

Introduction to **Information Retrieval**

Introducing ranked 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.

Problem with Boolean search: feast or famine

- Boolean queries often result in either too few (≈ 0) or too many (1000s) results.
 - Query 1: “*standard user dlink 650*” \rightarrow 200,000 hits
 - Query 2: “*standard user dlink 650 no card found*” \rightarrow 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
 - AND gives too few; OR gives too many

Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in **ranked retrieval models**, the system returns an ordering over the (top) documents in the collection with respect to 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 models have 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 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”.

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

Introduction to **Information Retrieval**

Introducing ranked retrieval

Introduction to **Information Retrieval**

Scoring with the Jaccard coefficient

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets A and B is the Jaccard coefficient
- $\text{jaccard}(A, B) = |A \cap B| / |A \cup B|$
- $\text{jaccard}(A, A) = 1$
- $\text{jaccard}(A, B) = 0$ if $A \cap B = 0$
- A and B don't have to be the same size.
- Always assigns a number between 0 and 1.

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
 - Later in this lecture, we'll use $|A \cap B| / \sqrt{|A \cup B|}$... instead of $|A \cap B| / |A \cup B|$ (Jaccard) for length normalization.

Introduction to **Information Retrieval**

Scoring with the Jaccard coefficient

Introduction to **Information Retrieval**

Term frequency weighting

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

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-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

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.
- In a sense, this is a step back: The positional index was able to distinguish these two documents
 - We will look at “recovering” positional information later on
 - For now: bag of words model

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.

NB: frequency = count in IR

Log-frequency weighting

- The log frequency weight of term t in d is

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

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

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

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

Log-frequency weighting

- The log frequency weight of term t in d is

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

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d :
- $$\text{score} = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$
- The score is 0 if none of the query terms is present in the document.

Introduction to **Information Retrieval**

Term frequency weighting

Introduction to **Information Retrieval**

(Inverse) Document frequency weighting

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 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
 - $df_t \leq N$
- 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.

Will turn out the base of the log is immaterial.

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.

Effect of idf on ranking

- Question: Does idf have an effect on ranking for one-term queries, like
 - iPhone

Effect of idf on ranking

- Question: Does idf have an effect on ranking for one-term queries, like
 - iPhone
- idf has no effect on ranking one term queries
 - 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**.

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
<i>insurance</i>	10440	3997
<i>try</i>	10422	8760

- Which word is a better search term (and should get a higher weight)?

Introduction to **Information Retrieval**

(Inverse) Document frequency weighting

Introduction to **Information Retrieval**

tf-idf weighting

tf-idf weighting

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

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

- **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**

Final ranking of documents for a query

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

Binary \rightarrow count \rightarrow 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 $\in \mathbb{R}^{|V|}$

Introduction to **Information Retrieval**

tf-idf weighting

Introduction to **Information Retrieval**

The Vector Space Model (VSM)

Documents as vectors

- Now 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

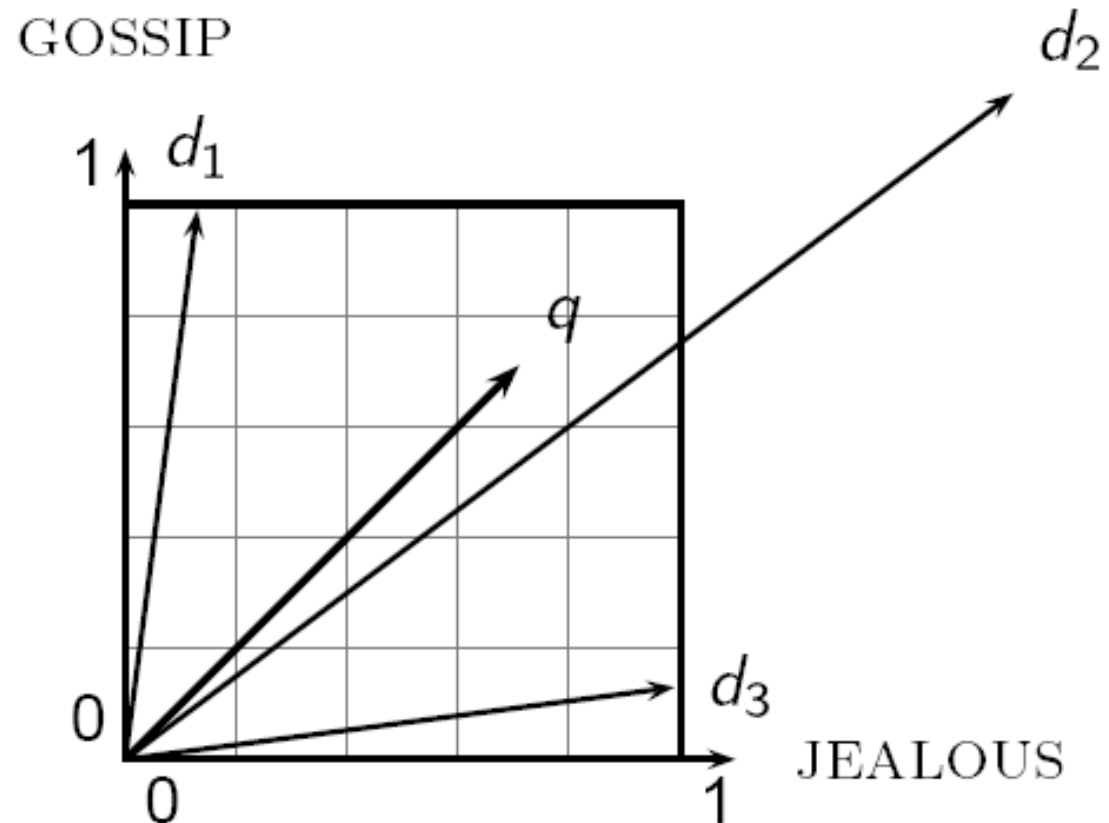
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \approx inverse of distance
- **Recall: We do this because we want to get away from the you're-either-in-or-out 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 \vec{d}_2 is large even though the distribution of terms in the query \vec{q} and the distribution of terms in the document \vec{d}_2 are very similar.



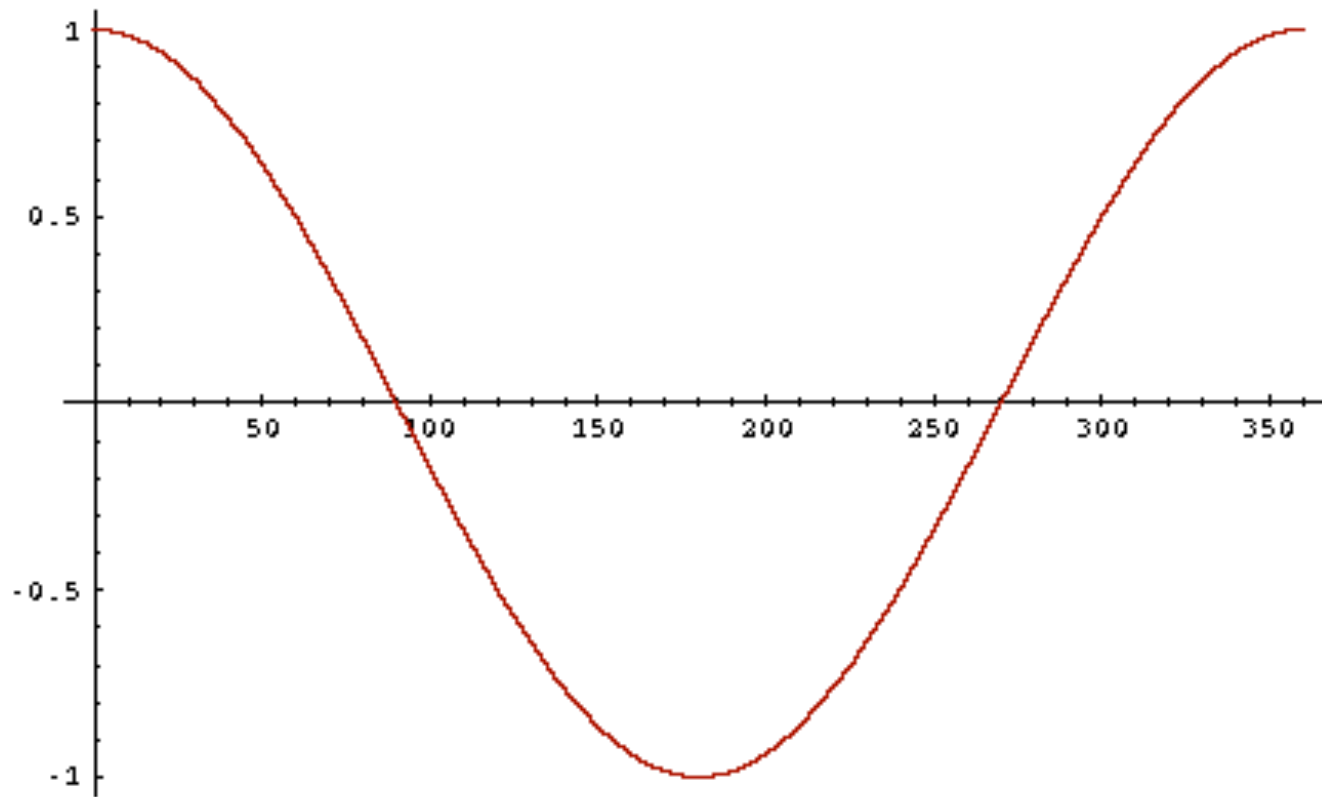
Use angle instead of distance

- 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 $\text{cosine}(\text{query}, \text{document})$
- Cosine is a monotonically decreasing function for the interval $[0^\circ, 180^\circ]$

From angles to cosines



- But how – *and why* – should we be computing cosines?

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(query,document)

Dot product

Unit vectors

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

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

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

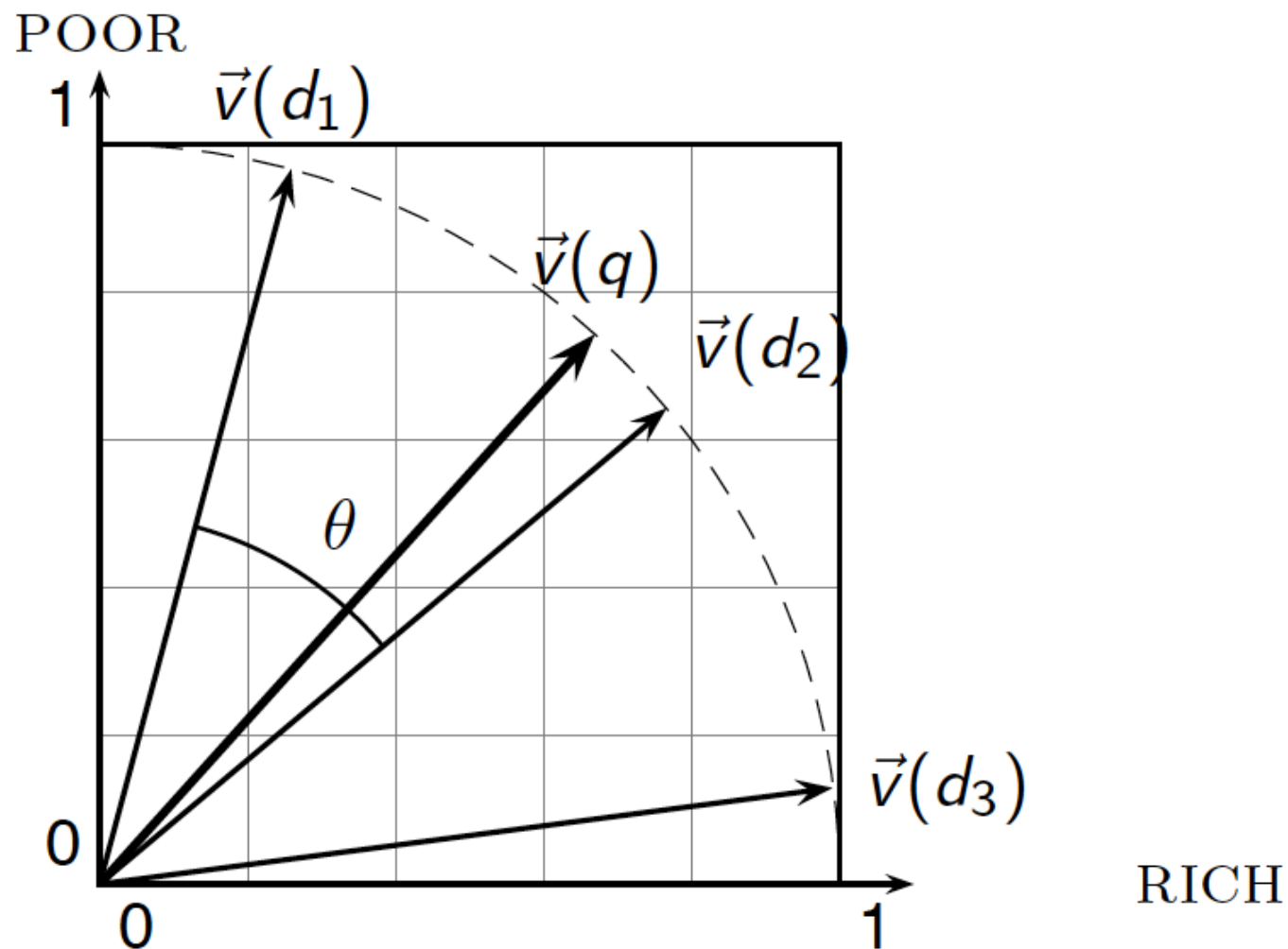
Cosine for length-normalized vectors

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

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

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)

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

$$\cos(\text{SaS}, \text{PaP}) \approx$$

$$0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx \mathbf{0.94}$$

$$\cos(\text{SaS}, \text{WH}) \approx \mathbf{0.79}$$

$$\cos(\text{PaP}, \text{WH}) \approx \mathbf{0.69}$$

Why do we have $\cos(\text{SaS}, \text{PaP}) > \cos(\text{SAS}, \text{WH})$?

Introduction to **Information Retrieval**

The Vector Space Model (VSM)

Introduction to **Information Retrieval**

Calculating tf-idf cosine scores
in an IR system

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$tf_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^\alpha, \alpha < 1$
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$				

Columns headed ‘n’ are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

tf-idf weighting has many variants

Term frequency		Document frequency		Normalization	
n (natural)	$tf_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^\alpha, \alpha < 1$
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$				

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: Inc.Itc
- 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 ...



A bad idea?

tf-idf example: Inc.Itc

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

Exercise: what is N , the number of docs?

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

$$\text{Score} = 0 + 0 + 0.27 + 0.53 = 0.8$$

Computing cosine scores

COSINESCORE(q)

```
1  float Scores[ $N$ ] = 0
2  float 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[ $d$ ] + =  $w_{t,d} \times w_{t,q}$ 
7  Read the array Length
8  for each  $d$ 
9  do Scores[ $d$ ] = Scores[ $d$ ] / Length[ $d$ ]
10 return Top  $K$  components of Scores[]
```

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

Introduction to **Information Retrieval**

Calculating tf-idf cosine scores
in an IR system

Introduction to **Information Retrieval**

Using many features to determine
relevance

Integrating multiple features to determine relevance

- Modern systems – especially on the Web – use a great number of features:
 - Arbitrary useful features – not a single unified model
 - Log frequency of query word in anchor text?
 - Query word in color on page?
 - # of images on page?
 - # of (out) links on page?
 - PageRank of page?
 - URL length?
 - URL contains “~”?
 - Page edit recency?
 - Page length?
- The *New York Times* (2008-06-03) quoted Amit Singhal as saying Google was using over 200 such features.

How to combine features to assign a relevance score to a document?

- Given lots of relevant features...
- You can continue to hand-engineer retrieval scores
- Or, you can build a classifier to learn weights for the features
 - Requires: labeled training data
 - This is the “learning to rank” approach, which has become a hot area in recent years
 - I only provide an elementary introduction here

Simple example:

Using classification for ad hoc IR

- Collect a training corpus of (q, d, r) triples
 - Relevance r is here binary (but may be multiclass, with 3–7 values)
 - Document is represented by a feature vector
 - $\mathbf{x} = (\alpha, \omega)$ α is cosine similarity, ω is minimum query window size
 - ω is the the shortest text span that includes all query words
 - Query term proximity is a **very important** new weighting factor
 - Train a machine learning model to predict the class r of a document-query pair

example	docID	query	cosine score	ω	judgment
Φ_1	37	linux operating system	0.032	3	relevant
Φ_2	37	penguin logo	0.02	4	nonrelevant
Φ_3	238	operating system	0.043	2	relevant
Φ_4	238	runtime environment	0.004	2	nonrelevant
Φ_5	1741	kernel layer	0.022	3	relevant
Φ_6	2094	device driver	0.03	2	relevant
Φ_7	3191	device driver	0.027	5	nonrelevant

Simple example:

Using classification for ad hoc IR

- A linear score function is then:

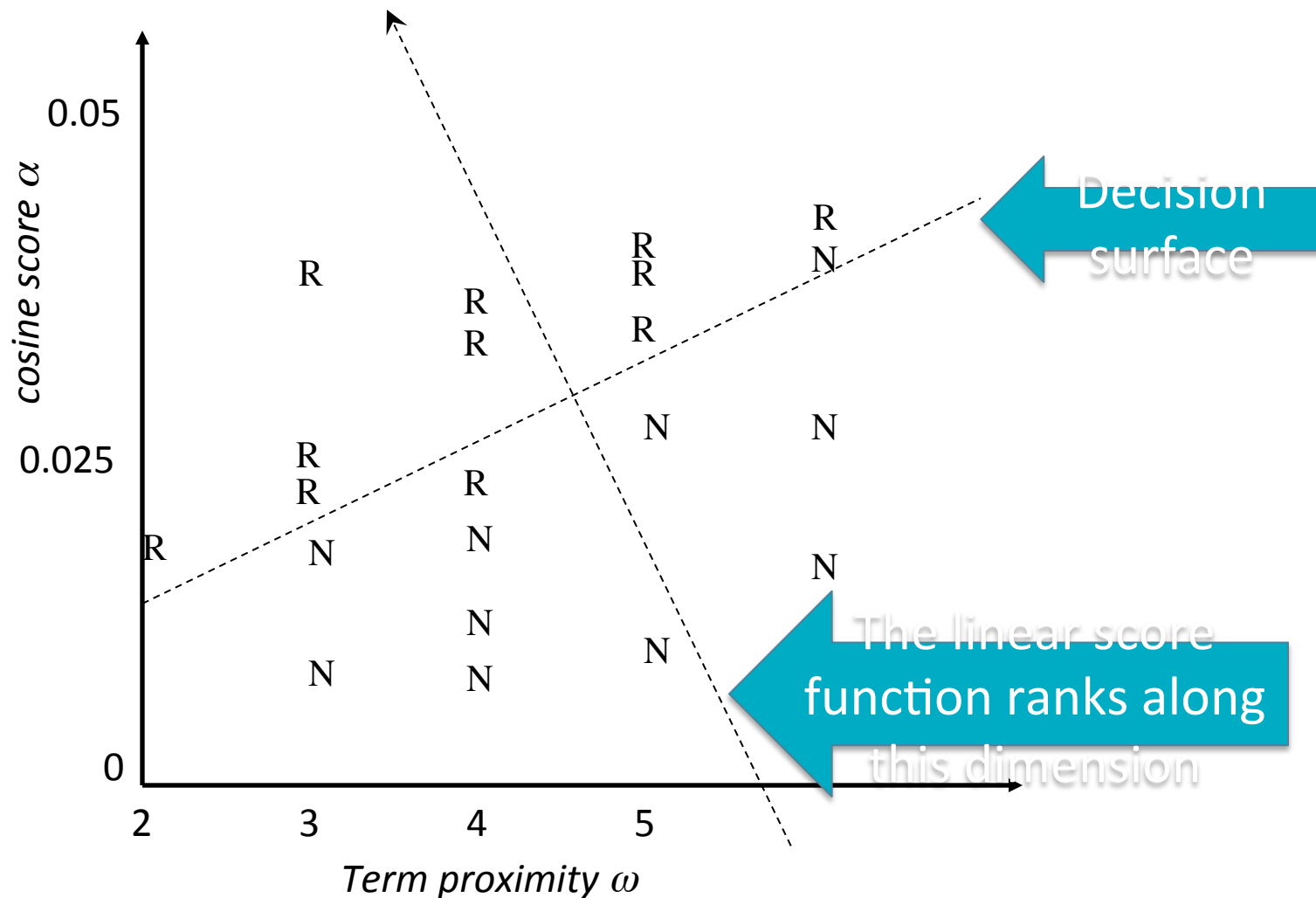
$$Score(d, q) = Score(\alpha, \omega) = a\alpha + b\omega + c$$

- And the linear classifier would be:

$$\text{Decide relevant if } Score(d, q) > \theta$$

- ... just like when we were doing text classification

Simple example: Using classification for ad hoc IR



Introduction to **Information Retrieval**

Using many features to determine
relevance

Introduction to **Information Retrieval**

Evaluating search engines

Measures for a search engine

- How fast does it index
 - Number of documents/hour
 - (Average document size)
- How fast does it search
 - Latency as a function of index size
- Expressiveness of query language
 - Ability to express complex information needs
 - Speed on complex queries
- Uncluttered UI
- Is it free?

Measures for a search engine

- All of the preceding criteria are *measurable*: we can quantify speed/size
 - we can make expressiveness precise
- The key measure: user happiness
 - What is this?
 - Speed of response/size of index are factors
 - But blindingly fast, useless answers won't make a user happy
- Need a way of quantifying user happiness with the results returned
 - Relevance of results to user's information need

Evaluating an IR system

- An **information need** is translated into a **query**
- Relevance is assessed relative to the **information need** *not* the **query**
- E.g., Information need: *I'm looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.*
- Query: **wine red white heart attack effective**
- You evaluate whether the doc addresses the information need, not whether it has these words

Evaluating ranked results

- Evaluation of a result set:
 - If we have
 - a benchmark document collection
 - a benchmark set of queries
 - assessor judgments of whether documents are relevant to queries

Then we can use Precision/Recall/F measure as before

- Evaluation of ranked results:
 - The system can return any number of results
 - By taking various numbers of the top returned documents (levels of recall), the evaluator can produce a *precision-recall curve*

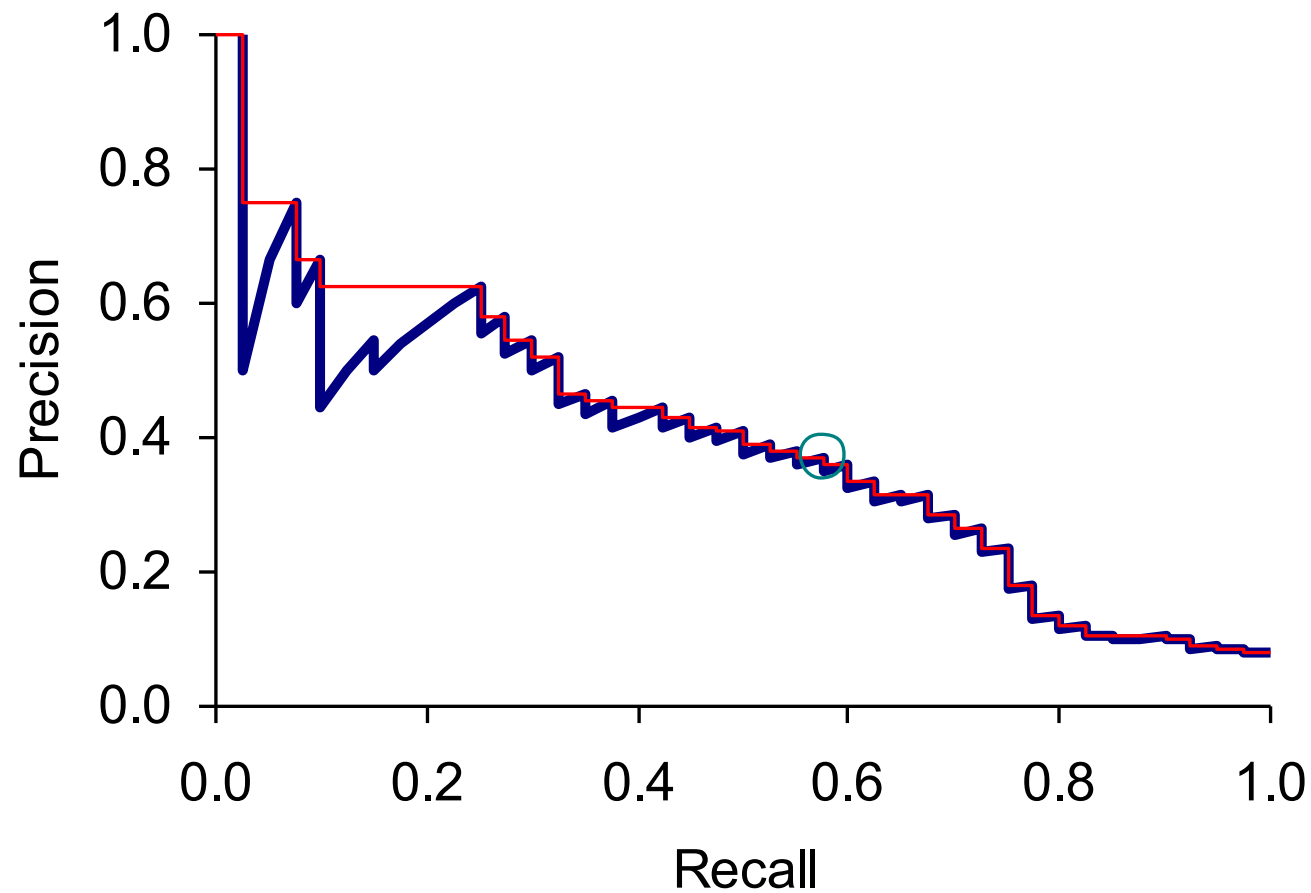
Recall/Precision

R P

- 1 R
- 2 N
- 3 N
- 4 R
- 5 R
- 6 N
- 7 R
- 8 N
- 9 N
- 10 N

Assume 10 rel docs
in collection

A precision-recall curve

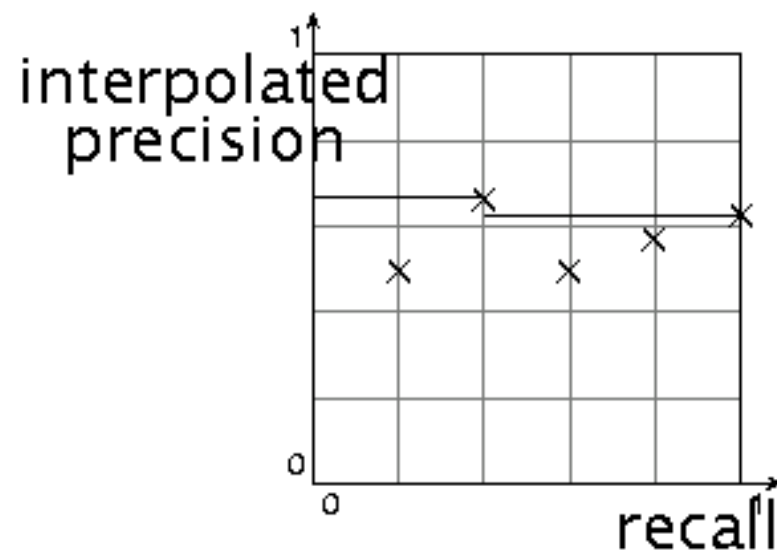
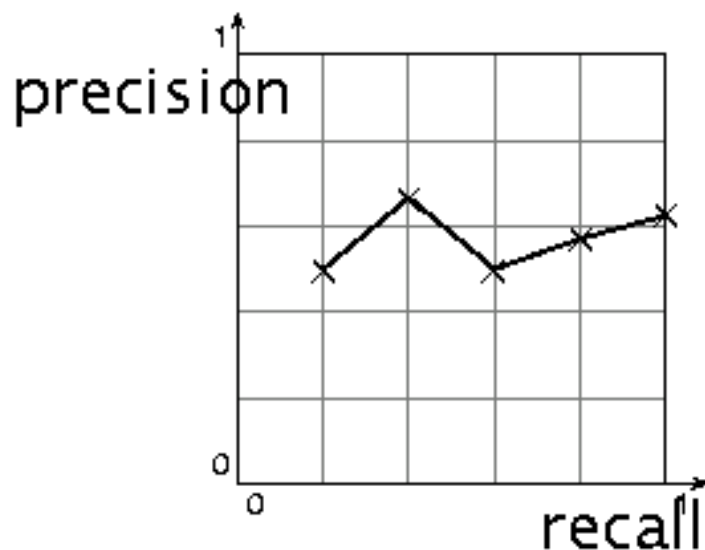


Averaging over queries

- A precision-recall graph for one query isn't a very sensible thing to look at
- You need to average performance over a whole bunch of queries.
- But there's a technical issue:
 - Precision-recall calculations place some points on the graph
 - How do you determine a value (interpolate) between the points?

Interpolated precision

- Idea: If locally precision increases with increasing recall, then you should get to count that...
- So you use the max of precisions to right of value

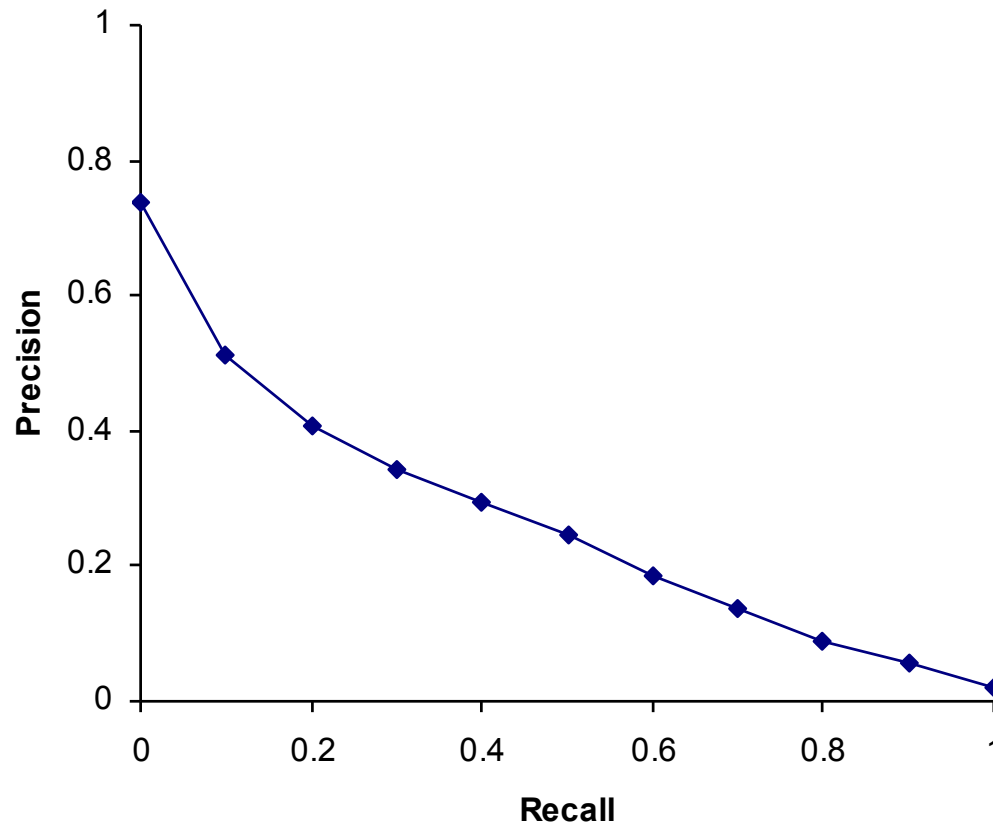


Evaluation

- Graphs are good, but people want summary measures!
 - Precision at fixed retrieval level
 - Precision-at- k : Precision of top k results
 - Perhaps appropriate for most of web search: all people want are good matches on the first one or two results pages
 - But: averages badly and has an arbitrary parameter of k
 - 11-point interpolated average precision
 - The standard measure in the early TREC competitions: you take the precision at 11 levels of recall varying from 0 to 1 by tenths of the documents, using interpolation (the value for 0 is always interpolated!), and average them
 - Evaluates performance at all recall levels

Typical (good) 11 point precisions

- SabIR/Cornell 8A1 11pt precision from TREC 8 (1999)



Two current evaluation measures...

- R-precision
 - If have known (though perhaps incomplete) set of relevant documents of size Rel , then calculate precision of top Rel docs returned
 - Perfect system could score 1.0.

Two current evaluation measures...

- Mean average precision (MAP)
 - AP: Average of the precision value obtained for the top k documents, each time a relevant doc is retrieved
 - Avoids interpolation, use of fixed recall levels
 - Does weight most accuracy of top returned results
 - MAP for set of queries is arithmetic average of APs
 - Macro-averaging: each query counts equally

Introduction to **Information Retrieval**

Evaluating search engines

Introduction to **Information Retrieval**

Web search

Brief (non-technical) history


- Early keyword-based engines ca. 1995–1997
 - Altavista, Excite, Infoseek, Inktomi, Lycos
 - Often not very good IR
- Paid search ranking: Goto (morphed into Overture.com → Yahoo!)
 - Your search ranking depended on how much you paid
 - Auction for keywords: **casino** was expensive!

Brief (non-technical) history

- 1998+: Link-based ranking pioneered by Google
 - Blew away all early engines save Inktomi
 - Great user experience in search of a business model
 - Meanwhile Goto/Overture's annual revenues were nearing \$1 billion
- Result: Google added paid search “ads” to the side, independent of search results
 - Yahoo followed suit, acquiring Overture (for paid placement) and Inktomi (for search)
- 2005+: Google gains search share, dominating in Europe and very strong in North America
 - Some strong regional players: Yandex (Russia), Baidu (China)
 - 2009: Yahoo! and Microsoft propose combined paid search offering

nigritude ultramarine - Google Search - Mozilla Firefox

File Edit View Go Bookmarks Yahoo! Tools Help




http://www.google.com/search?hl=en&q=nigritude+ultramarine&btnG=Google+Search

Go

Getting Started Latest Headlines

Y! Search Web Mail My Yahoo! Games Movies Music Answers Personals Sign In

pragh60@gmail.com | My Account | Sign out

 Web Images Groups News Froogle Local more »

nigritude ultramarine

Search

Advanced Search Preferences

Web

Results 1 - 10 of about 185,000 for **nigritude ultramarine**. (0.35 seconds)

Anil Dash: Nigritude Ultramarine
Do me a favor: Link to this post with the phrase **Nigritude Ultramarine**. ... Just placed a link to your **Nigritude Ultramarine** article on my weblog. Cheers! ...
www.dashes.com/anil/2004/06/04/nigritude_ultra - 101k - Mar 1, 2006 -
[Cached](#) - [Similar pages](#)

Nigritude Ultramarine FAQ
Nigritude Ultramarine FAQ - frequently asked questions about **nigritude ultramarine** and the realted SEO contest.
www.nigritudeultramaries.com/ - 59k - [Cached](#) - [Similar pages](#)

SEO contest - Wikipedia, the free encyclopedia
The **nigritude ultramarine** competition by SearchGuild is widely acclaimed as ...
Comparison of search results for **nigritude ultramarine** during and after the ...
en.wikipedia.org/wiki/Nigritude_ultramarine - 37k - [Cached](#) - [Similar pages](#)

Slashdot | How To Get Googled, By Hook Or By Crook
The current 3rd result showcases the "**Nigritude Ultramarine** Fighting Force" who ... When discussing **nigritude ultramarine** [slashdot.org] it is important to ...
slashdot.org/article.pl?sid=04/05/09/1840217 - 110k - [Cached](#) - [Similar pages](#)

The Nigritude Ultramarine Search Engine Optimization Contest
It's sweeping the web -- or at least search engine optimizers -- a new contest to rank tops for the term **nigritude ultramarine** on Google.
searchenginewatch.com/sereport/article.php/3360231 - 57k - [Cached](#) - [Similar pages](#)

Sponsored Links

Business Blogging Seminar
Coming to L.A. March 16
Top bloggers reveal key techniques
www.blogbusinesssummit.com
Los Angeles, CA

Full-Time SEO & SEM Jobs
Find companies big & small hiring full-time SEO & SEM pros right now
CareerBuilder.com

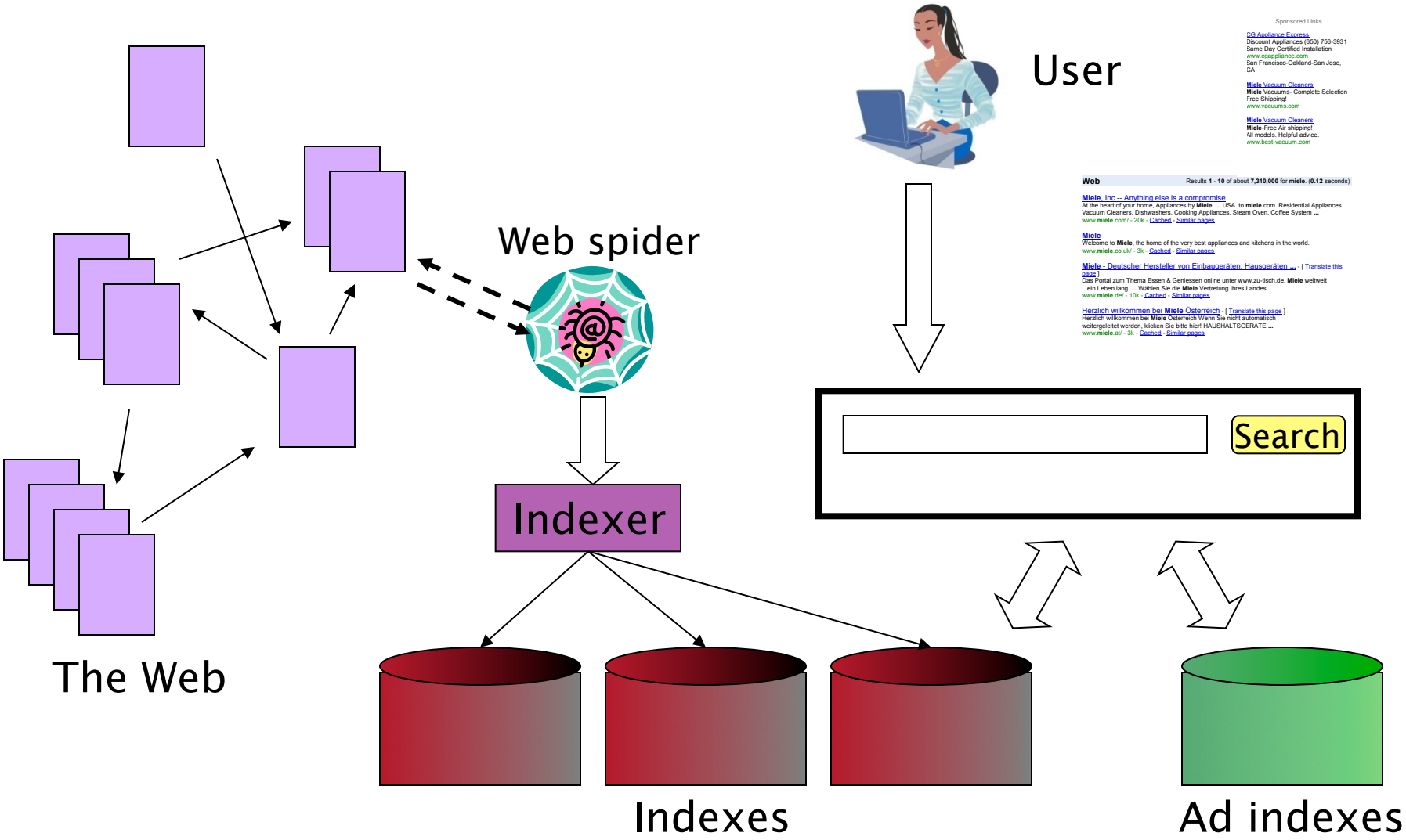
SEO Contests
Information on SEO Contests like the **Nigritude Ultramarine** contest.
www.seo-contests.com/

The SEO Book
Nigritude Ultramarine & SEO secrets
Fun, free, raw, & different.
www.seobook.com

Ultramarine - Companion
Music - Dance - Electronic
Overstock.com

Done

Web search basics

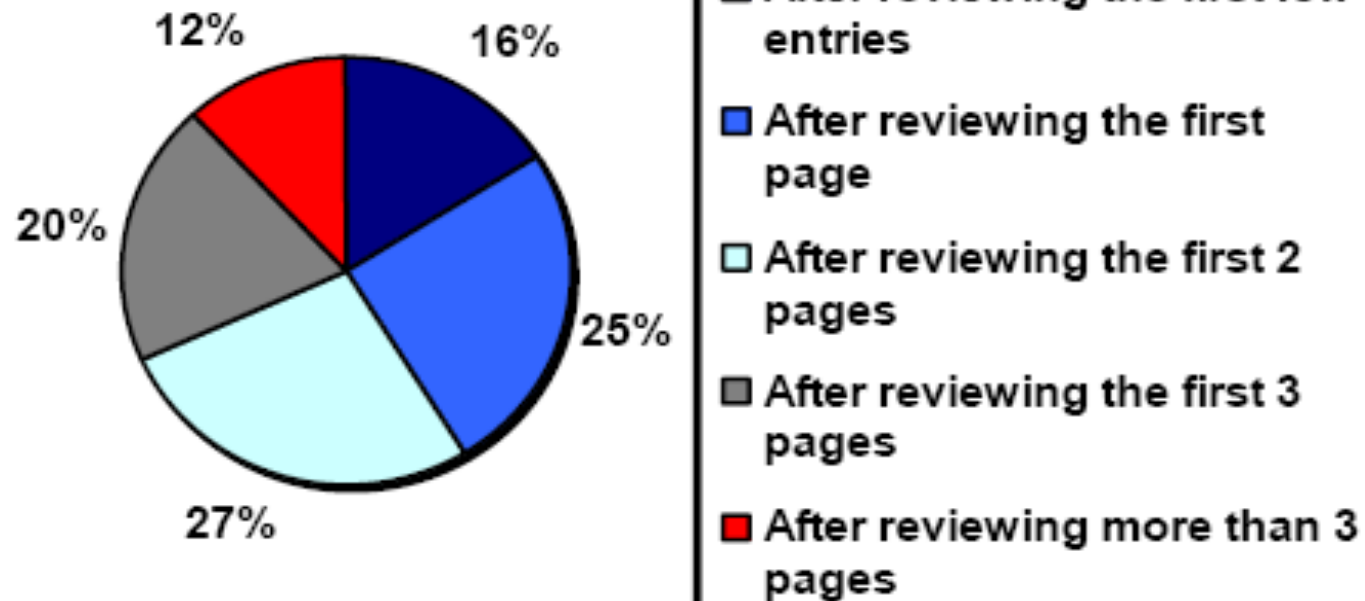


User Needs

- Need [Brod02, RL04]
 - **Informational** – want to learn about something (~40% / 65%)
Low hemoglobin
 - **Navigational** – want to go to that page (~25% / 15%)
United Airlines
 - **Transactional** – want to do something (web-mediated) (~35% / 20%)
 - Access a service
Seattle weather
 - Downloads
Mars surface images
 - Shop
Canon S410
 - **Gray areas**
 - Find a good hub
Car rental Brasil
 - Exploratory search “see what’s there”

How far do people look for results?

“When you perform a search on a search engine and don’t find what you are looking for, at what point do you typically either revise your search, or move on to another search engine? (Select one)”



(Source: iprospect.com WhitePaper_2006_SearchEngineUserBehavior.pdf)

Users' empirical evaluation of results

- Quality of pages varies widely
 - Relevance is not enough
 - Other desirable qualities (not traditional IR!!)
 - Content: Trustworthy, diverse, non-duplicated, well maintained
 - Web readability: display correctly & fast
 - No annoyances: pop-ups, etc
- Precision vs. recall
 - On the web, recall seldom matters
- What matters
 - Precision at 1? Precision above the fold?
 - Comprehensiveness – must be able to deal with obscure queries
 - Recall matters when the number of matches is very small
- User perceptions may be unscientific, but are significant over a large aggregate

Users' empirical evaluation of engines

- Relevance and validity of results
- UI – Simple, no clutter, error tolerant
- Trust – Results are objective
- Coverage of topics for polysemic queries
- Pre/Post process tools provided
 - Mitigate user errors (auto spell check, search assist,...)
 - Explicit: Search within results, more like this, refine ...
 - Anticipative: related searches
- Deal with idiosyncrasies
 - Web specific vocabulary (#lcot, C#, +Chris)
 - Impact on stemming, spell-check, etc.
 - Web addresses typed in the search box

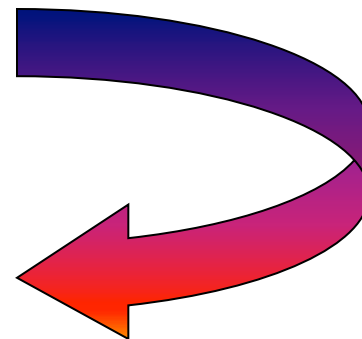
The trouble with paid search ads ...

- It costs money. What's the alternative?
- *Search Engine Optimization:*
 - “Tuning” your web page to rank highly in the algorithmic search results for select keywords
 - Alternative to paying for placement
 - Thus, intrinsically a marketing function
- Performed by companies, webmasters and consultants (“Search engine optimizers” -- SEOs) for their clients
- Some perfectly legitimate, some very shady

Simplest forms

- First generation engines relied heavily on *tf-idf*
 - The top-ranked pages for the query **maui resort** were the ones containing the most **maui**'s and **resort**'s
- SEOs responded with dense repetitions of chosen terms
 - e.g., **maui resort maui resort maui resort**
 - Often, the repetitions would be in the same color as the background of the web page
 - Repeated terms got indexed by crawlers
 - But not visible to humans on browsers

Pure word density cannot
be trusted as an IR signal



Adversarial IR

- Search engines have responded to this in many ways:
 - Quality/spam detection measures on pages
 - Use of other metrics such as link analysis, user votes
- But it's a fundamentally new world:
 - Before, we assumed that the documents just existed independently, and we could build an IR system for them
 - Now, the documents are being changed in ways that attempt to maximize their ranking in search results
- **Adversarial IR**: the unending (technical) battle between SEO's and web search engines
 - For more see: <http://airweb.cse.lehigh.edu/>

Introduction to **Information Retrieval**

Web search