The Vector Space Model

LBSC 708A/CMSC 838L

Session 3, September 18, 2001

Douglas W. Oard

Agenda

- Questions
- Ranked retrieval
- Vector space method
- Latent semantic indexing

Strong Points of Boolean Retrieval

- Accurate, if you know the right strategy
- Efficient for the computer
- More concise than natural language
- Easy to understand
- A standard approach
- Works across languages (controlled vocab.)

Weak Points of Boolean Retrieval

- Words can have many meanings (free text)
- Hard to choose the right words
  - Must be familiar with the field
- Users must learn Boolean logic
- Can find relationships that don’t exist
- Sometimes find too many documents
  - (and sometimes get none)

What is Relevance?

- **Relevance** relates a topic and a document
  - Duplicates are equally relevant by definition
  - Constant over time and across users
- **Pertinence** relates a task and a document
  - Accounts for quality, complexity, language, …
- **Utility** relates a user and a document
  - Accounts for prior knowledge
- We seek utility, but relevance is what we get!

Ranked Retrieval Paradigm

- Exact match retrieval often gives useless sets
  - No documents at all, or way too many documents
- Query reformulation is one “solution”
  - Manually add or delete query terms
- “Best-first” ranking can be superior
  - Select every document within reason
  - Put them in order, with the “best” ones first
  - Display them one screen at a time

Advantages of Ranked Retrieval

- Closer to the way people think
  - Some documents are better than others
- Enriches browsing behavior
  - Decide how far down the list to go as you read it
- Allows more flexible queries
  - Long and short queries can produce useful results

Ranked Retrieval Challenges

- “Best first” is easy to say but hard to do!
  - Probabilistic retrieval tries to approximate it
- How can the user understand the ranking?
  - It is hard to use a tool that you don’t understand
- Efficiency may become a concern
  - More complex computations take more time

Partial-Match Ranking

- Form several result sets from one long query
  - Query for the first set is the AND of all the terms
  - Then all but the 1st term, all but the 2nd, …
  - Then all but the first two terms, …
  - And so on until each single term query is tried
- Remove duplicates from subsequent sets
- Display the sets in the order they were made
  - Document rank within a set is arbitrary

Partial Match Example

information AND retrieval

Readings in Information Retrieval
Information Storage and Retrieval
Speech-Based Information Retrieval for Digital Libraries
Word Sense Disambiguation and Information Retrieval

information NOT retrieval

The State of the Art in Information Filtering

retrieval NOT information

Inference Networks for Document Retrieval
Content-Based Image Retrieval Systems
Video Parsing, Retrieval and Browsing
An Approach to Conceptual Text Retrieval Using the EuroWordNet …
Cross-Language Retrieval: English/Russian/French

Similarity-Based Queries

- Treat the query as if it were a document
  - Create a query bag-of-words
- Find the similarity of each document
  - Using the coordination measure, for example
- Rank order the documents by similarity
  - Most similar to the query first
- Surprisingly, this works pretty well!
  - Especially for very short queries

Document Similarity

- How similar are two documents?
  - In particular, how similar is their bag of words?

1: Nuclear fallout contaminated Montana.

2: Information retrieval is interesting.

3: Information retrieval is complicated.

	1	2	3
complicated			1
contaminated	1		
fallout	1		
information		1	1
interesting		1	
nuclear	1		
retrieval		1	1
siberia	1		

The Coordination Measure

- Count the number of terms in common
  - Based on Boolean bag-of-words
- Documents 2 and 3 share two common terms
  - But documents 1 and 2 share no terms at all
- Useful for “more like this” queries
  - “more like doc 2” would rank doc 3 ahead of doc 1
- Where have you seen this before?

Coordination Measure Example

	1	2	3
complicated			1
contaminated	1		
fallout	1		
information		1	1
interesting		1	
nuclear	1		
retrieval		1	1
siberia	1		

Query: complicated retrieval
Result: 3, 2

Query: interesting nuclear fallout
Result: 1, 2

Query: information retrieval
Result: 2, 3

Term Frequency

- Terms tell us about documents
  - If “rabbit” appears a lot, it may be about rabbits
- Documents tell us about terms
  - “the” is in every document -- not discriminating
- Documents are most likely described well by rare terms that occur in them frequently
  - Higher “term frequency” is stronger evidence
  - Low “collection frequency” makes it stronger still

The Document Length Effect

- Humans look for documents with useful parts
  - But probabilities are computed for the whole
- Document lengths vary in many collections
  - So probability calculations could be inconsistent
- Two strategies
  - Adjust probability estimates for document length
  - Divide the documents into equal “passages”

Computing Term Contributions

- "Okapi BM25 weights" are the best known
  - Discovered mostly through trial and error

Let N be the number of documents in the collection

Let n_t be the number of documents containing term t

Let $\mathrm{tf}_{t,d}$ be the frequency of term t in document d

Let $w_{t,d}$ be the contribution of term t to the relevance of document d

$$\text{Then } w_{t,d} = 0.4 + \frac{0.6 \cdot \mathrm{tf}_{t,d}}{\mathrm{tf}_{t,d} + 0.5 + 1.5\dfrac{\mathrm{length(d)}}{\mathrm{avglen}}} \cdot \frac{\log \dfrac{N + 0.5}{n_t}}{\log N + 1}$$

Incorporating Term Frequency

- High term frequency is evidence of meaning
  - And high IDF is evidence of term importance
- Recompute the bag-of-words
  - Compute TF * IDF for every element

Let N be the total number of documents

Let n of the N documents contain term i

Let $\mathrm{tf}_{i,j}$ be the number of times term i appears in document j

Then $w_{i,j} = \mathrm{tf}_{i,j} \cdot \log \frac{N}{n}$

Weighted Matching Schemes

- Unweighted queries
  - Add up the weights for every matching term
- User specified query term weights
  - For each term, multiply the query and doc weights
  - Then add up those values
- Automatically computed query term weights
  - Most queries lack useful TF, but IDF may be useful
  - Used just like user-specified query term weights

TF*IDF Example

	$tf_{i,j}$				idf_i	$w_{i,j}$			
	1	2	3	4		1	2	3	4
complicated			5	2	0.301			1.51	0.60
contaminated	4	1	3		0.125	0.50	0.13	0.38	
fallout	5		4	3	0.125	0.63		0.50	0.38
information	6	3	3	2	0.000				
interesting		1			0.602		0.60		
nuclear	3		7		0.301	0.90		2.11	
retrieval		6	1	4	0.125		0.75	0.13	0.50
siberia	2				0.602	1.20			

Unweighted query:
contaminated retrieval
Result: 2, 3, 1, 4

Weighted query:
contaminated(3) retrieval(1)
Result: 1, 3, 2, 4

IDF-weighted query:
contaminated retrieval
Result: 2, 3, 1, 4

Document Length Normalization

- Long documents have an unfair advantage
  - They use a lot of terms
    - So they get more matches than short documents
  - And they use the same words repeatedly
    - So they have much higher term frequencies
- Normalization seeks to remove these effects
  - Related somehow to maximum term frequency
  - But also sensitive to the of number of terms

“Cosine” Normalization

- Compute the length of each document vector
  - Multiply each weight by itself
  - Add all the resulting values
  - Take the square root of that sum
- Divide each weight by that length

Let $w_{i,j}$ be the unnormalized weight of term *i* in document *j*

Let $w'_{i,j}$ be the normalized weight of term *i* in document *j*

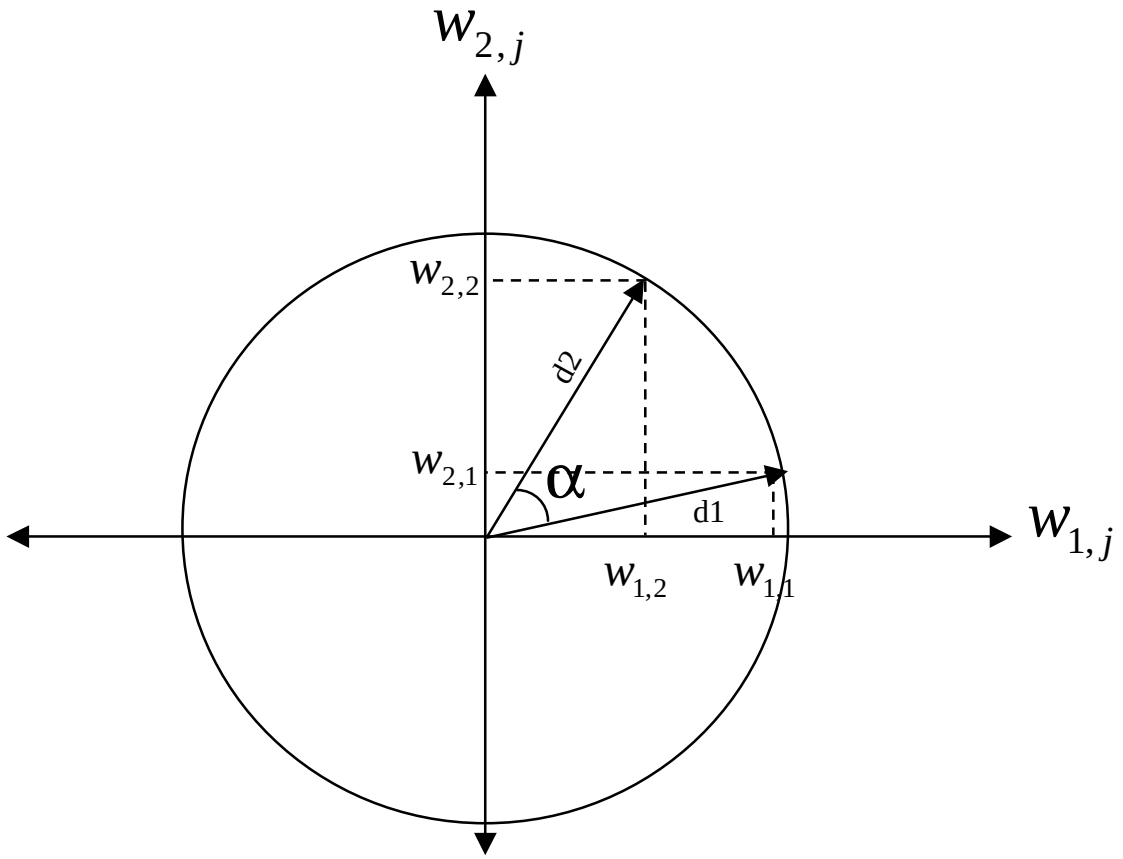
Then $w'_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_j w_{i,j}^2}}$

Cosine Normalization Example

	$tf_{i,j}$				idf_i	$w_{i,j}$				$w'_{i,j}$			
	1	2	3	4		1	2	3	4	1	2	3	4
complicated			5	2	0.301			1.51	0.60			0.57	0.69
contaminated	4	1	3		0.125	0.50	0.13	0.38		0.29	0.13	0.14	
fallout	5		4	3	0.125	0.63		0.50	0.38	0.37		0.19	0.44
information	6	3	3	2	0.000								
interesting		1			0.602		0.60				0.62		
nuclear	3		7		0.301	0.90		2.11		0.53		0.79	
retrieval		6	1	4	0.125		0.75	0.13	0.50		0.77	0.05	0.57
siberia	2				0.602	1.20				0.71			
					Length →	1.70	0.97	2.67	0.87				

Unweighted query: contaminated retrieval, Result: 2, 4, 1, 3 (compare to 2, 3, 1, 4)

Why Call It “Cosine”?



Let document 1 have unit length with coordinates $w_{1,1}$ and $w_{2,1}$

Let document 2 have unit length with coordinates $w_{1,2}$ and $w_{2,2}$

Then $\cos\alpha = \sqrt{(w_{1,1} \cdot w_{1,2}) + (w_{2,1} \cdot w_{2,2})}$

Interpreting the Cosine Measure

- Think of a document as a vector from zero
- Similarity is the angle between two vectors
  - Small angle = very similar
  - Large angle = little similarity
- Passes some key sanity checks
  - Depends on pattern of word use but not on length
  - Every document is most similar to itself

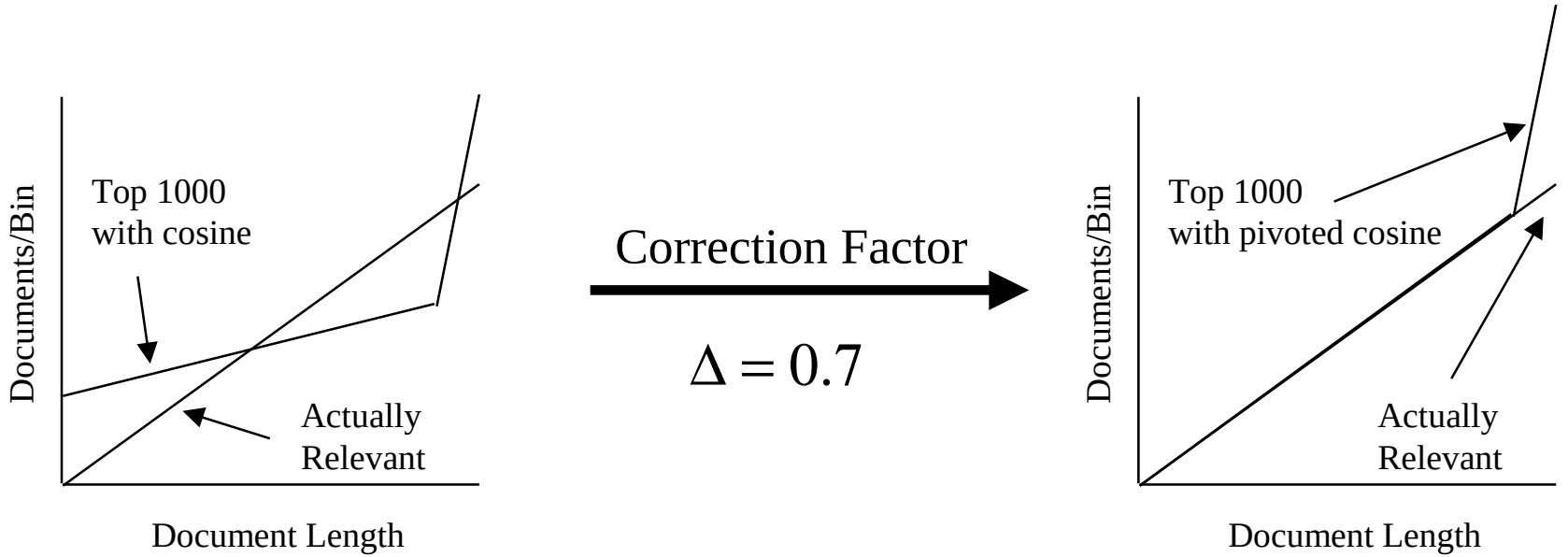
Summary So Far

- Find documents most similar to the query
- Optionally, Obtain query term weights
  - Given by the user, or computed from IDF
- Compute document term weights
  - Some combination of TF and IDF
- Normalize the document vectors
  - Cosine is one way to do this
- Compute inner product of query and doc vectors
  - Multiply corresponding elements and then add

Pivoted Cosine Normalization

- Start with a large test collection
  - Documents, topics, relevance judgments
- Sort the documents by increasing length
  - Divide into “bins” of 1,000 documents each
  - Find the number of relevant documents in each
- Use any normalization find the top 1,000 docs
  - Find the number of “top” documents in each bin
- Plot number of relevant and “top” documents

Sketches of the Plots



Slope correction: $(1-\Delta)+(\Delta \cdot \frac{\text{vector\_length}}{\text{average\_vector\_length}})$

Pivoted Unique Normalization

- Pivoting exacerbates the cosine plot's "tail"
  - Very long documents get an unfair advantage
- Coordination matching lacks such a tail
  - The number of unique terms grows smoothly
  - But pivoting is even more important (Δ=0.25)

Let t_j be the number of unique terms in document j

$$w_{i,j} = \frac{\frac{1+\log(\mathrm{tf}_{i,j})}{1+\log(\underset{i}{\mathrm{avg}}(\mathrm{tf}_{i,j}))}}{((1-\Delta)\cdot\underset{j}{\mathrm{avg}}(t_j))+(\Delta\cdot t_j)}$$

Passage Retrieval

- Another approach to long-document problem
  - Break it up into coherent units
- Recognizing topic boundaries is hard
  - But overlapping 300 word passages work fine
- Document rank is best passage rank
  - And passage information can help guide browsing

Stemming

- Suffix removal can improve performance
  - In English, word roots often precede modifiers
  - Roots often convey topicality better
- Boolean systems often allow truncation
  - limit? -> limit, limits, limited, limitation, …
- Stemming does automatic truncation
- More complex algorithms can find true roots
  - But better retrieval performance does not result

Porter Stemmer

- Nine step process, 1 to 21 rules per step
  - Within each step, only the first valid rule fires
- Rules rewrite suffixes. Example:

```
static RuleList step1a_rules[] = {
        101, "sses",   "ss",          3,   1, -1, NULL,
        102, "ies",    "i",           2,   0, -1, NULL,
        103, "ss",     "ss",          1,   1, -1, NULL,
        104, "s",      LAMBDA, 0, -1, -1, NULL,
        000, NULL, NULL,       0,  0,   0, NULL,
};
```

Latent Semantic Indexing

- Term vectors can reveal term dependence
  - Look at the matrix as a “bag of documents”
  - Compute term similarities using cosine measure
- Reduce the number of dimensions
  - Assign similar terms to a single composite
  - Map the composite term to a single dimension
- This can be done automatically
  - But the optimal technique muddles the dimensions
    - Terms appear anywhere in the space, not just on an axis

$W_{i,j}$ LSI Transformation $T_{i,k}^{(4)}$

	d1	d2	d3	d4
t1				1
t2				1
t3		1		
t4			1	
t5				1
t6				1
t7	1			
t8			1	
t9		1	1	1
t10	1			
t11				1
t12		1	1	
t13	1			
t14		1		
t15	1			
t16		1	1	
t17	1			
t18		2	1	1
t19			1	

$$W = T \times S \times D$$
$$t \times d \quad t \times k \quad k \times k \quad k \times d$$

	k1	k2	k3	k4
t1	0.11	0.39		0.07
t2	0.11	0.39		0.07
t3	0.18	-0.10		-0.39
t4	0.15	-0.16		0.44
t5	0.11	0.39		0.07
t6	0.11	0.39		0.07
t7			0.45	
t8	0.15	-0.16		0.44
t9	0.44	0.13		0.12
t10			0.45	
t11	0.11	0.39		0.07
t12	0.33	-0.26		0.05
t13			0.45	
t14	0.18	-0.10		-0.39
t15			0.45	
t16	0.33	-0.26		0.05
t17			0.45	
t18	0.63	0.02		-0.27
t19	0.15	-0.16		0.44

Computing Similarity

- First choose k
  - Never greater than the number of docs or terms
- Add the weighted vectors for each term
  - Multiply each vector by term weight
  - Sum each element separately
- Do the same for query or second document
- Compute inner product
  - Multiply corresponding elements and add

LSI Example

$W_{i,j}$

	d2	d3
t1		
t2		
t3	1	
t4		1
t5		
t6		
t7		
t8		1
t9	1	1
t10		
t11		
t12	1	1
t13		
t14	1	
t15		
t16	1	1
t17		
t18	2	1
t19		1

	k1	k2	k3	k4
t3	0.18	-0.10		-0.39
t9	0.44	0.13		0.12
t12	0.33	-0.26		0.05
t14	0.18	-0.10		-0.39
t16	0.33	-0.26		0.05
t18	0.63	0.02		-0.27
t18	0.63	0.02		-0.27
Sum2	2.72	-0.55		-1.10

	k1	k2	k3	k4
t4	0.15	-0.16		0.44
t8	0.15	-0.16		0.44
t9	0.44	0.13		0.12
t12	0.33	-0.26		0.05
t16	0.33	-0.26		0.05
t18	0.63	0.02		-0.27
t19	0.15	-0.16		0.44
Sum3	2.18	-0.85		1.26

$$d_2 \cdot d_3 = 5.00$$

$$\text{Sum}_2 \cdot \text{Sum}_3 = 5.01$$

<u>Removing Dimensions</u>

k1 and k2 = 6.40

k1 alone = 5.92

Benefits of LSI

- Removing dimensions can improve things
  - Assigns similar vectors to similar terms
- Queries and new documents easily added
  - “Folding in” as weighted sum of term vectors
- Gets the same cosines with shorter vectors

Weaknesses of LSI

- Words with several meanings confound LSI
  - Places them at the midpoint of the right positions
- LSI vectors are dense
  - Sparse vectors (tf*idf) have several advantages
- The required computations are expensive
  - But T matrix and doc vectors are done in advance
  - Query vector and cosine at query time
- The cosine may not be the best measure
  - Pivoted normalization can probably help

Two Minute Paper

- Vector space retrieval finds documents that are “similar” to the query. Why is this a reasonable thing to do?

- What was the muddiest point in today’s lecture?