# VECTOR SPACE SCORING

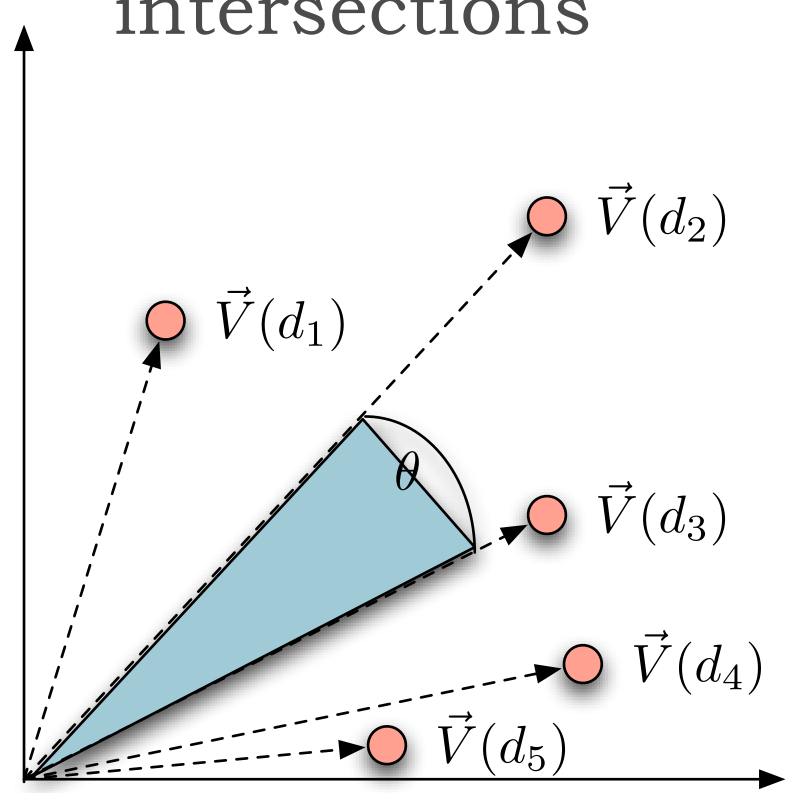
Introduction to
Information Retrieval
CS 150
Donald J. Patterson

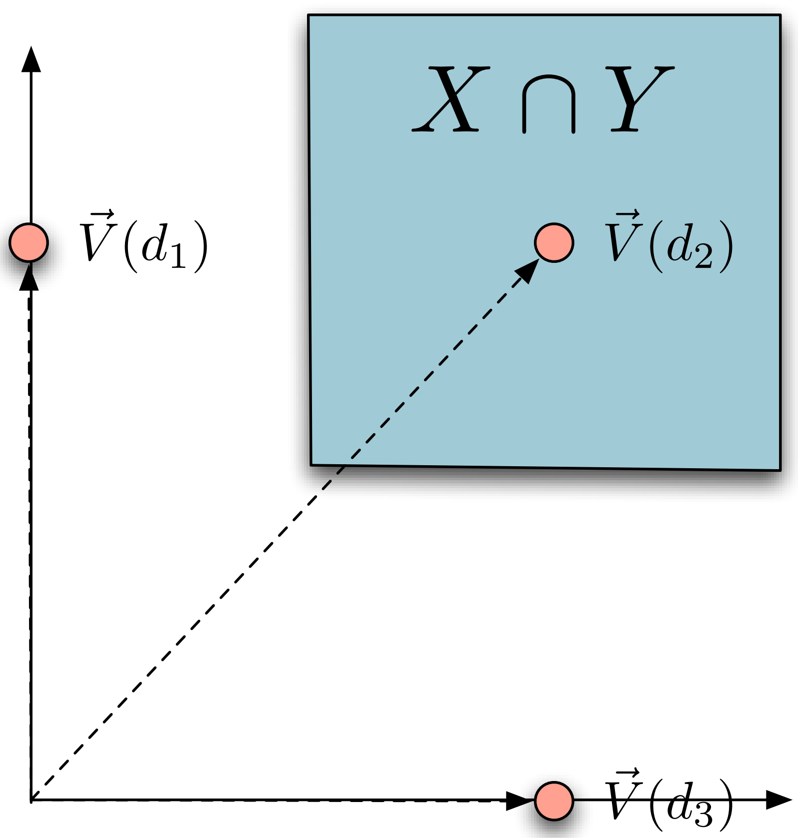
Content adapted from Hinrich Schütze
http://www.informationretrieval.org

# VECTOR SPACE SCORING

## VECTORS AND BOOLEAN QUERIES

- Ranked queries and Boolean queries don't work very well together
  - In term space
    - ranked queries select based on sector containment - cosine similarity
    - boolean queries select based on rectangle unions and intersections



# VECTOR SPACE SCORING

## VECTORS AND WILD CARDS

- How could we work with the query, “quick* print*” ?
  - Can we view this as a bag of words?
  - What about expanding each wild-card into the matching set of dictionary terms?
- Danger: Unlike the boolean case, we now have tfs and idfs to deal with
- Overall, not a great idea

# VECTOR SPACE SCORING

## VECTORS AND OTHER OPERATORS

- Vector space queries are good for no-syntax, bag-of-words queries
  - Nice mathematical formalism
  - Clear metaphor for similar document queries
  - Doesn't work well with Boolean, wild-card or positional query operators
  - But …

# QUERY LANGUAGE VS. SCORING

- Interfaces to the rescue
  - Free text queries are often separated from operator query language
  - Default is free text query
  - Advanced query operators are available in “advanced query” section of interface
  - Or embedded in free text query with special syntax
    - aka -term -”terma termb”

## ALTERNATIVES TO TF-IDF

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

$$\mathrm{WTF}(t, d)$$

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

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

# VECTOR SPACE SCORING

## 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
    - this score reduces that impact

# VECTOR SPACE SCORING

## 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
## LAUNDRY LIST

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

- 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
  - ltc.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 (incremental dot-product)
  - 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 tfidf 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

# VECTOR SPACE SCORING

## EFFICIENT COSINE RANKING

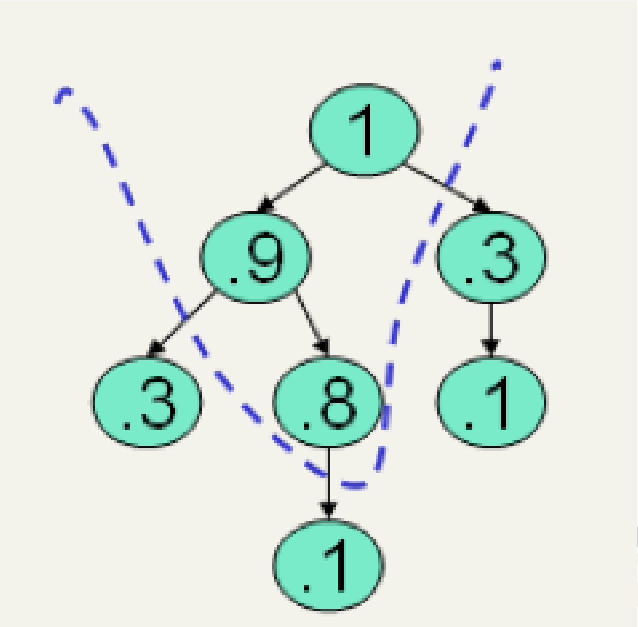
$\textsc{CosineScore}(q)$

$\quad \textsc{Initialize}(Scores[d \in D])$
$\quad \textsc{Initialize}(Magnitude[d \in D])$
$\quad \textbf{for}\ each\ term(t \in q)$
$\qquad \textbf{do}\ p \leftarrow \textsc{FetchPostingsList}(t)$
$\qquad\quad df_t \leftarrow \textsc{GetCorpusWideStats}(p)$
$\qquad\quad \alpha_{t,q} \leftarrow \textsc{WeightInQuery}(t, q, df_t)$
$\qquad\quad \textbf{for}\ each\ \{d, tf_{t,d}\} \in p$
$\qquad\qquad \textbf{do}\ Scores[d] += \alpha_{t,q} \cdot \textsc{WeightInDocument}(t, q, df_t)$
$\quad \textbf{for}\ d \in Scores$
$\qquad \textbf{do}\ \textsc{Normalize}(Scores[d], Magnitude[d])$
$\quad \textbf{return}\ top\ K \in Scores$

# VECTOR SPACE SCORING

## 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
- Java "TreeMap" for example



WESTMONT INSPIRED

COMPUTING LAB