Introduction to Information Retrieval

CS276
Information Retrieval and Web Search
Chris Manning and Pandu Nayak
Efficient scoring
Today’s focus

- **Retrieval** – get docs matching query from inverted index
- **Scoring+ranking**
  - Assign a score to each doc
  - Pick $K$ highest scoring docs
- Our emphasis today will be on doing this efficiently, rather than on the quality of the ranking
Background

- Score computation is a large (10s of %) fraction of the CPU work on a query
  - Generally, we have a tight budget on latency (say, 250ms)
  - CPU provisioning doesn’t permit exhaustively scoring every document on every query
- Today we’ll look at ways of cutting CPU usage for scoring, without compromising the quality of results (much)
- Basic idea: avoid scoring docs that won’t make it into the top $K$
Recap: Queries as vectors

- Vector space scoring
  - We have a weight for each term in each doc
  - Represent queries as vectors in the space
  - Rank documents according to their cosine similarity to the query in this space
    - Or something more complex: BM25, proximity, ...

- Vector space scoring is
  - Entirely query dependent
  - Additive on term contributions – no conditionals etc.
  - Context insensitive (no interactions between query terms)
TAAT vs DAAT techniques

- **TAAT** = “Term At A Time”
  - Scores for all docs computed concurrently, one query term at a time

- **DAAT** = “Document At A Time”
  - Total score for each doc (incl all query terms) computed, before proceeding to the next

- Each has implications for how the retrieval index is structured and stored
Efficient cosine ranking

- Find the $K$ docs in the collection “nearest” to the query $\Rightarrow K$ largest query-doc cosines.

Efficient ranking:
- Choosing the $K$ largest cosine values efficiently.
  - Can we do this without computing all $N$ cosines?
Safe vs non-safe ranking

- The terminology “safe ranking” is used for methods that guarantee that the $K$ docs returned are the $K$ absolute highest scoring documents
  - (Not necessarily just under cosine similarity)
- Is it ok to be non-safe?
- If it is – then how do we ensure we don’t get too far from the safe solution?
  - How do we measure if we are far?
Non-safe ranking

- Covered in depth in Coursera video (number 7)
- Non-safe ranking may be okay
  - Ranking function is only a proxy for user happiness
  - Documents close to top K may be just fine
- Index elimination
  - Only consider high-idf query terms
  - Only consider docs containing many query terms
- Champion lists
- High/low lists, tiered indexes
- Order postings by $g(d)$ (query-indep. quality score)
SAFE RANKING
Safe ranking

- When we output the top \( K \) docs, we have a proof that these are indeed the top \( K \)
- Does this imply we always have to compute all \( N \) cosines?
  - We’ll look at pruning methods
  - So we only fully score some \( J \) documents
- Do we have to sort the \( J \) cosine scores?
Computing the $K$ largest cosines: selection vs. sorting

- Typically we want to retrieve the top $K$ docs (in the cosine ranking for the query)
  - not to totally order all docs in the collection
- Can we pick off docs with $K$ highest cosines?
- Let $J$ = number of docs with nonzero cosines
  - We seek the $K$ best of these $J$
Use heap for selecting top $K$

- Binary tree in which each node’s value > the values of children
- Takes $2J$ operations to construct, then each of $K$ “winners” read off in $O(\log J)$ steps.
- For $J=1M$, $K=100$, this is about 10% of the cost of sorting.
WAND scoring

- An instance of DAAT scoring
- Basic idea reminiscent of branch and bound
  - We maintain a running *threshold* score – e.g., the $K^{th}$ highest score computed so far
  - We prune away all docs whose cosine scores are guaranteed to be below the threshold
  - We compute exact cosine scores for only the un-pruned docs

Index structure for WAND

- Postings ordered by docID
- Assume a special iterator on the postings of the form “go to the first docID greater than or equal to $X$”
- Typical state: we have a “finger” at some docID in the postings of each query term
  - Each finger moves only to the right, to larger docIDs
- Invariant – all docIDs lower than any finger have already been *processed*, meaning
  - These docIDs are either pruned away or
  - Their cosine scores have been computed
Upper bounds

- At all times for each query term \( t \), we maintain an upper bound \( UB_t \) on the score contribution of any doc to the right of the finger
  - Max (over docs remaining in \( t \)’s postings) of \( w_t(doc) \)

As finger moves right, \( UB \) drops
Pivoting

- Query: *catcher in the rye*
- Let’s say the current finger positions are as below

```
catcher: 273
rye: 304
in: 589
the: 762
```

Threshold = 6.8

- \( UB_{catcher} = 2.3 \)
- \( UB_{rye} = 1.8 \)
- \( UB_{in} = 3.3 \)
- \( UB_{the} = 4.3 \)
Prune docs that have no hope

- Terms sorted in order of finger positions
- Move fingers to 589 or right

**catcher**
- UB\(_{catcher}\) = 2.3
- Threshold = 6.8

**rye**
- UB\(_{rye}\) = 1.8

**in**
- UB\(_{in}\) = 3.3

**the**
- UB\(_{the}\) = 4.3
- Update UB’s
Compute 589’s score if need be

- If 589 is present in enough postings, compute its full cosine score – else some fingers to right of 589
- Pivot again ...

```plaintext
catcher
rye
in
the
```
WAND summary

- In tests, WAND leads to a 90+% reduction in score computation
  - Better gains on longer queries
- Nothing we did was specific to cosine ranking
  - We need scoring to be additive by term
- WAND and variants give us safe ranking
  - Possible to devise “careless” variants that are a bit faster but not safe (see summary in Ding+Suel 2011)
  - Ideas combine some of the non-safe scoring we considered