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
Vector Space Scoring

Efficient Cosine Ranking

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

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

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**

**CosineScore(q)**

1. **Initialize**(Scores[d ∈ D])
2. **Initialize**(Magnitude[d ∈ D])
3. for each term(t ∈ q)
   
   do \( p ← \text{FetchPostingsList}(t) \)

4. \( df_t ← \text{GetCorpusWideStats}(p) \)
5. \( \alpha_{t,q} ← \text{WeightInQuery}(t, q, df_t) \)
6. for each \( \{d, tf_{t,d}\} ∈ p \)
   
   do \( \text{Scores}[d] + = \alpha_{t,q} \cdot \text{WeightInDocument}(t, q, df_t) \)

7. for \( d ∈ \text{Scores} \)
   
   do \( \text{Normalize}((\text{Scores}[d], \text{Magnitude}[d])) \)

8. return top \( K ∈ \text{Scores} \)
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 $2\log n$ steps
  - For $n = 1M$, $k=100$ this is about 10% of the cost of sorting