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.
    - Office hours have changed (we now have 3 TAs instead of 4)

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. There are still folks who turned in only a CMS copy of the homework – the CMS copy did not get graded (they should see a TA to turn in a hardcopy).
- Homework 2: Part of it involves downloading and running the Lucene IR system (but no programming). Note that "computer problems" will not excuse late submissions! There are computer labs available in Gates for both undergrads and masters students if you need a machine. Due Thursday 10/2 at beginning of class (hardcopy plus CMS).

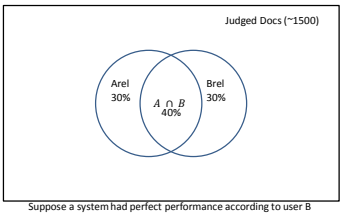
Brief Look at Homework 1 Question

- A proposed project has the stated goal of improving the effectiveness of a retrieval system for general use to 90% recall and 90% precision.
  - (1) [5 points] What would be the F measure of such a system?
  - (2) [5 points] Do you think this goal can be achieved? Why or why not?
- Delicate real-life question: How do you prepare a grant proposal to a funding agency that wants that result?

Information Retrieval Users

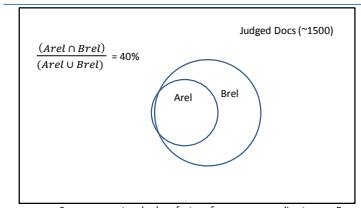
- Users are central!
  - The goal of any IR search is to satisfy the information needs of the end user
- But end users
  - Do not agree with each other as to what they want
  - Do not even agree with themselves (over a period of time)
- Precision and Recall are end user numbers
  - Dependent on a particular user notion of relevance
  - Are not purely objective measures

Judgment Agreement (Average)



Suppose a system had perfect performance according to user B
(retrieves exactly the documents B thought relevant)
What is the recall and precision according to user A?

Judgment Agreement (Normal single topic)



Suppose a system had perfect performance according to user B
What is the recall and precision according to user A?

System Upper-bounds

- Expected upper bound is about 75% precision at 75% recall
  - For general case, with random queries and users
- Particular applications might be higher
  - Navigational queries: goal is to go to one particular page

Previous lectures

- Overview
- Evaluation 1
- Indexing
  - Text acquisition
  - Text transformation
  - Index construction
    - Dictionary
- Retrieval
  - Boolean model

Boolean retrieval

- Queries define a set of documents to be retrieved
  - Use Boolean operatives (AND, OR, AND NOT)
  - Use proximity, wild-cards, location info

User query:
Are there any cases which discuss negligent maintenance or failure to maintain aids to navigation such as lights, buoys, or channel markers?

Intermediary query:
NEGLECT! FAIL! NEGLIG! /5 MAINT! REPAIR! /P NAVIGAT! /5 AID EQUIP! LIGHT BUOY "CHANNEL MARKER"

Boolean Retrieval

- Advantages
  - User has complete control over returned set
  - Supports a complex query language, many different features
  - Results are predictable, relatively easy to explain
  - Efficient processing since many documents can be eliminated from search
- Disadvantages
  - User has complete control over returned set (unranked set)
    - Effectiveness depends entirely on user
  - Supports a complex query language, many different features
  - Simple queries usually don't work well – searching "by numbers"
  - Complex queries are difficult to form
  - Complex queries can be difficult to process efficiently

Today's Lecture (following Manning slides rather than Croft)

- Tf*idf Weighting
- Vector Space Model

Ranked Retrieval Models

- Rather than returning a set of documents, return an ordering of documents to the user – the top documents are ranked
- Rather than a complex query language, use natural language queries
  - Generally true, but can have ranking with Boolean type query languages, or can retrieve sets with natural language queries.

Ranked Retrieval Models

- Advantages
  - Result set size is not an issue
    - User will look at however many documents they want
  - Simple query language
- Disadvantages
  - User must trust that the ranking algorithm works (mostly)

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in [0, 1] – to each document
- This score measures how well document and query "match".
  - Equivalently, how similar the document and query are.

Query-document matching scores

- We need a way of assigning a score to a query/document pair
- Let's start with a one-term query
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard coefficient

- Seen previously as a measure of overlap of two sets *A* and *B*
- jaccard*(A,B)* = $|A \cap B| / |A \cup B|$
- jaccard*(A,A)* = 1
- jaccard*(A,B)* = 0 if $A \cap B = 0$
- *A* and *B* don't have to be the same size.
- Always assigns a number between 0 and 1.
- So nice mathematical properties

Jaccard coefficient: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: *ides of march*
- Document 1: *caesar died in march*
- Document 2: *the long march*

Issues with Jaccard for scoring

- It doesn't consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length

Binary term-document incidence matrix

	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

- Each document is represented by a binary vector

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Each document is a count vector in $\mathbb{N}^{v}$: a column below

	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

Term frequency tf

- The term frequency $tf_{t,d}$ 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

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

- 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 $= \sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$
- The score is 0 if none of the query terms is present in the document.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
- Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)
- A document containing this term is very likely to be relevant to the query *arachnocentric*
- → We want a high weight for rare terms like *arachnocentric*.

Document frequency, continued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- 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 *high, increase, and line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df_t is the document frequency of *t*: the number of documents that contain *t*
  - df_t is an inverse measure of the informativeness of *t*
  - df_t will tend to grow as *N* grows
- We define the idf (inverse document frequency) of *t* by

$$idf_t = \log_{10}(N/df_t)$$

  - We use *log* (N/df_t) instead of N/df_t to "dampen" the effect of idf.

idf example, suppose *N* = 1 million

term	df_t	idf_t
calpurnia	1	
animal	100	
sunday	1,000	
fly	10,000	
under	100,000	
the	1,000,000	

$$idf_t = \log_{10}(N/df_t)$$

- There is one idf value for each term *t* in a collection

Collection vs. Document frequency

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is a better search term (and should get a higher weight)?
- Lucene uses Collection Frequency (ease of indexing)

Effect of idf on ranking

- idf has no effect on ranking one term queries like "iPhone"
  - At least not directly.
- idf affects the ranking of documents for queries with at least two terms
  - For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.
- idf factors tell how important a term is in general (or in this collection)
  - tf factors are an indication how important a term is within a document

Tf*idf weighting

- The tf*idf weight of a term is the product of its tf weight and its idf weight.

$$w_{t,d} = \log(1+tf_{t,d}) \times \log_{10}(N/df_t)$$

- The standard (best known) weighting scheme in information retrieval
  - Alternative names: tf.idf, tf x idf, tf-idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Score for a document given a query

$$Score(q,d) = \sum_{t \in q \cap d} tf.idf_{t,d}$$

- There are many variants
  - How "tf" is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - …

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

- Each document is now represented by a real-valued vector of tf*idf weights $\in \mathbb{R}^{|V|}$

Documents as vectors

- So we have a $|V|$-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.

Queries as vectors

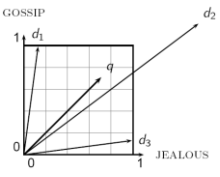
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the you're-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

Formalizing vector space proximity

- First cut: distance between two points
  - (= distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance between q and d_2 is large even though the distribution of terms in the query q and the distribution of terms in the document d_2 are very similar.



Use angle instead of distance

- Thought experiment: take a document d and append it to itself. Call this document d'.
- "Semantically" d and d' have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.

- Key idea: Rank documents according to angle with query.

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°, 180°]

cosine(query,document)

Dot product

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

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

- cos(q,d) is the cosine similarity of q and d … or,
  - equivalently, the cosine of the angle between q and d.

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

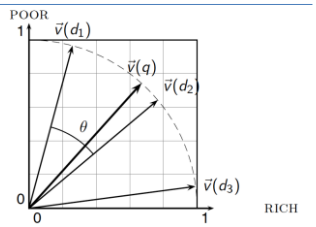
Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

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

for q, d length-normalized.

Cosine similarity illustrated



Cosine similarity amongst 3 documents

How similar are the novels SaS: *Sense and Sensibility* PaP: *Pride and Prejudice*, and WH: *Wuthering Heights*?

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

Term frequencies (counts) (no idf)

3 documents example

Log frequency weighting

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

After length normalization

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

cos(SaS,PaP) ≈
$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0$
≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

Why do we have cos(SaS,PaP) > cos(SaS,WH)?

Computing cosine scores

```
CosineScore(q)
float Scores[N] = 0
float Length[N]
for each query term t
do calculate w_{t,q} and fetch postings list for t
    for each pair(d, tf_{t,d}) in postings list
    do Scores[d] += w_{t,d} × w_{t,q}
Read the array Length
for each d
do Scores[d] = Scores[d]/Length[d]
return Top K components of Scores[]
```

Tf*idf weighting has many variants

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)	$1/u$
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)	$\frac{1 + \log(\mathrm{tf}_{t,d})}{1 + \log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

- Columns headed 'n' are acronyms for weight schemes.

Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table
- A very standard weighting scheme is: lnc.ltc
- Document: logarithmic tf (l as first character), no idf and cosine normalization
  - idf in documents presents problems in dynamic collections
- Query: logarithmic tf (l in leftmost column), idf (t in second column), cosine normalization …

Tf*idf example: lnc.ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

Term	Query						Document				Prod
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

Doc length = $\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$

Score = 0+0+0.27+0.53 = 0.8

Tf*idf performance across collections

Table 4. Performance results for eight term-weighting methods averaged over 5 collections

Term-weighting methods	Rank of method and ave. precision	CACM 3204 docs 64 queries	CISI 1460 docs 112 queries	CRAN 1397 docs 225 queries	INSPEC 12,684 docs 84 queries	MED 1033 docs 30 queries	Averages for 5 collections
1. Best fully weighted (tfc·nfx)	Rank	1	14	19	3	19	11.2
	P	0.3630	0.2189	0.3841	0.2626	0.5628	
2. Weighted with inverse frequency f not used for docs (txc·nfx)	Rank	25	14	7	4	32	16.4
	P	0.3252	0.2189	0.3950	0.2626	0.5542	
3. Classical tf × idf No normalization (tfx·tfx)	Rank	29	22	219	45	132	84.4
	P	0.3248	0.2166	0.2991	0.2365	0.5177	
4. Best weighted probabilistic (nxx·bpx)	Rank	55	208	11	97	60	86.2
	P	0.3090	0.1441	0.3899	0.2093	0.5449	
5. Classical idf without normalization (bfx·bfx)	Rank	143	247	183	160	178	182
	P	0.2535	0.1410	0.3184	0.1781	0.5062	
6. Binary independence probabilistic (bxx·bpx)	Rank	166	262	154	195	147	159
	P	0.2376	0.1233	0.3266	0.1563	0.5116	
7. Standard weights cosine normalization (original Smart) (txc·txc)	Rank	178	173	137	187	246	184
	P	0.2102	0.1539	0.3408	0.1620	0.4641	
8. Coordination level binary vectors (bxx·bxx)	Rank	196	284	280	258	281	260
	P	0.1848	0.1033	0.2414	0.0944	0.4132	

- Performance differs across collections IPM 1988 Salton and Buckley

Summary – vector space ranking

- Represent the query as a weighted tf*idf vector
- Represent each document as a weighted tf*idf vector
- Compute the cosine similarity score for the query vector and each document vector
- Rank documents with respect to the query by score
- Return the top *K* (e.g., *K* = 10) to the user

Vector Space Model

- Advantages
  - Simple computation for ranking purposes
  - Empirically works well with tf*idf weights
  - Intuitively appealing
    - We'll see several examples later (feedback, clustering)
  - No model imposed notion of weights
    - Can be used with application dependent weights
- Disadvantages
  - No model imposed notion of weights
    - Weighting crucial, but model gives no insights
  - Basic model assumes term independence