Information Retrieval
INFO/CS 4300

- Instructor: Chris Buckley
  - Office hours Wednesdays 11am Gates 231
- Piazza will be the main communication tool
  - https://piazza.com/cornell/fall2014/info4300/home
  - Lecture notes will appear there.
  - TA office hours and locations appear there.

Course Admin

- Critique 1, Homework 1 – Graded, grades available on-line through CMS, hard copy can be picked up in the homework return room in Gates 216, open Mon-Fri noon-4pm
- Homework 2: Due today at beginning of class (hardcopy plus CMS).
- Warning: class **may** get interrupted late today by guest visitor (Amit Singal, Director of Search at Google).

Previous Lectures

- Overview
- Evaluation 1
- Indexing
- Retrieval
  - Boolean Model
  - Tf*idf weighting
  - Vector Space Model
  - Retrieval Optimization (DAAT, TAAT, safe vs non-safe)
  - Basic Probabilistic Model

Retrieval Models

- Older models
  - Boolean retrieval (still used, special applications)
  - Vector Space model (still used with tf*idf weighting for general retrieval)
  - Basic Probabilistic model
- Newer Probabilistic Models
  - BM25
  - Language models
- Newer tf*idf variants
  - Pivoted unique normalization
- Combining evidence (later in course)
  - Inference networks
  - Learning to Rank

Probability Ranking Principle

- Robertson (1977)
  - "If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request,
  - where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose,
  - the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

Binary Independence Model

- Scoring function is

$$\sum_{i:d_i=1} \log \frac{p_i(1-s_i)}{s_i(1-p_i)}$$

- Query provides information about relevant documents.
- If we assume p_i constant, s_i approximated by entire collection, get ***idf-like weight***

$$\log \frac{0.5(1-\frac{n_i}{N})}{\frac{n_i}{N}(1-0.5)} = \log \frac{N-n_i}{n_i}$$

Contingency Table

	Relevant	Non-relevant	Total
$d_i = 1$	r_i	$n_i - r_i$	n_i
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - n_i$
Total	R	$N - R$	N

$$p_i = (r_i + 0.5)/(R + 1)$$

$$s_i = (n_i - r_i + 0.5)/(N - R + 1)$$

(the +0.5, +1 factors are *smoothing* values)
Gives scoring function:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

Basic Probabilistic Model

- Advantages
  - Gives (with enough assumptions) basis for *idf* weighting model.
    - Important since gives insight into how to incorporate additional information
  - Can be extended
- Disadvantages
  - Poor performance in itself
  - Smoothing needed for any application
  - Natural term dependence extensions don't work in practice.

BM25

- Popular and effective ranking algorithm based on binary independence model
  - adds document and query term weights

$$\sum_{i \in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i}$$

  - k_1, k_2 and K are parameters whose values are set empirically
  - $K = k_1((1-b) + b \cdot \frac{dl}{avdl})$ where *dl* is doc length
  - Typical TREC value for k_1 is 1.2, k_2 varies from 0 to 1000, b = 0.75

- r_i is the # of relevant documents containing term i
- (set to 0 if no relevancy info is known)
- n_i is the # of docs containing term i
- N is the total # of docs in the collection
- R is the number of relevant documents for this query
  - (set to 0 if no relevancy info is known)
- f_i is the frequency of term i in the doc under consideration
- qf_i is the frequency of term i in the query
- k_1 determines how the tf component of the term weight changes as f_i increases. (if 0, then tf component is ignored.) Typical value for TREC is 1.2; so f_i is very non-linear (similar to the use of *log f* in term wts of the vector space model) -- after 3 or 4 occurrences of a term, additional occurrences will have little impact.
- k_2 has a similar role for the query term weights. Typical values (see slide) make the equation less sensitive to k_2 than k_1 because query term frequencies are much lower and less variable than doc term frequencies.
- K is more complicated. Its role is basically to normalize the tf component by document length.
- b regulates the impact of length normalization. (0 means none; 1 is full normalization.)

BM25 Example

- Query with two terms, "president lincoln", (*qf = 1)*
- No relevance information (*r and R are* zero)
- *N* = 500,000 documents
- *"president"* occurs in 40,000 documents (n_1 = 40,000)
- *"lincoln"* occurs in 300 documents (n_2 = 300)
- "president" occurs 15 times in doc (f_1 = 15)
- *"lincoln"* occurs 25 times (f_2 = 25)
- document length is 90% of the average length (*dl/avdl* = .9)
- k_1 = 1.2, b = 0.75, and k_2 = 100
- K = 1.2 · (0.25 + 0.75 · 0.9) = 1.11

BM25 Example

$$\begin{aligned} BM25(Q,D) &= \\ &\log \frac{(0+0.5)/(0-0+0.5)}{(40000-0+0.5)/(500000-40000-0+0+0.5)} \\ &\times \frac{(1.2+1)15}{1.11+15} \times \frac{(100+1)1}{100+1} \\ &+ \log \frac{(0+0.5)/(0-0+0.5)}{(300-0+0.5)/(500000-300-0+0+0.5)} \\ &\times \frac{(1.2+1)25}{1.11+25} \times \frac{(100+1)1}{100+1} \\ \\ &= \log 460000.5/40000.5 \cdot 33/16.11 \cdot 101/101 \\ &+ \log 499700.5/300.5 \cdot 55/26.11 \cdot 101/101 \\ &= 2.44 \cdot 2.05 \cdot 1 + 7.42 \cdot 2.11 \cdot 1 \\ &= 5.00 + 15.66 = 20.66 \end{aligned}$$

BM25 Example

- Effect of term frequencies

Frequency of "president"	Frequency of "lincoln"	BM25 score
15	25	20.66
15	1	12.74
15	0	5.00
1	25	18.2
0	25	15.66

BM25

- Advantages
  - Works well in practice
  - Principled use of parameters
    - Parameters well defined for specific purposes
    - Easy to know what to change for particular applications
    - Parameters reasonably insensitive to minor changes
      - What works well on one collection will work well on another similar collection
  - Incorporates relevance information within the model
- Disadvantages
  - Complicated thus hard to extend
  - Parameters still need to be set
  - Particular parameter form somewhat ad hoc

Retrieval Models

- Older models
  - Boolean retrieval (still used, special applications)
  - Vector Space model (still used with tf*idf weighting for general retrieval)
  - Basic Probabilistic model
- Newer Probabilistic Models
  - BM25
  - Language models
- Newer tf*idf variants
  - Pivoted unique normalization
- Combining evidence (Later in course)
  - Inference networks
  - Learning to Rank

Language Model

- *Unigram language model*
  - probability distribution over the words in a language
  - generation of text consists of pulling words out of a "bucket" according to the probability distribution (and replacing them)
- *N-gram language model*
  - some applications use bigram and trigram language models where probabilities depend on previous words

Language Model

- A *topic* in a document or query can be represented as a language model
  - i.e., words that tend to occur often when discussing a topic will have high probabilities in the corresponding language model
- *Multinomial* distribution over words
  - text is modeled as a finite sequence of words, where there are *t* possible words at each point in the sequence
  - commonly used, but not only possibility
  - doesn't model *burstiness*

LMs for Retrieval

- 3 possibilities:
  - probability of generating the query text from a document language model
  - probability of generating the document text from a query language model
  - comparing the language models representing the query and document topics
- Models of topical relevance

Query-Likelihood Model

- Rank documents by the probability that the query could be generated by the document model (i.e. same topic)
- Start with a query, so calculate P(D|Q) to rank the documents
- Use Bayes' Rule

$$p(D|Q) \stackrel{rank}{=} P(Q|D)P(D)$$

- Assuming prior is uniform, unigram model

$$P(Q|D) = \prod_{i=1}^{n} P(q_i|D)$$

Estimating Probabilities

- Obvious estimate for unigram probabilities is

$$P(q_i|D) = \frac{f_{q_i,D}}{|D|}$$

- *Maximum likelihood estimate*
  - makes the observed value of $f_{q_i,D}$ (the frequency of term q_i in document D) most likely
- Problem: If query words are missing from document, score for the document will be 0
  - Missing 1 out of 6 query words (score of 0) is the same as missing 5 out of 6

Example

- D = (tropical, tropical, fish, fish, fish, water, aquarium)
- Q = salt water tropical fish

- P ("tropical" | D) = 2/7
- P ("fish" | D) = 3/7
- P ("water" | D) = 1 / 7
- P ("salt" | D) = 0/7

- P (Q | D) = 2/7 * 3/7 * 1/7 * 0

Smoothing

- Document texts are a *sample* from the language model
  - Missing words should not have zero probability of occurring
- *Smoothing* is a technique for estimating probabilities for missing (or unseen) words
  - lower (or *discount*) the probability estimates for words that are seen in the document text
  - assign that "left-over" probability to the estimates for the words that are not seen in the text

Estimating Probabilities

- Estimate for unseen words is $\alpha_D P(q_i|C)$
  - $P(q_i|C)$ is the probability for query word *i* in the *collection* language model for collection *C* (background probability)
  - α_D is a parameter
- Estimate for words that occur in a query is

  $(1 - \alpha_D)\, P(q_i|D) + \alpha_D P(q_i|C)$
- Different forms of estimation come from different α_D

Jelinek-Mercer Smoothing

- α_D is a constant, λ
- Gives estimate of

$$p(q_i|D) = (1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}$$

- Ranking score

$$P(Q|D) = \prod_{i=1}^{n}((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

- Use logs for convenience
  - accuracy problems multiplying small numbers

$$\log P(Q|D) = \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

Is *tf*idf* Weight There?

$$\begin{aligned}\log P(Q|D) &= \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) \\ &= \sum_{i:f_{q_i,D}>0} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) + \sum_{i:f_{q_i,D}=0} \log(\lambda\frac{c_{q_i}}{|C|}) \\ &= \sum_{i:f_{q_i,D}>0} \log\frac{((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})}{\lambda\frac{c_{q_i}}{|C|}} + \sum_{i=1}^{n} \log(\lambda\frac{c_{q_i}}{|C|}) \\ &\stackrel{rank}{=} \sum_{i:f_{q_i,D}>0} \log\left(\frac{((1-\lambda)\frac{f_{q_i,D}}{|D|}}{\lambda\frac{c_{q_i}}{|C|}} + 1\right)\end{aligned}$$

- proportional to the term frequency, inversely proportional to the collection frequency

Dirichlet Smoothing

- α_D depends on document length

$$\alpha_D = \frac{\mu}{|D|+\mu}$$

- Gives probability estimation of

$$p(q_i|D) = \frac{f_{q_i,D} + \mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

- and document score

$$\log P(Q|D) = \sum_{i=1}^{n} \log\frac{f_{q_i,D} + \mu\frac{c_{q_i}}{|C|}}{|D|+\mu}$$

Query Likelihood Example

- For the term "president"
  - $f_{qi,D}$ = 15, c_{qi} = 160,000
- For the term "lincoln"
  - $f_{qi,D}$ = 25, c_{qi} = 2,400
- number of word occurrences in the document |d| is assumed to be 1,800
- number of word occurrences in the collection is 10^9
  - 500,000 documents times an average of 2,000 words
- μ = 2,000

Query Likelihood Example

$$\begin{aligned}QL(Q,D) &= \log\frac{15 + 2000 \times (1.6 \times 10^5/10^9)}{1800+2000} \\ &\quad + \log\frac{25 + 2000 \times (2400/10^9)}{1800+2000} \\ &= \log(15.32/3800) + \log(25.005/3800) \\ &= -5.51 + -5.02 = -10.53\end{aligned}$$

- Negative number because summing logs of small numbers

Query Likelihood Example

Frequency of "president"	Frequency of "lincoln"	QL score	
15	25	-10.53	1
15	1	-13.75	3
15	0	-19.05	5
1	25	-12.99	2
0	25	-14.40	4

BM25 comparison

- Effect of term frequencies

Frequency of "president"	Frequency of "lincoln"	BM25 score	QL	
15	25	20.66	1	1
15	1	12.74	**4**	3
15	0	5.00	5	5
1	25	18.2	2	2
0	25	15.66	**3**	4

QL Language Models

- Advantages
  - Principled probabilistic model
  - Intuitively appealing – captures notions we think are important
  - Smoothing itself is appealing
  - Extensible, including smoothing
  - Works well
- Disadvantages
  - Parameter estimation
  - Smoothing methods are somewhat ad hoc (but they are targeted!)

Retrieval Models

- Older models
  - Boolean retrieval (still used, special applications)
  - Vector Space model (still used with tf*idf weighting for general retrieval)
  - Basic Probabilistic model (Not used)
- Newer Probabilistic Models
  - BM25
  - Language models
- Newer tf*idf variants
  - Pivoted unique normalization
- Combining evidence (later in course)
  - Inference networks
  - Learning to Rank

Improved vector space retrieval

- Standard weighting scheme is tf*idf and cosine similarity
  - We've discussed tf*idf variants (and you'll see more of this next week with project 1) and possible alternatives
  - Can cosine document length normalization be improved?
    - Recall that cosine normalization put all vectors on a unit hypersphere
    - Nice theoretically – is it what we want in practice?
    - The answer turns out to be no, but this was discovered in a very roundabout fashion

Case study: Query expansion digression

- I was looking at query expansion techniques in the mid 1990's (early TREC days where the field was doing experimenting with long documents in test collections for the first time)
- Basic tf*idf weights with cosine normalization
- Discovered a method of expanding (lengthening) queries by adding related terms with appropriate weights
  - Worked very well improving retrieval results, in fact too well!!!
  - Was still getting improvements (small, but significant) by adding 200-300 terms.

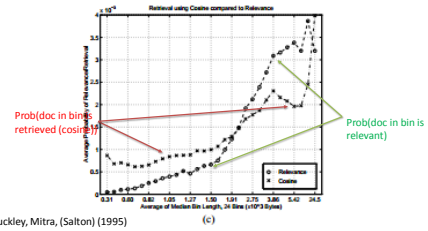
Query expansion => length normalization

- Query expansion results yielded very smooth improvement curves as terms were added. At the end of even 100 added terms, the terms being added were common terms that looked visually to be pretty random. Either
  - A. I had discovered something fundamental like an "atomic" unit or measurement of semantic information content, and how to weight it. Or
  - B. Something else was going on
- I alternated between the two possibilities for a couple of weeks
- The answer turned out to be B.
  - Adding random common terms (with low weights) to the query increases the score of documents randomly (wrt original query), but will tend to increase the score of longer documents more than shorter documents!
  - Cosine length normalization turned out to be biased against long documents

Investigating length normalization

- We had had length normalization on our list of factors to investigate for a while. Steve Robertson and Okapi had just come out with BM25 which approached length normalization differently.
- How do you investigate length normalization issues?
- First step, figure out what is happening in practice now
- We divided the document collection into bins according to length
- Calculated Prob (retrieval of D | D in bin i)
- Calculated Prob (relevance of D | D in bin i)

Prob (cosine retrieval) vs Prob (relevance)

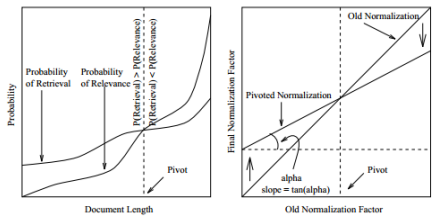


Singhal, Buckley, Mitra, (Salton) (1995)

Cosine favors shorter documents

- In an ideal world, we would hope the two plots would be much closer.
- If we change our retrieval normalization to match Prob(Relevance), will that improve performance?
- Amit Singhal took over finding methods to do this.
- First step was introducing pivoted normalization.

Pivot old normalization around a point

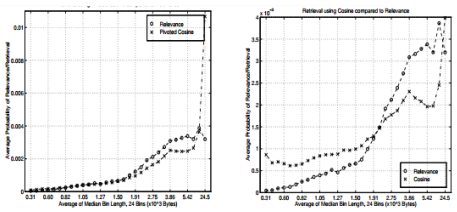


Pivoted_normalization = (1.0 – slope) * pivot + slope * old_normalization

Pivoted Normalization

- *Pivoted_normalization = (1.0 – slope) * pivot + slope * old_normalization*
- If we take the pivot point as being the average old normalization document, and divide by it (doesn't affect ranking)
- *Pivoted_normalization = (1.0 – slope) + slope** $\frac{old\ normalization}{average\ old\ normalization}$
- A document of length average old normalization will be unchanged
- *Slope* can be interpreted as our "belief in length"
- Derived very differently, but similar to Okapi's BM25 length normalization factor (done first) $K = k_1((1-b) + b \cdot \frac{dl}{avdl})$

Pivoted cosine normalization

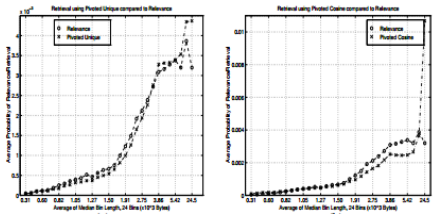


Plot on left matches relevance much better than old plot on right
MAP performance improves by 11.7%

Pivoted unique terms

- Still a problem with very long documents.
- Analyzing further, issue is the tf contribution in very long documents
  - Even standard taking logs is too much (for the length part)
- Instead of cosine of *log (1.0+tf)* weighting or something similar, use the number of unique terms.
- *Pivoted_unique = (1-slope) * pivot + slope * # of unique terms*
- Again can take the pivot as the average number of unique terms in a doc

Pivoted unique vs pivoted cosine



Matches relevance better in the longer documents
MAP performance is another 6% better – total improvement of 18.3% over cosine

Pivoted unique: Final tf*idf weighting

- Lnu.ltc where Lnu weighting in documents is

$$\frac{\frac{(1+\log(tf))}{1+\log(average\ tf)}}{(1.0 - slope) + slope * \frac{(\#\ of\ unique\ terms)}{average\ \#\ of\ unique\ terms}}$$

Where *slope = 0.20* works well across collections

Retrieval Models

- Older models
  - Boolean retrieval (still used, special applications)
  - Vector Space model (still used with tf*idf weighting for general retrieval)
  - Basic Probabilistic model
- Newer Probabilistic Models
  - BM25
  - Language models
- Newer tf*idf variants
  - Pivoted unique normalization
- Combining evidence (later in course)
  - Inference networks
  - Learning to Rank

Features of these newer models

- All work about the same as far as test collection evaluation goes
- All require estimating parameter(s)
  - The estimations are not motivated by the models
  - But most parameters are insensitive to small changes (should do reasonably on other collections)
- Precursors, in some ways, to "Learning to Rank" models, covered later