2)

Solution: smoothing

One takes into account the collection model:

$$p(w|d) = (1 - \alpha_d) \frac{occ(i;d)}{\sum_i occ(i;d)} + \alpha_d \frac{occ(i;\mathcal{C})}{\sum_i occ(i;\mathcal{C})}$$

Example with Jelinek-Mercer smoothing: $\alpha_d = \lambda$

- $\mathcal{D}$: development set (collection, some queries and associated relevance judgements)
- $\lambda = 0$:
- Repeat till $\lambda = 1$
  - IR on $\mathcal{D}$ and evaluation (store evaluation score and associated λ)
  - $\lambda \leftarrow \lambda + \epsilon$
- Select best λ

Language model - QL (3)

Advantages and disadvantages

+ Theoretical framework: simple, well-founded, easy to implement and leading to very good results

+ Easy to extend to other settings as cross-language IR

- Training data to estimate smoothing parameters

- Conceptual deficiency for (pseudo-)relevance feedback

Complexity similar to vector space model

Evaluation of IR systems

Relevance judgements

- Binary judgements: the doc is relevant (1) or not relevant (0) to the query
- Multi-valued judgements:
  Perfect > *Excellent* > *Good* > *Correct* > *Bad*
- Preference pairs: doc d_A more relevant than doc d_B to the query

Several (large) collections with many (> 30) queries and associated (binary) relevance judgements: TREC collections (trec.nist.gov), CLEF (www.clef-campaign.org), FIRE (fire.irsi.res.in)

Common evaluation measures

- MAP (Mean Average Precision)
- MRR (Mean Reciprocal Rank)
  - For a given query q, let r_q be the rank of the first relevant document retrieved
  - MRR: mean of r_q over all queries
- WTA (Winner Takes All)
  - If the first retrieved doc is relevant, $s_q = 1$; $s_q = 0$ otherwise
  - WTA: mean of s_q over all queries
- NDCG (Normalized Discounted Cumulative Gain)

NDCG

- NDCG at position k:

$$N(k) = \overbrace{Z_k}^{\text{normalization}} \underbrace{\sum_{j=1}^{k}}_{\text{cumul}} \overbrace{(2^{p(j)} - 1)}^{\text{gain}} \; / \; \underbrace{\log_2(j+1)}_{\text{position discount}}$$

- Averaged over all queries

G : Gain

Relevance	Value (gain)
Perfect (5)	$31 = 2^5 - 1$
Excellent (4)	$15 = 2^4 - 1$
Good (3)	$7 = 2^3 - 1$
Correct (2)	$3 = 2^2 - 1$
Bad (0)	$0 = 2^1 - 1$

DCG : Discounted CG

Discounting factor: $\frac{\ln(2)}{\ln(j+1)}$

Doc. (rg)	Rel..	Gain	CG	DCG
1	*Perf. (5)*	31	31	31
2	*Corr. (2)*	3	$34 = 31 + 3$	$32,9 = 31 + 3 \times 0,63$
3	*Exc. (4)*	15	49	$40,4$
4	*Exc. (4)*	15	64	$46,9$
...	...	...	...	...

Ideal ranking: max DCG

Document (rank)	Relevance	Gain	max DCG
1	*Perfect (5)*	31	31
3	*Excellent (4)*	15	$40,5$
4	*Excellent (4)*	15	48
...	...	...	...

Normalized DCG

Doc. (rang)	Rel.	Gain	DCG	max DCG	NDCG
1	*Perfect (5)*	31	31	31	1
2	*Correct (2)*	3	$32,9$	$40,5$	$0,81$
3	*Excellent (4)*	15	$40,4$	48	0.84
4	*Excellent (4)*	15	$46,9$	$54,5$	0.86
...	...	...	...	...	

Remarks on evaluation measures

- Measures for a given position (e.g. list of 10 retrieved documents)
- NDCG is more general than MAP (multi-valued relevance vs binary relevance)
- Non continuous (and thus non derivable)