Acknowledgment: Manning, C.D., Raghavan, P. and Schütze, H., *Introduction to Information Retrieval, Cambridge University Press. 2008.*

INFO 624

Information Retrieval Systems

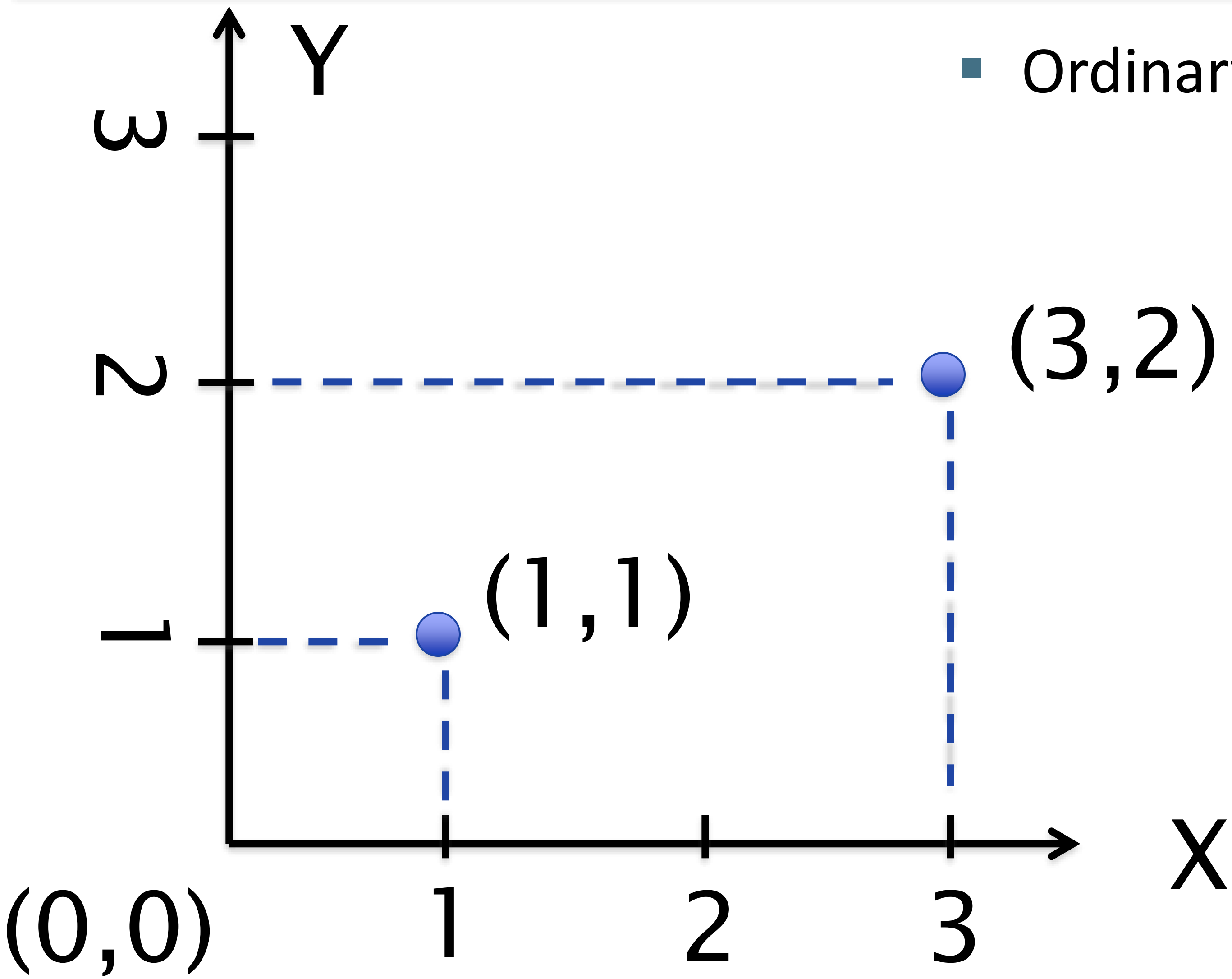
Vector Space Model:

TF*IDF Term Weighting and Cosine Scoring

Weimao Ke

wk@drexel.edu

Review – Coordinates



- Ordinary coordinates

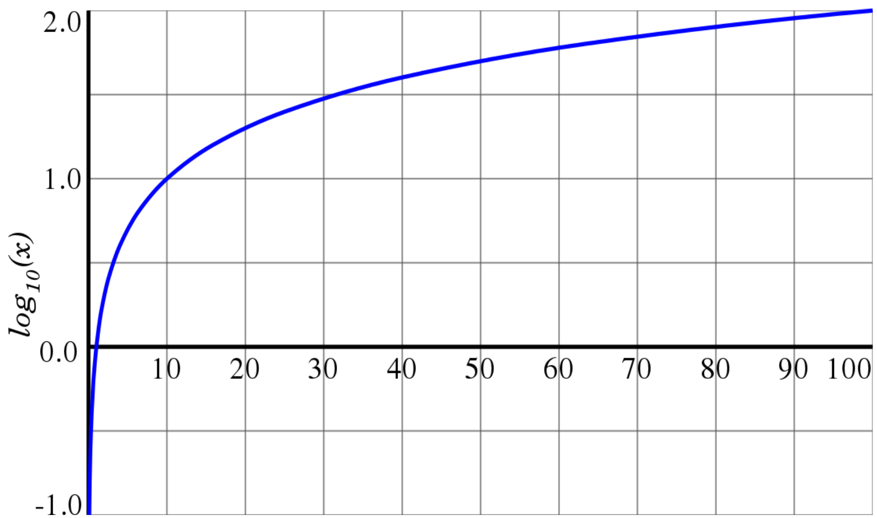
Review –Logarithm

- Logarithm: reduce a num to its order

$$\log(10^{2}) = 2$$

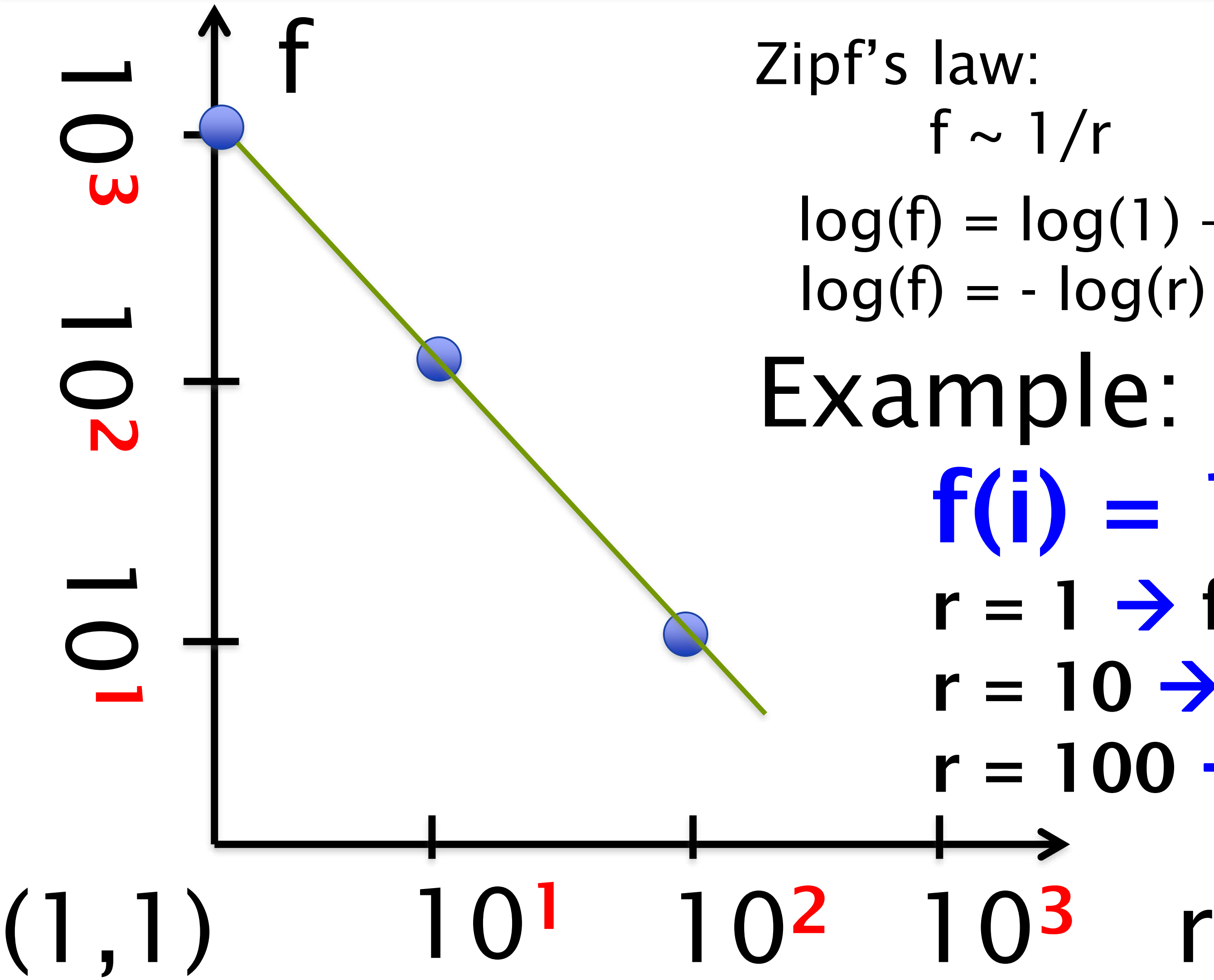
$$\log(10^{3}) =$$

$$\log(10^{4}) =$$



Source: [Wikipedia]

Review – log coordinates & Zipf's laws



Zipf's law:

$$f \sim 1/r$$

$$\log(f) = \log(1) - \log(r)$$
$$\log(f) = -\log(r)$$

Example:

$f(i) = 1000/r$

$r = 1 \rightarrow f = 1000$

$r = 10 \rightarrow f = 100$

$r = 100 \rightarrow f(1) = 10$

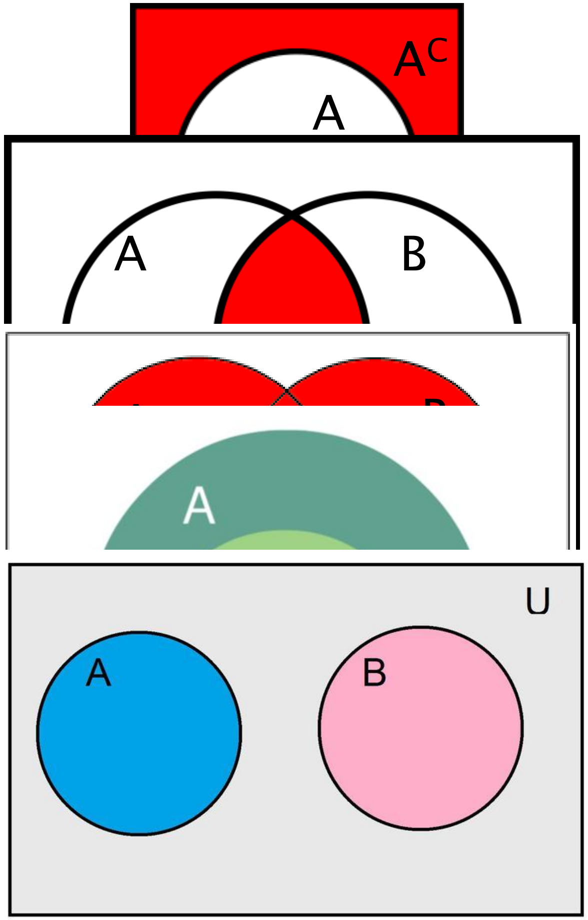
This lecture

- Preparation: Sets and Vectors
- Ranked retrieval
- Scoring documents
  - Term frequency and weighting schemes
  - Vector space scoring (cosine similarity)

SETS AND VECTORS

Sets: Definitions

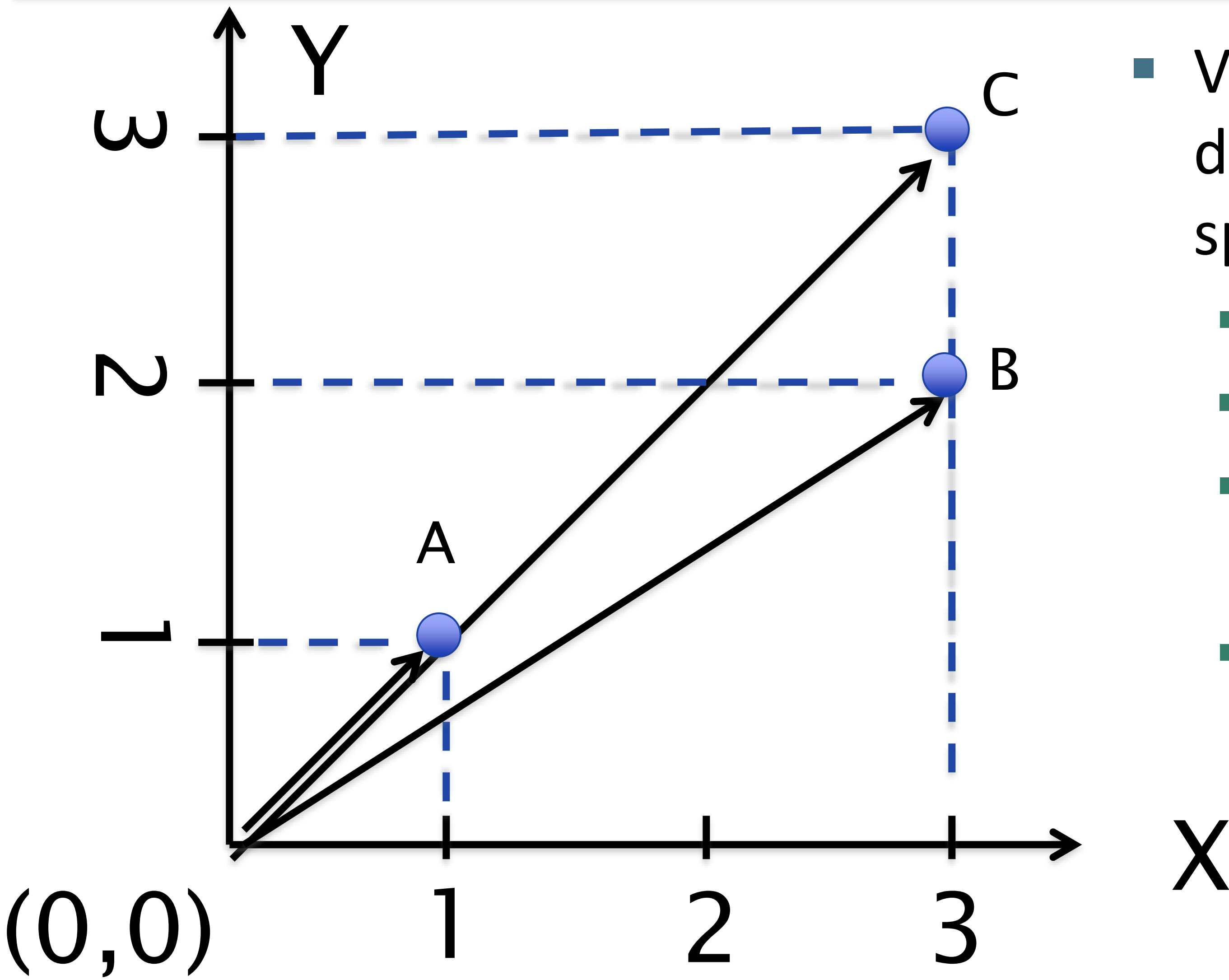
- What is a *set?*
  - a well-defined collection of objects or elements
  - e.g., A = {red, white, blue}
- The *universal set*: **U**
  - a complete set of objects or elements.
- The *complement* of A: $\mathbf{A^c}$
  - consists of all elements in **U** that are not in **A**
- The *intersection* of two sets **A** and **B**: $\mathbf{A \cap B}$
  - the set of elements that belongs to both **A** and **B**
- The *union* of two sets A and B: $\mathbf{A \cup B}$
  - the set of elements that belongs to either **A** or **B**.
- The *cardinality of* ***A***: **|A|**
  - the number of elements in **A**
- The *empty* set: $\varnothing$ **or {}**
  - contains no element
- If all elements of set **B** are elements of set **A**,
  - **B** is a *subset of* **A**, denoted $\mathbf{B \subset A}$,
- If no element of one set is an element of the other
  - Two sets **A** and **B** are *mutually exclusive* : $\mathbf{A \cap B = \varnothing}$



Vectors: Definitions

- A *vector* $\vec{D}$
  - combination of a magnitude and a direction.
- A *unit vector*
  - a vector having the magnitude of 1.
- Two vectors $\vec{A}$ and $\vec{B}$ are *equal* if they have the same magnitude and direction.
- Examples: force, speed

Vector examples



- Vectors in a 2-dimensional space:
  - A = (1,1)
  - B = (3,2)
  - C = (3,3)
  - C = 3A

RANKED RETRIEVAL

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- It takes a lot of skill to come up with a query that produces a manageable number of hits.

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in ranked retrieval models, the system returns an ordering over the (top) documents in the collection with respect to a query
- Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval models have normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We may just show the top *k* (≈ 10) results (at a time)
  - We don't overwhelm the user

  - Premise: the ranking algorithm works

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful (relevant) to the searcher
- How can we rank/order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match".

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
  - If the query term does not occur in the document: score should be 0
  - The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Scoring based on set operations

- Documents and query as sets:
- The *similarity* (*measure of association*) of two sets **A** and **B**
  - Simple matching function

$$SIM(A,B) = |A \cap B|$$

  - Dice's coefficient

$$SIM(A,B) = \frac{2|A \cap B|}{|A| + |B|}$$

  - Jaccard's coefficient

$$SIM(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Jaccard coefficient

- A commonly used measure of overlap of two sets *A* and *B*
  - jaccard(A,B) = $|A \cap B| / |A \cup B|$
  - jaccard$(A,A) = 1$
  - jaccard$(A,B) = 0$ if $A \cap B = 0$
- *A* and *B* don't have to be the same size.

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
  - <u>Query</u>: *ides of march*
  - <u>Document</u> 1: *caesar died in march*
  - <u>Document</u> 2: *the long march*

$$SIM(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Query: A = {ides, of, march}
For doc 1: B = {caesar, died, in, march}

$$SIM_1(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|\{march\}|}{|\{ideas, of, march, caesar, died, in\}|} = \frac{1}{6}$$

For doc 2: B = {the, long, march}

$$SIM_2(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|\{march\}|}{|\{ideas, of, march, the, long\}|} = \frac{1}{5}$$

Issues with Jaccard for scoring

- It doesn't consider *term frequency*
  - (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms.
  - Jaccard doesn't consider this information
  - We need a more sophisticated way of normalizing for length

TERM WEIGHTS

Binary term-document incidence matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	0
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

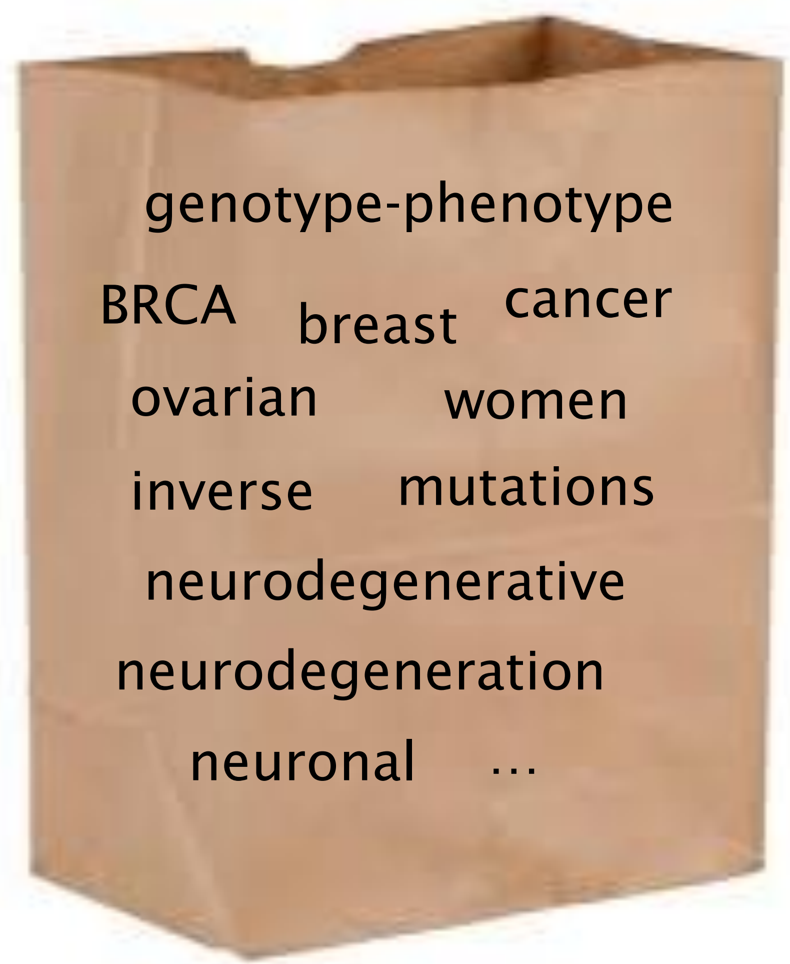
Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector : a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Bag of words model

Genotype-Phenotype Correlations in BRCA Mutation Carriers
Breast cancer following ovarian cancer in BRCA mutation carriers
Breast cancer, BRCA mutations, and attitudes regarding pregnancy
Surgical management of breast cancer in BRCA-mutation carriers
Cancer risk management decision making for BRCA women
Inverse association between cancer and neurodegenerative disease
Molecular neurodegeneration: basic biology and disease pathways
Mechanisms of neurodegeneration and axonal dysfunction
Dysfunction of neuronal calcium signaling in neuroinflammation and neurodegeneration
Epigenetic mechanisms of neurodegeneration in Huntington's disease



Bag of words model

- Doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.
  - A major IR approach
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
  - But maybe it is good enough in many cases

Term frequency tf

- The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d}, & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

  - 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.
- Score for a document-query pair:
  - Sum over terms *t* in both *q* and *d*:
  - Score(q,d) $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
    - The score is 0 if none of the query terms is present in the document.

Document frequency

- Consider the following terms:

  Instrument → Violin → Stradivari

  (broad, frequent) → (specific, rare)

- → We want a high weight for rare terms

Document frequency

- Rare terms are more informative than frequent terms
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
  - A document containing this term (rare and specific) is very likely to be relevant to the query *arachnocentric*
- Consider a term in the query that is common in the collection (e.g., *the, a, I, etc.*)
  - These common terms are not likely to be helpful to judge the relevance between the query and documents
- → We want a high weight for rare terms

Document frequency, continued

- Frequent terms (used frequently in many documents) are less informative than rare terms
  - Recall stop words: too frequent to be useful…
  - Consider a query term that is frequent in the collection (e.g., *high, under, go*)
- We will use document frequency (DF) to capture this

higher weights

lowers weights

rare terms
(small DF)

frequent terms
(small DF)

idf weight

- df_t is the document frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $\text{df}_t \leq N$ *(the total number of documents in a collection)*
- We define the idf (inverse document frequency) of t by

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

  - We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

the base of the log doesn't matter

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

There is one idf value for each term t in a collection.

tf-idf weighting

- The tf-idf weight of a term is the product of its TF weight and its IDF weight.

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

or $$w_{t,d} = \mathrm{tf}_{t,d} \times \log_{10}(N/\mathrm{df}_t)$$

  - Best known weighting scheme in information retrieval
  - Alternative names: tf.idf, tf x idf, TF*IDF, etc.
  - Increases with the number of occurrences within a document
  - Increases with the rarity of the term in the collection

Ranking of documents for a query

- One approach to ranking based on TF*IDF

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

VECTOR SPACE & RANKING

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0	0	0	0
Brutus	3.0	8.3	0	1.0	0	0
Caesar	2.3	2.3	0	0.5	0.3	0.3
Calpurnia	0	11.2	0	0	0	0
Cleopatra	17.7	0	0	0	0	0
mercy	0.5	0	0.7	0.9	0.9	0.3
worser	1.2	0	0.6	0.6	0.6	0

Each document is now represented by a real-valued vector of tf-idf weights

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0	0	0	0
Brutus	3.0	8.3	0	1.0	0	0
Caesar	2.3	2.3	0	0.5	0.3	0.3
Calpurnia	0	11.2	0	0	0	0
Cleopatra	17.7	0	0	0	0	0
mercy	0.5	0	0.7	0.9	0.9	0.3
worser	1.2	0	0.6	0.6	0.6	0

Documents as vectors

- So we have a |V|-dimensional vector space
  - |V| is the number of unique terms in index
  - Terms are axes/coordinates/dimensions of the space
  - Documents are points or vectors in this space
  - Very high-dimensional
    - tens of millions of dimensions when you apply this to a web search engine
    - These are very sparse vectors - most entries are zero.

Queries as vectors

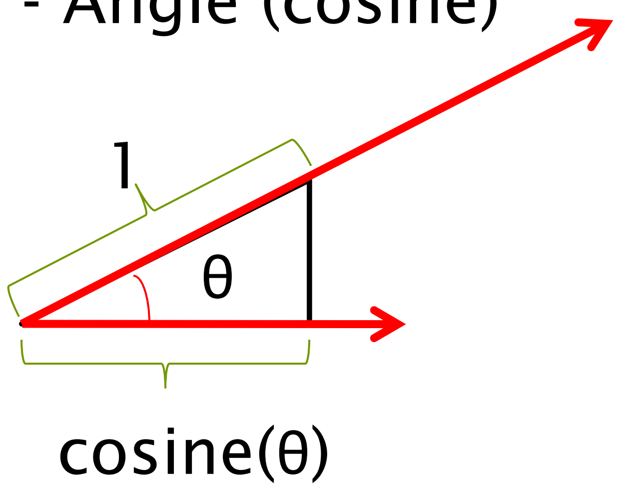
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
  - Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
  - Instead: rank more relevant documents higher than less relevant documents

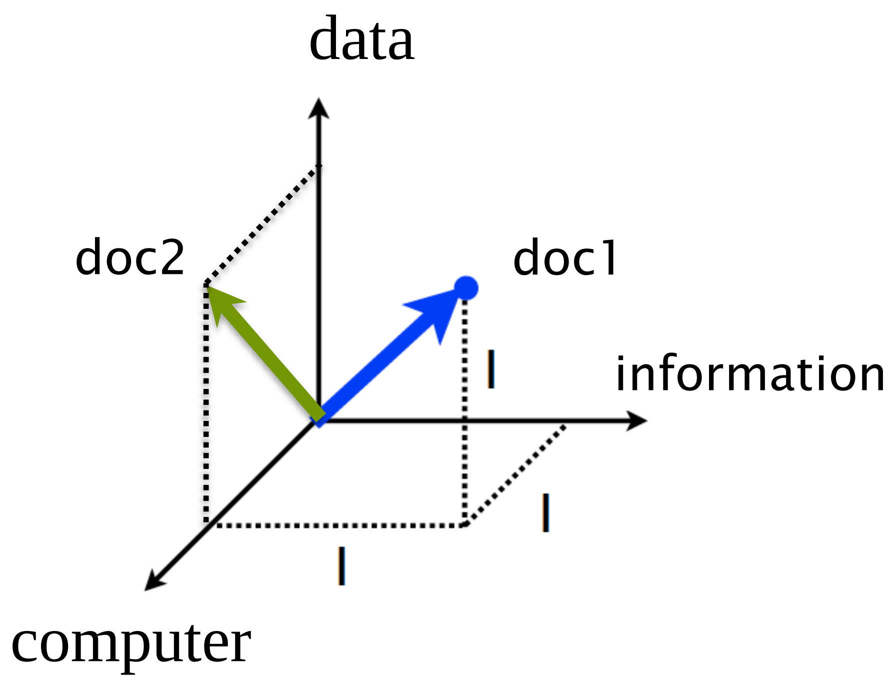
Vector Space Model

- Document / query representation
  - Terms as dimensions
    - using binary, frequency, or TF*IDF weights
  - Vector from the origin

(TF)	doc1	doc2
information	1	0
computer	1	1
science	1	1

- Scoring
  - Distance
  - Angle (cosine)



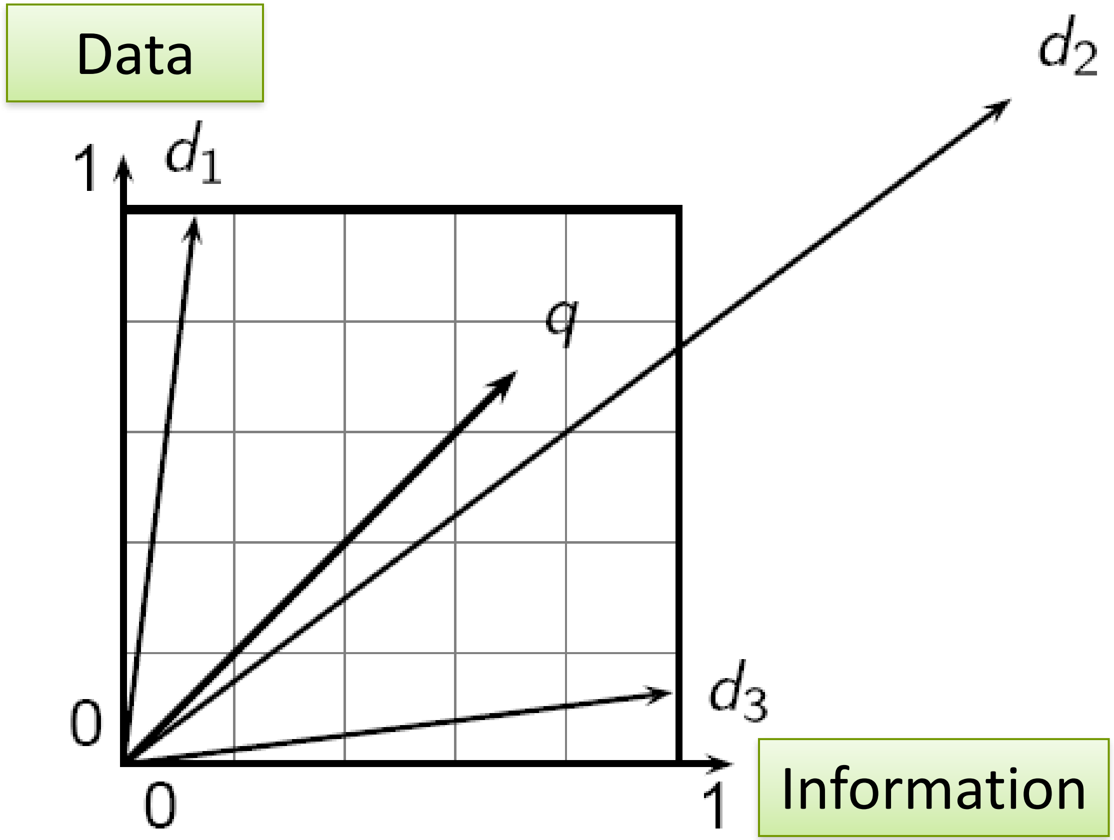


Formalizing vector space proximity

- First approach: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
  - Euclidean distance is large for documents of different lengths.
  - Why?

Why distance is a bad idea

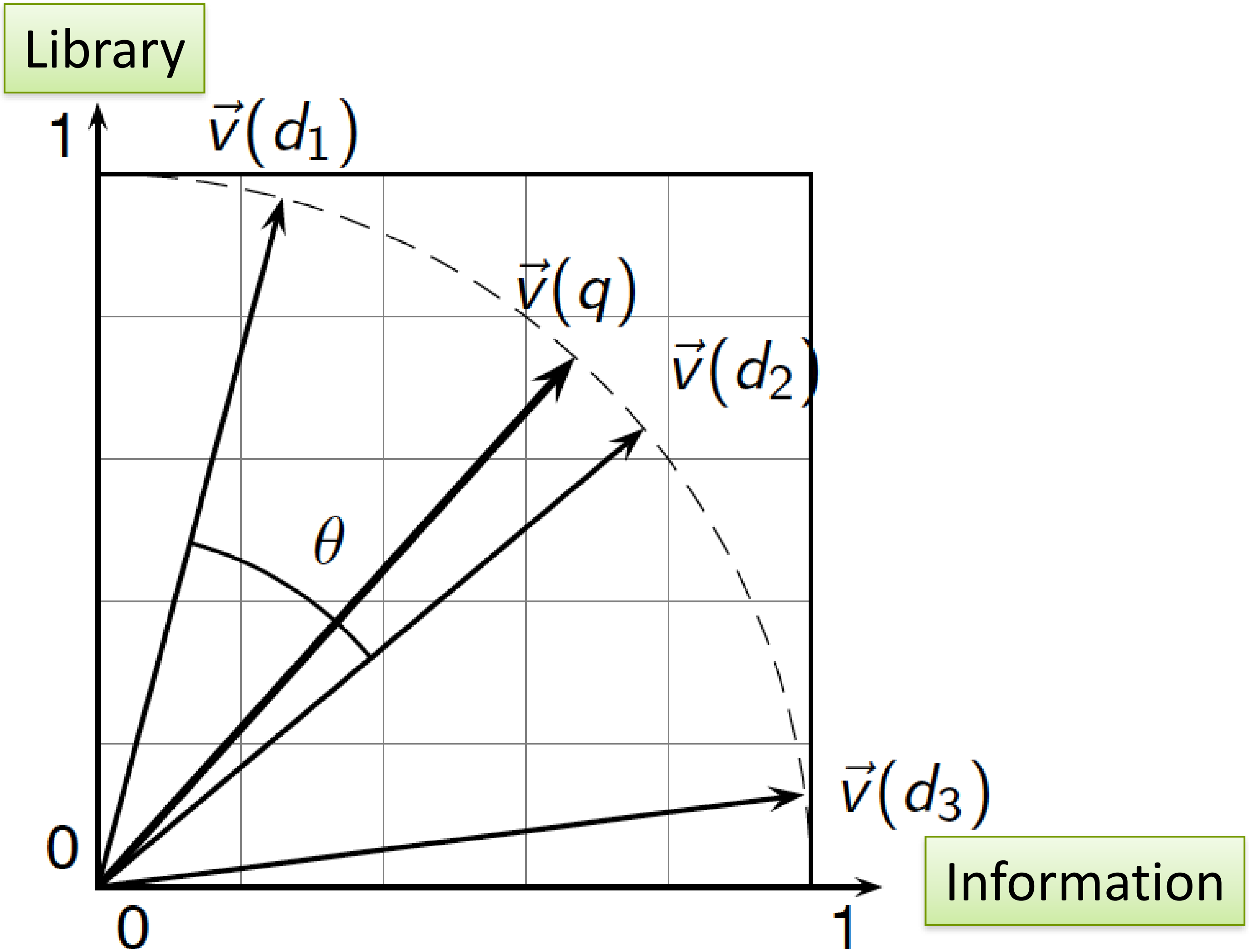
The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Use angle instead of distance

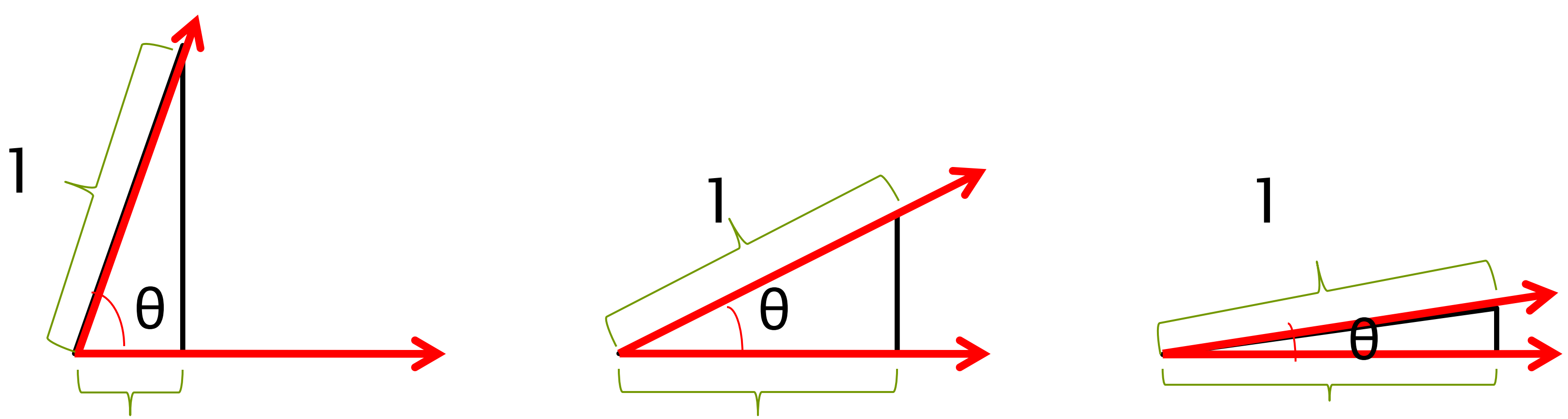
- Thought experiment
  - Take a document *d* and append it to itself.
  - Call this new document *d*' (everything doubled).
  - "Semantically" d and d' have the same content? Yes.
    - Cosines measure the directions: directions the same. Good.
    - Euclidean measure distance. Distance between the two docs?
- To measure how close/similar they are:
  - The Euclidean distance between the two documents can be quite large
  - The angle between the two documents is 0, corresponding to maximal similarity. (Cosine = 1)
  - Key idea: Rank documents according to angle with query.

Cosine similarity illustrated



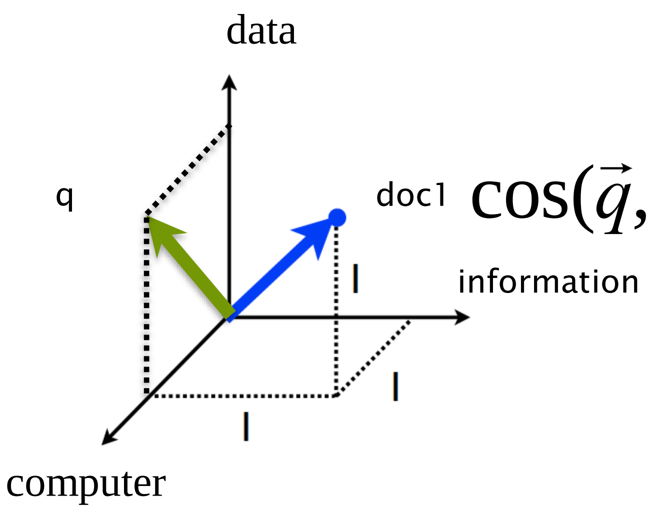
From angles to cosines

- The following two notions are equivalent.
  - Rank documents in increasing order of the angle between query and document
  - Rank documents in decreasing order of cosine(query, doc)
- The smaller the angle, the larger the cosine score



Small cosine cosine Large cosine

cosine(query,document)



$$\cos(\vec{q},\vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}}$$

(TF)	d	q
information	1	0
data	1	1
computer	1	1

q_i is the weight of term *i* in the query (e.g., using tf*idf)
d_i is the weight of term *i* in the document (e.g., using tf*idf)
n is the number of unique terms

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Vector space ranking

- Using TF*IDF and Cosine together:
  - Represent the query as a weighted tf-idf vector
  - Represent each document as a weighted tf-idf vector
  - Compute the cosine similarity score for the query vector and each document vector
  - Rank documents with respect to the query by score
  - Return the top *K* (e.g., *K* = 10) to the user

Exercise

Solution in **IRS03-VectorPracticeSolution.pdf**

$$\cos(\vec{q},\vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}}$$

- Query: Encryption Risk
- Documents:
  - D1: Risk Management for Security
  - D2: National Security Risk Assessment
  - D3: Encryption for Security in Bank Transactions
  - D4: Managing Security Risk with Encryption
- Terms/dimensions:
  - National, Security, Encryption, Risk
- Tasks:
  - Binary vector representation for each document
    - You could use TF*IDF but let's make it simple for now
  - Binary vector representation for the query
  - Cosine similarity score of each document to the query
  - Ranking of document according to their cosine scores

SUPPLEMENT

Vectors: Definitions

- The *magnitude* of **A**, denoted by |**A**|, is given by:

$$|A| = \sqrt{(A_x)^2 + (A_y)^2 + (A_z)^2}$$

- The dot (scalar) product of two vectors **A** and **B: A•B**
  - the product of the magnitude of **A** and **B** and the cosine of the angle θ between them
  - A•B = |A||B|COSθ= $A_1B_1 + A_2B_2 + \ldots + A_nB_n$

- In a 3-dimensional space, COSθ is given by:

$$\cos\theta = \frac{A \bullet B}{|A||B|} = \frac{A_xB_x + A_yB_y + A_zB_z}{\sqrt{(A_x)^2 + (A_y)^2 + (A_z)^2}\sqrt{(B_x)^2 + (B_y)^2 + (B_z)^2}}$$

- In an n-dimensional space, COSθ (*Cosine Similarity*) is given by:

$$\cos\theta = \frac{\sum_{i=1}^{n} A_iB_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

Text Analysis: Term Weighting

- Automatic Indexing
  - Zipf Distribution: rank*frequency = constant

Word	Frequency (f)	Rank (r)	r*f
I	2,653	1	2653
all	1,311	2	2622
but	926	3	2778
when	717	4	2868
…			
	1	2990	2990

- Resolving power
  - ability of terms to identify relevant items and to distinguish them from nonrelevant material
  - Luhn: Resolving power of a term peak in mid-frequency range
- Term Weighting
  - *tf·idf* formula
    - *tf* = term frequency
    - *idf* = inverse document frequency

$$w_{ki} = f_{ki} \log \frac{N_d}{d_k}$$

w_{ki} = weight of term k in document i
f_{ki} = frequency of term k in document i (*tf*)
N_d = number of documents in collection
d_k = number of documents in which term k appears (postings)

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
  - iPhone
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

Word	Collection frequency (total # documents)	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is a better search term (and should get a higher weight)?

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\left\|\vec{x}\right\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

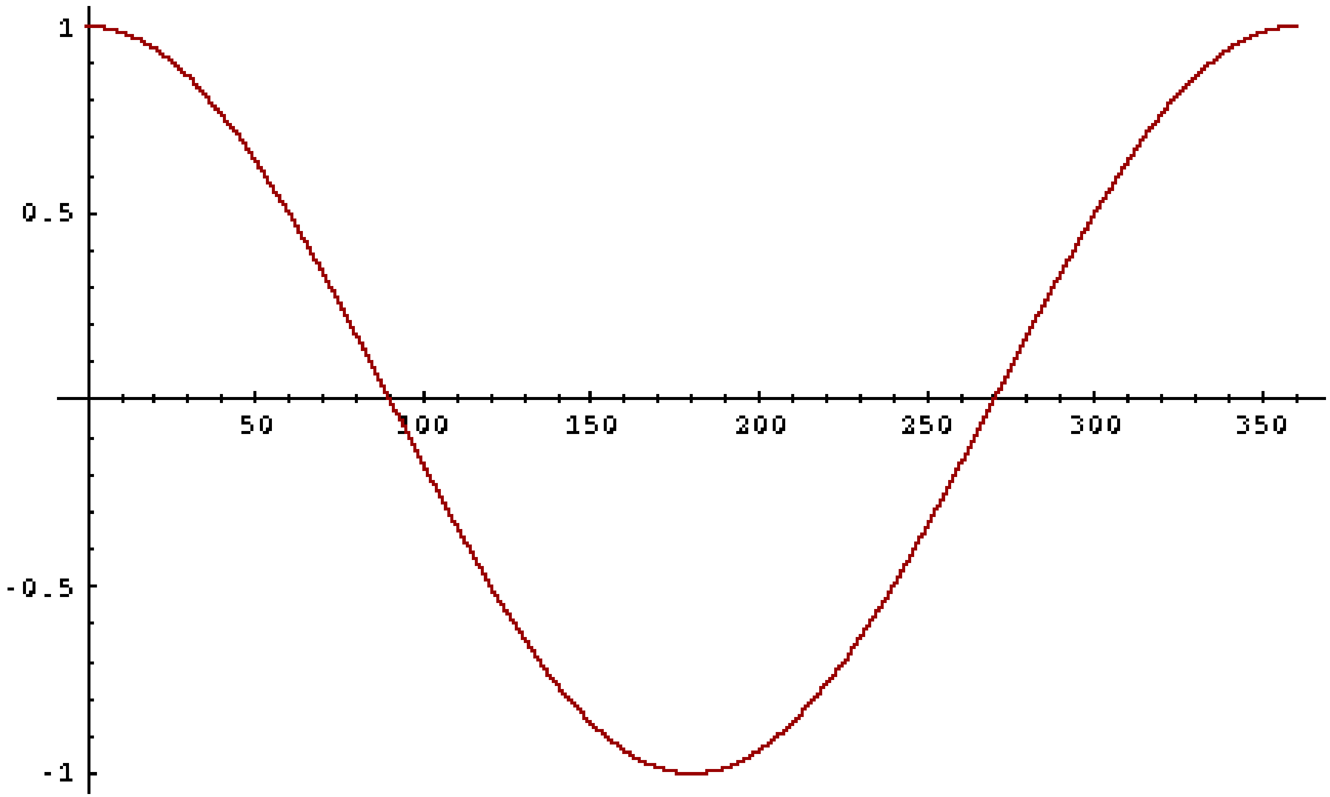
Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

$$\cos(\vec{q},\vec{d}) = \vec{q} \cdot \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

for q, d length-normalized.

From angles to cosines



- But how – *and why* – should we be computing cosines?

Cosine similarity amongst 3 documents

How similar are the novels

SaS: *Sense and Sensibility*

PaP: *Pride and Prejudice*, and

WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SAS,WH)?

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

tf-idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Exercise: what is *N*, the number of docs?

Doc length $= \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8