Indexing & querying text

September 2018

Pierre-Edouard Portier

http://peportier.me/teaching_2018_2019/

LA-Times

- http://trec.nist.gov/data/docs_eng.html
- Daily articles from 1989 and 1990
- Let’s have a look at file la010189

Tokenization

- Language-dependent
- Compound words,…
- For your first iteration, keep it simple…
  - Space character as a delimiter
  - Remove punctuation,…
  - Or use a library… http://www.nltk.org/api/nltk.tokenize.html
- Many options
  - removing tags <BYLINE>,…, <P>,...<TYPE>
  - <number>, <money>,... to replace instances of numbers or currencies

Stemming

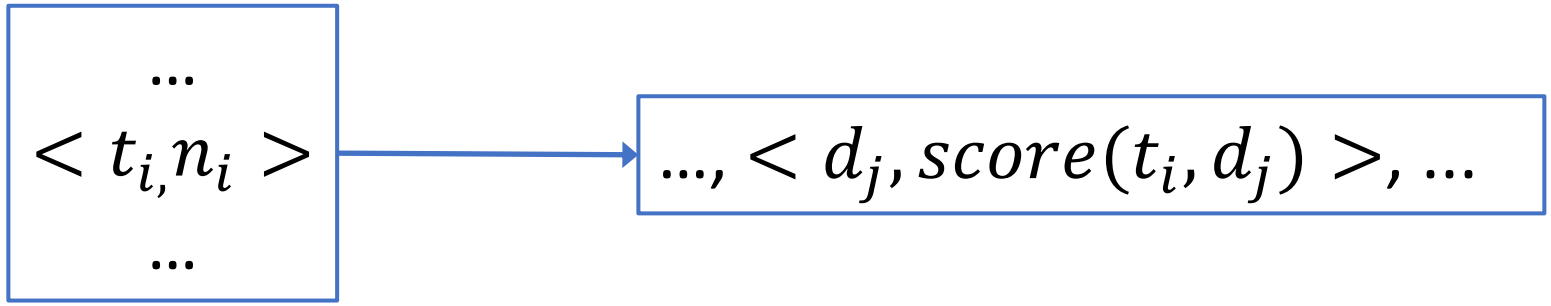
- https://en.wikipedia.org/wiki/Stemming
- “stems”, “stemmer”, “stemming” → “stem”
- Language-dependent
- For sure, a gain of space…
- For your first iteration, keep it simple…
  - Use Porter’s algorithm
  - (Or don’t use stemming)

Stop words removal

- https://en.wikipedia.org/wiki/Stop_words
- Can increase performance
- May limit phrase search
- May remove good signals for text classification, sentiment analysis, etc.

Inverted file (IF)

- Mapping from *vocabulary of terms* (VOC) to *posting lists* (PL)



- VOC maps a pair `<term, size-of-PL>` to
  - an offset in the PL-FILE, or an index in a memory-mapped table
- VOC can be a hash-map or a search-tree (e.g. B-Tree)
- PL are contiguous parts of the PL-FILE to minimize seek time on rotating drives

$$score(t_i, d_j)$$

Term-frequency

$$tf(t,d) = 1 + \log\left(n_{t,d}\right) \quad or \quad 0 \quad if \quad n_{t,d} = 0$$

$n_{t,d}$: frequency of term t in document d

Inverse-document-frequency

$$idf(t) = \log \frac{|D|}{1 + |\{d \in D \mid n_{t,d} > 0\}|}$$

D : the set of all the documents

- Discriminative power for term t

$$idf(t) = \log \frac{|D|}{1 + |\{d \in D \mid n_{t,d} > 0\}|}$$

- $P(t) = \frac{|\{d \in D \mid n_{t,d} > 0\}|}{|D|}$ estimate of the probability of presence of t in a document
- $P(t) \leq 1$, and $\log(P(t)) \leq 0$, thus the inverted ratio: $idf(t) = -\log(P(t))$
- *Independence*:

$$p(AB) = p(A)p(B) \quad \equiv \quad \log\big(p(AB)\big) = \log\big(p(A)\big) + \log(p(B))$$

- *Additivity*:

$$idf(t_1 \wedge t_2) = idf(t_1) + idf(t_2)$$

- The assumption of independence matches with the *Vector Space Model* (VSM)
  - idf values are scaling factors applied to the dimensions of the VSM
- log compresses the scale, so that large and small quantities can be compared (Zipf's law)

  - https://plus.maths.org/content/mystery-zipf
  - http://www-personal.umich.edu/~mejn/courses/2006/cmplxsys899/powerlaws.pdf

$$score(t_i, d_j) = tf(t_i, d_j) \times idf(t_i)$$

Aggregation of the scores for each term of the query

- Monotonous aggregation function (e.g., sum)
  - $F(x_1, \ldots, x_m) \leq F(x'_1, \ldots, x'_m)$ whenever $x_i \leq x'_i$ for all i
- Summation is similar to cosine similarity if
  - Document vectors are normalized
  - Query term frequency is at most 1
- For your 1st iteration:
  - Focus on the sum as an aggregation function
  - Consider only conjunctive and/or disjunctive queries

Ranked queries, a naïve algorithm

- $Q = t_1 \wedge \cdots \wedge t_n$, or $Q = t_1 \vee \cdots \vee t_n$
- The PL are ordered by DOC ID
- Parallel scan of the PL to find each DOC ID for which *at least one* (OR-QUERY) or *all* (AND-QUERY) the query terms are present
- $score(d_j) = \sum_{i=1}^{n} score(t_i, d_j)$
- At the end of the parallel scan, the d_j must be sorted by their score
- For AND-QUERY the search is linear in the size of the smallest PL
- OR-QUERY requires a full scan of the PL

Naïve top-k algorithm

- `s(t1):    (d1,0.50), (d2,0.90), (d3,0.40), (d4,0.6), (d5,0.8), (d6,0.7)`
- `s(t2):    (d1,0.74), (d2,0.75), (d3,0.85), (d4,0.7), (d5,0.8), (d6,0.74)`
- `s(t1+t2): (d1,0.62), (d2,0.825), (d3,0.625), (d4,0.65), (d5,0.8), (d6,0.72)`
- Monotonic aggregative function: mean, sum, min,…
  Let’s take the mean
- Top-3: d2, d5, d6
- At least one access to each doc for each query term (i.e., linear time in the nb of docs… if they are sorted by ids)

Fagin's top-k query algorithm

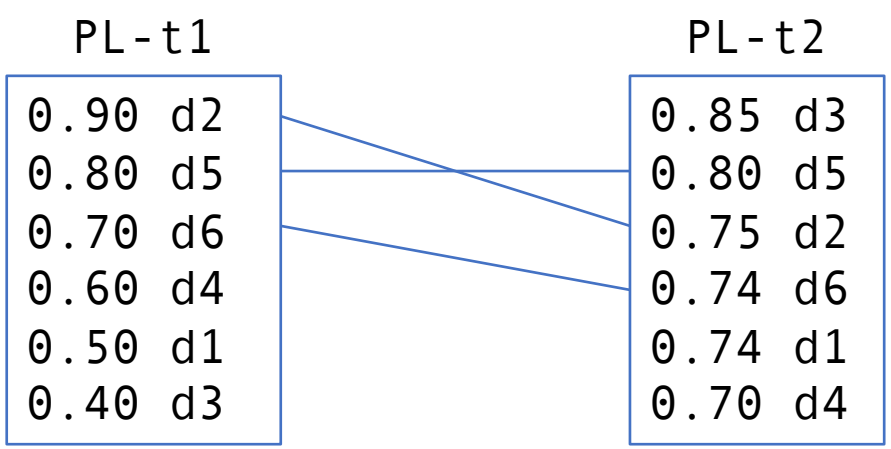
- Sorted access:
  access to the PL in non-increasing order of the scores
- Random access:
  quick access to $s(t_i, d_j)$ (e.g., binary search on PL sorted by doc-id)

Fagin's top-k query algorithm (FA)

1. $M = \emptyset; C = \emptyset;$
2. Repeat until |C|=k
    1. Sorted access in parallel to the qt (let d be the doc met)
    2. If d has been seen for all the qt
        1. Remove d from M
        2. Insert (d,s(t1+t2+…, d)) into C
    3. Else if d is seen for the first time
        1. Insert d into M
3. For each d ∈ M
    1. Random access to all remaining qt to compute the aggregated score of d
    2. Insert (d,s(t1+t2+…, d)) into C
4. Return the top-k docs in C

With high probability, the number of documents visited is $O(k^{\frac{1}{m}}|D|^{1-\frac{1}{m}})$
With m the number of query terms.
What happens when m grows? → $O(|D|)$

k=3



This algorithm is correct.

Let D be an unseen doc
with scores: x1, x2

Let S be one of the docs returned
with scores: y1, y2

Then x1 <= y1, x2 <= y2, etc.

Let F be a monotonic aggregation fn.

F(x1,x2) <= F(y1,y2)

M = (d2,0.90),(d3,0.85),(d5,0.80),(d6,0.70),

C = (d5,0.80),(d2,0.825), (d6,0.72),

R = (d2,0.825),(d5,0.80),(d6,0.72),

d3: 0.625 < 0.72

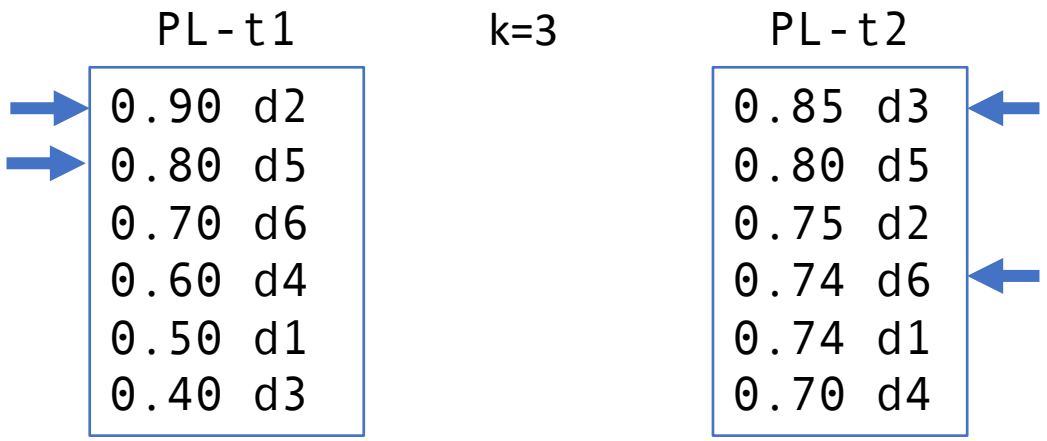
Ronald Fagin. Combining fuzzy information from multiple systems.
In *Proceedings of the ACM Symposium on Principles of Database Systems*, pages 216–226, 1996

Fagin's Threshold Algorithm (TA)

- "Simple" threshold algorithm earns 2014 Gödel Prize
- Optimal Aggregation Algorithms for Middleware (PODS, 2001)
- An optimal way to retrieve the top-k results for monotonous score aggregation

Fagin's threshold algorithm (TA)

1. $C = \emptyset$; $\tau = +\infty$; $\mu_{min} = +\infty$;
2. Repeat until at least k docs have been seen with grade at least τ
    1. Sorted access in parallel to each query's terms (qt) (let d be the doc met)
        1. Rand access to all remaining qt to compute the combined score $\mu(d)$
        2. If |C|<k
            1. Insert (d,$\mu(d)$) into C
            2. $\mu_{min} = \min\{\mu(d)|d \in C\}$
        3. Else if $\mu_{min} < \mu(d)$
            1. Remove the document with the smallest score from C
            2. Insert $(d, \mu(d))$ into C
            3. $\mu_{min} = \min\{\mu(d)|d \in C\}$
        4. If at least one doc has been seen under sorted access for each qt
            1. Let τ_i be the score of the doc before the next doc that will be seen under sorted access for qt i
            2. $\tau = \mu(\tau_1, \dots, \tau_m)$
3. Return C



C = (d2,0.825) (d3,0.625) (d5,0.80) (d6,0.72)

$\tau = +\infty$ μ(0.90,0.85)=0.875 μ(0.80,0.75)=0.775 μ(0.70,0.74)=0.72

$\mu_{min} = +\infty$ 0.825 0.625 0.72

ϵ-approximation of top-k answers

- Given a collection of k docs D1,…Dk, for each D not among them:
- $(1+\epsilon)F(D_i) \geq F(D)$
- Modify TA with:
- `Repeat until at least k docs have been seen with grade at least` $\frac{\tau}{1+\epsilon}$

Merge-based algorithm to build the IF

- Start building an IF in memory.
- When the memory is full, flush the PL and the VOC of this partial IF to the disk.
- When all the documents have been seen, merge the flushed PL to obtain the IF.
- Note: the partial PL should be easily accessed in sorted order of theirs VOC…
  - E.g. http://www.grantjenks.com/docs/sortedcontainers/

PL Compression

- Advantage:
  - More of the IF can be stored in main memory
- Requirement:
  - Time to read a compressed PL from disk + decompressing it *must be less than*
  - Time to read an uncompressed PL from disk
- Strategy:
  - Sort the PL by doc-id and store the deltas between consecutive doc-ids
  - Use variable-byte encoding for these deltas

Variable Byte (Vbyte) encoding

- Given integer v

1. IF $v < 128$
   1. v is encoded on the last 7 bit of a byte b
      1. On first entry in this branch, the first bit of b is set to 1
      2. Otherwise, the first bit of b is set to 0
   2. Return b
2. ELSE
   1. let v' and k s.t. $v = k \times 128 + v'$
   2. $b \leftarrow VBYTE(v')$
   3. Return $VBYTE(k)$ concatenated with b

VBYTE decoding

- $VBYTE(v) = b_{n-1} \ldots b_1 \, b_0$
- $v = \; b_{n-1} \times 128^{n-1} + \; \ldots + b_1 \times 128 + \; (b_0 - 128)$

VBYTE example

- $v = 130$
- $VBYTE(v) = 0000\ 0001\ 1000\ 0010$
- $v = (0000\ 0001)_2 \times 128\ +\ \ (0000\ 0010)_2$

Further information

- Zobel, Justin, and Alistair Moffat. "Inverted files for text search engines." *ACM computing surveys (CSUR)* 38.2 (2006): 6.
- Petri, Matthias, and Alistair Moffat. "Compact inverted index storage using general-purpose compression libraries." *Software: Practice and Experience* 48.4 (2018): 974-982.
  - Applying Zlib or BZip2 after Vbyte encoding
- Broder, Andrei Z., et al. "Efficient query evaluation using a two-level retrieval process." *Proceedings of the twelfth international conference on Information and knowledge management*. ACM, 2003.
  - Contrary to Fagin's TA, only the PL sorted by docid are required.

Towards “Meaning” through vector spaces

- High-dimensional representation of words based on their contexts
  - the context can be the document, a sentence, a few words,…
- Distributional hypothesis
  - Words that occur in the same contexts tend to have related/similar meanings
- Similar vectors (e.g., cosine similarity) lead to synonyms, vocabulary of related words,…
  - Nearest neighbors: http://www.cs.ubc.ca/research/flann/
  - E.g. can be used for query expansion
- Dimension reduction is needed for
  - Fighting the sparseness (Zipf’s law…)
  - Efficiency
- Singular Value Decomposition
  - High computational cost

Random indexing

- Each doc is assigned an index vector
  - Filled mostly with zeros except a few +1 and -1 randomly located
  - Expected orthogonal (related to the Johnson and Lindenstrauss lemma, see [1])
  - Dimension: between 100 and 10000
  - Number of non-zero elements: between 2 and 20
  - Mean: 0
- Each word is assigned a context vector
  - Initialized to zero
  - Each time the word occur in a doc, the index vector of that doc is added to the context vector of the word
- Semantic vectors for doc by adding the columns that the doc activates[2]

[1] QasemiZadeh, Behrang, and Siegfried Handschuh. "Random indexing explained with high probability." *International Conference on Text, Speech, and Dialogue*. Springer, Cham, 2015.

[2] Kanerva, Pentti. "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors." *Cognitive Computation* 1.2 (2009): 139-159.

Groups and githubs

- https://lite.framacalc.org/4D8T8ylzQO

Merci