Indexing & Querying Text

September 2016

Pierre-Edouard Portier

http://liris.cnrs.fr/pierre-edouard.portier/teaching_2016_2017/

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
- Etc.
- 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
  - If you think of minor hacks (e.g., removing tags <BYLINE>,…, <P>,…<TYPE>)
    - Ask yourself: are they necessary?

STEMMING

- [HTTPS://EN.WIKIPEDIA.ORG/WIKI/STEMMING](https://en.wikipedia.org/wiki/Stemming)
- "STEMS", "STEMMER", "STEMMING" → "STEM" ☺
- "MARKETING", "MARKETS" → "MARKET" ☹
- LANGUAGE-DEPENDENT
- 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

Inverted File (IF)

- Mapping between *vocabulary of terms* (VOC) and *posting lists* (PL)

$$\begin{matrix} \ldots \\ < t_{i,} n_i > \\ \ldots \end{matrix} \longrightarrow \ldots, < d_j, score(t_i, d_j) >, \ldots$$

VOC PL

- VOC maps a pair <term, size-of-PL> to
  - the value of an offset in the PL-FILE, or an index of 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

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

- The denominator is a normalization factor used to remove the influence of the document's length

INVERSE-DOCUMENT-FREQUENCY

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

- DISCRIMINANT POTENTIAL FOR TERM t

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

- $\frac{|\{d \in D \mid n_{t,d} > 0\}|}{|D|}$ ESTIMATES THE PROBABILITY FOR THE PRESENCE OF TERM t IN ANY DOCUMENT
- INDEPENDENCE: $p(AB) = p(A)p(B) \equiv \log(p(AB)) = \log(p(A)) + \log(p(B))$
- ADDITIVITY: $idf(t_1 \wedge t_2) = idf(t_1) + idf(t_2)$
- $\frac{|\{d \in D \mid n_{t,d} > 0\}|}{|D|} \leq 1$, AND THE LOG CAN BE NEGATIVE, THUS THE INVERTED RATIO
- LOG COMPRESSES THE SCALE SO THAT LARGE AND SMALL QUANTITIES CAN BE COMPARED
- THE ASSUMPTION OF INDEPENDENCE MATCHES WITH THE *VECTOR SPACE MODEL (VSM)*
  - IDF VALUES CAN BE CONSIDERED AS DIMENSION WEIGHTS

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

Aggregation Function

- Monotonous score aggregation function (e.g., sum)
- Cosine similarity is reduced to summation by pre-normalizing vector lengths to 1
- For your 1st iteration:
  - you can focus on the sum as an aggregation function
  - Consider only conjunctive and 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_{t_i} to find each DOC ID d_j for which *at least one* (OR-QUERY) or *all* (AND-QUERY) the terms of the query are present
- $score(d_j) = \sum_{i=1}^{n} score(t_i, d_j)$
- At the end of the parallel scan, the d_j are 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

Fagin's Threshold Algorithm (TA)

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

- 0. $R \leftarrow \emptyset$; $\tau \leftarrow +\infty$;
- 1. ; <u>DO</u> $0 \leq i < n \rightarrow$
  - $d^i \leftarrow$ Doc from PL_{t_i} (*sorted by weights*) with the next best $score(t_i, d^i)$
  - ; $score(d^i) \leftarrow score(t_i, d^i) + \sum_{j \neq i} score(t_j, d^i)$
  - NB. $score(t_j, d^i)$ can be obtained from a binary search on PL_{t_j} (*sorted by doc ids*)
  - ; <u>IF</u> $|R| < k \rightarrow$
    - $R \leftarrow R \cup \{d^i\}$
  - | $|R| \geq k \rightarrow$
    - $dm \leftarrow argmin_{d' \in R}\, score(d')$
    - ; <u>IF</u> $score(d^i) > score(dm) \rightarrow R \leftarrow (R \backslash \{dm\}) \cup \{d^i\}$ <u>FI</u>
  - <u>FI</u>
- <u>OD</u>
- 2. ; $\tau \leftarrow \sum_{i=0}^{n-1} score(t_i, d^i)$
- 3. ; <u>IF</u> $|R| < k \rightarrow GOTO\ 1$ <u>FI</u>
- 4. ; <u>IF</u> $\min_{d' \in R} score(d') < \tau \rightarrow GOTO\ 1$ <u>FI</u>
- 5. ; <u>RETURN</u> R

Merge-based algorithm to build the IF

- Start building an IF in memory.
- When the memory is full, flush the PL of this partial IF to the disk.
- When all the documents have been seen, merge the flushed PL to obtain the IF.

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

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

VBYTE EXAMPLE

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

ZIPF'S LAW

- HTTPS://PLUS.MATHS.ORG/CONTENT/MYSTERY-ZIPF
- HTTP://WWW-PERSONAL.UMICH.EDU/~MEJN/COURSES/2006/CMPLXSYS899/POWERLAWS.PDF
- DO PL LENGTHS FOLLOW A ZIPF'S LAW?
- WHAT IS THE BEST ESTIMATE OF THE PARAMETER "A" ON OUR DATASET?
  - E.G., LEAST SQUARES ESTIMATE