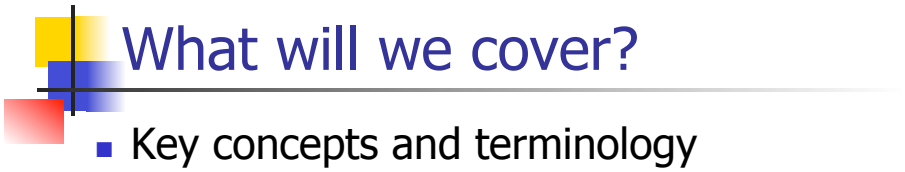
Indexing and Query Processing



What will we cover?

- Key concepts and terminology
- Inverted index structures
  - Organization, creation, maintenance
  - Compression
  - Distribution
- Answering queries with inverted indexes
  - Cosine, phrase (not probabilistic)
  - Optimizations
  - Fielded queries
- Other approaches (?): signatures, suffix trees

Some Scenarios

- Web search
  - Lots of storage and CPU
  - Frequent new documents
  - Updates of existing documents
  - Lots & lots of queries
  - Absolutely current?

Scenarios (2)

- Company repository
  - Servers might be provisioned for collection size
  - Documents might be fairly static
  - Moderate to low query rate
- Desktop search
  - What % of your disk is documents?
  - Low query rates, but don't want to hog disk, CPU
  - Would like to be very up to date: last save

Why isn't this a database problem?

- Indexing selected columns vs. indexing nearly everything

  In both cases, want to be able to read an "entry" rapidly
- Identifying the "parts" of the "record" to index
  - words, stems, phrases
  - column values in a particular row

DB vs. IR indexing (cont.)

- Kinds of queries supported
  - `A=v  A<v`
  - `{kw1,kw2,kw3}  "kw1 kw2 kw3"`
- What's involved in producing a prefix of the answer: ranking vs. sorting
- Update patterns
  - Change a row: two index entries
  - Change a document: many index entries

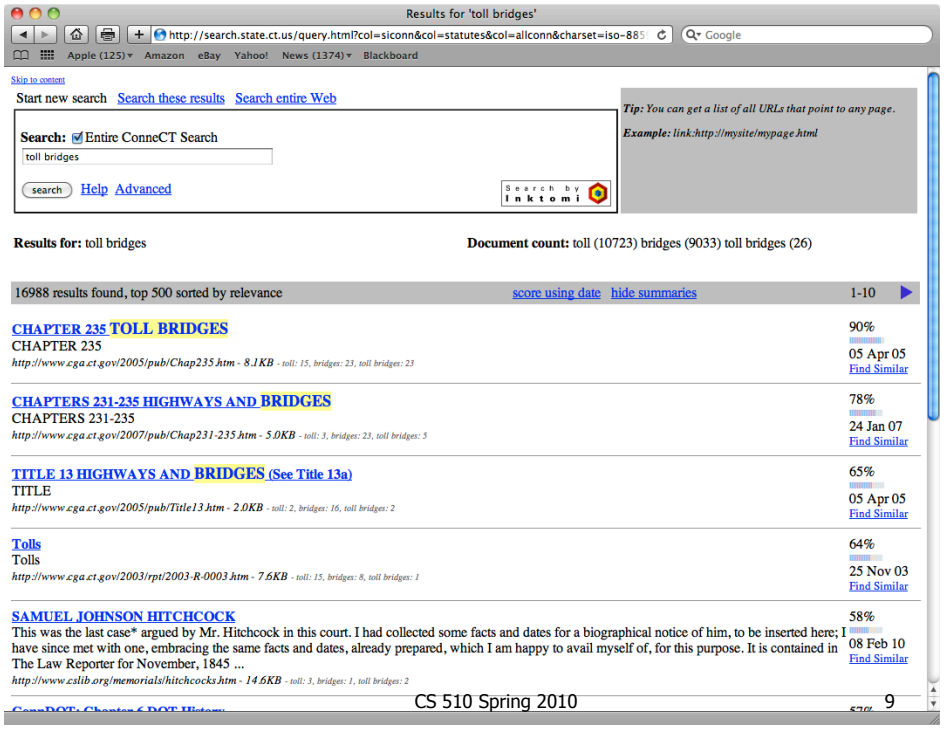
Weighted Matching

- Rather than being 0 or 1, match is a numeric score
- Can be used for defining the result set (via a threshold)
- Can be used for ranking results obtained by other methods
  - Google is Boolean retrieval + ranking (on relevance and quality

Relevance Scores

- Relevance of document to the query may be the main means of ranking for a "closed" collection
  - New articles
  - Tech support notes
  - Reviews
- Some search interfaces give you the relevance scores

  http://search.state.ct.us/



Examples of Numeric Scores

1. How many different terms from the query are in the document
2. How many occurrences of query terms are in the document
3. +2 for query term in title, +1 for body (different *zones* in the document)

Query: exploring moon

Exploring the Outer Planets

While landing on a gas giant is unlikely, it may be possible on a planetary moon.

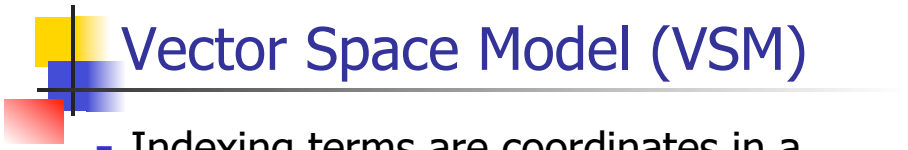
Exploring the Moon

The moon was the first extra-terrestrial body to have an earth-launched lander.

How to support this kind of scoring?
Qualified terms:

```
moon.title
moon.body
exploring.title
```

Have different inverted lists for each.



Vector Space Model (VSM)

- Indexing terms are coordinates in a high-dimensional information space
- Documents and queries represented as *n*-dimensional vectors: (w_1, w_2, …, w_n)
  - *n* = total number of terms
  - w_i is weight of the *i*-th term
    - derived from a term-weighting algorithm
    - often uses some form of word-frequency calculation
  - vector is conceptual, not stored directly

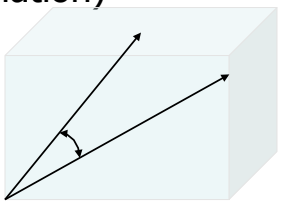
Example Vectors

Weight = term count

	(black crows red rooster)
D1. The red rooster crows.	(0 1 1 0)
D2. The crows are black.	(1 1 0 0)
D3. The red rooster and the black rooster.	(1 0 1 2)

VSM: Score is Vector Similarity

- Allows assignment of non-binary weights to index terms
- Allows computation of similarity between documents and queries
  - Usually calculated as the cosine of the angle between two vectors $\vec{d}_j$ and $\vec{q}$ (or a variation on that calculation)



Cosine Measure

$w_{d,t}$ = weight of term t in document d

$w_{q,t}$ = weight of term t in query q

$W_d = (\Sigma_t (w_{d,t})^2)^{1/2}$

$W_q = (\Sigma_t (w_{q,t})^2)^{1/2}$

Similarity of q and d

$S_{q,d} = (\Sigma_t w_{d,t} * w_{q,t}) / W_d * W_q$

Cosine Example

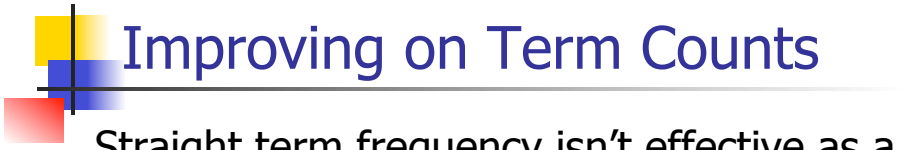
- Similarity of D1 and D2

$$\frac{(0 \bullet 1 + 1 \bullet 1 + 1 \bullet 0 + 1 \bullet 0)}{3^{1/2} \bullet 2^{1/2}}$$

- Similarity of D1 and D3

$$\frac{(0 \bullet 1 + 1 \bullet 0 + 1 \bullet 1 + 1 \bullet 2)}{3^{1/2} \bullet 6^{1/2}}$$

- Can treat query as a "mini-document"



Improving on Term Counts

Straight term frequency isn't effective as a term weight

On web:

computer	forensics	course
706M	7.5M	641M

Document with `(3 1 6)` would score the same as a document with `(2 5 3)`

"Boost" terms based on rarity.

TF*IDF

- Frequency of term t in document d
  - *Term Frequency* (TF) $f_{d,t}$
- Frequency of term in the entire collection
  - *Document Frequency* (DF) is # documents with term
  - Low document frequency = good discriminator
- TF*IDF used as term weight for a document
  - IDF is inverse of document frequency (usually calculated as log (N/f_t) where N = #docs in collection, f_t = #docs with term t)

Example TF*IDF Weights

Suppose 5 billion documents

```
computer: log(5B/706M) =
forensics: log(5B/7.5M) =
course: log(5B/641M) =

(3 1 6) →
(2 5 3) →
```


Variants of TF*IDF

- Many variations on term-weighting have been tried, e.g.
  - Logarithmic term frequencies
  - Term frequencies normalized to max term frequency and scaled to fall in range 0.5 – 1
  - Most useful terms seem to be ones with medium frequency
- Query terms may be weighted or binary
  - Weighted may be useful for long queries, such as documents or long descriptions of information needs

Another Weight Model

Used by Zobel & Moffat (why the fudge factors?)

Main thing here is what the inputs are

(Base of logarithm isn't critical)

$w_{q,t} = \ln(1 + N/f_t)$

$w_{d,t} = 1 + \ln(f_{d,t})$

Calculating Cosine Measure (s. 15)

What can we calculate in advance of seeing the query?

- $f_{d,t}$, hence $w_{d,t}$, hence W_d
- Note that $w_{q,t}$ only depends on t

  So can compute query term weights in advance for all terms

  (then select the ones we need for a given query)

What changes on update?

- $f_{d,t}$, $w_{d,t}$, W_d for new or updated document d
- What about $w_{q,t}$?

  Recall $w_{q,t} = \ln(1 + N/f_t)$
  - Consider adding one new document
  - New ratio is $(N+1)/f_t$ or $(N+1)/(f_t+1)$
  - 1314/65

Can omit W_q factor sometimes

If we are just comparing $S_{q,d}$ values, then we can ignore W_q, since it is the same for every d.

(but perhaps not if we are combining $S_{q,d}$ with other measures)

Naïve calculation

```
For each d in D
  S ← 0
  For each t in q
     S ← S + (w_{d,t} * w_{q,t})
  S_{d,q} ← S/W_d*W_q
```

Note that not every query term needs to be in the document

Could modify to set $S_{d,q} \leftarrow 0$ if any query term is missing

Better calculation scheme

```
For each t in q
  For each d containing t
     …
```

Will need to keep a partial result (accumulator) for multiple documents

Phrase queries

- Need to tell "`eggs and ham`" from "`ham and eggs`"
- Options
  - Search for terms, then post-process by checking actual documents
  - Remember positions for each term in each document
  - Index phrases or partial phrases (for example, all two-term combinations)

What Should Indexing Speed up?

- Boolean queries

  Need to know which documents contain particular words; combine lists of documents
- Cosine (or other) similarity

  Finding documents in order of similarity to a query
- Phrase (or proximity) queries

  Determining adjacent (or nearby) word occurrences

Sample Documents

d1

```
Would you like them
in a house?
Would you like them
with a mouse?
```

d2

```
Say!
In the dark?
Here in the dark?
Would you, could you,
in the dark?
```

d3

```
I would not, could not in a tree.
Not in a car! You let me be.
```

d4

```
You may like them.
You will see.
You may like them
in a tree!
```

d5

```
You do not like them.
So you say.
Try them! Try them!
And you may.
Try them and you may, I say
```


Inverted List Structure

Term dictionary, with doc frequencies

t	a	be	car	could	dark	here	house	I	in	let
f_t	3	1	1	2	1	1	1	2	4	1

t	like	may	me	mouse	not	say	see	so	the
f_t	3	2	1	1	2	2	1	1	1

t	them	tree	try	will	with	would	you
f_t	3	2	1	1	1	3	5

Inverted List Structure (2)

Term frequencies, in each document

t: <d, $f_{d,t}$>, …

```
a: <d1,2>, <d3,2>, <d4,1>
be: <d3,1>
…
in: <d1,1>, <d2,3>, <d3,2>, <d4,1>
…
you: <d1,2>, <d2,2>, <d3,1>,
  <d4,3>, <d5,4>
```


Inverted list structure (3)

Document info: doc id, W_d

For example, d3

```
I would not, could not in a tree.
Not in a car! You let me be.

I would not could in a tree car you
1   1    3    1   2  2  1    1   1

let me be
 1  1  1
```

Do we need the document contents?

Example similarity calculation

q = {not, in, tree}

$w_{q,t}$ = 1.25 0.81 1.25

$W_q = (1.25^2 + 0.81^2 + 1.25^2)^{1/2}$
≈ 1.94

$$S_{q,d3} = \frac{(1.25*2.099 + 0.81*1.693 + 1.25*1)}{4.37*1.94}$$

$\approx (2.62 + 1.37 + 1.25)/2.25$

$\approx$ **2.33**

Indexing word positions

Might want to list all word positions in an index

for phrase or proximity queries

For term t, let $f_{d,t} = k$

<d; k; p_1, p_2,… p_k>

in: <d1; 1; 5>, <d2; 3; 2,6,13>,
<d3; 2; 6,10>, <d4; 1; 12>

Might associate info with each occurrence:
In title? Capitalized? Font size?

Problem with word positions

Slows down search when all you need is word frequency

Could have two indexes, with and without word positions

Phrase index

Count of occurrences of phrase

- There are a lot of phrases
- How to choose which ones?
- One possibility: index just two-word phrases where the first word is common.

`red: "red flag" "red light" "red hot" "red square"`

Then `"red hot coals"` is processed as `"red hot" + coals`, plus position check