Vector Space Scoring

Introduction to Information Retrieval
INF 141/ CS 121
Donald J. Patterson

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

Vector Space Model

- Define: Vector Space Model
  - Representing a set of documents as vectors in a common vector space.
  - It is fundamental to many operations
    - (query,document) pair scoring
    - document classification
    - document clustering
  - Queries are represented as a document
    - A short one, but mathematically equivalent

Vector Space Model

- Define: Vector Space Model
  - A document, d, is defined as a vector: $\vec{V}(d)$
    - One component for each term in the dictionary
    - Assume the term is the tf-idf score

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

- A corpus is many vectors together.
- A document can be thought of as a point in a multi-dimensional space, with axes related to terms.

Vector Space Model

- Recall our Shakespeare Example:

	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	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

$\vec{V}(d_6)_7$

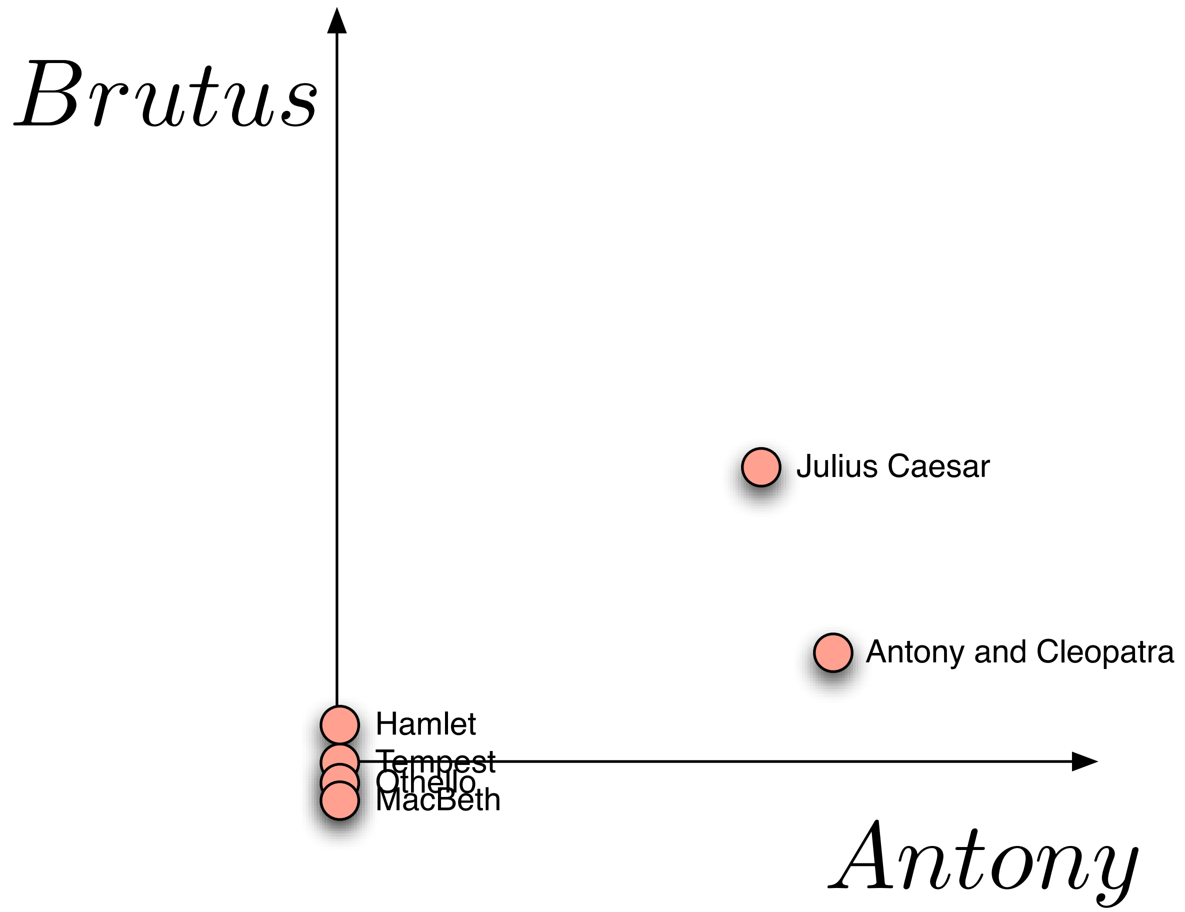
Vector Space Model

- Recall our Shakespeare Example:

	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	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

Vector Space Model

- Recall our Shakespeare Example:



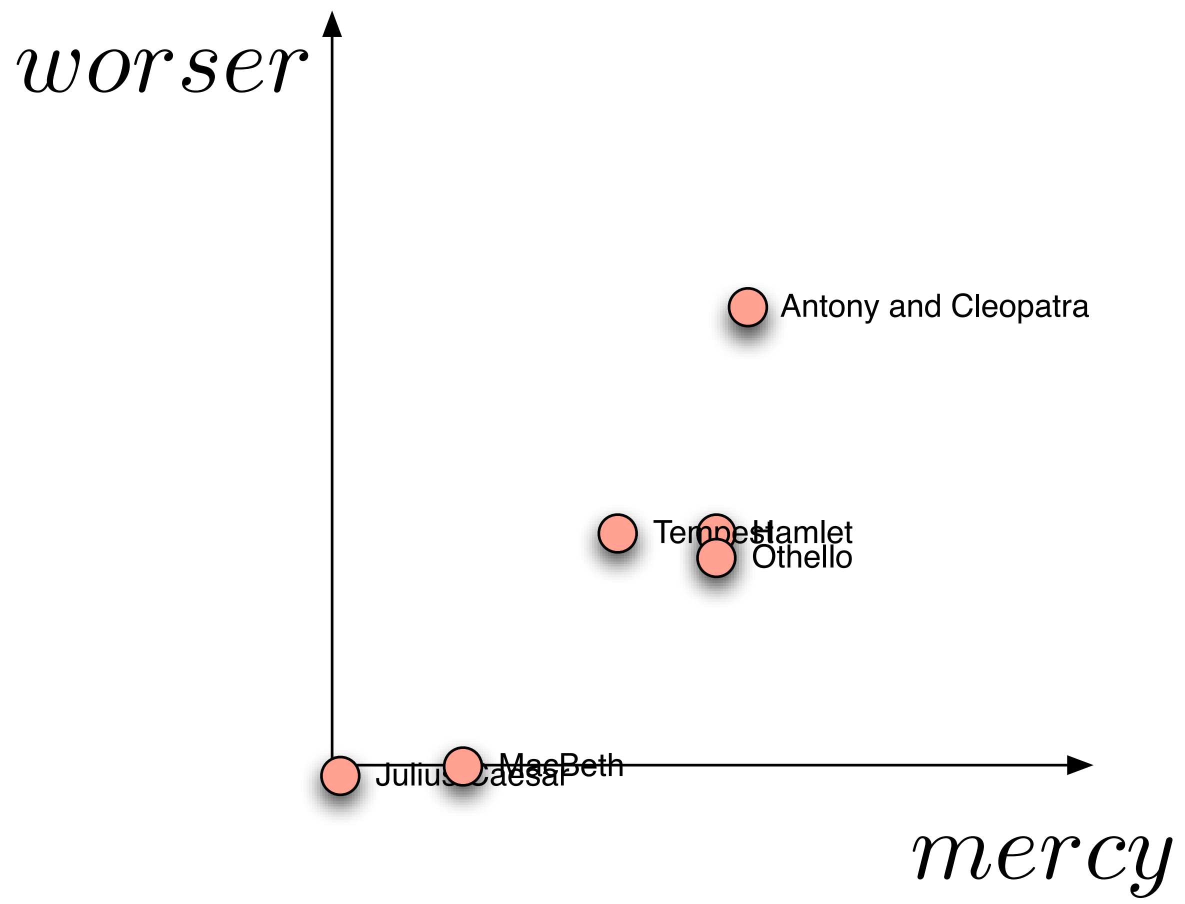
Vector Space Model

- Recall our Shakespeare Example:

	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	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

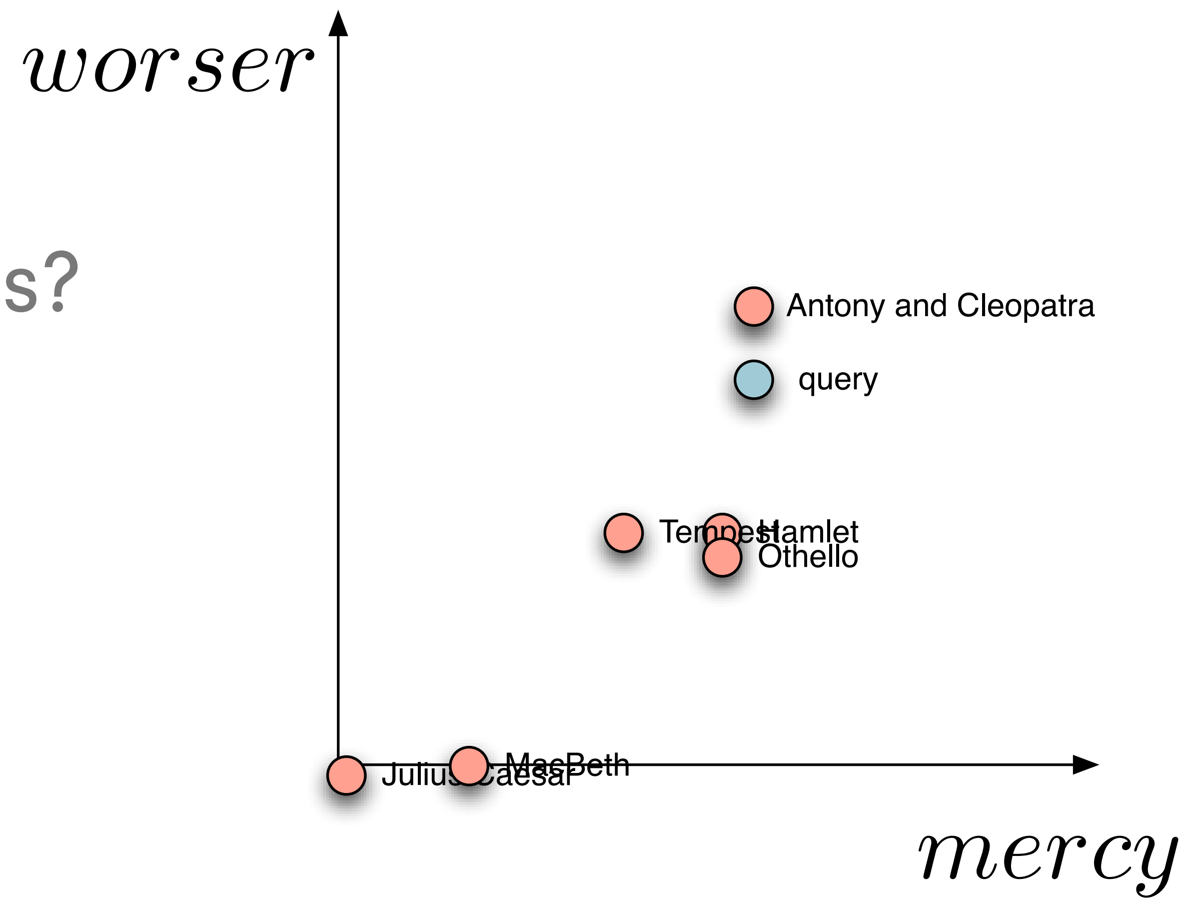
Vector Space Model

- Recall our Shakespeare Example:



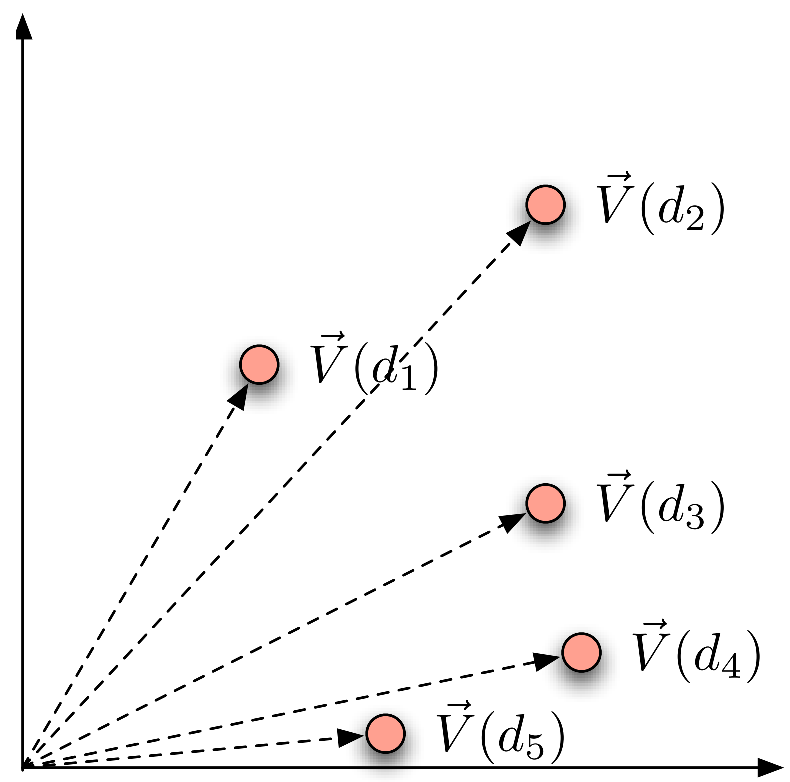
Query as a vector

- So a query can also be plotted in the same space
  - "worser mercy"
  - To score, we ask:
    - How similar are two points?
  - How to answer?



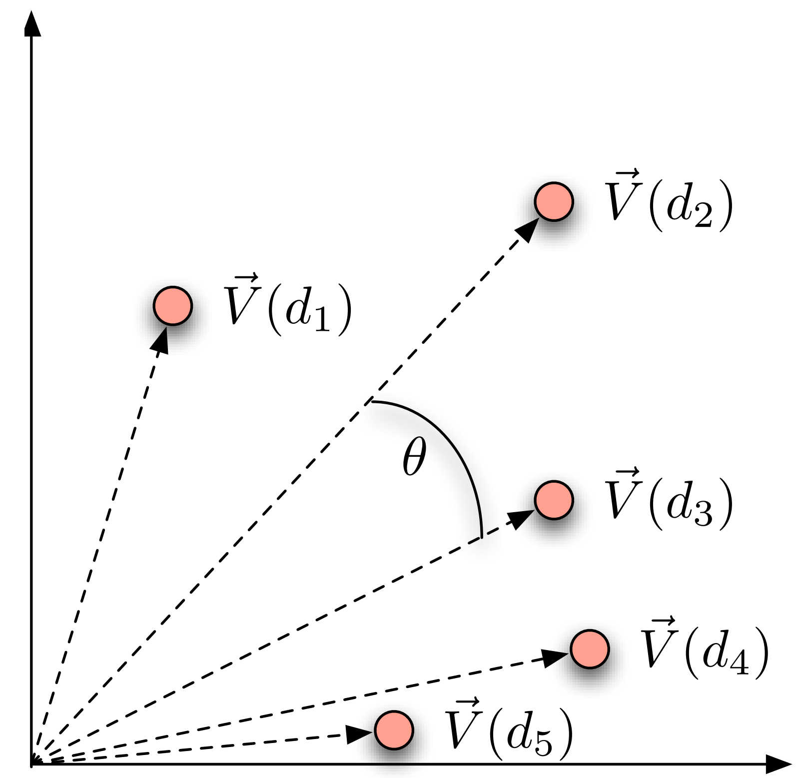
Score by magnitude

- How to answer?
  - Similarity of magnitude?
    - But, two documents, similar in content, different in length can have large differences in magnitude.



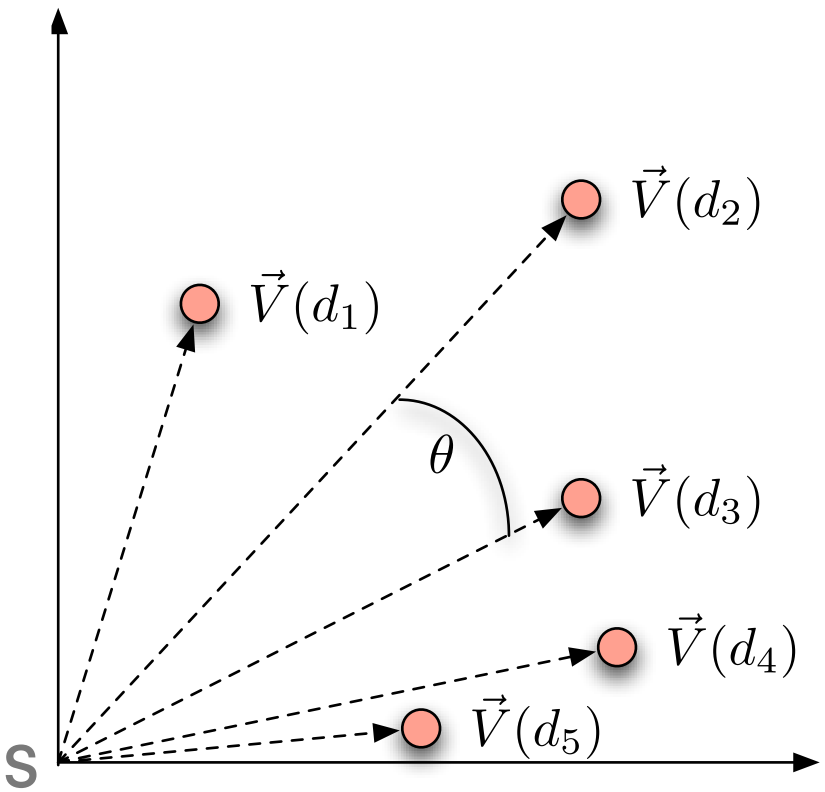
Score by angle

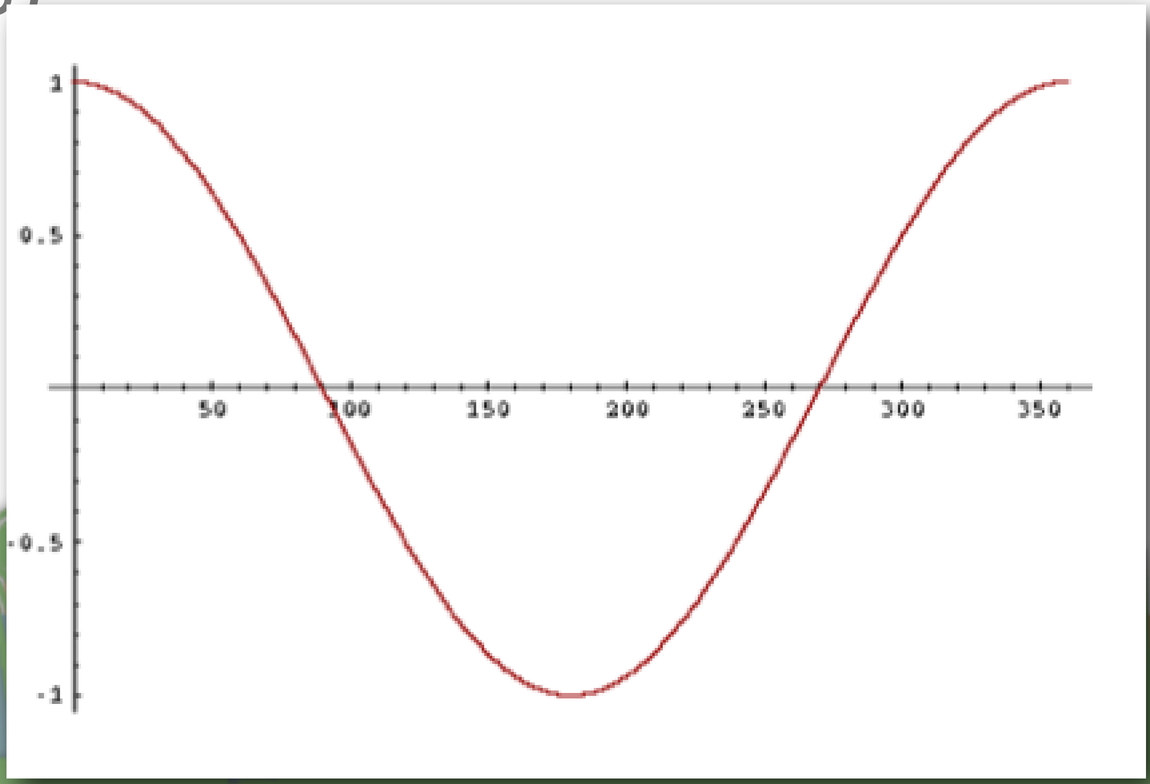
- How to answer?
  - Similarity of relative positions, or
  - difference in angle
    - Two documents are similar if the angle between them is 0.
    - As long as the ratios of the axes are the same, the documents will be scored as equal.
    - This is measured by the dot product



Score by angle

- Rather than use angle
  - use cosine of angle
  - When sorting cosine and angle are equivalent
  - Cosine is monotonically decreasing as a function of angle over (0 ... 180)





Big picture

- Why are we turning documents and queries into vectors
  - Getting away from Boolean retrieval
  - Developing ranked retrieval methods
  - Developing scores for ranked retrieval
  - Term weighting allows us to compute scores for document similarity
  - Vector space model is a clean mathematical model to work with

Big picture

- Cosine similarity measure
  - Gives us a symmetric score
    - if d_1 is close to d_2, d_2 is close to d_1
  - Gives us transitivity
    - if d_1 is close to d_2, and d_2 close to d_3, then
    - d_1 is also close to d_3
  - No document is closer to d_1 than itself
  - If vectors are normalized (length = 1) then
    - The similarity score is just the dot product (fast)

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!

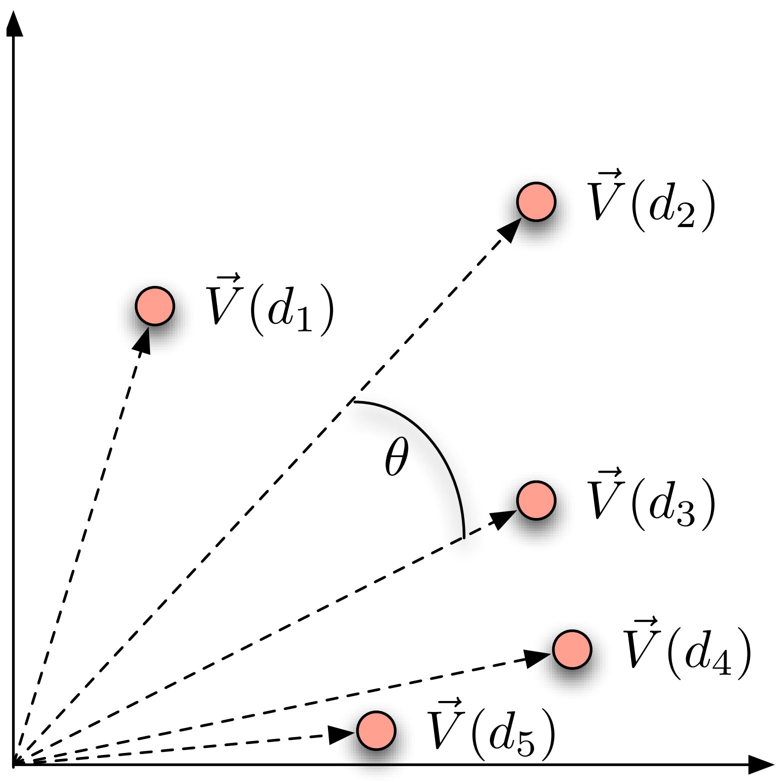
Cosine Similarity Score

- Also called cosine similarity

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

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

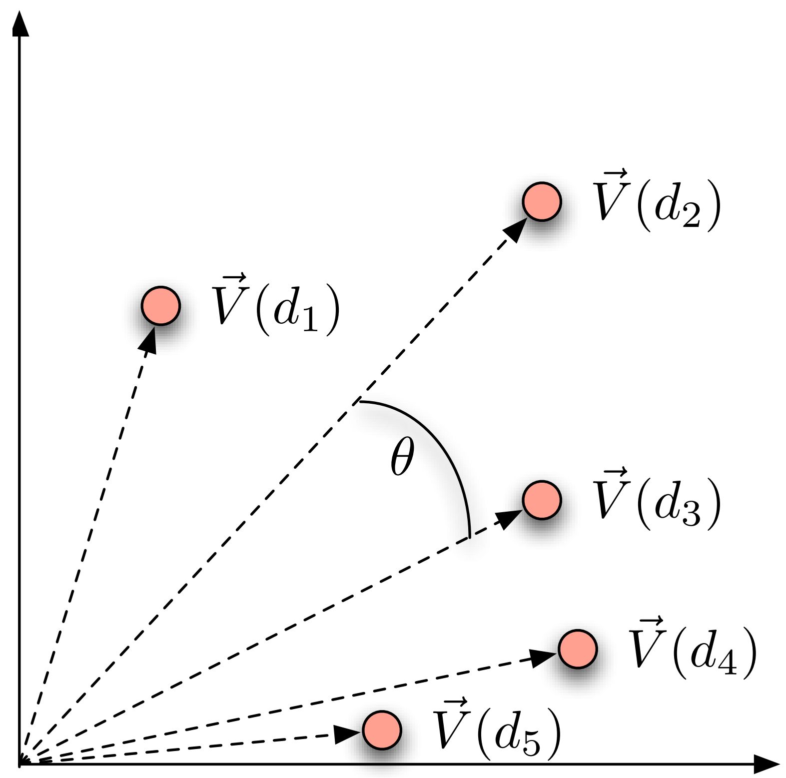
$$sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|}$$



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

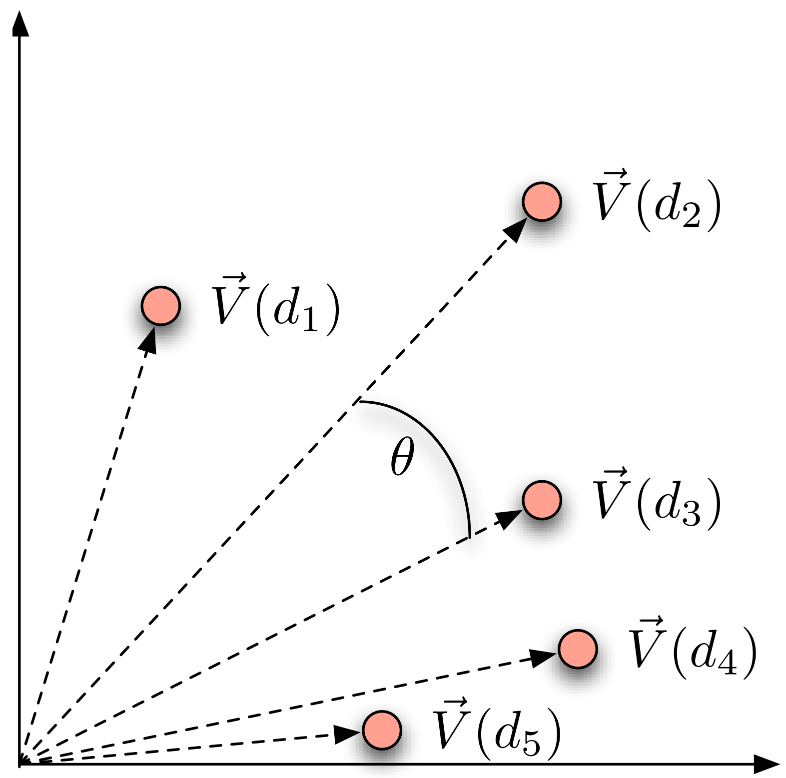
$$\begin{aligned} \vec{V}(d_1) \cdot \vec{V}(d_2) &= (13.1 * 11.4) + (3.0 * 8.3) + (2.3 * 2.3) + (0 * 11.2) + (17.7 * 0) + (0.5 * 0) + (1.2 * 0) \\ &= 179.53 \end{aligned}$$

Cosine Similarity Score

- Define: Euclidean Length

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

	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



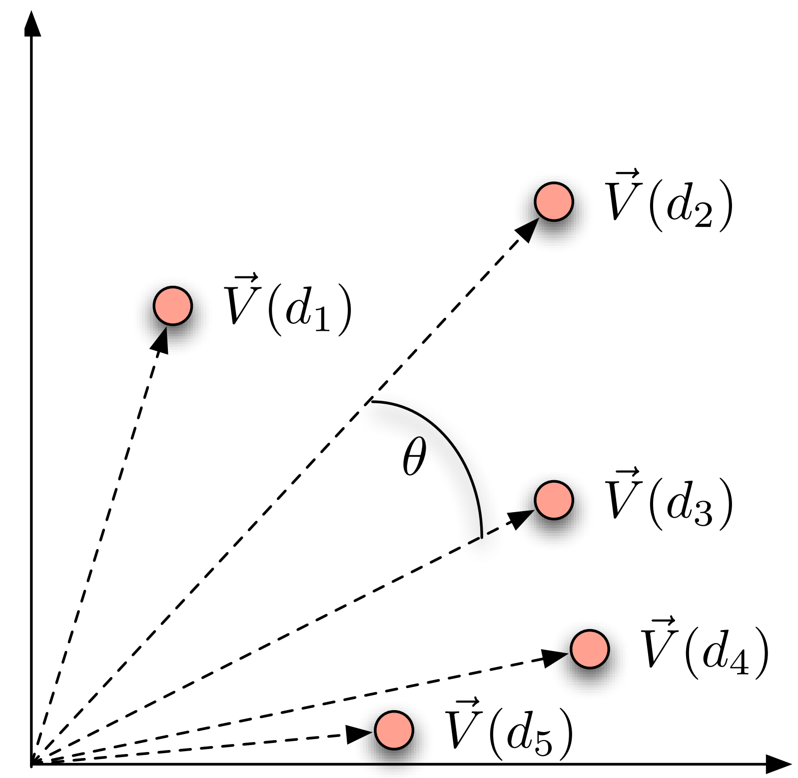
$$\begin{aligned} |\vec{V}(d_1)| &= \sqrt{(13.1 * 13.1) + (3.0 * 3.0) + (2.3 * 2.3) + (17.7 * 17.7) + (0.5 * 0.5) + (1.2 * 1.2)} \\ &= 22.38 \end{aligned}$$

Cosine Similarity Score

- Define: Euclidean Length

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

	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

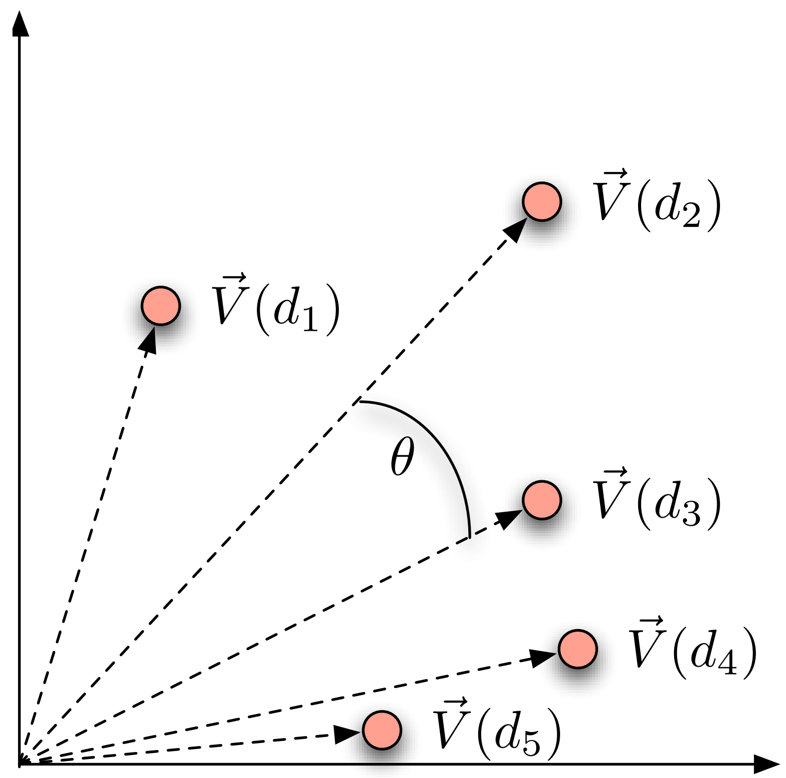


$$\begin{aligned} |\vec{V}(d_1)| &= \sqrt{(11.4 * 11.4) + (8.3 * 8.3) + (2.3 * 2.3) + (11.2 * 11.2)} \\ &= 18.15 \end{aligned}$$

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 frequent words in common
      - (the, a , an, of) (same number of each)
    - Two docs that have no words in common
    - Two docs that have many rare words in common
      - (mocha, volatile, organic, shade-grown)

Exercise

tf =

24	24	0	24	24	24	24	24	24	24	24	24	24	24	24
10	10	0	10	10	10	10	10	10	10	10	10	10	10	10
24	24	0	24	24	24	24	24	24	24	24	24	24	24	24
12	12	0	11	11	11	11	11	11	11	11	11	11	11	11
10	10	0	10	10	10	10	10	10	10	10	10	10	10	10
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0

df =

14
14
14
14
14
1
1
1
1
1
1
2
2
2

Exercise

```
c = 15;
tf =load('tf.txt','-ASCII');
df = load('df.txt','-ASCII');
tfidf = zeros(size(tf));

for i =1:size(tf,1)
for j = 1:size(tf,2)
if tf(i,j) == 0
    tfidf(i,j) = (0) * log2(c/df(i));
else
    tfidf(i,j) = (1+log2(tf(i,j))) * log2(c/df(i));
end
end
end
```

Exercise

tfidf =

0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4564	0.4564	0	0.4439	0.4439	0.4439	0.4439	0.4439
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0	0	0	0	3.9069	0	0	0
0	0	3.9069	0	0	0	0	0
0	0	3.9069	0	0	0	0	0
0	0	0	0	3.9069	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	3.9069	0	0	0
0	0	2.9069	2.9069	0	0	0	0
0	0	2.9069	2.9069	0	0	0	0
0	0	2.9069	2.9069	0	0	0	0

More of the same →

Exercise

tfidf =

0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4564	0.4564	0				0.4439	0.4439
0.4302	0.4302	0				0.4302	0.4302
0	0	0				0	0
0	0	3.9069				0	0
0	0	3.9069				0	0
0	0	0				0	0
0	0	0	0	0	0	0	0
0	0	0	0	3.9069	0	0	0
0							
0							
0							

More of the same

```
>> num = tfidf(:,1)'*tfidf(:,2)

num =

    1.1964
```

```
>> denom = sqrt(tfidf(:,1)'*tfidf(:,1)).*sqrt(tfidf(:,2)'*tfidf(:,2))

denom =

    1.1964
```

```
>> score = num/denom

score =

    1.0000
```

Exercise

tfidf =

0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4564	0.4564	0	0.4439	0.4439	0.4439	0.4439	0.4439
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0	0	0	0	3.9069	0	0	0
0	0	3.9069	0	0	0	0	0
0	0	3.9069	[illegible]	[illegible]	[illegible]	0	0
0	0	0	[illegible]	[illegible]	[illegible]	0	0
0	0	0	[illegible]	[illegible]	[illegible]	0	0
0	0	0	[illegible]	[illegible]	[illegible]	0	0
0	0	2.9069	2.9[illegible]	[illegible]	[illegible]	0	0
0	0	2.9069	2.9[illegible]	[illegible]	[illegible]	0	0
0	0	2.9069	2.9069	0	0	0	0

More of the same →

```
>> num = tfidf(:,2)'*tfidf(:,3)

num =

     0
```

```
>> denom = sqrt(tfidf(:,2)'*tfidf(:,2)).*sqrt(tfidf(:,3)'*tfidf(:,3))

denom =

    8.1765
```

```
>> score = num/denom

score =

     0
```

Exercise

tfidf =

0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0.5559	0.5559	0	0.5559	0.5559	0.5559	0.5559	0.5559
0.4564	0.4564	0	0.4439	0.4439	0.4439	0.4439	0.4439
0.4302	0.4302	0	0.4302	0.4302	0.4302	0.4302	0.4302
0	0	0	0	3.9069	0	0	0
0	0	3.9069	0	0	0	0	0
0	0	3.9069	0				0
0	0	0	0				0
0	0	0	0				0
0	0	0	0				0
0	0	2.9069	2.9069				0
0	0	2.9069	2.9069				0
0	0	2.9069	2.9069	0	0	0	0

More of the same →

```
>> num = tfidf(:,3)'*tfidf(:,4)

num =

   25.3500
```

```
>> denom = sqrt(tfidf(:,3)'*tfidf(:,3)).*sqrt(tfidf(:,4)'*tfidf(:,4))

denom =

   38.5062
```

```
>> score = num/denom

score =

    0.6583
```

Exercise

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

```
>> score = num/denom

score =

    1.0000
```

```
>> score = num/denom

score =

     0
```

```
>> score = num/denom

score =

    0.6583
```

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’t effectively discriminate results.

LUCI