LIVE SESSIONS

ABSTRACT WORD AND DOCUMENT MODEL

word → sequence of characters separated by delimiters → some languages don't have easy delimiters

└→ Sometimes case sensitive   AT&T → at

WORD ⟷ INTEGER WORD IDs

WORD AND DOCUMENT IDs

DOCUMENT IDs → Randomly assigned
⇓
Better assignment for Index Compression

MAP : WORD → WORD IDs
└→ Can be compressed

DOCUMENT REPRESENTATION

- SEQUENCE ↓ Complete Representation
- SET ↓ Loses Information

INDEXING — Transposing + Compression ⟹ BINARY MATRIX: Doc IDs × Word IDs

INCIDENT VECTORS AND BOOLEAN QUERIES :

└→ Document is either relevant or irrelevant ⟹ No notion of relative relevance → Emails, Library Catalogue

Phrase Queries
↓
Need to keep track of positions of words

INVERTED INDEX : → variable-size posting list → DATA STRUCTURE

- Variable Arrays
- Linked lists

Can be compressed ⟸ [

★ INDEXER STEPS
① Tokenisation
② Sort → a. Terms
b. Document ID
③ Dictionary and Postings → Also calculate document frequency.
↓
USE OF DOCUMENT FREQUENCY
↓
INTERSECTION always reduces set size.

★ If you start with a smaller set, overall process will consume less time.

STORAGE

TERM Doc. FREQUENCY → POSTING LISTS
Pointers
POSTING LISTS → DISK (Small Corpus)
POSTING LISTS → RAM on Cluster of Servers (Large Corpus)

QUERY OPTIMISATION :

- Merging / Process in increasing order of frequency

STREAMS OR CHANNELS

- Raw text streams
- Lowercase token stream
- Pos stream
  └→ Compound Token

★ Named Entity Tag. Ex. PERSON, GEOLOCATION

POSITIONAL QUERIES :

- "A" within x words of "B"
- Phrase "A B"

★ SOLUTION : Retain relative offset of position in indexes

⇒ Why compression? Disk transfer is slow but decompression algorithms are fast

COMPRESSION TECHNIQUES

① Store GAPS vs DOC IDS directly → Higher bit requirement

CASE ① ⇒ High frequency word → Gaps will be low. ⇒ VARIABLE LENGTH CODES to reduce size

CASE ② ⇒ Low frequency word → Few large gaps ⇒ lower bit consumption.

VARIABLE LENGTH ENCODING :

① VARIABLE BYTE CODE → For every byte _ - - - - - - -

Continuation bit

Payload

★ Not very efficient for small gaps → Gap ~ 5 (3-bits)

② UNARY CODE → n 1s with a final 0. → Useful for cases where number is very small

$Pr(N=n) \propto 2^{-n}$

⇒ Not very efficient in general.

③ GAMMA CODE - Cutoff leading bit

- Encode length of offset and append in front.

In Unary

Ex. 13 → 1110 101

BIT REQUIREMENT → $2\lfloor \log G \rfloor + 1$

Q. Find the probability for which gamma code is optimal?

Compressing The Term List :

→ Dictionary-As-A-String : → Pointer to the next word shows end of current word
→ Can do binary search over the string

Space : 4 bytes for frequency
4 bytes for pointer to Posting
3 bytes per term pointer
Avg 8 bytes in term string

400K terms x 19 ⇒ 7.6 MB vs. 11.2MB for fixed width

→ Blocking : Adjacent words share prefix → Example

ADAM → CANCEL 0 APPEND ANT
ADAMANT
ADVERSE → CANCEL 5 APPEND VERSE

vs 7.1MB

What is k=4?

Positional Information

[DOC ID ; POSITIONS] → d gaps, p gaps

★ Occurence of words in a document are "BURSTY"

Live Session - 4

- GFS / HDFS
- MAP REDUCE → Bulk synchronous parallel computation paradigm → INDEX CONSTRUCTION
  - MAP : collection → list(termID, docID)
  - REDUCE : reduce (<term ID, list(docID)> ...
  - → (postings list 1, postings list 2 ...)
- SSTABLE → Sorted String Table : Immutable Key-Value Pair
- BIG TABLE (Hbase)
- PERCOLATOR

Parsers : → Reads document (1 at a time) and emits (term, doc) pairs

Map Reduce For Word Count :

MAP

for each doc d
  for each word t
    output (t,1)

REDUCE (<t, list(counts)>)

total ← 0
for each count in
  total += count
output <t, (total)>

★ Query log is so skewed that great amount of computation is required for most popular words

Partitioning Strategy ★

- TERM-PARTITIONED → Handles subrange of terms
- DOCUMENT-PARTITIONED → Handles subrange of documents ⇓ BETTER LOAD BALANCING

But disadvantage is broadcasting query to all partitions → Network Overhead

★ Partitioning for fault tolerance and memory constraint

Other Real Word Issues :

→ Source format and language detection
→ Sentence and word delimiter → Abreviation vs Full stop
→ Case normalisation (MIT vs mit)
→ Morphological normalization ("stemming")
→ Compound word detection
→ Multilingual dictionary.

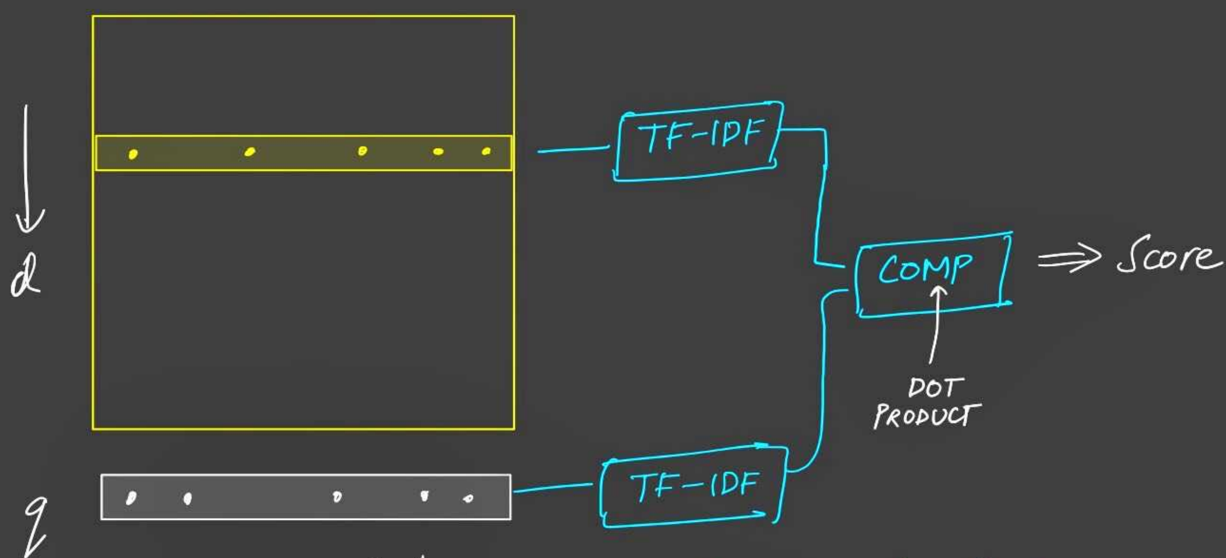
RELEVANCE RANKING:

* Writing precise boolean queries not good for most users

NEED: Average web query ⇒ 2-3 words
Bits of entropy in query = 10 bits (5 bits per word) ~ 1024 items
Corpus > 10 G Docs
Query Result > 10 M hits

Ranked Retrieval ⟶ Between $\wedge$ and $\vee$
* Returns results according to relevance

BOOLEAN SEARCH: Feast or Famine
* Intersection Caching: Performance boost



1-Bit Similarity
Jaccard Coefficient
Cosine similarity
⟶ Different words with diverse information content are all treated at par.

RARITY: ⟶ Encoded in Inverse Document Frequency (IDF) ⟶ Can store in dictionary

TF x IDF: * Rescaling word dimension according to relevance.

$l_w(x) = TF(x,w) \cdot IDF(w)$ Normalise to final vector: $x_w = l_w(x) / L(x)$

$$L(x) = \sqrt{\sum_w l_w(x)}$$

BASIC TFIDF VECTOR SPACE SCORING:

→ Init accumulator map: score [docid]
→ In decreasing order of IDF order of query words get(x, TF(x,w))
score[x] += TF(x,w) * IDF(w)
→ Divide each score by length of score ⇒ COSINE

⟶ Report top k

COMPUTATION TIME

QUERIES → Rare terms
- ★ Accumulator management
- ★ Sparse of dense map?

QUERIES → Frequent terms
- ★ Bit processing for decompressing postings
- ★ Arithmetic to update accumulators
- ★ Wasted effort in computing score to throw away

SCORE / IMPACT ORDERING

→ order postings by decreasing impact

⊛ PROBLEM: Each word will have a different list

QUIT / CONTINUE HEURISTICS

QUIT: Once |Score| exceeds some value, quit and report answers

CONTINUE: Stop creating new accumulators but continue processing for remaining words

TERM IMPACT

Term impact $= \dfrac{TF(x,\omega) * IDF(\omega)}{L(x)}$

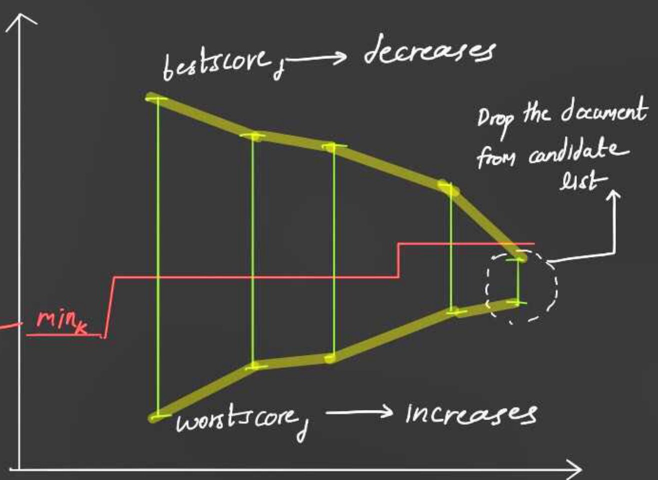
★ Documents sorted with Impact IDs

Q. When people do impact ordered posting list, do they do something clever to encode document ID?

INVERTED INDEX → Doc ID Ordered Posting
- ★ DAAT → Completely score a document before moving to next

INVERTED INDEX → Impact ordered
- ★ TAAT
- → Threshold Algorithm
- → quit / continue

Increases



★★★ Introduce randomness to counter PRESENTATION BIAS

RELEVANCE SUPERVISION

- → Regression
- → Ordinal Regression
- → Complete Rank order
- → Prefix of rank order
- → Pairwise preferences

LOSS AND REWARD:

① L1 or L2 $\quad |\hat{y}-y|^2$ or $|\hat{y}-y|$

y → Gold Score

$\hat{y}$ → Predicted Score

② RELATIVE AGGREGATED GOODNESS

$$\frac{\sum_{v \in \hat{T}_k} y(v)}{\sum_{v \in T_k} y(v)} \in [0,1]$$

③ PAIR PREFERENCE VIOLATION

If $u < v$ but $y(u) > y(v)$

Count number of violations

④ RANK CORRELATION

★ Compute rank correlation with unrealistic ground truth ranking.

⑤ PREFIX RANK CORRELATION

m pairs v, ω from $T_k \cup \hat{T}_k$

Concordant pairs $\Rightarrow (y(v)-y(\omega))(\hat{y}(v)-\hat{y}(\omega)) > 0$

Discordant pairs $\Rightarrow (y(v)-y(\omega))(\hat{y}(v)-\hat{y}(\omega)) < 0$

CONCEPT
- → Precision @ k
- → Recall @ k

Mean Reciprocal Rank

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{r_q}$$

* Dropping from $1 \to 2$ as bad as $2 \to \infty$

** Good for navigation queries

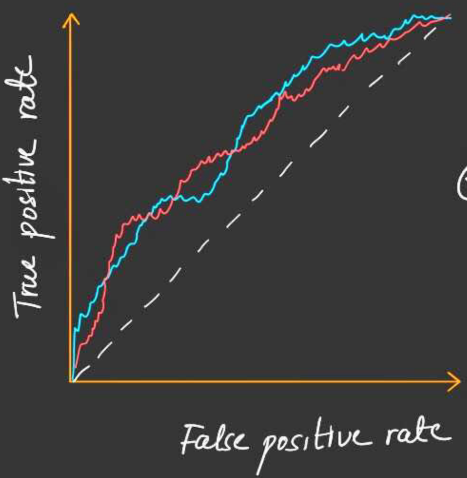
Truncated at rank $k \Rightarrow MRR = \frac{1}{|Q|} \sum_{q \in Q,\ r_q \leq k} (1/r_q)$

* Mean rank is not a good metric.
1 "hard" query can mess up entire score

ROC → Receiver Operating Characteristic Curve

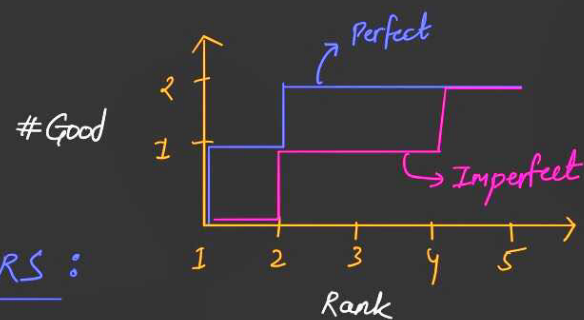
Plotting true positive @ $i = (n_i^+ / n^+)$
vs false positive @ $i = (n_i^- / n^-)$

Good Ranking functions ⇒ Area close to 1
Effectively random ⇒ $1/2$



AUC
* Area under the curve.

Another AUC Example ⇒
Good vs Rank



* Area between 2 plots related to number of Discordant Pairs Q.

Concordant and Discordant Pairs:

Following is the ranking assigned by engine:
$\Rightarrow 1 \leq p_1 < p_2 \cdots p_R$

$r_{engine} = d_1^+, d_2^-, d_3^+, d_4^+, d_5^-, d_6^-, d_7^+, d_8^-$

$r_{ideal} = d_1^+, d_3^+, d_4^+, d_7^+ ; d_2^-, d_5^-, d_6^-, d_8^-$

$$\sum_{i=1}^{R} (p_i - 1) - (i - 1) = Q.$$

$$\Rightarrow \sum_{i=1}^{R} p_i = Q + \frac{R(R+1)}{2}$$

$$= Q + \binom{R+1}{2}$$

Average Precision:

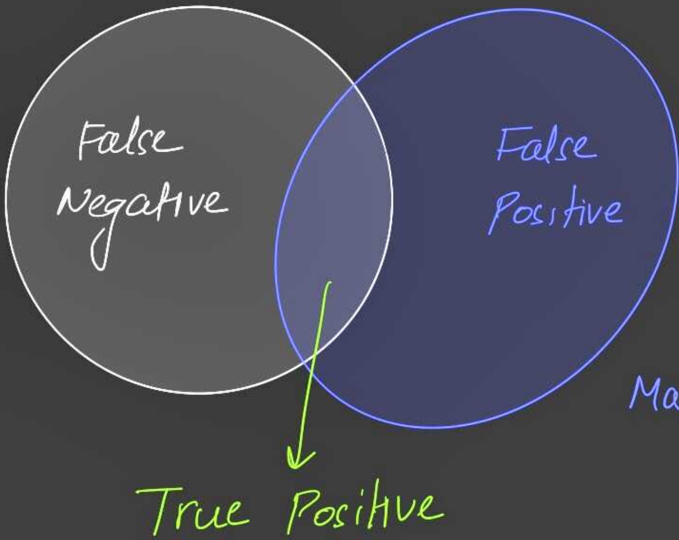
$$Avg\ Prec = \frac{1}{R} \sum_{i=1}^{R} \left(\frac{i}{p_i}\right)$$

* Bounding average precision given Q
→ Minimise average precision subject to Q

$$\mathcal{L}(p_1, \dots p_R ; \lambda) = \frac{1}{R} \sum_{i=1}^{R} \frac{i}{p_i} + \lambda\left(\sum_{i=1}^{R} p_i - Q - \binom{R+1}{2}\right)$$

$$\frac{\partial \mathcal{L}}{\partial p_i} = \frac{-i}{R p_i^2} + \lambda = 0 \Rightarrow p_i^* = \sqrt{\frac{i}{R\lambda}}$$

$$\Rightarrow AvgPrec(r_{engine}, r_{ideal}) \geq \left(\sum_{i=1}^{R} \sqrt{i}\right)^2 / R\left(Q + \binom{R+1}{2}\right)$$



$$Recall = \frac{TP}{FN+TP}$$

$$Precision = \frac{TP}{FP+TP}$$

→ HARMONIC MEAN

$$F_1 = \frac{2RP}{R+P}$$

Maximise F_1 → F_1 Score

NORMALISED DISCOUNTED CUMULATIVE GAIN :

$$NDCG_q = Z_q \underbrace{\sum_{j=1}^{L}}_{\text{CUMULATIVE}} \frac{2^{r_q(j)} - 1}{\log(1+j)}$$

$2^{r_q(j)} - 1$ → GAIN

$r_q(j) \in \{0,1\}$ — 0: IRRELEVANT, 1: RELEVANT

$\log(1+j)$ → RANK DISCOUNT

Z_q → Normalization so that perfect ordering has value 1

WORD EMBEDDINGS :

word → real vector $\mathbb{R}^{300}$

* Similar meaning vectors have higher cosine and low distances

Document ⇒ 300 dimensional vector (DENSE)

Clustering requires LOCALLY SENSITIVE HASH FUNCTION

⇓

** Similarity Search

Q. Why hash functions?

* Tree-based indices do not do well in higher dimensions