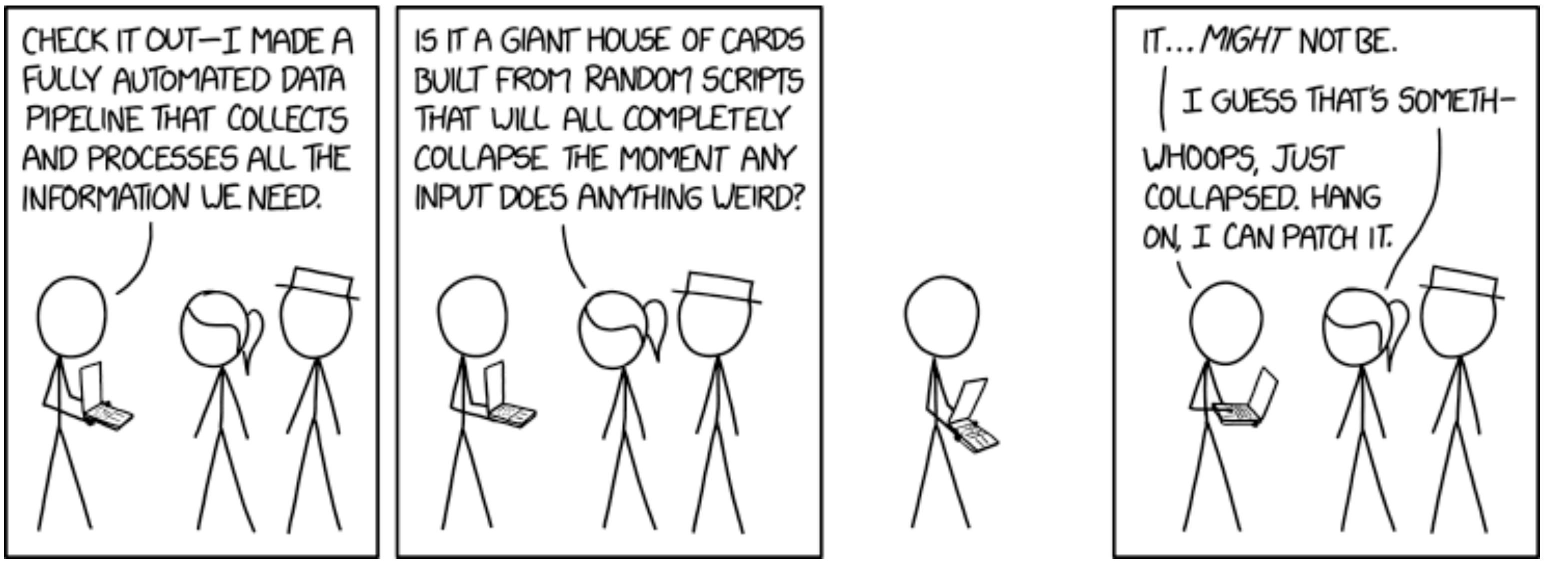


xkcd.com/2054/

Text Mining and Information Retrieval: Part II

Laura Brown

Some slides adapted from P. Smyth; Han, Kamber, & Pei;
Tan, Steinbach, & Kumar; C. Volinsky; R. Tibshirani; D. Kauchak
and http://nlp.stanford.edu/IR-book/

Outline

- Ranked Retrieval
- Scoring documents
  - Jaccard coefficient
  - Bag-of-Words Model
- Vector space scoring

Ranked Retrieval

Introduction to Information Retrieval

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don't.
- Good for expert users with precise understanding of their needs and the collection.
  - Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
  - Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
  - Most users don't want to wade through 1000s of results.
    - This is particularly true of web search.

Ranked retrieval models

- In ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top *k* (≈ 10) results
  - We don’t overwhelm the user
- Premise: the ranking algorithm works

Scoring documents

Introduction to Information Retrieval

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".

Take 1: Jaccard coefficient

- A common measure of overlap of two sets *A* and *B*

$$\text{jaccard}(A,B) = |A \cap B| \;/\; |A \cup B|$$

$$\text{jaccard}(A,A) = 1$$

$$\text{jaccard}(A,B) = 0 \text{ if } A \cap B = 0$$

- *A* and *B* don't have to be the same size.
- Always assigns a number between 0 and 1.

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*

$$\text{jaccard}(A,B) = |A \cap B| \,/\, |A \cup B|$$

jaccard(Q, doc1) = |{ march} | / | { ides of march Caesar died in } = 1/6

jaccard(Q, doc2) = |{ march} | / | { ides of march the long } = 1/5

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 often more informative than frequent terms. Jaccard doesn't consider this information
- We need a more sophisticated way of normalizing for length

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.

Bag-of-Words Model

Introduction to Information Retrieval

Recall: 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 $\in \{0,1\}^{|V|}$

Term-document count matrix

- 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

Note, that the term-document matrix, can be transposed for a document-term matrix; this format more closely matches the typical data form of [samples x variables]

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 bag of words model.

Term frequency tf

- The term frequency $\mathrm{tf}_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
  - Note: Frequency means count in IR
- 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

$$w_{t,d} = \begin{cases} 1 + \log_{10} tf_{td} & if \;\; tf_{t,d} > 0 \\ 0 & otherwise \end{cases}$$

- 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*:

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

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

Rare terms are more informative

- 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*.

Bag-of-Words Model, continued

Introduction to Information Retrieval

idf weight

- df_t is the <u>document</u> frequency of t: the number of documents that contain t
  - df_t is an inverse measure of the informativeness of t
  - $\mathrm{df}_t \leq N$
- We define the idf (inverse document frequency) of t by

$$idf_t = \log_{10} \frac{N}{df_t}$$

  - 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	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$idf_t = \log_{10} \frac{N}{df_t}$$

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

tf-idf weighting

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

$$w_{t,d} = (\log\left(tf_{t,d}\right) + 1) * \log_{10}(\frac{N}{df_t})$$

- Best known weighting scheme in information retrieval
  - Note: the “-” in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x 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 $\in \mathbb{R}^{|V|}$

Vector Representation

Introduction to Information Retrieval

Documents as vectors

- We have a |V|-dimensional vector space

  Terms are axes of the space

- Documents are points or vectors in this space
  - This is similar to how we discussed data in classification problems
- 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

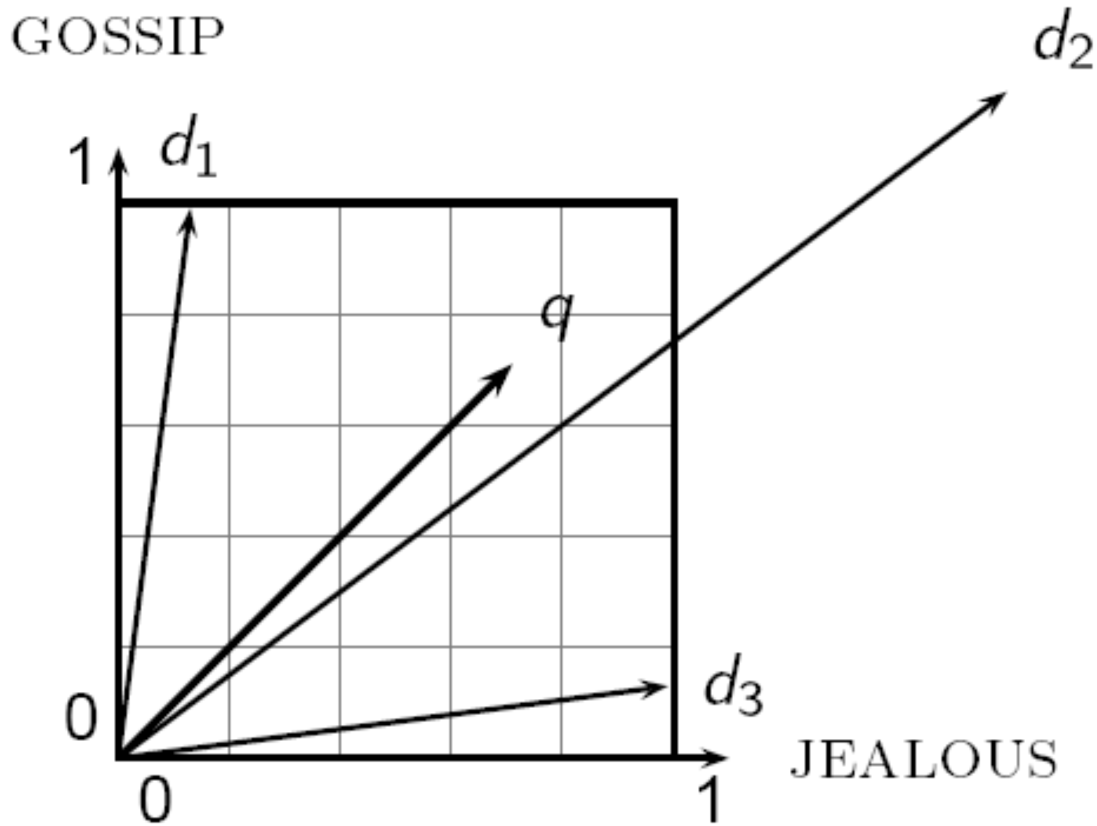
Formalizing vector space proximity

- First idea: 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.



Key idea: Rank documents according to angle with query.

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)
    - Long and short documents now have comparable weights

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 $\vec{d}$.

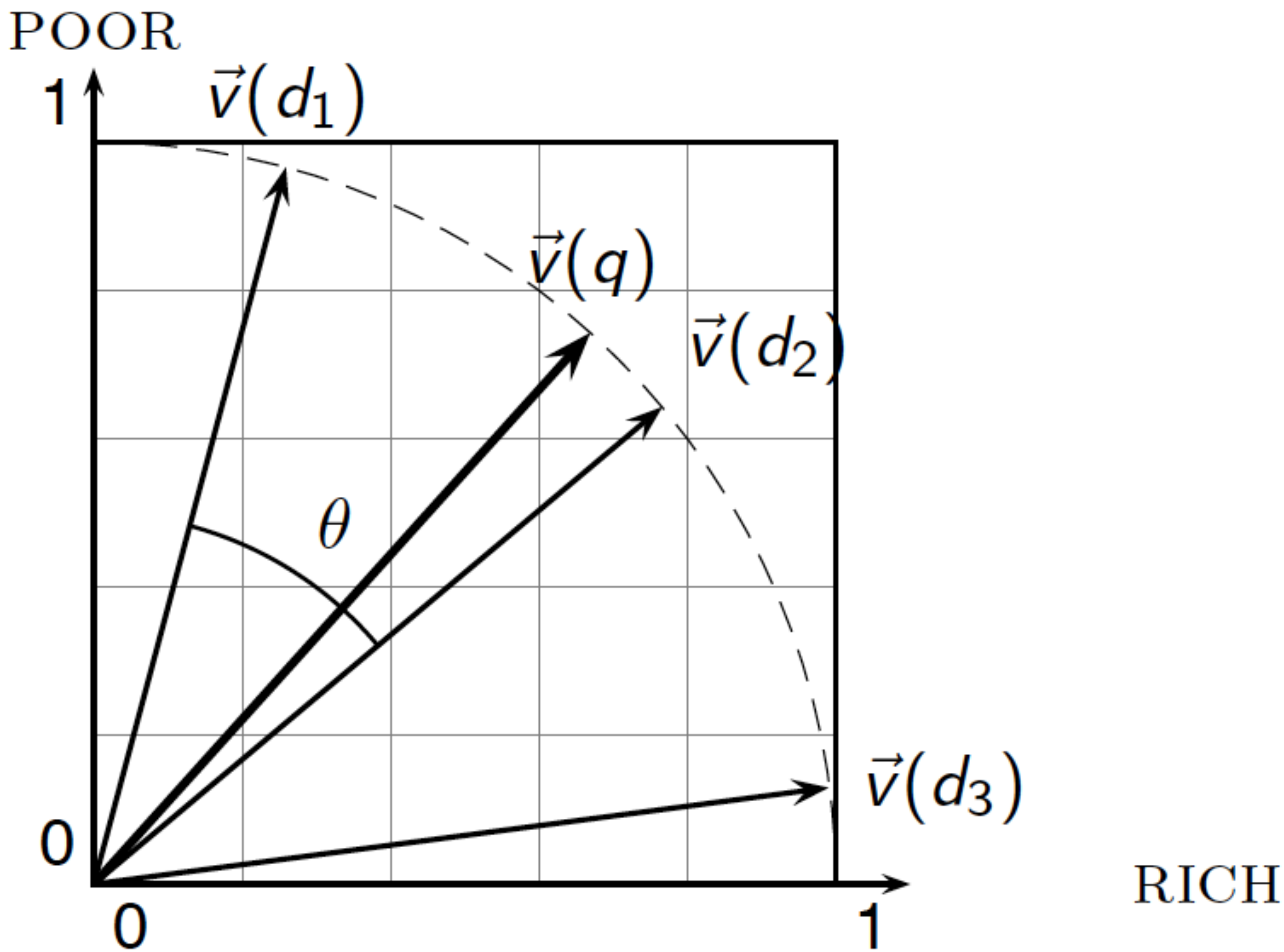
cosine for length-normalized vectors

- 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



Example: Vector Representation

Introduction to Information Retrieval

Example: cosine similarity with 3 docs

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)

Note: To simplify this example, we don't do idf weighting.

3 documents example contd.

- 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) ≈ 0.94
cos(SaS,WH) ≈ 0.79
cos(PaP,WH) ≈ 0.69

tf-idf 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)	$1/\mathit{CharLength}^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t\in d}(\mathrm{tf}_{t,d}))}$				

Weighting may differ in queries vs. docs

- 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
- 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

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

Evaluation

Introduction to Information Retrieval

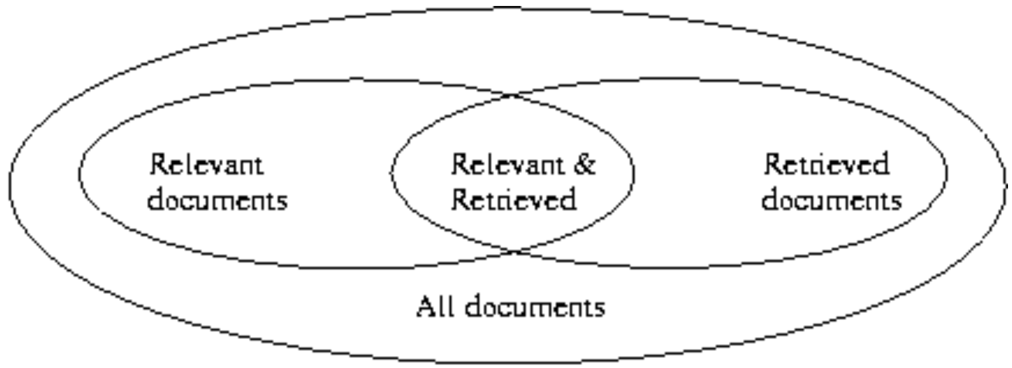
Evaluation

You have developed an IR system.

How do you tell if users are happy?

- Search returns products relevant to users
  - How do you assess this at scale?
- Search results get clicked a lot
  - Misleading information can cause a user to click
- Users buy after using the search engine
- Repeat visitors/buyers
- …
- Most common proxy for user happiness?
  - Relevance of search results

Evaluating Text Retrieval: Unranked



- Precision: the percentage of retrieved documents that are in fact relevant to the query (i.e., "correct" responses)

$$precision = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Retrieved\}|}$$

- Recall: the percentage of documents that are relevant to the query and were retrieved

$$recall = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Relevant\}|}$$

Precision vs. Recall

	Truth:Relevant	Truth:Not Relevant
Algorithm:Relevant	TP	FP
Algorithm: Not Relevant	FN	TN

We've been here before!

- Precision = TP / (TP+FP)
- Recall = TP / (TP+FN)
- Trade off:
  - If algorithm is “picky”: precision high, recall low
  - If algorithm is “relaxed”: precision low, recall high
- BUT: recall often hard if not impossible to calculate

Evaluation: Rank-based Measures

Introduction to Information Retrieval

Rank-based Measures

- Binary relevance
  - Precision@K (P@K)
  - Recall@K (R@K)
  - Mean Average Precision (MAP)

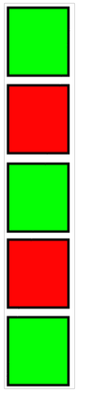
- Multiple levels of relevance
  - Normalized Discounted Cumulative Gain (NDCG)

Precision@K

- Set a rank threshold K
- Compute % relevant in top K

$$precision = \frac{|\{Relevant\} \cap \{Retrieved\}|}{|\{Retrieved\}|}$$

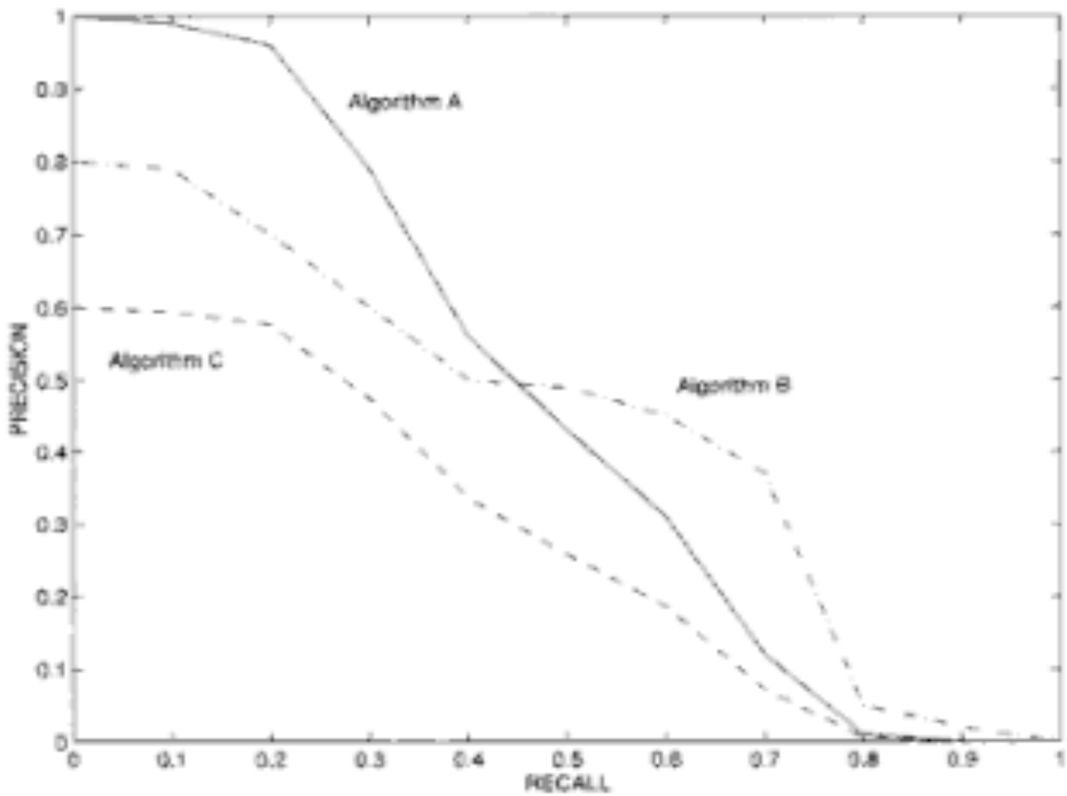
- Ignores documents ranked lower than K
- Ex:
  - Prec@3 of 2/3
  - Prec@4 of 2/4
  - Prec@5 of 3/5



- In similar fashion we have Recall@K

Precision Recall Curve

- At different thresholds, we can plot a point for precision vs. recall

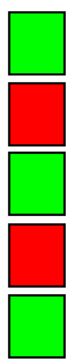


Mean Average Precision

- Consider rank position of each ***relevant*** doc
  - K_1, K_2, … K_R

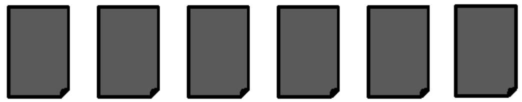
- Compute Precision@K for each K_1, K_2, … K_R

- Average precision = average of P@K

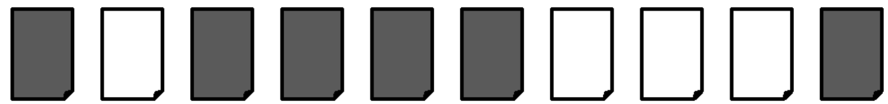
- Ex:  has AvgPrec of $\frac{1}{3} \cdot \left(\frac{1}{1} + \frac{2}{3} + \frac{3}{5} \right) \approx 0.76$

- MAP is Average Precision across multiple queries/rankings

Example: Average Precision

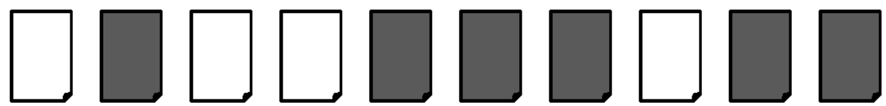

= the relevant documents

Ranking #1



Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

Ranking #2

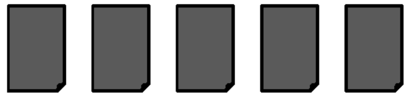


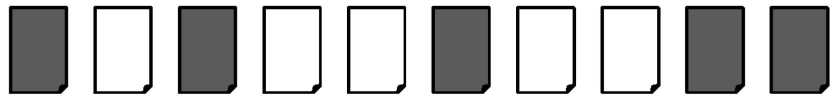
Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67	0.67	0.83	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

$$\text{Ranking \#1: } (1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

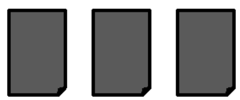
$$\text{Ranking \#2: } (0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$$

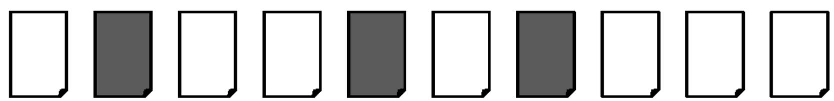
Example: MAP

 = relevant documents for query 1

Ranking #1 

Recall	0.2	0.2	0.4	0.4	0.4	0.6	0.6	0.6	0.8	1.0
Precision	1.0	0.5	0.67	0.5	0.4	0.5	0.43	0.38	0.44	0.5

 = relevant documents for query 2

Ranking #2 

Recall	0.0	0.33	0.33	0.33	0.67	0.67	1.0	1.0	1.0	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.33	0.43	0.38	0.33	0.3

$$\textit{average precision query 1} = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$
$$\textit{average precision query 2} = (0.5 + 0.4 + 0.43)/3 = 0.44$$

$$\textit{mean average precision} = (0.62 + 0.44)/2 = 0.53$$

Other Measures

- Discounted Cumulative Gain
- User Behavior
  - User Clicks
  - User Clicks sequence
- Eye-tracking User Study
  - Higher positions receive more user attention (eye fixation) and clicks than lower positions.
- A/B testing of web search