CMPSCI 546 (590R)
Applied Information Retrieval

Retrieval Models

Retrieval Model Overview

- Vector Space model
- BM25
- Language models

Vector Space Model

- Documents and query represented by a vector of term weights
- Collection represented by a matrix of term weights

$$D_i = (d_{i1}, d_{i2}, \ldots, d_{it}) \qquad Q = (q_1, q_2, \ldots, q_t)$$

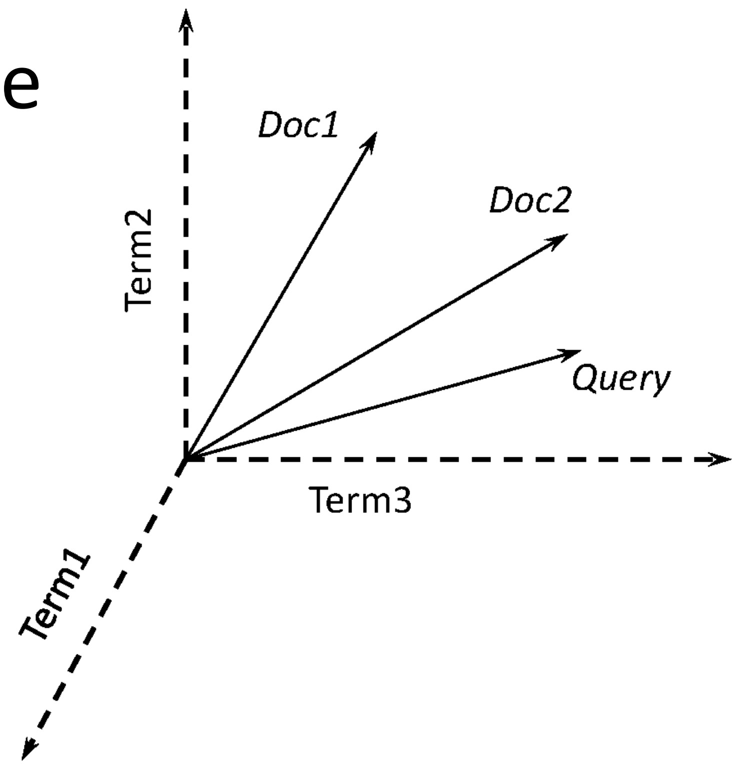
$$\begin{array}{ccccc}
 & Term_1 & Term_2 & \ldots & Term_t \\
Doc_1 & d_{11} & d_{12} & \ldots & d_{1t} \\
Doc_2 & d_{21} & d_{22} & \ldots & d_{2t} \\
\vdots & \vdots & & & \\
Doc_n & d_{n1} & d_{n2} & \ldots & d_{nt}
\end{array}$$

Bag of words model

- Vector representation doesn't consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the <u>bag of words</u> model

Vector Space Model

- Documents “near” the query’s vector are more likely to be relevant to the query



- Warning: 3-d pictures useful, but can be misleading for high-dimensional space

Term frequency tf

- The term frequency $tf_{d,t}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- We want to use tf when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Log-frequency weighting

- The log frequency weight of term t in d is

$$w_{d,t} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{d,t}, & \text{if } \mathrm{tf}_{d,t} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4, etc.

- Score for a document-query pair: sum over terms *t* in both *q* and *d*:

$$score = \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{d,t})$$

- The score is 0 if none of the query terms is present in the document.

Document frequency

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *people, percent, up*)
- A document containing such a term is more likely to be relevant than a document that doesn’t
- But it’s not a sure indicator of relevance.
- For frequent terms, we want high positive weights for words like *people, percent,* and *up*
  - But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- n_k is the document frequency of term *k*: the number of documents that contain *k*
  - n_k is an *inverse* measure of the informativeness of *k*
  - $n_k \leq N$
- We define the idf (inverse document frequency) of *k* by:

$$\mathrm{idf}_k = \log_{10} (N/\mathrm{n}_k)$$

  - We use log (N/n_k) instead of N/n_k to “dampen” the effect of idf.

Term Weights

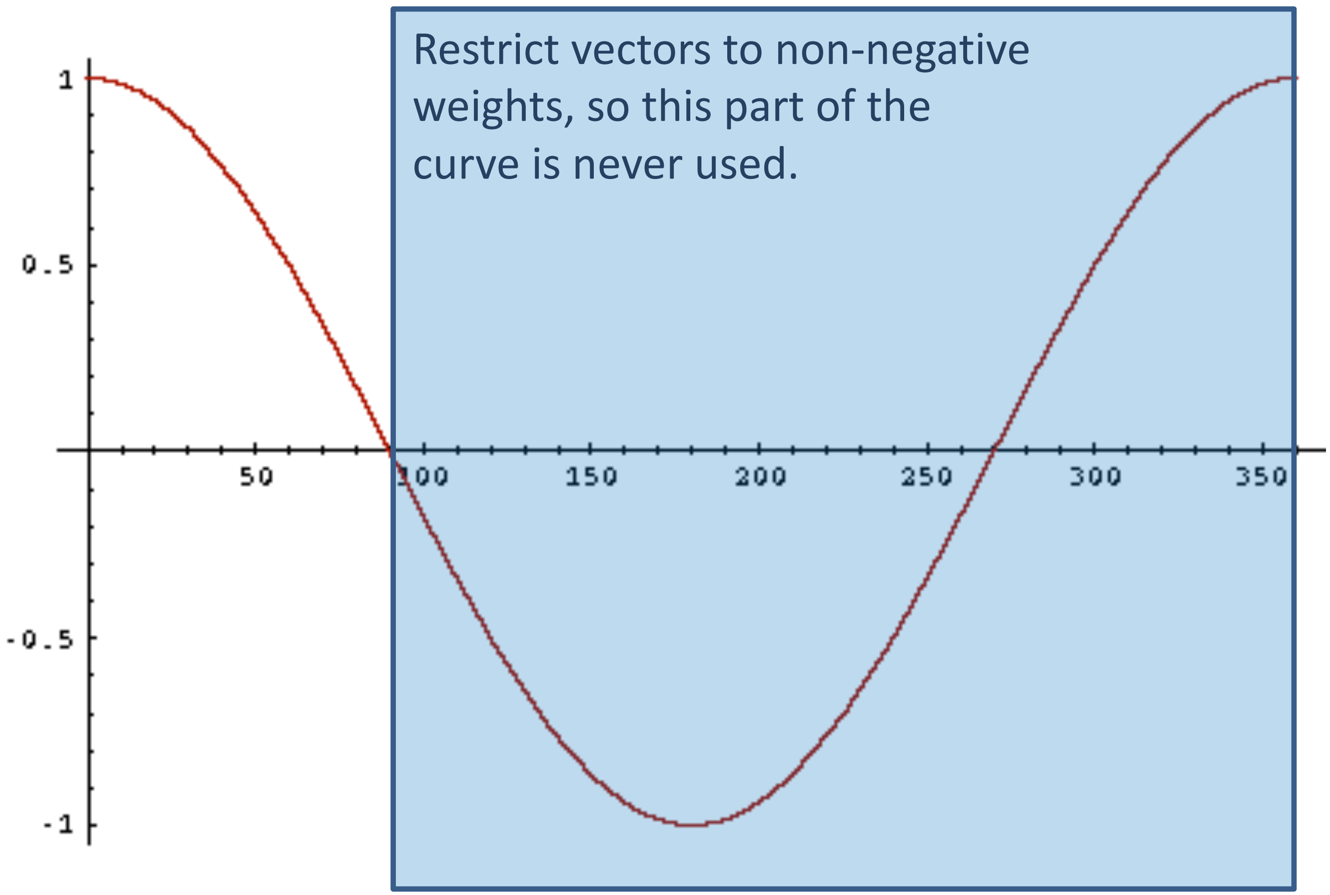
- *tf·idf* weight
  - Term frequency weight measures importance in document
  - Inverse document frequency measures importance in collection
  - Heuristic combination

$$d_{ik} = \frac{(\log(f_{ik})+1)\cdot\log(N/n_k)}{\sqrt{\sum_{k=1}^{t}[(\log(f_{ik})+1.0)\cdot\log(N/n_k)]^2}}$$

From angles to cosines

- The following two notions are equivalent.
  - Rank documents in decreasing order of the angle between query and document
  - Rank documents in increasing order of cosine(query,document)
- Cosine is a monotonically decreasing function for the interval [0^{o}, 180^{o}]

From angles to cosines



- But how should we be computing cosines?

cosine(query,document)

Dot product

Unit vectors

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q} \bullet \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|} \bullet \frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i is the weight of term *i* in the query
d_i is the weight of term *i* in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

If create unit vectors, then just dot product to calculate

Components of Similarity

- The "inner product" (aka dot product) is the key to the similarity function

$$\vec{d}_i \cdot \vec{q} = \sum_{j=1}^{t} d_{ij} q_j$$

Example:

$$[1\ 2\ 3\ 0\ 2] \cdot [2\ 0\ 1\ 0\ 2]$$
$$= 1\times 2 + 2\times 0 + 3\times 1 + 0\times 0 + 2\times 2 = 9$$

- The denominator handles document length normalization (optional, but helps efficiency)

$$\left|\vec{d}_i\right| = \sqrt{\sum_{j=1}^{t} d_{ij}^2}$$

Example:

$$|\,[1\ 2\ 3\ 0\ 2]\,|$$
$$= \sqrt{1+4+9+0+4} = \sqrt{18} \approx 4.24$$

Computing cosine scores

CosineScore(q)

float Scores$[N] = 0$

float Length$[N]$ // Length of each document (from indexing)

for each query term t

do calculate $w_{t,q}$ and fetch postings list for t

  for each pair$(d, \mathrm{tf}_{t,d})$ in postings list

  do $Scores[d] += w_{t,d} \times w_{t,q}$

Read the array *Length*

Note that $w_{t,d}$ is probably Based on $\mathrm{tf}_{t,d}$ and idf_t

for each d

do $Scores[d] = Scores[d] / Length[d]$

return Top K components of $Scores[]$

Variations of TF-IDF

weighting scheme	document term weight	query term weight
1	$f_{i,j} * \log \frac{N}{n_i}$	$(0.5 + 0.5 \frac{f_{i,q}}{max_i \ f_{i,q}}) * \log \frac{N}{n_i}$
2	$1 + \log f_{i,j}$	$\log(1 + \frac{N}{n_i})$
3	$(1 + \log f_{i,j}) * \log \frac{N}{n_i}$	$(1 + \log f_{i,q}) * \log \frac{N}{n_i}$

BM25

- BM25 was created as the result of a series of experiments on variations of the probabilistic model
- A good term weighting is based on three principles
  - inverse document frequency
  - term frequency
  - document length normalization
- The classic probabilistic model covers only the first of these principles
- This reasoning led to a series of experiments which led to the BM25 ranking formula

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

Language models

- Based on the notion of probabilities and processes for generating text
- Documents are ranked based on the probability that they generated the query
- Best/partial match

Unigram Language Model

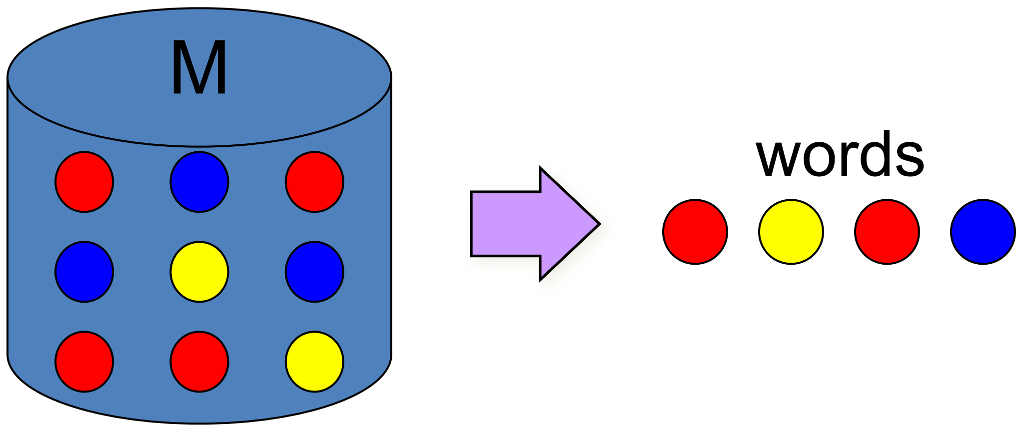
- Assume each word is generated independently
  - Obviously, this is not true…
  - But it seems to work well in practice!
- The probability of a string, given a model:

$$P(q_1 \ldots q_k \mid M) = \prod_{i=1}^{k} P(q_i \mid M)$$

The probability of a sequence of words decomposes into a product of the probabilities of individual words

A Physical Metaphor

- Colored balls are randomly drawn from an urn (with replacement)



P (🔴 🟡 🔴 🔵) = P (🔴) $\times$ P (🟡) $\times$ P (🔴) $\times$ P (🔵)

$= (4/9) \times (2/9) \times (4/9) \times (3/9)$

Query-Likelihood Model

- Rank documents by the probability that the query could be generated by the document model (i.e. same topic)
- Given a query, we are interested in P(D|Q)
- One way is to use Bayes' Rule

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

- Assuming prior is uniform, and assuming independence, we get the unigram model

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

Estimating Probabilities

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

- Obvious estimate for unigram probabilities is

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

- *Maximum likelihood estimate*
  - makes the observed value of $f_{q_i;D}$ most likely
- If query words are missing from document, score will be zero
  - Missing 1 out of 4 query words same as missing 3 out of 4

Smoothing

- Document texts are a *sample* from the language model
  - Missing words should not have zero probability of occurring
  - A document is a very small sample of words, and the maximum likelihood estimate will be inaccurate.
- *Smoothing* is a technique for estimating probabilities for missing (or unseen) words
  - reduce (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 is smoothed:

$$(1 - \alpha_D)\, P(q_i|D) + \alpha_D\, P(q_i|C)$$

- 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

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

Dirichlet Smoothing

- α_D depends on document length

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

$(1 - \alpha_D)\ P(q_i|D) + \alpha_D P(q_i|C)$

- Gives probability estimation of

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

- and document score

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

- Jelinek-Mercer with document-dependent parameter