Vector Space Scoring

Introduction to Information Retrieval
INF 141
Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org

Alternatives to tf-idf

- Sublinear tf scaling
  - 20 occurrences of “mole” does not indicate 20 times the relevance
  - This motivated the WTF score.

$$
\begin{array}{l}
\mathrm{WTF}(t,d) \\
\quad \textbf{if } tf_{t,d} = 0 \\
\qquad \textbf{then } return(0) \\
\qquad \textbf{else } \; return(1 + log(tf_{t,d}))
\end{array}
$$

  - There are other variants for reducing the impact of repeated terms

TF Normalization

- Normalize tf weights by maximum tf in that document

$$ntf_{t,d} = \alpha + (1 - \alpha)\frac{tf_{t,d}}{tf_{max}(d)}$$

  - alpha is a smoothing term from (0 - 1.0) ~0.4 in practice
  - This addresses a length bias.
  - Take one document, repeat it, WTF goes up

TF Normalization

- Normalize tf weights by maximum tf in that document

$$ntf_{t,d} = \alpha + (1 - \alpha)\frac{tf_{t,d}}{tf_{max}(d)}$$

  - a change in the stop word list can change weights drastically - hard to tune
  - still based on bag of words model
    - one outlier word, repeated many times might throw off the algorithmic understanding of the content

Vector Space Scoring : Alternatives to tf-idf

Laundry List

Term Frequency		*Document Frequency*		*Normalization*	
$(n)atural$	$tf_{t,d}$	$(n)o$	1	$(n)one$	1
$(l)ogarithm$	$1 + log(tf_{t,d})$	$(t)idf$	$log\frac{\lvert corpus\rvert}{df_t}$	$(c)osine$	$\frac{1}{\sqrt{w_1{}^2+w_2{}^2+\ldots+w_m{}^2}}$
$(a)ugmented$	$\alpha + (1-\alpha)\frac{tf_{t,d}}{tf_{max}(d)}$	$(p)robidf$	$max\{0, log(\frac{\lvert corpus\rvert - dft}{df_t})$	$(u)pivoted$	$1/u$
$(b)oolean$	$tf_{t,d} > 0?1 : 0$			$(b)yte$	$1/CharLength^{\alpha}, \alpha < 1$
$(L)ogaverage$	$\frac{1+log(tf_{t,d})}{1+log(ave_{t\in d}(tf_{t,d}))}$				

- SMART system of describing your IR vector algorithm
  - ddd.qqq (ddd = document weighting) (qqq = query weighting)
  - first is term weighting, second is document, then normalization
  - lnc.ltc is what?

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

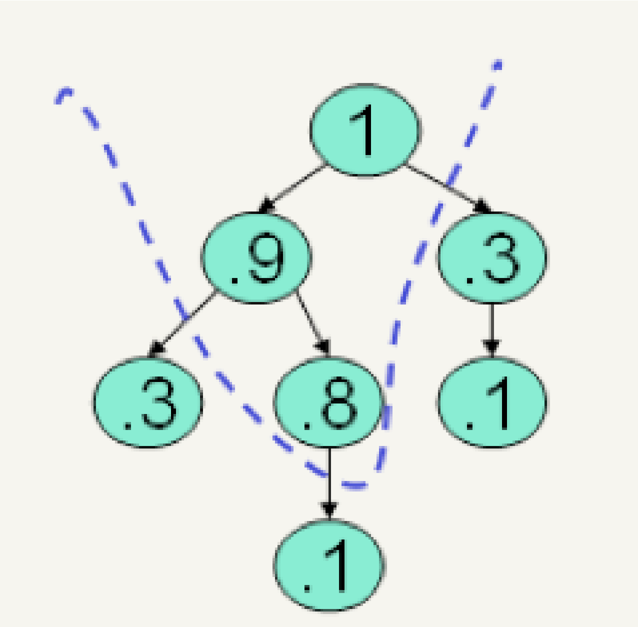
- For the web
  - an array of accumulators in memory is infeasible
  - so only create accumulators for docs that occur in postings list
    - dynamically create accumulators
  - put the tf_d scores in the postings lists themselves
  - limit docs to non-zero cosines on rare words
    - or non-zero cosines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

Efficient Cosine Ranking

```
CosineScore(q)
Initialize(Scores[d ∈ D])
Initialize(Magnitude[d ∈ D])
for each term(t ∈ q)
    do p ← FetchPostingsList(t)
       df_t ← GetCorpusWideStats(p)
       α_{t,q} ← WeightInQuery(t, q, df_t)
       for each {d, tf_{t,d}} ∈ p
           do Scores[d] += α_{t,q} · WeightInDocument(t, q, df_t)
for d ∈ Scores
    do Normalize(Scores[d], Magnitude[d])
return top K ∈ 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



LUCI