LBSC 796/INFM 718R: Week 5

Indexing

Jimmy Lin
College of Information Studies
University of Maryland

Monday, February 27, 2006

Muddy Points

- Probability of a single word vs. probability of a sequence
- Uniform priors

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

- Smoothing
- Difference between vector space and language models

Why smoothing?

I think language modeling is very very cool

↓

Model M

P(w)	w
0.125	cool
0.125	I
0.125	is
0.125	language
0.125	modeling
0.125	think
0. 250	very

What are the probabilities?

P("language modeling think")
P("cool language modeling")
P("cool cool modeling")
P("language modeling retrieval")

Some Questions for Today

- How long will it take to do a search?
- How do I measure speed?
- How big a computer will I need?
  - How much RAM?
  - How much disk space?
- What if more documents arrive?
- How do I index the Web?

Assumptions

- Optimize for retrieval
  - Searching should be fast: users are impatient
- Indexing speed isn't critical
  - Document collection is relatively stable
- When do these assumptions start to break down?

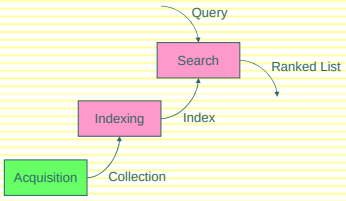
Algorithms

- What is an algorithm?

a precise rule (or set of rules) specifying how to solve some problem

- Think "recipe" for problem solving
- Today's focus: algorithms for information retrieval
  - What data structures are involved?
  - How much space do they take up?
  - How fast are they?

Indexing



The Starting Point

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8
aid	0	0	0	1	0	0	0	1
all	0	1	0	1	0	1	0	0
back	1	0	1	0	0	0	1	0
brown	1	0	1	0	1	0	1	0
come	0	1	0	1	0	1	0	1
dog	0	0	1	0	1	0	0	0
fox	0	0	1	0	1	0	1	0
good	0	1	0	1	0	1	0	1
jump	0	0	1	0	0	0	0	0
lazy	1	0	1	0	1	0	1	0
men	0	1	0	1	0	0	0	1
now	0	1	0	0	0	1	0	1
over	1	0	1	0	1	0	1	1
party	0	0	0	0	0	1	0	1
quick	1	0	1	0	0	0	0	0
their	1	0	0	0	1	0	1	0
time	0	1	0	1	0	1	0	0

The term-document matrix has "bag of words" information about the collection

Small yet Fast

- Can we make this data structure smaller, keeping in mind the need for fast retrieval?
- Observations:
  - The nature of the search problem requires us to quickly find which documents contain a term
  - The term-document matrix is very sparse
  - Some terms are more useful than others

Why is size important?

- RAM
  - Typical size: 1 GB
  - Typical access speed: 50 ns
- Hard drive:
  - Typical size: 80 GB (my laptop)
  - Typical access speed: 10 ms
- Hard drive is 200,000x slower than RAM!
- Why is this an important issue?

Postings

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5	Doc 6	Doc 7	Doc 8	Postings
aid	0	0	0	1	0	0	0	1	4, 8
all	0	1	0	1	0	1	0	0	2, 4, 6
back	1	0	1	0	0	0	1	0	1, 3, 7
brown	1	0	1	0	1	0	1	0	1, 3, 5, 7
come	0	1	0	1	0	1	0	1	2, 4, 6, 8
dog	0	0	1	0	1	0	0	0	3, 5
fox	0	0	1	0	1	0	1	0	3, 5, 7
good	0	1	0	1	0	1	0	1	2, 4, 6, 8
jump	0	0	1	0	0	0	0	0	3
lazy	1	0	1	0	1	0	1	0	1, 3, 5, 7
men	0	1	0	1	0	0	0	1	2, 4, 8
now	0	1	0	0	0	1	0	1	2, 6, 8
over	1	0	1	0	1	0	1	1	1, 3, 5, 7, 8
party	0	0	0	0	0	1	0	1	6, 8
quick	1	0	1	0	0	0	0	0	1, 3
their	1	0	0	0	1	0	1	0	1, 5, 7
time	0	1	0	1	0	1	0	0	2, 4, 6

An Inverted Index

Term	Postings
aid	4, 8
all	2, 4, 6
back	1, 3, 7
brown	1, 3, 5, 7
come	2, 4, 6, 8
dog	3, 5
fox	3, 5, 7
good	2, 4, 6, 8
jump	3
lazy	1, 3, 5, 7
men	2, 4, 8
now	2, 6, 8
over	1, 3, 5, 7, 8
party	6, 8
quick	1, 3
their	1, 5, 7
time	2, 4, 6

What goes in the postings?

- Boolean retrieval
  - Just the document number
- Ranked Retrieval
  - Document number and term weight (*tf.idf*, ...)
- Proximity operators
  - Word offsets for each occurrence of the term

The Retrieval Process

- During retrieval:
  - Find the relevant postings based on query terms
  - Manipulate the postings based on the query
  - Return appropriate documents
- Example with Boolean queries

Further Compressing the Index

- Postings can still be quite large
  - Especially if you have a large collection

  e.g., 1 million documents → 20-bit document numbers
- Idea: encode differences instead of document numbers

  37, 42, 43, 48, 97, 98, 243 →
  37, 5, 1, 5, 49, 1, 145
- Many other ways to compress the postings
- What about dropping unimportant terms from the index?
  - How much space does stopword removal save?

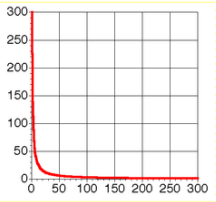
Zipf's Law

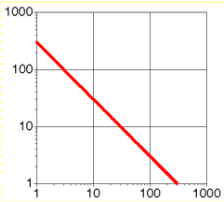
- George Kingsley Zipf (1902-1950) observed that for many frequency distributions, the *n*th most frequent event is related to its frequency in the following manner:

$$f \cdot r = c \quad \text{or} \quad f = \frac{c}{r}$$

f = frequency
r = rank
c = constant

Zipfian Distribution





Zipfian Distribution

- Key points:
  - A few elements occur very frequently
  - A medium number of elements have medium frequency
  - Many elements occur very infrequently
- What's the big deal?
  - English words obey Zipf's Law

Word Frequency in English

Frequency of 50 most common words in English (sample of 19 million words)

the	1130021	from	96900	or	54958
of	547311	he	94585	about	53713
to	516635	million	93515	market	52110
a	464736	year	90104	they	51359
in	390819	its	86774	this	50933
and	387703	be	85588	would	50828
that	204351	was	83398	you	49281
for	199340	company	83070	which	48273
is	152483	an	76974	bank	47940
said	148302	has	74405	stock	47401
it	134323	are	74097	trade	47310
on	121173	have	73132	his	47116
by	118863	but	71887	more	46244
as	109135	will	71494	who	42142
at	101779	say	66807	one	41635
mr	101679	new	64456	their	40910
with	101210	share	63925		

Does it fit Zipf's Law?

The following shows *rf*1000/n*

r is the rank of word *w* in the sample
f is the frequency of word *w* in the sample
n is the total number of word occurrences in the sample

the	59	from	92	or	101
of	58	he	95	about	102
to	82	million	98	market	101
a	98	year	100	they	103
in	103	its	100	this	105
and	122	be	104	would	107
that	75	was	105	you	106
for	84	company	109	which	107
is	72	an	105	bank	109
said	78	has	106	stock	110
it	78	are	109	trade	112
on	77	have	112	his	114
by	81	but	114	more	114
as	80	will	117	who	106
at	80	say	113	one	107
mr	86	new	112	their	108
with	91	share	114		

IR and Zipf's Law

- What's the relevance of Zipf's Law to information retrieval?

Other Zipfian Distributions

- Library book checkout patterns
- Website popularity
- Incoming Web page requests
- Outgoing Web page requests
- Document size on Web

How big is the inverted index?

- Postings take up most of the space
- Heap's Law tells us about vocabulary size

$$V = Kn^{\beta}$$

$K \approx 20, \beta \approx 0.6$

V is vocabulary size
n is corpus size (number of documents)
K and β are constants

- When adding new documents, the system is likely to have seen terms already
- But the postings keep growing

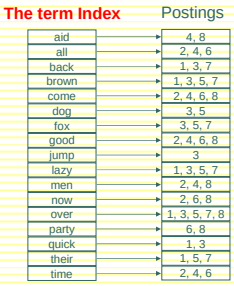
How big are the postings?

- Very compact for Boolean retrieval
  - About 10% of the size of the documents
- Not much larger for ranked retrieval
  - Perhaps 20% of collection size
- Enormous for proximity operators
  - Sometimes larger than the document collection

So far…

- We have discussed:
  - The data structure used for indexing and retrieval
  - Methods for compressing the inverted index
- We have not discussed:
  - How the inverted index is actually built
  - How a system looks up the query terms
  - How fast all these operations are

Decoupling the Inverted Index



Terms in the Collection

- Let's focus on the term index
- The postings are relatively simple
  - During indexing: once you find the correct postings, add information from current document
  - During retrieval: once you find the correct postings, manipulate based on query operator
- Questions
  - How do you find the correct posting quickly?
  - What happens when you come across a new term?

How do you measure speed?

- We want to know if one algorithm is faster than another
- What do we measure?
  - Number of clock cycles?
  - Number of instructions?
  - Clock time?
- What's wrong with these measures?

Big-O Notation

- A mathematical method for quantifying the running time of an algorithm
  - Abstracts away from actual implementation
  - Abstracts away from specific machine architecture
- Useful for theoretical considerations of speed
- Best illustrate with examples…

Linear Dictionary Lookup

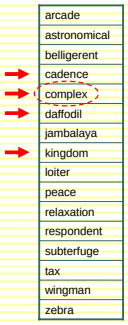
Suppose we want to find the word "complex"

Found it!

- How long does this take, in the worst case?
- Running time is proportional to number of entries in the dictionary
- This algorithm is O(*n*) = linear time algorithm

With a Sorted Dictionary

Let's try again, except this time with a sorted dictionary: find "complex"



- How long does this take, in the worst case?

Found it!

Binary Search: Analysis

- Algorithm:
  - Look in the middle entry of a region, call this *x*
  - If the entry you're looking for comes before *x*, then look in first half, otherwise look in second half
  - Repeat until you find what you're looking for
- Analysis:
  - Each time we look up an entry, we cut down the number to consider by a half
  - How many times can you divide a number by 2?
- This algorithm is O(lg *n*)

Which is faster?

- Two algorithms:
  - O(*n*): Sequentially search through every entry
  - O(lg *n*): Binary search
- Big-O notation
  - Tells us the asymptotic worst case running time of an algorithm
  - Allows us to compare the speed of different algorithms

What was glossed over?

- How much effort did it take to sort the dictionary in the first place?
- General strategy: reduce search time by doing work ahead of time

Selection Sort

- One of the simplest sorting algorithms:
  - Find smallest number, move it to the first position
  - Find 2nd smallest number, move it to the 2nd position
  - Find 3rd smallest number, move it to the 3rd position
  - …
- This algorithm is $O(n^2)$
  = quadratic (polynomial) algorithm
- Faster sorting algorithm: Quick sort = O(*n* log *n*)

Satisfiability

- Given a boolean expression, is there an assignment of TRUE and FALSE that makes the entire expression true?
- 3-SAT: expression grouped into clauses of three variables

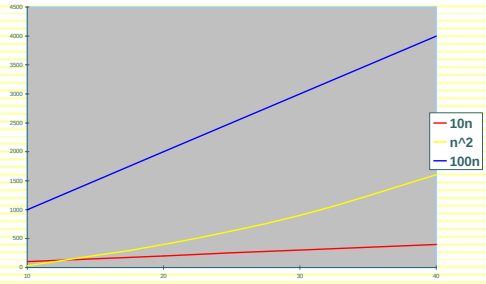
  $E = (x_1 \text{ or } \sim x_2 \text{ or } \sim x_3) \text{ and } (x_1 \text{ or } x_2 \text{ or } x_4)$

  Solution: x_1 = TRUE, x_2 = TRUE, x_3 = TRUE, x_4 = TRUE
- The simplest algorithm is to try all possible combinations
- This algorithm is $O(2^n)$
  = exponential algorithm

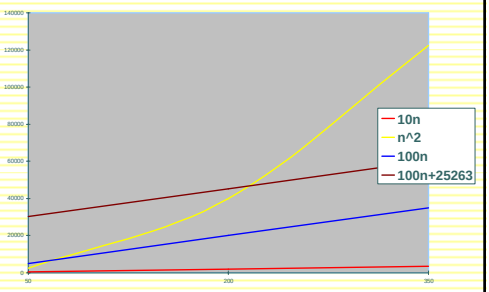
Do the details matter?

- Arrange the following from fastest to slowest:
  3. O(n log n)
  4. O(n^2)
  1. O(log n)
  2. O(n)
  5. O(2^n)
- What about?
  - 10n, 100n, 1000n
  - n^2, $2n^2$, $2n^2 + 100$

"Asymptotic" Complexity



"Asymptotic" Complexity



Irrelevant Details

- Big-O notation is concerned with
  - Asymptotic complexity
  - Worst-case running time
- Therefore
  - 10n, 100n, 1000n → O(n)
  - n^2, $2n^2$, $2n^2 + 100$ → O(n^2)
  - lg n, $\log_8 n$, log n, ln n → O(log n)

How big?

- Constant, i.e., O(1)
  - n doesn't matter
- Sublinear, e.g., O(log n)
  - n = 65536 → log n = 16
- Linear, i.e. O(n)
  - n = 65536 → n = 65536
- Polynomial, e.g., O(n^2)
  - n = 65536 → n^2 = 4,294,967,296
- Exponential, e.g., O(2^n)
  - n = 65536 → beyond astronomical

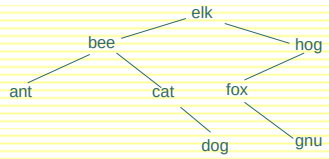
Linear Term Index

- Keep terms in a list, use binary search to look up
- Advantages
  - Quick searching
  - Economic use of storage
- Disadvantages
  - Long seek times
- Are there alternatives for storing the term index?

Binary Tree

- A tree structure where
  - Each node has at most two children
  - The left child comes before the parent
  - The right child comes after the parent

Input: elk, hog, bee, fox, cat, gnu, ant, dog



Binary Tree Operations

- Searching (lookup *x*):
  - Start at the root, compare current node to *x*
  - Follow left child if *x* comes before current node
  - Follow right child if *x* comes after current node
  - Repeat until *x* is found
- Indexing (adding term *x*) :
  - Search for position where you would expect to see *x*
  - Insert *x* there

Binary Trees

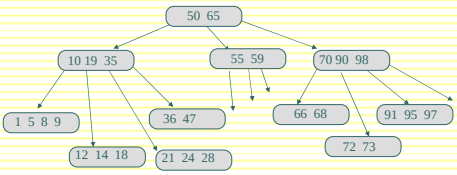
- Advantages
  - Can be searched quickly
  - Easy to add an extra term
  - Economical use of storage (although less so than linear term list)
- Disadvantages
  - Trees tend to become unbalanced

Unbalanced Trees

- The shape of a binary trees depends on insertion order of terms
- What's the worst case scenario?
- The need for balanced trees

B-Trees

- A balanced, multiway search tree



Indexing and Searching

- Indexing
  - Access the term index
  - Find the correct postings and add document accordingly
  - Relatively slow process (hours, even days)
- Searching
  - Lookup the query terms in term index
  - Read the postings
  - Manipulate the postings based on query
  - Very fast (seconds)

Indexing the Web

- We've assumed that the document collection is readily available
- How do we index the Web?

Web Crawlers

- How do Web search engines know what to index?
- Main idea:
  - Start with known sites
  - Record information for these sites
  - Follow the links from each site
  - Record information found at new sites
  - Repeat

Web Crawling Algorithm

- Put a set of known sites on a queue
- Repeat the until the queue is empty:
  - Take the first page off of the queue
  - Check to see if this page has been processed
  - If this page has not yet been processed:
    - Add this page to the index
    - Add each link on the current page to the queue
    - Record that this page has been processed

Web Crawling Issues

- Size
- Freshness
- Different languages
- The static collection assumption
- Adversaries
- Dynamic content
- "Dark matter"
- Technical problems
  - Missing/broken links
  - Invalid HTML

Summary

- Information retrieval systems are only useful if they're responsive
  - Users are impatient
  - Interaction is critical to the information seeking process
- Building fast systems that operate on large collections is hard!
  - Tradeoffs in different data structures
  - Tradeoffs in searching/indexing cost

One Minute Paper

- What was the muddiest point in today's class?