Information Retrieval & Social Web

CS 525/DS 595

Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

Reminder

- HW1 due date is tomorrow
- Notify me your project team members by tomorrow

Previous Class…

Query Optimization

Query optimization

- Consider a query that is an *AND* of *n* terms.
- For each of the *n* terms, get its postings, then *AND* them together.
- What is the best order for query processing?

Brutus	2	4	8	16	32	64	128	
Caesar	1	2	3	5	8	16	21	34
Calpurnia	13	16						

Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Previous Class…

Query Optimization

Preprocessing Documents
➔Tokenization, Normalization, Stemming, Stop words

Initial stages of text processing

- Tokenization
  - Cut character sequence into word tokens
    - Deal with ***“John’s”***, ***a state-of-the-art solution***
- Normalization
  - Map text and query term to same form
    - You want ***U.S.A.*** and ***USA*** to match
- Stemming
  - We may wish different forms of a root to match
    - ***authorize***, ***authorization***
- Stop words
  - We may omit very common words (or not)
    - ***the, a, to, of***

Previous Class…

Query Optimization

Preprocessing Documents
→Tokenization, Normalization, Stemming, Stop words

Skip pointers & Positional index

Proximity Queries in Search Engines

- Google Search supports
  - keyword1 AROUND(n) keyword2
- Bing
  - keyword1 near:n keyword2 where n=the number of maximum separating words.
- Yahoo
  - keyword1 NEAR keyword2
- Exalead
  - keyword1 NEAR/n keyword2 where n is the number of words.

E.g., hotel around(5) terminal vs hotel around(3) terminal at Google

Today…

- Need a better index than simple <term: docs>
- How can we improve on our basic index?
  - **Skip pointers**: faster postings merges
  - **Positional index**: Phrase queries and Proximity queries
  - **Permuterm index**: Wildcard queries

Today…

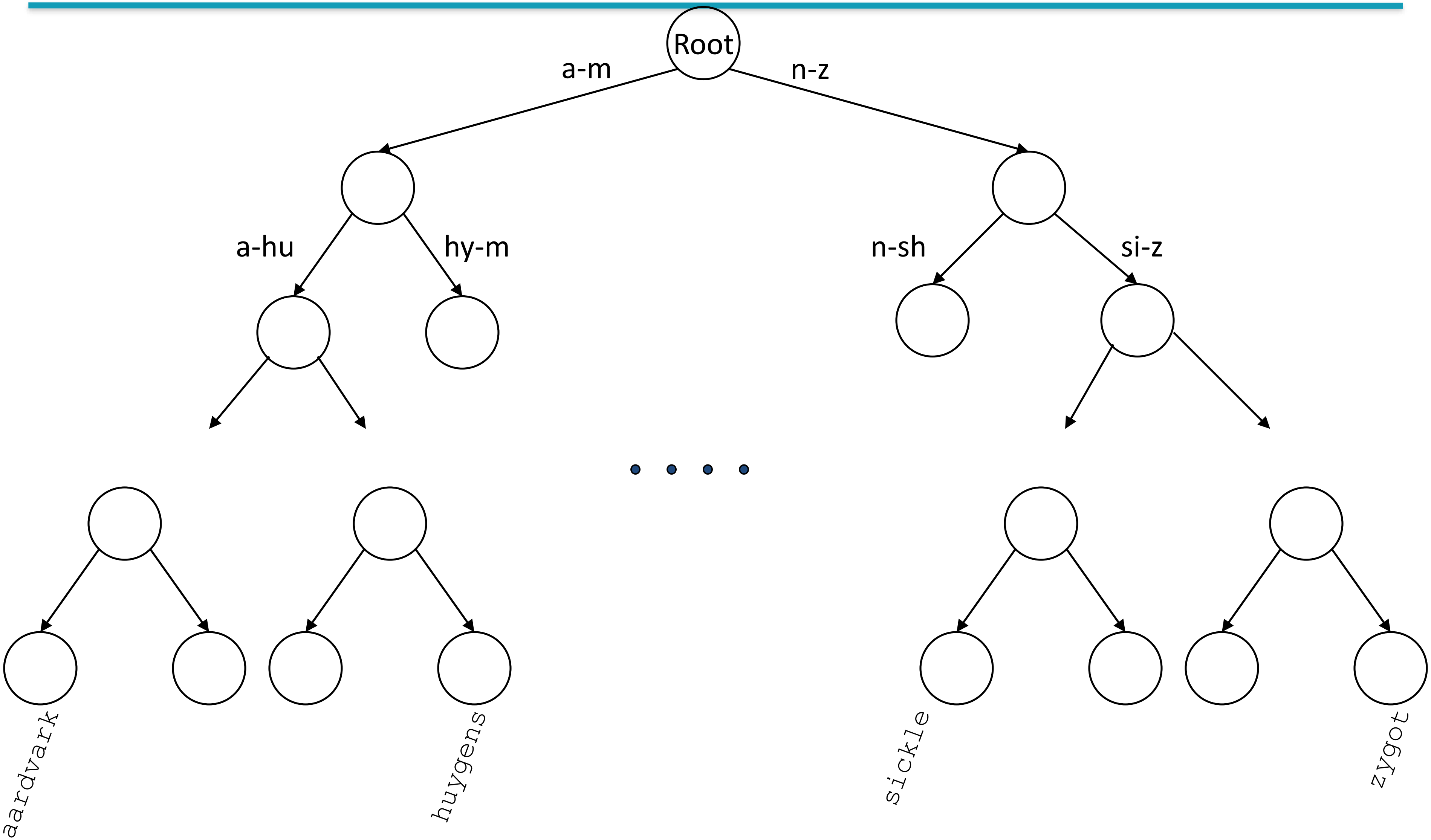
- Need a better index than simple <term: docs>
- How can we improve on our basic index?
  - **Skip pointers**: faster postings merges
  - **Positional index**: Phrase queries and Proximity queries
  - **Permuterm index**: Wildcard queries

Wild-card queries

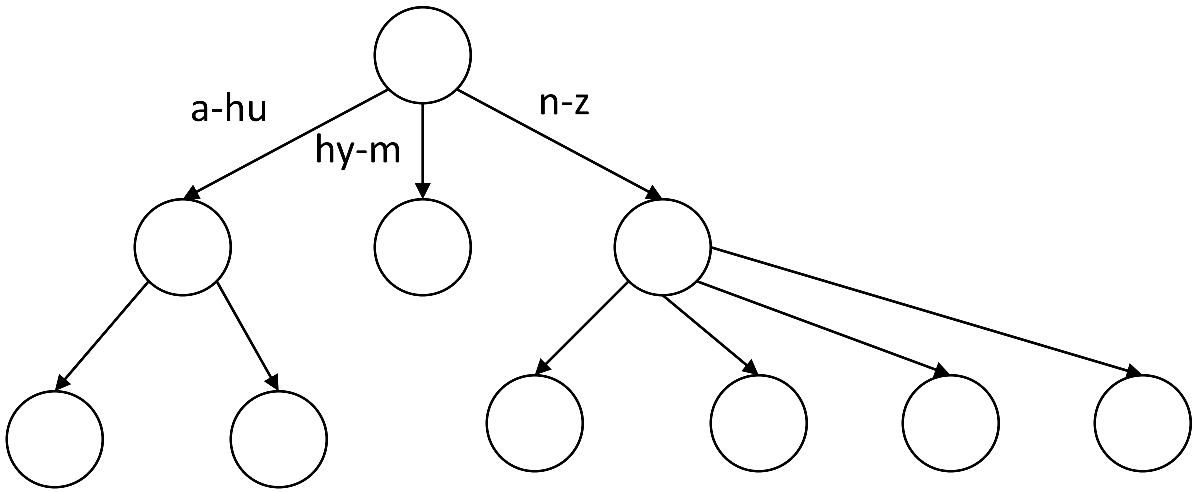
Wild-card queries: *

- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***

Tree: binary tree



Tree: B-tree



- Definition: Every internal nodel has a number of children in the interval [*a*,*b*] where *a, b* are appropriate natural numbers, e.g., [2,4].

Wild-card queries: *

- ***mon*:*** find all docs containing any word beginning with "mon".
- Easy with binary tree (or B-tree) lexicon: retrieve all words in range: ***mon ≤ w < moo***
- ****mon:*** find words ending in "mon": harder
  - Maintain an additional B-tree for terms *backwards.*

  Can retrieve all words in range: ***nom ≤ w < non****.*

Exercise: from this, how can we enumerate all terms meeting the wild-card query ***pro*cent*** ?

Query processing

- At this point, we have an enumeration of all terms in the dictionary that match the wild-card query.
- We still have to look up the postings for each enumerated term.
- E.g., consider the query:

  se*ate *AND* ***fil*er***

  This may result in the execution of many Boolean *AND* queries.

B-trees handle *'s at the end of a query term

- How can we handle *'s in the middle of query term?
  - ***co*tion***
- We could look up ***co**** AND ****tion*** in a B-tree and intersect the two term sets
  - Expensive
- The solution: transform wild-card queries so that the *'s occur at the end
- This gives rise to the **Permuterm** Index.

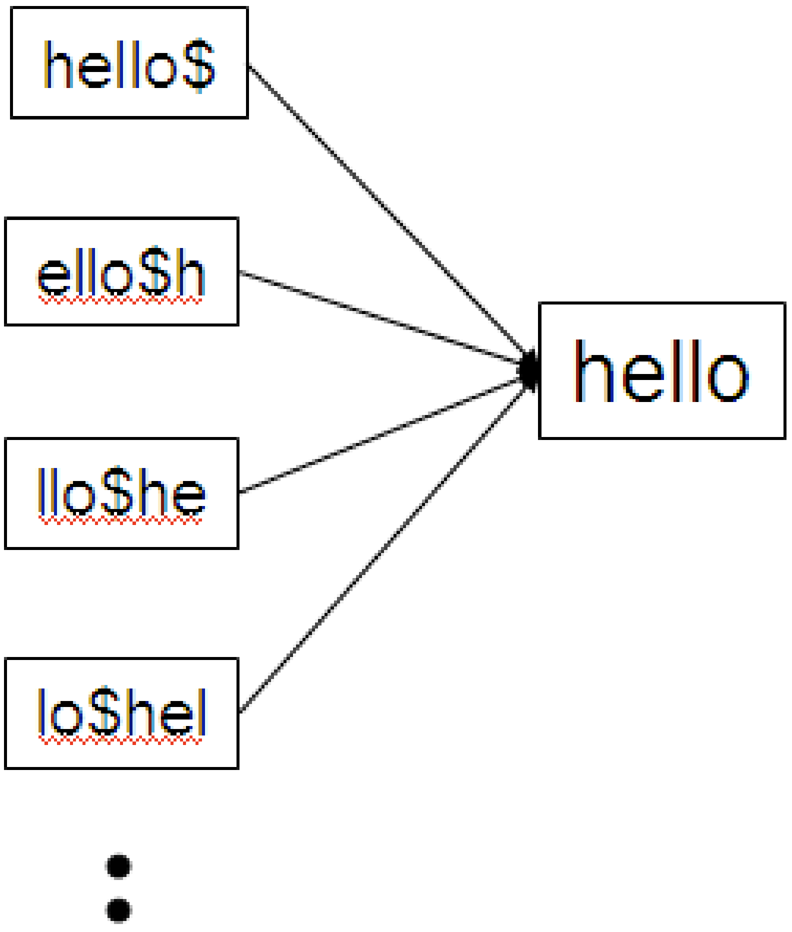
Permuterm index

- For term ***hello***, index under:
  - ***hello$, ello$h, llohe, lohel, o$hell, $hello***

  where $ is a special symbol.

Query = ***hel*o***
X=*hel*, Y=*o*
Lookup ***o$hel****

- Queries:
  - **X** lookup on **X$** **X*** lookup on $**X***
  - ***X** lookup on **X$*** ***X*** lookup on **X***
  - **X*Y** lookup on **Y$X***
  - **X*Y*Z** ??? Exercise!


hello$
ello$h
llo$he
lo$hel
hello

Permuterm query processing

- Rotate query wild-card to the right
- Now use B-tree lookup as before.
- *Permuterm problem: ≈ quadruples lexicon size*

Empirical observation for English.

Vector Space Retrieval

Take-away today

- Ranking search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- Term frequency: This is a key ingredient for ranking.
- Tf-idf ranking: best known traditional ranking scheme
- Vector space model: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)

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 of the collection.
- Also good for applications: Applications can easily consum 1000s of results.
- Not good for the majority of users
- Most users are not capable 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.

Empirical investigation of the effect of ranking

- How can we measure how important ranking is?
- Observe what searchers do when they are searching in a controlled setting
  - Videotape them
  - Ask them to “think aloud”
  - Interview them
  - Eye-track them
  - Time them
  - Record and count their clicks
- The following slides are from Dan Russell’s JCDL talk
- Dan Russell is the “Über Tech Lead for Search Quality & User Happiness” at Google.

So.. Did you notice the FTD official site?

To be honest, I didn't even look at that.

At first I saw "from $20" and $20 is what I was looking for.

To be honest, 1800-flowers is what I'm familiar with and why I went there next even though I kind of assumed they wouldn't have $20 flowers

And you knew they were expensive?

I knew they were expensive but I thought "hey, maybe they've got some flowers for under $20 here…"

But you didn't notice the FTD?

No I didn't, actually… that's really funny.

Interview video

Rapidly scanning the results

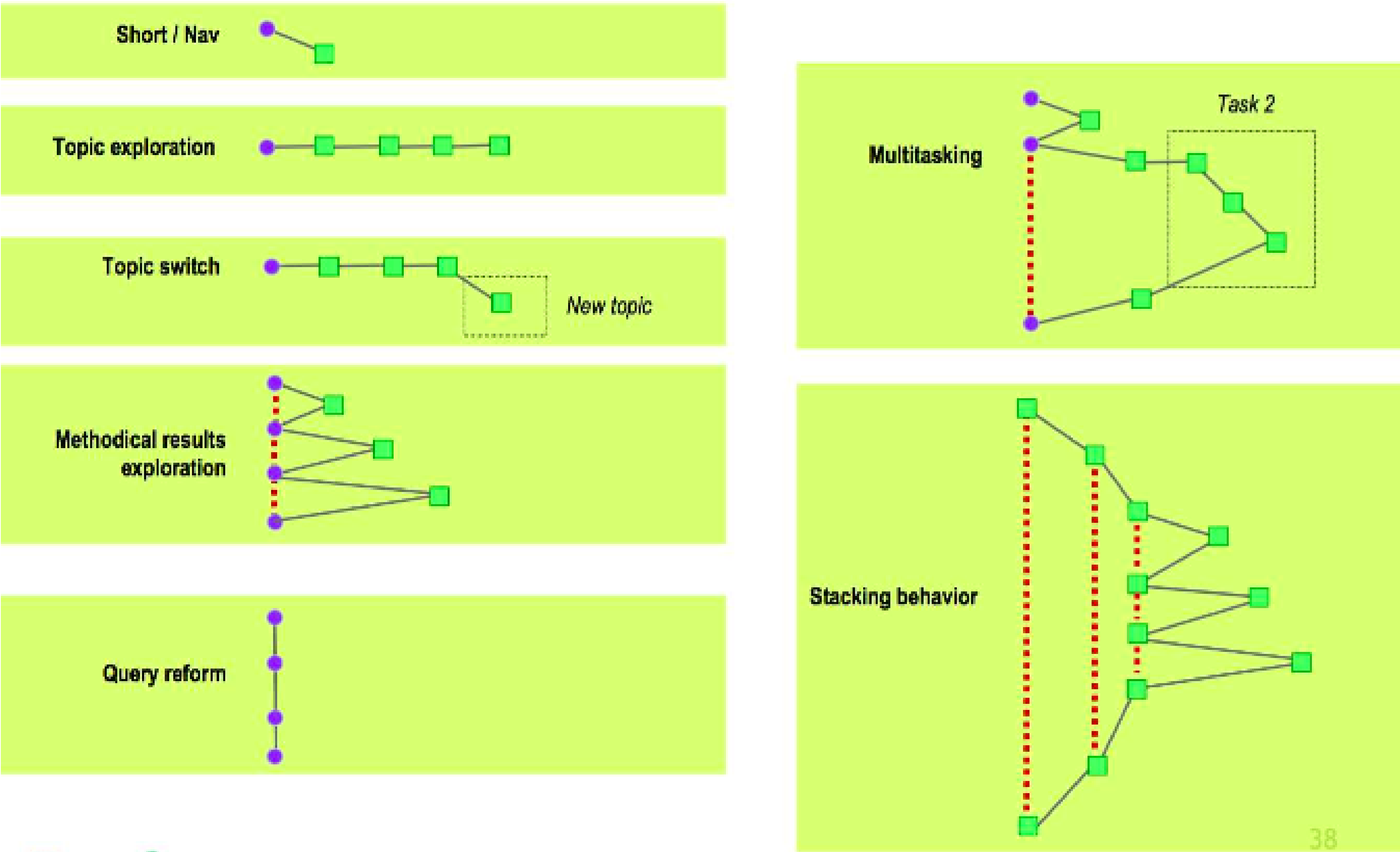
Note scan pattern:

Page 3: Result 1
Result 2
Result 3
Result 4
Result 3
Result 2
Result 4
Result 5
Result 6 <click>

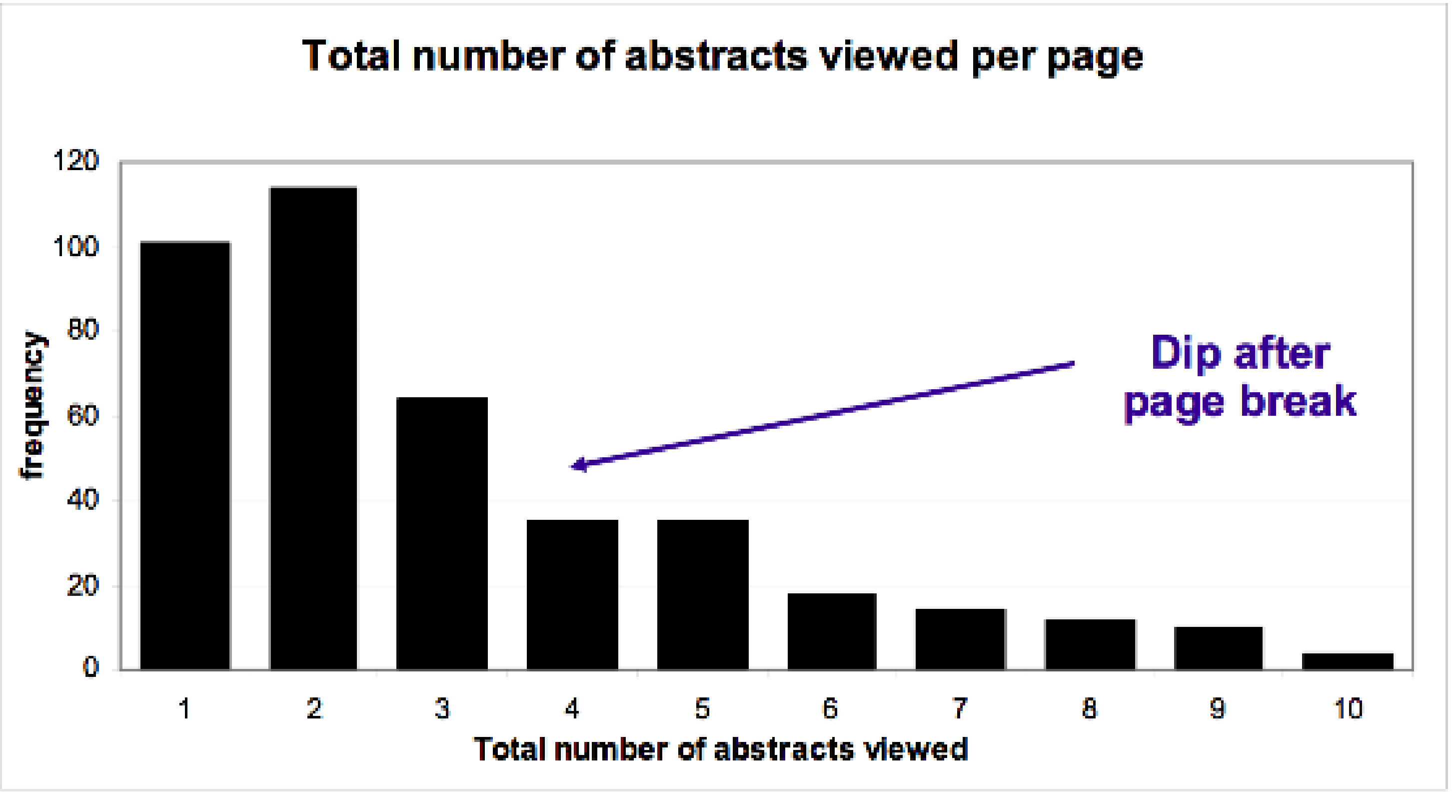
Q: Why do this?

A: What's learned later influences judgment of earlier content.

Kinds of behaviors we see in the data



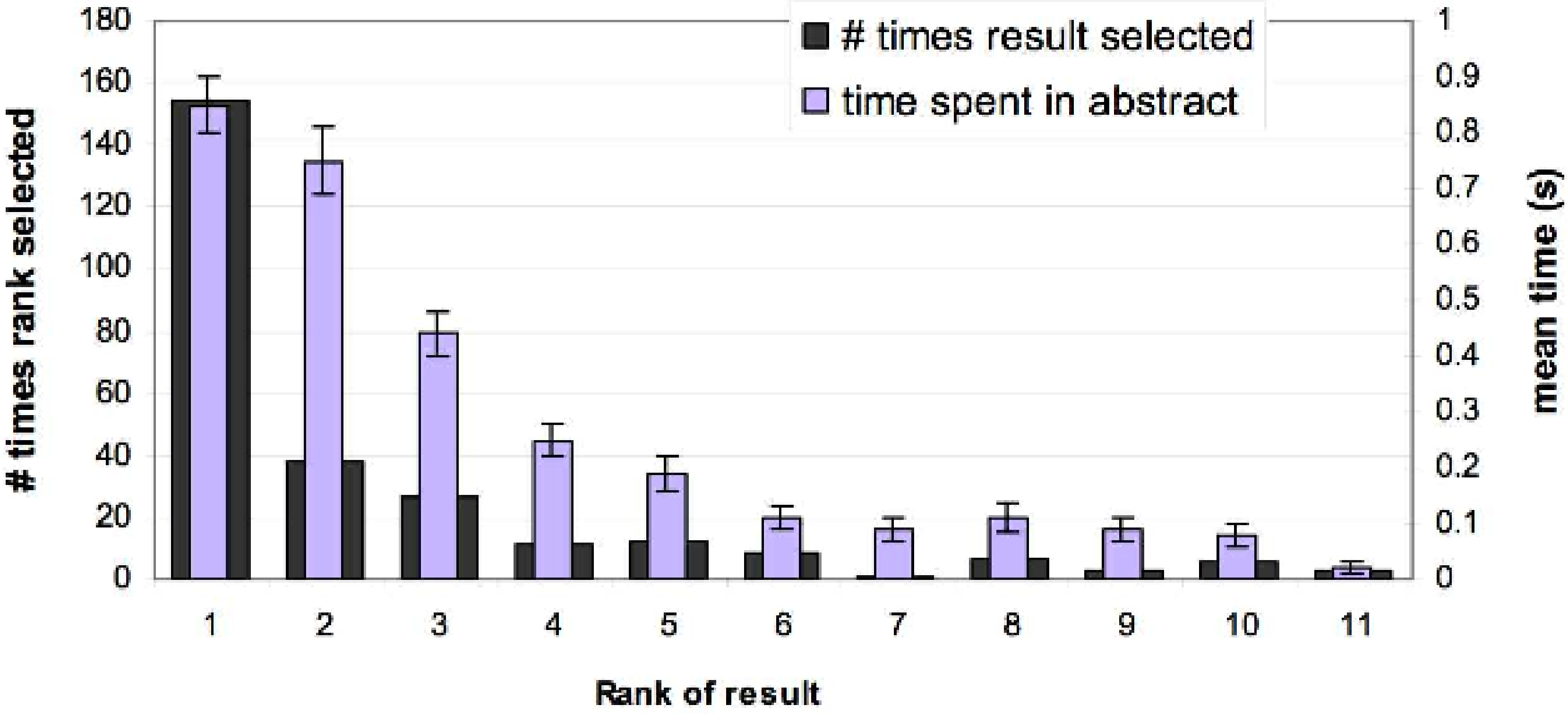
How many links do users view?



Mean: 3.07 Median/Mode: 2.00

Google

Looking vs. Clicking

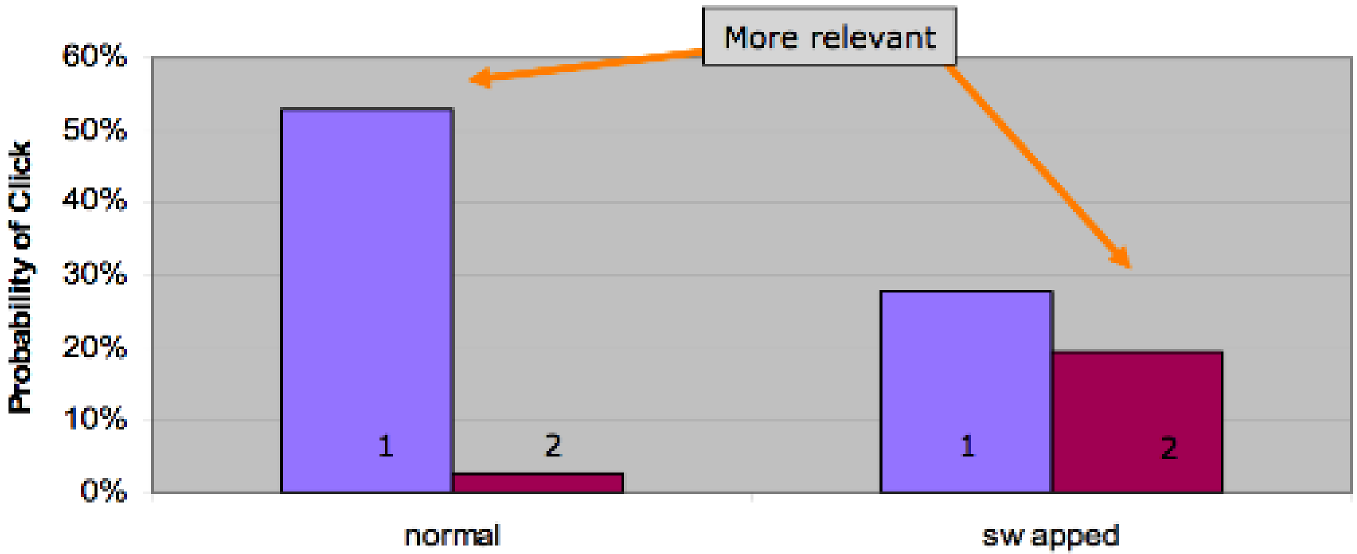


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

Presentation bias – reversed results

- Order of presentation influences where users look **AND** where they click



Importance of ranking: Summary

- Viewing abstracts: Users are a lot more likely to read the abstracts of the top-ranked pages (1, 2, 3, 4) than the abstracts of the lower ranked pages (7, 8, 9, 10).
- Clicking: Distribution is even more skewed for clicking
- In 1 out of 2 cases, users click on the top-ranked page.
- Even if the top-ranked page is not relevant, 30% of users will click on it.
- → Getting the ranking right is very important.
- → Getting the top-ranked page right is most important.

Scoring as the basis of ranked retrieval

Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.
- How can we accomplish such a ranking of the documents in the collection with respect to a query?
- Assign a score to each query-document pair, say in [0, 1].
- This score measures how well document and query "match".

Factors impacting query-document score

- …
- …
- …

Query-document matching scores

- How do we compute the score of 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
- We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let *A* and *B* be two sets
- Jaccard coefficient:

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$(A \neq \emptyset \text{ or } B \neq \emptyset)$

- 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.

Jaccard coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
  - Query: “ides of March”
  - Document “Caesar died in March”
  - JACCARD(q, d) = 1/6

What’s wrong with Jaccard?

- It doesn’t consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.

Term Frequency

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	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 as a binary vector $\in \{0, 1\}^{|V|}$.

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
. . .						

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the order of words in a document.
- *John is quicker than Mary and Mary is quicker than John* are represented the same way.
- This is called a 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 because:
- A document with tf = 10 occurrences of the term is more relevant than a document with tf = 1 occurrence of the term.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Instead of raw frequency: Log frequency weighting

- The log frequency weight of term t in d is defined as follows

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

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

$$\textsc{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

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

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

- q: [information on cars] d: "all you've ever wanted to know about cars"

- q: [information on cars] d: "information on trucks, information on planes, information on trains"

Exercise

- Compute the Jaccard matching score and the tf matching score for the following query-document pairs.
- q: [information on cars] d: “all you’ve ever wanted to know about cars”
- q: [information on cars] d: “information on trucks, information on planes, information on trains”
- q: [red cars and red trucks] d: “cops stop red cars more often”

TF-IDF Weighting

Frequency in document vs. frequency in collection

- In addition, to term frequency (the frequency of the term in the document) . . .
- . . .we also want to use the frequency of the term in the collection for weighting and ranking.

Desired weight for rare terms

- Rare terms are more informative than frequent terms.
- 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.
- → We want high weights for rare terms like ARACHNOCENTRIC.

Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn't . . .
- . . . but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → For frequent terms like GOOD, INCREASE and LINE, we want positive weights . . .
- . . . but lower weights than for rare terms.

Document frequency

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights for frequent words like GOOD, INCREASE and LINE.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

idf weight

- df_t is the document frequency, the number of documents that *t* occurs in.
- df_t is an inverse measure of the informativeness of term *t*.
- We define the idf weight of term t as follows:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

  (*N* is the number of documents in the collection.)
- idf_t is a measure of the informativeness of the term.
- [log N/df_t] instead of [N/df_t] to "dampen" the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.

Examples for idf

- Compute idf_t using the formula: $\mathrm{idf}_t = \log_{10} \frac{1{,}000{,}000}{\mathrm{df}_t}$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on ranking

- idf affects the ranking of documents for queries with at least two terms.
- For example, in the query "arachnocentric line", idf weighting increases the relative weight of ARACHNOCENTRIC and decreases the relative weight of LINE.
- idf has little effect on ranking for one-term queries.

Collection frequency vs. Document frequency

word	collection frequency	document frequency
INSURANCE	10440	3997
TRY	10422	8760

- Collection frequency of *t*: number of tokens of *t* in the collection
- Document frequency of *t*: number of documents *t* occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df (and idf) is better for weighting than cf (and “icf”).

tf-idf weighting

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

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

- tf-weight
- idf-weight
- 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

Summary: tf-idf

- Assign a tf-idf weight for each term t in each document *d*:
  $$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$
- The tf-idf weight . . .
  - . . . increases with the number of occurrences within a document. (term frequency)
  - . . . increases with the rarity of the term in the collection. (inverse document frequency)

The Vector Space Model

Binary incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	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 as a binary vector $\in \{0, 1\}^{|V|}$.

Count matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
. . .						

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$.

Binary → count → weight matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0
MERCY	1.51	0.0	1.90	0.12	5.25	0.88
WORSER	1.37	0.0	0.11	4.15	0.25	1.95
. . .						

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

Documents as vectors

- Each document is now represented as a real-valued vector of tf-idf weights $\in \mathrm{R}^{|V|}$.
- So we have a $|V|$-dimensional real-valued 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 web search engines
- Each vector is very sparse - most entries are zero.

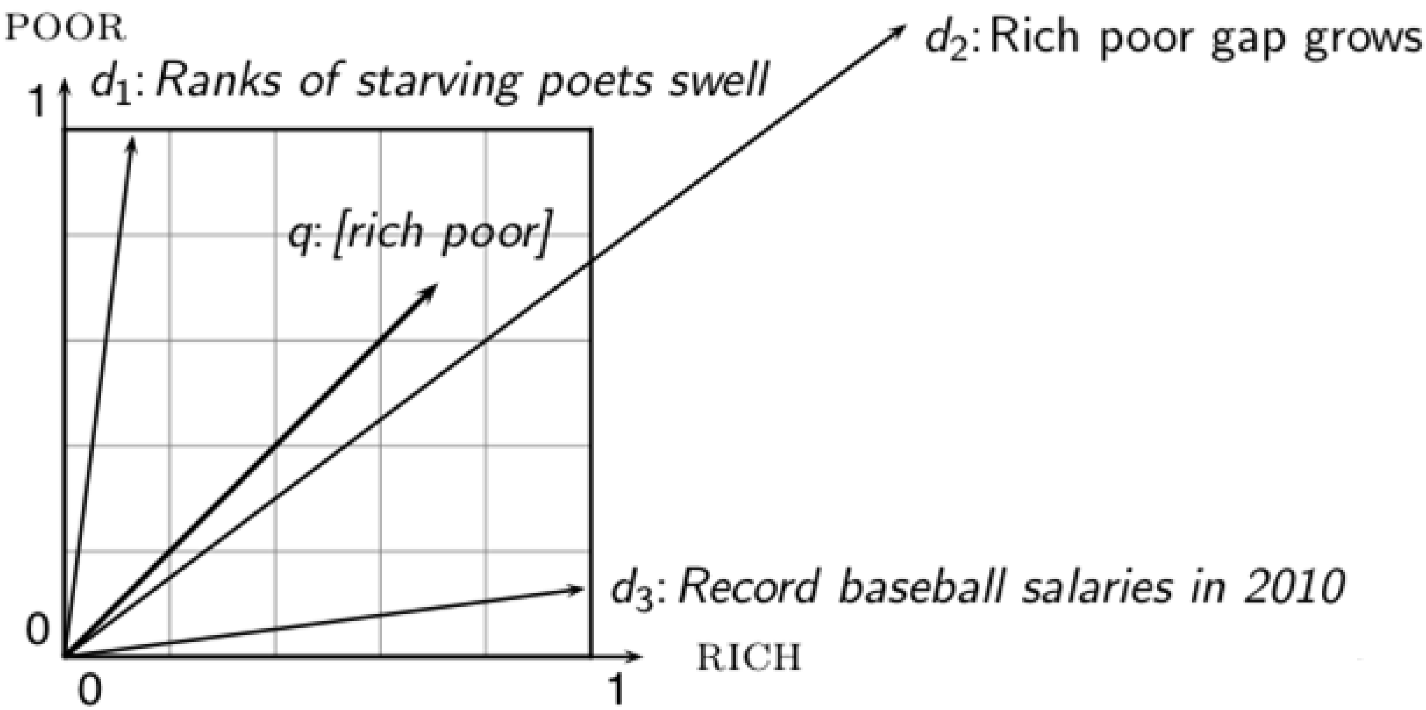
Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
- proximity = similarity
- proximity ≈ negative distance
- Recall: We're doing this because we want to get away from the you're-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents

How do we formalize vector space similarity?

- First cut: (negative) 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 of $\vec{q}$ and $\vec{d_2}$ is large although the distribution of terms in the query q
and the distribution of terms in the document d_2 are very similar.

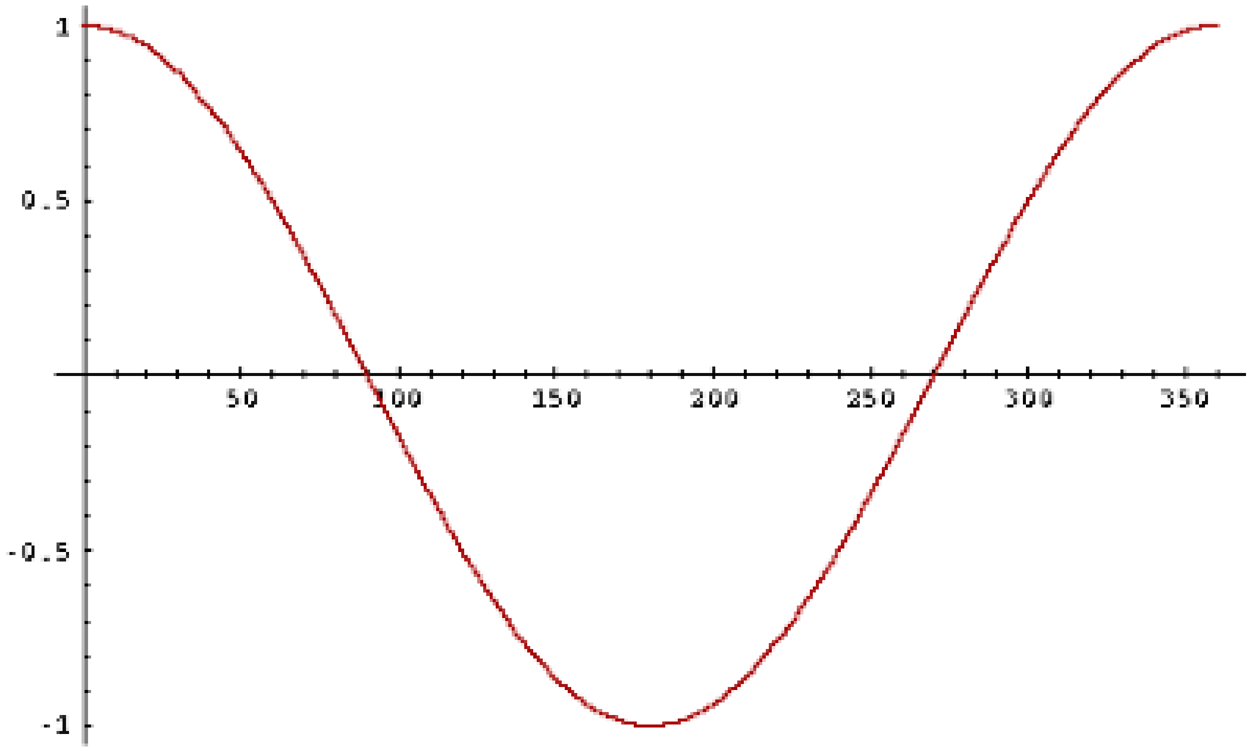
Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document d and append it to itself. Call this document *d′*. *d′* is twice as long as *d*.
- "Semantically" *d* and *d′* have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though the Euclidean distance between the two documents can be quite large.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order
  - Rank documents according to cosine(query,document) in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$

Cosine



Length normalization

- How do we compute the cosine?
- A vector can be (length-) normalized by dividing each of its components by its length – here we use the L_2 norm:
  $||x||_2 = \sqrt{\sum_i x_i^2}$
- This maps vectors onto the unit sphere . . .
- . . . since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$
- As a result, longer documents and shorter documents have weights of the same order of magnitude.
- Effect on the two documents *d* and *d′* (*d* appended to itself) from earlier slide: they have identical vectors after length-normalization.

Cosine similarity between query and document

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

- 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.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.
- This is the cosine similarity of $\vec{q}$ and $\vec{d}$. . . . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.