

# Corpus-wide statistics



## Corpus-wide statistics

- **Collection Frequency**, cf
- Define: The total number of occurrences of the term in the entire corpus



## Corpus-wide statistics

- **Collection Frequency**, cf
  - Define: The total number of occurrences of the term in the entire corpus
- **Document Frequency**, df
  - Define: The total number of documents which contain the term in the corpus



## Corpus-wide statistics

<i>Word</i>	<i>Collection Frequency</i>	<i>Document Frequency</i>
-------------	-----------------------------	---------------------------

<i>insurance</i>	10440	3997
------------------	-------	------

<i>try</i>	10422	8760
------------	-------	------



## Corpus-wide statistics

<i>Word</i>	<i>Collection Frequency</i>	<i>Document Frequency</i>
-------------	-----------------------------	---------------------------

<i>insurance</i>	10440	3997
------------------	-------	------

<i>try</i>	10422	8760
------------	-------	------

- This suggests that df is better at discriminating between documents



## Corpus-wide statistics

<i>Word</i>	<i>Collection Frequency</i>	<i>Document Frequency</i>
-------------	-----------------------------	---------------------------

<i>insurance</i>	10440	3997
------------------	-------	------

<i>try</i>	10422	8760
------------	-------	------

- This suggests that df is better at discriminating between documents
- How do we use df?



Querying

# Corpus-wide statistics



# Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights





# Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
- “tf-idf”



# Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - $tf$  = term frequency



## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document



## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document
  - idf = inverse document frequency



## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document
  - idf = inverse document frequency
    - a measure of the informativeness of a term



## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document
  - idf = inverse document frequency
    - a measure of the informativeness of a term
    - it's rarity across the corpus



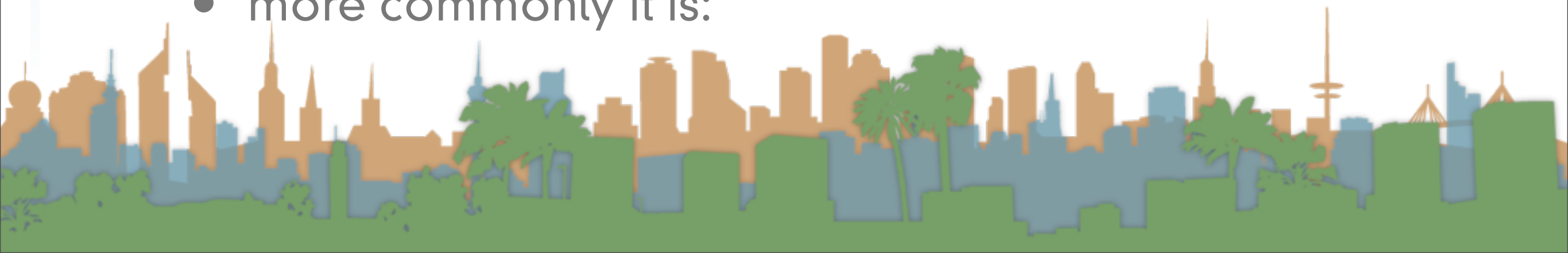
## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document
  - idf = inverse document frequency
    - a measure of the informativeness of a term
    - it's rarity across the corpus
    - could be just a count of documents with the term



## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document
  - idf = inverse document frequency
    - a measure of the informativeness of a term
    - it's rarity across the corpus
    - could be just a count of documents with the term
    - more commonly it is:





## Corpus-wide statistics

- Term-Frequency, Inverse Document Frequency Weights
  - “tf-idf”
  - tf = term frequency
    - some measure of term density in a document
  - idf = inverse document frequency
    - a measure of the informativeness of a term
    - it's rarity across the corpus
    - could be just a count of documents with the term
    - more commonly it is:

$$idf_t = \log \left( \frac{|corpus|}{df_t} \right)$$

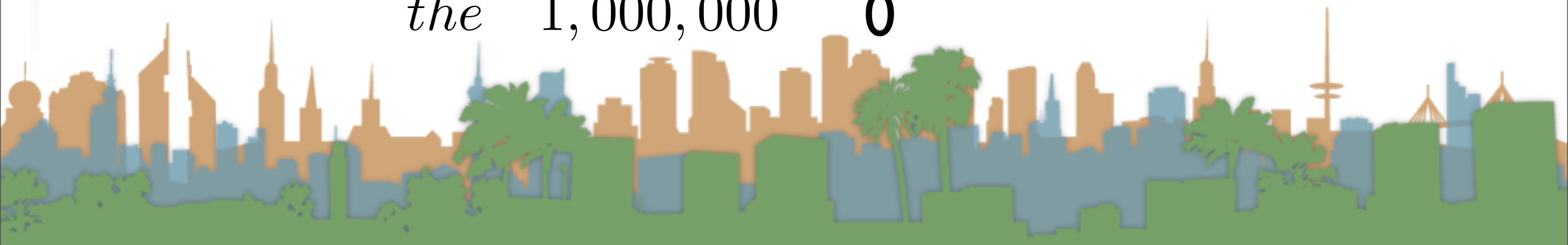


## TF-IDF Examples

$$idf_t = \log \left( \frac{|corpus|}{df_t} \right)$$

$$idf_t = \log_{10} \left( \frac{1,000,000}{df_t} \right)$$

<i>term</i>	<i>df<sub>t</sub></i>	<i>idf<sub>t</sub></i>
<i>calpurnia</i>	1	<b>6</b>
<i>animal</i>	10	<b>4</b>
<i>sunday</i>	1000	<b>3</b>
<i>fly</i>	10,000	<b>2</b>
<i>under</i>	100,000	<b>1</b>
<i>the</i>	1,000,000	<b>0</b>



## TF-IDF Summary

- Assign tf-idf weight for each term  $t$  in a document  $d$ :

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

- Increases with number of occurrences of term in a doc.
- Increases with rarity of term across entire corpus
- Three different metrics
  - term frequency
  - document frequency
  - collection/corpus frequency



## Now, real-valued term-document matrices

- Bag of words model
- Each element of matrix is tf-idf value

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



# Vector Space Scoring

- That is a nice matrix, but
  - How does it relate to scoring?
  - Next, vector space scoring



# 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

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

- 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:

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





## Vector Space Model

- Recall our Shakespeare Example:

	$\vec{V}(d_1)$					
	<i>Antony and Cleopatra</i>	<i>Julius Caesar</i>	<i>The Tempest</i>	<i>Hamlet</i>	<i>Othello</i>	<i>Macbeth</i>
<i>Antony</i>	13.1	11.4	0.0	0.0	0.0	0.0
<i>Brutus</i>	3.0	8.3	0.0	1.0	0.0	0.0
<i>Caesar</i>	2.3	2.3	0.0	0.5	0.3	0.3
<i>Calpurnia</i>	0.0	11.2	0.0	0.0	0.0	0.0
<i>Cleopatra</i>	17.7	0.0	0.0	0.0	0.0	0.0
<i>mercy</i>	0.5	0.0	0.7	0.9	0.9	0.3
<i>worser</i>	1.2	0.0	0.6	0.6	0.6	0.0



## Vector Space Model

- Recall our Shakespeare Example:

	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	<i>Antony and Cleopatra</i>	<i>Julius Caesar</i>	<i>The Tempest</i>	<i>Hamlet</i>	<i>Othello</i>	<i>Macbeth</i>
<i>Antony</i>	13.1	11.4	0.0	0.0	0.0	0.0
<i>Brutus</i>	3.0	8.3	0.0	1.0	0.0	0.0
<i>Caesar</i>	2.3	2.3	0.0	0.5	0.3	0.3
<i>Calpurnia</i>	0.0	11.2	0.0	0.0	0.0	0.0
<i>Cleopatra</i>	17.7	0.0	0.0	0.0	0.0	0.0
<i>mercy</i>	0.5	0.0	0.7	0.9	0.9	0.3
<i>worser</i>	1.2	0.0	0.6	0.6	0.6	0.0



## Vector Space Model

- Recall our Shakespeare Example:

	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	<i>Antony and Cleopatra</i>	<i>Julius Caesar</i>	<i>The Tempest</i>	<i>Hamlet</i>	<i>Othello</i>	<i>Macbeth</i>
<i>Antony</i>	13.1	11.4	0.0	0.0	0.0	0.0
<i>Brutus</i>	3.0	8.3	0.0	1.0	0.0	0.0
<i>Caesar</i>	2.3	2.3	0.0	0.5	0.3	0.3
<i>Calpurnia</i>	0.0	11.2	0.0	0.0	0.0	0.0
<i>Cleopatra</i>	17.7	0.0	0.0	0.0	0.0	0.0
<i>mercy</i>	0.5	0.0	0.7	0.9	0.9	0.3
<i>worser</i>	1.2	0.0	0.6	0.6	0.6	0.0

$\vec{V}(d_6)_7$



## Vector Space Model

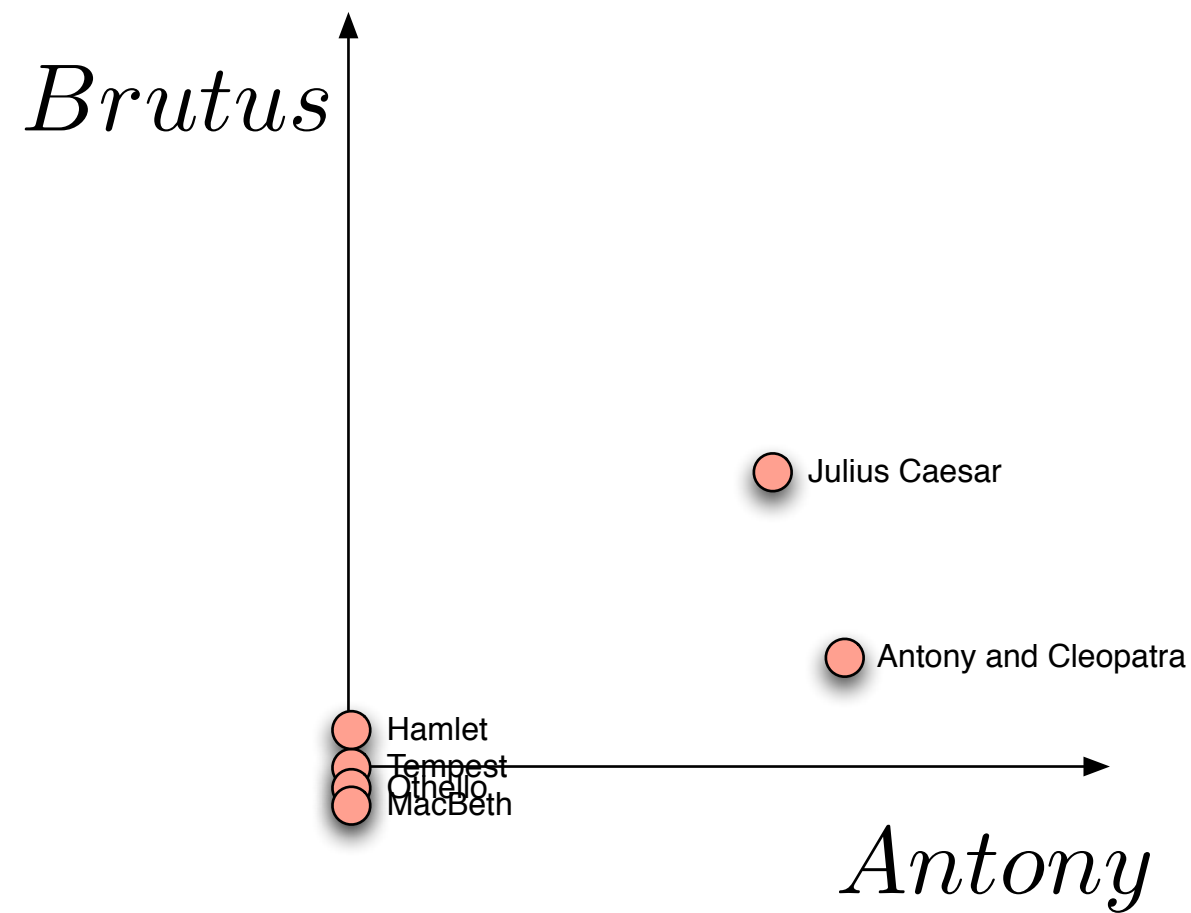
- Recall our Shakespeare Example:

	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	<i>Antony and Cleopatra</i>	<i>Julius Caesar</i>	<i>The Tempest</i>	<i>Hamlet</i>	<i>Othello</i>	<i>Macbeth</i>
<i>Antony</i>	13.1	11.4	0.0	0.0	0.0	0.0
<i>Brutus</i>	3.0	8.3	0.0	1.0	0.0	0.0
<i>Caesar</i>	2.3	2.3	0.0	0.5	0.3	0.3
<i>Calpurnia</i>	0.0	11.2	0.0	0.0	0.0	0.0
<i>Cleopatra</i>	17.7	0.0	0.0	0.0	0.0	0.0
<i>mercy</i>	0.5	0.0	0.7	0.9	0.9	0.3
<i>worser</i>	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:

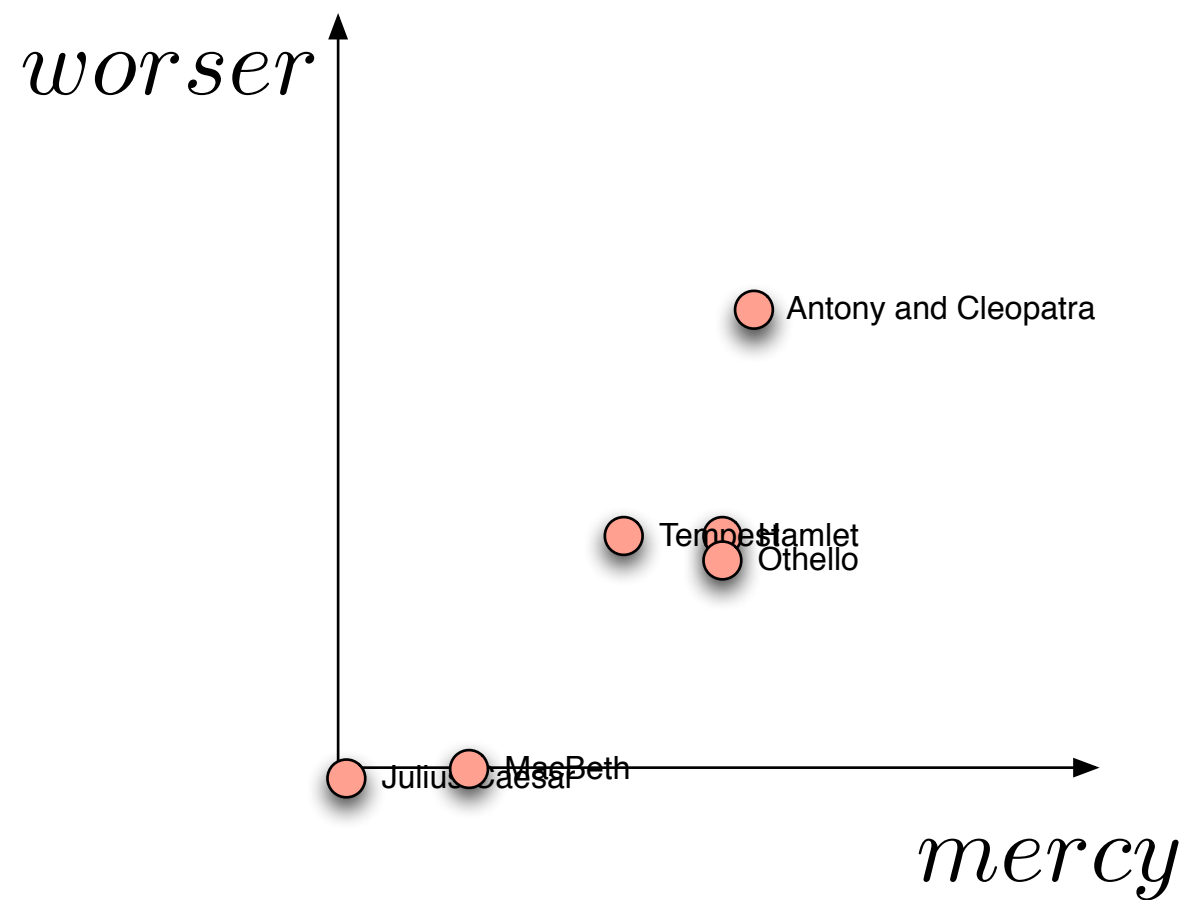
	$\vec{V}(d_1)$	$\vec{V}(d_2)$				$\vec{V}(d_6)$
	<i>Antony and Cleopatra</i>	<i>Julius Caesar</i>	<i>The Tempest</i>	<i>Hamlet</i>	<i>Othello</i>	<i>Macbeth</i>
<i>Antony</i>	13.1	11.4	0.0	0.0	0.0	0.0
<i>Brutus</i>	3.0	8.3	0.0	1.0	0.0	0.0
<i>Caesar</i>	2.3	2.3	0.0	0.5	0.3	0.3
<i>Calpurnia</i>	0.0	11.2	0.0	0.0	0.0	0.0
<i>Cleopatra</i>	17.7	0.0	0.0	0.0	0.0	0.0
<i>mercy</i>	0.5	0.0	0.7	0.9	0.9	0.3
<i>worser</i>	1.2	0.0	0.6	0.6	0.6	0.0





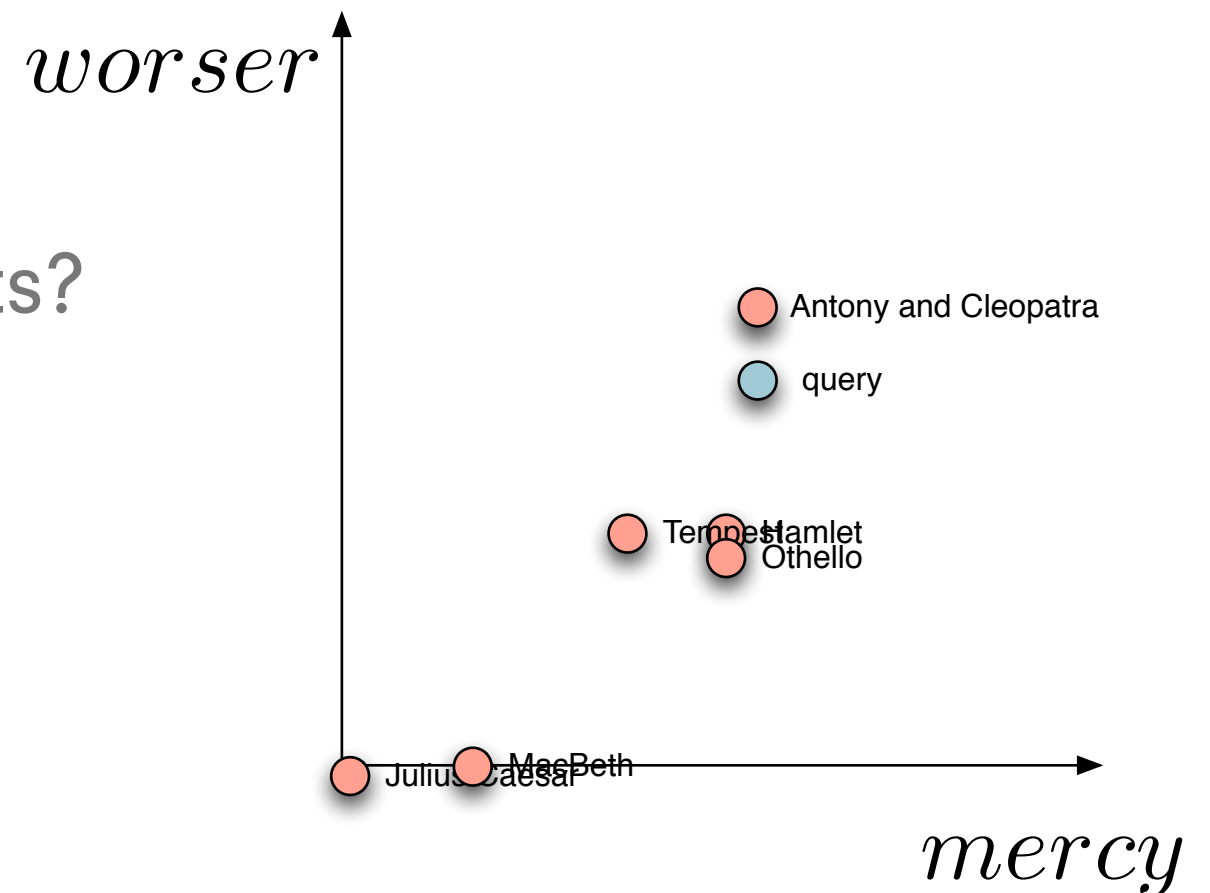
## Vector Space Model

- Recall our Shakespeare Example:



## Query as a vector

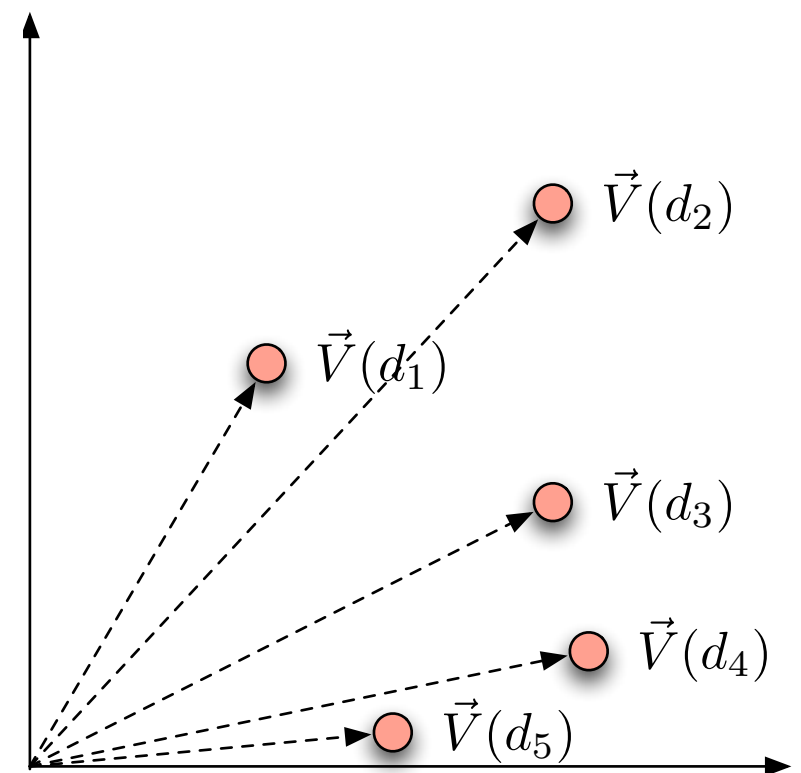
- So a query can also be plotted in the same space
- “worser mercy”
- To score, we ask:
  - How similar are two points?
- How to answer?





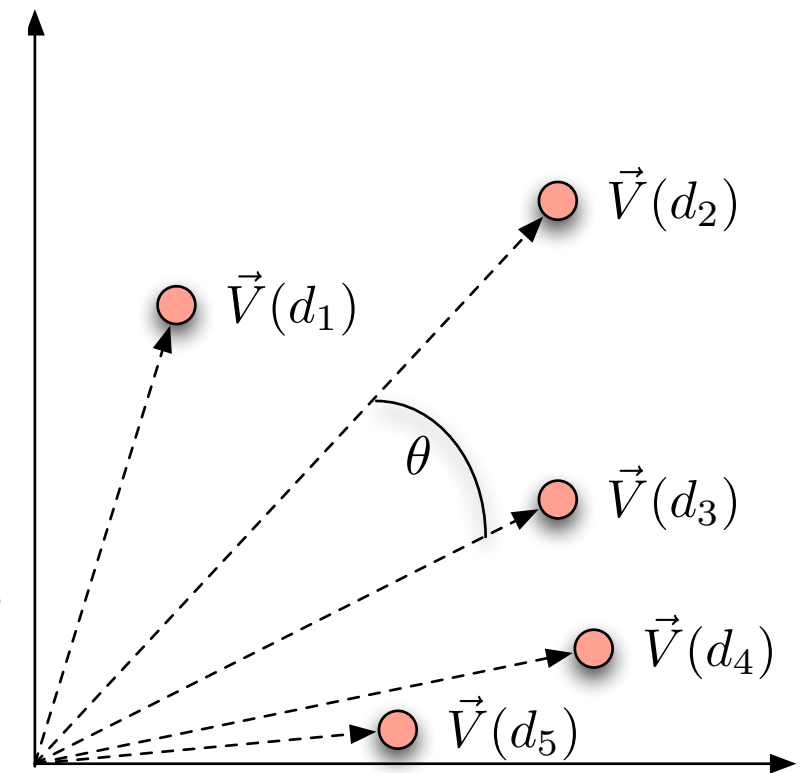
## Score by magnitude

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



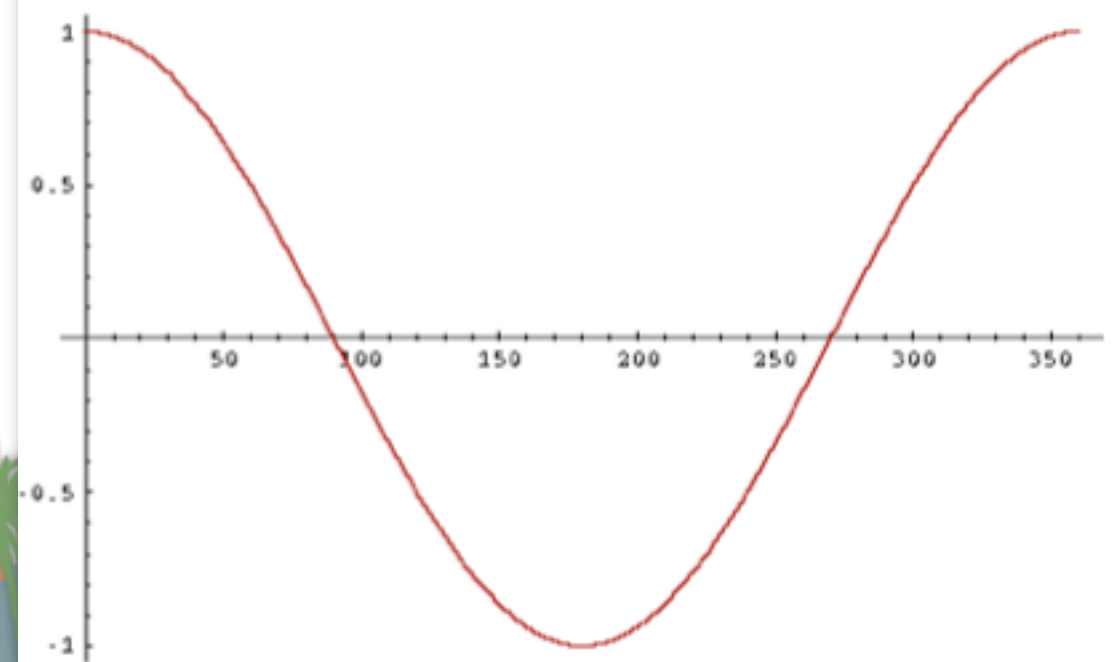
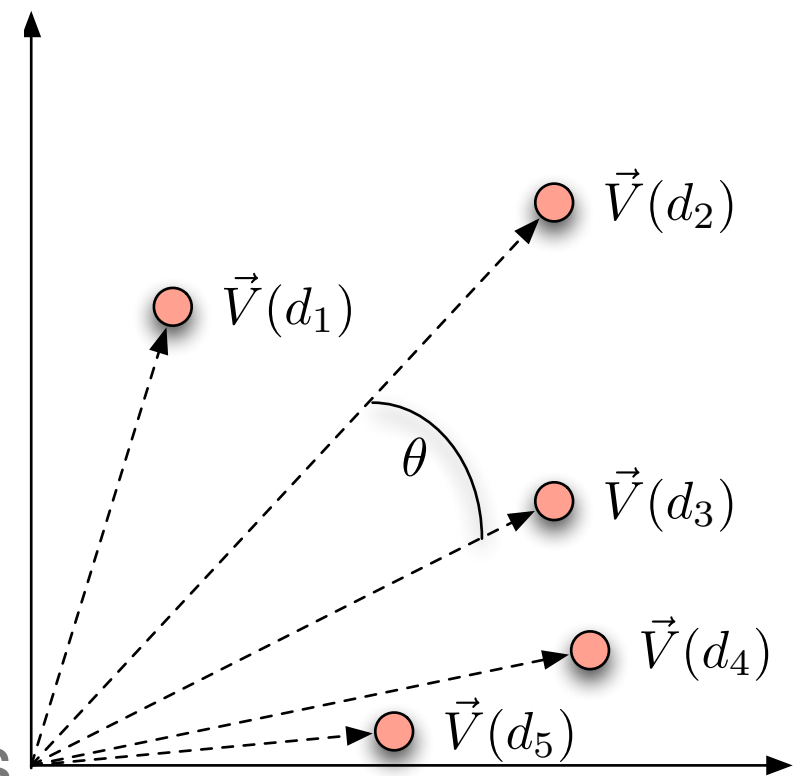
## Score by angle

- 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  $(0 \dots 180)$



## 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)

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

- Note that  $q$  is very sparse!

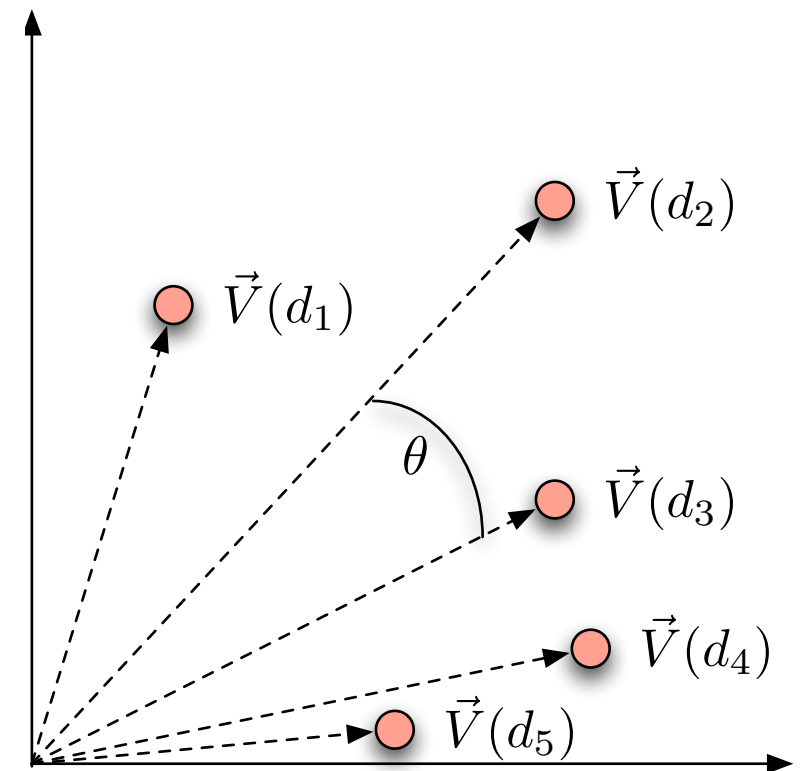


## Cosine Similarity Score

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

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

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





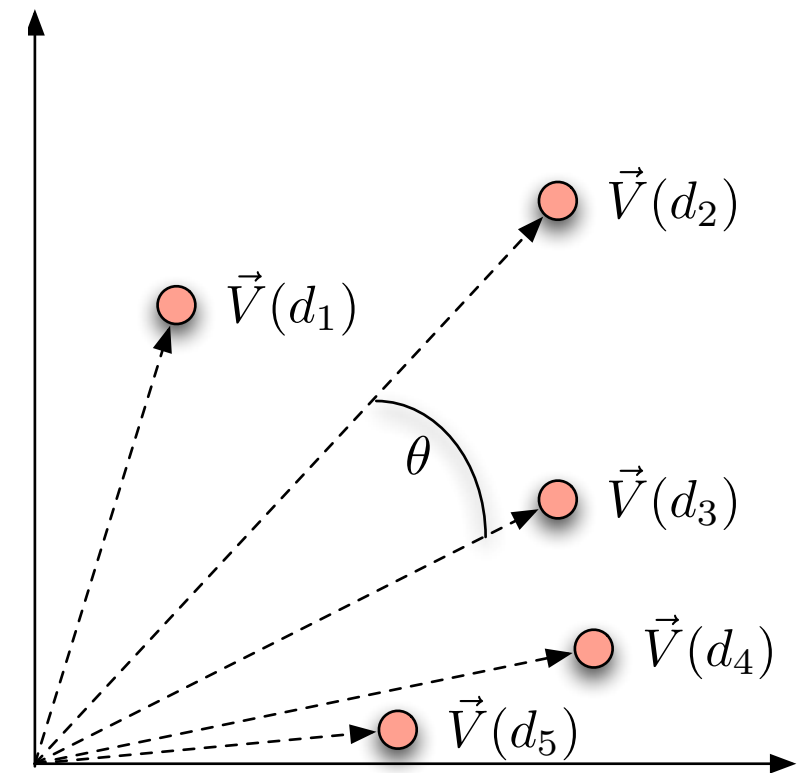
# Vector Space Scoring

## Cosine Similarity Score

- Define: dot product

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

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



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



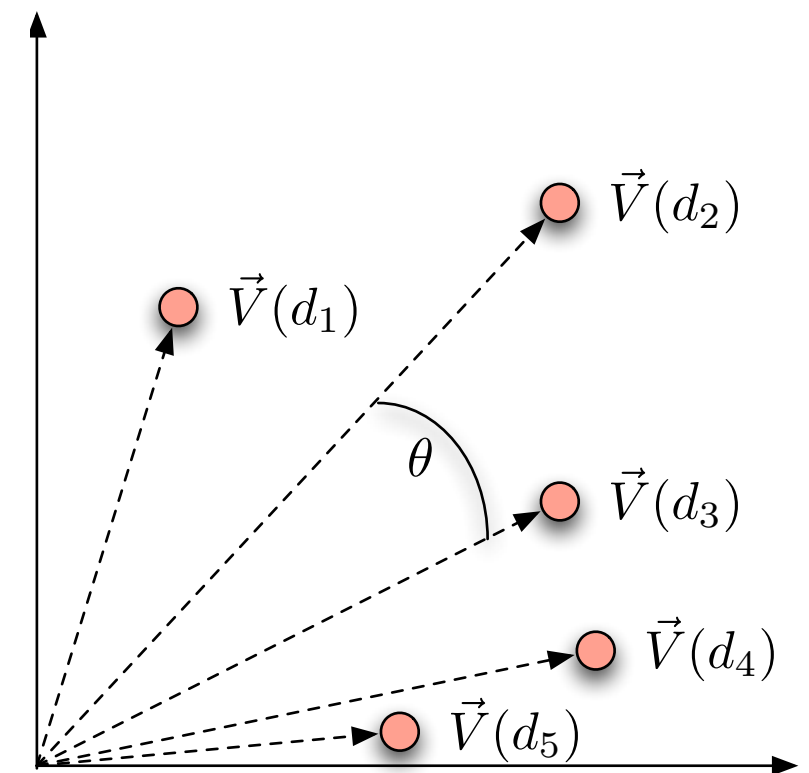


## Cosine Similarity Score

- Define: Euclidean Length

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

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



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

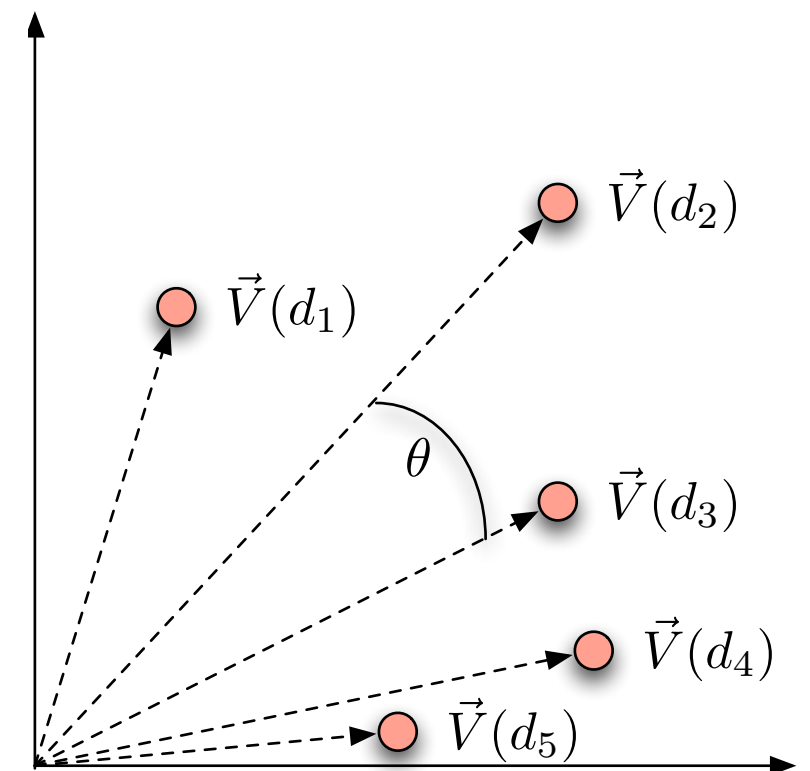


## Cosine Similarity Score

- Define: Euclidean Length

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

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

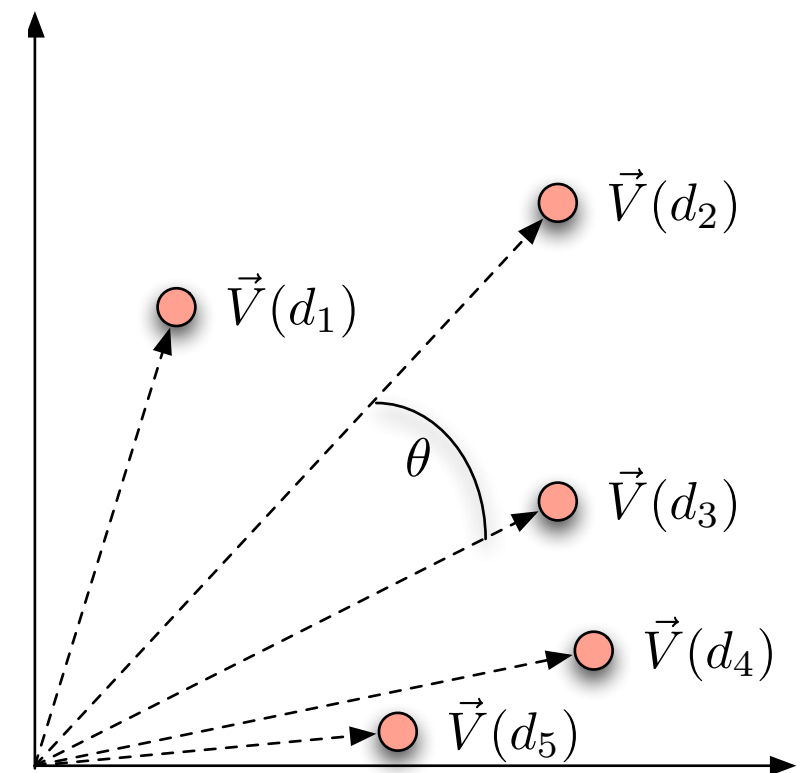


$$\begin{aligned} |\vec{V}(d_1)| &= \sqrt{(11.4 * 11.4) + (8.3 * 8.3) + (2.3 * 2.3) + (11.2 * 11.2)} \\ &= 18.15 \end{aligned}$$

## Cosine Similarity Score

- Example

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



## 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’t effectively discriminate results.





# Multiple queries for phrases and vectors

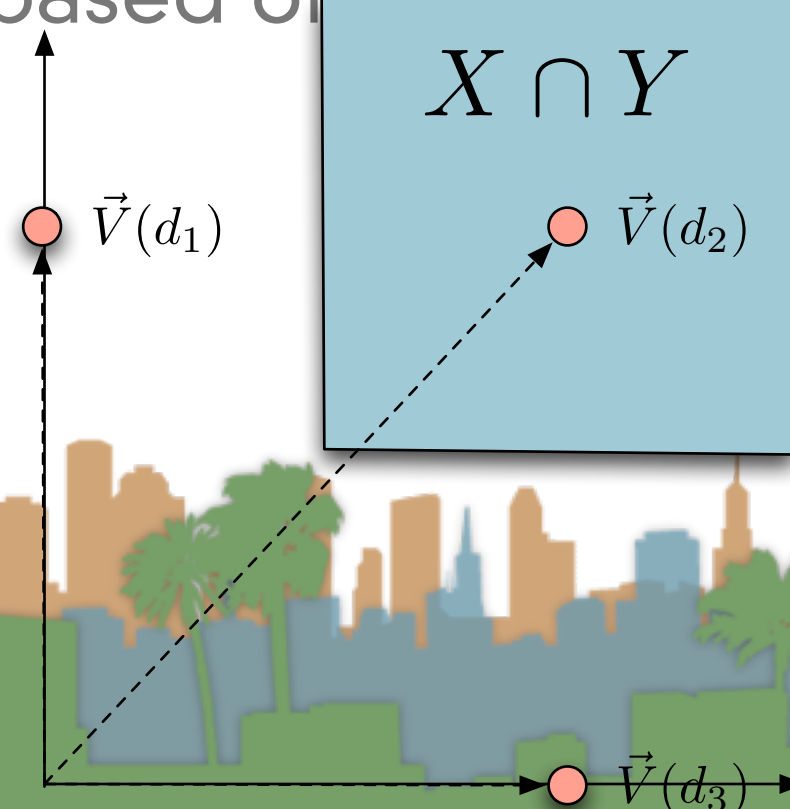
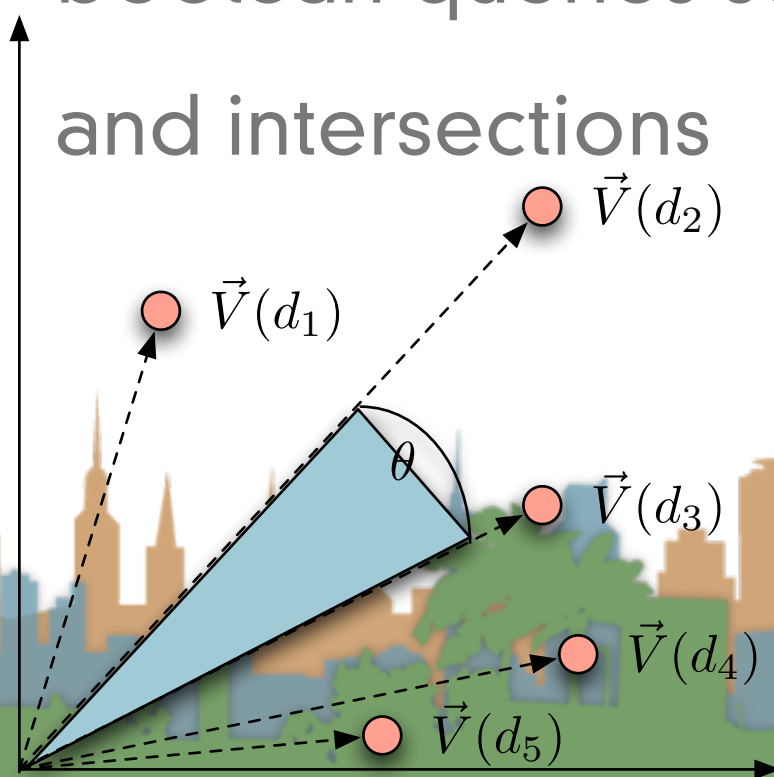
- Query: “rising interest rates”
- Iterative refinement:
  - Run the phrase query vector with 3 words as a term.
  - If not enough results, run 2-phrase queries and fold into results: “rising interest” “interest rates”
  - If still not enough results run query with three words as separate terms.





## 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



# Vectors and wild cards



# Vectors and wild cards

- How could we work with the query, “quick\* print\*” ?



# Vectors and wild cards

- How could we work with the query, “quick\* print\*” ?
- Can we view this as a bag of words?



### 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?



### 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 tf's and idf's to deal with



### 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 tf's and idf's 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.

$WTF(t, d)$

1    **if**  $tf_{t,d} = 0$

2        **then**  $return(0)$

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

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

