Lecture 3 Overview

- Recap Algorithms for Inverted File Construction
- Chapter 5: Index Compression
  - Dictionaries (IIR 5.2)
  - Inverted files (IIR 5.3)
- Chapter 6: Term Weighting
  - Zipf distribution & IDF, tf-idf
- Chapter 7: Vector Space Models
  - cosine comparisons

Lexicons

- We want to store certain information for each word
  - The number of documents containing the word
  - An offset into a postings file
  - Often, we also want to know the total number of occurrences of the word
- We want minimal storage requirements and fast operations (insert, find)
  - While scanning documents
  - When processing user queries

How much memory do we need?

- Thought Experiment

  (following Managing Gigabytes, Chap. 4)
  - 1M word lexicon; 12 bytes for each value
  - 4-byte ints for #docs, #occurs, #fileoffset
    - 12MB in 'values'
    - What about the keys?
- Approach #1: Fixed size strings
    - Assume none is longer than 20 characters (or truncate)
    - 20 * 1M = 20 MB

Fixed-size strings

	#docs	#occ	offset
retraining	723	926	
retreat	2944	3337	
retrieval	385	680	
retrieve	626	685	
retrieved	417	487	
retrieving	271	317	
retroactive	853	1075	
retrofit	243	531	

- TREC-8 Collection
  - 2 GB text
  - ~500k docs
  - ~500k unique terms
- Space for lexicon with 1M terms
  - 12 + 20 = 32 MB
- To find a query term
  - Load the array in memory
  - Use binary search
- Problem?

Terminated strings

retrainingretreatretrievalretrieveretrievedretrievingretr
oactiveretrofit

term ptr	#docs	#occ	offset
0	723	926	
10	2944	3337	
17	385	680	
26	626	685	
34	417	487	
…	271	317	
…	853	1075	
…	243	531	

- **Space for lexicon**
  - Avg word len = ~8 chars
  - 12 + (9 + 4) = 25 MB (symbol)
  - 12 + (8+4) = 24 MB (subtract)
- **To find a query term**
  - Load the array in memory
  - Use binary search
  - Follow extra pointer
- **Can we do better?**

Terminated strings + Blocking

retraining$retreat$retrieval$retrieve$retrieved
$retrieving$retroactive$retrofit

term ptr	
0	0th
34	4th
…	
…	
…	
…	
…	
…	

#docs	#occ	offset
723	926	
2944	3337	
385	680	
626	685	
417	487	
271	317	
853	1075	
243	531	

- **Same as before, but split ‘arrays’ up into string pointers and info**
- **Space for lexicon**
  - With 4:1 blocking
  - 12 + (9+1) = 22 MB
- **To find a query term**
  - Load the array in memory
  - Use binary search
    - (log V) - 2
  - Extra pointer + linear search of small block
- **Can we do better still?**

Term'd strings; blocking; front-coding

retrainingretreatretrievalretrieveretrievedretrievingretroactiveretro fit

(retra) 5,5,ining 4,3,eat 4,5,ieval 7,1,e 8,1,d …, 0,1,s 1,2,ad

3-in-4 front-coding so binary search can be used

term ptr		#docs	#occ	offset
0	0th	723	926	
34	4th	2944	3337	
…		385	680	
…		626	685	
…		417	487	
…		271	317	
…		853	1075	
…		243	531	

- Same as before, but realize lots of repetition in 'key space'
- Space savings
  - For English lexicons, 40%
  - 12 + (5+1) = 18 MB
- To find a query term
  - Load the array in memory
  - Use binary search
  - Extra pointer + linear search of small block
- Can we do better?

Minimal-Perfect Hash Functions

- Hashtables maps k keys onto integers 0 through m
  - For m>=k, as m approaches k, collisions increase
  - Birthday paradox
- Perfect hash function maps k keys onto k integers without collisions
  - *Minimal* perfect hashing maps k keys onto integers 0 to k-1
- Can such a function be found
  - Yes, if set of keys is fixed, using fast, probabilistic algorithm
    - In fact, even functions that preserve the order of the input keys can be found! (a = 0, ... zoology = k-1)
    - What is the catch?

The hash *function* takes up space

- For an order preserving function
- At least, n log(n) bits are required for n keys
- For 1 M keys, that's 1.25 bytes / key
  - 5 MB in space vs. previous 6 MB
  - Starting to spend a lot of effort for small gains

Really Large Lexicons

- Chinese is a non-alphabetic language
  - O(10k) symbols
- N-gram indexing represents 'terms' as 3-character sequences
  - 10,000^3 = 10^12 terms (1,000 billion)
- Instead of in-memory, go to external storage
  - Slows down indexing significantly
  - But, caching can help
  - Same idea as databases, use B+-trees

Index Compression

- In general, an index requires less space than a text
  - Even for a comprehensive index
    - Rule of thumb: 10-30%
  - Why does it take up less space?
- Why reduce the size of an inverted file?
  - We can’t fit it all in memory like a lexicon
  - We are trading time to reduce space and are happy – what is gained?

Binary Warm-up

Compressing Integers

- Binary representation
  - Better than ASCII!
- Three other coding schemes
  - Unary code
  - Gamma code
  - Delta code

Encoding Numbers in Binary

- Representing numbers as character symbols is inefficient
  - '7856' in ASCII is: $0110111\ 0111000\ 0110101\ 0110110_2$ (28 bits)
  - 7856_{10} in binary is: 1111010110000_2 (13 bits)
- Example: small binary numbers

	$2^4 = 16$	$2^3=8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
16	1	0	0	0	0
31	1	1	1	1	1

$3 = (0 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$
$= 0 + 0 + 0 + 2 + 1$

$9 = (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$
$= 0 + 8 + 0 + 0 + 1$

What are we compressing?

- Say a term occurs in a list of documents:
  - 3, 17, 32, 100, 105, 1000, 2000
- We'd like to map these numbers to smaller values, gaps
    - 3,14,15,68,5,895,1000
  - No loss of information
    - Consider a frequent term, it occurs in almost every other document
    - …, 1, 2, 3, 2, 1, 1, 2, 3, 1, 2, 7, 2, 1, …
  - These smaller numbers are more compressible
    - Only a few can be really big

Unary Coding

- Only for integers greater than or equal to zero (MG: one)
- Represent x as:

  x one bits, followed by a single zero bit

  (MG: use x-1 'one bits')

  - unary(1) = 10
  - unary(2) = 110
  - unary(5) = 111110
  - unary(1000) = 1111111……..1111111110
    - 1000 1's followed by a 0
  - unary(1 million) = don't ask!
- Code is 'good' for <u>small</u> numbers!

Elias's Gamma Code (1975)

- **Represent x in two parts:**

  unary(floor(log(x))) followed by

  $x-2^{(floor(log\ x))}$ in binary

  floor(log(x)) is the number of binary 'suffix' bits; use base 2

  - **gamma(2) = 10 followed by 0 = 10 0**
  - **gamma(5) = unary(2) followed by binary($5-2^2$)**
    = unary(2) and binary(1)
    = 110 01
  - **gamma(1000) = unary(9) and binary(488) in 9 bits**
    - **Only takes 19 bits**
    - **gamma(1 million) only requires 39 bits**
- **Is this better than unary?**

Elias's Delta Code (1975)

- **(See exercise IIR 5.9)**
- **Just like previous method**
  - **Except, store the number of binary suffix bits using the Gamma code**
- **Represent x in two parts:**

  gamma(floor(log(x))) followed by

  $x-2^{(floor(log\ x))}$ in binary – using floor(log x) bits

  - **delta(2) = gamma(1)=0 followed by 0 = 0 0**
  - **delta(5) = gamma(2) followed by binary($5-2^2$)**
    = gamma(2) and binary(1)
    = 100 01
  - **delta(1000) = gamma(9) and binary(488) in 9 bits**
    - **Only takes 16 bits**
    - **delta(1 million) only requires 28 bits**

What difference does it make?

Method	Bible (KJV) (N=~30,000)	TREC Disks 1-3 (N=~720,000)
Unary	262	1918
Binary	15	20
Gamma	6.51	6.63
Delta	6.23	6.38
Local Interpolative	5.24	5.18

Bits per posting. Based on Managing Gigabytes 2nd, pg 129

What about the term frequency?

- **We just explained how to encode document ids**
  - **Using gaps**
- **But postings lists are usually pairs**
  - **(docid, freq), (docid, freq), (docid, freq), ...**
- **How should we encode term frequencies with each document?**

Docid Reassignment

- Normally docids are assigned in order that documents are scanned for indexing
  - For indexing, the ordering is arbitrary
- Some assignments of ids to documents are better than others
  - Average gap length can be reduced

D1: Dewey defeats Truman
D2: Berlin airlift prevents starvation
D3: Truman defeats Dewey
D4: Eisenhower leaves Berlin to Red Army

Berlin	2	4
Dewey	1	3
Truman	1	3

D1: Dewey defeats Truman
D2: Truman defeats Dewey
D3: Berlin airlift prevents starvation
D4: Eisenhower leaves Berlin to Red Army

Berlin	3	4
Dewey	1	2
Truman	1	2

d-gap Compression

- TSP approximation
  - Shieh et al., 'Inverted file compression through document identifier reassignment', Information Processing and Management, 39(1), 117-131, 2003
  - 15-20% reduction seems possible
- Greedy algorithm
  - Chris Buckley on TREC Terabyte corpus
- Why can't we just gzip the entire inverted file?

Dual Files

- Inverted files are term-referenced lists of docids and term-counts
  - A document-referenced list of termids and term-counts is a ‘dual file’
  - Like a column of the term-document matrix instead of a row
  - The same compression techinques may be applied
- Uses
  - for finding all terms that appear in a given document
  - to compare two documents directly

Dual Files cont’d

- Access to each ‘vector’ requires a different disk seek
  - Thus can not afford to do many
- Application
  - Finding expansion terms for a query (aka, blind relevance feedback)
  - Take the top 10 ranked documents and look for terms that might be useful to augment the original query with
    - Then do a second pass of retrieval

Beyond Boolean

- Chapter 6: Term Weighting
  - Zipf distribution & IDF, tf-idf
  - Gloss over IIR 6.1
- Chapter 7: Vector Space Models
  - Cosine comparison
  - Efficiently computing cosine

Documents & Queries

- Each of the queries and documents might be considered as:
  - A set of terms (Boolean approach)
    - “words”, morphologically normalized words, character n-grams, etc.
  - A representation not based on ‘sets’

Bag of Words Representation

- Original Text
- When in the Course of human Events, it becomes necessary for one People to dissolve the Political Bands which have connected them with another, and to assume among the Powers of the Earth, the separate and equal Station to which the Laws of Nature and of Nature's God entitle them, a decent Respect to the Opinions of Mankind requires that they should declare the causes which impel them to the Separation.

- Set of terms
- a,among,and,another,assume, Bands,becomes,causes,connected,Course,decent,declare,dissolve,Earth,entitle,equal,Events,for,God,have,human,impel,in,it,Laws,Mankind,Nature,Natures's,necessary,of,one,Opinions,People,Political,Powers,requires,Respect,separate,Separation,should,Station,that,the,them,they,to,When,which,with

- Bag of terms
- a(1),among(1),and(3), ...

Common Term Assumption

- Only documents that share features with the query are relevant
  - We speak generally of *indexing terms;* for now, assume ordinary words are used.
    - Many, many variants exists
      - Terms can be weighted differently
      - Terms need not be simple words (e.g., two word phrases)

- Or, if a document and the query share no words in common, the document is not relevant
  - And should be given a low score
  - (or not even scored)

Introduction

- Goal of IR is to retrieve all and only the "relevant" documents in a collection for a particular user with a particular need for information
  - Relevance is a central concept in IR theory
  - What is it?
- We can't afford to evaluate all documents
  - So what assumptions could we make to try and find good ones?

On Relevance

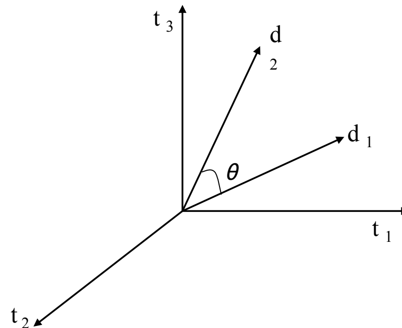
- Relevance is hard to nail down
    - 5 versions of the same fishing boat sinking
    - The loss of 5 separate vessels
  - Answer depends on your task
- Humans only agree 70-80% of the time
  - In fact, individuals changes their minds over time about document relevance
- Who you are matters
  - The FDA is about to release a new heart medication
  - Are you a patient, a primary-care physician, a cardiologist, a medical researcher, competing pharmaceutical company
- Presentation of results matters
- Binary, or not?

Vector-space Model

- **Binary ‘weights’ are too limiting, use term frequency information**
  - **Note on nomeclature: <u>term frequency</u> when used in the literature, indicates an ordinal count – how many times does a term occur in a given document or query**
  - **<u>relative term frequency</u> indicates the percentage**
- **Documents and queries are n-dimensional vectors**
  - **Components indicate the number of occurrences of the given term**
- **The framework is algebraic vector arithmetic**
  - **vectors have length, can be added together**
- **Documents are ranked against queries using a vector comparison**
  - **Sample metrics: Cosine (most common), Inner product, Dice**

Vector-space: Illustration

- Each axis represents one term
- Each document and each query is represented by a vector that describes the terms contained in the collection
- Various measures can be used to determine document similarity; cosine is a common measure
- 100,000 is a typical number of dimensions



<u>Cosine:</u>

$$Sim(\vec{d}, \vec{q}) = \frac{\vec{d} \bullet \vec{q}}{|\vec{d}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,d} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w^2_{i,d}} \times \sqrt{\sum_{i=1}^{t} w^2_{i,q}}}$$

Vector Representation

- Documents and Queries are represented as vectors
- Position 1 corresponds to term 1, position 2 to term 2, position t to term t
- The *weight* of the term is stored in each position

$$D_i = w_{1i}, w_{2i}, \ldots, w_{ti}$$

$$Q = w_{1q}, w_{2q}, \ldots, w_{tq}$$

$$w = 0 \text{ if a term is absent}$$

Term Weighting

- Salton and Buckley (Cornell)
  - In early work (circa 1988) investigated various term weighting schemes
  - Several standard test collections used
- Term weighting gave the vector model a leg-up over other classic models

Assigning Weights to Terms

- Binary Weights
- Raw term frequency (= raw counts)
- 1+log(tf)
  - More occurrences better, but tapers off
- tf x idf
  - Zipf distribution
  - Want to weight terms highly if they are
    - frequent in relevant documents … BUT ALSO
    - infrequent in the collection as a whole

Binary Weights

- Only the presence (1) or absence (0) of a term is included in the vector

docs	*t1*	*t2*	*t3*
D1	1	0	1
D2	1	0	0
D3	0	1	1
D4	1	0	0
D5	1	1	1
D6	1	1	0
D7	0	1	0
D8	0	1	0
D9	0	0	1
D10	0	1	1

Raw Term Weights

- The frequency of occurrence for the term in each document is included in the vector

docs	t1	t2	t3
D1	2	0	3
D2	1	0	0
D3	0	4	7
D4	3	0	0
D5	1	6	3
D6	3	5	0
D7	0	8	0
D8	0	10	0
D9	0	0	1
D10	0	3	5

1 + log(tf) is a variant (see IIR: 6.4.1)

Term Independence Assumptions

- Different terms are considered independent / orthogonal
  - Patently false: doctor/hospital, good/bad car/automobile/SUV
  - Many correlations exist
- Equal Importance Corollary
  - Different terms are equally important
  - Also not true, e.g., the/Kennedy/applet

Zipf Distribution (IIR Chap 5)

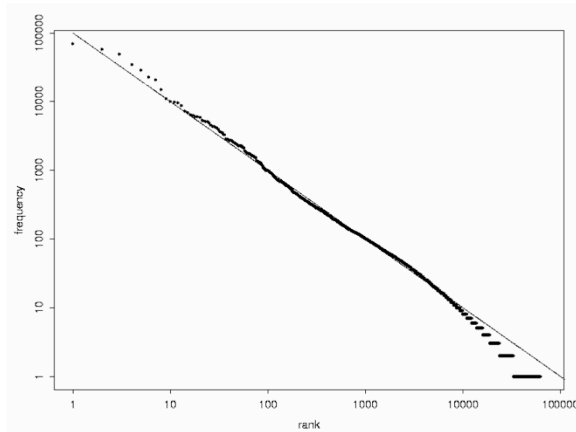
- **The product of the frequency of words (f) and their rank (r) is approximately constant**
  - **Rank = order of words frequency of occurrence**

$$f = C * 1/r$$
$$C \cong N/10$$

- **Another way to state this is with an approximately correct rule of thumb:**
  - **Say the most common term occurs C times**
  - **The second most common occurs C/2 times**
  - **The third most common occurs C/3 times**
  - **…**

Zipf’s law

- **The *k*th most frequent term has frequency proportional to *1/k*.**
  - **IIR Figure 5.2 is similar**



Assigning Weights

- tf x idf weights:
  - term frequency (tf) = raw counts
  - inverse document frequency (idf) -- a way to deal with the problems of the Zipf distribution
- Goal: assign a tf * idf weight to each term in each document

Inverse Document Frequency

- Document frequency is the number of documents a term occurs in
  - Its strictly a property of a term
- Medium document frequency terms appear to be the best for IR
  - Rare terms will only affect a few documents
  - Common terms don't discriminate
- IDF (inverse relative doc frequency)
  - Log motivated by term distribution
  - Several variants
    - Use base 2 logs

$$IDF(t) = \log_2\left(\frac{N}{df(t)}\right)$$

Frequency vs. Resolving Power

The most frequent words are not the most descriptive.

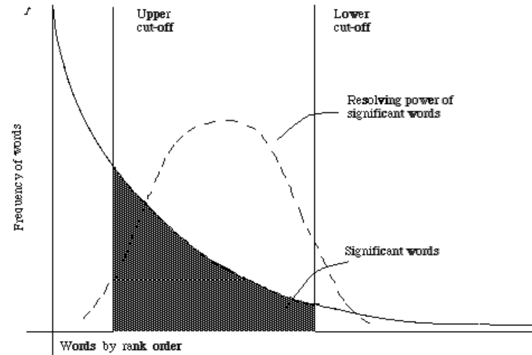


Figure 2.1. A plot of the hyperbolic curve relating f, the frequency of occurrence and r, the rank order (Adaped from Schultz[44] page 120)

Inverse Document Frequency

- **IDF provides high values for rare words and low values for common words**
- **Thus, each dimension can be weighted differently**
  - **Terms that are too common are unimportant**
  - **Decrease the importance of “the” and increase the importance of “Kennedy”**
  - **Weight each term (dimension) by a multiplicative factor**

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 1$$

$$\log\left(\frac{10000}{20}\right) = 8.96$$

$$\log\left(\frac{10000}{1}\right) = 13.2$$

tf x idf

$$w_{ik} = tf_{ik} * \log(N / df_i)$$

T_i = term i

tf_{ik} = frequency of term T_i in document D_k

idf_i = inverse document frequency of term T_i in C

N = total number of documents in the collection C

df_i = the number of documents in C that contain T_i

$$idf_i = \log\left(\frac{N}{df_i}\right)$$

Weighting schemes

- We have seen something of
  - Binary
  - Raw term weights
  - TF/IDF (or TF-IDF or TF*IDF or TFxIDF)
- There are many other possibilities
  - IDF alone
  - Normalized term frequency
- Which scheme is best?
  - Salton and Buckley looked into this, “Term-Weighting Approaches in Automatic Text Retrieval”

Numerous weighting schemes

- **Typical: $w_{ij} = tf_{ij} * idf_i$**
- **Salton and Buckley (1988) suggest for documents**

$$w_{ij} = (tf_{ij} * idf_i) / (\sum^{t}_{i=1} (tf_{ij} * idf_i)^2)^{0.5}$$

- **For queries**

$$w_{iq} = (0.5 + 0.5 * tf_{iq} / tf_{max,q}) * idf_i$$

Normalizing Vectors

- **A vector can be *normalized***
  - **Divide each of its components by its length – here we use the L_2 norm**

$$w_{ik} = \frac{tf_{ik} \log(N/df_i)}{\sqrt{\sum_{z=1}^{t} (tf_{zk})^2 [\log(N/df_z)]^2}} \qquad \|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- **This maps vectors onto the unit sphere:**
  - **The vector length is 1**
  - **Long documents lose advantage**

$$|\vec{d}_j| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1$$

$$CosineSim(\vec{d},\vec{q}) = \frac{\vec{d} \bullet \vec{q}}{|\vec{d}| \times |\vec{q}|} = \frac{\sum_{i=1}^{t} w_{i,d} \times w_{i,q}}{\sqrt{\sum_{i=1}^{t} w^2_{i,d}} \times \sqrt{\sum_{i=1}^{t} w^2_{i,q}}}$$

Example

- **Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights. tf* weights**

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

See IIR pg 97
weights are raw tf

	SaS	PaP	WH
affection	0.996	0.993	0.847
jealous	0.087	0.120	0.466
gossip	0.017	0.000	0.254

- **cos(SAS, PAP) = .996 x .993 + .087 x .120 + .017 x 0.0 = 0.999**
- **cos(SAS, WH) = .996 x .847 + .087 x .466 + .017 x .254 = 0.889**

Courtesy of Manning and Raghavan

Cosine similarity exercise

- ***Rank the following by decreasing cosine similarity.***
  - ***Assume tf-idf weighting:***
  - **Two docs that have only frequent words** *(the, a, an, of)* **in common.**
  - **Two docs that have no words in common.**
  - **Two docs that have many rare words in common** *(wingspan, tailfin).*

Courtesy of Manning and Raghavan

Queries in the vector space model

Central idea: the query as a vector:

- **We regard the query as short *document***
- **We return the documents ranked by the closeness of their vectors to the query, also represented as a vector.**

$$sim(d_j, d_q) = \frac{\vec{d}_j \cdot \vec{d}_q}{\left|\vec{d}_j\right|\left|\vec{d}_q\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,q}^2}}$$

- **Note that d_q is very sparse!**

Courtesy of Manning and Raghavan

Computing Similarity

- **Document frequency is stored in the dictionary**

$$w_{ik} = tf_{ik} \cdot \log\left(\frac{N}{df_i}\right)$$

$w_{ik} = 0$ if document k does not contain term i

- **Raw term frequency is stored in the inverted file postings list**

Vector-space: Implementation

- Observe that query length need only be computed once, and that document length can be precomputed
- Also, a document with no terms in common with a query gets a score of zero
  - This observation facilitates efficient computation of document scores

Cosine Example

D1	D2	D3	D4	D5	D6	D7	D8	Q
apple	apple	apple	banana	apple	pineapple	kiwi	strawberry	apple
banana	kiwi	orange	kiwi	grape	pineapple	pineapple	watermelon	orange
grape	kiwi	orange	strawberry	grape		pineapple		
kiwi	orange	orange		orange				
orange								

Words	DF	IDF
apple	4	1
banana	2	2
grape	2	2
kiwi	4	1
orange	4	1
pineapple	2	2
strawberry	2	2
watermelon	1	3

TFxIDF	D1	D3	Query
apple	1	1	1
banana	2	0	0
grape	2	0	0
kiwi	1	0	0
orange	1	3	1
Sum-of-Squares	11	10	2
Length	3.3166	3.1623	1.4142
Dot product	2	4	2
Sim	0.4264	0.8944	1

Processing a Query

- For each term in the query
  - Count number of times the term occurs – this is the *tf* for the query term
  - Find the term in the inverted dictionary file and get:
    - df_k : the number of documents in the collection with this term
    - Loc : the location of the postings list in the inverted file
    - Calculate Query Weight: w_{kq}
    - Retrieve df_k entries starting at Loc in the postings file
      - This is the postings list…

Processing a Query

- Alternative strategies…
  - Retrieve all of the dictionary entries first, before getting any postings information
    - Why would anybody do this?
  - Just process each term in sequence
- How can we tell how many results there will be?
  - It is possible to put a limitation on the number of items returned
    - How might this be done?

Queries: storing scores

- Like Hashed Boolean OR:
  - Put each document ID from each postings list into hash table
    - Each docid is the key, and the document's score is a value
- Loop over each term in the query
  - Loop down each docid in the postings list
    - Calculate Document weight w_{ik} for the current term
    - Multiply Query weight w_{qk} and Document weight w_{ik} and add it to score
- Scan hash table contents
  - If weights were not normalized, divide doc scores by length
  - throw (docid,scores) into a heap and then return top k
- Alternatives to a hash table / heap
  - sorted array of 'accumulators':
    - one array cell per document in the collection

Problems with Vector Space

- There is no theoretical basis for the assumption of a term space
  - it is more for visualization than having any real basis
  - most similarity measures work about the same regardless of model
- Terms are not really orthogonal dimensions
  - Terms are not independent of all other terms

Vector-space: Summary

- Advantages
  - Achieves good performance
  - 30+ year standard approach
  - Ranks all documents wrt the query
- Disadvantages
  - Assumes orthogonal vector space
  - Dealing with document weights
- Extensions
  - Approximating cosine (efficiently)
  - Pruning postings lists without hurting rankings (much)