CSCI 5417
Information Retrieval Systems

Jim Martin

Lecture 5
9/6/2011

Today 9/6

- Vector space model
- New homework

Recap

- We've covered a variety of types of indexes
- And a variety of ways to build indexes
- And a variety of ways to process tokens
- And boolean search
- Now what?

Beyond Boolean

- Thus far, our queries have been Boolean
  - Docs either match or they don't
- Ok for expert users with precise understanding of their needs and the corpus
- Not good for (the majority of) users with poor Boolean formulation of their needs
- Most users don't want to wade through 1000's of results (or get 0 results)
  - Hence the popularity of search engines which provide a ranking.

Scoring

- Without some form of ranking, boolean queries usually result in too many or too few results.
- With ranking, the number of returned results is irrelevant.
  - The user can start at the top of a ranked list and stop when their information need is satisfied

Ranked Retrieval

- Given a query, assign a numerical score to each doc in the collection
- Return documents to the user based on the ranking derived from that score
- How?
  - A considerable amount of the research in IR over the last 20 years...
    - Extremely empirical in nature

Back to Term x Document Matrices

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Documents and terms can be thought of as vectors of 1's a 0's

Back to Term x Document Matrices

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Consider *instead* the number of occurrences of a term t in a document d, denoted $\mathrm{tf}_{t,d}$

Scoring: Beyond Boolean AND

- Given a free-text query *q* and a document *d* define

$$\text{Score}(q,d) = \Sigma_{t \in q}\, \text{tf}_{t,d}$$

That is, simply add up the term frequencies of all query terms in the document

Holding the query static, this assigns a score to each document in a collection; now rank documents by this score.

Term Frequency: *Local Weight*

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear, but it does seem like more is better, a lot isn't proportionally better than a few
  - One scheme commonly used:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0,\ \ 1 + \log tf_{t,d} \text{ otherwise}$$

Potential Problem

Consider query ***ides of march***

- *Julius Caesar* has 5 occurrences of ***ides***
- No other play has ***ides***
- ***march*** occurs in over a dozen
- SO... *Julius Caesar* should do well since it has counts from both ides and march

BUT all the plays contain ***of,*** some many times. So by this scoring measure, the top-scoring play is likely to be the one with the most number of ***of***'s

Term Frequency $tf_{t,d}$

- ***Of*** is a frequent word overall. Longer docs will have more ***of***s. But not necessarily more ***march*** or ***ides***
- Hence longer docs are favored because they're more likely to contain frequent query terms
  - Probably not a good thing

Global Weight

- Which of these tells you more about a doc?
  - 10 occurrences of *hernia*?
  - 10 occurrences of *the*?
- Would like to attenuate the weights of *common terms*
  - But what does "common" mean?
  - 2 options: Look at
    - Collection frequency
      - The total number of occurrences of a term in the entire collection of documents
    - Document frequency

9/6/11 13

Collection vs. Document Frequency

Consider...

Word	*cf*	*df*
try	10422	8760
insurance	10440	3997

9/6/11 14

Inverse Document Frequency

- So how can we formalize that?
  - Terms that appear across a large proportion of the collection are less useful. They don't distinguish among the docs.
  - So let's use that proportion as the key.
  - And let's think of boosting useful terms rather than demoting useless ones.

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

Reuters RCV1 800K docs

- Logarithms are base 10

term	df_t	idf_t
car	18,165	1.65
auto	6723	2.08
insurance	19,241	1.62
best	25,235	1.5

tf x idf (or tf.idf or tf-idf)

- We still ought to pay attention to the local weight... so

$$w_{t,d} = tf_{t,d} \times \log(N / df_t)$$

$tf_{t,d}$ = frequency of term t in document d

N = total number of documents

df_t = the number of documents that contain term t

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

Summary: TfxIdf

- "TFxIDF is usually used to refer to a family of approaches.

term frequency		document frequency		normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1+\log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5+\frac{0.5\times\mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log\frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$ (Section 17.4.4)
b (boolean)	$\begin{cases}1 & \text{if } \mathrm{tf}_{t,d} > 0\\ 0 & \text{otherwise}\end{cases}$			b (byte size)	$1/CharLength^{\alpha}, \alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Real-valued term vectors

- Still <u>Bag of words</u> model
- Each is a vector in $\mathbb{R}^M$
  - Here log-scaled *tf.idf*

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Assignment 2

- Download and install Lucene
- How does Lucene handle (using standard methods)
  - Case, stemming, stop lists and multiword queries
- Download index the medical.txt collection
  - DocID, abstracts, titles, keywords, and text
  - How big is the resulting index?
    - Terms and size of index
  - Retrieve document IDs (from the lucene hits) from the queries in queries.txt
  - Compare against relevance judgments in qrels.txt

Assignment 2

- Collection
  - 54,710 medical abstracts
    - All in a single file
  - 63 queries with relevance judgments

Sample Doc

```
.I 15
.U
87049104
.S
Am J Emerg Med 8703; 4(6):552-3
.M
Adolescence; Atropine/*TU; Baclofen/*PO; Bradycardia/CI/*DT; Case Report; Human;
Hypotension/CI/*DT; Male.
.T
Atropine in the treatment of baclofen overdose.
.P
JOURNAL ARTICLE.
.W
A patient suffering baclofen overdose successfully treated with atropine is
reported. Three hours after admission for ingestion of at least 300 mg baclofen as
a single dose, the patient became comatose and subsequently bradycardic, hypo
tensive, and hypothermic. A prompt increase in heart rate and blood pressure
followed administration of 1 mg of atropine sulfate. Atropine appears to be useful
in treating cases of baclofen overdose complicated by bradycardia and hypotension.
.A
Cohen MB; Gailey RA; McCoy GC.
```

Sample Query

```
<top>
<num> Number: OHSU4
<title> 58 yo with cancer and
hypercalcemia
<desc> Description:
effectiveness of etidronate in
treating hypercalcemia of malignancy
</top>
```

9/6/11 23

Qrels

OHSU1	87316326	1
OHSU1	87202778	1
OHSU1	87157536	2
OHSU1	87157537	2
OHSU1	87097544	2
OHSU1	87316316	1
OHSU2	87230756	1
OHSU2	87076950	1
OHSU2	87254296	2
OHSU2	87058538	2
OHSU2	87083927	2
OHSU2	87309677	2

9/6/11 24

Evaluation

- As we’ll see in Chapter 8, there are lots of ways to do evaluation. Which mostly lead to different design decisions.
- For this assignment, we’ll use R-precision (see page 148).
  - Basically, if a given query has N relevant docs, then we look at the top N returned results and compute precision within that set.
  - So if we found all and only relevant docs we get a 1.
  - Then we average that over the set of queries we’re using.

9/6/11 25

Assignment

- Part 1
  - Do a straightforward (not too stupid) lucene search solution for this dataset
  - Measure how well it works with R-Precision
- Part 2
  - Make it better

9/6/11 26

Back to Scoring

- Ok, we've change our document representation (the term-document matrix)
- How does that help scoring?

9/6/11 27

Documents as Vectors

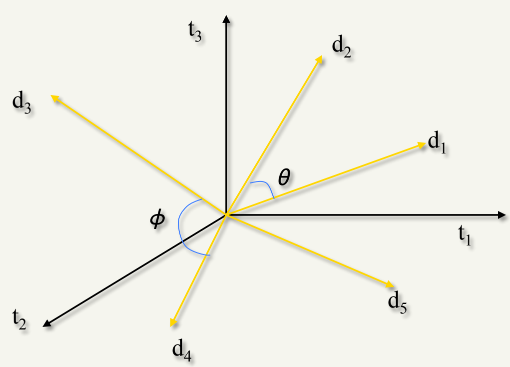
- Each doc *j* can now be viewed as a vector of *tf×idf* values, one component for each term
- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 200,000+ dimensions

9/6/11 28

Why turn docs into vectors?

- First application: Query-by-example
  - Given a doc D, find others “like” it.
- Now that D is a vector, find vectors (docs) “near” it.

Intuition



Hypothesis: Documents that are “close together” in the vector space talk about the same things.

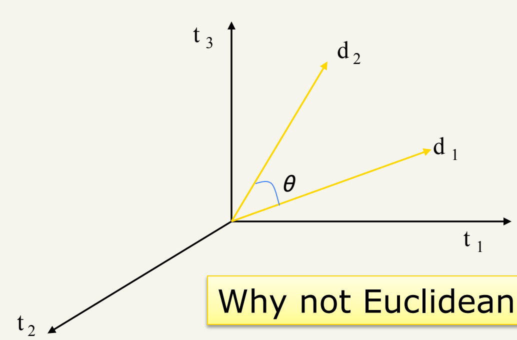
The Vector Space Model

Queries are just short documents

- Take the freetext query as short document
- Return the documents ranked by the closeness of their vectors to the query vector.

Cosine Similarity

Similarity between vectors d_1 and d_2 *captured* by the cosine of the angle x between them.



Why not Euclidean distance?

Cosine similarity

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{M} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{M} w_{i,j}^2}\sqrt{\sum_{i=1}^{M} w_{i,k}^2}}$$

- Cosine of angle between two vectors
  - The denominator involves the lengths of the vectors.

Normalization

Normalized vectors

- For normalized vectors, the cosine is simply the dot product:

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

So...

- Basic ranked retrieval scheme is to
  - Treat queries as vectors
  - Compute the dot-product of the query with *all* the docs
  - Return the ranked list of docs for that query.

But...

- What do we know about the document vectors?
- What do we know about query vectors?

Scoring

(1) N documents. Each gets a score.

(2) Get the lengths for later use

(3) Iterate over the query terms

(6) Accumulate the scores for a doc, a term at a time

(9) Normalize by doc vector length

CosineScore(q)

1 float $Scores[N] = 0$

2 Initialize $Length[N]$

3 **for each** query term t

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

5 **for each** pair(d, $tf_{t,d}$) in postings list

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

7 Read the array $Length[d]$

8 **for each** d

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

10 **return** Top K components of $Scores[]$

Next Time

Should have read up through Chapter 6.

Move on to 7.