Correction!!!

Clarification of TF-IDF score

- Some of the slides show this formula:

$$tfidf(t,d) \quad = \quad (1 + log(tf_{t,d})) * log\left(\frac{|corpus|}{df_{t,d}}\right)$$

- Precisely it should be:

$$tfidf(t,d) \quad = \quad WTF(t,d) * log\left(\frac{|corpus|}{df_{t,d}}\right)$$

- The difference is just the special case when tf = 0

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

if $tf_{t,d} = 0$

  then $return(0)$

  else $return(1 + log(tf_{t,d}))$

Queries in the vector space model

- Central idea: the query is a vector
  - We regard the query as a short document
  - We return the documents ranked by the closeness of their vectors to the query (also a vector)

$$sim(q, d_i) = \frac{\vec{V}(q) \cdot \vec{V}(d_i)}{|\vec{V}(q)||\vec{V}(d_i)|}$$

  - Note that q is very sparse!

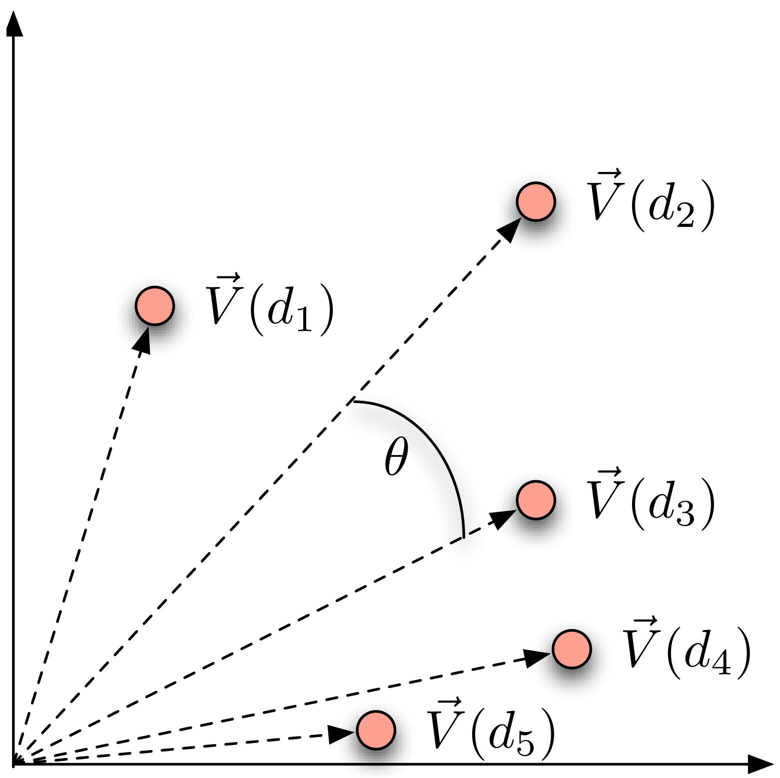
Cosine Similarity Score

- Also called cosine similarity

$$\vec{V}(d_1) \cdot \vec{V}(d_2) = \frac{|\vec{V}(d_1)||\vec{V}(d_2)|}{cos(\theta)}$$

$$cos(\theta) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$

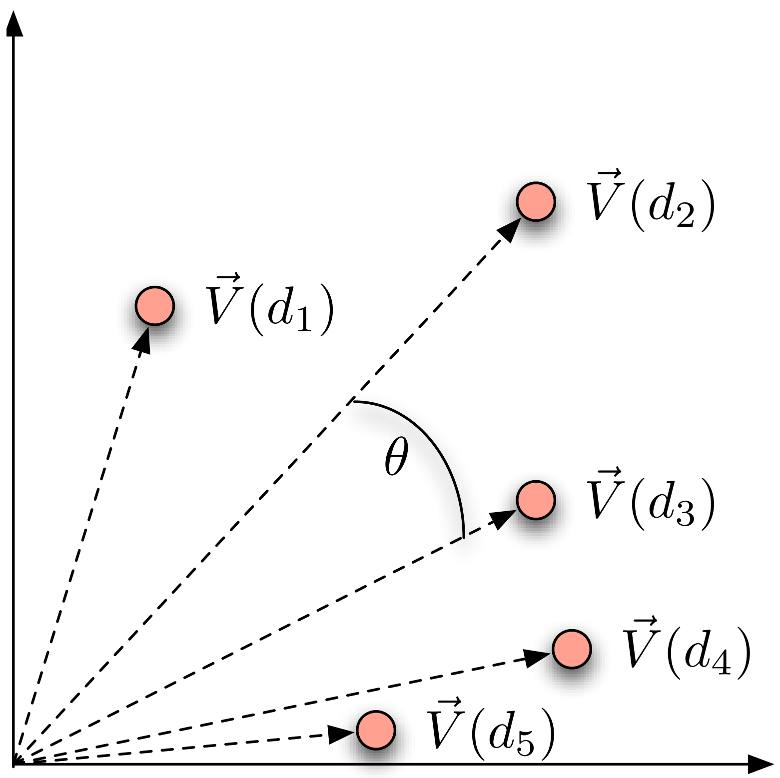


Cosine Similarity Score

- Define: dot product

$$\vec{V}(d_1) \cdot \vec{V}(d_2) = \sum_{i=t_1}^{t_n} (\vec{V}(d_1)_i \vec{V}(d_2)_i)$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0



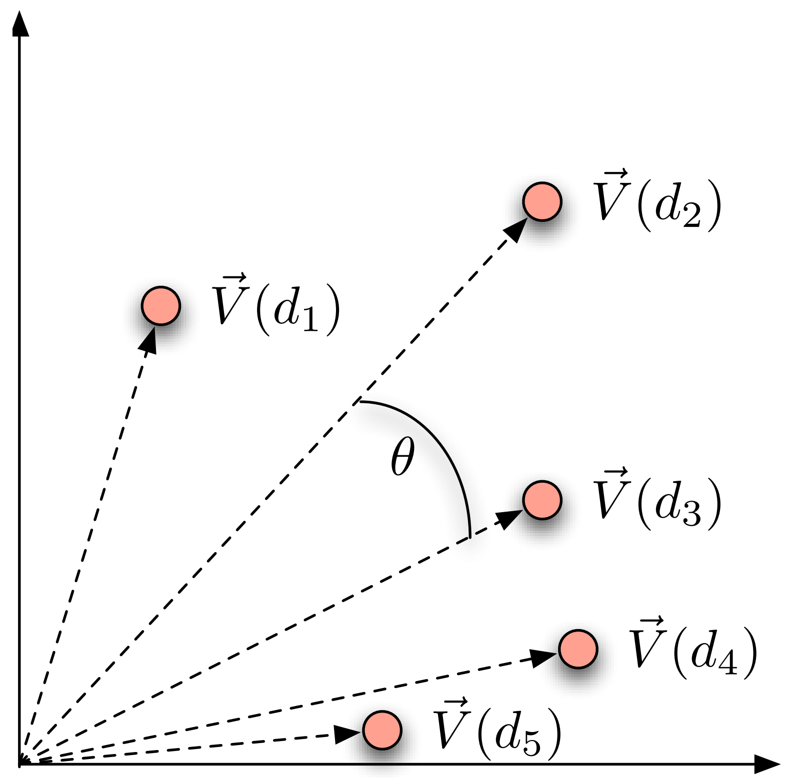
$$\begin{aligned} \vec{V}(d_1) \cdot \vec{V}(d_2) &= (13.1 * 11.4) + (3.0 * 8.3) + (2.3 * 2.3) + (0 * 11.2) + (17.7 * 0) + (0.5 * 0) + (1.2 * 0) \\ &= 179.53 \end{aligned}$$

Cosine Similarity Score

- Define: Euclidean Length

$$|\vec{V}(d_1)| \quad = \quad \sqrt{\sum_{i=t_1}^{t_n} (\vec{V}(d_1)_i \vec{V}(d_1)_i)}$$

	Antony and Cleopatra	*Julius Caesar*	*The Tempest*	*Hamlet*	*Othello*	*Macbeth*
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0



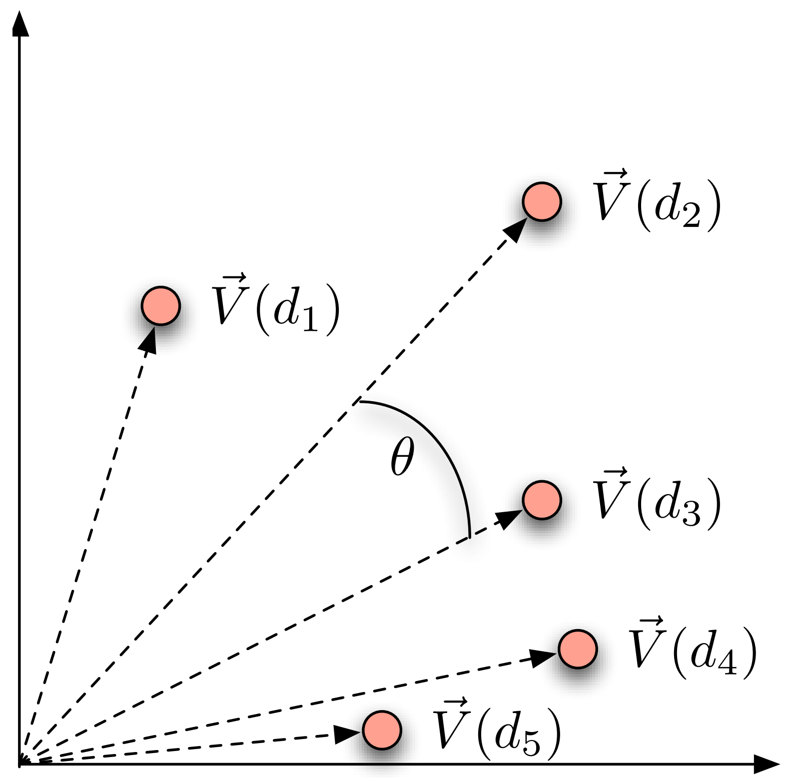
$$\begin{aligned} |\vec{V}(d_1)| &= \sqrt{(13.1 * 13.1) + (3.0 * 3.0) + (2.3 * 2.3) + (17.7 * 17.7) + (0.5 * 0.5) + (1.2 * 1.2)} \\ &= 22.38 \end{aligned}$$

Cosine Similarity Score

- Define: Euclidean Length

$$|\vec{V}(d_1)| = \sqrt{\sum_{i=t_1}^{t_n} (\vec{V}(d_1)_i \vec{V}(d_1)_i)}$$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

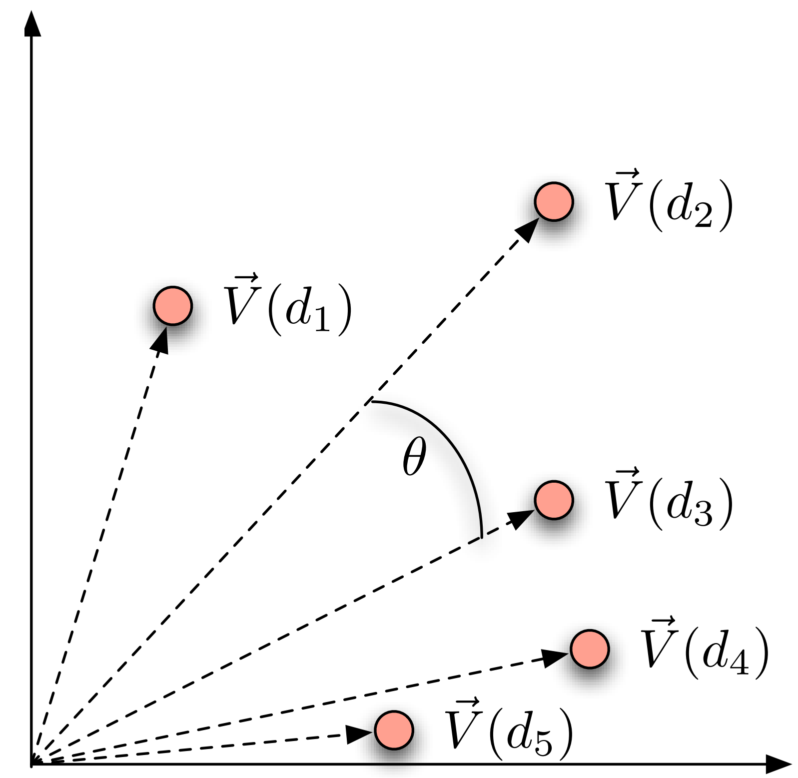


$$|\vec{V}(d_1)| = \sqrt{(11.4 * 11.4) + (8.3 * 8.3) + (2.3 * 2.3) + (11.2 * 11.2)}$$
$$= 18.15$$

Cosine Similarity Score

- Example

$$
\begin{aligned}
sim(d_1, d_2) &= \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|} \\
&= \frac{179.53}{22.38 * 18.15} \\
&= 0.442
\end{aligned}
$$



Exercise

- Rank the following by decreasing cosine similarity.
  - Assume tf-idf weighting:
    - Two docs that have only frequent words in common
      - (the, a , an, of)
    - Two docs that have no words in common
    - Two docs that have many rare words in common
      - (mocha, volatile, organic, shade-grown)

Spamming indices

- This was invented before spam
- Consider:
  - Indexing a sensible passive document collection
  - vs.
  - Indexing an active document collection, where people, companies, bots are shaping documents to maximize scores
- Vector space scoring may not be as useful in this context.

Interaction: vectors and phrases

- Scoring phrases doesn’t naturally fit into the vector space world:
  - How do we get beyond the “bag of words”?
  - “dark roast” and “pot roast”
  - There is no information on “dark roast” as a phrase in our indices.
- Biword index can treat some phrases as terms
  - postings for phrases
  - document wide statistics for phrases

Interaction: vectors and phrases

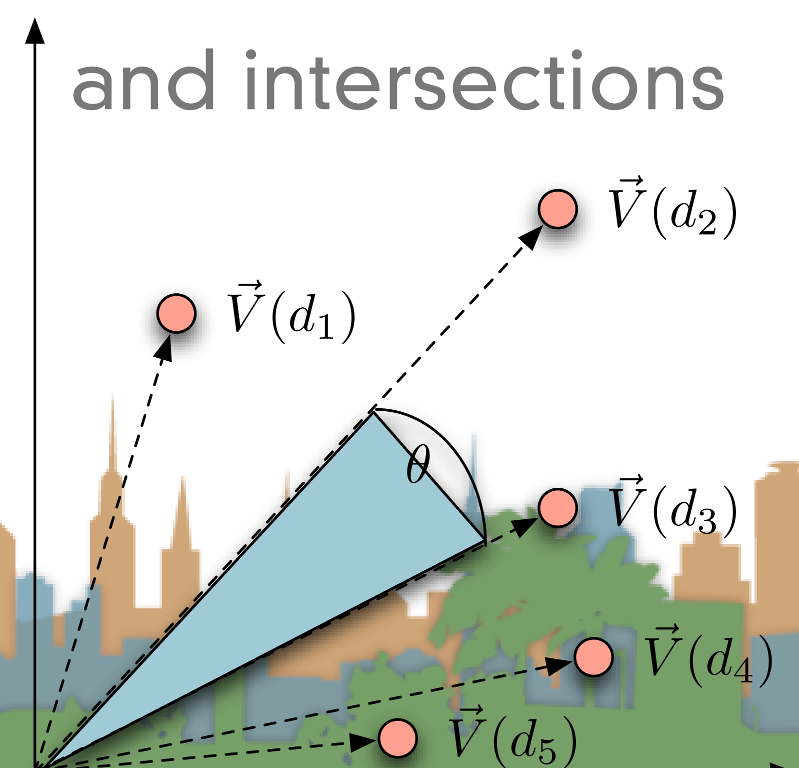
- Theoretical problem:
  - Axes of our term space are now correlated
    - There is a lot of shared information in “light roast” and “dark roast” rows of our index
- End-user problem:
  - A user doesn’t know which phrases are indexed and can more effectively discriminate results.

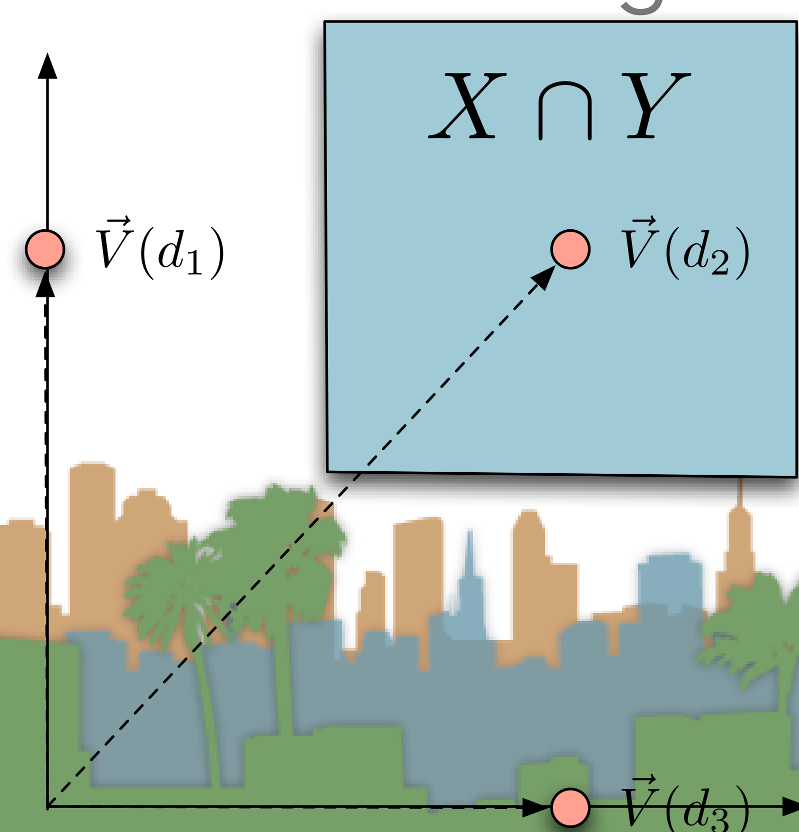
Multiple queries for phrases and vectors

- Query: “rising interest rates”
- Iterative refinement:
  - Run the phrase query vector with 3 words as a term.
  - If not enough results, run 2-phrase queries and fold into results: “rising interest” “interest rates”
  - If still not enough results run query with three words as separate terms.

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





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

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.

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

  - 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 wieghts 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?