MATRIX DECOMPOSITION AND LATENT SEMANTIC INDEXING (LSI)

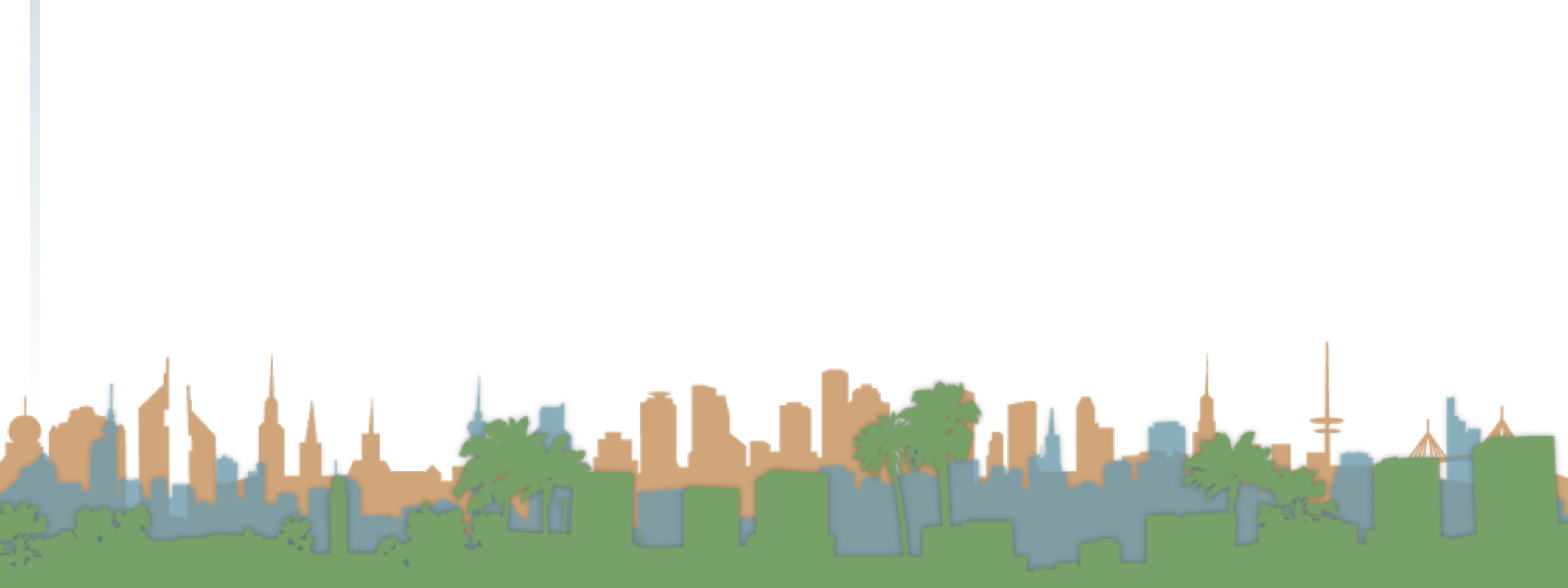
Introduction to Information Retrieval

CS 150
Donald J. Patterson

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

Outline

- Introduction
- Linear Algebra Refresher



Star Cluster NGC 290 - ESA & NASA

Star Cluster NGC 290 - ESA & NASA

- A picture of the sky is two dimensional
- The stars are not in two dimensions
- When we take a photo of stars we are projecting them into 2-D
  - projecting can be defined mathematically
- When we see two stars that are close..
  - They may not be close in space
- When we see two stars that appear far...
  - They may not be far in 3-D space

Star Cluster NGC 290 - ESA & NASA

- When we see two stars that are close in a photo
  - They really are close for some applications
  - For example pointing a big telescope at them
  - Large shared telescopes order their views according to how “close” they are.

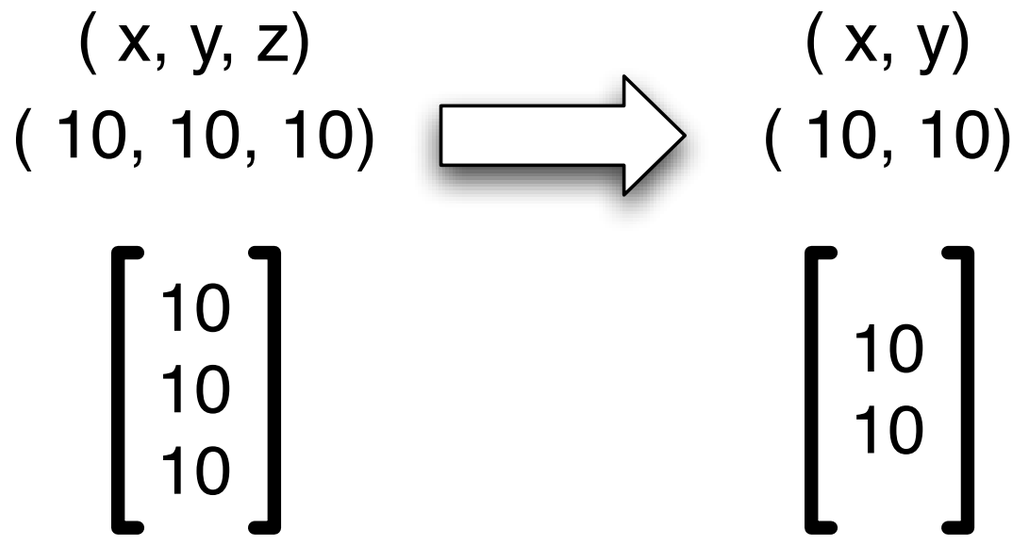
Overhead projector example

Overhead projector example

- Depending on where we put the light (and the wall) we can make things in three dimensions appear close or far away in two dimensions.
- Even though the "real" position of the 3-d objects never moved.

Mathematically speaking

- This is taking a 3-D point and projecting it into 2-D



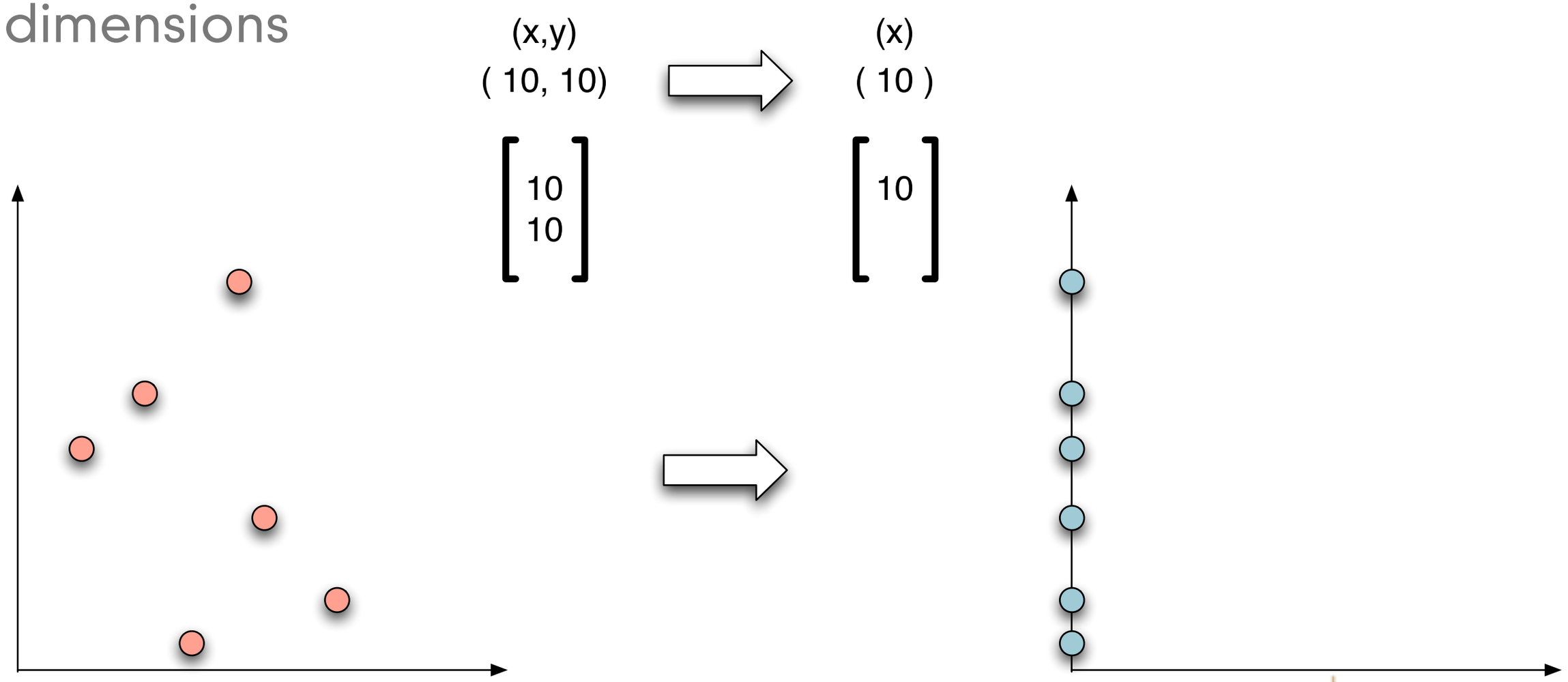
- The arrow in this picture acts like the overhead projector

Mathematically speaking

- We can project from any number of dimensions into any other number of dimensions.
- Increasing dimensions adds redundant information
  - But sometimes useful
    - Support Vector Machines (kernel methods) do this effectively
- Latent Semantic Indexing always reduces the number of dimensions

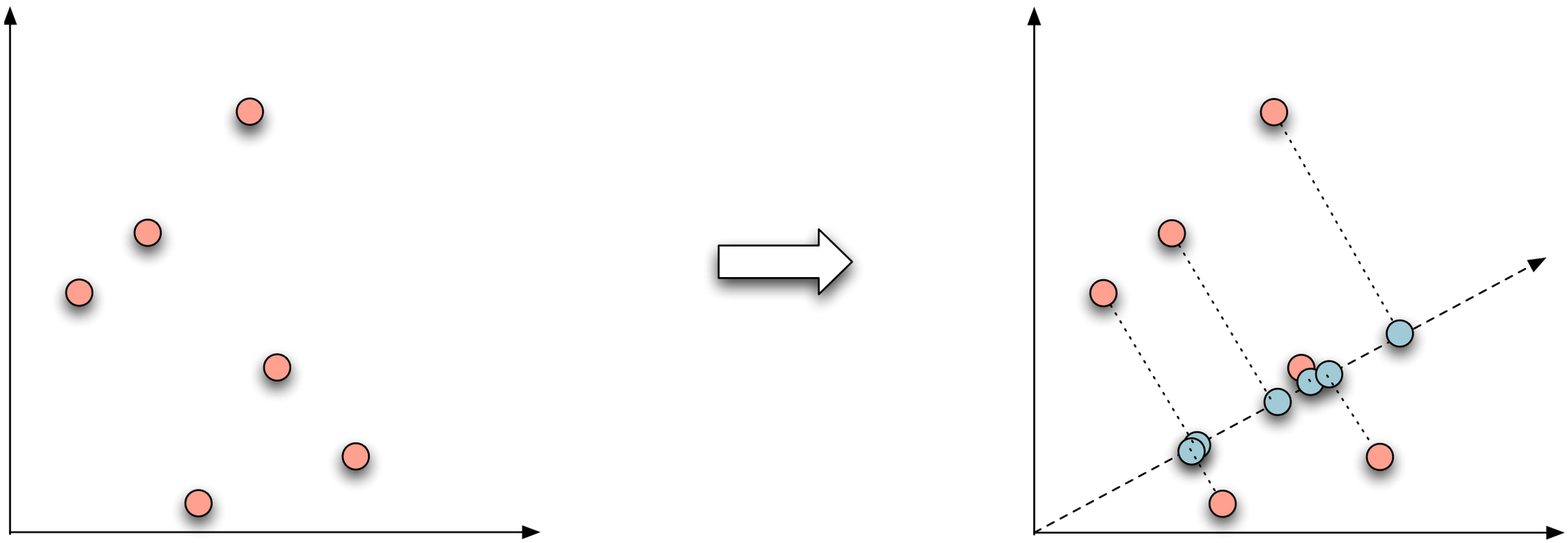
Mathematically speaking

- Latent Semantic Indexing always reduces the number of dimensions



Mathematically speaking

- Latent Semantic Indexing can project on an arbitrary axis, not just a principal axis



Mathematically speaking

- Our documents were just points in an N-dimensional term space
- We can project them also

Mathematically speaking

- Latent Semantic Indexing makes the claim that these new axes represent semantics - deeper meaning than just a term

Mathematically speaking

- A term vector that is projected on new vectors may uncover deeper meanings
  - For example
    - Transforming the 3 axes of a term matrix from “ball” “bat” and “cave” to
      - An axis that merges “ball” and “bat”
      - An axis that merges “bat” and “cave”
- Should be able to separate differences in meaning of the term “bat”
- Bonus: less dimensions is faster

Linear Algebra Refresher

- Let C be an M by N matrix with real-valued entries
  - for example our term document matrix

M=3, N=5

$$C = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 2 & 3 & 2 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

- A matrix with the same number of rows and columns is called a square matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

- An M by M matrix with elements only on the diagonal is called a diagonal matrix

$$\begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

- The identity matrix is a diagonal matrix with ones on the main diagonal

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix Decomposition

- Singular Value Decomposition
  - Splits a matrix into three matrices
  - Such that
    - If
    - then
    - and
    - and
  - also Sigma is almost a diagonal matrix

$$\begin{matrix} & U & \Sigma & V^T \end{matrix}$$

$$C = U\Sigma V^T$$

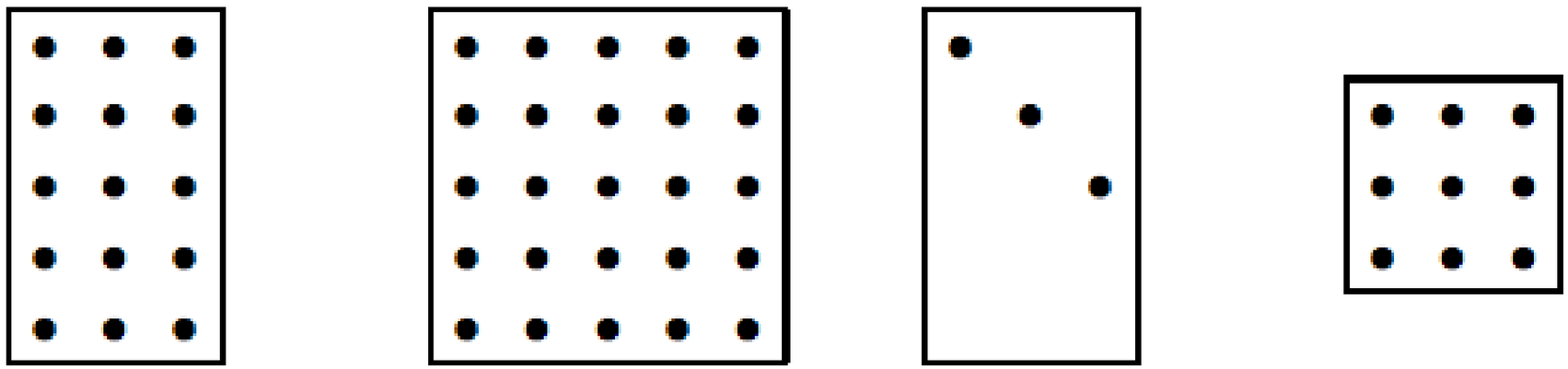
$$C \ is \ (M \ by \ N)$$

$$U \ is \ (M \ by \ M)$$

$$\Sigma \ is \ (M \ by \ N)$$

$$V^T \ is \ (N \ by \ N)$$

Matrix Decomposition



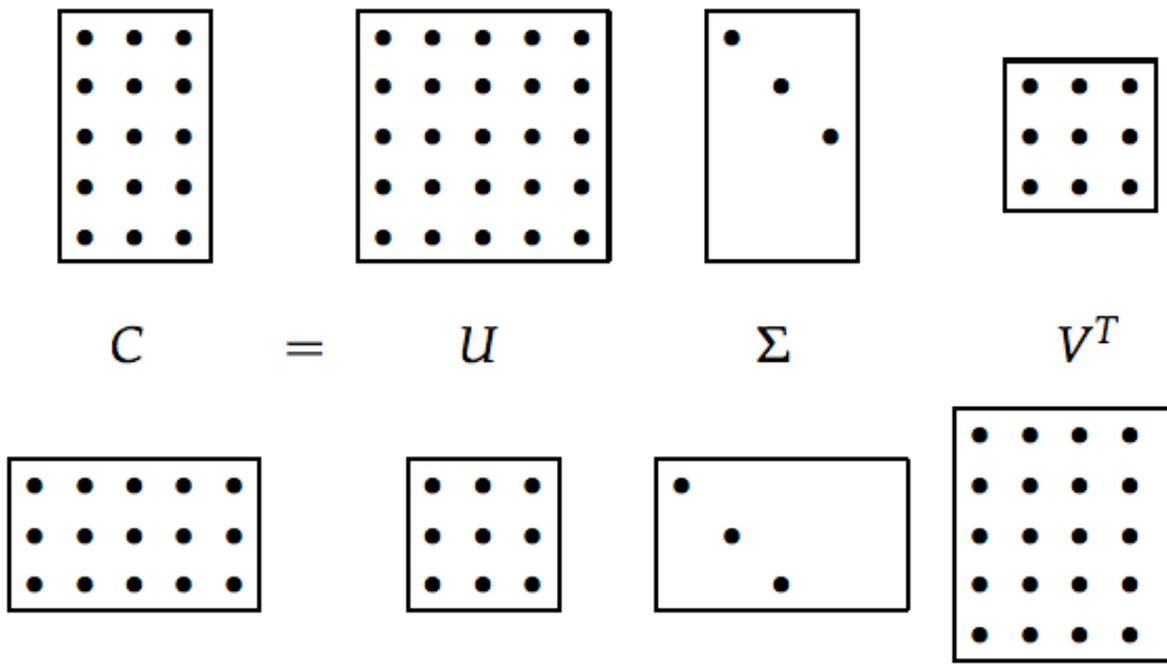
$$C = U \Sigma V^T$$

Matrix Decomposition

- Singular Value Decomposition
  - Is a technique that splits a matrix into three components with these properties.
  - They also have some other properties which are relevant to latent semantic indexing

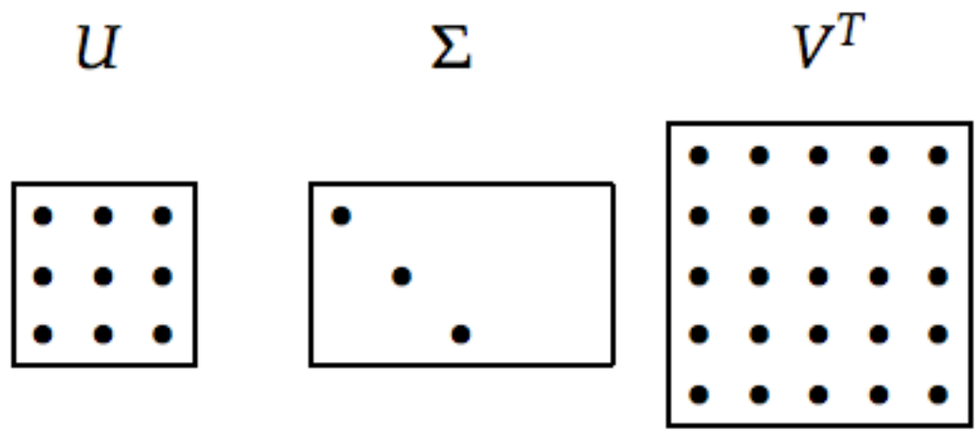
Matrix Decomposition

- Singular Value Decomposition
  - Is a technique that splits a matrix into three components with these properties.



Matrix Decomposition

- Singular Value Decomposition
  - SVD enables lossy compression of your term-document matrix
    - reduces the dimensionality or the rank
    - you can arbitrarily reduce the dimensionality by putting zeros in the bottom right of sigma
  - this is a mathematically optimal way of reducing dimensions



Matrix Decomposition

- Singular Value Decomposition
  - If the old dimensions were based on terms
    - after reducing the rank of the matrix the dimensionality is based on concepts or semantics
      - a concept is a linear combination of terms

Matrix Decomposition

- Singular Value Decomposition
- 4 dimensions to 3 dimensions

$$\begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix}$$

Matrix Decomposition

- Singular Value Decomposition
- 4 dimensions to 3 dimensions

$$\begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix} * \begin{vmatrix} td_{dim_1} \\ td_{dim_2} \\ td_{dim_3} \\ td_{dim_4} \end{vmatrix}$$

Matrix Decomposition

- Singular Value Decomposition

$$\begin{vmatrix} SVD_{dim_1} \\ SVD_{dim_2} \\ SVD_{dim_3} \end{vmatrix} = \begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix} * \begin{vmatrix} td_{dim_1} \\ td_{dim_2} \\ td_{dim_3} \\ td_{dim_4} \end{vmatrix}$$

Matrix Decomposition

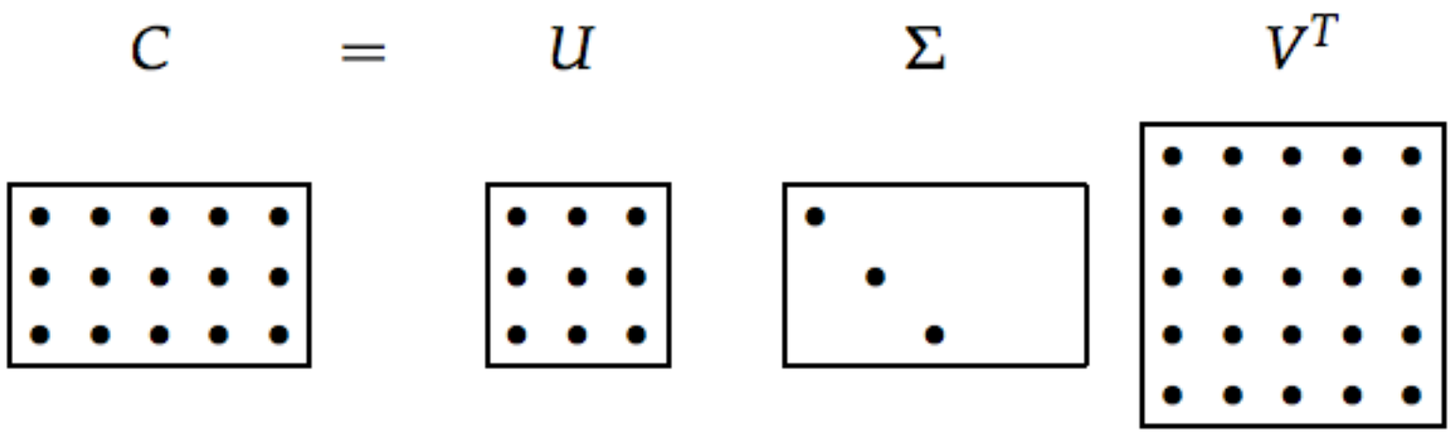
- Singular Value Decomposition

$$\begin{vmatrix} SVD_{dim_1} \\ SVD_{dim_2} \\ SVD_{dim_3} \end{vmatrix} = \begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix} * \begin{vmatrix} td_{dim_1} \\ td_{dim_2} \\ td_{dim_3} \\ td_{dim_4} \end{vmatrix}$$

Matrix Decomposition

- Singular Value Decomposition

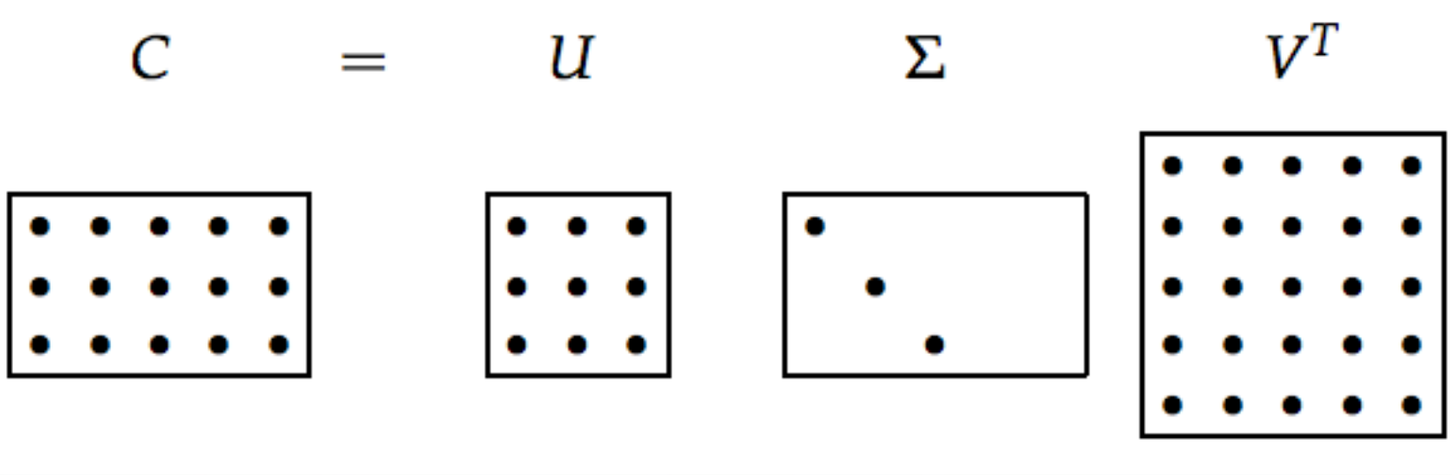
$$\begin{vmatrix} SVD_{dim_1} \\ SVD_{dim_2} \\ SVD_{dim_3} \end{vmatrix} = \begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix} * \begin{vmatrix} td_{dim_1} \\ td_{dim_2} \\ td_{dim_3} \\ td_{dim_4} \end{vmatrix}$$



Matrix Decomposition

- Singular Value Decomposition

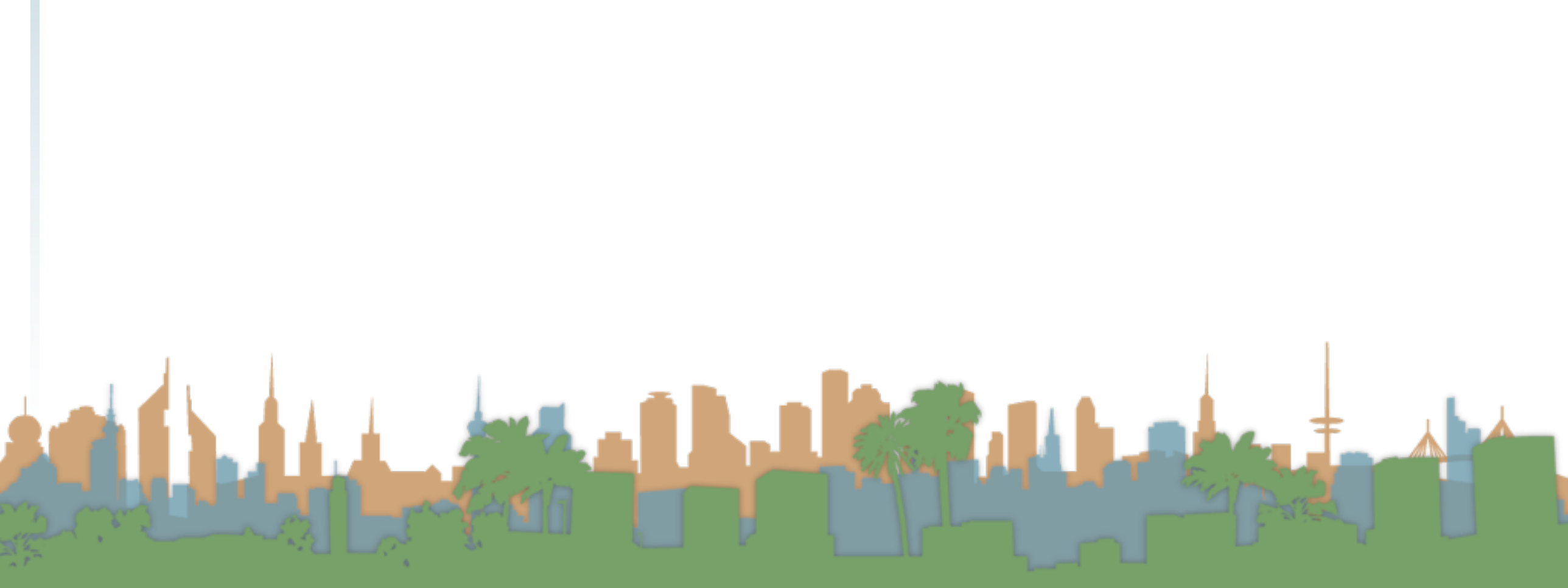
$$\begin{vmatrix} SVD_{dim_1} \\ SVD_{dim_2} \\ SVD_{dim_3} \end{vmatrix} = \begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix} * \begin{vmatrix} td_{dim_1} \\ td_{dim_2} \\ td_{dim_3} \\ td_{dim_4} \end{vmatrix}$$



Matrix Decomposition

- Singular Value Decomposition

$$\begin{vmatrix} SVD_{dim_1} \\ SVD_{dim_2} \\ SVD_{dim_3} \end{vmatrix} = \begin{vmatrix} a & b & c & d \\ a' & b' & c' & d' \\ a'' & b'' & c'' & d'' \end{vmatrix} * \begin{vmatrix} td_{dim_1} \\ td_{dim_2} \\ td_{dim_3} \\ td_{dim_4} \end{vmatrix}$$



Matrix Decomposition

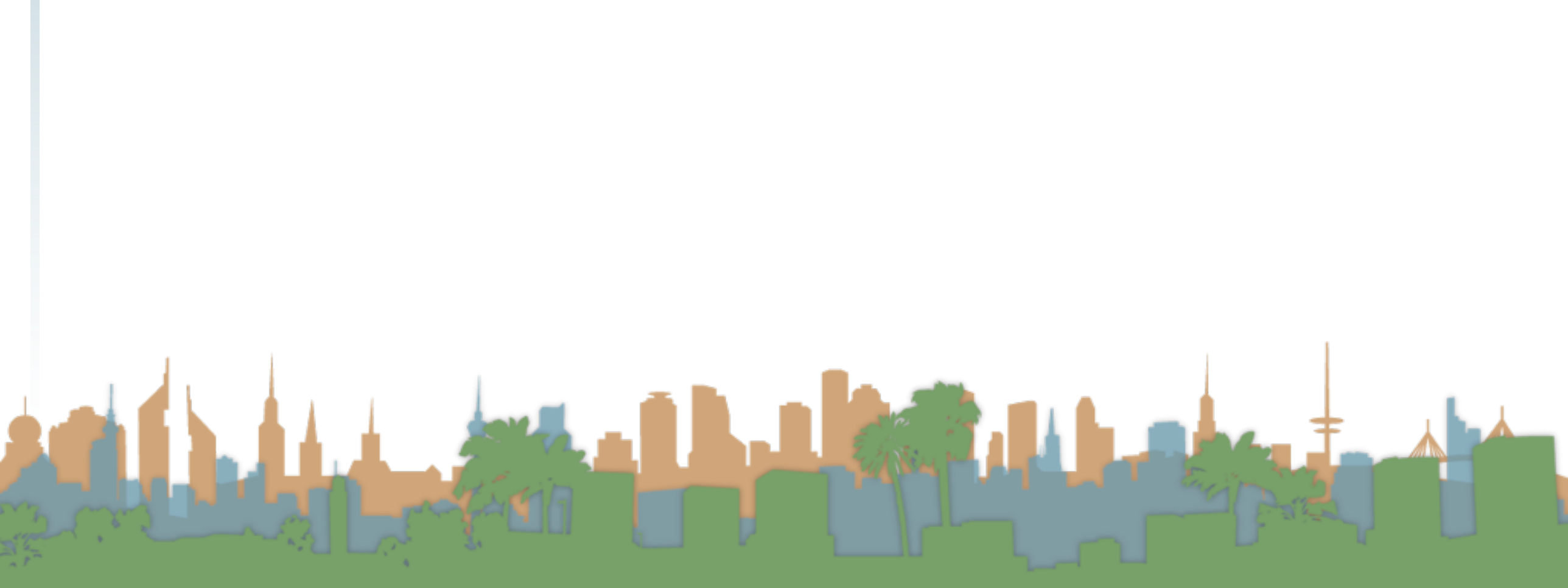
- Singular Value Decomposition
  - SVD is an algorithm that gives us $\Sigma\ U\ V^T$
  - With these quantities we can reduce dimensionality
  - With reduced dimensionality
    - synonyms are mapped onto the same location
      - “bat” “chiroptera”
    - polysemies are mapped onto different locations
      - “bat” (baseball) vs. “bat” (small furry mammal)

Latent Semantic Indexing - Linear Algebra Refresher

- Computing SVD takes a significant amount of CPU
- It is possible to add documents to a corpus without recalculating SVD
  - The result becomes an approximation
  - To get mathematical guarantees the whole SVD needs to be computed from scratch
- LSI doesn’t support negation queries
- LSI doesn’t support boolean queries

Matrix Decomposition

- “I am not crazy”
  - Netflix



Matrix Decomposition

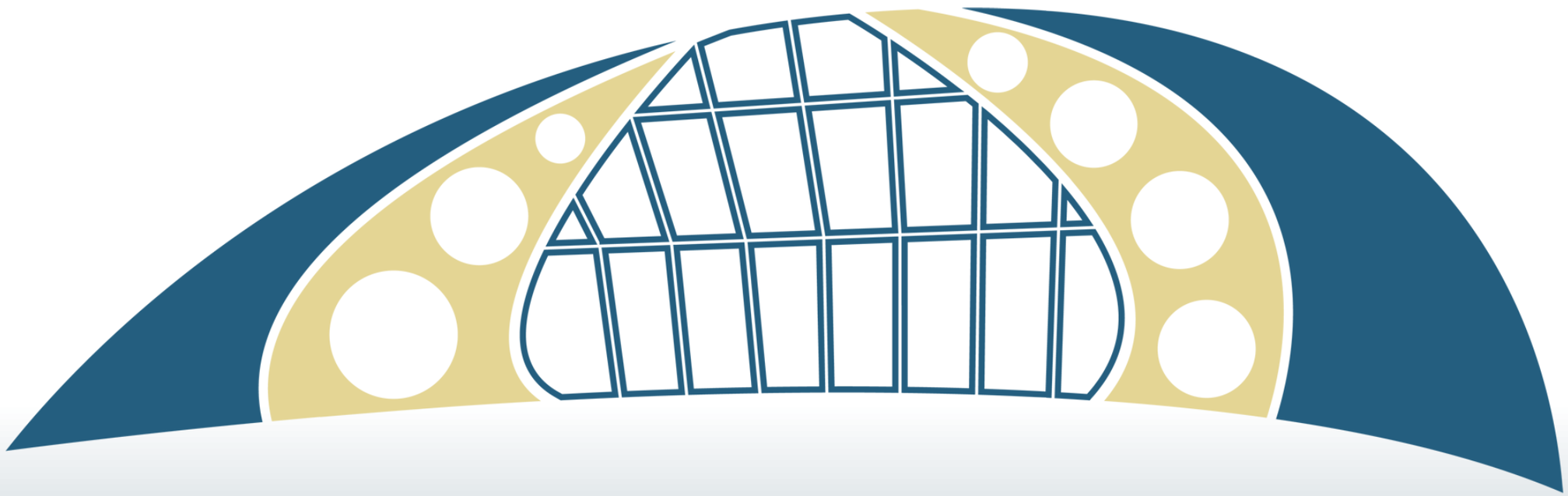
- "I am not crazy"
  - Netflix
  - Machine translations
    - Just like "bat" and "chiroptera" map the same
    - "bat" and "murciélago" can map to the same thing

The math is hard but it's beautiful and powerful

La matemáticas es dura pero es hermosa y de gran alcance

Jene mathematisch ist hart, aber ist und an langer Reichweite schön

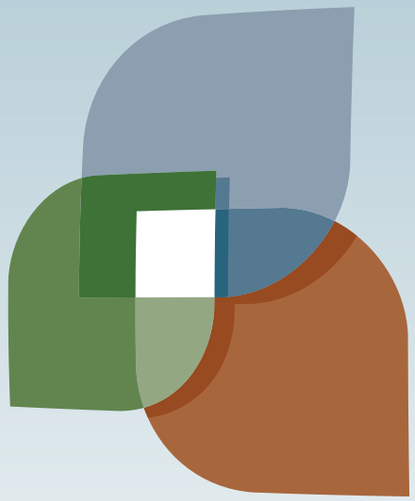
That one mathematically is hard, but is beautiful and at long range



WESTMONT INSPIRED

COMPUTING LAB

next...



LUCI

VECTOR SPACE SCORING

Introduction to Information Retrieval

CS 150
Donald J. Patterson

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

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 SCORING

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

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

- 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 SCORING

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 SCORING

VECTOR SPACE MODEL

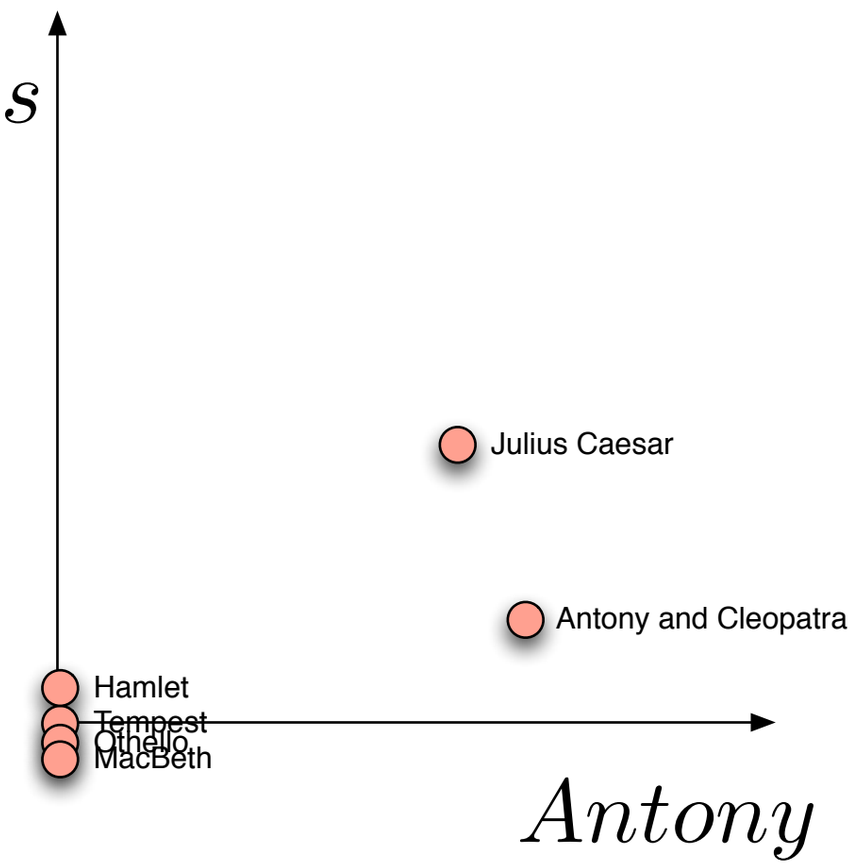
- Recall our Shakespeare Example:

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

VECTOR SPACE MODEL

- Recall our Shakespeare Example:



VECTOR SPACE SCORING

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 SCORING

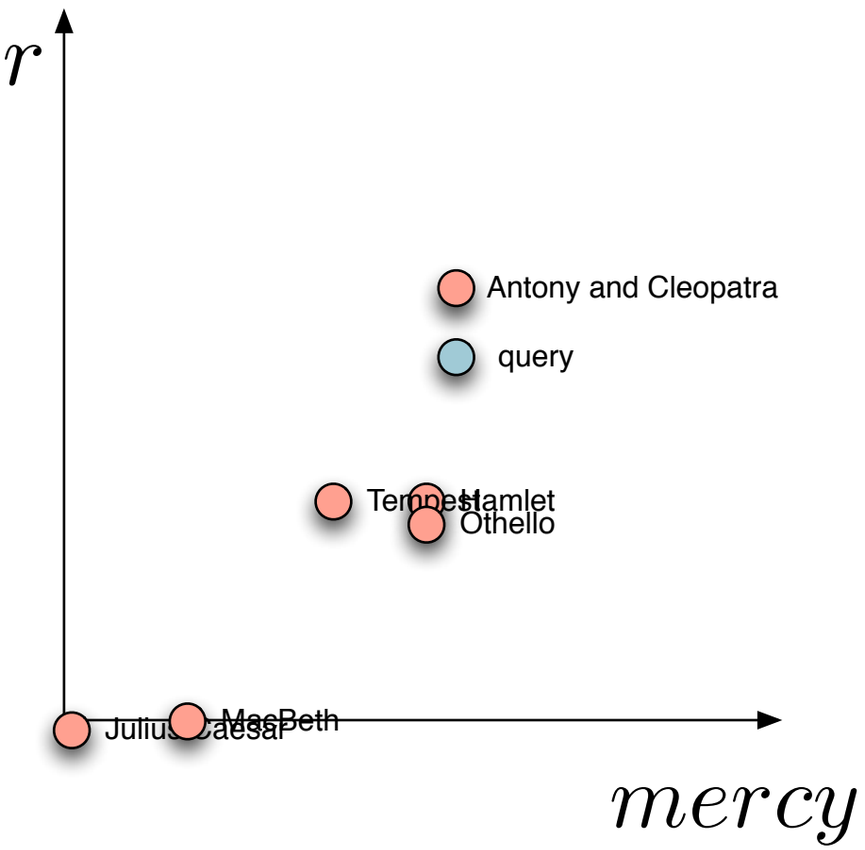
VECTOR SPACE MODEL

- Recall our Shakespeare Example:

VECTOR SPACE SCORING

QUERY AS A VECTOR

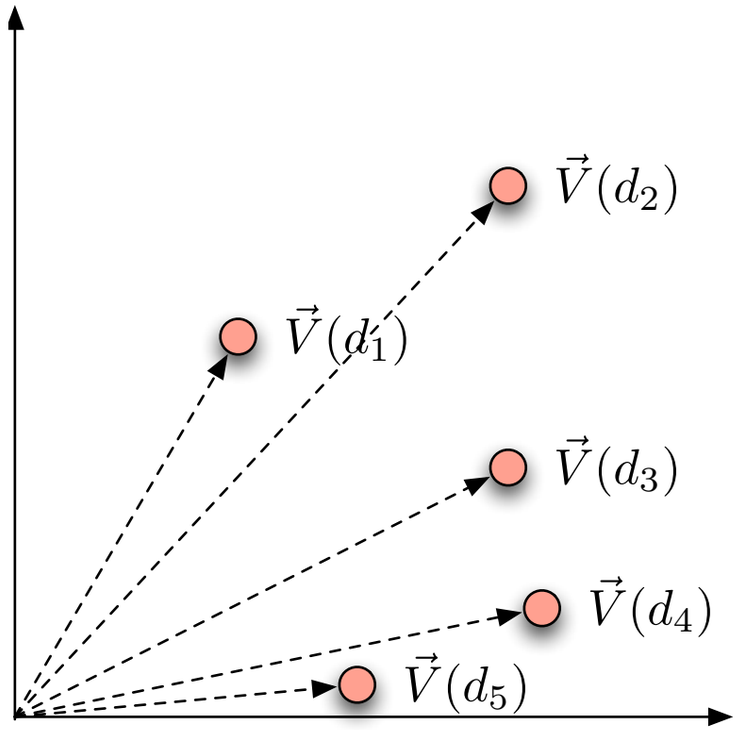
- So a query can also be plotted in the same space
  - "worser mercy"
  - To score, we ask:
    - How similar are two points?
- How to answer?



VECTOR SPACE SCORING

SCORE BY MAGNITUDE

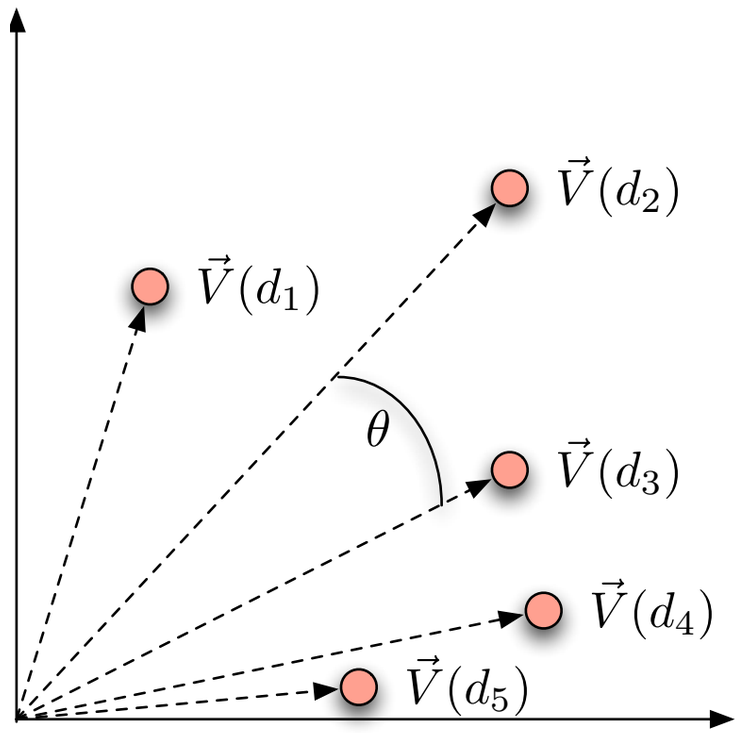
- How to answer?
  - Similarity of magnitude?
    - But, two documents, similar in content, different in length can have large differences in magnitude.



VECTOR SPACE SCORING

SCORE BY ANGLE

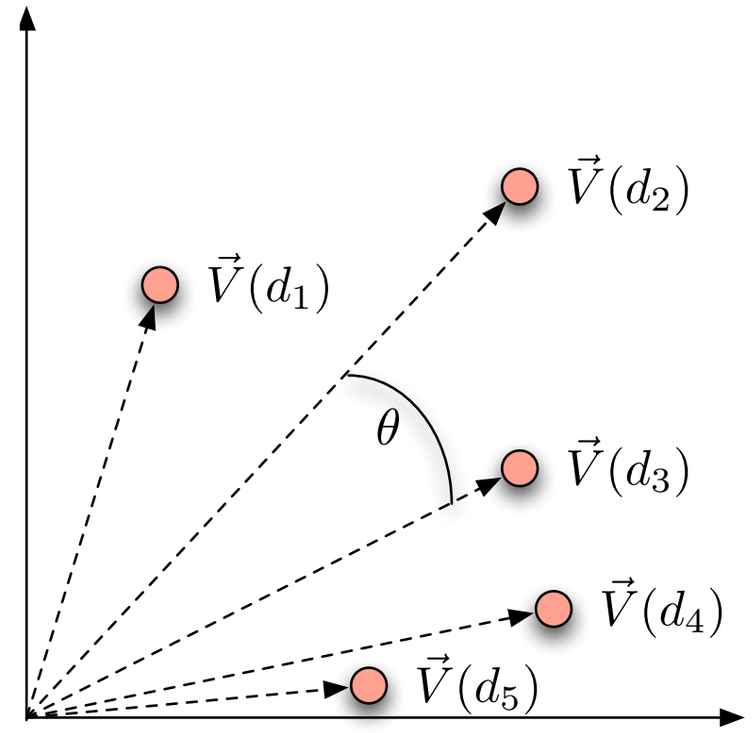
- How to answer?
  - Similarity of relative positions, or
  - difference in angle
    - Two documents are similar if the angle between them is 0.
    - As long as the ratios of the axes are the same, the documents will be scored as equal.
    - This is measured by the dot product

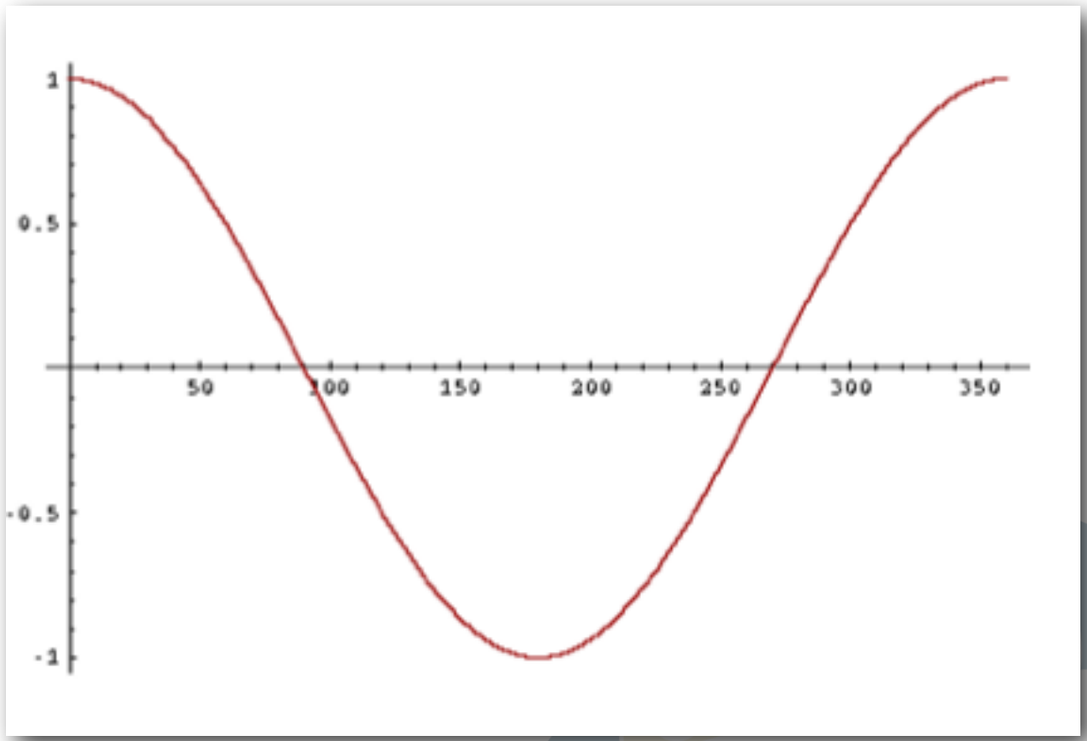


VECTOR SPACE SCORING

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)





VECTOR SPACE SCORING

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

VECTOR SPACE SCORING

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)

VECTOR SPACE SCORING

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!

VECTOR SPACE SCORING

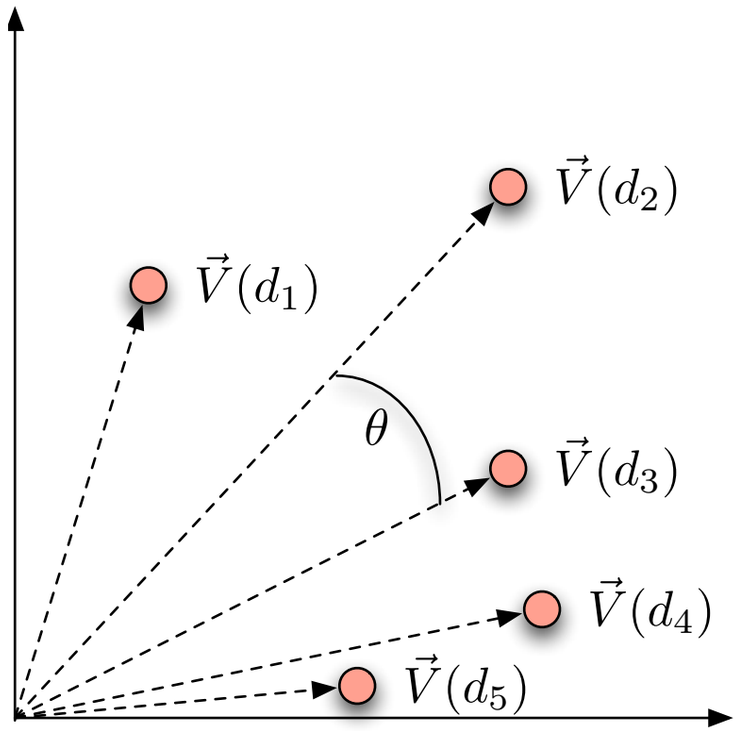
COSINE SIMILARITY SCORE

- Also called cosine similarity

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

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

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



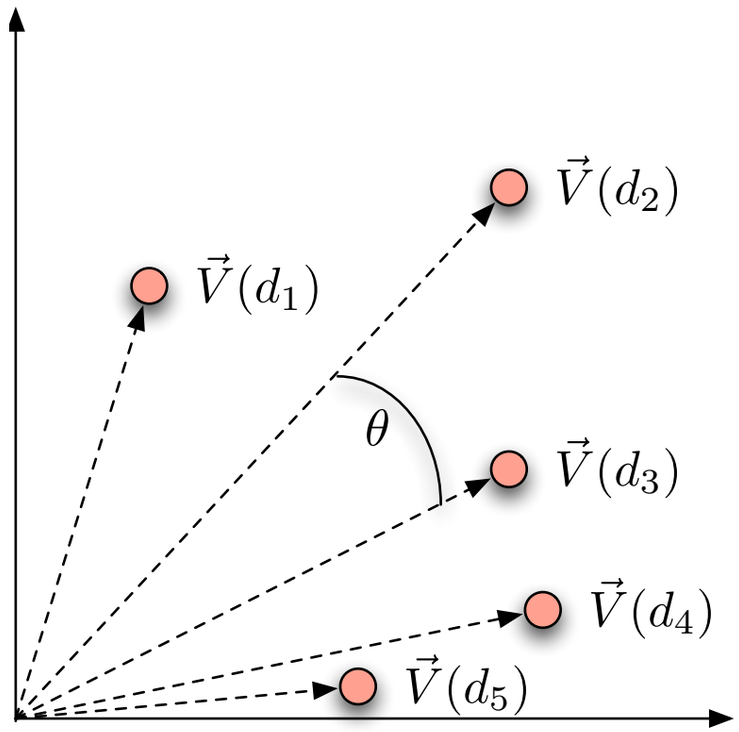
VECTOR SPACE SCORING

COSINE SIMILARITY SCORE

- Define: dot product

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

	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



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

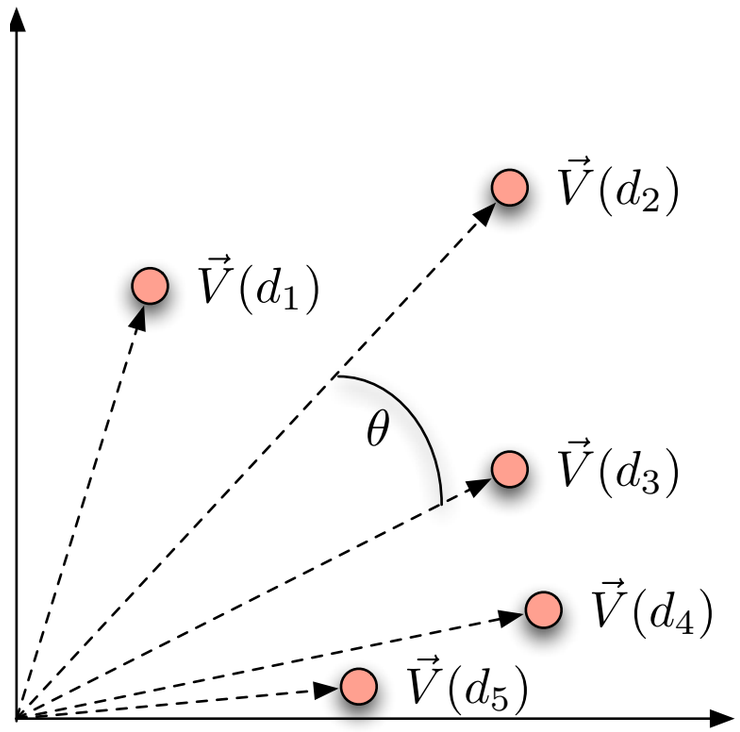
VECTOR SPACE SCORING

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



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

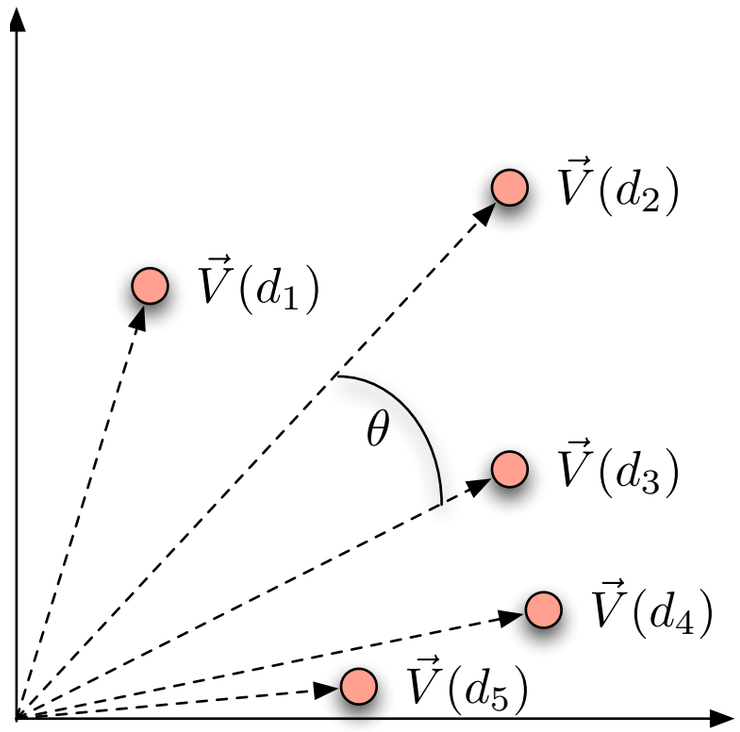
VECTOR SPACE SCORING

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



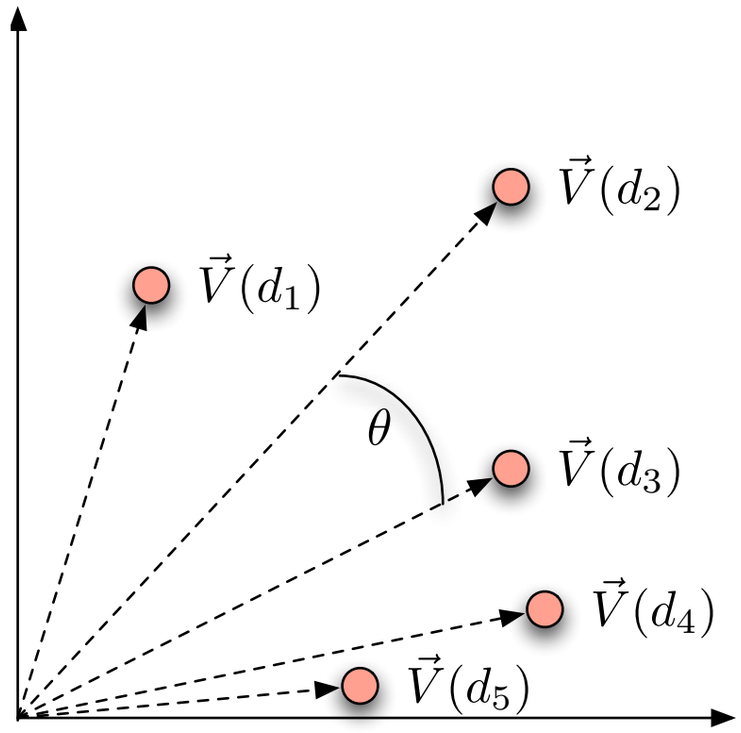
$$|\vec{V}(d_1)| = \sqrt{(11.4 * 11.4) + (8.3 * 8.3) + (2.3 * 2.3) + (11.2 * 11.2)}$$
$$= 18.15$$

VECTOR SPACE SCORING

COSINE SIMILARITY SCORE

- Example

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



VECTOR SPACE SCORING

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)

VECTOR SPACE SCORING

EXERCISE

tf =

24	24	0	24	24	24	24	24	24	24	24	24	24	24	24	14
10	10	0	10	10	10	10	10	10	10	10	10	10	10	10	14
24	24	0	24	24	24	24	24	24	24	24	24	24	24	24	14
12	12	0	11	11	11	11	11	11	11	11	11	11	11	11	14
10	10	0	10	10	10	10	10	10	10	10	10	10	10	10	14
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	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	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	2
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	2
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	2

df =

VECTOR SPACE SCORING

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
```

VECTOR SPACE SCORING

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 →

VECTOR SPACE SCORING

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         0         0    0.4439    0.4439
    0.4302    0.4302         0         0         0         0    0.4302    0.4302
         0         0         0         0         0         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         0         0
         0         0         0         0    3.9069         0         0         0
         0
         0
         0
```

```
>> 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
```

More of the same →

VECTOR SPACE SCORING

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    0.430                                    0.4302
         0         0         0                                             0
         0         0    3.9069                                             0
         0         0    3.9069                                             0
         0         0         0                                             0
         0         0         0                                             0
         0         0         0                                             0
         0                                                                  0
         0
```

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

num =

     0
```

More of the same →

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

denom =

    8.1765
```

```
>> score = num/denom

score =

     0
```

VECTOR SPACE SCORING

EXERCISE

```
tfidf =

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

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

num =

   25.3500
```

More of the same →

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

denom =

   38.5062
```

```
>> score = num/denom

score =

    0.6583
```

VECTOR SPACE SCORING

EXERCISE

- Rank the following by decreasing cosine similarity.
  - Assume tf-idf weighting:
    - Two docs that have 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)

```
>> score = num/denom

score =

    1.0000
```

```
>> score = num/denom

score =

     0
```

```
>> score = num/denom

score =

    0.6583
```

VECTOR SPACE SCORING

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

VECTOR SPACE SCORING

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.

VECTOR SPACE SCORING

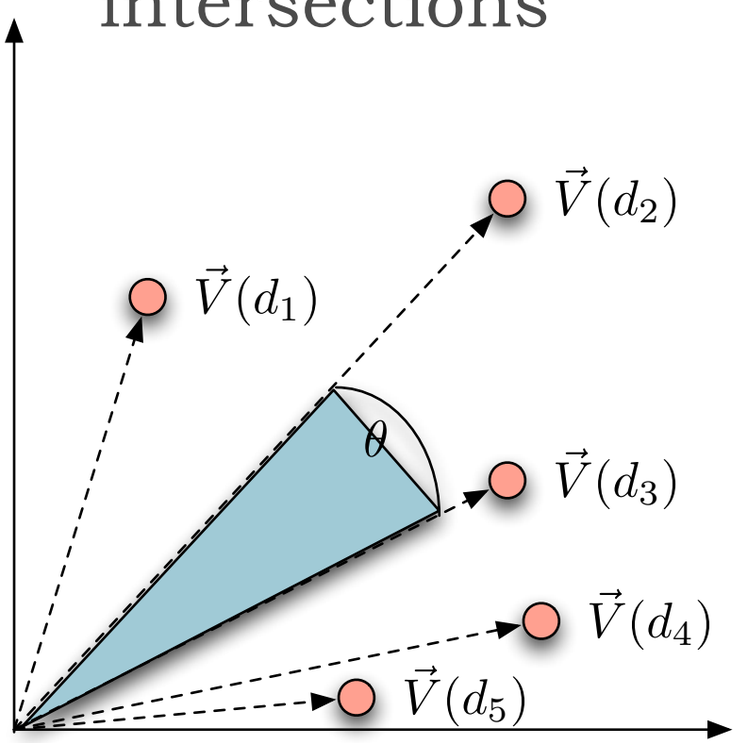
MULTIPLE QUERIES FOR PHRASES AND VECTORS

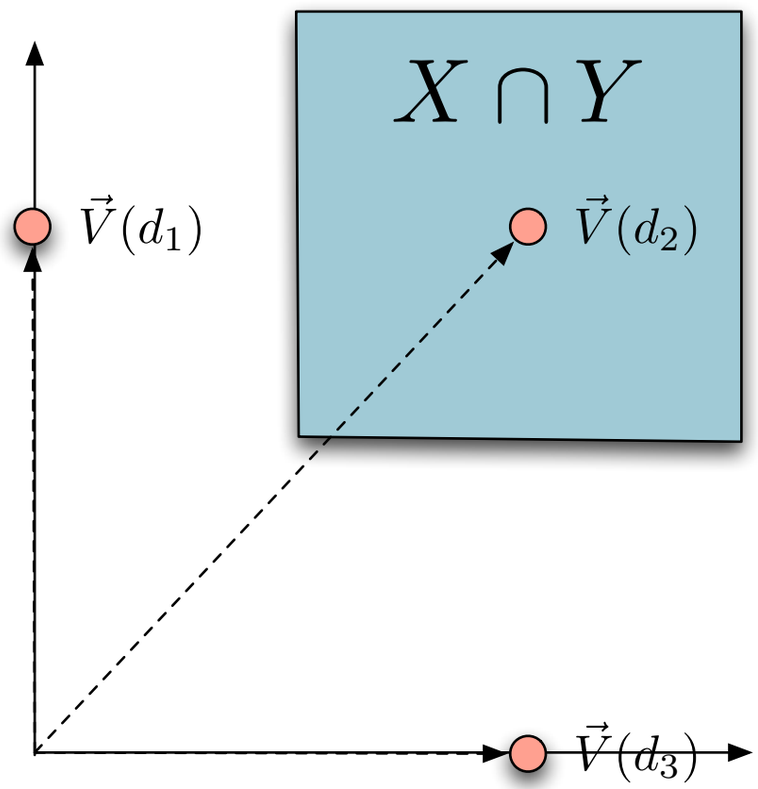
- Query: “rising interest rates”
- Iterative refinement:
  - Run the phrase query vector with 3 words as a term.
  - If not enough results, run 2-phrase queries and fold into results: “rising interest” “interest rates”
  - If still not enough results run query with three words as separate terms.

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”

VECTOR SPACE SCORING

ALTERNATIVES TO TF-IDF

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

$$\begin{array}{l}\mathrm{WTF}(t,d)\\ 1 \quad \textbf{if } tf_{t,d}=0\\ 2 \quad\quad \textbf{then } return(0)\\ 3 \quad\quad \textbf{else } \quad return(1+log(tf_{t,d}))\end{array}$$

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

- 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?

VECTOR SPACE SCORING

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)

$$A_j = \sum_t (w_{q,t} w_{d,t})$$

VECTOR SPACE SCORING

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

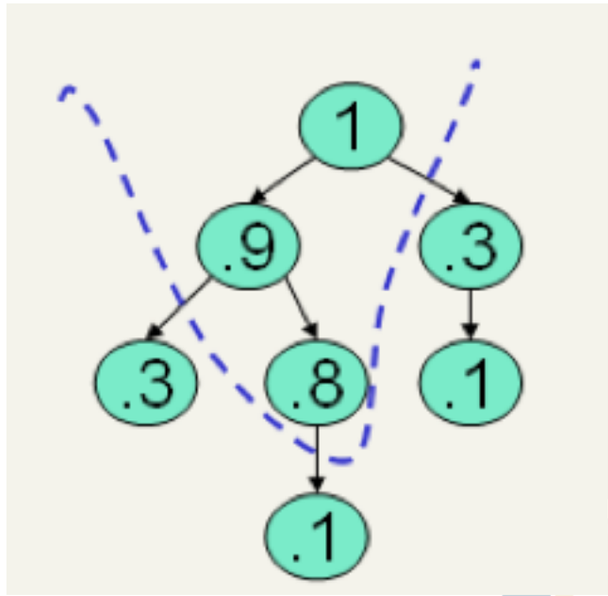
EFFICIENT COSINE RANKING

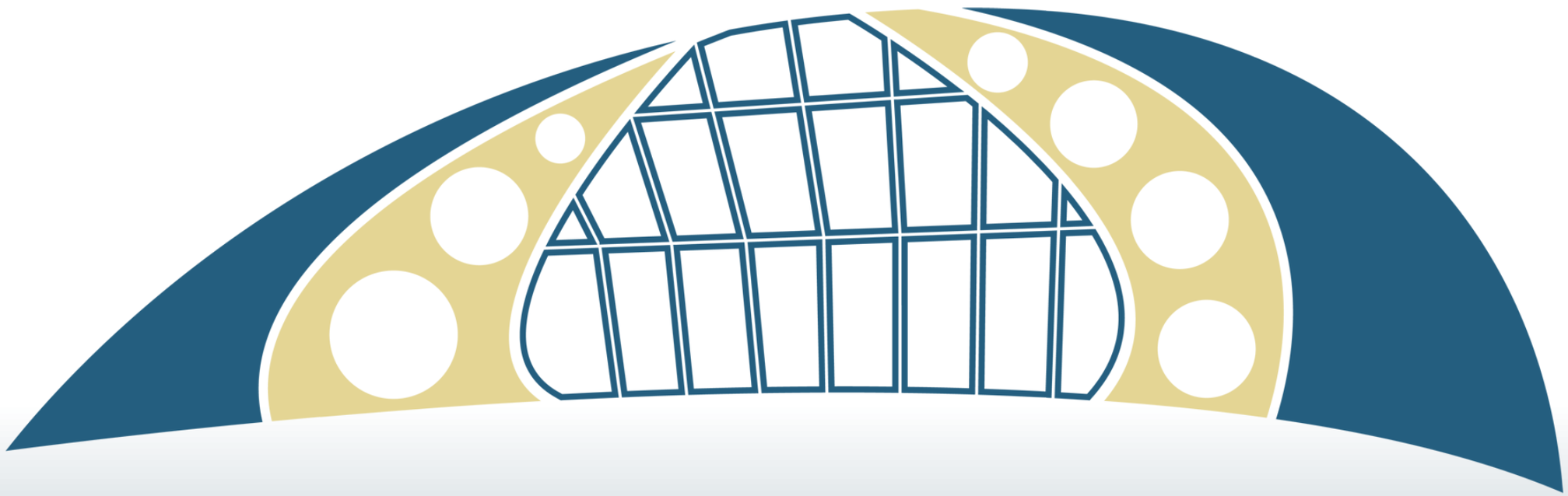
```
CosineScore(q)
 1  Initialize(Scores[d ∈ D])
 2  Initialize(Magnitude[d ∈ D])
 3  for each term(t ∈ q)
 4       do p ← FetchPostingsList(t)
 5          df_t ← GetCorpusWideStats(p)
 6          α_{t,q} ← WeightInQuery(t, q, df_t)
 7          for each {d, tf_{t,d}} ∈ p
 8               do Scores[d] += α_{t,q} · WeightInDocument(t, q, df_t)
 9  for d ∈ Scores
10       do Normalize(Scores[d], Magnitude[d])
11  return top K ∈ 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