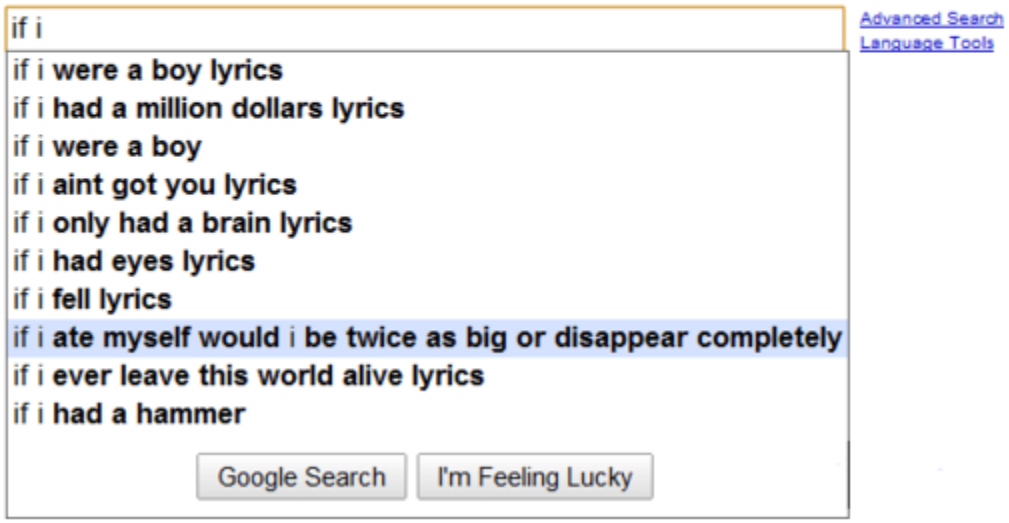
IR1: Introduction to Information Retrieval



Mid-Semester: Where are we?

- So far, we have talked about fundamentals of Web Systems
  - Protocols for communication (like HTTP, TCP)
  - Building blocks (request/response cycle, JSON objects, RESTful paradigm)
  - Web site behavior (GET/POST requests, servers, clients)
- You've done full stack development (P1, P2, P3)
  - Flask servers, HTML, CSS, JavaScript
  - React framework, node, pip
  - Virtual environments, bash scripting, selenium
- We've talked about distributed systems
  - MapReduce, GFS, Blockchain: applications of web systems beyond websites

Review: Blockchain

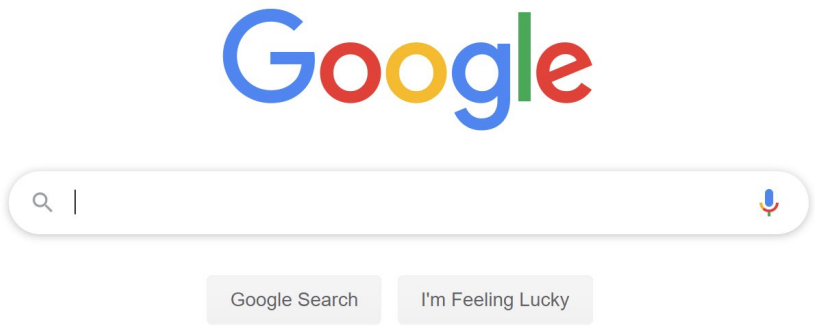
- **Blockchain** is a type of distributed system that enables **transactions** to occur within a **distributed ledger**
  - Blockchain allows for multiple participants in the system to gain a reliable view of the transaction ledger
  - Blockchain serves as the basis for enabling **cyptocurrencies** like Bitcoin and Litecoin
- Cryptocurreny: report transactions between *wallets*
  - Implementation: public key infrastructure. Each sender/recipient signs transactions
- Idea: **incentivize** participants in the network to correctly log transactions
  - (Loosely) transactions broadcasted everywhere
    - Miners try hashing batches of transactions
    - Eventually, one miner finds a hash, tells everyone else, providing a **proof of work**
    - That **block** gets added to the chain!

One-Slide Summary: Information Retrieval I

- The problem of **information retrieval** is construed as:
  - Given a **query document** and a **corpus of documents**, find all **documents** that relate to the query
    - Want *all* documents that relate to the query (no “false negatives”)
    - Want *only* the documents that relate to the query (no “false positives”)
- IR systems *rank order* documents from a corpus by relatedness
  - We pick the *top k* documents according to some algorithm
    - Spoiler: PageRank by Google might be a solid choice
- We **represent** documents as vectors of numbers to help develop **similarity** metrics
  - How related are these two documents?
  - Then, you rank them by similarity score

Google Search

- Somewhat profitable company you may have heard of
- Searching for things you want is **hard**
- Consider: if you search for *"Dijkstra's algorithm"*
  - If you search through *all documents*
    - Do you care about Dijkstra's personal history or biography?
    - Do you care about other algorithms?
    - Dijkstra's algorithm is graph-based… maybe you want other graph algorithms?



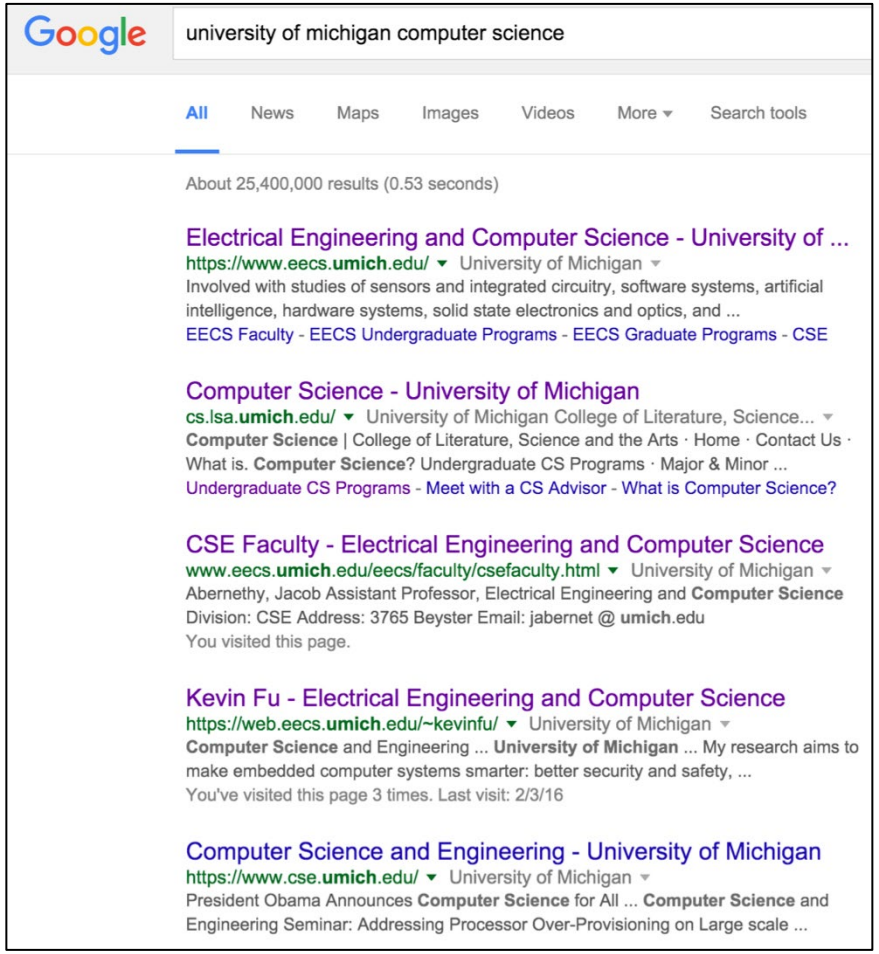
Document Search

- If you were Google, what would you need to do?
  - Search bar lets users type in a **query**
  - But what do you use as the document **corpus?**
    - Basically download the internet (good luck lol)
      - Can you do this dynamically?
    - For a future lecture: apply search engine optimization (SEO) to influence how your site gets represented in a search engine's corpus

- Given a **query**, iterate through all documents in the **corpus** (or **index**), compute **similarity** between query and document
  - Count occurrences of query tokens in each document?
  - Count documents where query appears one or more times?
  - Count documents where query *doesn't* appear?

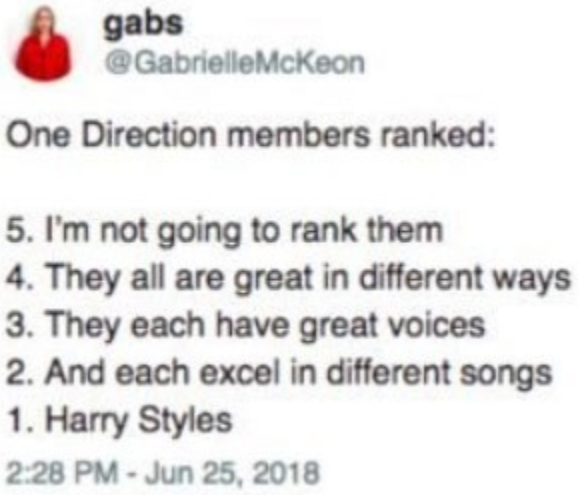
Key problem: ranking results

- 33% clicks on top result
  - "I'm feeling lucky"
  - It's not enough to get the best document as the 100th result
- Different ranking methods
  - Words on page
  - Importance of page using links



Goal of ranking algorithms

- Which web pages (documents) does the person searching want to find?
  - Given query “kangaroos”? “animals”?



Kangaroos live in Australia and jump.

Cows live all over the world. Unlike kangaroos, they cannot jump.

Aluminum foil is shiny.

Thought questions

- What is the role of the corpus/index? Why would searching be difficult without one?

- Why might the structure of the corpus/index change depending on which ranking algorithm we use?

Boolean Retrieval

- Does the **query** appear in a **document?**
  - Boolean decision. No counting, no sorting
  - “Kangaroos”

- Augment: composition of tokens in query
  - “Kangaroos AND NOT cows”

- Basically: evaluate a Boolean predicate for each document.
  - If true: document returned
  - If false: document ignored

Boolean Retrieval

- Query: **Kangaroos**

Kangaroos live in Australia and jump.	Cows live all over the world. Unlike **kangaroos**, they cannot jump.	Aluminum foil is shiny.
1	*1*	*0*

Boolean Retrieval

- Query: **Kangaroos** and NOT **cows**

Kangaroos live in Australia and jump.	**Cows** live all over the world. Unlike **kangaroos**, they cannot jump.	Aluminum foil is shiny.
1	*0*	*0*

Index Construction for Boolean retrieval

- **Inverted index**: words to documents

Document 0

Kangaroos can jump.

Document 1

Cows can not jump.

Term	Documents
kangaroo	0
can	0, 1
jump	0, 1
cow	1
now	1

Boolean Retrieval using Inverted Index

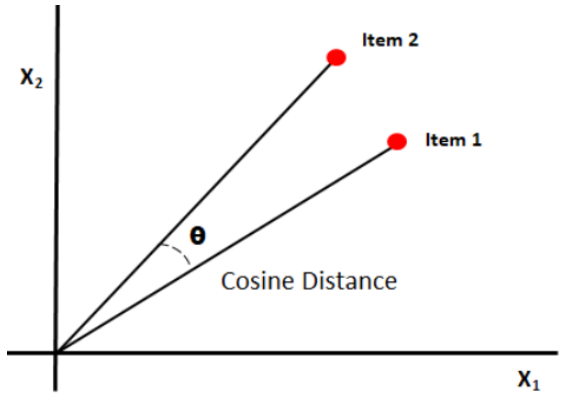
Term	Documents
kangaroo	0
can	0, 1
jump	0, 1
cow	1
now	1

- Given **kangaroo AND NOT cow**
  - “Kangaroo” -> document 0
  - “NOT cow” -> document 0
  - AND operation
    - Document 0
- What are the benefits/drawbacks of Boolean retrieval?
- What if you have a giant index?
  - Related: from earlier, why is an index necessary here?

Vector Space Model

- Boolean retrieval is not particularly rich
  - Yes/no per document, but not really a way to *rank* results

- We can **embed** documents into a **vector space** to allow relating two documents by measuring properties of the vector
  - Notably, what is the **angle** formed by the two vectors



Boolean index to vectors

Term	Documents
kangaroo	0
can	0, 1
jump	0, 1
cow	1
now	1

- Assign each *term* to be indices in a vector
  - For each document i, we assign $i_j = 1$ if document i contains term j
- Btw: what about ordering?
- Document 0: [1, 1, 1, 0, 0]
- Document 1: [0, 1, 1, 1, 1]
- You can work backwards: [1, 0, 1, 0, 1] = ?

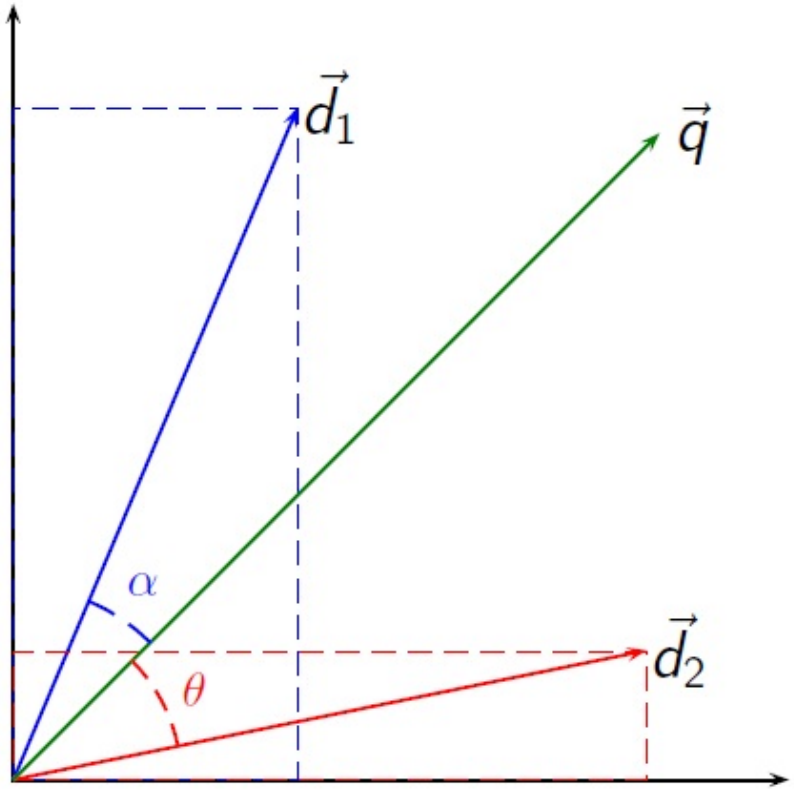
Documents as vectors

- A document is a vector
- Each dimension represents a word
  - Doesn't capture order
  - Doesn't capture counts
- # of dimensions: # of unique words in *all* documents
  - How many words are there?
  - Also, for a future lecture: should "kangaroo" and "kangaroos" be treated the same?

Document Similarity in Vector Space

- Want: Number that captures how “close” two vectors are
- *Vector space similarity* can be represented with a **cosine** of the angle between two vectors

- Recall that cosine:
  - Depends on two adjacent vector lengths
  - =1 when angle is zero
    (points are identical)
  - *Smaller* when angle is *greater*
  - *Larger* when angle is *lesser*



Vector space similarity (and demo)

- Euclidean dot product formula for computing cosine similarity

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

$$Sim(Di,Dj) = \sum_{k=1}^{t} w_{ik} * w_{jk}$$

- (protip: import numpy and import math)

Thought Questions

- If the query is "cows are cool", is the word "cool" represented in the vector made from the query? Does that matter?
- What is the index data structure if we're using vectors to represent documents?
- In general, will there be more 0s or 1s in most vectors?

Thought Questions

- If the query is "cows are cool", is the word "cool" represented in the vector made from the query? Does that matter?
  - If a query contains a token ***not*** represented in the index, you *ignore it*
  - Whether it matters depends or not. If a user searches for “cows are cool,” perhaps they *don’t* want an article about “cows are awful”
- What is the index data structure if we're using vectors to represent documents?
- In general, will there be more 0s or 1s in most vectors?
  - 1’s are pretty sparse (consider: how many words are there?)

Next: Adding More Information to Vectors

- Right now, vectors contain only 0 or 1 depending on whether they contain a word

```
doc1 = [1, 1, 1, 0, 0]
```

- What else can be used to enhance the information conveyed by the vector?
  - Recall our goal: compare *query* to *document* for ranking **similarity**
- Cunning plan: let's try using real numbers instead of 0/1

Term frequency

- We can count the number of times a **term** appears in a document
  - A **term frequency** is the number of times a word appears in each document!

Kangaroos can jump. Kangaroos live in Australia.

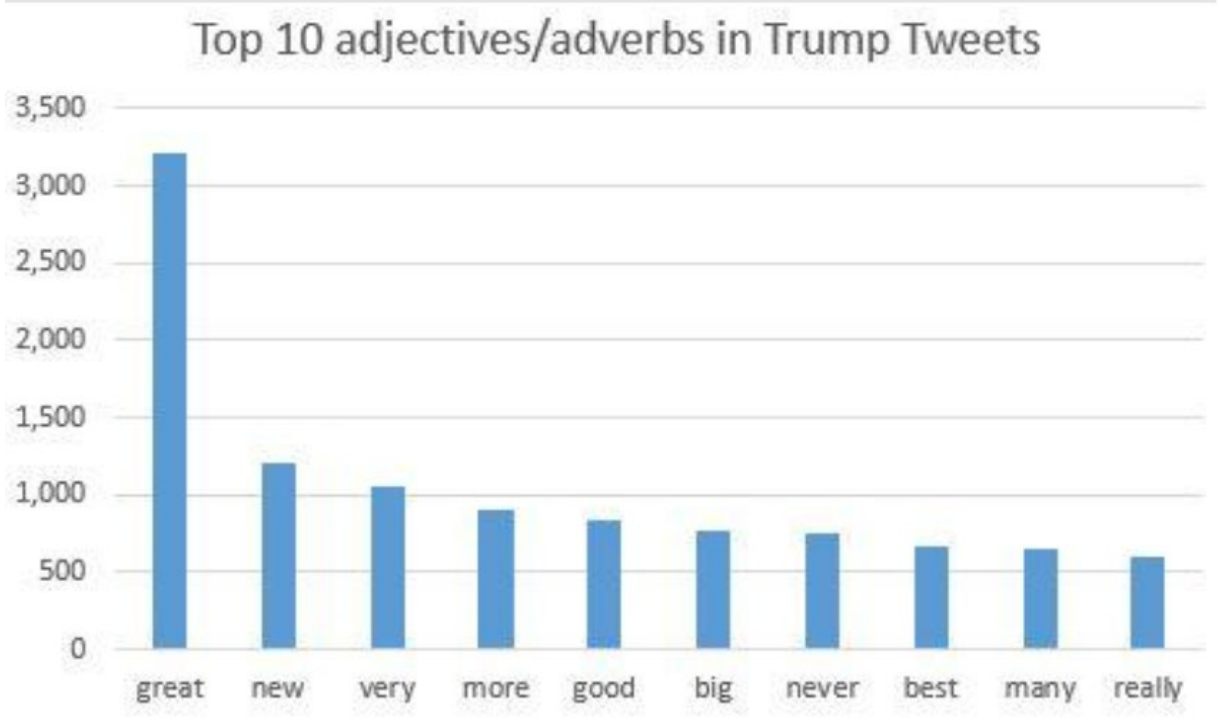
Term	Term Frequency
Kangaroos	2
Can	1
Jump	1
Live	1
In	1
Australia	1

Unlike kangaroos, cows cannot jump. Cows like eating grass.

Term	Term Frequency
Unlike	1
Kangaroos	2
Cows	2
Cannot	1
Jump	1
Like	1
Eating	1
Grass	1

Problems with Term Frequency

- Across large corpora, some words appear more frequently than others
  - Consider transcriptions of Kevin's lectures. How many times does "uh" appear?



Word	Parts of speech	OEC rank	COCA rank[8]	Dolch le
the	Article	1	1	Pre-primer
be	Verb	2	2	Primer
to	Preposition	3	7, 9	Pre-primer
of	Preposition	4	4	Grade 1
and	Conjunction	5	3	Pre-primer
a	Article	6	5	Pre-primer
in	Preposition	7	6, 128, 3038	Pre-primer
that	Conjunction et al.	8	12, 27, 903	Primer
have	Verb	9	8	Primer

Problems with Term Frequency

- Since “the” occurs so frequently, it will bias vectors
  - Two unrelated vectors may have a large component due to “the”
  - Below: TFs of common words in “fake news” tweets. Note “the”, “to”, and “of” dominate all the vectors, even if each tweet is unrelated

1	the	to	of	and	a	that	trump	in	s	is	he	it	for	his	on	with	was	this	be	t	as	you	has	not	have	are	the
2	13	14	6	14	14	4	8	2	11	2	9	2	7	4	0	0	0	5	2	6	4	8	2	1	3	1	
3	16	9	6	5	11	5	6	3	4	4	4	3	2	1	2	5	3	0	0	3	3	0	1	1	1	0	
4	18	17	10	8	11	5	1	7	5	5	3	8	3	3	3	7	4	2	2	6	0	7	2	2	2	3	
5	24	10	8	6	6	8	11	5	8	10	4	3	3	3	5	1	4	6	3	2	1	2	2	3	2	2	
6	26	20	18	14	5	5	3	9	2	3	2	1	8	3	4	0	2	2	3	0	4	0	0	4	4	2	
7	11	10	12	7	6	5	0	7	1	5	0	2	2	1	2	1	3	6	6	0	1	4	0	2	3	2	
8	15	5	9	2	8	3	11	6	9	2	5	2	3	8	2	5	1	1	2	2	3	0	1	0	1	0	
9	23	9	7	9	10	10	7	5	5	1	5	2	5	8	4	3	5	2	2	1	3	0	0	3	1	1	
10	34	15	16	11	4	6	10	14	7	9	0	3	1	1	4	2	1	2	2	0	3	2	2	1	5	4	
11	7	4	7	7	9	3	5	1	6	6	3	2	5	5	1	2	0	6	0	2	0	2	1	0	1	4	
12	25	9	17	8	3	7	2	3	2	10	4	1	1	7	2	2	0	1	3	1	2	0	3	1	3	4	
13	11	9	4	5	5	13	2	5	2	4	4	2	2	1	3	3	1	1	2	1	1	2	0	2	0	3	
14	12	18	4	6	12	9	4	7	3	5	5	2	3	1	5	7	0	3	4	2	0	0	2	3	7	0	
15	15	6	7	5	5	3	10	3	8	5	8	4	4	0	0	0	1	1	3	4	5	1	3	2	0	0	
16	19	11	12	9	16	5	1	5	3	7	0	6	11	0	1	4	1	7	7	2	3	6	4	3	1	1	
17	14	9	9	11	8	13	7	6	0	8	10	0	2	5	4	1	4	3	2	0	2	0	1	1	2	0	
18	27	9	17	8	9	1	12	12	8	6	1	3	3	0	0	0	4	1	1	1	2	1	1	1	1	0	
19	23	3	7	4	4	9	9	7	10	3	4	1	1	0	3	1	3	0	1	0	1	0	0	3	2	2	
20	18	9	12	10	11	11	4	11	6	5	4	6	6	2	3	3	8	3	5	0	3	3	1	5	4	0	
21	11	22	9	7	3	2	1	4	6	4	4	2	7	1	3	2	0	1	6	2	2	8	0	4	0	4	
22	29	15	18	6	8	11	5	5	7	8	12	7	1	3	5	3	7	3	0	5	2	0	1	4	1	0	

Document frequency

- A word's **Document Frequency** is the fraction of documents in which that word appears
  - We can characterize a word's rarity by its DF (lower = rarer)
  - Usually, we use **inverse document frequency** (IDF = 1/DF) for rarity

Kangaroos can jump. Kangaroos live in Australia.

Australia is in the southern hemisphere.

One of the longest flights is from London to Sydney, Australia.

Term	DF
Kangaroos	1/3
Australia	3/3
In	2/3

Problems with Inverse Document Frequency

- Suppose we return documents that contain terms from the query with high IDF
  - e.g., if my **query** is "*Rumpelstiltskin*", odds are very few *documents* contain that word (*high IDF*), so just return those documents

- Problem: how do I scale IDFs among search terms?
- Consider the query: "Rumpelstiltskin book"
  - What if
    "Rumpelstiltskin" appears **10** out of 29 million documents and
    "book" appears in **1 million** out of 29 million?
  - **Is the term "book" 100,000 times less useful than "Rumpelstiltskin"?**
- **Solution**: apply log. $IDF = \log(\frac{N}{n_k})$

  (n_k is # documents containing k)
  (N is total number of documents)

Term Frequency - Inverse Document Frequency

- Idea: Combine TF and IDF for best of both worlds:
  - TF: pull documents that contain lots of instances of query tokens
  - IDF: pull documents according to rarity of given query tokens
  - “tf-idf”

- $\boldsymbol{w_{ik} = tf_{ik} \times \log \frac{N}{n_k}}$
  - T_k = term k in document i (D_i)
  - tf_{ik} = term frequency of T_k in D_i
  - N = number of documents in corpus
  - n_k= number of documents containing T_k

tf-idf applies to each term in each document

Documents
(i) 0 and 1

Kangaroos can jump. i=0

Kangaroos live in Australia. i=1

Words:

[kangaroos can jump live in australia]
k=0 k=1 k=2 k=3 k=4 k=5

tf-idf vectors

$$D_0 = [\, w_{00}, w_{01}, w_{02}, w_{03}, w_{04}, w_{05} \,]$$
$$D_1 = [\, w_{10}, w_{11}, w_{12}, w_{13}, w_{14}, w_{15} \,]$$

Computing td-idf over a corpus

- IDF is independent of a specific document, and applies to each word
  - When processing a corpus, you can compute IDF's for each word in vocabulary
  - Then, just multiply by tf when considering each document

    - IDF basically becomes a coefficient for each word T_k in the vocabulary

	T0	T1	…	Tk
D0	$tf_{00}IDF_0$	$tf_{01}IDF_1$		$tf_{0k}IDF_k$
D1	$tf_{10}IDF_0$			
…				
Dn	$tf_{n0}IDF_0$			$tf_{nk}IDF_k$
	IDF0	IDF1	IDF2	IDF3

tf-idf normalization

- If you have a longer document, **tf** terms are naturally higher than in shorter documents (why?)

- Normalize term weights
  - Longer documents not given more weight
  - Normalize to sum-of-squares

$$w_{ik} = \frac{tf_{ik} \log(N/n_k)}{\sqrt{\sum_{k=1}^{t} (tf_{ik})^2 [\log(N/n_k)]^2}}$$

- Some references use non-normalized tf-idf
  - $w_{ik} = tf_{ik} \log(N/n_k)$

Vector space similarity

- Similarity of two docs is:

$$Sim(Di,Dj) = \sum_{k=1}^{t} w_{ik} * w_{jk}$$

Normalized ahead of time, when computing term weights.

$$\text{sim}(d_j, q) = \frac{\mathbf{d_j} \cdot \mathbf{q}}{\|\mathbf{d_j}\| \, \|\mathbf{q}\|} = \frac{\sum_{i=1}^{N} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{N} w_{i,j}^2} \sqrt{\sum_{i=1}^{N} w_{i,q}^2}}$$

Not normalized ahead of time

Thought questions (compared to binary retrieval)

- What is one reason why term frequency gives us more information than only a 1 or 0?
- What is one reason why inverse document frequency gives more information than a 1 or 0?
- If you knew that a search engine used tf-idf, what is a strategy that could help your site become a top result?