Efficient Cosine Ranking

- Find the k docs in the corpus "nearest" to the query
  - the k largest query-doc cosines
- Efficient ranking means:
  - Computing a single cosine efficiently
  - Computing the k largest cosine values efficiently
    - Can we do this without computing all n cosines?
      - n = number of documents in corpus

Efficient Cosine Ranking

- Computing a single cosine
  - Use inverted index
  - At query time use an array of accumulators Aj to accumulate component-wise sum
  - Accumulate scores as postings lists are being processed (numerator of similarity score)

$$A_j = \sum_t (w_{q,t} w_{d,t})$$

Efficient Cosine Ranking

- For the web
  - an array of accumulators in memory is infeasible
  - so only create accumulators for docs that occur in postings list
    - dynamically create accumulators
  - put the tf_d scores in the postings lists themselves
  - limit docs to non-zero cosines on rare words
    - or non-zero cosines on all words
    - reduces number of accumulators

Efficient Cosine Ranking

$\textsc{CosineScore}(q)$

$\quad \textsc{Initialize}(Scores[d \in D])$

$\quad \textsc{Initialize}(Magnitude[d \in D])$

$\quad$ **for** $each\ term(t \in q)$

$\quad\quad$ **do** $p \leftarrow \textsc{FetchPostingsList}(t)$

$\quad\quad\quad df_t \leftarrow \textsc{GetCorpusWideStats}(p)$

$\quad\quad\quad \alpha_{t,q} \leftarrow \textsc{WeightInQuery}(t, q, df_t)$

$\quad\quad\quad$ **for** $each\ \{d, tf_{t,d}\} \in p$

$\quad\quad\quad\quad$ **do** $Scores[d] += \alpha_{t,q} \cdot \textsc{WeightInDocument}(t, q, df_t)$

$\quad$ **for** $d \in Scores$

$\quad\quad$ **do** $\textsc{Normalize}(Scores[d], Magnitude[d])$

$\quad$ **return** $top\ K \in Scores$

Use heap for selecting the top K Scores

- Binary tree in which each node's value > the values of children
- Takes 2N operations to construct
  - then each of k "winners" read off in 2logn steps
  - For n =1M, k=100 this is about 10% of the cost of sorting