- cosine similarity

<u>General / Subtle Notes From Ang</u>

- If you understand all tutorials and information from slides you will at least get a first class.
- Search: Why are skip pointers not useful for queries of the form x OR y
    - Found lots of documents with similar questions to Assignment 2 and other questions relevant to information retrieval!

- Need to know reason term has high weight
    - One reason is tf
    - The other is idf

- Remember cosine similarity between query and document
    - Easy if you understand inner product(dot product)

Rich's List

- The term vocabulary
- Dictionaries and tolerant retrieval
- Index compression - basic idea of why, how lossy vs lossless know
- Postings compression
- Link analysis: Random walks, Markov chains, calculate pagerank (definitely on the exam) issues of this approach."

<u>What to Revise</u> (In Ang's Notes **)**

Lecture 1

- Need to know pritty much all of it

Lecture 2

- Need to know pritty much all of it

Lecture 3

- Tolerant retrieval
- Know the exercises on wild cards, permuterm etc.
- Jaccard, contexts, soundex on index (rules will be provided)
- Levenshtein

Lecture 4

- Not covered

Lecture 5

- Heaps Law + Zipf's Law (very important)
- Blocking, storing gaps
- Know bytecodes, (static variables)?, gamma
- Entropy / entropy document

Lecture 6

- Term frequency & weighting
- Vector space, similarity, tf-idf (very important) (need to know formulas)

- Scoring Example (ranked retrieval I think!)

Lecture 7

- Retrieval of lists (know all formulas)
- Champion list, clustering, idea of what they are
- Basic knowledge of fields of zones
- Aggregate scores basic ideas

Lecture 8

- Difference between query and information needs
- Precision & recall
- Combined measure F, know definition and what happens when alpha(a) = ½
- Kappa model / statistic
- Judges for comparing search

Lecture 2

- biword indexe
- positional index

Lecture 5

- front coding

<u>Quick Notes</u>

- Clustering: Given a set of docs, group them into clusters based on their contents.
- Classification: Given a set of topics, plus a new doc D, decide which topic(s) D belongs to.
- Ranking: Can we learn how to best order a set of documents, e.g., a set of search results

- Tokenization has language issues e.g. tokenizing from English to French / Arabic
- Stop words / list: get rid of common words that bear no meaning, care.. need them for
  - Phrase queries: “King of Denmark”
  - Poems: “To be or not to be”
- Case folding: reduce all letters to lowercase.
- Lemmatization: reduce inflectional/variant forms to base form e.g.
  - am, are, is = be
  - car, cars, car's, cars' = car
  - the boy's cars are different colors = the boy car be different color
- Skip Pointers [Lecture 2]

- Term document incidence matrix is a chart that shows if a term appears in a document by a 1 or 0.
- Inverted index shows the docID’s in which a term occurs.
- Cosine similarity finds out how close a query and a document are together.

- How to judge whether a retrieval system is good or not;
  - How fast does index

- How fast does it search
- What is the cost per searching for a query

- Why would you not use the boolean retrieval model over the vector retrieval model?
  - Requires knowledge of how to construct boolean queries
  - All or nothing
- Why is accuracy not a useful measure for web information retrieval?
  - In a nutshell: ‘all or nothing rule’
  - In an IR system, only a small fraction of the documents are relevant. Even if we have a good IR system that only returns the relevant documents, when compared with a poor system (for example that always returns nothing) there is little difference in accuracy, thus this measurement can’t help evaluate an IR system.
  - Simple trick to maximize accuracy in IR: always say no and return nothing. You then get 99.99% accuracy on most queries.

- Soundex - Class of heuristics to expand a query into phonetic equivalents.

- Heaps Law - How many distinct terms are there in the term vocabulary.
- Heaps Law is the simplest possible relationship between collection size and vocabulary size in log-log space.
- Heaps law: M = kT^b
- M is the size of the vocabulary, T is the number of tokens in the collection.
- Typical values for the parameters k and b are: $30 \le k \le 100$ and $b \approx 0.5$. Thus $M \approx k\sqrt{T}$
- Notice logM = logk + blogT (y = c + bx)
- Example One
  - Angs Sample (.png)
  -
  - Looking at a collection of web pages, you find that there are 8,000 different terms in the first 30,000 tokens and 25,000 different terms in the first 7,000,000 tokens. Assume a search engine indexes a total of 60,000,000,000 (6 × 10 ^ 10) pages, containing 400 tokens on average. What is the size of the vocabulary of the indexed collection as predicted by Heaps’ law?
  -
  - Equation 1
    - log(M1) = logk + blog(T1)
    - log(8,000) = logk + blog(30,000)
  - Equation 2
    - log(M2) = logk + blog(T2)
    - log(25,000) = logk + blog(7,000,000)
  -
  - To get b (take equation 1 from equation 2) (drop the logk)
    - log(25,000) - log(8,000) = b * log(7,000,000) - log(30,000)
    - thus b = (log(25,000) - log(8,000)) / (log(7,000,000) - log(30,000))
    - b = 0.20897587542 ≈ b = 0.209
  -
  - To get k (sub b into either equation)

        - log(8,000) = logk + 0.209 * log(30,000)
        - logk = log(8,000) - (0.209 * log(30,000))
        - logk = 2.96747965343 (ang got 2.9675)
        - k = 10 ^ 2.96747965343
        - k = 927.854019418 (ang got 927.897: he only uses 10 ^ 2.9675)
    -
    - log(M) = logk + 0.209 * log(60,000,000,000 * 400)
    - log(M) = 2.96747965343 + 2.25263361133
    - log(M) = 5.76394380295 ≈ 5.763
    - thus M = 10 ^ 5.76394380295 = 580689.272345 ≈ 5.8 x 10 ^ 5

- Example Two
    - Angs Slides
    -
    - Looking at a collection of web pages, you find that there are 3000 different terms in the first 10,000 tokens and 30,000 different terms in the first 1,000,000 tokens. Assume a search engine indexes a total of 20,000,000,000 (2 × 10 ^ 10) pages, containing 200 tokens on average. What is the size of the vocabulary of the indexed collection as predicted by Heaps' law?
    -
    - Equation 1
        - log(M1) = logk + blog(T1)
        - log(3,000) = logk + blog(10,000)
    - Equation 2
        - log(M2) = logk + blog(T2)
        - log(30,000) = logk + blog(1,000,000)
    -
    - To get b (take equation 1 from equation 2)
        - log(30,000) - log(3,000) = b * log(1,000,000) - log(10,000)
        - thus b = (log(30,000) - log(3,000)) / (log(1,000,000) - log(10,000))
        - b = 0.5
    -
    - To get k (sub b into either equation)
        - log(3,000) = logk + 0.5 * log(10,000)
        - logk = log(3,000) - (0.5 * log(10,000))
        - logk = 1.47712125472
        - k = 10 ^ 1.47712125472
        - k = 30
    -
    - log(M) = logk + 0.5 * log(20,000,000,000 * 200)
    - log(M) = 1.47712125472 + 6.30102999566
    - log(M) = 7.77815125038 ≈ 7.778
    - thus M = 10 ^ 7.77815125038 = 59999999.9995 ≈ 5.9 x 10 ^ 7
    - //thus M = 10 ^ 7.77815125038 = 61376200.5165 ≈ 6 x 10 ^ 7

- Zipfs Law - How the terms are distributed across documents.

- Blocking - Store pointers to every kth term string. By increasing the block size, we get better compression. However, there is a tradeoff between compression and the speed of term lookup.
- Estimate the space usage (and savings compared to 7.6 MB) with blocking, for block sizes of k = 4, 8 and 16.
  - For k = 8.
  - For every block of 8, need to store extra 8 bytes for length
  - For every block of 8, can save 7 * 3 bytes for term pointer. (each block usually 4 bytes, but now only storing pointer to first letter so 1 byte)
  - Saving (+8 – 21)/8 * 400K(terms) = 0.65 MB
  - ie.
    - 8 extra for length & 7 x 3 less = 21 thus -8 +21 = 13 bytes saved
    - 7(one less than block as with 8 lengths only need 7 pointers to first letter) + 8(lengths) = 15 & 7 x 4(original bytes needed) = 28 thus 28 - 15 = 13

- Entropy = measure of randomness & measure of compressibility
- H(p1,…,pn) = p1log(1/p1) + p2log(1/p2) + … + pnlog(1/pn)
- Entropy enables one to compute the compressibility of data without actually needing to compress the data first!
- – plog(p) = **plog(1/p)**
  - p is the probability of an event
  - 1/p is the number of times the event occurs
  - log(k) measures how many bits are needed to represent the outcomes

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights for frequent words like GOOD, INCREASE and LINE.
- The document frequency is the number of documents in the collection that the term occurs in.
- The tf-idf weighting scheme assigns to term t a weight in document d given by

- Champion List - The idea of champion lists (sometimes also called fancy lists or top docs) is to precompute, for each term t in the dictionary, the set of the r documents with the highest weights for t; the value of r is chosen in advance. For tf-idf weighting, these would be the r documents with the highest tf values for term t. We call this set of r documents the champion list for term t.
- At query time, only compute scores for docs in the champion list of some query term. Pick the K top-scoring docs from amongst these

- Clustering is the grouping of a set of documents into clusters. Documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters. Clustering puts together documents that share many terms.

- A zone is a region of the doc that can contain an arbitrary amount of text, e.g., Title, Abstract, References

- Build inverted indexes on zones as well to permit querying e.g. find docs with merchant in the title zone and matching the query gentle rain

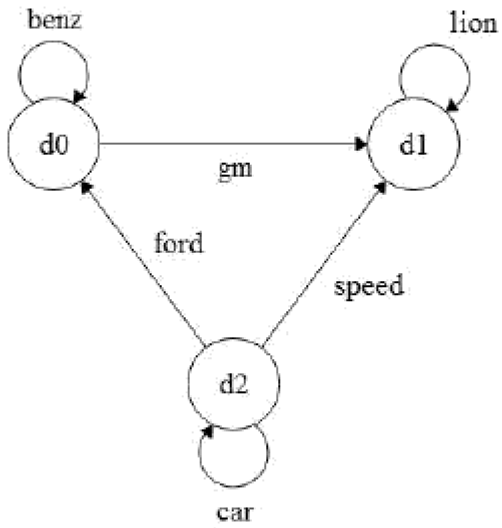
- Aggregate scores - We've seen that score functions can combine cosine, static quality, proximity, etc. How do we know the best combination? Some applications are expert-tuned. Increasingly common: machine-learned

- Relevance: query vs. information need
- Relevance to what?
- First take: relevance to the query, "Relevance to the query" is very problematic.
- Information need i : "I am looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine."
- This is an information need, not a query.
- Query q: [red wine white wine heart attack]
- Consider document d′: At heart of his speech was an attack on the wine industry lobby for downplaying the role of red and white wine in drunk driving.
- d′ is an excellent match for query q . . .
- d′ is not relevant to the information need i .
- User happiness can only be measured by relevance to an information need, not by relevance to queries.

- The Combined Measure F allows us to measure the tradeoff between precision and recall.
    - $F = \frac{2PR}{P + R}$
    - When alpha(a) = 1/2
        - we have the same weighting for precision and recall
        - if alpha(a) = 0.6 then we have more weighting for precision i.e. we want to focus more on precision than recall

- Kappa is a measure of how much judges agree or disagree.
    - $\frac{P(A) - P(E)}{1 - P(E)}$

- Transition Probability Matrix
    - With teleporting, we cannot get stuck in a dead end.
    - More generally, we require that the Markov chain be ergodic (ergodic is used to describe a dynamical system which, broadly speaking, has the same behavior averaged over time as averaged over the space of all the system's states).



Solution:

Link Matrix:

$$\begin{array}{c|ccc} & d_0 & d_1 & d_2 \\ \hline d_0 & 1 & 1 & 0 \\ d_1 & 0 & 1 & 0 \\ d_2 & 1 & 1 & 1 \end{array}$$

Transition matrix without teleporting:

$$\begin{array}{c|ccc} & d_0 & d_1 & d_2 \\ \hline d_0 & 0.5 & 0.5 & 0 \\ d_1 & 0 & 1 & 0 \\ d_2 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{array}$$

Transition matrix with teleporting $\alpha = 0.1$:

$$\begin{array}{c|ccc} & d_0 & d_1 & d_2 \\ \hline d_0 & 0.4833 & 0.4833 & 0.0333 \\ d_1 & 0.0333 & 0.9333 & 0.0333 \\ d_2 & 0.3333 & 0.3333 & 0.3333 \end{array}$$

	$P_t(d_0)$	$P_t(d_1)$	$P_t(d_2)$				
t_0	0	0	1	0.3333	0.3333	0.3333	$\mathrm{d}\vec{P}$
t_1	0.3333	0.3333	0.3333	0.28327	0.58324	0.13329	$\mathrm{d}\vec{P}^2$
t_2	0.28327	0.58324	0.13329	0.200752	0.725669	0.073279	$\mathrm{d}\vec{P}^3$

a = 0.1
0.1 / 3 = 0.0333
1 - (0.1 / 3 * 1) = 0.9666 / 2 = 0.4833
1 - (0.1 / 3 * 2) = 0.9333

To get 0.3333 (on t0 on left of 0.3333's)

- 0 * 0.4833 + 0 * 0.0333 + 1 * 0.3333 = 0.3333

To get 0.3333 (on t0 in middle of 0.3333's)

- 0 * 0.4833 + 0 * 0.9333 + 1 * 0.3333 = 0.3333

To get 0.3333 (on t0 in right of 0.3333's)

- 0 * 0.3333 + 0 * 0.0333 + 1 * 0.3333 = 0.3333

Copy these values down to t1 & repeat process

To get 0.28327

- 0.3333 * 0.4833 + 0.3333 * 0.0333 + 0.3333 * 0.3333
- 0.16108389 + 0.01109889 + 0.11108889 = 0.28327 (round to 5 decimal places)

To get 0.58324

- 0.3333 * 0.4833 + 0.3333 * 0.9333 + 0.3333 * 0.3333
- 0.16108389 + 0.31106889 + 0.11108889 = 0.58324

To get 0.13329

- 0.3333 * 0.0333 + 0.3333 * 0.0333 + 0.3333 * 0.3333
- 0.01109889 + 0.01109889 + 0.11108889 = 0.13328667 = 0.13329

Decode VB code of documents IDs: 000000101001011010010001 (Autumn 2012)

- 00000010 | 10010110 | 10010001 (break it up into 8-bit segments)
- if 8-bit block begins with 1, remove the 1, then remove all zeroes to next one
- else if 8-bit block does not begin with 1, remove all zeroes to the next 1
- Note: if block after first block begins with one, then it is part of the first block
  - in this case put their results together
  - i.e. 10 ...... 10110 => 100010110

- For each digit in 100010110 (from the right hand side)
  - if its a one
    - get its power for its position
    - i.e. for 100011111 its power is 1
    - i.e. for 100011110 its power is 2
    - i.e. for 100011100 its power is 4
    - i.e. for 100011000 its power is 8
  - add up the powers for all of the digit 1's
  - i.e. for 100010110
    - 2 + 4 + 16 + 256 = 278

- For each digit in 10001
- 1 + 16 = 17

- **Answer: 278 = 100010110 and 17 = 10001 thus doc 278 and doc 17**

Decode Gamma code of documents IDs: 11110100011111000101 (Autumn 2012)

- Break up into blocks, where they are separated by the groups of consecutive 1's
  - i.e. 111101000 & 11111000101
- Get rid of the 1's and the first zero
  - the amount of 1's removed should be equal to the number of digits remaining before the next group of consecutive 1's

  - i.e. take the ones and zero from 111101000

        - we get 1000
        - then add back on the 1 to the front (that we chopped off when encoding)
        - i.e. 11000
        - then use same rules as VB decoding with regards to powers
        - we get 8 + 16 = 24
    - i.e. next take the ones and zero from 11111000101
        - we get 00101
        - then 100101
        - then 1 + 4 + 32 = 37
- **Answer: 24 = 11000 gamma code: 111101000 and 37 = 100101 gamma code: 11111000101**