

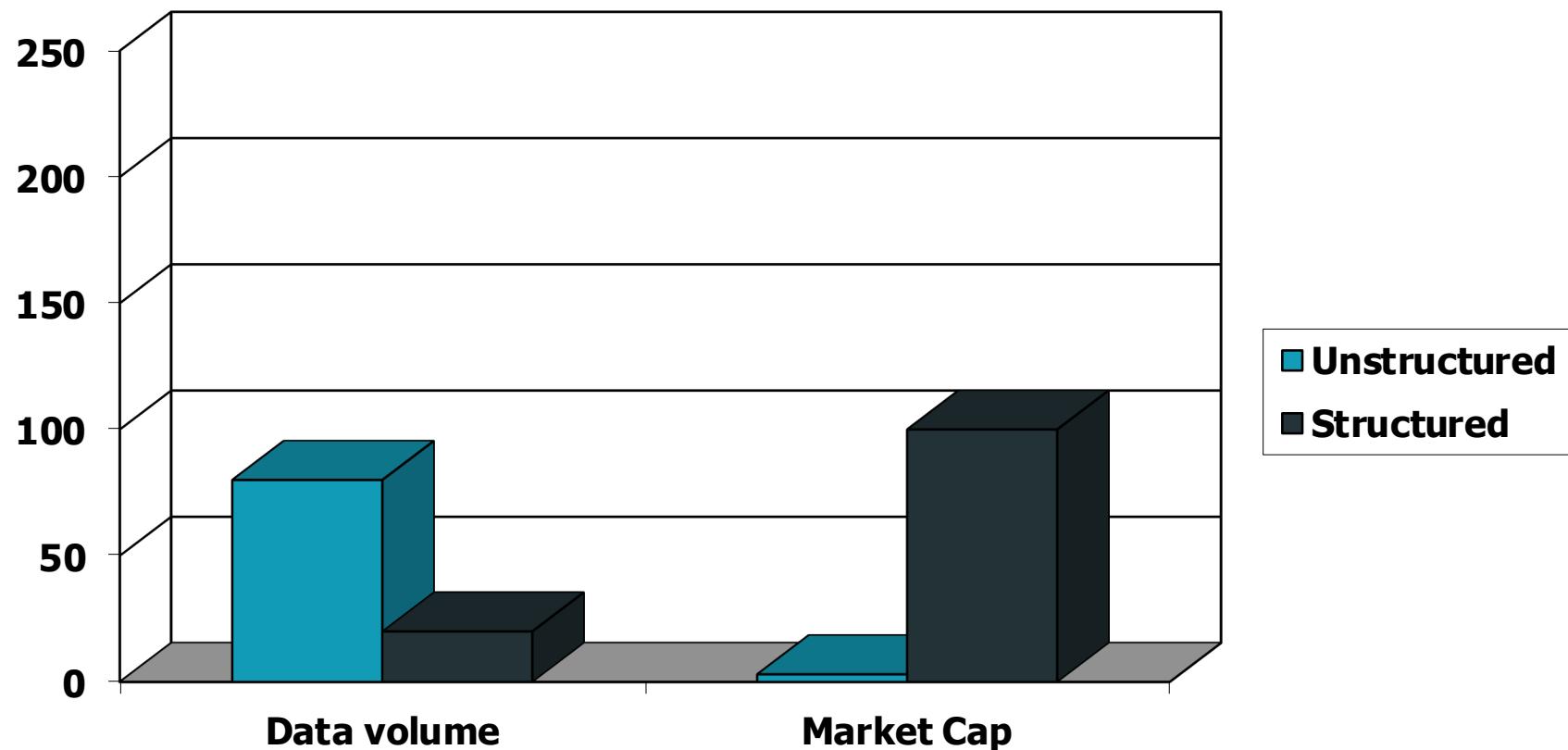
Introduction to **Information Retrieval**

Introducing Information Retrieval
and Web Search

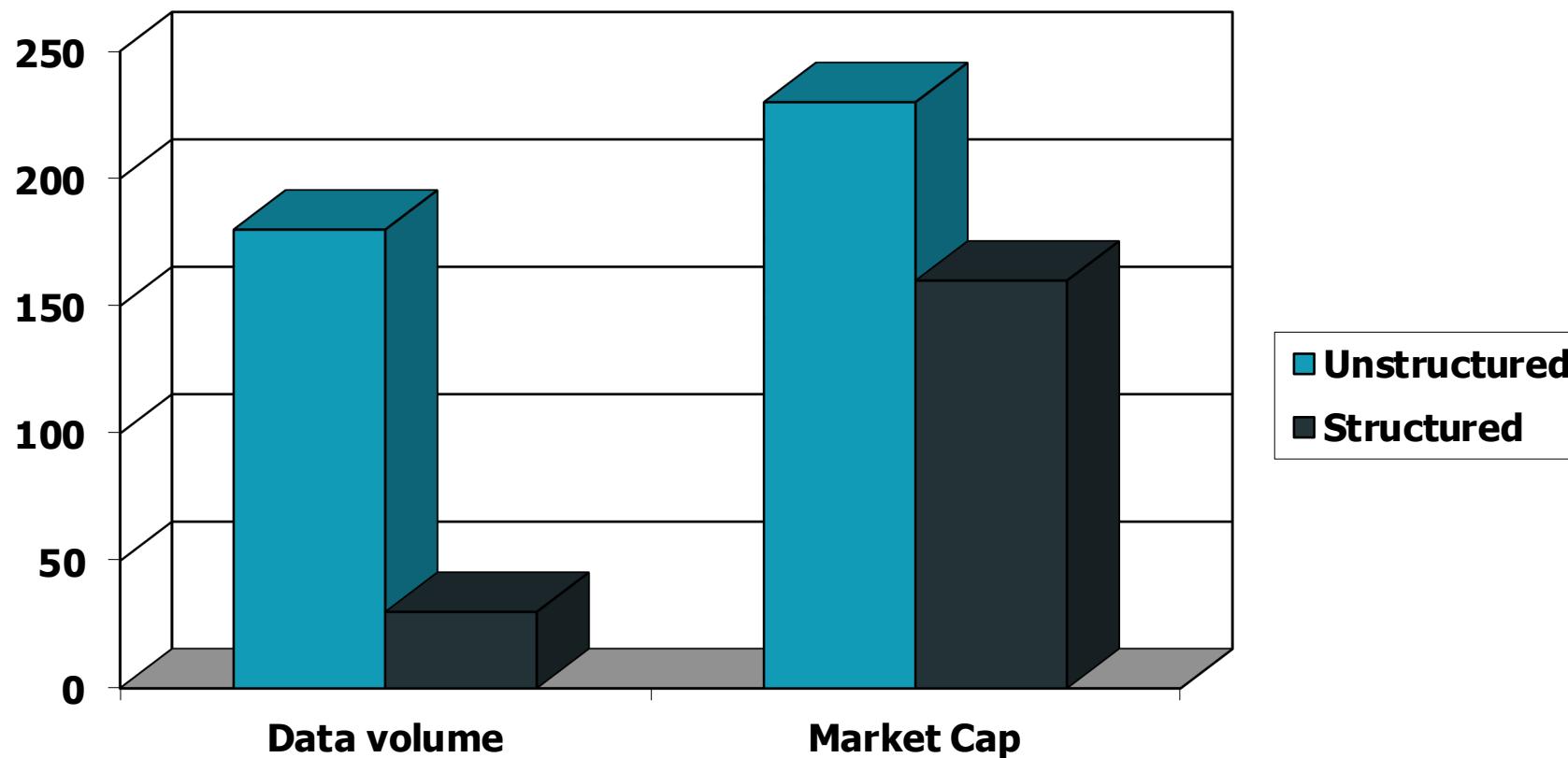
Information Retrieval

- Information Retrieval (IR) is **finding material** (usually documents) of an **unstructured nature** (usually text) that satisfies an **information need** from within **large collections** (usually stored on computers).
- These days we frequently think first of **web search**, but there are many other cases:
 - E-mail search
 - Searching your laptop
 - Corporate knowledge bases
 - Legal information retrieval

Unstructured (text) vs. structured (database) data in the mid-nineties



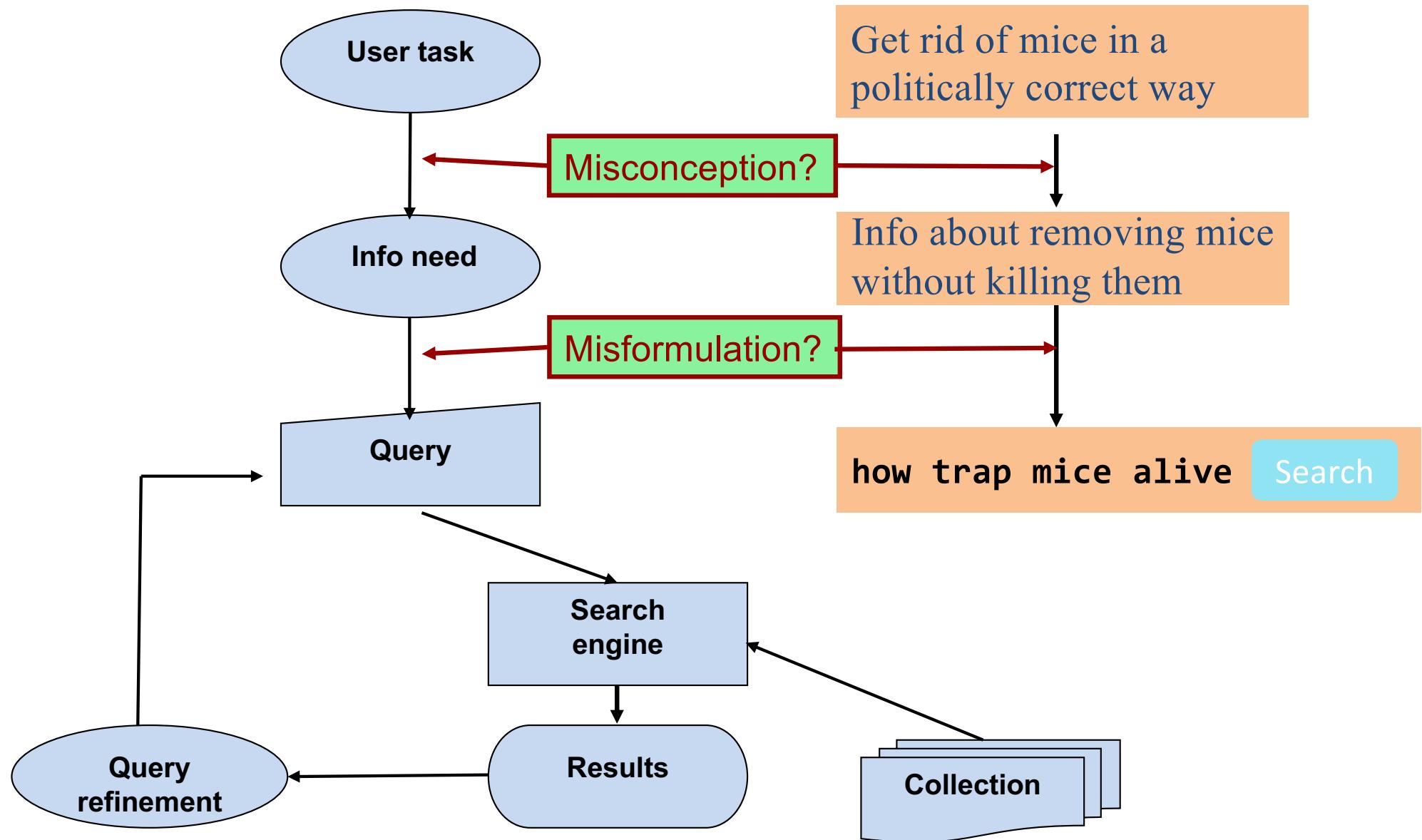
Unstructured (text) vs. structured (database) data today



Basic assumptions of Information Retrieval

- **Collection:** A set of documents
 - Assume it is a static collection for the moment
- **Goal:** Retrieve documents with information that is **relevant** to the user's **information need** and helps the user complete a **task**

The classic search model



How good are the retrieved docs?

- *Precision* : Fraction of retrieved docs that are relevant to the user's **information need**
- *Recall* : Fraction of relevant docs in collection that are retrieved
 - More precise definitions and measurements to follow later

Introduction to **Information Retrieval**

Term-document incidence matrices

Unstructured data in 1620

- Which plays of Shakespeare contain the words ***Brutus*** AND ***Caesar*** but ***NOT Calpurnia?***
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar***, then strip out lines containing ***Calpurnia?***
- Why is that not the answer?
 - Slow (for large corpora)
 - **NOT *Calpurnia*** is non-trivial
 - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible
 - Ranked retrieval (best documents to return)
 - Later lectures

Term-document incidence matrices

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

*Brutus AND Caesar BUT NOT
Calpurnia*

1 if play contains
word, 0 otherwise

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for *Brutus*, *Caesar* and *Calpurnia* (complemented) → bitwise AND.
 - 110100 AND
 - 110111 AND
 - 101111 =
 - **100100**

	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

Answers to query

- Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,
When Antony found Julius **Caesar** dead,
He cried almost to roaring; and he wept
When at Philippi he found **Brutus** slain.

- Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius **Caesar** I was killed i' the
Capitol; **Brutus** killed me.

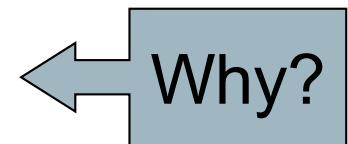


Bigger collections

- Consider $N = 1$ million documents, each with about 1000 words.
- Avg 6 bytes/word including spaces/punctuation
 - 6GB of data in the documents.
- Say there are $M = 500K$ *distinct* terms among these.

Can't build the matrix

- 500K x 1M matrix has half-a-trillion 0's and 1's.
- But it has no more than one billion 1's.
 - matrix is extremely sparse.
- What's a better representation?
 - We only record the 1 positions.

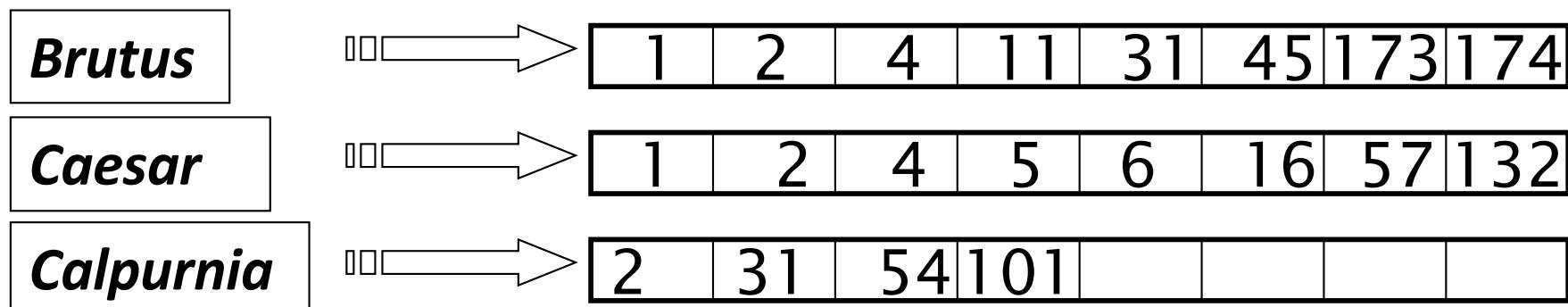


Introduction to **Information Retrieval**

The Inverted Index
The key data structure underlying modern IR

Inverted index

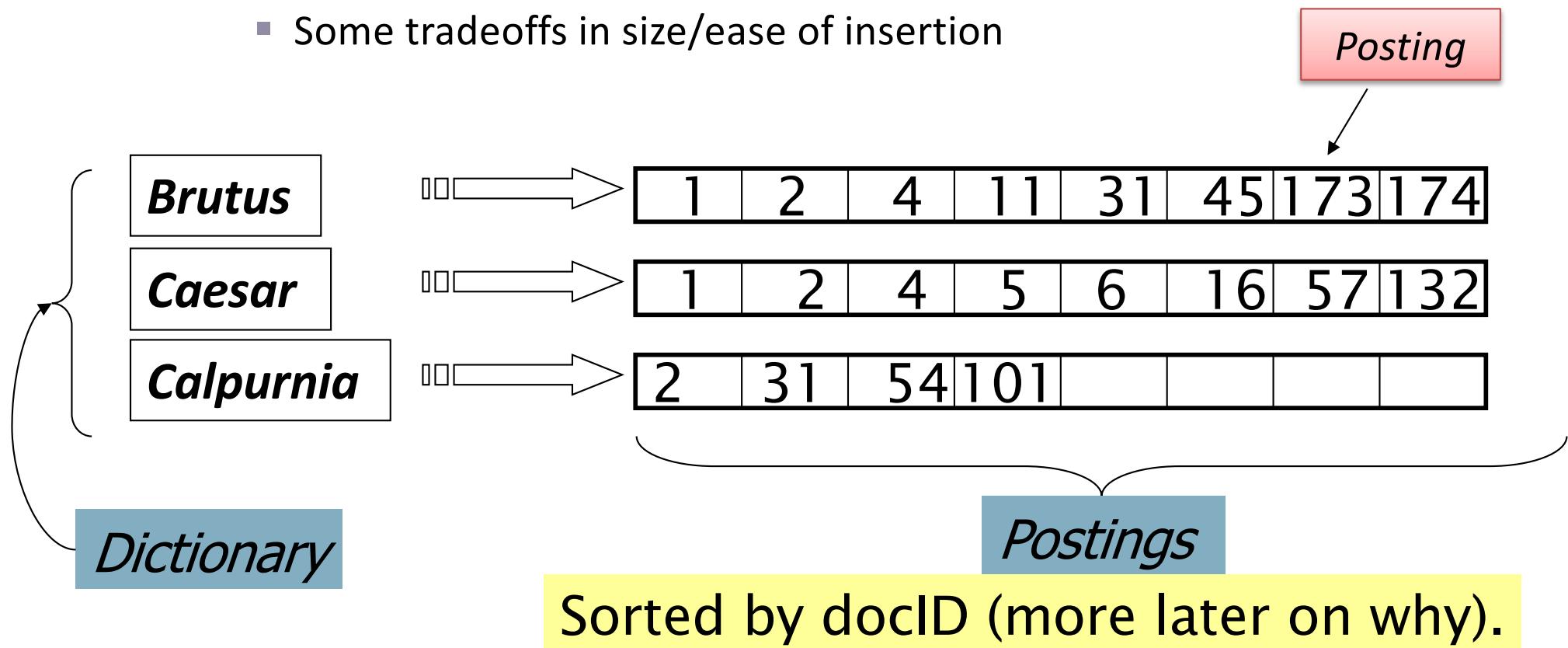
- For each term t , we must store a list of all documents that contain t .
 - Identify each doc by a **docID**, a document serial number
- Can we used fixed-size arrays for this?



What happens if the word **Caesar** is added to document 14?

Inverted index

- We need variable-size **postings lists**
 - On disk, a continuous run of postings is normal and best
 - In memory, can use linked lists or variable length arrays
 - Some tradeoffs in size/ease of insertion



Inverted index construction

Documents to be indexed



Friends, Romans, countrymen.
⋮

Tokenizer

Token stream

Friends

Romans

Countrymen

Linguistic modules

Modified tokens

friend

roman

countryman

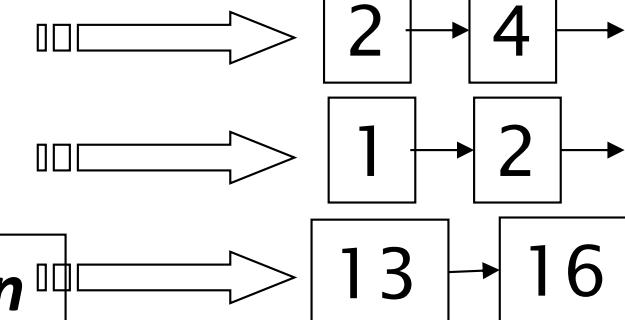
Indexer

Inverted index

friend

roman

countryman



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*

Indexer steps: Token sequence

- Sequence of (Modified token, Document ID) pairs.

Doc 1

I did enact Julius
Caesar I was killed
i' the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious



Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Indexer steps: Sort

- Sort by terms
 - At least conceptually
 - And then docID

Core indexing step

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
caesar	2
was	1
was	2
with	2

Indexer steps: Dictionary & Postings

- Multiple term entries in a single document are merged.
- Split into Dictionary and Postings
- Doc. frequency information is added.

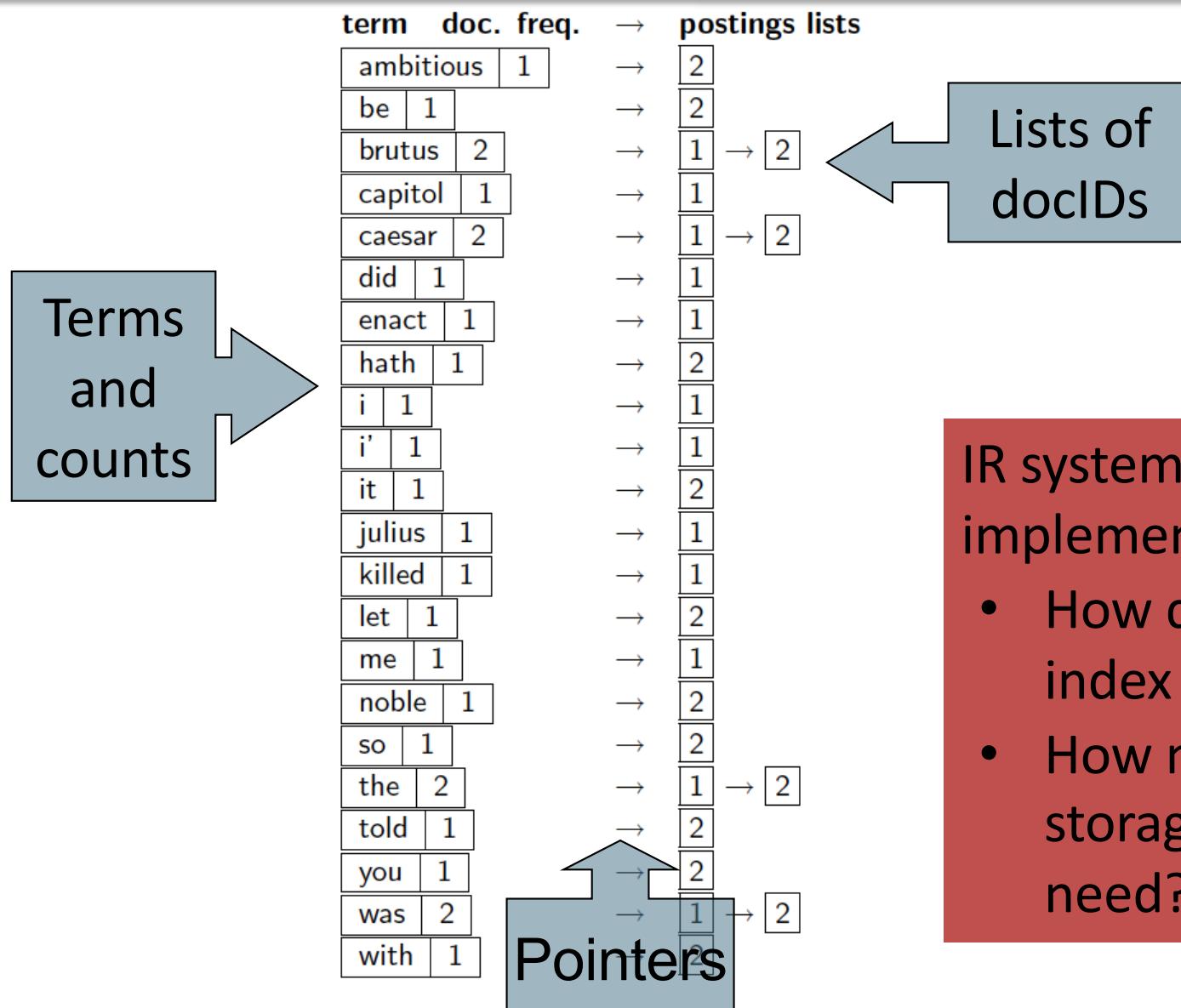
Why frequency?
Will discuss later.

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

→

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Where do we pay in storage?



IR system implementation

- How do we index efficiently?
 - How much storage do we need?

Introduction to **Information Retrieval**

Query processing with an inverted index

The index we just built

- How do we process a query?
 - Later – what kinds of queries can we process?

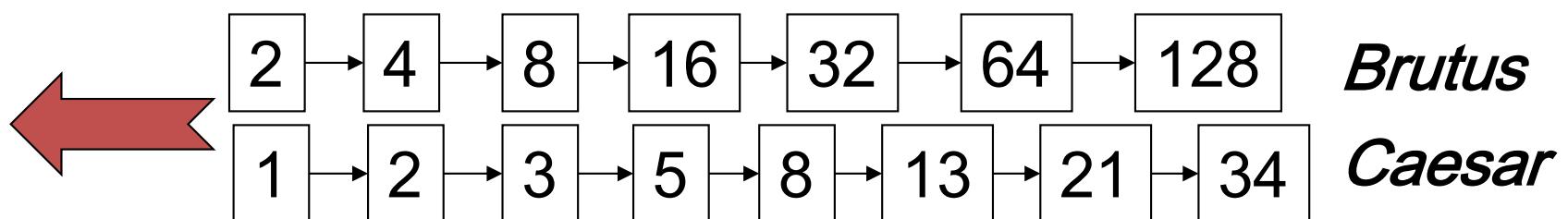


Query processing: AND

- Consider processing the query:

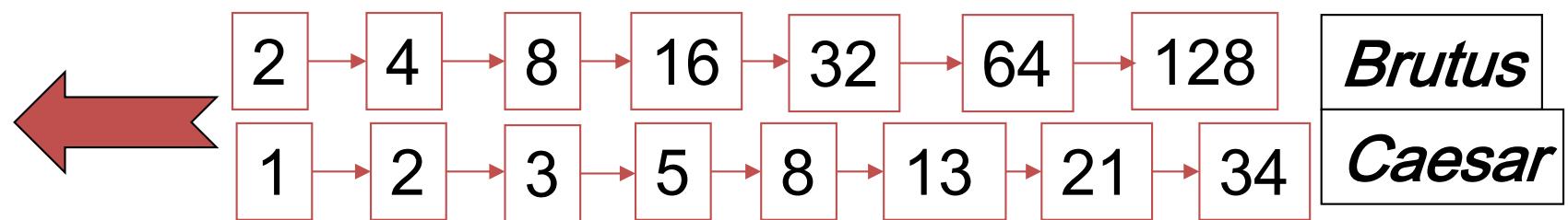
Brutus AND Caesar

- Locate ***Brutus*** in the Dictionary;
 - Retrieve its postings.
- Locate ***Caesar*** in the Dictionary;
 - Retrieve its postings.
- “Merge” the two postings (intersect the document sets):



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are x and y , the merge takes $O(x+y)$ operations.

Crucial: postings sorted by docID.

Intersecting two postings lists (a “merge” algorithm)

INTERSECT(p_1, p_2)

```
1  answer ← ⟨ ⟩  
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$   
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$   
4      then ADD(answer,  $\text{docID}(p_1)$ )  
5           $p_1 \leftarrow \text{next}(p_1)$   
6           $p_2 \leftarrow \text{next}(p_2)$   
7      else if  $\text{docID}(p_1) < \text{docID}(p_2)$   
8          then  $p_1 \leftarrow \text{next}(p_1)$   
9          else  $p_2 \leftarrow \text{next}(p_2)$   
10 return answer
```

Introduction to **Information Retrieval**

The Boolean Retrieval Model
& Extended Boolean Models

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
 - Boolean Queries are queries using *AND*, *OR* and *NOT* to join query terms
 - Views each document as a set of words
 - Is precise: document matches condition or not.
 - Perhaps the simplest model to build an IR system on
- Primary commercial retrieval tool for 3 decades.
- Many search systems you still use are Boolean:
 - Email, library catalog, macOS Spotlight

Example: WestLaw <http://www.westlaw.com/>

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992; new federated search added 2010)
- Tens of terabytes of data; ~700,000 users
- Majority of users *still* use boolean queries
- Example query:
 - What is the statute of limitations in cases involving the federal tort claims act?
 - **LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM**
 - /3 = within 3 words, /S = in same sentence

Example: WestLaw <http://www.westlaw.com/>

- Another example query:
 - Requirements for disabled people to be able to access a workplace
 - **disabl! /p access! /s work-site work-place (employment /3 place**
- Note that SPACE is disjunction, not conjunction!
- Long, precise queries; proximity operators; incrementally developed; not like web search
- Many professional searchers still like Boolean search
 - You know exactly what you are getting
- But that doesn't mean it actually works better...

Boolean queries:

More general merges

- Exercise: Adapt the merge for the queries:
Brutus AND NOT Caesar
Brutus OR NOT Caesar
- Can we still run through the merge in time $O(x+y)$?
What can we achieve?

Merging

What about an arbitrary Boolean formula?

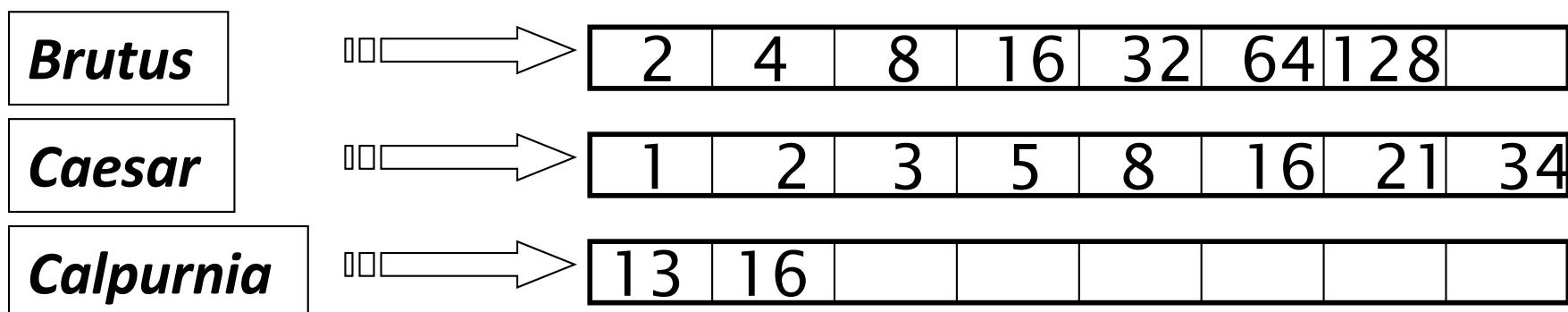
(Brutus OR Caesar) AND NOT

(Antony OR Cleopatra)

- Can we always merge in “linear” time?
 - Linear in what?
- Can we do better?

Query optimization

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



Query: *Brutus AND Calpurnia AND Caesar*

Query optimization example

- Process in order of increasing freq:
 - *start with smallest set, then keep cutting further.*

This is why we kept
document freq. in dictionary

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

Execute the query as (**Calpurnia AND Brutus**) AND **Caesar**.

Exercise

- Recommend a query processing order for

*(tangerine OR trees) AND
(marmalade OR skies) AND
(kaleidoscope OR eyes)*

- Which two terms should we process first?

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

More general optimization

- e.g., *(madding OR crowd) AND (ignoble OR strife)*
- Get doc. freq.'s for all terms.
- Estimate the size of each *OR* by the sum of its doc. freq.'s (conservative).
- Process in increasing order of *OR* sizes.

Query processing exercises

- **Exercise:** If the query is *friends AND romans AND (NOT countrymen)*, how could we use the freq of *countrymen*?
- **Exercise:** Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?
- **Hint:** Begin with the case of a Boolean *formula* query: in this, each query term appears only once in the query.

Exercise

- Try the search feature at
<http://www.rhymezone.com/shakespeare/>
- Write down five search features you think it could do better

Introduction to
Information Retrieval

Phrase queries and positional indexes

Phrase queries

- We want to be able to answer queries such as “***stanford university***” – as a phrase
- Thus the sentence “*I went to university at Stanford*” is not a match.
 - The concept of phrase queries has proven easily understood by users; one of the few “advanced search” ideas that works
 - Many more queries are *implicit phrase queries*
- For this, it no longer suffices to store only $\langle \text{term} : \text{docs} \rangle$ entries

A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
 - *friends romans*
 - *romans countrymen*
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

Longer phrase queries

- Longer phrases can be processed by breaking them down
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:

stanford university AND university palo AND palo alto

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Can have false positives!

Issues for biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary
 - Infeasible for more than biwords, big even for them
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Solution 2: Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

<***term***, number of docs containing ***term***;

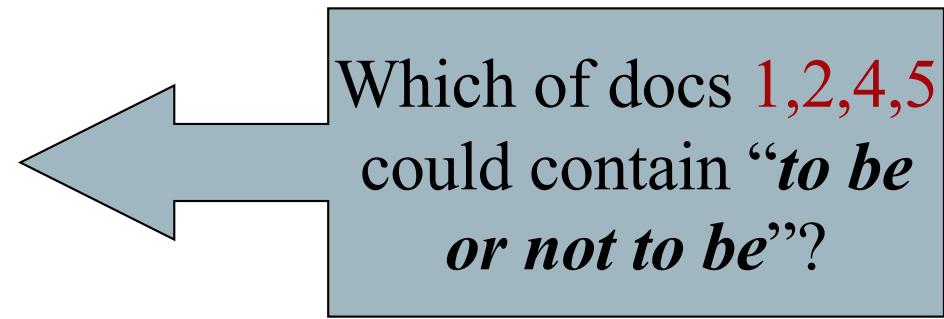
doc1: position1, position2 ... ;

doc2: position1, position2 ... ;

etc.>

Positional index example

<**be**: 993427;
1: 7, 18, 33, 72, 86, 231;
2: 3, 149;
4: 17, 191, 291, 430, 434;
5: 363, 367, ...>



Which of docs **1,2,4,5** could contain “**to be or not to be**”?

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality

Processing a phrase query

- Extract inverted index entries for each distinct term:
to, be, or, not.
- Merge their *doc:position* lists to enumerate all positions with “***to be or not to be***”.
 - ***to:***
 - 2:1,17,74,222,551; **4:8,16,190,429,433;** 7:13,23,191; ...
 - ***be:***
 - 1:17,19; **4:17,191,291,430,434;** 5:14,19,101; ...
- Same general method for proximity searches

Proximity queries

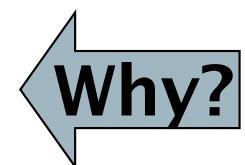
- **LIMIT! /3 STATUTE /3 FEDERAL /2 TORT**
 - Again, here, $/k$ means “within k words of”.
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of k ?
 - This is a little tricky to do correctly and efficiently
 - See Figure 2.12 of *IIR*

Positional index size

- A positional index expands postings storage *substantially*
 - Even though indices can be compressed
- Nevertheless, a positional index is now standardly used because of the power and usefulness of phrase and proximity queries ... whether used explicitly or implicitly in a ranking retrieval system.

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on average document size
 - Average web page has <1000 terms
 - SEC filings, books, even some epic poems ... easily 100,000 terms
- Consider a term with frequency 0.1%



Document size	Postings	Positional postings
1000	1	1
100,000	1	100

Rules of thumb

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of volume of original text
 - Caveat: all of this holds for “English-like” languages

Combination schemes

- These two approaches can be profitably combined
 - For particular phrases (“*Michael Jackson*”, “*Britney Spears*”) it is inefficient to keep on merging positional postings lists
 - Even more so for phrases like “*The Who*”
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
 - A typical web query mixture was executed in $\frac{1}{4}$ of the time of using just a positional index
 - It required 26% more space than having a positional index alone

Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search

Christopher Manning and Pandu Nayak

Wildcard queries and Spelling Correction

WILD-CARD QUERIES

Wild-card queries: *

- ***mon****: find all docs containing any word beginning with “mon”.
- Easy with binary tree (or B-tree) dictionary: 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***.

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

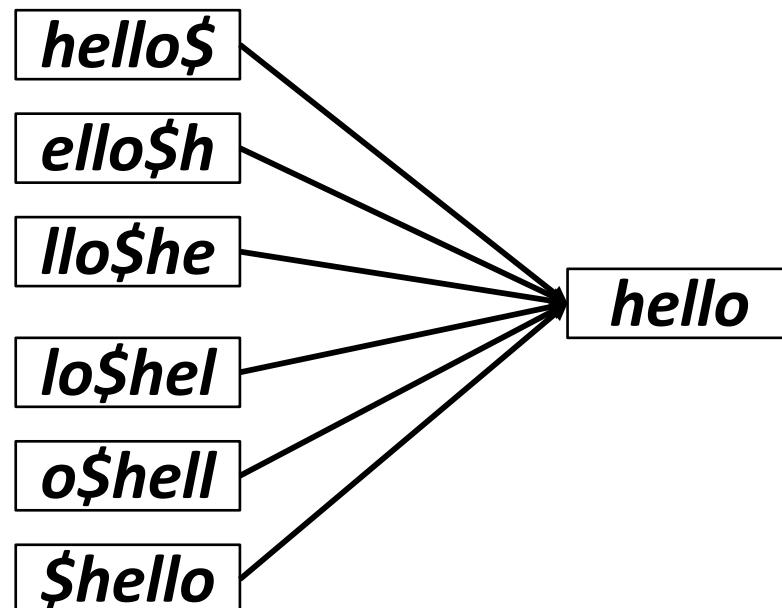
This may result in the execution of many Boolean *AND* queries.

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

- Add a \$ to the end of each term
- Rotate the resulting term and index them in a B-tree
- For term *hello*, index under:
 - *hello\$*, *ello\$h*, *llo\$he*, *lo\$hel*, *o\$hell*, *\$hello*where \$ is a special symbol.



Empirically, dictionary quadruples in size

Permuterm query processing

- (Add \$), rotate * to end, lookup in permuterm index
- Queries:
 - **X** lookup on **X\$** **hello\$** for **hello**
 - **X*** lookup on **\$X*** **\$hel*** for **hel***
 - ***X** lookup on **X\$*** **llo\$*** for ***llo**
 - ***X*** lookup on **X*** **ell*** for ***ell***
 - **X*Y** lookup on **Y\$X*** **lo\$h** for **h*lo**
 - **X*Y*Z** treat as a search for **X*Z** and post-filter
For **h*a*o**, search for **h*o** by looking up **o\$h***
and post-filter **hello** and retain **halo**

Bigram (k -gram) indexes

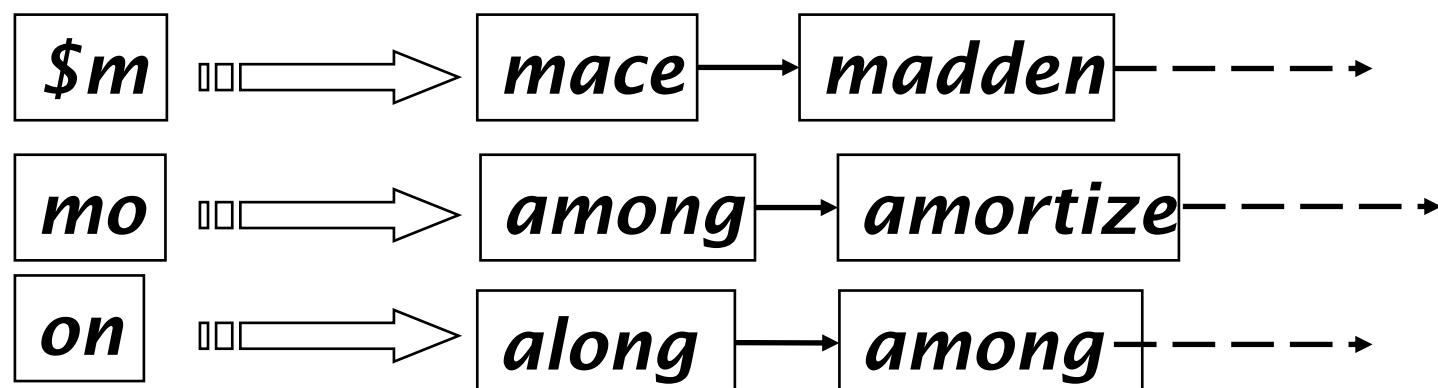
- Enumerate all k -grams (sequence of k chars) occurring in any term
- e.g., from text “***April is the cruelest month***” we get the 2-grams (*bigrams*)

```
$a,ap,pr,ri,il,I$, $i,is,s$, $t,th,he,e$, $c,cr,ru,  
ue,el,le,es,st,t$, $m,mo,on,nt,h$
```

- \$ is a special word boundary symbol
- Maintain a second inverted index from bigrams to dictionary terms that match each bigram.

Bigram index example

- The k -gram index finds *terms* based on a query consisting of k -grams (here $k=2$).



Processing wild-cards

- Query ***mon**** can now be run as
 - **\$m AND mo AND on**
- Gets terms that match AND version of our wildcard query.
- But we'd enumerate ***moon***.
- Must post-filter these terms against query.
- Surviving enumerated terms are then looked up in the term-document inverted index.
- Fast, space efficient (compared to permuterm).

Processing wild-card queries

- As before, we must execute a Boolean query for each enumerated, filtered term.
- Wild-cards can result in expensive query execution (very large disjunctions...)
 - pyth* AND prog*
- If you encourage “laziness” people will respond!

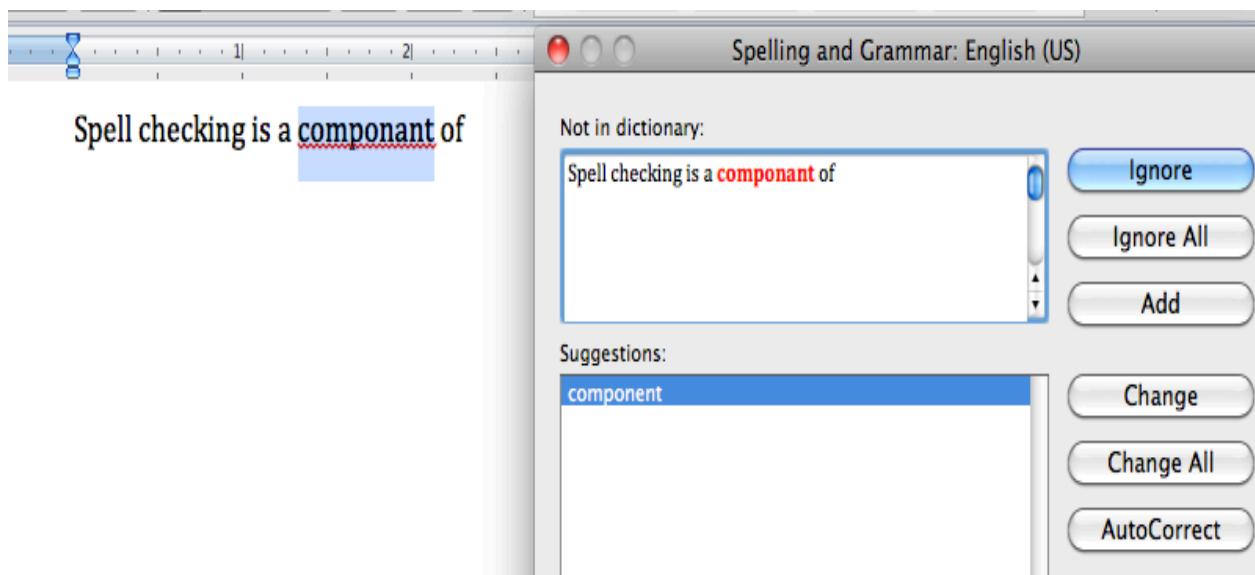
Search

Type your search terms, use '*' if you need to.
E.g., Alex* will match Alexander.

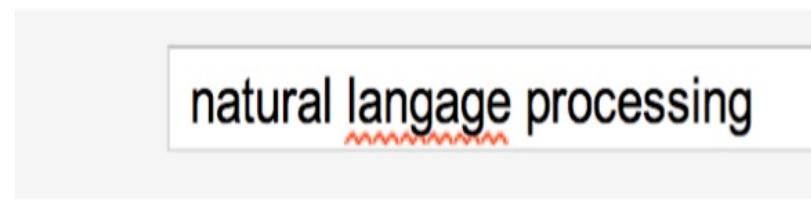
SPELLING CORRECTION

Applications for spelling correction

Word processing



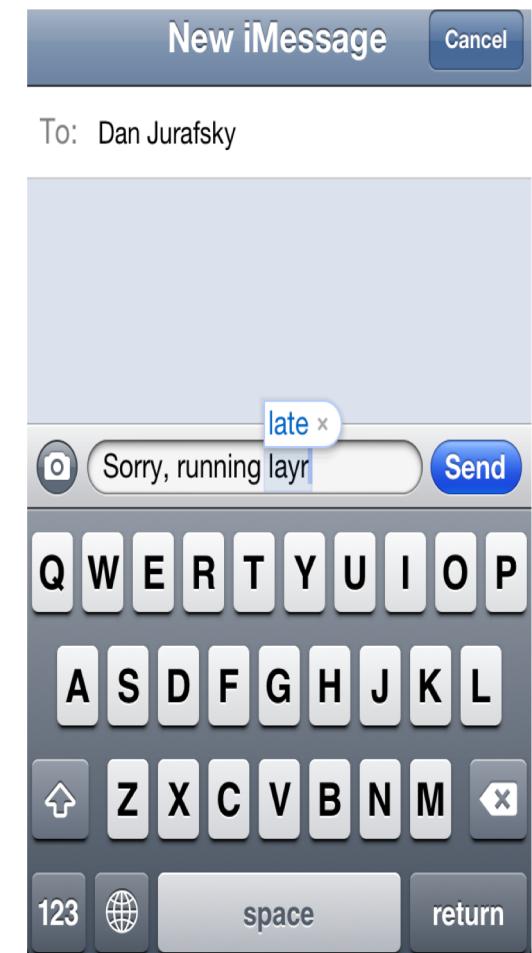
Web search



Showing results for natural *language* processing

Search instead for natural langage processing

Phones



Rates of spelling errors

Depending on the application, ~1–20% error rates

26%: Web queries [Wang et al. 2003](#)

13%: Retyping, no backspace: [Whitelaw et al. English&German](#)

7%: Words corrected retyping on phone-sized organizer

2%: Words uncorrected on organizer [Soukoreff &MacKenzie 2003](#)

1-2%: Retyping: [Kane and Wobbrock 2007, Gruden et al. 1983](#)

Spelling Tasks

- Spelling Error Detection
- Spelling Error Correction:
 - Autocorrect
 - hte → the
 - Suggest a correction
 - Suggestion lists

Types of spelling errors

- Non-word Errors
 - *graffe* → *giraffe*
- Real-word Errors
 - Typographical errors
 - *three* → *there*
 - Cognitive Errors (homophones)
 - *piece* → *peace*,
 - *too* → *two*
 - *your* → *you're*
- Non-word correction was historically mainly context insensitive
- Real-word correction almost needs to be context sensitive

Non-word spelling errors

- Non-word spelling error detection:
 - Any word not in a ***dictionary*** is an error
 - The larger the dictionary the better ... up to a point
 - (The Web is full of mis-spellings, so the Web isn't necessarily a great dictionary ...)
- Non-word spelling error correction:
 - Generate ***candidates***: real words that are similar to error
 - Choose the one which is best:
 - Shortest weighted edit distance
 - Highest noisy channel probability

Real word & non-word spelling errors

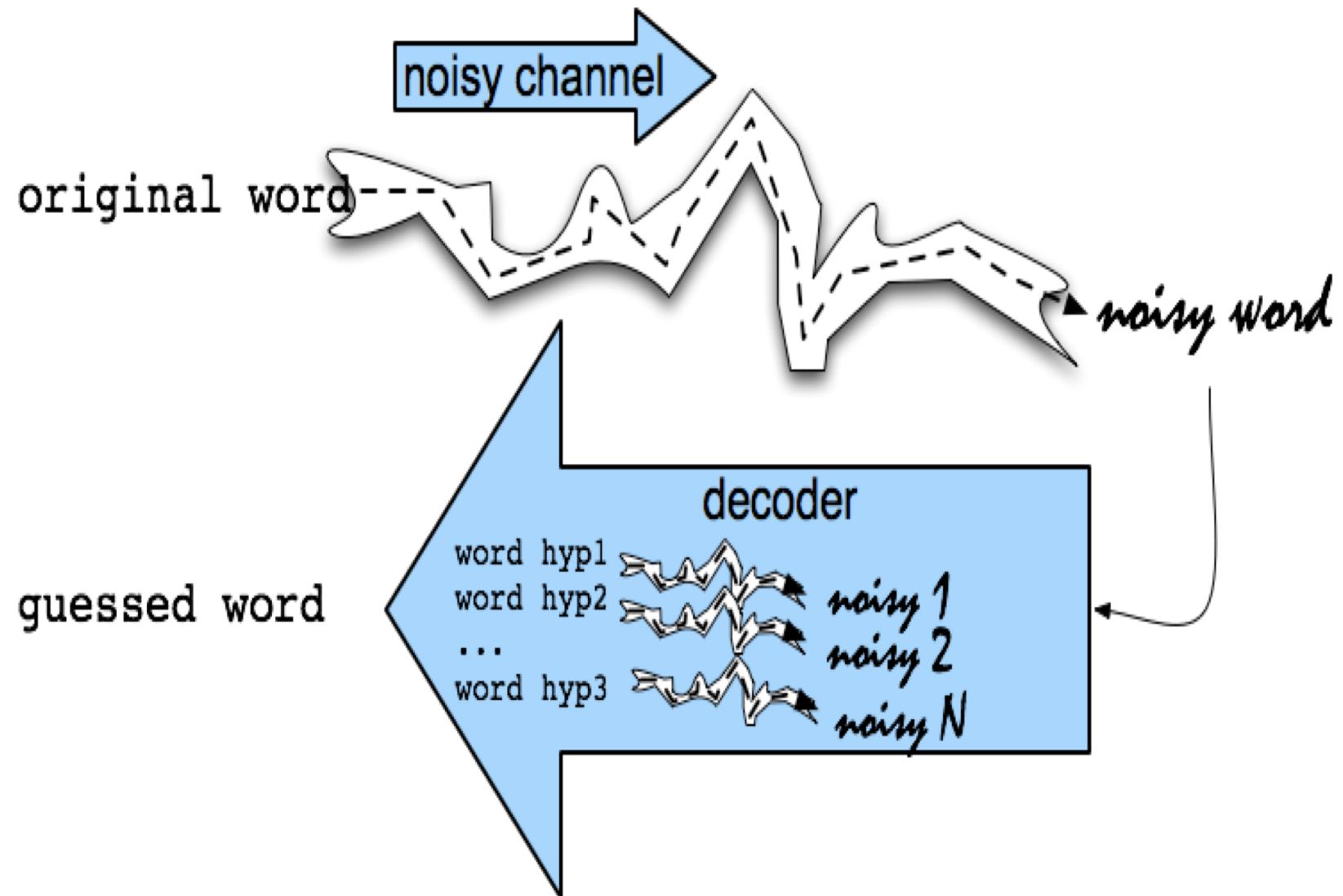
- For each word w , generate candidate set:
 - Find candidate words with similar *pronunciations*
 - Find candidate words with similar *spellings*
 - Include w in candidate set
- Choose best candidate
 - Noisy Channel view of spell errors
 - Context-sensitive – so have to consider whether the surrounding words “make sense”
 - *Flying form Heathrow to LAX → Flying from Heathrow to LAX*

Terminology

- We just discussed character bigrams and k-grams:
 - *st, pr, an ...*
- We can also have word bigrams and n-grams:
 - *palo alto, flying from, road repairs*

The Noisy Channel Model of Spelling **INDEPENDENT WORD SPELLING CORRECTION**

Noisy Channel Intuition



Noisy Channel = Bayes' Rule

- We see an observation x of a misspelled word
- Find the correct word \hat{w}

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w | x)$$

$$= \operatorname{argmax}_{w \in V} \frac{P(x | w)P(w)}{P(x)}$$

$$= \operatorname{argmax}_{w \in V} P(x | w)P(w)$$

↑
Noisy channel model

Prior



Bayes

History: Noisy channel for spelling proposed around 1990

- **IBM**
 - Mays, Eric, Fred J. Damerau and Robert L. Mercer. 1991. Context based spelling correction. *Information Processing and Management*, 23(5), 517–522
- **AT&T Bell Labs**
 - Kernighan, Mark D., Kenneth W. Church, and William A. Gale. 1990. [A spelling correction program based on a noisy channel model](#). Proceedings of COLING 1990, 205-210

Non-word spelling error example

acress

Candidate generation

- Words with similar spelling
 - Small *edit distance* to error
- Words with similar pronunciation
 - Small distance of pronunciation to error

Candidate Testing: Damerau-Levenshtein edit distance

- Minimal edit distance between two strings, where edits are:
 - Insertion
 - Deletion
 - Substitution
 - Transposition of two adjacent letters
- See *IIR* sec 3.3.3 for edit distance

Words within 1 of acress

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	–	deletion
acress	cress	–	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	–	s	insertion

Candidate generation

- 80% of errors are within edit distance 1
- Almost all errors within edit distance 2
- Also allow insertion of **space** or **hyphen**
 - `thisidea` → `this idea`
 - `inlaw` → `in-law`
- Can also allow merging words
 - `data base` → `database`
 - For short texts like a query, can just regard whole string as one item from which to produce edits

How do you generate the candidates?

1. Run through dictionary, check edit distance with each word
2. Generate all words within edit distance $\leq k$ (e.g., $k = 1$ or 2) and then intersect them with dictionary
3. Use a character k -gram index and find dictionary words that share “most” k -grams with word (e.g., by Jaccard coefficient)
 - see *IIR* sec 3.3.4
4. Compute them fast with a Levenshtein finite state transducer
5. Have a precomputed map of words to possible corrections

A paradigm ...

- We want the best spell corrections
- Instead of finding the very best, we
 - Find a subset of pretty good corrections
 - (say, edit distance at most 2)
 - Find the best amongst them
- *These may not be the actual best*
- This is a recurring paradigm in IR including finding the best docs for a query, best answers, best ads ...
 - Find a good candidate set
 - Find the top K amongst them and return them as the best

Let's say we've generated candidates: Now back to Bayes' Rule

- We see an observation x of a misspelled word
- Find the correct word \hat{w}

$$\hat{w} = \operatorname{argmax}_{w \in V} P(w | x)$$

$$= \operatorname{argmax}_{w \in V} \frac{P(x | w)P(w)}{P(x)}$$

$$= \operatorname{argmax}_{w \in V} P(x | w)P(w)$$

What's $P(w)$?

Language Model

- Take a big supply of words (your document collection with T tokens); let $C(w)$ = # occurrences of w

$$P(w) = \frac{C(w)}{T}$$

- In other applications – you can take the supply to be typed queries (suitably filtered) – when a static dictionary is inadequate

Unigram Prior probability

Counts from 404,253,213 words in Corpus of Contemporary English (COCA)

word	Frequency of word	$P(w)$
actress	9,321	.0000230573
cress	220	.0000005442
caress	686	.0000016969
access	37,038	.0000916207
across	120,844	.0002989314
acres	12,874	.0000318463

Channel model probability

- **Error model probability, Edit probability**
- *Kernighan, Church, Gale 1990*
- *Misspelled word $x = x_1, x_2, x_3 \dots x_m$*
- *Correct word $w = w_1, w_2, w_3, \dots, w_n$*
- $P(x/w)$ = probability of the edit
 - (deletion/insertion/substitution/transposition)

Computing error probability: confusion “matrix”

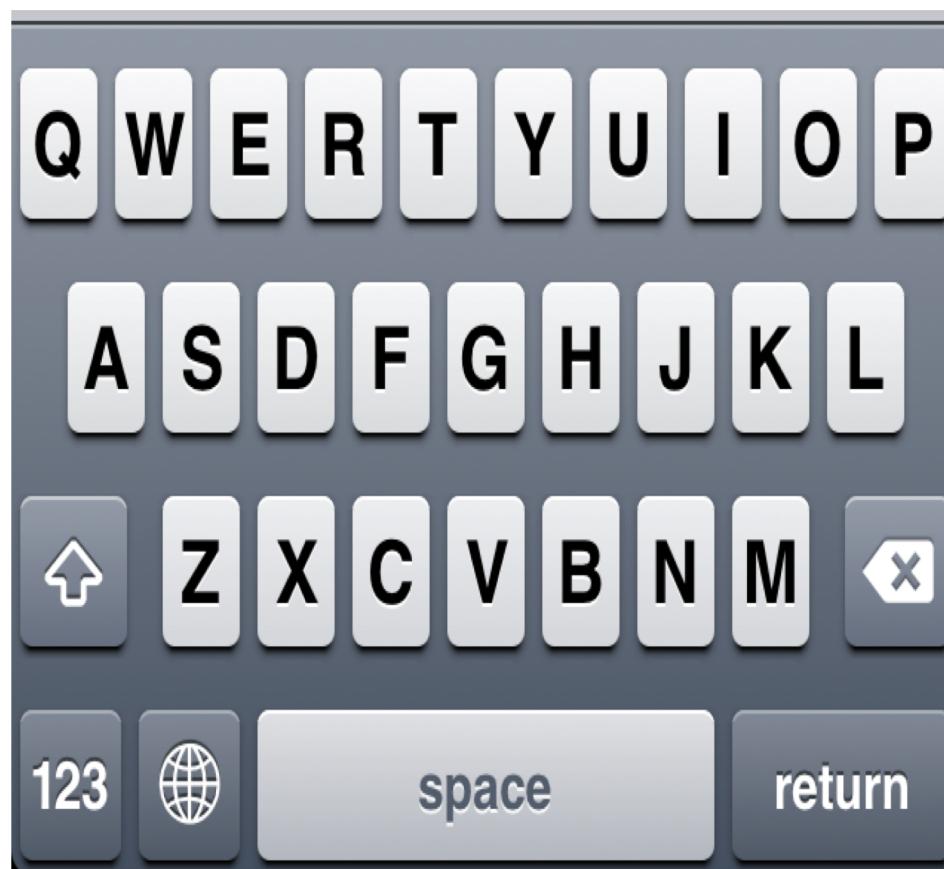
del[x,y]:	count(xy typed as x)
ins[x,y]:	count(x typed as xy)
sub[x,y]:	count(y typed as x)
trans[x,y]:	count(xy typed as yx)

Insertion and deletion conditioned on previous character

Confusion matrix for substitution

X	sub[X, Y] = Substitution of X (incorrect) for Y (correct)																									
	Y (correct)																									
a	0	0	7	1	342	0	0	2	118	0	1	0	0	3	76	0	0	1	35	9	9	0	1	0	5	0
b	0	0	9	9	2	2	3	1	0	0	0	5	11	5	0	10	0	0	2	1	0	0	8	0	0	0
c	6	5	0	16	0	9	5	0	0	0	1	0	7	9	1	10	2	5	39	40	1	3	7	1	1	0
d	1	10	13	0	12	0	5	5	0	0	2	3	7	3	0	1	0	43	30	22	0	0	4	0	2	0
e	388	0	3	11	0	2	2	0	89	0	0	3	0	5	93	0	0	14	12	6	15	0	1	0	18	0
f	0	15	0	3	1	0	5	2	0	0	0	3	4	1	0	0	0	6	4	12	0	0	2	0	0	0
g	4	1	11	11	9	2	0	0	0	1	1	3	0	0	2	1	3	5	13	21	0	0	1	0	3	0
h	1	8	0	3	0	0	0	0	0	0	2	0	12	14	2	3	0	3	1	11	0	0	2	0	0	0
i	103	0	0	0	146	0	1	0	0	0	0	6	0	0	49	0	0	0	2	1	47	0	2	1	15	0
j	0	1	1	9	0	0	1	0	0	0	0	2	1	0	0	0	0	0	5	0	0	0	0	0	0	0
k	1	2	8	4	1	1	2	5	0	0	0	0	5	0	2	0	0	0	6	0	0	0	4	0	0	3
l	2	10	1	4	0	4	5	6	13	0	1	0	0	14	2	5	0	11	10	2	0	0	0	0	0	0
m	1	3	7	8	0	2	0	6	0	0	4	4	0	180	0	6	0	0	9	15	13	3	2	2	3	0
n	2	7	6	5	3	0	1	19	1	0	4	35	78	0	0	7	0	28	5	7	0	0	1	2	0	2
o	91	1	1	3	116	0	0	0	25	0	2	0	0	0	0	14	0	2	4	14	39	0	0	0	18	0
p	0	11	1	2	0	6	5	0	2	9	0	2	7	6	15	0	0	1	3	6	0	4	1	0	0	0
q	0	0	1	0	0	0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
r	0	14	0	30	12	2	2	8	2	0	5	8	4	20	1	14	0	0	12	22	4	0	0	1	0	0
s	11	8	27	33	35	4	0	1	0	1	0	27	0	6	1	7	0	14	0	15	0	0	5	3	20	1
t	3	4	9	42	7	5	19	5	0	1	0	14	9	5	5	6	0	11	37	0	0	2	19	0	7	6
u	20	0	0	0	44	0	0	0	64	0	0	0	0	2	43	0	0	4	0	0	0	0	2	0	8	0
v	0	0	7	0	0	3	0	0	0	0	0	1	0	0	1	0	0	0	8	3	0	0	0	0	0	0
w	2	2	1	0	1	0	0	2	0	0	1	0	0	0	0	7	0	6	3	3	1	0	0	0	0	0
x	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0
y	0	0	2	0	15	0	1	7	15	0	0	0	2	0	6	1	0	7	36	8	5	0	0	1	0	0
z	0	0	0	7	0	0	0	0	0	0	0	7	5	0	0	0	2	21	3	0	0	0	0	3	0	

Nearby keys



Generating the confusion matrix

- Peter Norvig's list of errors
- Peter Norvig's list of counts of single-edit errors
 - All Peter Norvig's ngrams data links: <http://norvig.com/ngrams/>

Channel model

Kernighan, Church, Gale 1990

$$P(x|w) = \begin{cases} \frac{\text{del}[w_{i-1}, w_i]}{\text{count}[w_{i-1} w_i]}, & \text{if deletion} \\ \frac{\text{ins}[w_{i-1}, x_i]}{\text{count}[w_{i-1}]}, & \text{if insertion} \\ \frac{\text{sub}[x_i, w_i]}{\text{count}[w_i]}, & \text{if substitution} \\ \frac{\text{trans}[w_i, w_{i+1}]}{\text{count}[w_i w_{i+1}]}, & \text{if transposition} \end{cases}$$

Smoothing probabilities: Add-1 smoothing

- But if we use the confusion matrix example, unseen errors are impossible!
- They'll make the overall probability 0. That seems too harsh
 - e.g., in Kernighan's chart $q \rightarrow a$ and $a \rightarrow q$ are both 0, even though they're adjacent on the keyboard!
- A simple solution is to add 1 to all counts and then if there is a $|A|$ character alphabet, to normalize appropriately:

$$\text{If substitution, } P(x|w) = \frac{\text{sub}[x,w]+1}{\text{count}[w]+A}$$

Channel model for acress

Candidate Correction	Correct Letter	Error Letter	$x w$	$P(x w)$
actress	t	-	c ct	.000117
cress	-	a	a #	.00000144
caress	ca	ac	ac ca	.00000164
access	c	r	r c	.000000209
across	o	e	e o	.0000093
acres	-	s	es e	.0000321
acres	-	s	ss s	.0000342

Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$	$P(w)$	$10^9 * P(x/w) * P(w)$
actress	t	-	c ct	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	.00078
caress	ca	ac	ac c a	.00000164	.00000170	.0028
access	c	r	r c	.000000209	.0000916	.019
across	o	e	e o	.0000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss s	.0000342	.0000318	1.0 ⁴²

Candidate Correction	Correct Letter	Error Letter	x/w	$P(x/w)$	$P(w)$	10^9 $*P(x/w)P(w)$
actress	t	-	c c t	.000117	.0000231	2.7
cress	-	a	a #	.00000144	.000000544	.00078
caress	ca	ac	ac ca	.00000164	.00000170	.0028
access	c	r	r c	.000000209	.0000916	.019
across	o	e	e o	.0000093	.000299	2.8
acres	-	s	es e	.0000321	.0000318	1.0
acres	-	s	ss	.0000342	.0000318	1.0 ⁴³

Evaluation

- Some spelling error test sets
 - [Wikipedia's list of common English misspelling](#)
 - [Aspell filtered version of that list](#)
 - [Birkbeck spelling error corpus](#)
 - [Peter Norvig's list of errors \(includes Wikipedia and Birkbeck, for training or testing\)](#)

Context-Sensitive Spelling Correction

SPELLING CORRECTION WITH THE NOISY CHANNEL

Real-word spelling errors

- ...leaving in about fifteen **minuets** to go to her house.
- The design **an** construction of the system...
- Can they **lave** him my messages?
- The study was conducted mainly **be** John Black.
- 25-40% of spelling errors are real words **Kukich 1992**

Context-sensitive spelling error fixing

- For each word in sentence (phrase, query ...)
 - Generate *candidate set*
 - the word itself
 - all single-letter edits that are English words
 - words that are homophones
 - (all of this can be pre-computed!)
 - Choose best candidates
 - Noisy channel model

Noisy channel for real-word spell correction

- Given a sentence $x_1, x_2, x_3, \dots, x_n$
- Generate a set of candidates for each word x_i
 - Candidate(x_1) = $\{x_1, w_1, w'_1, w''_1, \dots\}$
 - Candidate(x_2) = $\{x_2, w_2, w'_2, w''_2, \dots\}$
 - Candidate(x_n) = $\{x_n, w_n, w'_n, w''_n, \dots\}$
- Choose the sequence W that maximizes $P(W | x_1, \dots, x_n)$

$$\begin{aligned}\hat{w} &= \operatorname{argmax}_{w \in V} P(w | x) \\ &= \operatorname{argmax}_{w \in V} P(x | w)P(w)\end{aligned}$$

Incorporating context words: Context-sensitive spelling correction

- Determining whether **actress** or **across** is appropriate will require looking at the context of use
- We can do this with a better **language model**
 - You learned/can learn a lot about language models in CS124 or CS224N
 - Here we present just enough to be dangerous/do the assignment
- A **bigram language model** conditions the probability of a word on (just) the previous word

$$P(w_1 \dots w_n) = P(w_1)P(w_2 | w_1) \dots P(w_n | w_{n-1})$$

Incorporating context words

- For unigram counts, $P(w)$ is always non-zero
 - if our dictionary is derived from the document collection
- This won't be true of $P(w_k | w_{k-1})$. We need to **smooth**
- We could use add-1 smoothing on this conditional distribution
- But here's a better way – interpolate a unigram and a bigram:

$$P_{\text{li}}(w_k | w_{k-1}) = \lambda P_{\text{uni}}(w_k) + (1-\lambda)P_{\text{bi}}(w_k | w_{k-1})$$

- $P_{\text{bi}}(w_k | w_{k-1}) = C(w_{k-1}, w_k) / C(w_{k-1})$

All the important fine points

- Note that we have several probability distributions for words
 - Keep them straight!
- You might want/need to work with log probabilities:
 - $\log P(w_1 \dots w_n) = \log P(w_1) + \log P(w_2 | w_1) + \dots + \log P(w_n | w_{n-1})$
 - Otherwise, be very careful about floating point underflow
- Our query may be words anywhere in a document
 - We'll start the bigram estimate of a sequence with a unigram estimate
 - Often, people instead condition on a start-of-sequence symbol, but not good here
 - Because of this, the unigram and bigram counts have different totals – not a problem

Using a bigram language model

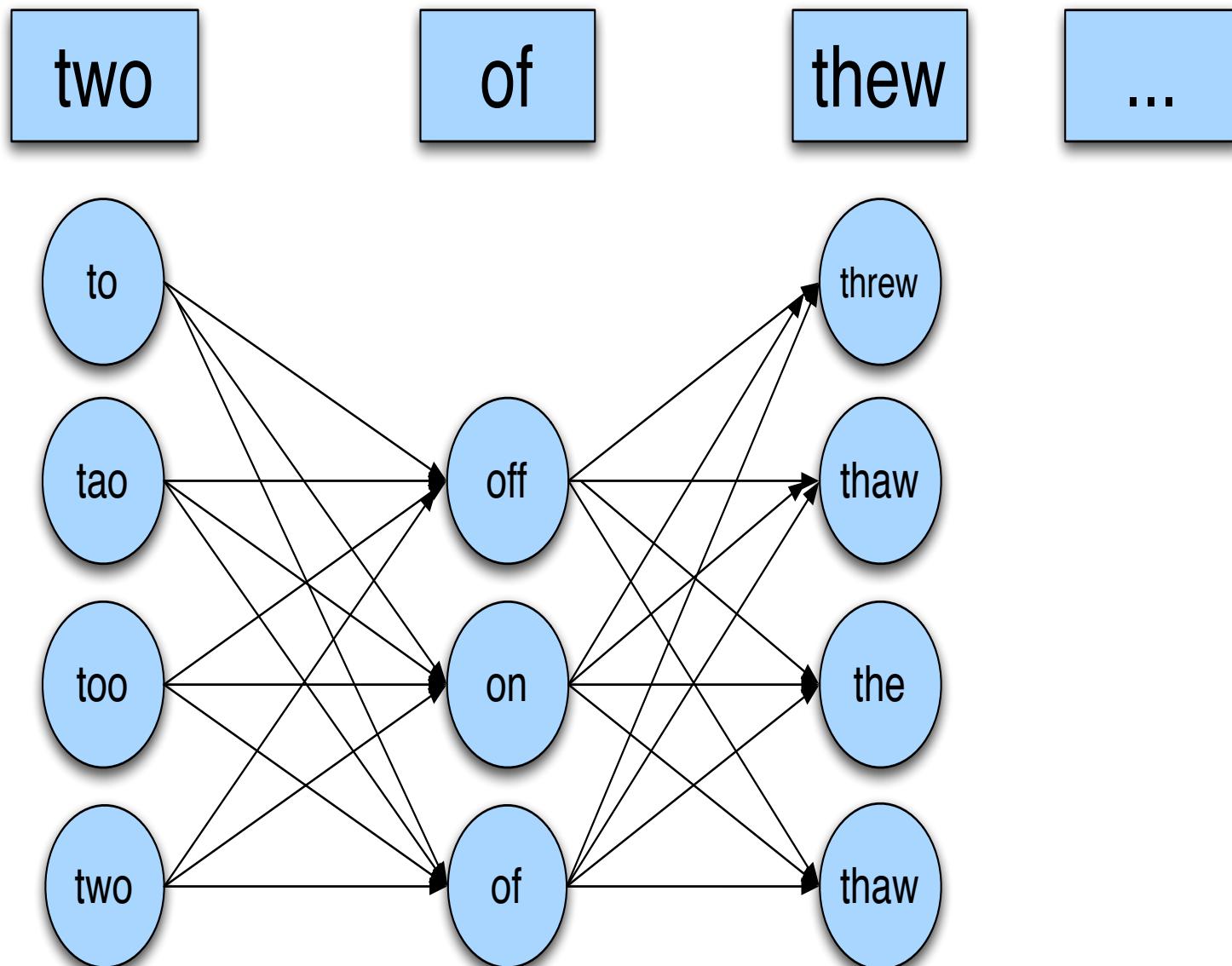
- “**a stellar and versatile across whose combination of sass and glamour...**”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress} \mid \text{versatile}) = .000021$
- $P(\text{across} \mid \text{versatile}) = .000021$
- $P(\text{whose} \mid \text{actress}) = .0010$
- $P(\text{whose} \mid \text{across}) = .000006$
- $P(\text{"versatile actress whose"}) = .000021 * .0010 = 210 \times 10^{-10}$
- $P(\text{"versatile across whose"}) = .000021 * .000006 = 1 \times 10^{-10}$

Using a bigram language model

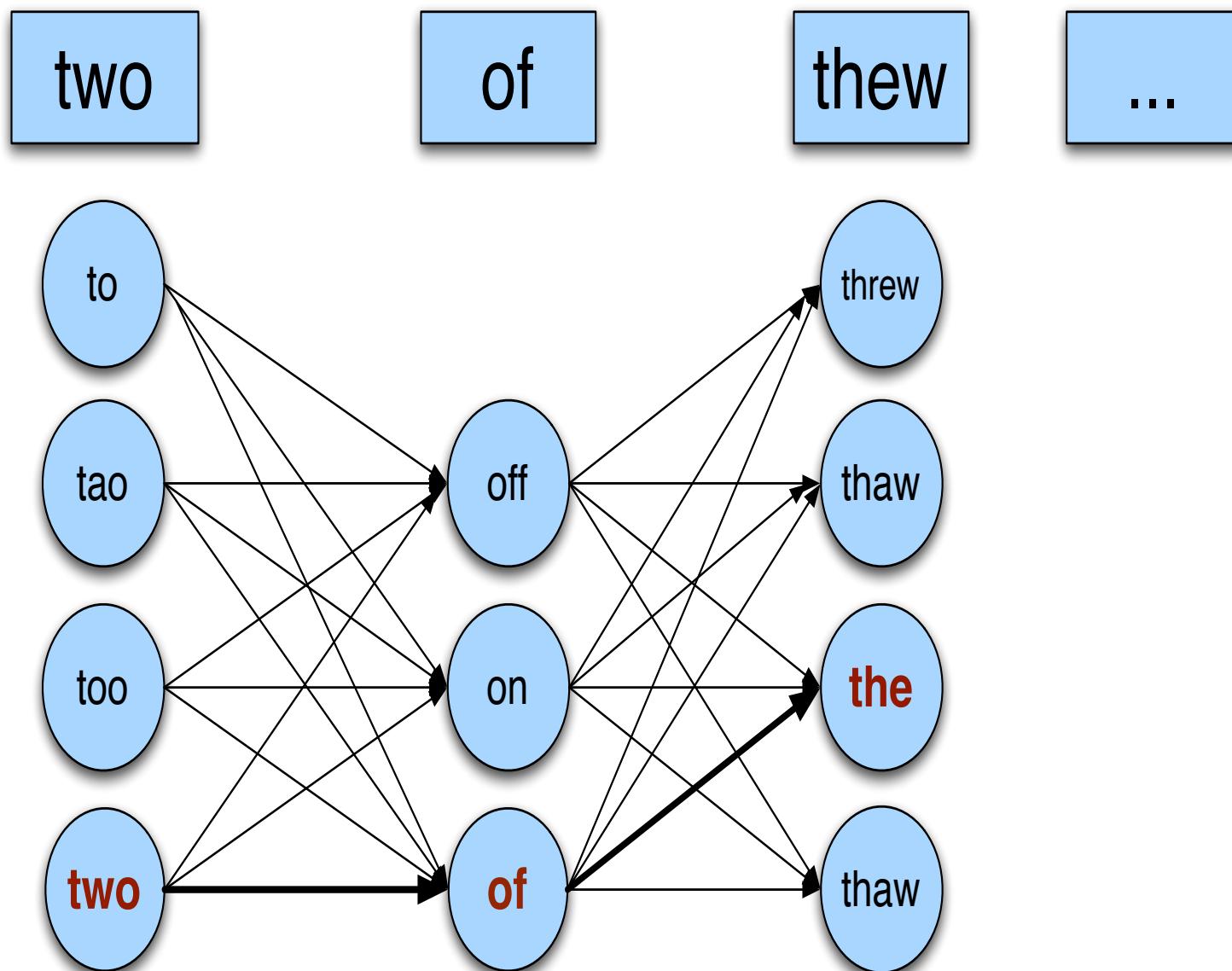
- “**a stellar and versatile across whose combination of sass and glamour...**”
- Counts from the Corpus of Contemporary American English with add-1 smoothing
- $P(\text{actress} \mid \text{versatile}) = .000021$ $P(\text{whose} \mid \text{actress}) = .0010$
- $P(\text{across} \mid \text{versatile}) = .000021$ $P(\text{whose} \mid \text{across}) = .000006$

- $P(\text{"versatile actress whose"}) = .000021 * .0010 = 210 \times 10^{-10}$
- $P(\text{"versatile across whose"}) = .000021 * .000006 = 1 \times 10^{-10}$

Noisy channel for real-word spell correction



Noisy channel for real-word spell correction



Simplification: One error per sentence

- Out of all possible sentences with one word replaced
 - w_1, w''_2, w_3, w_4 two off thew
 - w_1, w_2, w'_3, w_4 two of the
 - w'''_1, w_2, w_3, w_4 too of thew
 - ...
- Choose the sequence W that maximizes $P(W)$

Where to get the probabilities

- Language model
 - Unigram
 - Bigram
 - etc.
- Channel model
 - Same as for non-word spelling correction
 - Plus need probability for no error, $P(w/w)$

Probability of no error

- What is the channel probability for a correctly typed word?
- $P(\text{"the"} | \text{"the"})$
 - If you have a big corpus, you can estimate this percent correct
- But this value depends strongly on the application
 - .90 (1 error in 10 words)
 - .95 (1 error in 20 words)
 - .99 (1 error in 100 words)

Peter Norvig's “thew” example

x	w	x w	$P(x w)$	$P(w)$	$10^9 P(x w)P(w)$
thew	the	ew e	0.000007	0.02	144
thew	thew		0.95	0.0000009	90
thew	thaw	e a	0.001	0.0000007	0.7
thew	threw	h hr	0.000008	0.000004	0.03
thew	thwe	ew w	0.000003	0.00000004	0.0001

State of the art noisy channel

- We never just multiply the prior and the error model
- Independence assumptions → probabilities not commensurate
- Instead: Weight them

$$\hat{w} = \operatorname{argmax}_{w \in V} P(x | w) P(w)^\lambda$$

- Learn λ from a development test set

Improvements to channel model

- Allow richer edits [\(Brill and Moore 2000\)](#)
 - ent → ant
 - ph → f
 - le → al
- Incorporate pronunciation into channel [\(Toutanova and Moore 2002\)](#)
- Incorporate device into channel
 - Not all Android phones need have the same error model
 - But spell correction may be done at the system level

Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 2: The term vocabulary and postings lists

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-09

Overview

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Outline

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Inverted index

For each term t , we store a list of all documents that contain t .

BRUTUS	→	1	2	4	11	31	45	173	174
--------	---	---	---	---	----	----	----	-----	-----

CAESAR	→	1	2	4	5	6	16	57	132	...
--------	---	---	---	---	---	---	----	----	-----	-----

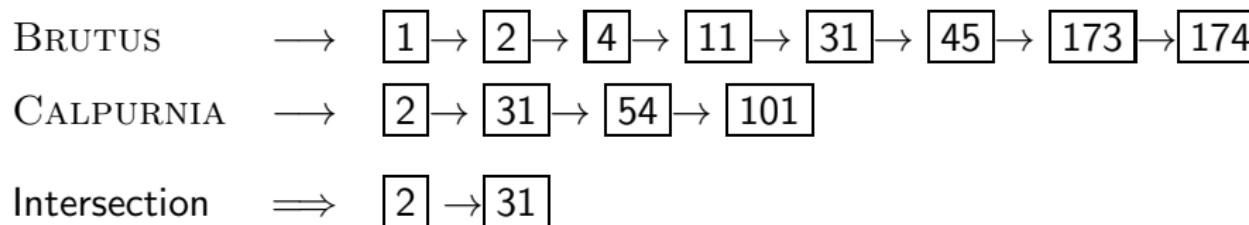
CALPURNIA	→	2	31	54	101
-----------	---	---	----	----	-----

⋮

 **dictionary**

 **postings**

Intersecting two postings lists



Constructing the inverted index: Sort postings

term	docID	term	docID
I	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
I	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	I	1
killed	1	I	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2
brutus	2	the	1
hath	2	the	2
told	2	told	2
you	2	you	2
caesar	2	was	1
was	2	was	2
ambitious	2	with	2



Westlaw: Example queries

Information need: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company *Query:* "trade secret" /s disclos! /s prevent /s employe!

Information need: Requirements for disabled people to be able to access a workplace *Query:* disab! /p access! /s work-site work-place (employment /3 place)

Information need: Cases about a host's responsibility for drunk guests *Query:* host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Does Google use the Boolean model?

- On Google, the default interpretation of a query $[w_1 \ w_2 \dots w_n]$ is $w_1 \text{ AND } w_2 \text{ AND } \dots \text{ AND } w_n$
- Cases where you get hits that do not contain one of the w_i :
 - anchor text
 - page contains variant of w_i (morphology, spelling correction, synonym)
 - long queries (n large)
 - boolean expression generates very few hits
- Simple Boolean vs. Ranking of result set
 - Simple Boolean retrieval returns matching documents in no particular order.
 - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

Take-away

- Understanding of the basic unit of classical information retrieval systems: **words** and **documents**: What is a document, what is a term?
- Tokenization: how to get from raw text to words (or tokens)
- More complex indexes: skip pointers and phrases

Outline

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
 - We know what a document is.
 - We can “machine-read” each document.
- This can be complex in reality.

Parsing a document

- We need to deal with format and language of each document.
- What format is it in? pdf, word, excel, html etc.
- What language is it in?
- What character set is in use?
- Each of these is a classification problem, which we will study later in this course (IIR 13).
- Alternative: use heuristics

Format/Language: Complications

- A single index usually contains terms of several languages.
- Sometimes a document or its components contain multiple languages/formats.
 - French email with Spanish pdf attachment
- What is the document unit for indexing?
- A file?
- An email?
- An email with 5 attachments?
- A group of files (ppt or latex in HTML)?
- Upshot: Answering the question “what is a document?” is not trivial and requires some design decisions.
- Also: XML

Outline

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Outline

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases: an equivalence class of tokens.

Normalization

- Need to “normalize” words in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define **equivalence classes** of terms.
- Alternatively: do asymmetric expansion
 - *window* → *window*, *windows*
 - *windows* → *Windows*, *windows*
 - *Windows* (no expansion)
- More powerful, but less efficient
- Why don't you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?

Normalization: Other languages

- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT ≠ mit

Tokenization: Recall construction of inverted index

- Input:

Friends, Romans, countrymen.

So let it be with Caesar

...

- Output:

friend

roman

countryman

so

...

- Each token is a candidate for a postings entry.

- What are valid tokens to emit?

Exercises

In June, the dog likes to chase the cat in the barn. – How many word tokens? How many word types? Why tokenization is difficult

– even in English. Tokenize: *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*

Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

Numbers

- 3/20/91
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it's a useful feature.
- Google example

Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年4月9日，莎拉波娃在美国第一大城市纽约度过了18岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Ambiguous segmentation in Chinese



The two

characters can be treated as one word meaning 'monk' or as a sequence of two words meaning 'and' and 'still'.

Other cases of “no whitespace”

- Compounds in Dutch, German, Swedish
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuuujunga (I can't hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, ...

Japanese

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるMOTTAINAIキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを800字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて10月20日までにお送りください。大賞受賞者には、50万円相当の旅行券とエコ製品2点の副賞が贈られます。

4

different “alphabets”: Chinese characters, hiragana syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese). End user can express query entirely in hiragana!

Arabic script

كتاب \Leftarrow k̄t̄ab̄
un bā t̄ i k
/kitābun/ ‘a book’

Arabic script: Bidirectionality

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← START

‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

Bidirectionality is not a problem if text is coded in Unicode.

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)

Outline

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
 - capitalized words in mid-sentence
 - MIT vs. mit
 - Fed vs. fed
 - ...
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
- Most web search engines index stop words.

More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Muller = Mueller)
- Thesauri: IIR 9 (semantic equivalence, car = automobile)

Lemmatization

- Reduce inflectional/variant forms to base form
- Example: *am, are, is* → *be*
- Example: *car, cars, car's, cars'* → *car*
- Example: *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form (the [lemma](#)).
- Inflectional morphology (*cutting* → *cut*) vs. derivational morphology (*destruction* → *destroy*)

Stemming

- Definition of stemming: Crude heuristic process that **chops off the ends of words** in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional **and** derivational
- Example for derivational: *automate, automatic, automation* all reduce to *automat*

Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
 - Sample command: Delete final *ement* if what remains is longer than 1 character
 - replacement → replac
 - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter stemmer: A few rules

Rule		Example
SSES	→ SS	caresses → caress
IES	→ I	ponies → poni
SS	→ SS	caress → caress
S	→	cats → cat

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco]
- (equivalence classes: {sweater,sweaters}, {tour,tours})
- Porter Stemmer equivalence class *oper* contains all of *operate*, *operating*, *operates*, *operation*, *operative*, *operatives*, *operational*.
- Queries where stemming hurts: [operational AND research], [operating AND system], [operative AND dentistry]

Exercise: What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

Outline

1 Recap

2 Documents

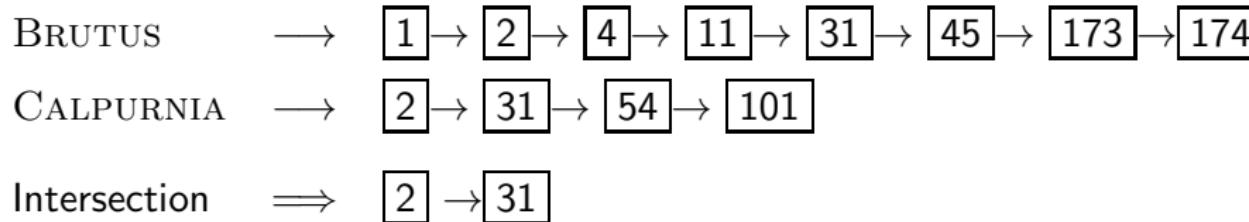
3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Recall basic intersection algorithm

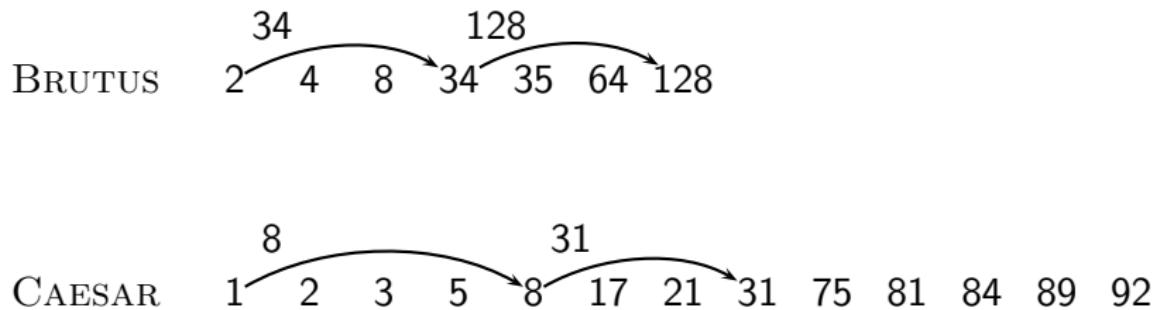


- Linear in the length of the postings lists.
- Can we do better?

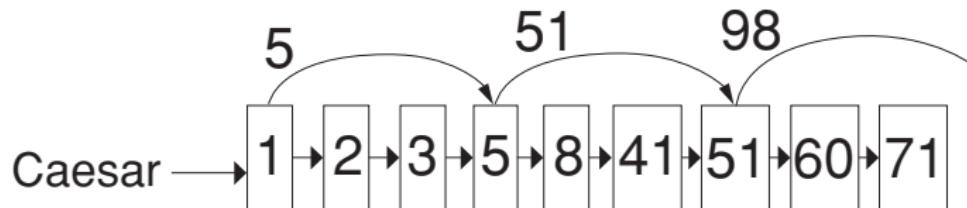
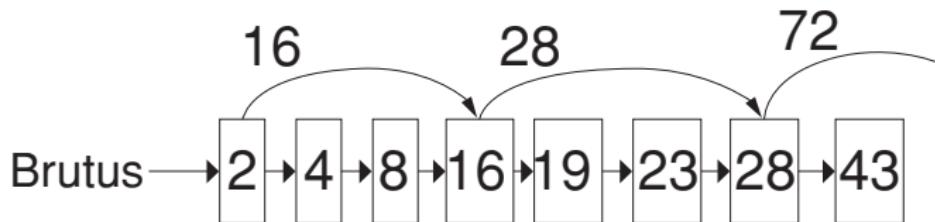
Skip pointers

- Skip pointers allow us to **skip** postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure intersection results are correct?

Basic idea



Skip lists: Larger example



Intersecting with skip pointers

```
INTERSECTWITHSKIPS( $p_1, p_2$ )
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then ADD(answer,  $\text{docID}(p_1)$ )
5       $p_1 \leftarrow \text{next}(p_1)$ 
6       $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then if  $\text{hasSkip}(p_1)$  and ( $\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2)$ )
9          then while  $\text{hasSkip}(p_1)$  and ( $\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2)$ )
10         do  $p_1 \leftarrow \text{skip}(p_1)$ 
11         else  $p_1 \leftarrow \text{next}(p_1)$ 
12     else if  $\text{hasSkip}(p_2)$  and ( $\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1)$ )
13         then while  $\text{hasSkip}(p_2)$  and ( $\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1)$ )
14         do  $p_2 \leftarrow \text{skip}(p_2)$ 
15         else  $p_2 \leftarrow \text{next}(p_2)$ 
16 return answer
```

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
- More skips: Each skip pointer skips only a few items, but we can frequently use it.
- Fewer skips: Each skip pointer skips many items, but we can not use it very often.

Where do we place skips? (cont)

- Simple heuristic: for postings list of length P , use \sqrt{P} evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help a lot.
- With today's fast CPUs, they don't help that much anymore.

Outline

1 Recap

2 Documents

3 Terms

- General + Non-English
- English

4 Skip pointers

5 Phrase queries

Phrase queries

- We want to answer a query such as [stanford university] – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to university* should **not** be a match.
- The concept of phrase query has proven easily understood by users.
- Significant part of web queries are phrase queries (explicitly entered or interpreted as such)
- Consequence for inverted index: it no longer suffices to store docIDs in postings lists.
- Two ways of extending the inverted index:
 - biword index
 - positional index

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- For example, *Friends*, *Romans*, *Countrymen* would generate two biwords: “*friends romans*” and “*romans countrymen*”
- Each of these biwords is now a vocabulary term.
- Two-word phrases can now easily be answered.

Longer phrase queries

- A long phrase like “*stanford university palo alto*” can be represented as the Boolean query “STANFORD UNIVERSITY” AND “UNIVERSITY PALO” AND “PALO ALTO”
- We need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large term vocabulary

Positional indexes

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: each posting is just a docID
- Postings lists in a **positional** index: each posting is a docID and **a list of positions**

Positional indexes: Example

Query: “*to*₁ *be*₂ *or*₃ *not*₄ *to*₅ *be*₆” TO, 993427:

⟨ 1: ⟨ 7, 18, 33, 72, 86, 231⟩;
2: ⟨1, 17, 74, 222, 255⟩;
4: ⟨ 8, 16, 190, 429, 433⟩;
5: ⟨363, 367⟩;
7: ⟨13, 23, 191⟩; ... ⟩

BE, 178239:

⟨ 1: ⟨ 17, 25⟩;
4: ⟨ 17, 191, 291, 430, 434⟩;
5: ⟨14, 19, 101⟩; ... ⟩ Document 4 is a match!

Proximity search

- We just saw how to use a positional index for phrase searches.
- We can also use it for proximity search.
- For example: employment /4 place
- Find all documents that contain EMPLOYMENT and PLACE within 4 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that have learned to adapt now place healthcare workers* is not a hit.

Proximity search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries etc.

“Proximity” intersection

```
POSITIONALINTERSECT( $p_1, p_2, k$ )
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4    then  $I \leftarrow \langle \rangle$ 
5     $pp_1 \leftarrow \text{positions}(p_1)$ 
6     $pp_2 \leftarrow \text{positions}(p_2)$ 
7    while  $pp_1 \neq \text{NIL}$ 
8      do while  $pp_2 \neq \text{NIL}$ 
9        do if  $|\text{pos}(pp_1) - \text{pos}(pp_2)| \leq k$ 
10       then ADD( $I, \text{pos}(pp_2)$ )
11       else if  $\text{pos}(pp_2) > \text{pos}(pp_1)$ 
12         then break
13        $pp_2 \leftarrow \text{next}(pp_2)$ 
14     while  $I \neq \langle \rangle$  and  $|I[0] - \text{pos}(pp_1)| > k$ 
15     do DELETE( $I[0]$ )
16     for each  $ps \in I$ 
17     do ADD(answer,  $\langle \text{docID}(p_1), \text{pos}(pp_1), ps \rangle$ )
18      $pp_1 \leftarrow \text{next}(pp_1)$ 
19      $p_1 \leftarrow \text{next}(p_1)$ 
20      $p_2 \leftarrow \text{next}(p_2)$ 
21   else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
22     then  $p_1 \leftarrow \text{next}(p_1)$ 
23     else  $p_2 \leftarrow \text{next}(p_2)$ 
24 return answer
```

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears etc
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.

“Positional” queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?
- Can you demonstrate on Google that phrase queries are more expensive than Boolean queries?

Take-away

- Understanding of the basic unit of classical information retrieval systems: **words** and **documents**: What is a document, what is a term?
- Tokenization: how to get from raw text to words (or tokens)
- More complex indexes: skip pointers and phrases

Resources

- Chapter 2 of IIR
- Resources at <http://cislmu.org>
 - Porter stemmer
 - A fun number search on Google

Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search

Basic inverted index construction

Index construction

- How do we construct an index?
- What strategies can we use with limited main memory?

Recall index construction

- Documents are parsed to extract words and these are saved with the Document ID.

Doc 1

I did enact Julius
Caesar I was killed
i' the Capitol;
Brutus killed me.

Doc 2

So let it be with
Caesar. The noble
Brutus hath told you
Caesar was ambitious



Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Key step

- After all documents have been parsed, the inverted file is sorted by terms.

We focus on this sort step.

Term	Doc #	Term	Doc #
I	1	ambitious	2
did	1	be	2
enact	1	brutus	1
julius	1	brutus	2
caesar	1	capitol	1
I'	1	caesar	1
was	1	caesar	2
killed	1	caesar	2
i'	1	did	1
the	1	enact	1
capitol	1	hath	1
brutus	1	I	1
killed	1	I	1
me	1	i'	1
so	2	it	2
let	2	julius	1
it	2	killed	1
be	2	killed	1
with	2	let	2
caesar	2	me	1
the	2	noble	2
noble	2	so	2
brutus	2	the	1
hath	2	the	2
told	2	told	2
you	2	you	2
caesar	2	was	1
was	2	was	2
ambitious	2	with	2

RCV1: Our collection for this lecture

- As an example for applying scalable index construction algorithms, we will use the Reuters RCV1 collection.
 - This is one year of Reuters newswire (part of 1995 and 1996)
- The collection isn't really large enough, but it's publicly available and is a plausible example.

A Reuters RCV1 document



You are here: Home > News > Science > Article

Go to a Section: U.S. International Business Markets Politics Entertainment Technology Sports Oddly Enough

Extreme conditions create rare Antarctic clouds

Tue Aug 1, 2006 3:20am ET

[Email This Article](#) | [Print This Article](#) | [Reprints](#)



SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

[+] Text [-]

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.

Reuters RCV1 statistics

■ symbol	statistic	value
■ N	documents	800,000
■ L	avg. # tokens per doc	200
■ M	terms (= word types)	400,000
■	avg. # bytes per token (incl. spaces/punct.)	6
■	avg. # bytes per token (without spaces/punct.)	4.5
■	avg. # bytes per term	7.5
■	non-positional postings	100,000,000

4.5 bytes per word token vs. 7.5 bytes per word type: why?

Sort-based index construction

- As we build the index, we parse docs one at a time.
 - The final postings for any term are incomplete until the end.
- At 8 bytes per $(termID, docID)$, demands a lot of space for large collections.
- $T = 100,000,000$ in the case of RCV1
 - So ... we can do this in memory today, but typical collections are much larger. E.g., the *New York Times* provides an index of >150 years of newswire
- Thus: We need to store intermediate results on disk.

Scaling index construction

- In-memory index construction does not scale
 - Can't stuff entire collection into memory, sort, then write back
- How can we construct an index for very large collections?
- Taking into account hardware constraints. . .
 - Memory, disk, speed, etc.
- Let's review some hardware basics

Hardware basics

- Servers used in IR systems now typically have several GB of main memory, sometimes tens of GB.
- Available disk space is several (2–3) orders of magnitude larger.
- Fault tolerance is very expensive: It's much cheaper to use many regular machines rather than one fault tolerant machine.

Hardware basics

- Access to data in memory is ***much*** faster than access to data on disk.
- Disk seeks: No data is transferred from disk while the disk head is being positioned.
- Therefore: Transferring one large chunk of data from disk to memory is faster than transferring many small chunks.
- Disk I/O is block-based: Reading and writing of entire blocks (as opposed to smaller chunks).
- Block sizes: 8KB to 256 KB.

Hardware assumptions (circa 2007)

■ symbol	statistic	value
■ s	average seek time	$5 \text{ ms} = 5 \times 10^{-3} \text{ s}$
■ b	transfer time per byte	$0.02 \mu\text{s} = 2 \times 10^{-8} \text{ s}$
■	processor's clock rate	10^9 s^{-1}
■ p	low-level operation (e.g., compare & swap a word)	$0.01 \mu\text{s} = 10^{-8} \text{ s}$
■	size of main memory	several GB
■	size of disk space	1 TB or more

Sort using disk as “memory”?

- Can we use the same index construction algorithm for larger collections, but by using disk instead of memory?
- No: Sorting $T = 100,000,000$ records on disk is too slow – too many disk seeks.
- We need an *external* sorting algorithm.

Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search

External memory indexing

BSBI: Blocked sort-based Indexing (Sorting with fewer disk seeks)

- 8-byte records (*termID, docID*)
- These are generated as we parse docs
- Must now sort 100M such 8-byte records by *termID*
- Define a Block $\sim 10M$ such records
 - Can easily fit a couple into memory
 - Will have 10 such blocks to start with
- Basic idea of algorithm:
 - Accumulate postings for each block, sort, write to disk
 - Then merge the blocks into one long sorted order

BSBINDEXCONSTRUCTION()

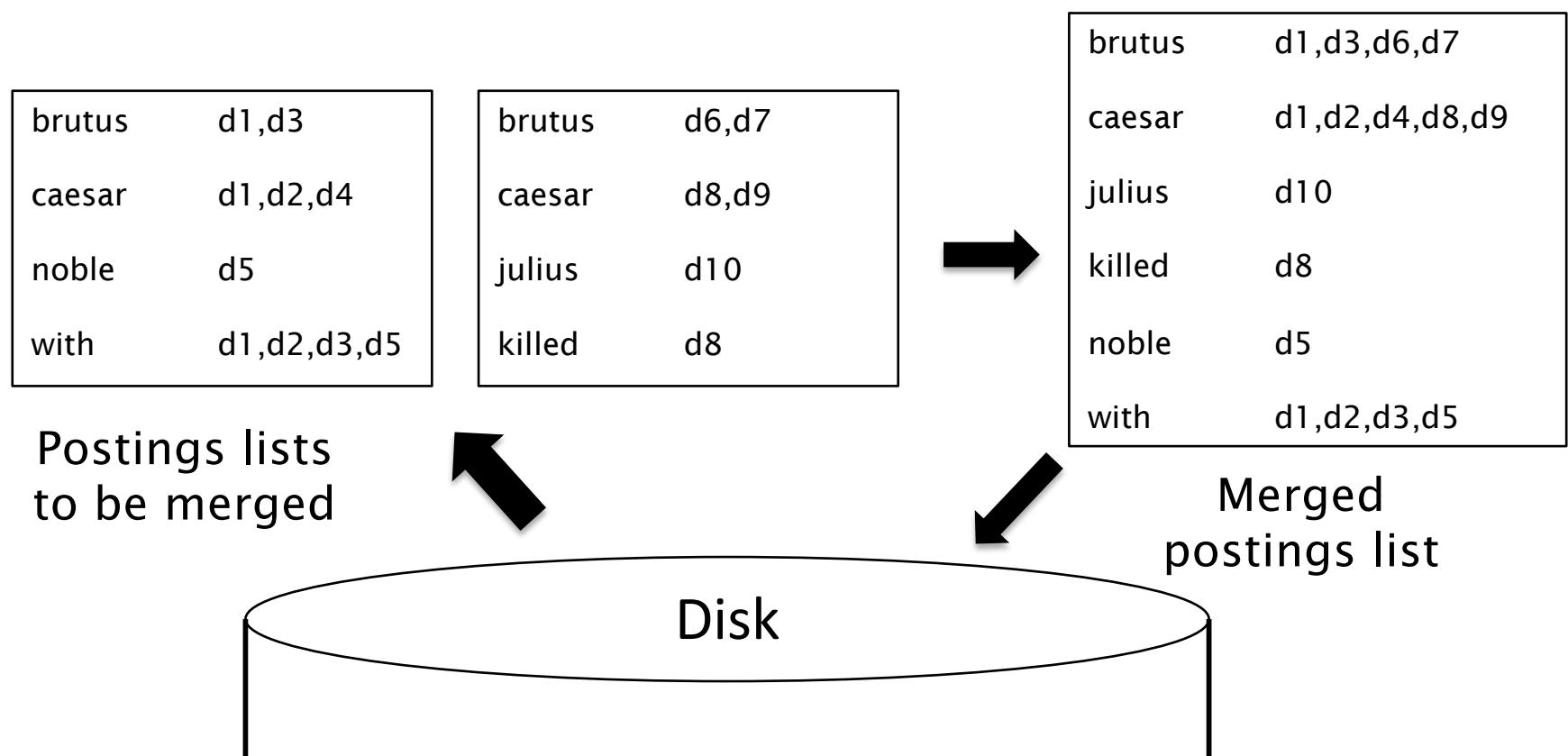
```
1   $n \leftarrow 0$ 
2  while (all documents have not been processed)
3  do  $n \leftarrow n + 1$ 
4       $block \leftarrow \text{PARSENEXTBLOCK}()$ 
5      BSBI-INVERT( $block$ )
6      WRITEBLOCKTODISK( $block, f_n$ )
7      MERGEBLOCKS( $f_1, \dots, f_n; f_{\text{merged}}$ )
```

Sorting 10 blocks of 10M records

- First, read each block and sort within:
 - Quicksort takes $O(N \ln N)$ expected steps
 - In our case $N=10M$
- 10 times this estimate – gives us 10 sorted *runs* of 10M records each.
- Done straightforwardly, need 2 copies of data on disk
 - But can optimize this

How to merge the sorted runs?

- Can do binary merges, with a merge tree of $\log_2 10 = 4$ layers.
- During each layer, read into memory runs in blocks of 10M, merge, write back.



How to merge the sorted runs?

- But it is more efficient to do a multi-way merge, where you are reading from all blocks simultaneously
 - Open all block files simultaneously and maintain a read buffer for each one and a write buffer for the output file
 - In each iteration, pick the lowest termID that hasn't been processed using a priority queue
 - Merge all postings lists for that termID and write it out
- Providing you read decent-sized chunks of each block into memory and then write out a decent-sized output chunk, then you're not killed by disk seeks

Remaining problem with sort-based algorithm

- Our assumption was: we can keep the dictionary in memory.
- We need the dictionary (which grows dynamically) in order to implement a term to termID mapping.

SPIMI:

Single-pass in-memory indexing

- Key idea 1: Generate separate dictionaries for each block – no need to maintain term-termID mapping across blocks.
- Key idea 2: Don't sort. Accumulate postings in postings lists as they occur.
- With these two ideas we can generate a complete inverted index for each block.
- These separate indexes can then be merged into one big index.

SPIMI-Invert

SPIMI-INVERT(*token_stream*)

```
1  output_file = NEWFILE()
2  dictionary = NEWHASH()
3  while (free memory available)
4  do token  $\leftarrow$  next(token_stream)
5    if term(token)  $\notin$  dictionary
6      then postings_list = ADDTODICTIONARY(dictionary, term(token))
7      else postings_list = GETPOSTINGSLIST(dictionary, term(token))
8    if full(postings_list)
9      then postings_list = DOUBLEPOSTINGSLIST(dictionary, term(token))
10     ADDTOPOSTINGSLIST(postings_list, docID(token))
11  sorted_terms  $\leftarrow$  SORTTERMS(dictionary)
12  WRITEBLOCKTODISK(sorted_terms, dictionary, output_file)
13  return output_file
```

- Merging of blocks is analogous to BSBI.

SPIMI in action

Input token

Caesar d1

with d1

Brutus d1

Caesar d2

with d2

Brutus d3

with d3

Caesar d4

noble d5

with d5

Dictionary

brutus d1 d3

with d1 d2 d3 d5

noble d5

caesar d1 d2 d4

Sorted
dictionary

brutus d1 d3

caesar d1 d2 d4

noble d5

with d1 d2 d3 d5

SPIMI: Compression

- Compression makes SPIMI even more efficient.
 - Compression of terms
 - Compression of postings
- More on this later ...

Original publication on SPIMI: Heinz and Zobel (2003)

Introduction to
Information Retrieval

CS276: Information Retrieval and Web Search

Distributed indexing

Distributed indexing

- For web-scale indexing (don't try this at home!):
 - must use a distributed computing cluster
- Individual machines are fault-prone
 - Can unpredictably slow down or fail
- How do we exploit such a pool of machines?

Web search engine data centers

- Web search data centers (Google, Bing, Baidu) mainly contain commodity machines.
- Data centers are distributed around the world.
- Estimate: Google ~1 million servers, 3 million processors/cores (Gartner 2007)

Massive data centers

- If in a non-fault-tolerant system with 1000 nodes, each node has 99.9% uptime, what is the uptime of the entire system?
- Answer: 37% - meaning, 63% of the time one or more servers is down.
- Exercise: Calculate the number of servers failing per minute for an installation of 1 million servers.

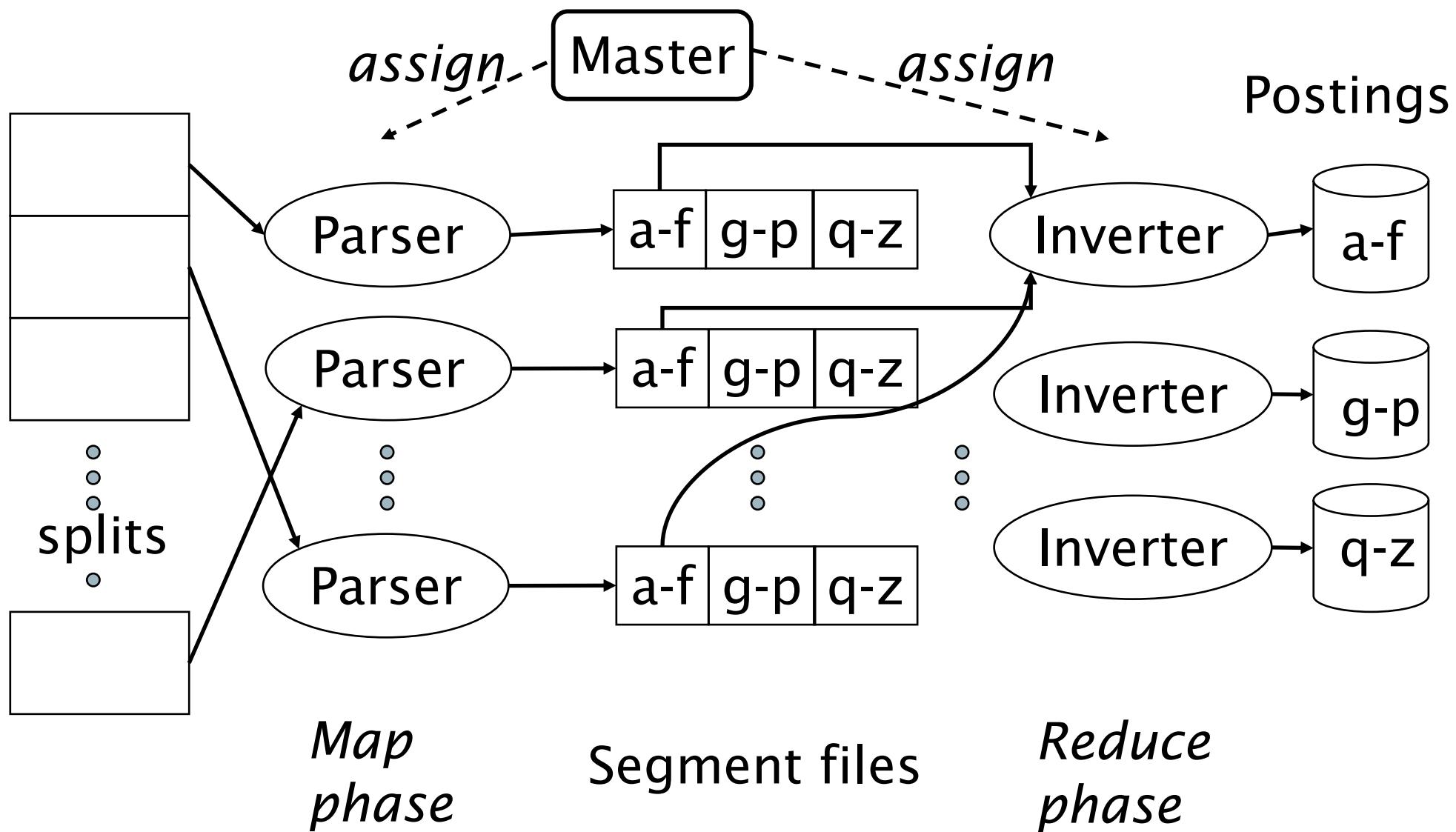
Distributed indexing

- Maintain a *master* machine directing the indexing job
 - considered “safe”.
- Break up indexing into sets of (parallel) tasks.
- Master machine assigns each task to an idle machine from a pool.

Parallel tasks

- We will use two sets of parallel tasks
 - Parsers
 - Inverters
- Break the input document collection into *splits*
- Each split is a subset of documents (corresponding to blocks in BSBI/SPIMI)

Data flow



Parsers

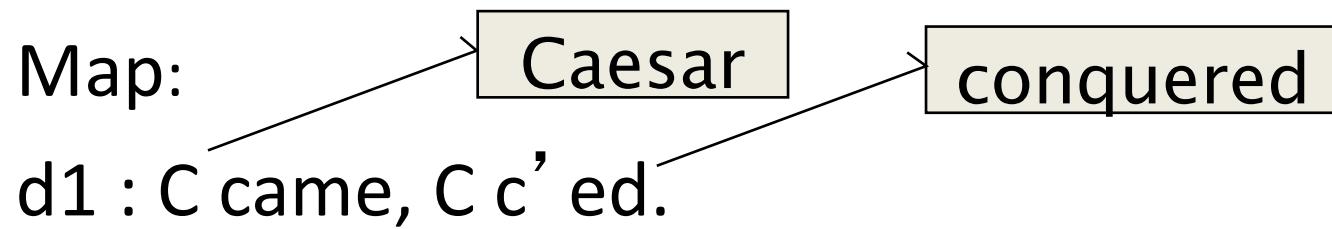
- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into j partitions
- Example: Each partition is for a range of terms' first letters
 - (e.g., **a-f**, **g-p**, **q-z**) – here $j = 3$.
- Now to complete the index inversion

Inverters

- An inverter collects all (term,doc) pairs (= postings) for one term-partition.
- Sorts and writes to postings lists

Example for index construction

Map:



d2 : C died.

→

<C,d1>, <came,d1>, <C,d1>, <c' ed, d1>, <C, d2>, <died,d2>

Reduce:

(<C,(d1,d1,d2)>, <died,(d2)>, <came,(d1)>, <c' ed,(d1)>)

→

(<C,(d1:2,d2:1)><died,(d2:1)>, <came,(d1:1)>, <c' ed,(d1:1)>)

Index construction

- Index construction was just one phase.
- Another phase: transforming a term-partitioned index into a document-partitioned index.
 - *Term-partitioned*: one machine handles a subrange of terms
 - *Document-partitioned*: one machine handles a subrange of documents
- As we'll discuss in the web part of the course, most search engines use a document-partitioned index ... better load balancing, etc.

MapReduce

- The index construction algorithm we just described is an instance of *MapReduce*.
- MapReduce (Dean and Ghemawat 2004) is a robust and conceptually simple framework for distributed computing ...
- ... without having to write code for the distribution part.
- They describe the Google indexing system (ca. 2002) as consisting of a number of phases, each implemented in MapReduce.

Schema for index construction in MapReduce

- **Schema of map and reduce functions**
- map: input \rightarrow list(k, v) reduce: (k, list(v)) \rightarrow output
- **Instantiation of the schema for index construction**
- map: collection \rightarrow list(termID, docID)
- reduce: (<termID1, list(docID)>, <termID2, list(docID)>, ...) \rightarrow (postings list1, postings list2, ...)

Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search
Dynamic indexing

Dynamic indexing

- Up to now, we have assumed that collections are static.
- They rarely are:
 - Documents come in over time and need to be inserted.
 - Documents are deleted and modified.
- This means that the dictionary and postings lists have to be modified:
 - Postings updates for terms already in dictionary
 - New terms added to dictionary

Simplest approach

- Maintain “big” main index
- New docs go into “small” auxiliary index
- Search across both, merge results
- Deletions
 - Invalidation bit-vector for deleted docs
 - Filter docs output on a search result by this invalidation bit-vector
- Periodically, re-index into one main index

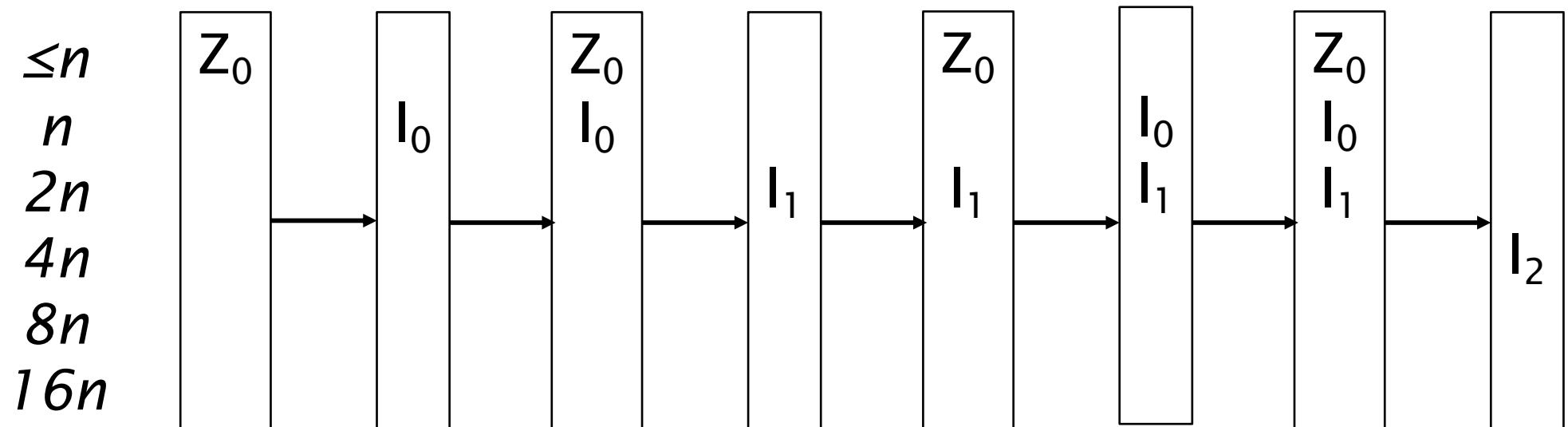
Issues with main and auxiliary indexes

- Problem of frequent merges – you touch stuff a lot
- Poor performance during merge
- Actually:
 - Merging of the auxiliary index into the main index is efficient if we keep a separate file for each postings list.
 - Merge is the same as a simple append.
 - But then we would need a lot of files – inefficient for OS.
- Assumption for the rest of the lecture: The index is one big file.
- In reality: Use a scheme somewhere in between (e.g., split very large postings lists, collect postings lists of length 1 in one file etc.)

Logarithmic merge

- Maintain a series of indexes, each twice as large as the previous one
 - At any time, some of these powers of 2 are instantiated
- Keep smallest (Z_0) in memory
- Larger ones (I_0, I_1, \dots) on disk
- If Z_0 gets too big ($> n$), write to disk as I_0
- or merge with I_0 (if I_0 already exists) as Z_1
- Either write merge Z_1 to disk as I_1 (if no I_1)
- Or merge with I_1 to form Z_2

Logarithmic merge in action



LMERGEADDTOKEN(*indexes*, Z_0 , *token*)

```
1   $Z_0 \leftarrow \text{MERGE}(Z_0, \{\text{token}\})$ 
2  if  $|Z_0| = n$ 
3    then for  $i \leftarrow 0$  to  $\infty$ 
4      do if  $I_i \in \text{indexes}$ 
5        then  $Z_{i+1} \leftarrow \text{MERGE}(I_i, Z_i)$ 
6          ( $Z_{i+1}$  is a temporary index on disk.)
7           $\text{indexes} \leftarrow \text{indexes} - \{I_i\}$ 
8        else  $I_i \leftarrow Z_i$  ( $Z_i$  becomes the permanent index  $I_i$ .)
9           $\text{indexes} \leftarrow \text{indexes} \cup \{I_i\}$ 
10         BREAK
11      $Z_0 \leftarrow \emptyset$ 
```

LOGARITHMICMERGE()

```
1   $Z_0 \leftarrow \emptyset$  ( $Z_0$  is the in-memory index.)
2   $\text{indexes} \leftarrow \emptyset$ 
3  while true
4  do LMERGEADDTOKEN(indexes,  $Z_0$ , GETNEXTTOKEN())
```

Logarithmic merge

- Auxiliary and main index:
 - T/n merges where T is # of postings and n is size of auxiliary
 - Index construction time is $O(T^2/n)$ as in the worst case a posting is touched T/n times
- Logarithmic merge: Each posting is merged at most $O(\log(T/n))$ times, so complexity is $O(T \log(T/n))$
- So logarithmic merge is much more efficient for index construction
- But query processing now requires the merging of $O(\log(T/n))$ indexes
 - Whereas it is $O(1)$ if you just have a main and auxiliary index

Further issues with multiple indexes

- Collection-wide statistics are hard to maintain
- E.g., when we speak of spell-correction: which of several corrected alternatives do we present to the user?
 - We may want to pick the one with the most hits
 - How do we maintain the top ones with multiple indexes and invalidation bit vectors?
 - One possibility: ignore everything but the main index for such ordering
- Will see more such statistics used in results ranking

Dynamic indexing at search engines

- All the large search engines now do dynamic indexing
- Their indices have frequent incremental changes
 - News items, blogs, new topical web pages
- But (sometimes/typically) they also periodically reconstruct the index from scratch
 - Query processing is then switched to the new index, and the old index is deleted

Earlybird: Real-time search at Twitter

- Requirements for real-time search
 - Low latency, high throughput query evaluation
 - High ingestion rate and immediate data availability
 - Concurrent reads and writes of the index
 - Dominance of temporal signal

Earlybird: Index organization

- Earlybird consists of multiple index segments
 - Each segment is relatively small, holding up to 2^{23} tweets
 - Each posting in a segment is a 32 bit word: 24 bits for the tweet id and 8 bits for the position in the tweet
- Only one segment can be written to at any given time
 - Small enough to be in memory
 - New postings are simply appended to the postings list
 - But the postings list is traversed backwards to prioritize newer tweets
- The remaining segments are optimized for read-only
 - Postings sorted in reverse chronological order (newest first)

Other sorts of indexes

- Positional indexes
 - Same sort of sorting problem ... just larger 
- Building character n-gram indexes:
 - As text is parsed, enumerate n -grams.
 - For each n -gram, need pointers to all dictionary terms containing it – the “postings”

Resources for today's lecture

- Chapter 4 of IIR
- MG Chapter 5
- Original publication on MapReduce: Dean and Ghemawat (2004)
- Original publication on SPIMI: Heinz and Zobel (2003)
- Earlybird: Busch et al, ICDE 2012

Introduction to **Information Retrieval**

CS276: Information Retrieval and Web Search

Pandu Nayak and Prabhakar Raghavan

Lecture 6: Scoring, Term Weighting and the
Vector Space Model

This lecture; IIR Sections 6.2-6.4.3

- Ranked retrieval
- Scoring documents
- Term frequency
- Collection statistics
- Weighting schemes
- Vector space scoring

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*”: 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**, 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 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
- $\text{jaccard}(A,B) = |A \cap B| / |A \cup B|$
- $\text{jaccard}(A,A) = 1$
- $\text{jaccard}(A,B) = 0$ 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

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.

Recall (Lecture 2): 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

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.

Term frequency tf

- The term frequency $\text{tf}_{t,d}$ 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

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

- $0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms t in both q and d :
- 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.

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*.

Collection vs. Document frequency

- Collection frequency of t is the number of occurrences of t in the collection
- Document frequency of t is the number of documents in which t occurs
- Example:

Word	Collection frequency	Document frequency
<i>insurance</i>	10440	3997
<i>try</i>	10422	8760

- Which word is for better search (gets higher weight)

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
 - $\text{df}_t \leq N$
- We define the idf (inverse document frequency) of t by

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

- We use $\log (N/\text{df}_t)$ 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

$$\text{idf}_t = \log_{10} (N/\text{df}_t)$$

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

Effect of idf on ranking

- 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.

tf-idf weighting

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

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

- 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

Score for a document given a query

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

- 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

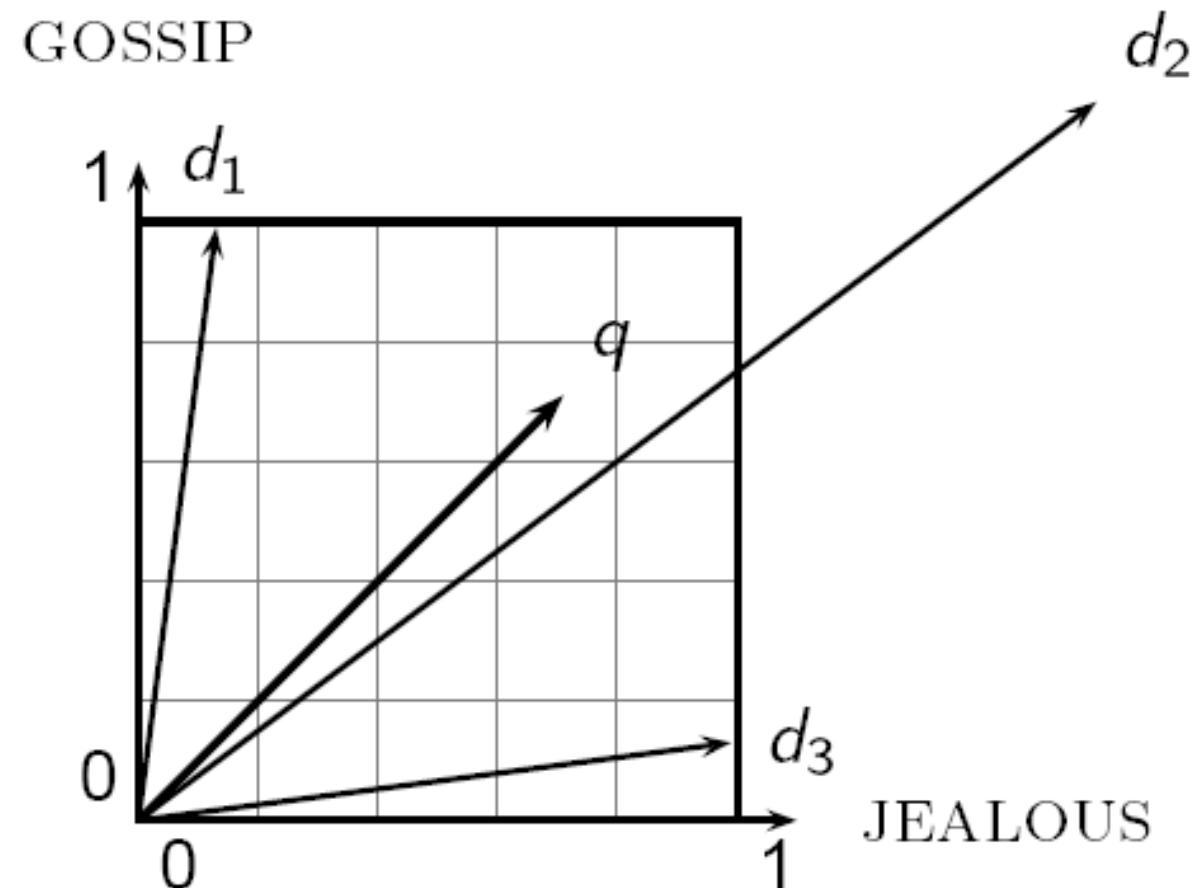
- Key idea 1: Do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity \approx inverse of distance

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 \vec{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

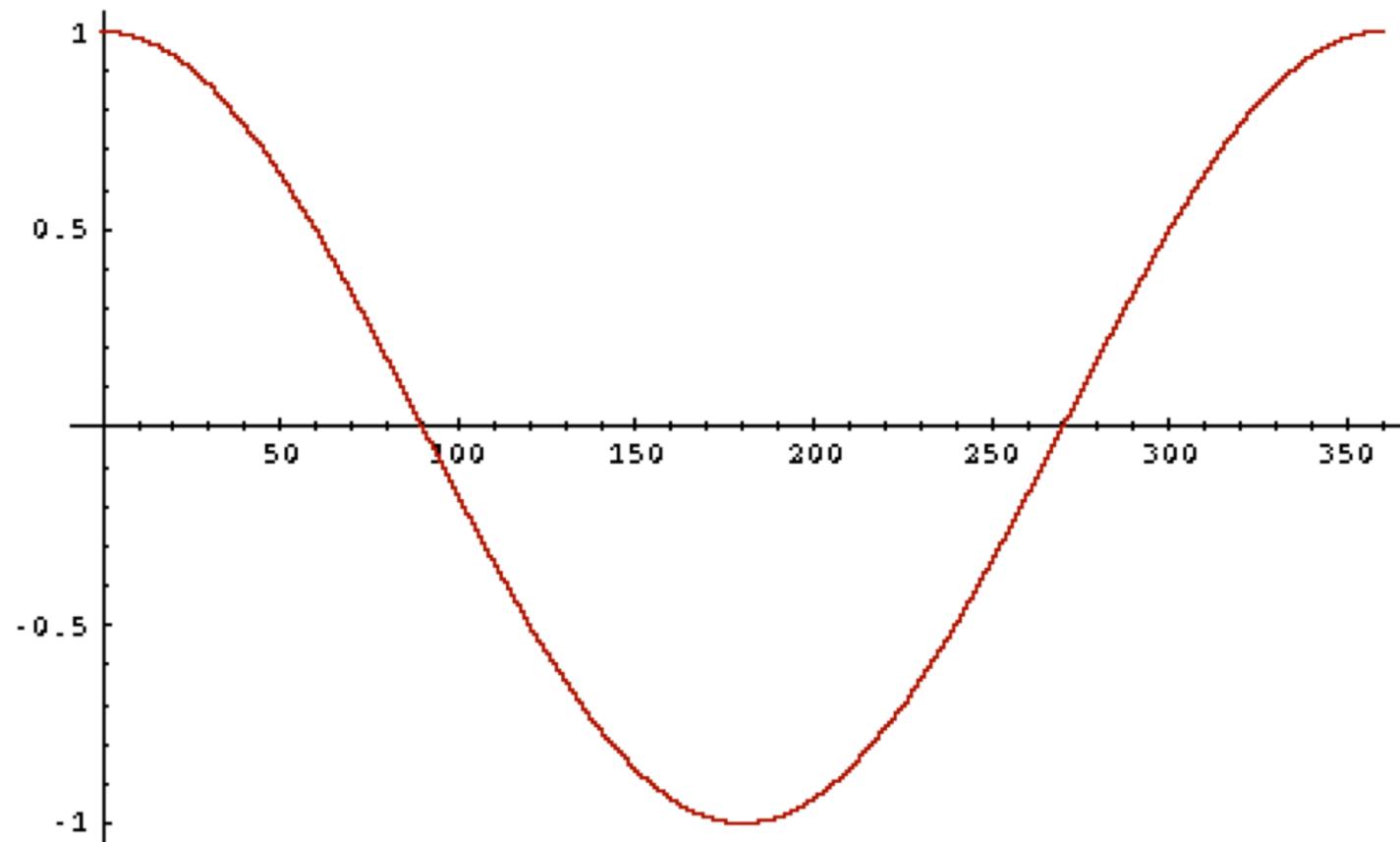
- Thought experiment: take a document d and append it to itself. Call this document d' .
- “Semantically” d and d' have the same content
- The Euclidean distance between the two documents can be quite large
- The angle between the two documents is 0, corresponding to maximal similarity.

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

From angles to cosines



- But how 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)

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

Dot product Unit vectors

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} .

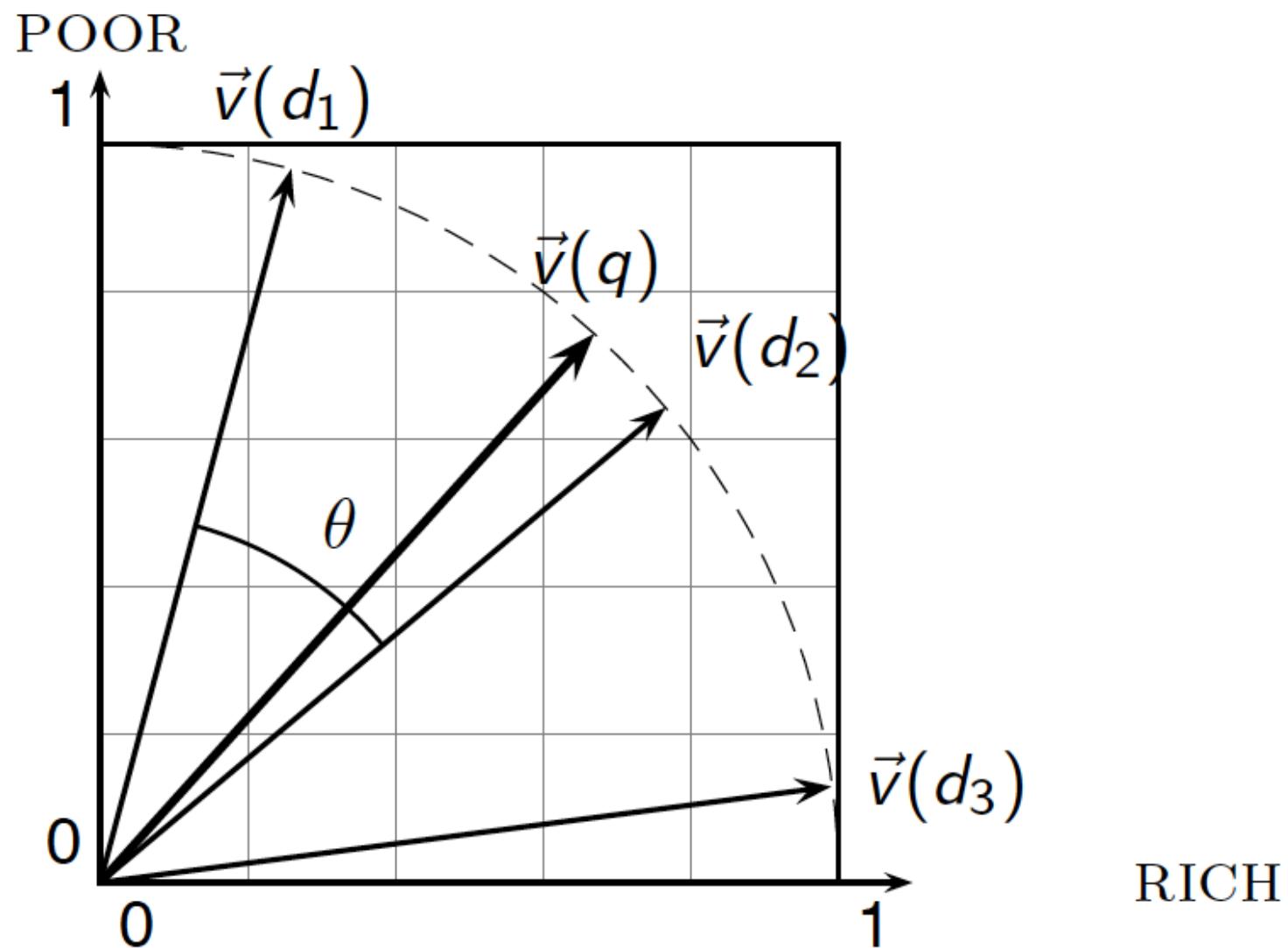
Cosine for length-normalized vectors

- 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



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

$$\begin{aligned} \text{dot(SaS,PaP)} &\approx 12.1 \\ \text{dot(SaS,WH)} &\approx 13.4 \\ \text{dot(PaP,WH)} &\approx 10.1 \end{aligned}$$

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

$$\begin{aligned} \cos(\text{SaS},\text{PaP}) &\approx 0.94 \\ \cos(\text{SaS},\text{WH}) &\approx 0.79 \\ \cos(\text{PaP},\text{WH}) &\approx 0.69 \end{aligned}$$

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 $Scores[]$

Computing cosine scores

- Previous algorithm scores term-at-a-time (TAAT)
- Algorithm can be adapted to scoring document-at-a-time (DAAT)
- Storing $w_{t,d}$ in each posting could be expensive
 - ...because we'd have to store a floating point number
 - For tf-idf scoring, it suffices to store $tf_{t,d}$ in the posting and idf_t in the head of the postings list
- Extracting the top K items can be done with a priority queue (e.g., a heap)

tf-idf weighting has many variants

Term frequency	Document frequency	Normalization
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
I (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.ooo*, using the acronyms from the previous table
- A very standard weighting scheme is: Inc.ltc
- Document: logarithmic tf (**I as first character**), no idf and cosine normalization
- Query: logarithmic tf (**I in leftmost column**), idf (**t in second column**), cosine normalization ...

tf-idf example: Inc.ltc

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

Term	Query						Document				Pro d
	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

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

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

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

Resources for today's lecture

- IIR 6.2 – 6.4.3
- <http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html>
 - Term weighting and cosine similarity tutorial for SEO folk!

Computing scores in a complete search system

Chapter 7 - IIR



SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Quick Recap



SAPIENZA
UNIVERSITÀ DI ROMA

Term Frequency Weight

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

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



Inverse Document Frequency - IDF

- The document frequency df_t is defined as the number of documents that t occurs in.
- We define the idf weight of term t as follows:

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

- idf is a measure of the informativeness of the term.



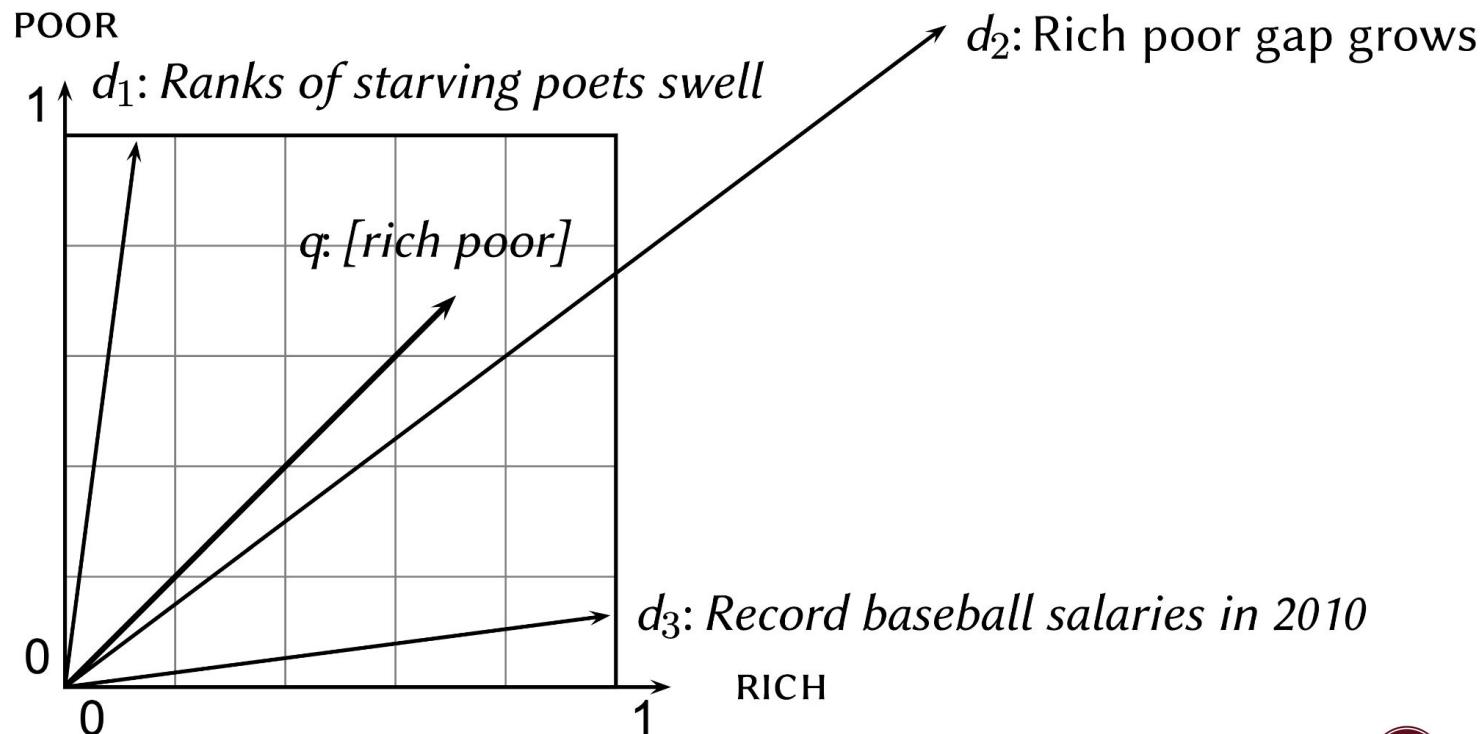
TF-IDF Weight

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

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



Issue with Euclidean Distance



Cosine Similarity between Query and Document

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

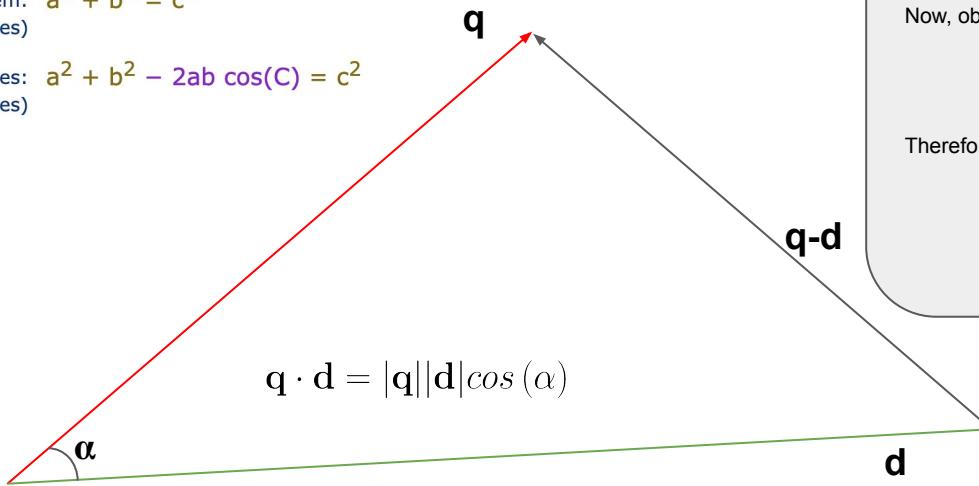
- q_i is the tf-idf weight of term i in the query.
- d_i is the tf-idf weight of term i in the document.
- $|\vec{q}|$ and $|\vec{d}|$ are the lengths of vectors \vec{q} and \vec{d} , respectively.
- $\vec{q}/|\vec{q}|$ and $\vec{d}/|\vec{d}|$ are length-1 vectors (= normalized)



Cosine Similarity between Query and Document

Pythagoras Theorem: $a^2 + b^2 = c^2$
(only for Right-Angled Triangles)

Law of Cosines: $a^2 + b^2 - 2ab \cos(C) = c^2$
(for all triangles)



By the Law of Cosines:
 $|q - d|^2 = |d|^2 + |q|^2 - 2|d||q|\cos(\alpha)$

Now, observe that:

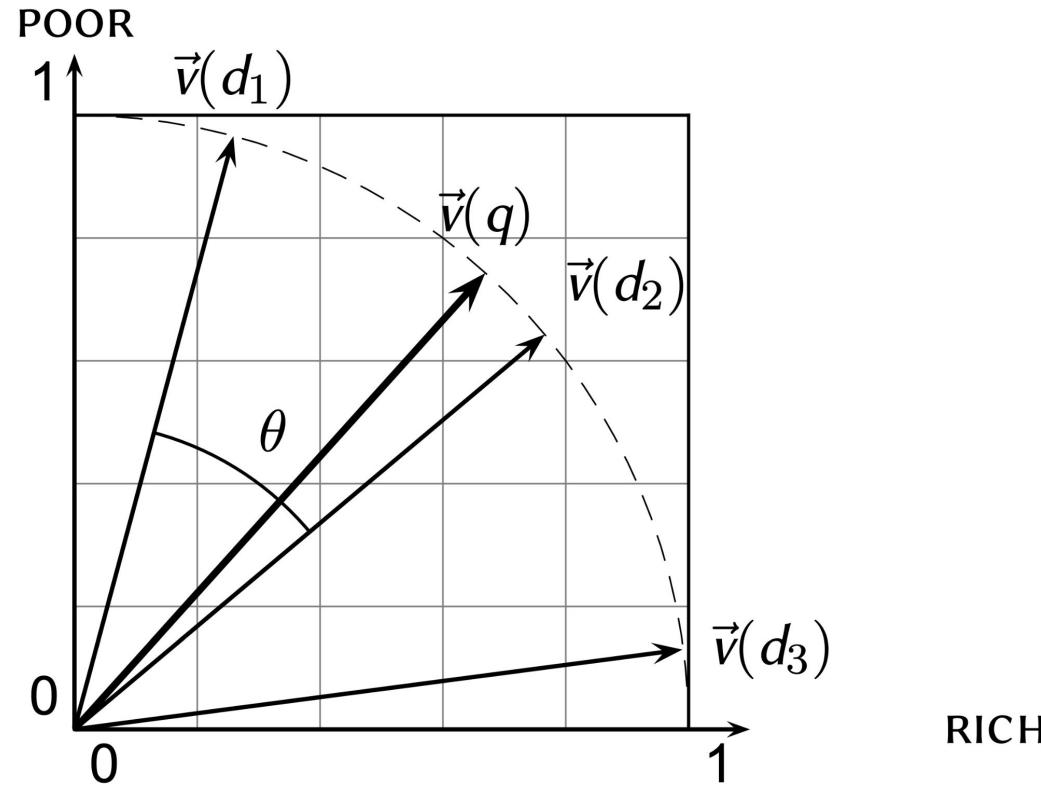
$$\begin{aligned} |q - d|^2 &= (q - d) \cdot (q - d) \\ &= q \cdot q - 2(q \cdot d) + d \cdot d \\ &= |q|^2 + |d|^2 - 2(q \cdot d) \end{aligned} = = =$$

Therefore:

$$\begin{aligned} |d|^2 + |q|^2 - 2|d||q|\cos(\alpha) &= |q|^2 + |d|^2 - 2(q \cdot d) \\ \Rightarrow 2|d||q|\cos(\alpha) &= 2(q \cdot d) \\ \Rightarrow q \cdot d &= |d||q|\cos(\alpha) \end{aligned}$$



Length Normalization



Cosine Similarity Illustrated

- Query: “best car insurance”. Document: “car insurance auto insurance”

word	query					document			product
	tf-raw	tf-wght	df	idf	tf-idf weight	tf-raw	tf-wght	n'lized	
auto	0	0	5000	2.3	0	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	0.68	2.04

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

$$1/1.92 \approx 0.52$$

$$1.3/1.92 \approx 0.68$$

- Final similarity score between query and document:

$$\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08$$



Is Cosine Enough?

- Query q: “anti-doping rules Beijing 2008 olympics”
- Compare three documents
 - d1 → a short document on anti-doping rules at 2008 Olympics
 - d2 → a long document that consists of a copy of d1 and 5 other news stories, all on topics different from Olympics/anti-doping
 - d3 → a short document on anti-doping rules at the 2004 Athens Olympics
- What ranking do we expect in the vector space model?
 - d2 is likely to be ranked below d3 ...
 - ...but d2 is more relevant than d3.

Can we do something about this?



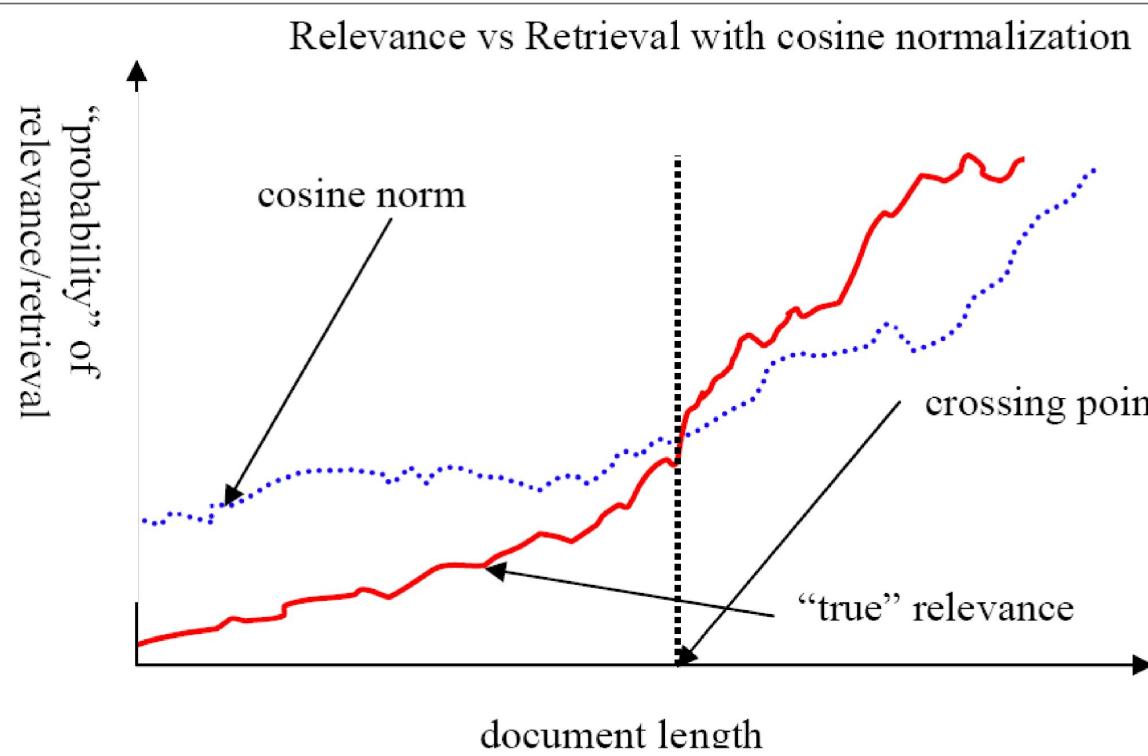
SAPIENZA
UNIVERSITÀ DI ROMA

Pivot Normalization

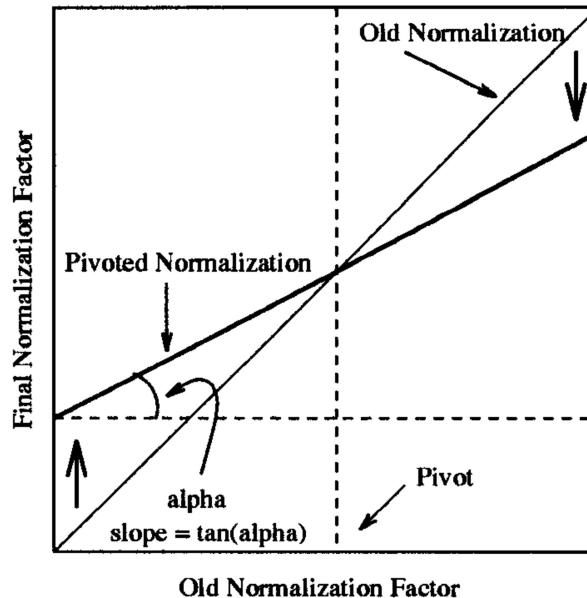
- Cosine normalization produces weights that are too large for short documents and too small for long documents (on average).
- Adjust cosine normalization by linear adjustment: “turning” the average normalization on the pivot
- Effect: Similarities of short documents with query decrease; similarities of long documents with query increase.
- This removes the unfair advantage that short documents have.
- Note that “pivoted” scores are no longer bounded by 1.



Predicted and true probability of relevance



Pivot normalization



- pivoted normalization = $(1.0 - \text{slope}) \times \text{pivot} + \text{slope} \times \text{old normalization}$



Effect on Effectiveness: Amit Singhal's experiments

Cosine	Pivoted Cosine Normalization				
	Slope				
	0.60	0.65	0.70	0.75	0.80
6,526	6,342	6,458	6,574	6,629	6,671
0.2840	0.3024	0.3097	0.3144	0.3171	0.3162
Improvement	+ 6.5%	+ 9.0%	+10.7%	+11.7%	+11.3%

- (relevant documents retrieved and (change in) average precision)



Ranking



SAPIENZA
UNIVERSITÀ DI ROMA

The Importance of Ranking

- Users want to look at a few results – not thousands.
- It's very hard to write queries that produce a few results.
- Even for expert searchers
- → Ranking is important because it effectively reduces a large set of results to a very small one.



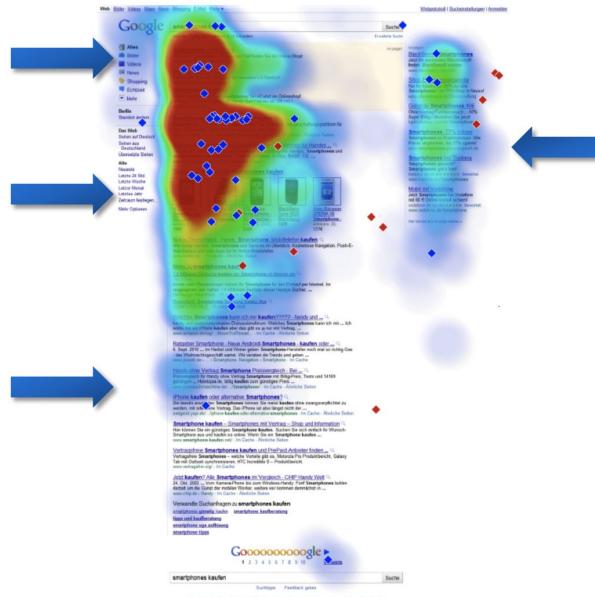
Attention of Users

Desktop search engine result page

Strong focus on the AdWords results

Rich Media elements move the attention/fixation points down the site

Nearly no focus on the organic results below the fold



People look more intensely on the right hand side than on the organic results below the fold

thinkinsights
with Google

Source: Eye Square Eye Tracking Study, 2011
Base: Respondents with contact to the stationary advertising on Google (n=38 Stationary)
Info: Aggregation over three brands

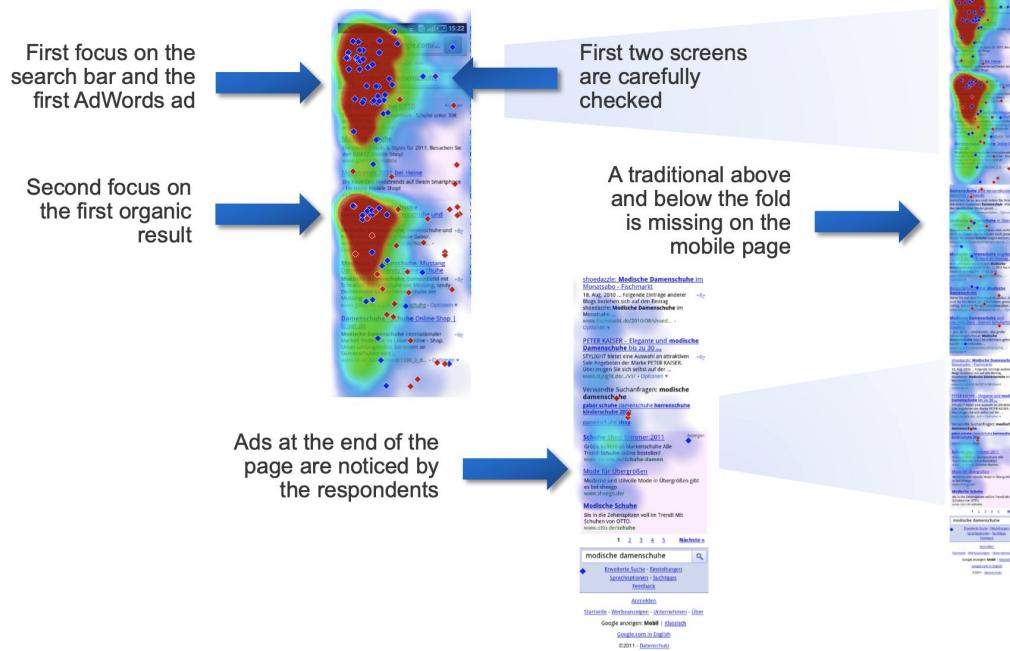
Google Think Insights – Eye Tracking Study, July 2011 7



SAPIENZA
UNIVERSITÀ DI ROMA

Attention of Users

Mobile search engine results page



Source: Eye Square Eye Tracking Study, 2011
Base: Respondents with contact to the mobile advertising on Google (n=50 mobile)
Info: Aggregation over three brands

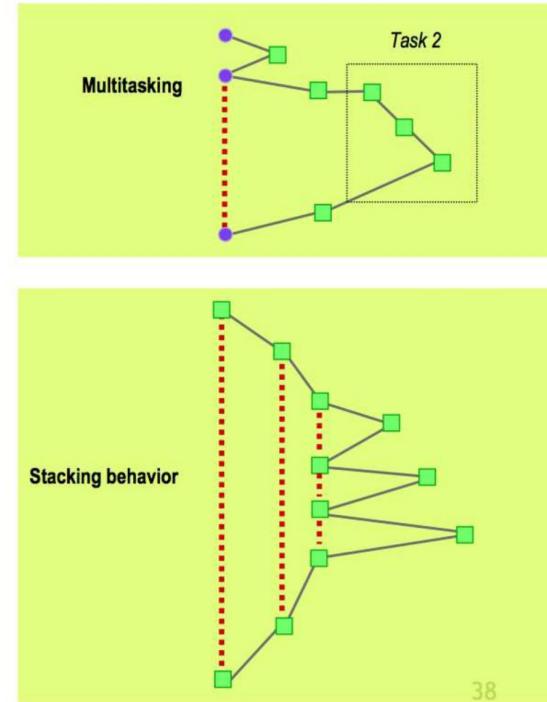
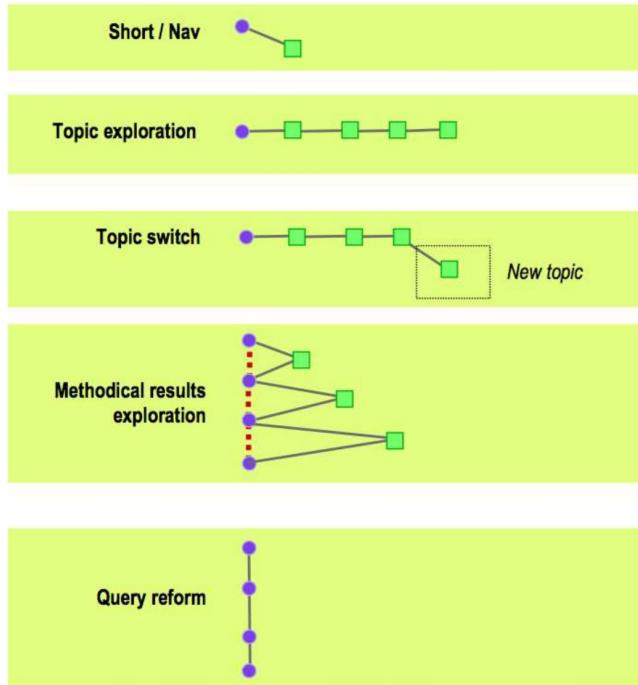
Google Think Insights – Eye Tracking Study, July 2011 8

thinkinsights
with Google



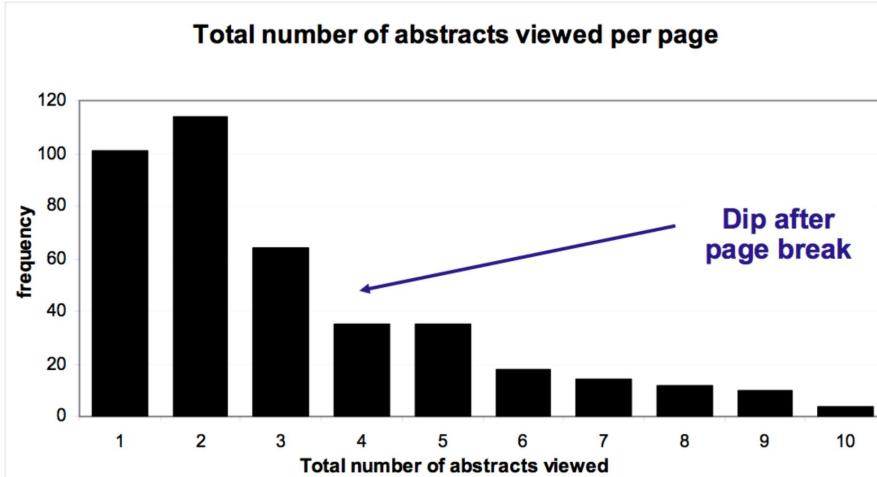
SAPIENZA
UNIVERSITÀ DI ROMA

Users' Behavior



Users' Behavior

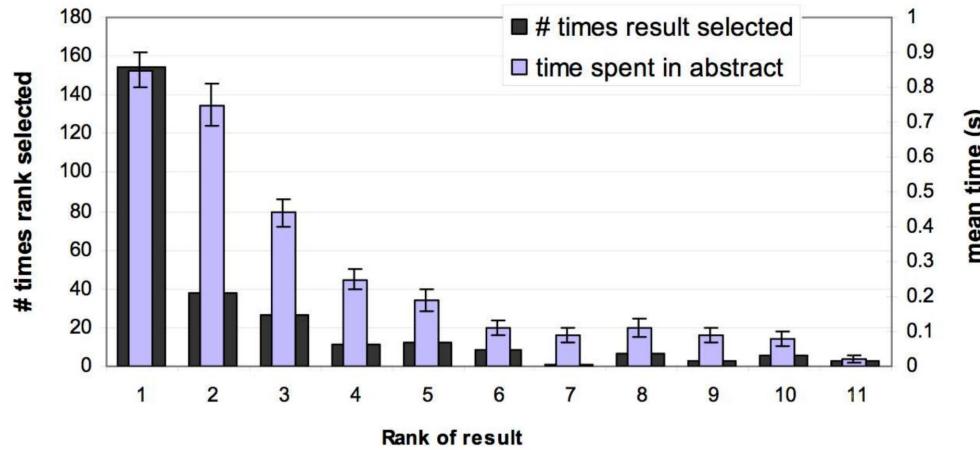
How many links do users view?



Mean: 3.07 Median/Mode: 2.00

Users' Behavior

Looking vs. Clicking



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

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.**

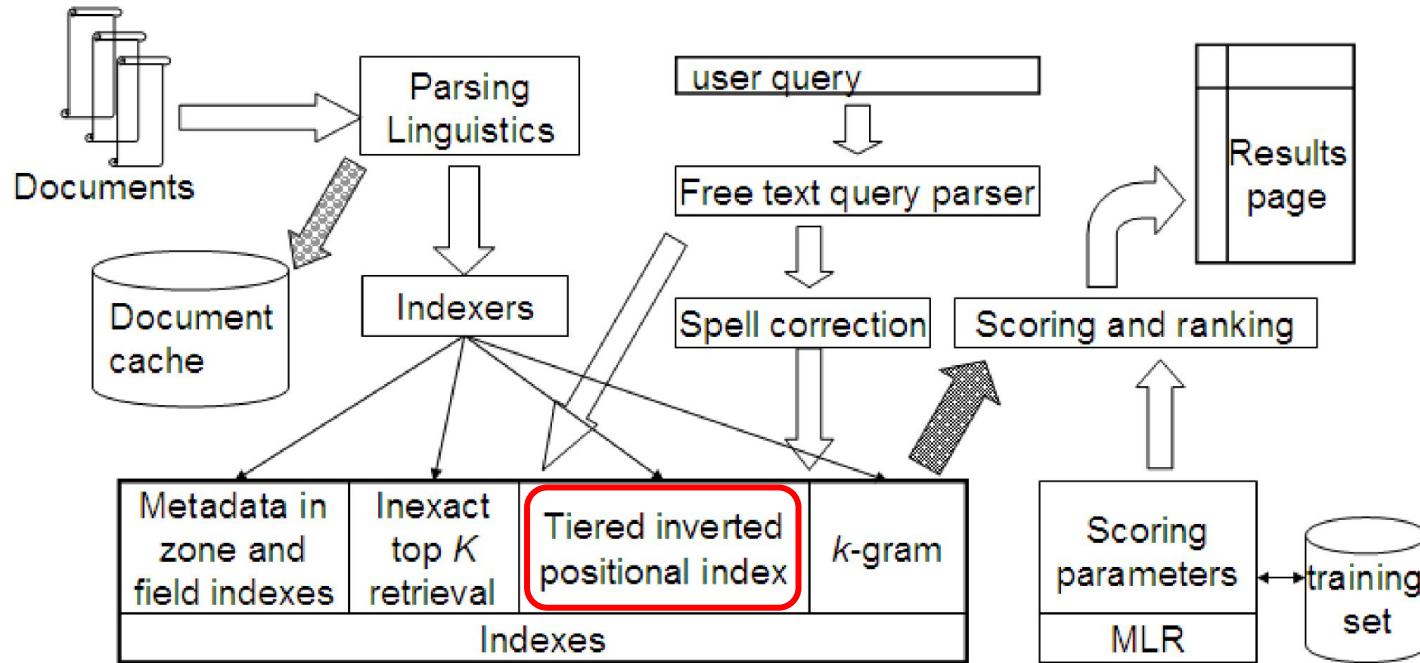


Real(istic) Search System



SAPIENZA
UNIVERSITÀ DI ROMA

The Architecture

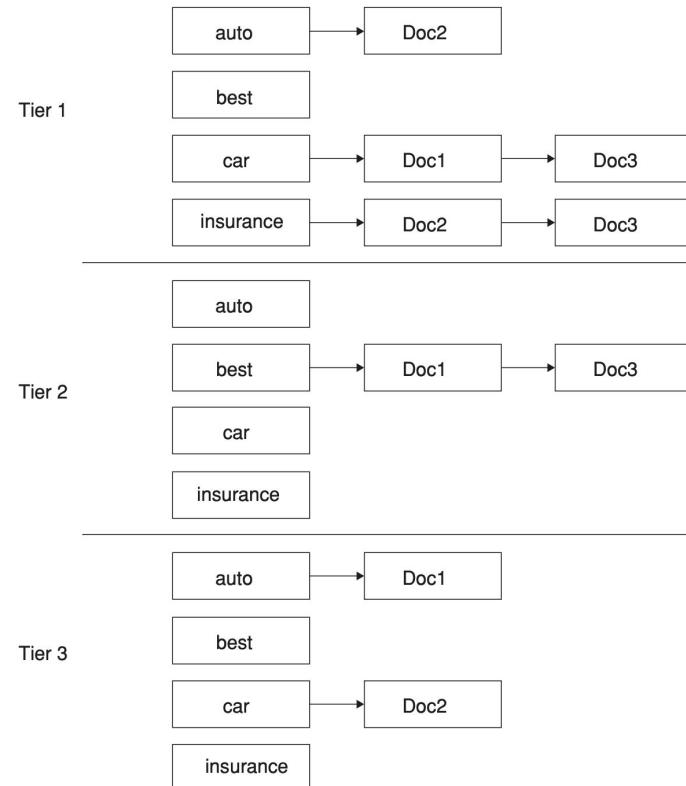


Tiered Index

- Basic idea:
 - Create several tiers of indexes, corresponding to importance of indexing terms
 - During query processing, start with highest-tier index
 - If highest-tier index returns at least k (e.g., $k = 100$) results: stop and return results to user
 - If we've only found $< k$ hits: repeat for next index in tier cascade
- Example: two-tier system
 - Tier 1: Index of all titles
 - Tier 2: Index of the rest of documents
 - Pages containing the search words in the title are better hits than pages containing the search words in the body of the text.



Simple Example



The Importance of Tiered Indexes

- The use of tiered indexes is believed to be one of the reasons that Google search quality was significantly higher initially (2000/01) than that of competitors.
 - (along with PageRank, use of anchor text and proximity constraints)



Efficient Scoring



SAPIENZA
UNIVERSITÀ DI ROMA

Now we also need term frequencies in the index

BRUTUS	→	1,2	7,3	83,1	87,2	...
--------	---	-----	-----	------	------	-----

CAESAR	→	1,1	5,1	13,1	17,1	...
--------	---	-----	-----	------	------	-----

CALPURNIA	→	7,1	8,2	40,1	97,3	
-----------	---	-----	-----	------	------	--

term frequencies



SAPIENZA
UNIVERSITÀ DI ROMA

Term Frequencies in the Inverted Index

- Thus: In each posting, store $tf_{t,d}$ in addition to docID d .
- As an integer frequency, not as a (log-)weighted real number. . .
 - . . . because real numbers are (more) difficult to compress.
- Overall, additional space requirements are small: a byte per posting or less



How do we compute Top-K results?

- We usually don't need a complete ranking.
- We just need the top k for a small k (e.g., $k = 100$).
- If we don't need a complete ranking, is there an efficient way of computing just the top k ?
- Naive:
 - Compute scores for all N documents
 - Sort
 - Return the top k
- Not very efficient



How do we compute Top-K results?

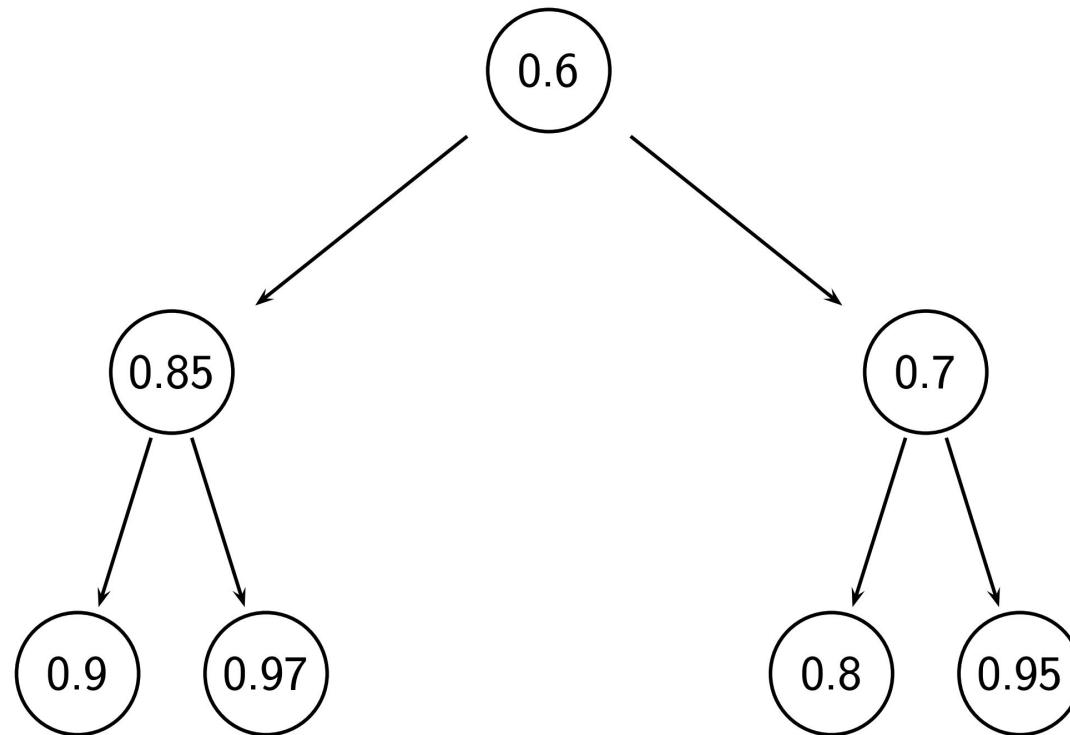
- We usually don't need a complete ranking.
- We just need the top k for a small k (e.g., k = 100).
- If we don't need a complete ranking, is there an efficient way of computing just the top k?
- Naive:
 - Compute scores for all N documents
 - Sort
 - Return the top k
- Not very efficient

Alternatively, use a Min-Heap



SAPIENZA
UNIVERSITÀ DI ROMA

What's a Min-Heap



How do we compute Top-K results?

- Goal: Keep the top k documents seen so far
- Use a binary min heap
- To process a new document d' with score s' :
 - Get current minimum h_m of heap ($O(1)$)
 - If $s' \leq h_m$ skip to next document
 - If $s' > h_m$ heap-delete-root ($O(\log k)$)
 - Heap-add d'/s' ($O(\log k)$)



Even more efficient solutions?

- Ranking has time complexity $O(N)$ where N is the number of documents.
- Optimizations reduce the constant factor, but they are still $O(N)$, $N > 10^{10}$
- Are there sublinear algorithms?
- What we're doing in effect: solving the k-nearest neighbor (kNN) problem for the query vector (= query point).
- There are no general solutions to this problem that are sublinear.



Even more efficient solutions: Heuristics

- Idea 1: Reorder postings lists
 - Instead of ordering according to docID . . .
 - . . . order according to some measure of “expected relevance”.
- Idea 2: Heuristics to prune the search space
 - Not guaranteed to be correct . . .
 - . . . but fails rarely.
 - In practice, close to constant time.
 - For this, we'll need the concepts of document-at-a-time processing and term-at-a-time processing.



Orderings different than DocID

- So far: postings lists have been ordered according to docID.
- Alternative: a query-independent measure of “goodness” of a page
- Example: **PageRank $g(d)$ of page d , a measure of how many “good” pages hyperlink to d**
- Order documents in postings lists according to PageRank:
 $g(d_1) > g(d_2) > g(d_3) > \dots$
- Define composite score of a document:
$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$
- This scheme supports early termination:
 - We do not have to process postings lists in their entirety to find top k .



Orderings different than DocID

- Order documents in postings lists according to PageRank:
$$g(d_1) > g(d_2) > g(d_3) > \dots$$
- Define composite score of a document:
$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$
- Suppose: (i) $g \rightarrow [0, 1]$; (ii) $g(d) < 0.1$ for the document d we're currently processing; (iii) smallest top k score we've found so far is 1.2
 - Then all subsequent scores will be < 1.1 .
- So we've already found the top k and can stop processing the remainder of postings lists.



Document-at-a-Time

- Both docID-ordering and PageRank-ordering impose a consistent ordering on documents in postings lists.
- Computing cosines in this scheme is document-at-a-time.
- We complete computation of the query-document similarity score of document d_i before starting to compute the query-document similarity score of d_{i+1} .
 - Alternative: term-at-a-time processing



Weight sorted posting lists

- Idea: *don't process postings that contribute little to final score*
- Order documents in postings list according to weight
- Simplest case: normalized tf-idf weight (rarely done: hard to compress)
- Documents in the top k are likely to occur early in these ordered lists.
- → Early termination while processing postings lists is unlikely to change the top k.
- But:
 - We no longer have a consistent ordering of documents in postings lists.
 - We no longer can employ document-at-a-time processing.



Term-at-a-Time Processing

- Simplest case: completely process the postings list of the first query term
- Create an accumulator for each docID you encounter
- Then completely process the postings list of the second query
- Term
 - . . . and so forth



Term-at-a-Time Processing

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 $Scores[]$

The elements of the array “Scores” are called **accumulators**.



SAPIENZA
UNIVERSITÀ DI ROMA

Accumulators

- For the web (20 billion documents), an array of accumulators A in memory is infeasible.
 - Thus: Only create accumulators for docs occurring in postings lists
- This is equivalent to: Do not create accumulators for docs with zero scores (i.e., docs that do not contain any of the query terms)



Accumulators

BRUTUS	→	1,2	7,3	83,1	87,2	...
CAESAR	→	1,1	5,1	13,1	17,1	...
CALPURNIA	→	7,1	8,2	40,1	97,3	

- For query: [Brutus Caesar]:
- Only need accumulators for 1, 5, 7, 13, 17, 83, 87
- Don't need accumulators for 3, 8 etc.



Enforcing Conjunctive Search

- We can enforce conjunctive search (a la Google): only consider documents (and create accumulators) if all terms occur.
- Example: just one accumulator for [Brutus Caesar] in the
- example above . . .
 - . . . because only d1 contains both words.



Ranking: Summary

- Ranking is **very expensive** in applications where we have to compute similarity scores for all documents in the collection.
- In most applications, the vast majority of documents have **similarity score 0** for a given query → lots of potential for speeding things up.
- However, there is **no fast nearest neighbor algorithm** that is guaranteed to be correct even in this scenario.
- In practice: **use heuristics** to prune search space – usually works very well.



Introduction to
Information Retrieval

CS276
Information Retrieval and Web Search

Chris Manning and Pandu Nayak

Systems issues

Background

- Score computation is a large (10s of %) fraction of the CPU work on a query
 - Generally, we have a tight budget on latency (say, 250ms)
 - CPU provisioning doesn't permit exhaustively scoring every document on every query
- Today we'll look at ways of cutting CPU usage for scoring, without compromising the quality of results (much)
- Basic idea: avoid scoring docs that won't make it into the top K

Safe vs non-safe ranking

- The terminology “safe ranking” is used for methods that guarantee that the K docs returned are the K absolute highest scoring documents
- Is it ok to be non-safe?

Ranking function is only a proxy

- User has a task and a query formulation
- Ranking function matches docs to query
- Thus the ranking function is anyway a proxy for user happiness
- If we get a list of K docs “close” to the top K by the ranking function measure, should be ok

Recap: 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, measured by cosine similarity

Efficient cosine ranking

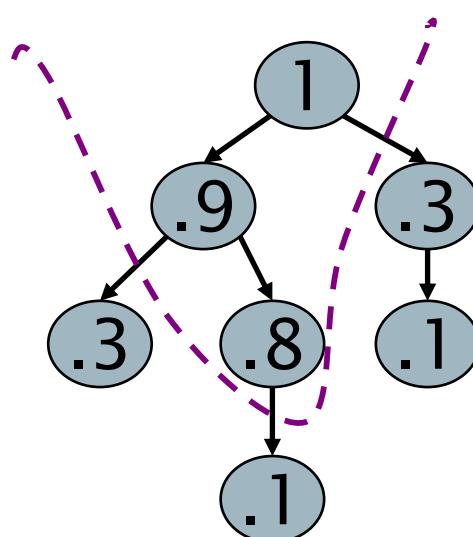
- Find the K docs in the collection “nearest” to the query $\Rightarrow K$ largest query-doc cosines.
- Efficient ranking:
 - Computing a single cosine efficiently.
 - Choosing the K largest cosine values efficiently.
 - Can we do this without computing all N cosines?

Computing the K largest cosines: selection vs. sorting

- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
 - not to totally order all docs in the collection
- Can we pick off docs with K highest cosines?
- Let J = number of docs with nonzero cosines
 - We seek the K best of these J

Use heap for selecting top K

- Binary tree in which each node's value > the values of children
- Takes $2J$ operations to construct, then each of K “winners” read off in $2\log J$ steps.
- For $J=1M$, $K=100$, this is about 10% of the cost of sorting.



Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- **Can we avoid all this computation?**
- Yes, but may sometimes get it wrong
 - a doc *not* in the top K may creep into the list of K output docs
 - As noted earlier, this may not be a bad thing

SPEEDING COSINE COMPUTATION BY PRUNING

Generic approach

- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K , but has many docs from among the top K
 - Return the top K docs in A
- Think of A as pruning non-contenders
- The same approach is also used for other (non-cosine) scoring functions
- Will look at several schemes following this approach

Index elimination

- Basic cosine computation algorithm only considers docs containing at least one query term
- Take this further:
 - Only consider high-idf query terms
 - Only consider docs containing many query terms

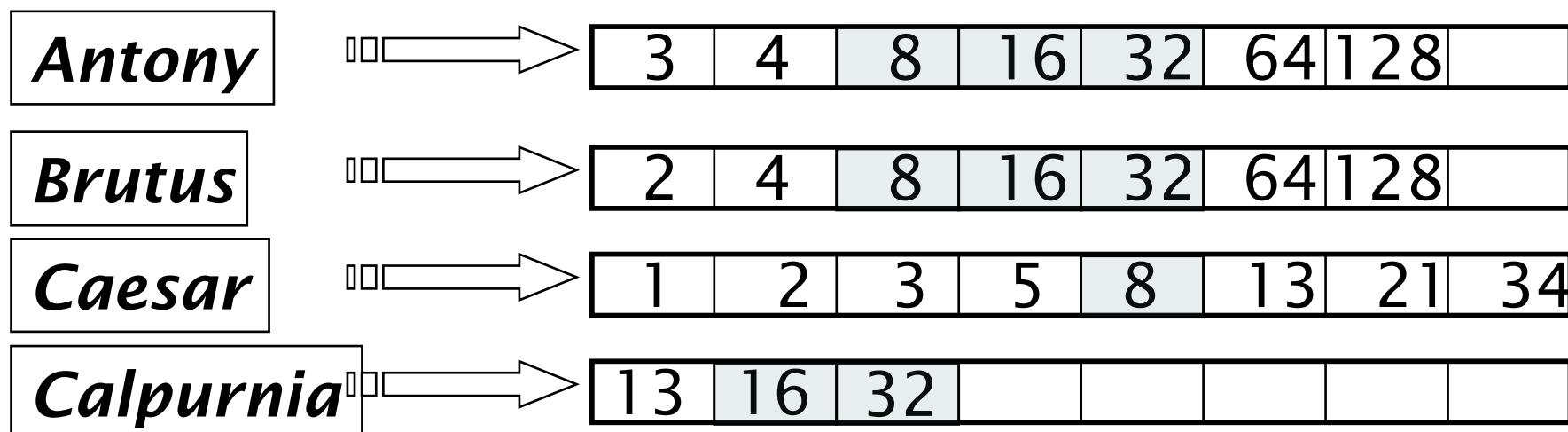
High-idf query terms only

- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: ***in*** and ***the*** contribute little to the scores and so don't alter rank-ordering much
- Benefit:
 - Postings of low-idf terms have many docs → these (many) docs get eliminated from set A of contenders

Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms
 - Say, at least 3 out of 4
 - Imposes a “soft conjunction” on queries seen on web search engines (early Google)
- Easy to implement in postings traversal

3 of 4 query terms



Scores only computed for docs 8, 16 and 32.

Champion lists

- Precompute for each dictionary term t , the r docs of highest weight in t 's postings
 - Call this the champion list for t
 - (aka fancy list or top docs for t)
- Note that r has to be chosen at index build time
 - Thus, it's possible that $r < K$
- At query time, only compute scores for docs in the champion list of some query term
 - Pick the K top-scoring docs from amongst these

Exercises

- How can Champion Lists be implemented in an inverted index?

QUERY-INDEPENDENT DOCUMENT SCORES

Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
 - Wikipedia among websites
 - Articles in certain newspapers
 - A paper with many citations
 - Many bitlys, likes, or bookmarks
 - Pagerank

Quantitative

Modeling authority

- Assign to each document a *query-independent quality score* in $[0,1]$ to each document d
 - Denote this by $g(d)$
- Thus, a quantity like the number of citations is scaled into $[0,1]$
 - Exercise: suggest a formula for this.

Net score

- Consider a simple total score combining cosine relevance and authority
- $\text{net-score}(q, d) = g(d) + \cosine(q, d)$
 - Can use some other linear combination
 - Indeed, any function of the two “signals” of user happiness
- Now we seek the top K docs by net score

Top K by net score – fast methods

- First idea: Order all postings by $g(d)$
- Key: this is a common ordering for all postings
- Thus, can concurrently traverse query terms' postings for
 - Postings intersection
 - Cosine score computation
- Exercise: write pseudocode for cosine score computation if postings are ordered by $g(d)$

Why order postings by $g(d)$?

- Under $g(d)$ -ordering, top-scoring docs likely to appear early in postings traversal
- In time-bound applications (say, we have to return whatever search results we can in 50 ms), this allows us to stop postings traversal early
 - Short of computing scores for all docs in postings

Champion lists in $g(d)$ -ordering

- Can combine champion lists with $g(d)$ -ordering
- Maintain for each term a champion list of the r docs with highest $g(d) + \text{tf-idf}_{td}$
- Seek top- K results from only the docs in these champion lists

CLUSTER PRUNING

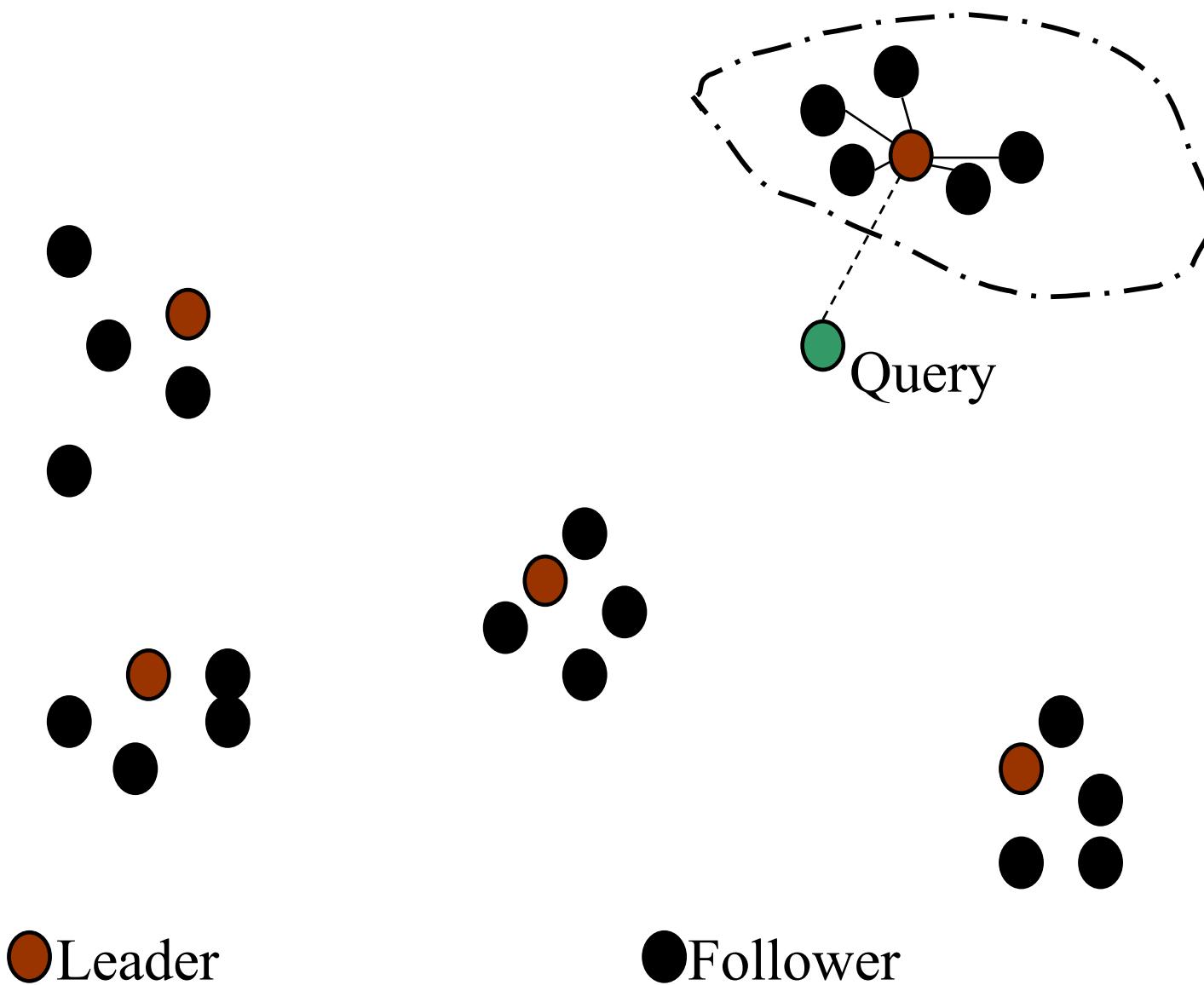
Cluster pruning: preprocessing

- Pick \sqrt{N} docs at random: call these *leaders*
- For every other doc, pre-compute nearest leader
 - Docs attached to a leader: its *followers*;
 - Likely: each leader has $\sim \sqrt{N}$ followers.

Cluster pruning: query processing

- Process a query as follows:
 - Given query Q , find its nearest *leader* L .
 - Seek K nearest docs from among L 's followers.

Visualization



Why use random sampling

- Fast
- Leaders reflect data distribution

General variants

- Have each follower attached to $b1=3$ (say) nearest leaders.
- From query, find $b2=4$ (say) nearest leaders and their followers.
- Can recurse on leader/follower construction.

TIERED INDEXES

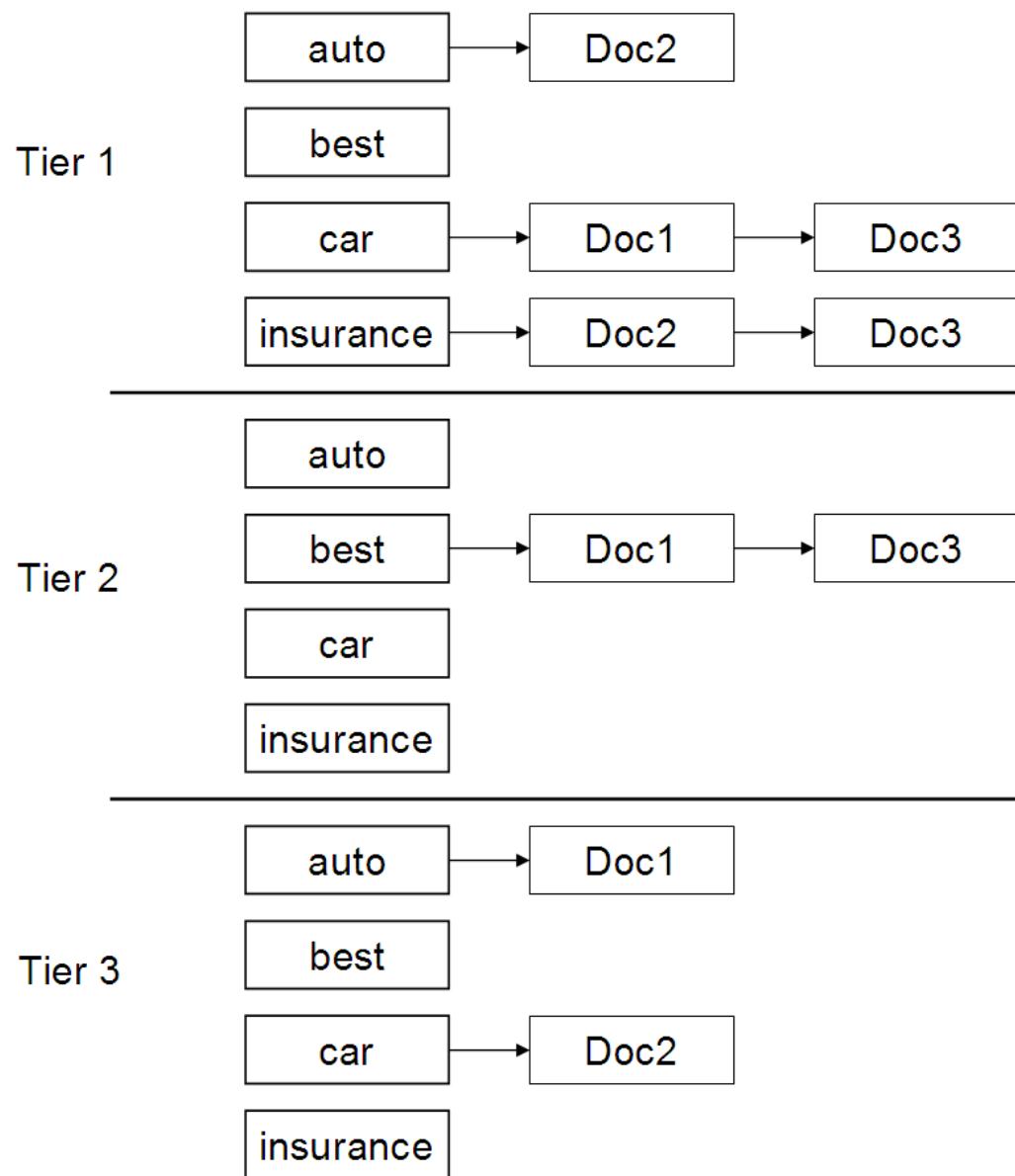
High and low lists

- For each term, we maintain two postings lists called *high* and *low*
 - Think of *high* as the champion list
- When traversing postings on a query, only traverse *high* lists first
 - If we get more than K docs, select the top K and stop
 - Else proceed to get docs from the *low* lists
- Can be used even for simple cosine scores, without global quality $g(d)$
- A means for segmenting index into two tiers

Tiered indexes

- Break postings up into a hierarchy of lists
 - Most important
 - ...
 - Least important
- Can be done by $g(d)$ or another measure
- Inverted index thus broken up into tiers of decreasing importance
- At query time use top tier unless it fails to yield K docs
 - If so drop to lower tiers

Example tiered index



Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!
- How do we compute scores in order to pick off top K ?
 - Two ideas follow

1. Early termination

- When traversing t 's postings, stop early after either
 - a fixed number of r docs
 - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
 - One from the postings of each query term
- Compute only the scores for docs in this union

2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
 - High idf terms likely to contribute most to score
- As we update score contribution from each query term
 - Stop if doc scores relatively unchanged
- Can apply to cosine or some other net scores

SAFE RANKING

Safe vs non-safe ranking

- The terminology “safe ranking” is used for methods that guarantee that the K docs returned are the K absolute highest scoring documents
 - (Not necessarily just under cosine similarity)

Safe ranking

- When we output the top K docs, we have a proof that these are indeed the top K
- Does this imply we always have to compute all N cosines?
 - We'll look at pruning methods
 - So we only fully score some J documents

WAND scoring

- An instance of DAAT scoring
- Basic idea reminiscent of branch and bound
 - We maintain a running *threshold* score – e.g., the K^{th} highest score computed so far
 - We prune away all docs whose cosine scores are guaranteed to be below the threshold
 - We compute exact cosine scores for only the un-pruned docs

Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process.

Index structure for WAND

- Postings ordered by docID
- Assume a special iterator on the postings of the form “go to the first docID greater than or equal to X ”
- Typical state: we have a “finger” at some docID in the postings of each query term
 - Each finger moves only to the right, to larger docIDs
- Invariant – all docIDs lower than any finger have already been *processed*, meaning
 - These docIDs are either pruned away or
 - Their cosine scores have been computed

Upper bounds

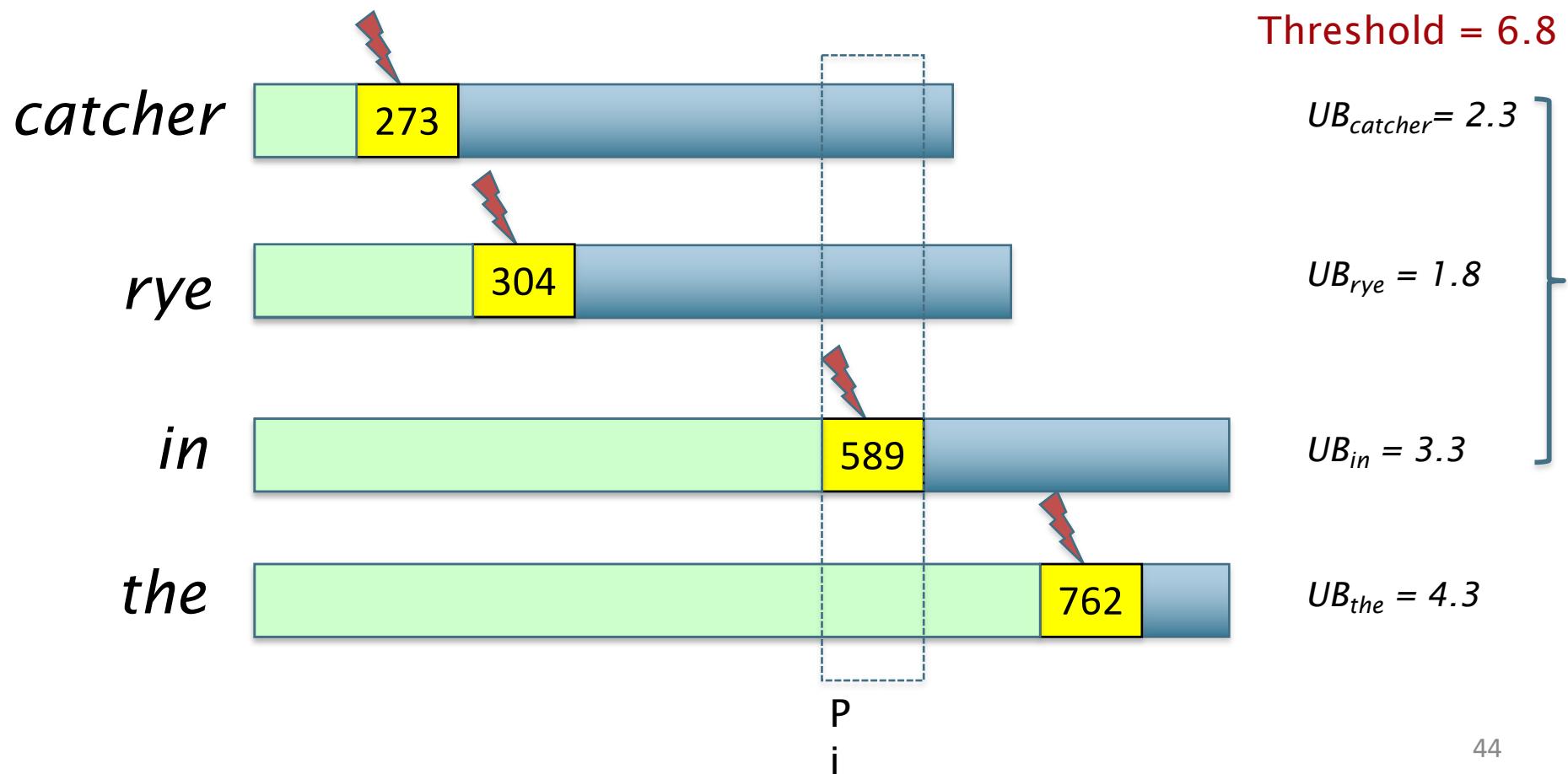
- At all times for each query term t , we maintain an *upper bound* UB_t on the score contribution of any doc to the right of the finger
 - Max (over docs remaining in t 's postings) of $w_t(\text{doc})$



As finger moves right, UB drops

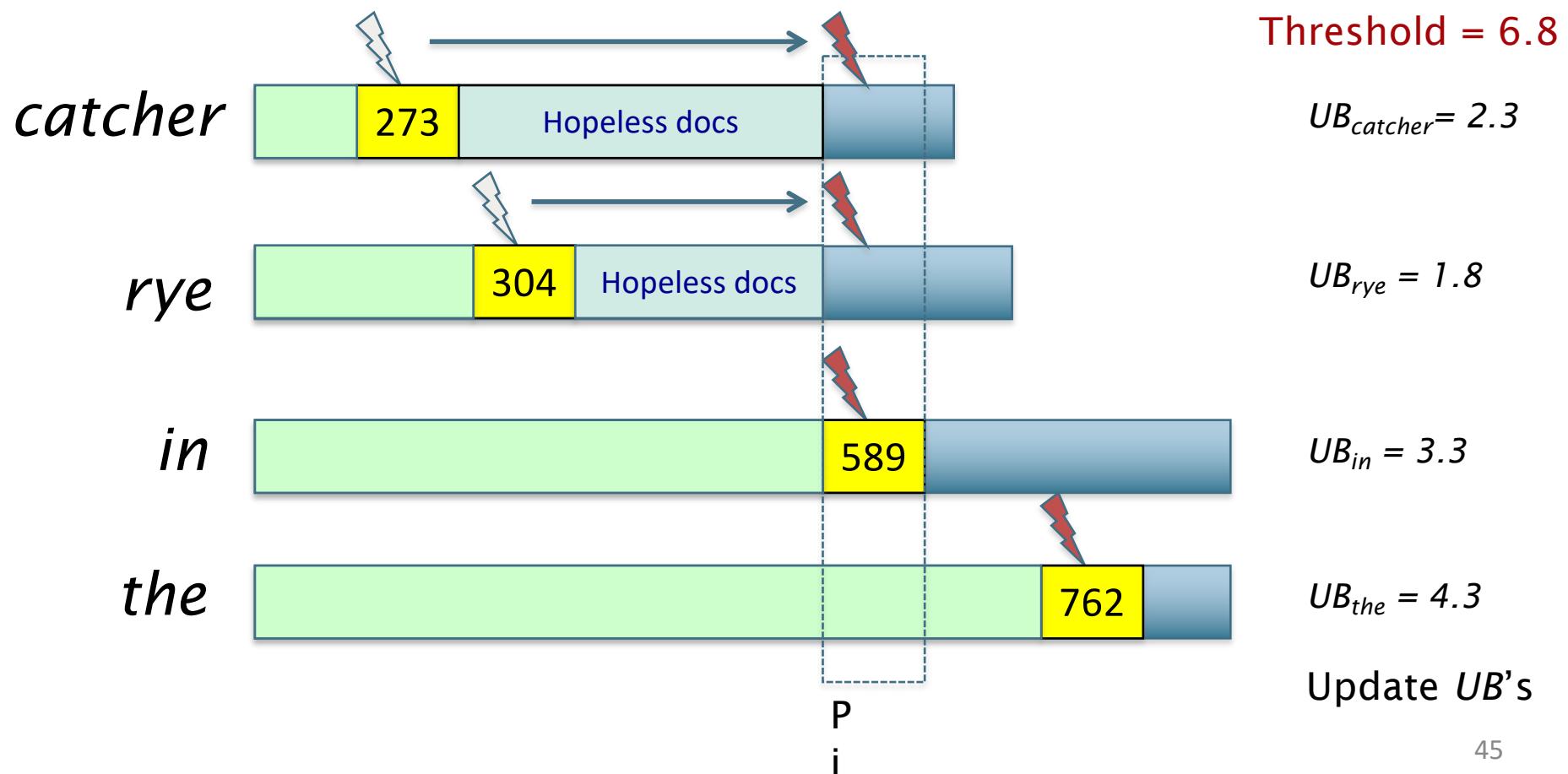
Pivoting

- Query: *catcher in the rye*
- Let's say the current finger positions are as below



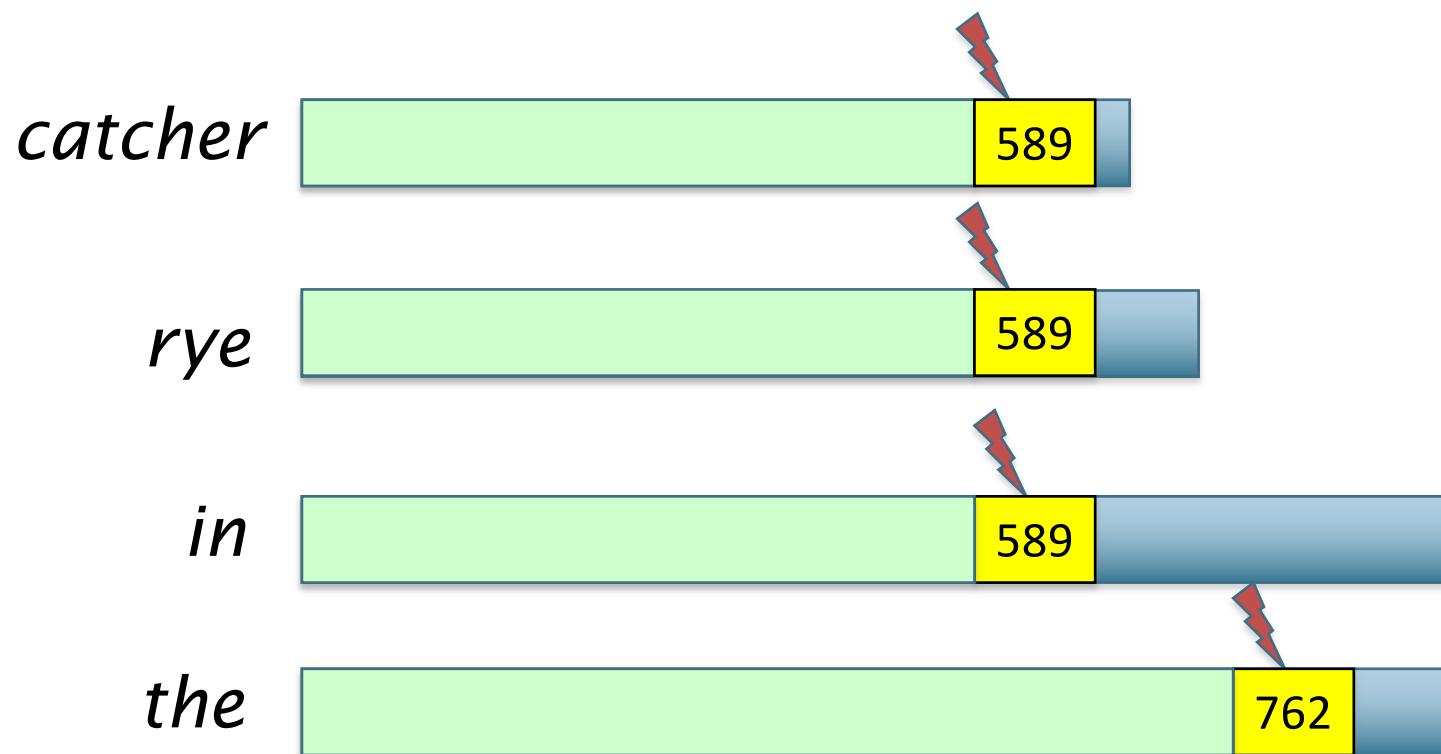
Prune docs that have no hope

- Terms sorted in order of finger positions
- Move fingers to 589 or right



Compute 589's score if need be

- If 589 is present in enough postings, compute its full cosine score – else some fingers to right of 589
- Pivot again ...



WAND summary

- In tests, WAND leads to a 90+% reduction in score computation
 - Better gains on longer queries
- Nothing we did was specific to cosine ranking
 - We need scoring to be *additive* by term
- WAND and variants give us safe ranking
 - Possible to devise “careless” variants that are a bit faster but not safe (see summary in Ding+Suel 2011)
 - Ideas combine some of the non-safe scoring we considered

FINISHING TOUCHES FOR A COMPLETE SCORING SYSTEM

Query term proximity

- Free text queries: just a set of terms typed into the query box – common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- Let w be the smallest window in a doc containing all query terms, e.g.,
- For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)
- Would like scoring function to take this into account – how?

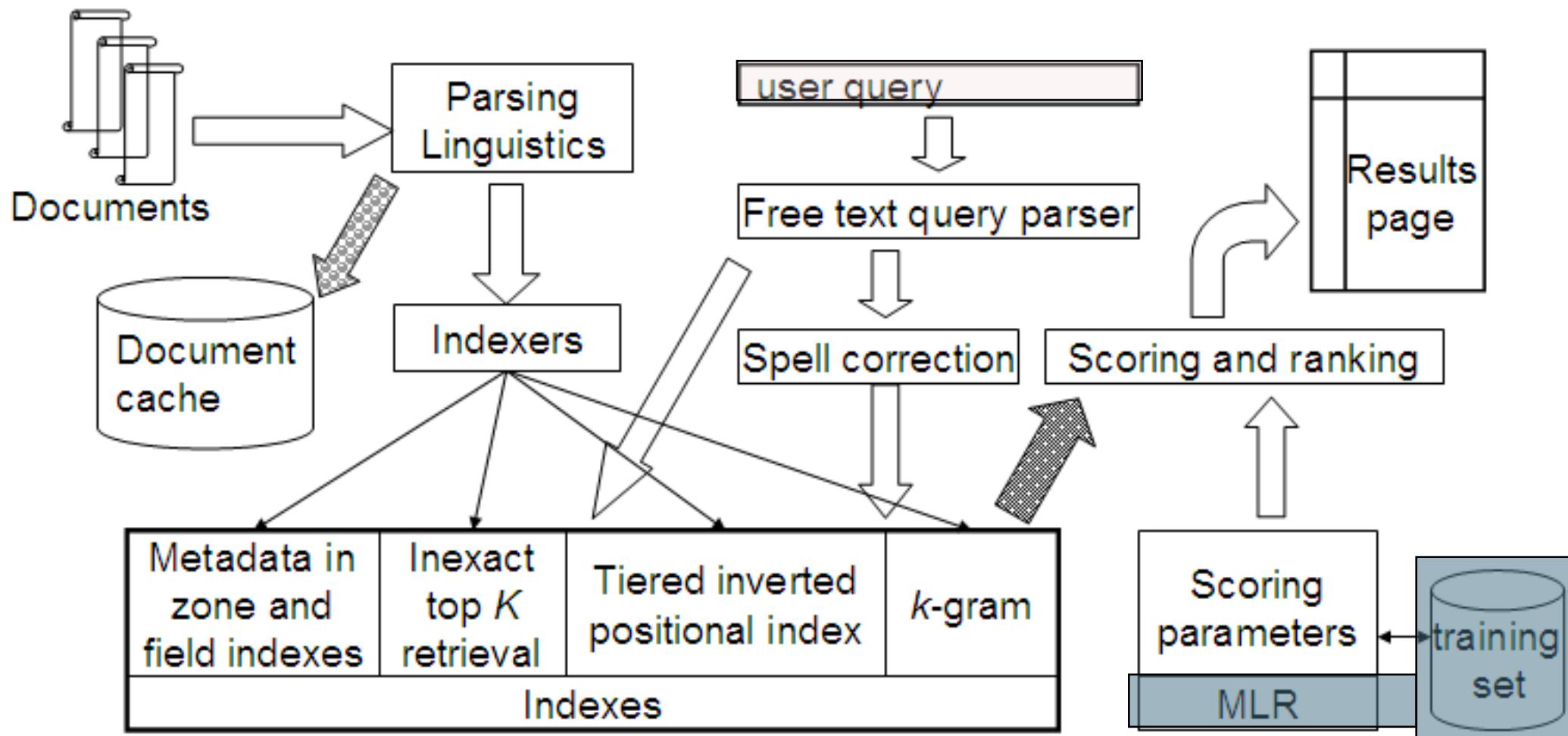
Query parsers

- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*
 - Run the query as a phrase query
 - If $<K$ docs contain the phrase *rising interest rates*, run the two phrase queries *rising interest* and *interest rates*
 - If we still have $<K$ docs, run the vector space query *rising interest rates*
 - Rank matching docs by vector space scoring
- This sequence is issued by a query parser

Aggregate scores

- We've seen that score functions can combine cosine, static quality, proximity, etc.
- **How do we know the best combination?**
- Some applications – expert-tuned
- Increasingly common: machine-learned

Putting it all together



Introduction to Information Retrieval

<http://informationretrieval.org>

IIR 12: Language Models for IR

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2013-05-21

Overview

- 1 Recap
- 2 Feature selection
- 3 Language models
- 4 Language Models for IR
- 5 Discussion

Outline

- 1 Recap
- 2 Feature selection
- 3 Language models
- 4 Language Models for IR
- 5 Discussion

Naive Bayes classification rule

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k | c)]$$

- Each conditional parameter $\log \hat{P}(t_k | c)$ is a weight that indicates how good an indicator t_k is for c .
- The prior $\log \hat{P}(c)$ is a weight that indicates the relative frequency of c .
- The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
- We select the class with the most evidence.

Parameter estimation

- Prior:

$$\hat{P}(c) = \frac{N_c}{N}$$

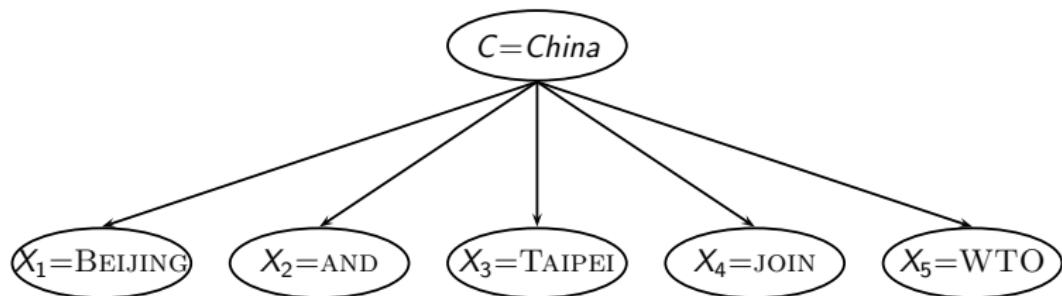
where N_c is the number of docs in class c and N the total number of docs

- Conditional probabilities:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)}$$

where T_{ct} is the number of tokens of t in training documents from class c (includes multiple occurrences)

Add-one smoothing to avoid zeros

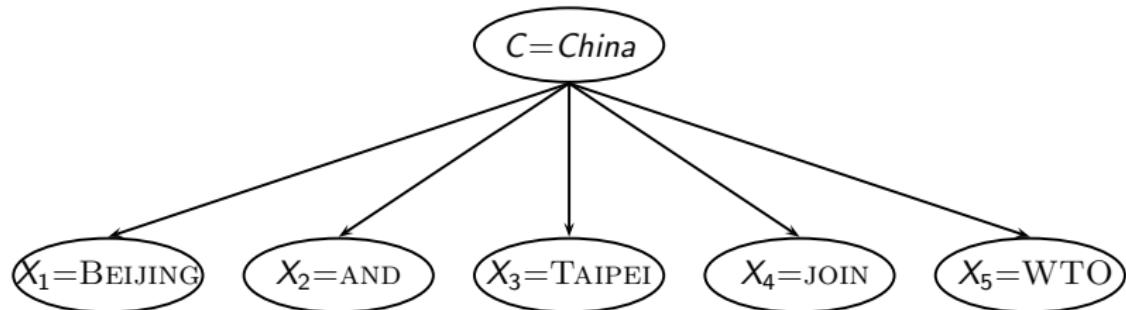


- Without add-one smoothing: if there are no occurrences of WTO in documents in class China, we get a zero estimate for the corresponding parameter:

$$\hat{P}(\text{WTO}|\text{China}) = \frac{T_{\text{China}, \text{WTO}}}{\sum_{t' \in V} T_{\text{China}, t'}} = 0$$

- With this estimate: $[d \text{ contains WTO}] \rightarrow [P(\text{China}|d) = 0]$.
- We must smooth to get a better estimate $P(\text{China}|d) > 0$.

Naive Bayes Generative Model



$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- Generate a class with probability $P(c)$
- Generate each of the words (in their respective positions), conditional on the class, but **independent of each other**, with probability $P(t_k|c)$

Take-away today

- Feature selection for text classification: How to select a subset of available dimensions
- Statistical language models: Introduction
- Statistical language models in IR
- Discussion: Properties of different probabilistic models in use in IR

Outline

- 1 Recap
- 2 Feature selection
- 3 Language models
- 4 Language Models for IR
- 5 Discussion

Feature selection

- In text classification, we usually represent documents in a **high-dimensional** space, with each dimension corresponding to a term.
- In this lecture: axis = dimension = word = term = feature
- Many dimensions correspond to rare words.
- Rare words can mislead the classifier.
- Rare misleading features are called **noise features**.
- Eliminating noise features from the representation **increases efficiency and effectiveness** of text classification.
- Eliminating features is called **feature selection**.

Example for a noise feature

- Let's say we're doing text classification for the class *China*.
- Suppose a rare term, say ARACHNOCENTRIC, has no information about *China* . . .
- . . . but all instances of ARACHNOCENTRIC happen to occur in *China* documents in our training set.
- Then we may learn a classifier that incorrectly interprets ARACHNOCENTRIC as evidence for the class *China*.
- Such an incorrect generalization from an accidental property of the training set is called **overfitting**.
- Feature selection reduces overfitting** and improves the accuracy of the classifier.

Basic feature selection algorithm

SELECTFEATURES(\mathbb{D}, c, k)

- 1 $V \leftarrow \text{EXTRACTVOCABULARY}(\mathbb{D})$
- 2 $L \leftarrow []$
- 3 **for each** $t \in V$
- 4 **do** $A(t, c) \leftarrow \text{COMPUTEFEATUREUTILITY}(\mathbb{D}, t, c)$
- 5 APPEND($L, \langle A(t, c), t \rangle$)
- 6 **return** FEATURESWITHLARGESTVALUES(L, k)

How do we compute A , the feature utility?

Different feature selection methods

- A feature selection method is mainly defined by the feature utility measure it employs
- Feature utility measures:
 - Frequency – select the most frequent terms
 - Mutual information – select the terms with the highest mutual information
 - Mutual information is also called **information gain** in this context.
 - Chi-square (see book)

Mutual information

- Compute the feature utility $A(t, c)$ as the **mutual information** (MI) of term t and class c .
- MI tells us “how much information” the term contains about the class and vice versa.
- For example, if a term’s occurrence is independent of the class (same proportion of docs within/without class contain the term), then MI is 0.
- Definition:

$$I(U; C) = \sum_{e_t \in \{1, 0\}} \sum_{e_c \in \{1, 0\}} P(U=e_t, C=e_c) \log_2 \frac{P(U=e_t, C=e_c)}{P(U=e_t)P(C=e_c)}$$

How to compute MI values

- Based on maximum likelihood estimates, the formula we actually use is:

$$\begin{aligned} I(U; C) = & \frac{N_{11}}{N} \log_2 \frac{NN_{11}}{N_{1\cdot} N_{\cdot 1}} + \frac{N_{01}}{N} \log_2 \frac{NN_{01}}{N_{0\cdot} N_{\cdot 1}} \\ & + \frac{N_{10}}{N} \log_2 \frac{NN_{10}}{N_{1\cdot} N_{\cdot 0}} + \frac{N_{00}}{N} \log_2 \frac{NN_{00}}{N_{0\cdot} N_{\cdot 0}} \end{aligned}$$

- N_{10} : number of documents that contain t ($e_t = 1$) and are not in c ($e_c = 0$); N_{11} : number of documents that contain t ($e_t = 1$) and are in c ($e_c = 1$); N_{01} : number of documents that do not contain t ($e_t = 1$) and are in c ($e_c = 1$); N_{00} : number of documents that do not contain t ($e_t = 1$) and are not in c ($e_c = 1$); $N = N_{00} + N_{01} + N_{10} + N_{11}$.

How to compute MI values (2)

- Alternative way of computing MI:

$$I(U; C) = \sum_{e_t \in \{1, 0\}} \sum_{e_c \in \{1, 0\}} P(U=e_t, C=e_c) \log_2 \frac{N(U=e_t, C=e_c)}{E(U=e_t)E(C=e_c)}$$

- $N(U=e_t, C=e_c)$ is the count of documents with values e_t and e_c .
- $E(U=e_t, C=e_c)$ is the expected count of documents with values e_t and e_c if we assume that the two random variables are independent.

MI example for *poultry*/EXPORT in Reuters

	$e_c = e_{\text{poultry}} = 1$	$e_c = e_{\text{poultry}} = 0$	
$e_t = e_{\text{EXPORT}} = 1$	$N_{11} = 49$	$N_{10} = 27,652$	Plug
$e_t = e_{\text{EXPORT}} = 0$	$N_{01} = 141$	$N_{00} = 774,106$	

these values into formula:

$$\begin{aligned} I(U; C) &= \frac{49}{801,948} \log_2 \frac{801,948 \cdot 49}{(49+27,652)(49+141)} \\ &\quad + \frac{141}{801,948} \log_2 \frac{801,948 \cdot 141}{(141+774,106)(49+141)} \\ &\quad + \frac{27,652}{801,948} \log_2 \frac{801,948 \cdot 27,652}{(49+27,652)(27,652+774,106)} \\ &\quad + \frac{774,106}{801,948} \log_2 \frac{801,948 \cdot 774,106}{(141+774,106)(27,652+774,106)} \\ &\approx 0.000105 \end{aligned}$$

MI feature selection on Reuters

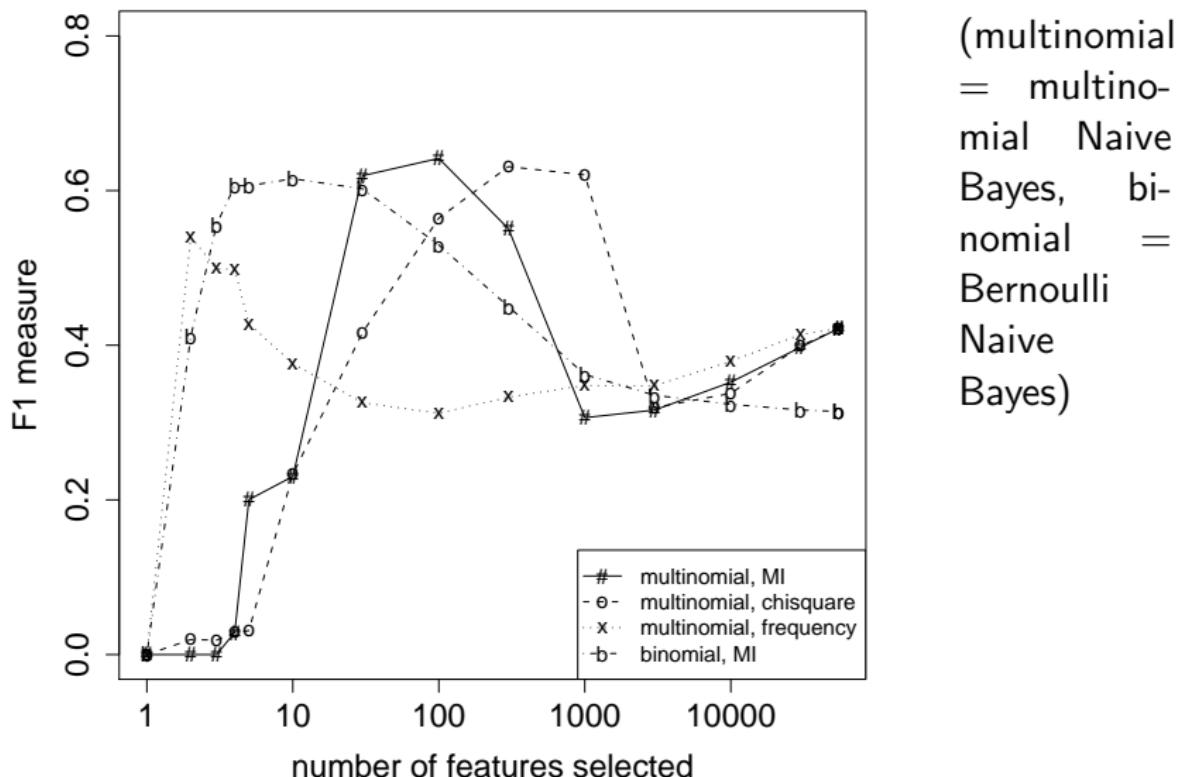
Class: *coffee*

term	MI
COFFEE	0.0111
BAGS	0.0042
GROWERS	0.0025
KG	0.0019
COLOMBIA	0.0018
BRAZIL	0.0016
EXPORT	0.0014
EXPORTERS	0.0013
EXPORTS	0.0013
CROP	0.0012

Class: *sports*

term	MI
SOCKER	0.0681
CUP	0.0515
MATCH	0.0441
MATCHES	0.0408
PLAYED	0.0388
LEAGUE	0.0386
BEAT	0.0301
GAME	0.0299
GAMES	0.0284
TEAM	0.0264

Naive Bayes: Effect of feature selection



Feature selection for Naive Bayes

- In general, feature selection is necessary for Naive Bayes to get decent performance.
- Also true for many other learning methods in text classification: **you need feature selection for optimal performance.**

Exercise

(i) Compute the “export” /POULTRY contingency table for the “Kyoto” / JAPAN in the collection given below. (ii) Make up a contingency table for which MI is 0 – that is, term and class are independent of each other. “export” /POULTRY table:

	$e_c = e_{poultry} = 1$	$e_c = e_{poultry} = 0$
$e_t = e_{EXPORT} = 1$	$N_{11} = 49$	$N_{10} = 27,652$
$e_t = e_{EXPORT} = 0$	$N_{01} = 141$	$N_{00} = 774,106$

Collection:

	docID	words in document	in $c = \text{Japan?}$
training set	1	Kyoto Osaka Taiwan	yes
	2	Japan Kyoto	yes
	3	Taipei Taiwan	no
	4	Macao Taiwan Shanghai	no
	5	London	no

Outline

- 1 Recap
- 2 Feature selection
- 3 Language models
- 4 Language Models for IR
- 5 Discussion

Using language models (LMs) for IR

- ➊ LM = language model
- ➋ We view the document as a generative model that generates the query.
- ➌ What we need to do:
- ➍ Define the precise generative model we want to use
- ➎ Estimate parameters (different parameters for each document's model)
- ➏ Smooth to avoid zeros
- ➐ Apply to query and find document most likely to have generated the query
- ➑ Present most likely document(s) to user
- ➒ Note that 4–7 is very similar to what we did in Naive Bayes.

What is a language model?

We can view a finite state automaton as a deterministic language



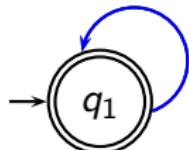
model.

I wish I wish I wish I wish ... Cannot generate: "wish I wish"

or "I wish I" Our basic model: each document was generated by a

different automaton like this except that these automata are probabilistic.

A probabilistic language model



w	$P(w q_1)$	w	$P(w q_1)$
STOP	0.2	toad	0.01
the	0.2	said	0.03
a	0.1	likes	0.02
frog	0.01	that	0.04
	

This

is a **one-state probabilistic finite-state automaton** – a **unigram language model** – and the state emission distribution for its one state q_1 . STOP is not a word, but a special symbol indicating that

the automaton stops. frog said that toad likes frog STOP

$$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$$

$$= 0.0000000000048$$

A different language model for each document

language model of d_1		language model of d_2	
w	$P(w .)$	w	$P(w .)$
STOP	.2	toad	.01
the	.2	said	.03
a	.1	likes	.02
frog	.01	that	.04
	

query: frog said that toad likes frog STOP $P(\text{query}|M_{d1}) = 0.01$

$$\cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2 = 0.000000000048 = 4.8 \cdot 10^{-12}$$

$$P(\text{query}|M_{d2}) = 0.01 \cdot 0.03 \cdot 0.05 \cdot 0.02 \cdot 0.02 \cdot 0.01 \cdot 0.2$$

$$= 0.0000000000120 = 12 \cdot 10^{-12} \quad P(\text{query}|M_{d1}) < P(\text{query}|M_{d2})$$

Thus, document d_2 is “more relevant” to the query “frog said that toad likes frog STOP” than d_1 is.

Outline

- 1 Recap
- 2 Feature selection
- 3 Language models
- 4 Language Models for IR
- 5 Discussion

Using language models in IR

- Each document is treated as (the basis for) a language model.
- Given a query q
- Rank documents based on $P(d|q)$

•

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}$$

- $P(q)$ is the same for all documents, so ignore
- $P(d)$ is the prior – often treated as the same for all d
 - But we can give a higher prior to “high-quality” documents, e.g., those with high PageRank.
- $P(q|d)$ is the probability of q given d .
- For uniform prior: ranking documents according to $P(q|d)$ and $P(d|q)$ is equivalent.

Where we are

- In the LM approach to IR, we attempt to model the query generation process.
- Then we rank documents by the probability that a query would be observed as a random sample from the respective document model.
- That is, we rank according to $P(q|d)$.
- Next: how do we compute $P(q|d)$?

How to compute $P(q|d)$

- We will make the same conditional independence assumption as for Naive Bayes.
-

$$P(q|M_d) = P(\langle t_1, \dots, t_{|q|} \rangle | M_d) = \prod_{1 \leq k \leq |q|} P(t_k | M_d)$$

($|q|$: length of q ; t_k : the token occurring at position k in q)

- This is equivalent to:

$$P(q|M_d) = \prod_{\text{distinct term } t \text{ in } q} P(t | M_d)^{\text{tf}_{t,q}}$$

- $\text{tf}_{t,q}$: term frequency (# occurrences) of t in q
- Multinomial model (omitting constant factor)

Parameter estimation

- Missing piece: Where do the parameters $P(t|M_d)$ come from?
- Start with maximum likelihood estimates (as we did for Naive Bayes)
-

$$\hat{P}(t|M_d) = \frac{\text{tf}_{t,d}}{|d|}$$

($|d|$: length of d ; $\text{tf}_{t,d}$: # occurrences of t in d)

- As in Naive Bayes, we have a problem with zeros.
- A single t with $P(t|M_d) = 0$ will make $P(q|M_d) = \prod P(t|M_d)$ zero.
- We would give a single term “veto power”.
- For example, for query [Michael Jackson top hits] a document about “top songs” (but not using the word “hits”) would have $P(q|M_d) = 0$. – That's bad.
- We need to smooth the estimates to avoid zeros.

Smoothing

- Key intuition: A nonoccurring term is possible (even though it didn't occur), ...
- ... but no more likely than would be expected by chance in the collection.
- Notation: M_c : the collection model; cf_t : the number of occurrences of t in the collection; $T = \sum_t \text{cf}_t$: the total number of tokens in the collection.

-

$$\hat{P}(t|M_c) = \frac{\text{cf}_t}{T}$$

- We will use $\hat{P}(t|M_c)$ to “smooth” $P(t|d)$ away from zero.

Jelinek-Mercer smoothing

- $P(t|d) = \lambda P(t|M_d) + (1 - \lambda)P(t|M_c)$
- Mixes the probability from the document with the general collection frequency of the word.
- High value of λ : “conjunctive-like” search – tends to retrieve documents containing all query words.
- Low value of λ : more disjunctive, suitable for long queries
- Correctly setting λ is very important for good performance.

Jelinek-Mercer smoothing: Summary



$$P(q|d) \propto \prod_{1 \leq k \leq |q|} (\lambda P(t_k|M_d) + (1 - \lambda)P(t_k|M_c))$$

- What we model: The user has a document in mind and generates the query from this document.
- The equation represents the probability that the document that the user had in mind was in fact this one.

Example

- Collection: d_1 and d_2
- d_1 : Jackson was one of the most talented entertainers of all time
- d_2 : Michael Jackson anointed himself King of Pop
- Query q : Michael Jackson
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(0/11 + 1/18)/2] \cdot [(1/11 + 2/18)/2] \approx 0.003$
- $P(q|d_2) = [(1/7 + 1/18)/2] \cdot [(1/7 + 2/18)/2] \approx 0.013$
- Ranking: $d_2 > d_1$

Exercise: Compute ranking

- Collection: d_1 and d_2
- d_1 : Xerox reports a profit but revenue is down
- d_2 : Lucene narrows quarter loss but revenue decreases further
- Query q : revenue down
- Use mixture model with $\lambda = 1/2$
- $P(q|d_1) = [(1/8 + 2/16)/2] \cdot [(1/8 + 1/16)/2] = 1/8 \cdot 3/32 = 3/256$
- $P(q|d_2) = [(1/8 + 2/16)/2] \cdot [(0/8 + 1/16)/2] = 1/8 \cdot 1/32 = 1/256$
- Ranking: $d_1 > d_2$

Dirichlet smoothing



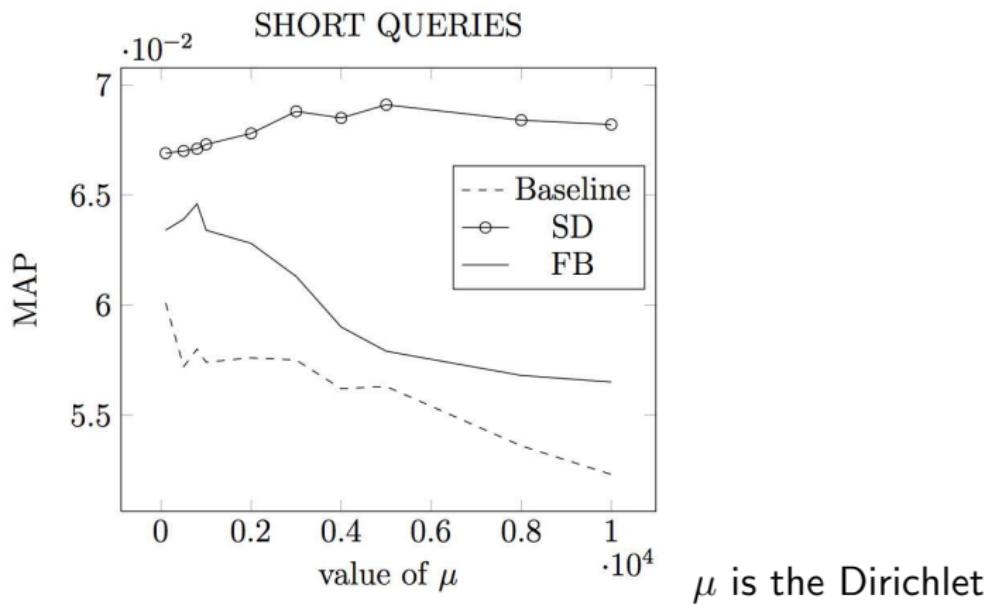
$$\hat{P}(t|d) = \frac{\text{tf}_{t,d} + \alpha \hat{P}(t|M_c)}{L_d + \alpha}$$

- The background distribution $\hat{P}(t|M_c)$ is the prior for $\hat{P}(t|d)$.
- Intuition: Before having seen any part of the document we start with the background distribution as our estimate.
- As we read the document and count terms we update the background distribution.
- The weighting factor α determines how strong an effect the prior has.

Jelinek-Mercer or Dirichlet?

- Dirichlet performs better for keyword queries, Jelinek-Mercer performs better for verbose queries.
- Both models are sensitive to the smoothing parameters – you shouldn't use these models without parameter tuning.

Sensitivity of Dirichlet to smoothing parameter



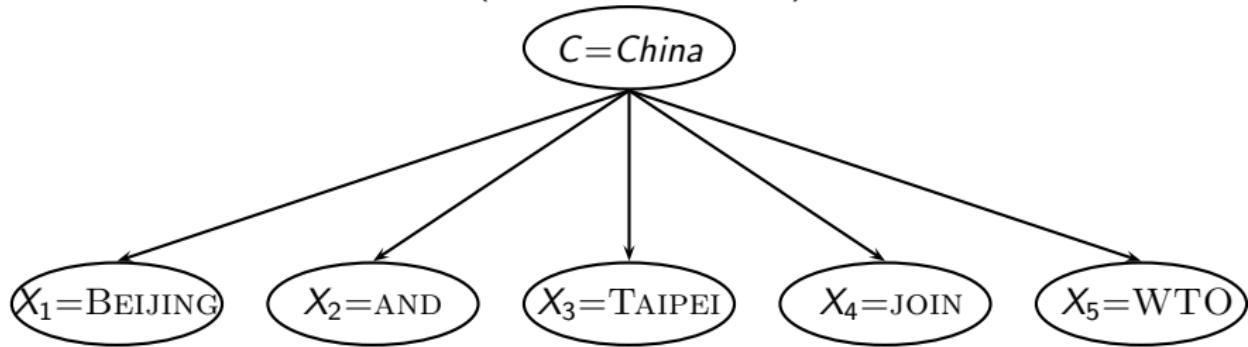
smoothing parameter (called α on the previous slides)

Outline

- 1 Recap
- 2 Feature selection
- 3 Language models
- 4 Language Models for IR
- 5 Discussion

Language models are generative models

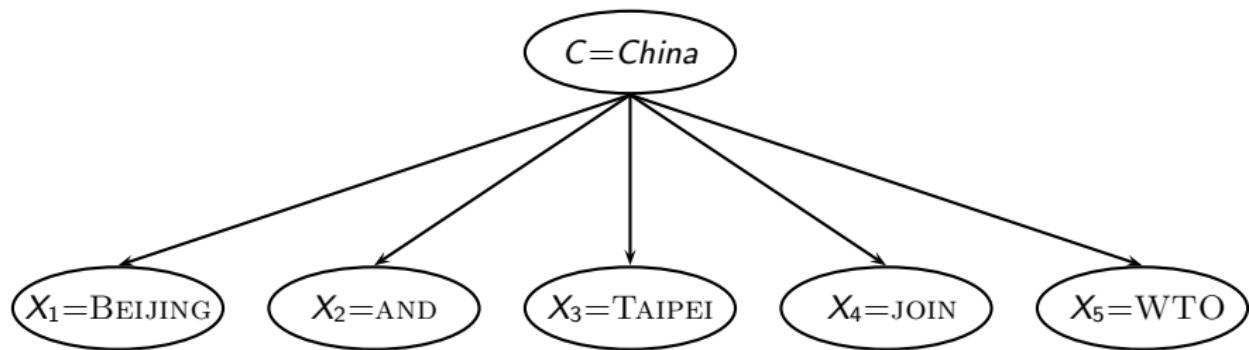
We have assumed that queries are generated by a probabilistic process that looks like this: (as in Naive Bayes)



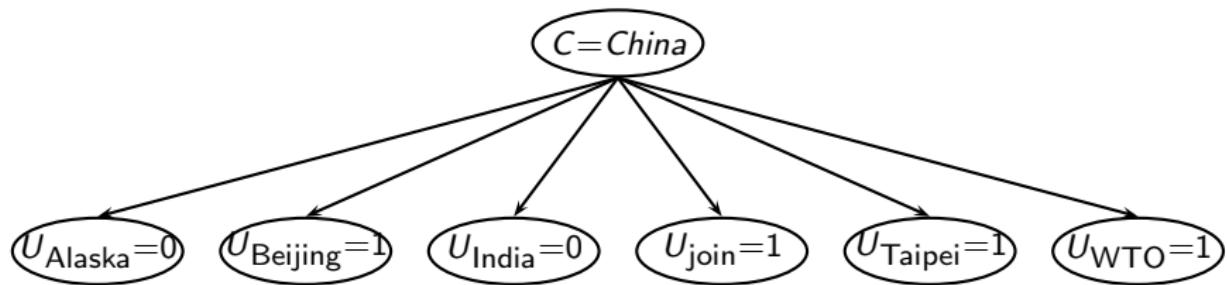
Naive Bayes and LM generative model s

- We want to classify document d .
We want to classify a query q .
 - Classes: e.g., geographical regions like *China, UK, Kenya*.
Each document in the collection is a different class.
- Assume that d was generated by the generative model.
Assume that q was generated by a generative model
- Key question: Which of the classes is most likely to have generated the document? *Which document (=class) is most likely to have generated the query q ?*
 - Or: for which class do we have the most evidence? *For which document (as the source of the query) do we have the most evidence?*

Naive Bayes Multinomial model / IR language models



Naive Bayes Bernoulli model / Binary independence model



Comparison of the two models

	multinomial model / IR language model	Bernoulli model / BIM
event model	generation of (multi)set of tokens	generation of subset of vocabulary
random variable(s)	$X = t$ iff t occurs at given pos	$U_t = 1$ iff t occurs in doc
doc. representation	$d = \langle t_1, \dots, t_k, \dots, t_{n_d} \rangle, t_k \in V$	$d = \langle e_1, \dots, e_i, \dots, e_M \rangle, e_i \in \{0, 1\}$
parameter estimation	$\hat{P}(X = t c)$	$\hat{P}(U_i = e c)$
dec. rule: maximize	$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(X = t_k c)$	$\hat{P}(c) \prod_{t_i \in V} \hat{P}(U_i = e_i c)$
multiple occurrences	taken into account	ignored
length of docs	can handle longer docs	works best for short docs
# features	can handle more	works best with fewer
estimate for THE	$\hat{P}(X = \text{the} c) \approx 0.05$	$\hat{P}(U_{\text{the}} = 1 c) \approx 1.0$

Vector space (tf-idf) vs. LM

Rec.	precision			significant
	tf-idf	LM	%chg	
0.0	0.7439	0.7590	+2.0	
0.1	0.4521	0.4910	+8.6	
0.2	0.3514	0.4045	+15.1	*
0.4	0.2093	0.2572	+22.9	*
0.6	0.1024	0.1405	+37.1	*
0.8	0.0160	0.0432	+169.6	*
1.0	0.0028	0.0050	+76.9	
11-point average	0.1868	0.2233	+19.6	*

The

language modeling approach always does better in these experiments but note that where the approach shows significant gains is at higher levels of recall.

Vector space vs BM25 vs LM

- BM25/LM: based on probability theory
- Vector space: based on similarity, a geometric/linear algebra notion
- Term frequency is directly used in all three models.
 - LMs: raw term frequency, BM25/Vector space: more complex
- Length normalization
 - Vector space: Cosine or pivot normalization
 - LMs: probabilities are inherently length normalized
 - BM25: tuning parameters for optimizing length normalization
- idf: BM25/vector space use it directly.
- LMs: Mixing term and collection frequencies has an effect similar to idf.
 - Terms rare in the general collection, but common in some documents will have a greater influence on the ranking.
- Collection frequency (LMs) vs. document frequency (BM25, vector space)



Language models for IR: Assumptions

- Simplifying assumption: **Queries and documents are objects of the same type.** Not true!
 - There are other LMs for IR that do not make this assumption.
 - The vector space model makes the same assumption.
- Simplifying assumption: **Terms are conditionally independent.**
 - Again, vector space model (and Naive Bayes) make the same assumption.
- Cleaner statement of assumptions than vector space
- Thus, better theoretical foundation than vector space
 - ... but “pure” LMs perform much worse than “tuned” LMs.

Take-away today

- Feature selection for text classification: How to select a subset of available dimensions
- Statistical language models: Introduction
- Statistical language models in IR
- Discussion: Properties of different probabilistic models in use in IR

Resources

- Chapter 13 of IIR (feature selection)
- Chapter 12 of IIR (language models)
- Resources at <http://cislmu.org>
 - Ponte and Croft's 1998 SIGIR paper (one of the first on LMs in IR)
 - Zhai and Lafferty: A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst. (2004).
 - Lemur toolkit (good support for LMs in IR)

Computing PageRank & HITS

Fabrizio Silvestri

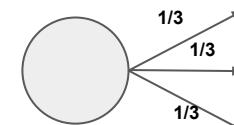


SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Recap: The Google Problem

- Imagine a user doing a random walk on web pages:
 - Start at a random page
 - At each step, go out of the current page along one of the links on that page, equiprobably
- In the long run" each page has a long-term visit rate – use this as the page's score
- Variant: rather than equiprobable, use text and link information to have probability of following a link: intelligent surfer

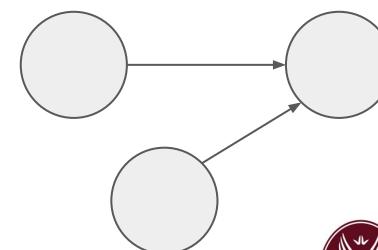


Recap: Ergodic Markov Chains

- For any ergodic Markov chain, there is a unique long-term visit rate for each state.
 - Steady-state probability distribution.
 - Over a long time-period, we visit each state in proportion to this rate.
 - It doesn't matter where we start.
- Ergodic: no periodic patterns
 - Teleportation ensures ergodicity



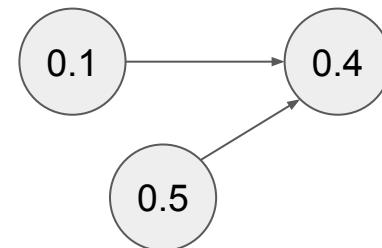
Not Ergodic



Recap: Probability Vectors

- A probability (row) vector $x = (x_1, \dots, x_n)$ tells us where the walk is at any point.
- E.g., $(000\dots 1\dots 000)$ means we're in state i .

i



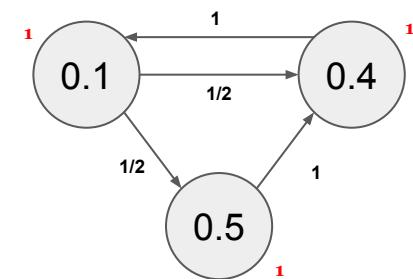
- More generally, the vector $x = (x_1, \dots, x_n)$ means the walk is in state i with probability x_i .
 - $x_1 + x_2 + \dots + x_n = 1$



Recap: Dynamics of Markov Chains

- $u_0 = (0.1, 0.4, 0.5)$

- $A = \begin{pmatrix} 0, & 0.5, & 0.5 \\ 1, & 0, & 0 \\ 0, & 1, & 0 \end{pmatrix}$



- $u_1 = u_0 A = (0.4, 0.55, 0.05)$
- $u_2 = u_1 A = (u_0 A)A = u_0 A^2 = (0.55, 0.25, 0.2)$
- $u_3 = u_0 A^3 = (0.25, 0.475, 0.275)$



Perron-Frobenius Theorem

Theorem (5)

Let $A \geq 0$ be an irreducible $n \times n$ matrix. Then,

- 1. A has a positive real eigenvalue equal to its spectral radius $\rho(A)$.
- 2. To $\rho(A)$ there corresponds an eigenvector $x > 0$.
- 3. $\rho(A)$ increases when any entry of A increases.
- 4. $\rho(A)$ is a simple eigenvalue of A .
- 5. There is not other nonnegative eigenvector of A different from x .

The largest Eigenvalue of a Stochastic Matrix is 1

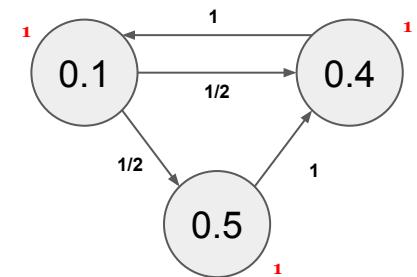
1. If A is a stochastic matrix, then $A\mathbf{1} = \mathbf{1}$, since each row of A sums to 1. Therefore, 1 is an eigenvalue of A .
2. Suppose there exists $\lambda > 1$ and nonzero \mathbf{x} such that $\mathbf{x}A = \lambda\mathbf{x}$, and let x_i be the largest element of \mathbf{x} .
 - a. Since any scalar multiple of \mathbf{x} will also satisfy this equation we can assume, without loss of generality, that $x_i > 0$.
 - b. Since the rows of A are nonnegative and sum to 1, each entry in $\lambda\mathbf{x}$ is a convex combination of the elements of \mathbf{x} . Thus no entry in $\lambda\mathbf{x}$ can be larger than x_i . But since $\lambda > 1$, $\lambda x_i > x_i$, which is a contradiction.

Therefore, the largest eigenvalue of A is 1.



PageRank as an EigenProblem

- $u_m = u_{m-1}A$
 - $u = uA \Rightarrow 1u = uA$
- The stable distribution u is an (left) eigenvector of A
- In fact, A being stochastic, and positive definite has all real eigenvalues $1 = \lambda_1 > \lambda_2 > \dots > \lambda_k$
 - 1 is the largest eigenvalues
- So, the stable distribution is the principal eigenvector of A

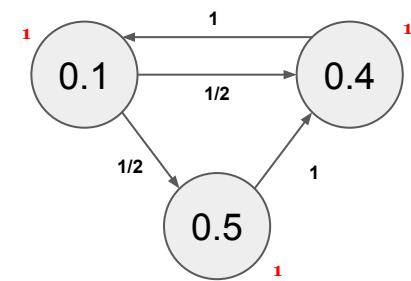


Observation: The “Shape” of A^k

$$A = \begin{pmatrix} 0, & 0.5, & 0.5 \\ 1, & 0, & 1 \\ 0, & 1, & 0 \end{pmatrix}$$

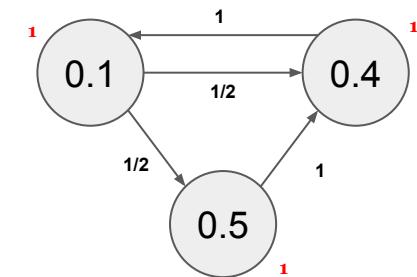
$$A^5 = \begin{pmatrix} 0.5, & 0.375, & 0.125 \\ 0.25, & 0.5, & 0.25 \\ 0.5, & 0.25, & 0.25 \end{pmatrix}$$

$$A^{60} = \begin{pmatrix} 0.4, & 0.4, & 0.2 \\ 0.4, & 0.4, & 0.2 \\ 0.4, & 0.4, & 0.2 \end{pmatrix}$$



How Do You Find It? The Power Method

- First suppose A is diagonalizable
 - $A = P\Lambda P^{-1}$
 - $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n)$
- x_i is the eigenvector corresponding to λ_i
- Let's consider $u_0 = \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n$
- $u_0 A^k = \sum \alpha_j x_j A^k = \sum \alpha_j \lambda_j^k x_j = \lambda_1^k (\alpha_1 x_1 + \sum_{j>1} \alpha_j (\lambda_j / \lambda_1)^k x_j)$
- Therefore $\lim_{k \rightarrow \infty} (u_0 A^k / \lambda_1^k) = \alpha_1 x_1$



How Do You Find It? The Power Method

1. Choose a starting vector $\mathbf{u}_0 \in \mathbb{R}^n$ with $\|\mathbf{u}_0\|_1 = 1$

2. $k = 0$;

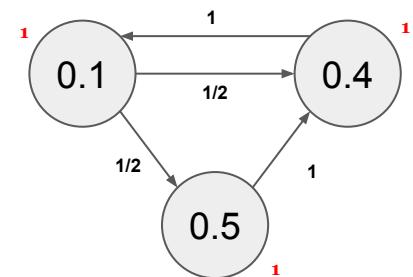
3. repeat

$$k := k + 1;$$

$$\mathbf{y} := \mathbf{x}_{k-1} A;$$

$$\mathbf{x}_k := \mathbf{y} / \|\mathbf{y}\|_1$$

4. until a convergence criterion is satisfied



The rate of convergence is directly proportional to the ratio λ_2 / λ_1



How Do You Find It? The Power Method

```
[ ] 1 import numpy as np
2
3 A = np.array(
4     [
5         [0, 0.5, 0.5],
6         [1, 0, 0],
7         [0, 1, 0],
8     ]
9 )
```



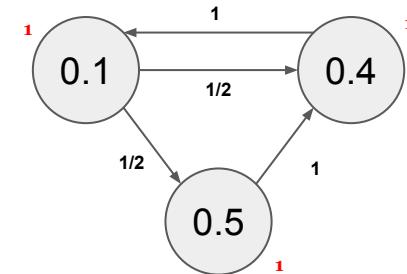
```
[ ] 1 u = np.array([1, 0, 0])
2 k = 0
3 max_iter = 100
4 converged = False
5 epsilon = 1e-10
6 while not converged:
7     k += 1
8     old_u = np.copy(u)
9     y = np.dot(u, A)
10    u = y / np.linalg.norm(y, 1)
11    if np.linalg.norm(old_u - u) < epsilon or k > max_iter:
12        converged = True
13
14 print("The method converged to {} in {} iterations.".format(u, k))
```

The method converged to [0.4 0.4 0.2] in 67 iterations.



```
1 print(np.dot(u, A))
```

[0.4 0.4 0.2]



Accelerating PageRank Computation

- Web Graph Compression to fit in internal memory
- Efficient external-memory implementation
- Distributed PageRank Computation
- Mathematical approaches
- Combination of the above strategies



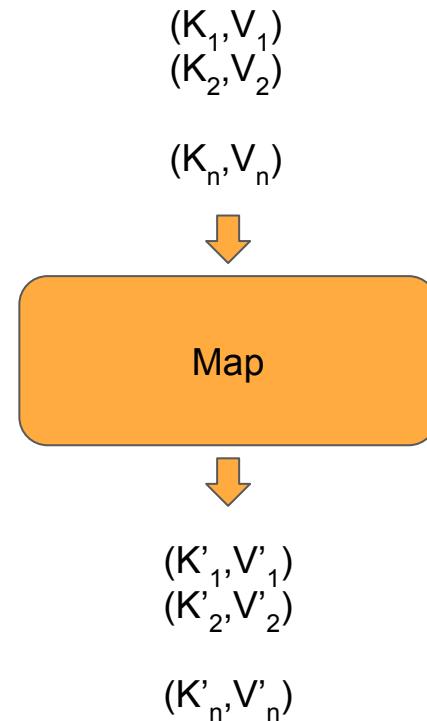
MapReduce

- What is it?
 - Programming model originally proposed by Google
 - A combination of the Map and Reduce models with an associated implementation
 - Used for processing and generating large data sets
- MapReduce is highly scalable and can be used across many computers.
- Many small machines can be used to process jobs that normally could not be processed by a large machine.



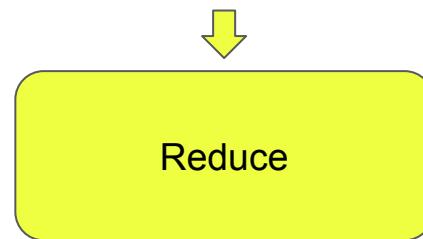
Map Abstraction

- Inputs a key/value pair
 - Key is a reference to the input value
 - Value is the data set on which to operate
- Evaluation
 - Function defined by user
 - Applies to every value in value input
 - Might need to parse input
- Produces a new list of key/value pairs
 - Can be different type from input pair



Reduce Abstraction

- Starts with intermediate Key / Value pairs
- Ends with finalized Key / Result (Value) pairs
- Starting pairs are sorted by key
- Iterator supplies the values for a given key to the Reduce function.
- Typically a function that:
 - Starts with a large number of key/value pairs
 - One key/value for each word in all files being grepped (including multiple entries for the same word)
 - Ends with very few key/value pairs
 - One key/value for each unique word across all the files with the number of instances summed into this entry
- Broken up so a given worker works with input of the same key.

$$(K_1, V_{11}, V_{12}, V_{13}, \dots, V_{1m})$$
$$(K_2, V_{21}, V_{22}, V_{23}, \dots, V_{1g})$$
$$(K_n, V_{n1}, V_{n2}, V_{n3}, \dots, V_{nl})$$

$$(K_1, R_1)$$
$$(K_2, R_2)$$
$$(K_n, R_n)$$


Reduce Example

```
def reduce(key, listOfValues):  
    result = 0  
    for x in listOfValues:  
        result += x  
    return (key, result)
```



PageRank: Map

```
map(key: [url, pagerank], value: outlink_list)
    for each outlink in outlink_list
        emit(key: outlink, value: pagerank/len(outlink_list))
    emit(key: url, value: outlink_list)
```



PageRank: Reduce

```
reduce(key: url, value: list_pr_or_urls)
    outlink_list = []
    pagerank = 0

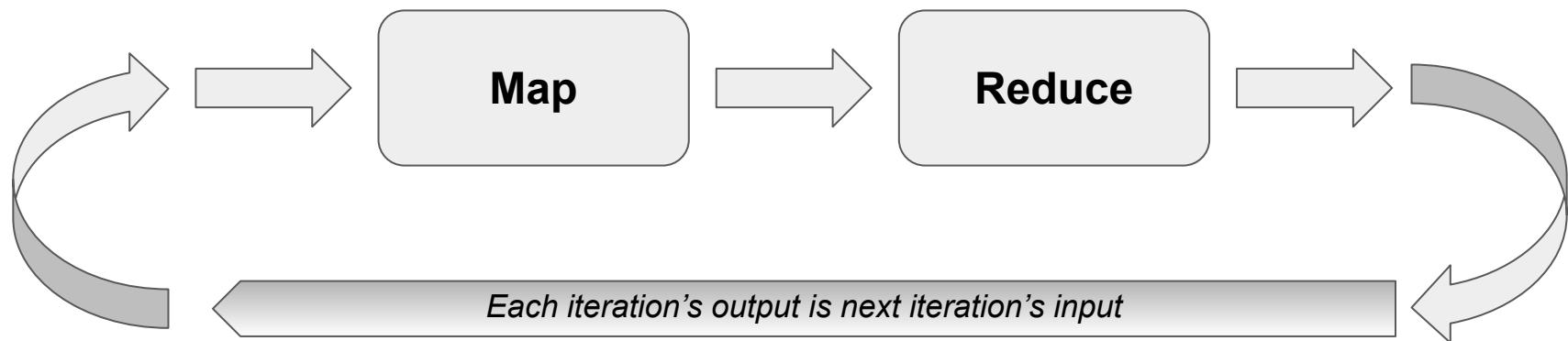
    for each pr_or_urls in list_pr_or_urls
        if is_list(pr_or_urls)
            outlink_list = pr_or_urls
        else
            pagerank += pr_or_urls

    pagerank = 1 - DAMPING_FACTOR + ( DAMPING_FACTOR * pagerank )

    emit(key: [url, pagerank], value: outlink_list)
```



PageRank: MapReduce



MapReduce's PageRank Issue

- The problem of stragglers in real world's graphs
- Number of iterations unknown in advance, how do you decide how to spawn Mappers and Reducers?



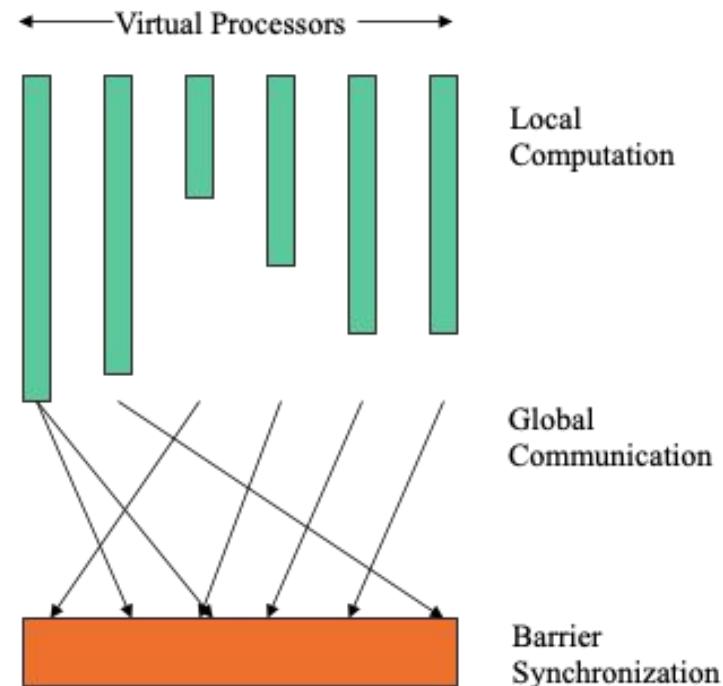
Bulk Synchronous Parallelism

- Computational model of parallel computation consisting of:
 - A set of processor-memory pairs.
 - A communications network that delivers messages in a point-to-point manner.
 - A mechanism for the efficient barrier synchronization for all or a subset of the processes.
 - There are no special combining, replicating, or broadcasting facilities.

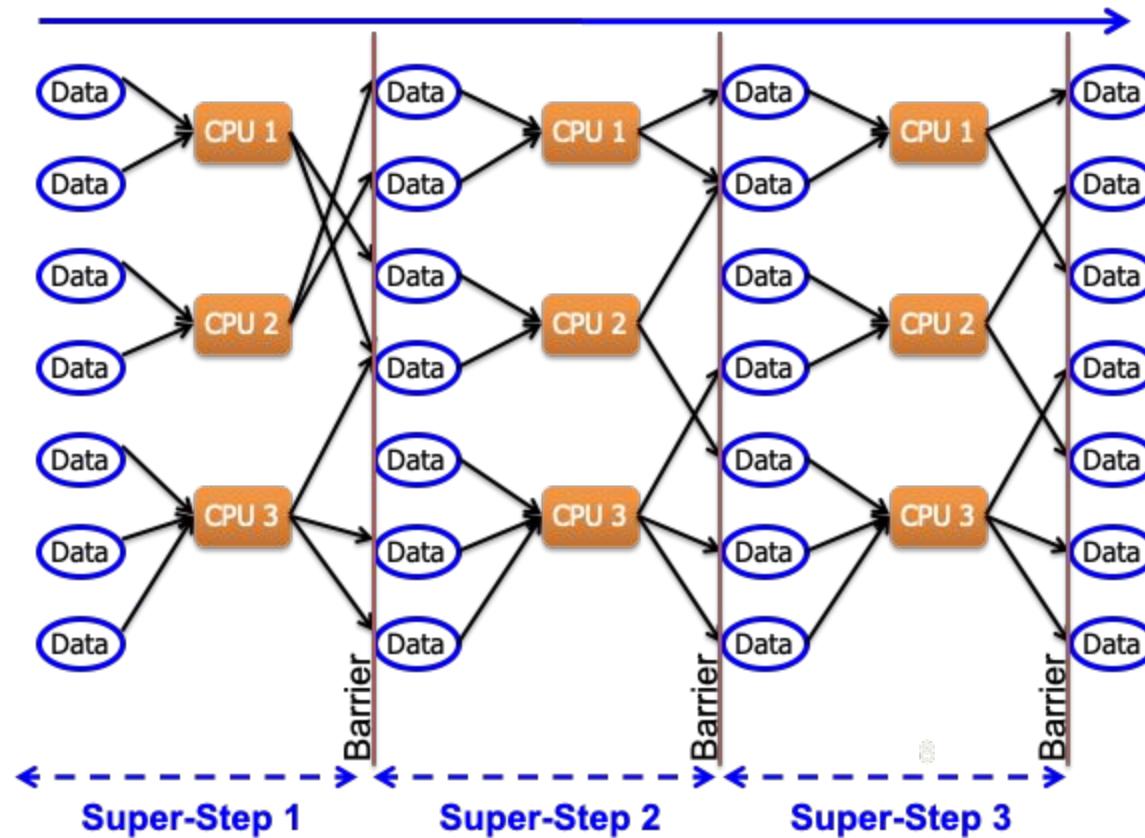


What does the BSP Programming Style Look Like?

- Vertical Structure
 - Sequential composition of “supersteps”.
 - Local computation
 - Process Communication
 - Barrier Synchronization
- Horizontal Structure
 - Concurrency among a fixed number of virtual processors.
 - Processes do not have a particular order.
 - Locality plays no role in the placement of processes on processors.
 - $p = \text{number of processors}$.



What does the BSP Programming Style Look Like?



BSP Programming Style

- Properties:
 - Simple to write programs.
 - Independent of target architecture.
 - Performance of the model is predictable.
- Considers computation and communication at the level of the entire program and executing computer instead of considering individual processes and individual communications.
- Renounces locality as a performance optimization.
 - Good and bad
 - BSP may not be the best choice for which locality is critical i.e. low-level image processing.



Google's Pregel

- Pregel is a large-scale graph-parallel distributed analytics engine inspired by BSP
- Some Characteristics:
 - In-Memory (opposite to MapReduce)
 - High scalability
 - Automatic fault-tolerance
 - Flexibility in expressing graph algorithms
 - Message-Passing programming model
 - Tree-style, master-slave architecture
 - Synchronous



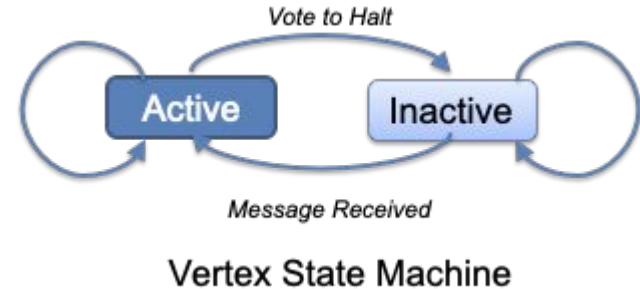
Entities and Super-Steps

- The computation is described in terms of vertices, edges and a sequence of super-steps
- You give Pregel a directed graph consisting of vertices and edges
 - Each vertex is associated with a modifiable user-defined value
 - Each edge is associated with a source vertex, value and a destination vertex
- During a super-step:
 - A user-defined function F is executed at each vertex V
 - F can read messages sent to V in superset $S - 1$ and send messages to other vertices that will be received at superset $S + 1$
 - F can modify the state of V and its outgoing edges
 - F can alter the topology of the graph

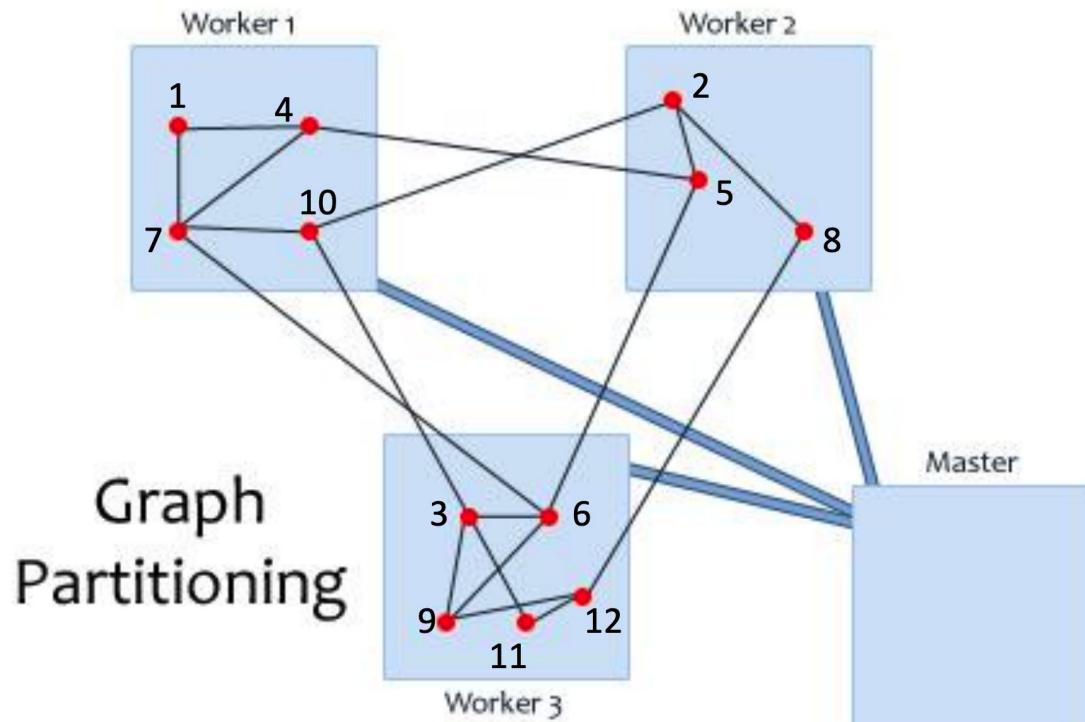


Algorithm Termination

- Algorithm termination is based on every vertex voting to halt
 - In super-step 0, every vertex is active
 - All active vertices participate in the computation of any given super-step
 - A vertex deactivates itself by voting to halt and enters an inactive state
 - A vertex can return to active state if it receives an external message
- A Pregel program terminates when all vertices are simultaneously inactive and there are no messages in transit



Graph Partitioning



The Pregel API in C++

You subclass the Vertex Class

```
template <typename VertexValue,  
typename EdgeValue,  
typename MessageValue>  
  
class Vertex {  
public:  
    virtual void Compute(MessageIterator* msgs) = 0;  
  
    const string& vertex_id() const;  
    int64 superstep() const;  
    const VertexValue& GetValue();  
    VertexValue* MutableValue();  
    OutEdgeIterator GetOutEdgeIterator();  
  
    void SendMessageTo(const string& dest_vertex,  
                      const MessageValue& message);  
  
    void VoteToHalt();  
};
```

To define the types for vertices, edges and messages

Override the compute function to define the computation at each superstep

To get the value of the current vertex

To modify the value of the vertex

To pass messages to other vertices



PageRank in Pregel

```
class PageRankVertex:public Vertex<double, void, double> {
public:
    virtual void Compute (MessageIterator * msgs)
    {
        const double DAMPING_FACTOR = 0.85;
        const int64 MAX_ITER = 30;
        if (superstep() >= 1) {
            double sum = 0;
            for ( ; !msgs->done(); msgs->Next())
                sum += msgs->Value();
            *MutableValue() = (1 - DAMPING_FACTOR) + (DAMPING_FACTOR * sum);
        }
        if (supersteps() < MAX_ITER) {
            const int64 n = GetOutEdgeIterator().size();
            SendMessageToAllNeighbors(GetValue() / n);
        } else {
            VoteToHalt();
        }
    }
};
```



Using Pregel in Apache Spark (PageRank implementation)

```
import org.apache.spark.graphx.GraphLoader

// Load the edges as a graph
val graph = GraphLoader.edgeListFile(sc, "data/graphx/followers.txt")
// Run PageRank
val ranks = graph.pageRank(0.0001).vertices
// Join the ranks with the usernames
val users = sc.textFile("data/graphx/users.txt").map { line =>
    val fields = line.split(",")
    (fields(0).toLong, fields(1))
}
val ranksByUsername = users.join(ranks).map {
    case (id, (username, rank)) => (username, rank)
}
// Print the result
println(ranksByUsername.collect().mkString("\n"))
```



HITS - Kleinberg 1999

- **Hyperlink-Induced Topic Search (HITS)**
- In response to a query, instead of an ordered list of pages each meeting the query, find two sets of inter-related pages:
 - Hub pages are good lists of links on a subject
 - e.g., “Bob’s list of cancer-related links.”
 - Authority pages occur recurrently on good hubs for the subject
- Best suited for “broad topic” queries rather than for page-finding queries
- Gets at a broader slice of common opinion

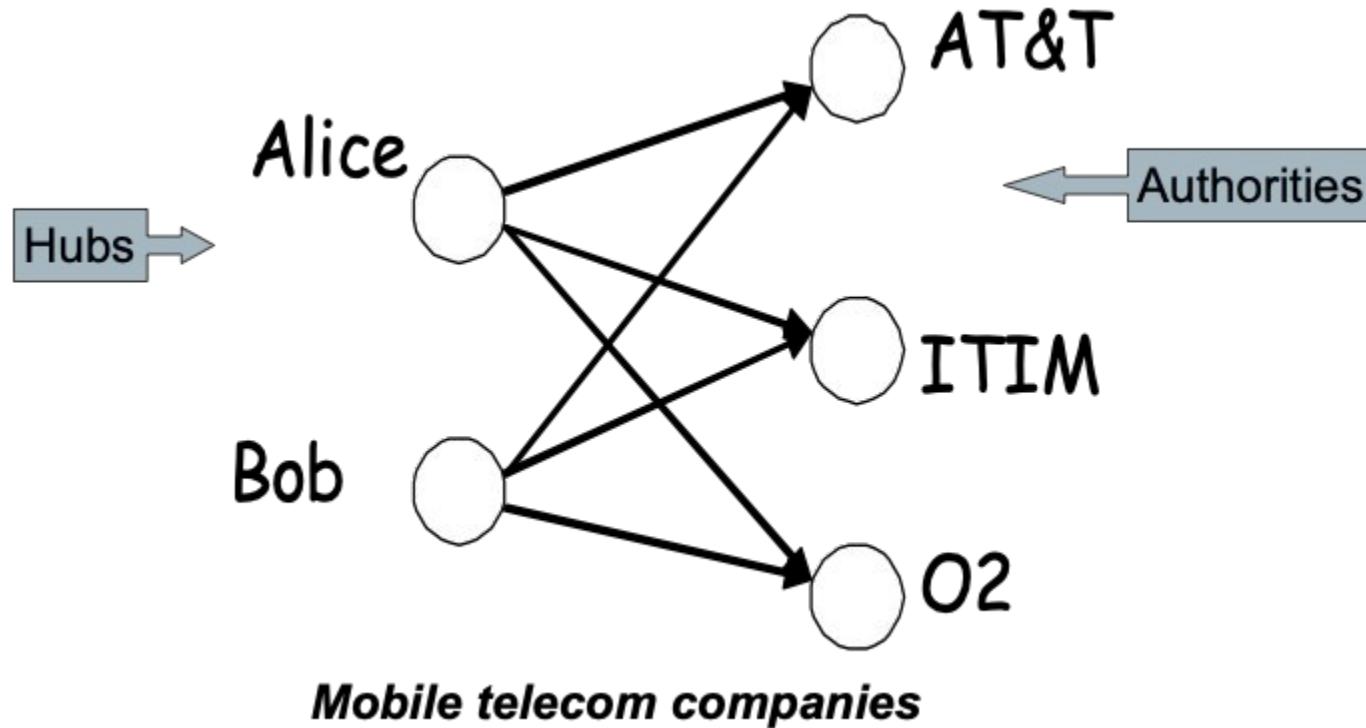


Hubs and Authorities

- Thus, a good hub page for a topic points to many authoritative pages for that topic.
- A good authority page for a topic is pointed to by many good hubs for that topic.
- Circular definition – will turn this into an iterative computation.



Ideally



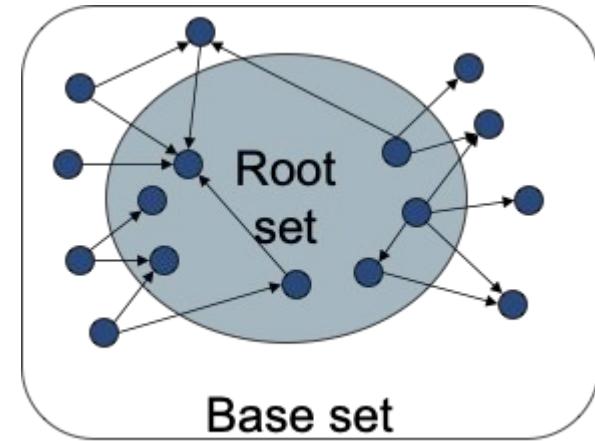
The High Level Algorithm

- Extract from the web a **base set** of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;
→ iterative algorithm.



The Base Set

- Given text query (say browser), use a text index to get all pages containing browser.
 - Call this the **root set** of pages.
- Add in any page that either
 - points to a page in the root set, or
 - is pointed to by a page in the root set.
- Call this the **base set**.



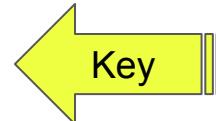
Get in-links (and out-links) from a *connectivity server* (a.k.a. Graph DB)



SAPIENZA
UNIVERSITÀ DI ROMA

The High Level Algorithm

- Compute, for each page x in the base set, a hub score $h(x)$ and an authority score $a(x)$.
- Initialize: for all x , $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;
- Iteratively update all $h(x)$, $a(x)$;
- After iterations
 - output pages with highest $h()$ scores as top hubs
 - highest $a()$ scores as top authorities.

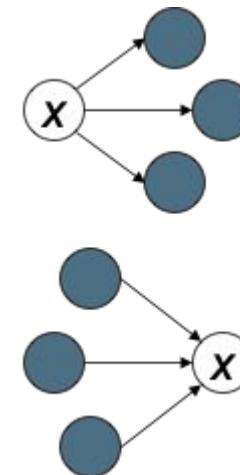


Iterative Update of $h(x)$ and $a(x)$

- Repeat the following updates, for all x :

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$



Scaling

- To prevent the $h()$ and $a()$ values from getting too big, can scale down after each iteration.
- Scaling factor doesn't really matter:
 - we only care about the relative values of the scores.



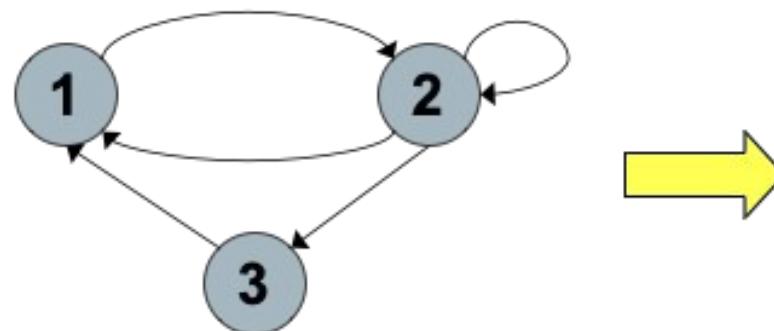
How many Iterations

- Claim: relative values of scores will converge after a few iterations:
 - in fact, suitably scaled, $h()$ and $a()$ scores settle into a steady state!
 - proof of this comes later.
- In practice, ~5 iterations get you close to stability.



Proof of Convergence

- $n \times n$ adjacency matrix \mathbf{A} :
 - each of the n pages in the base set has a row and column in the matrix.
 - Entry $A_{ij} = 1$ if page i links to page j , else = 0.



	1	2	3
1	0	1	0
2	1	1	1
3	1	0	0



Hub/Authority Vectors

- View the hub scores $h()$ and the authority scores $a()$ as vectors with n components.
- Recall the iterative updates

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$



In Matrix Form

- $h = Aa$
- $a = A^T h$

Covariance

- $h = AA^T h$
- h is the principal eigenvector of AA^T

Gram

- $a = A^T A a$
- a is the principal eigenvector of $A^T A$

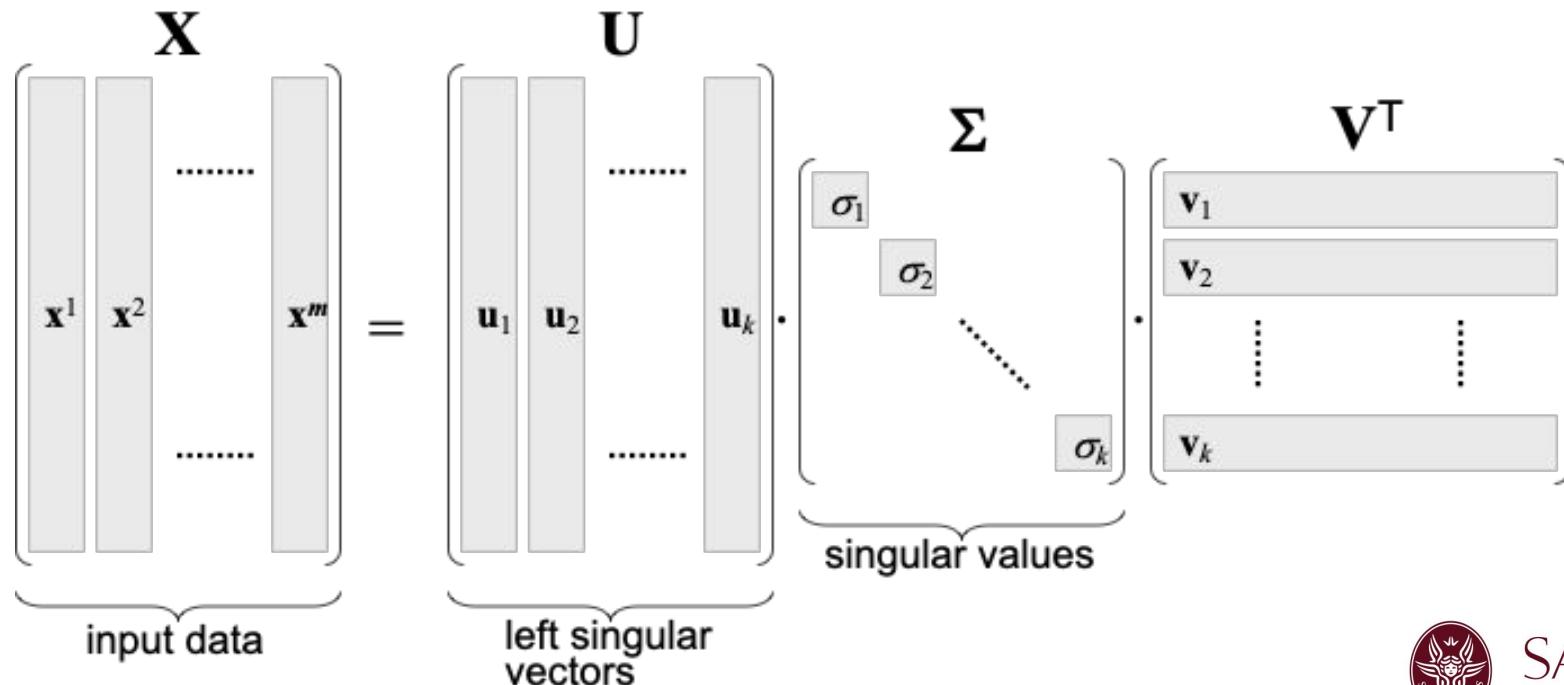
Can be computed using Power Method, in this case guaranteed to converge



SAPIENZA
UNIVERSITÀ DI ROMA

Singular Value Decomposition

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$



SVD and HITS

- V are the eigenvectors of the covariance matrix $A^T A$
- U are the eigenvectors of the Gram (inner-product) matrix $A A^T$
- Therefore, to compute Hub and Authorities score we could compute the SVD of the matrix A
 - SVD is not as scalable as the Power Method



Example of Authorities... found in 1999

- (java) Authorities
 - .328 <http://www.gamelan.com/> Gamelan
 - .251 <http://java.sun.com/> JavaSoft Home Page
 - .190 <http://www.digitalfocus.com/> ... The Java Developer: How Do I ...
 - .190 <http://lightyear.ncsa.uiuc.edu/~srp/java/javabooks.html>
 - .183 <http://sunsite.unc.edu/javafaq/javafaq.html> comp.lang.java FAQ
- (censorship) Authorities
 - .378 <http://www.eff.org/> EFFweb—The Electronic Frontier Foundation
 - .344 <http://www.eff.org/blueribbon.html> The Blue Ribbon Campaign for Online Free Speech
 - .238 <http://www.cdt.org/> The Center for Democracy and Technology
 - .235 <http://www.vtw.org/> Voters Telecommunications Watch
 - .218 <http://www.aclu.org/> ACLU: American Civil Liberties Union



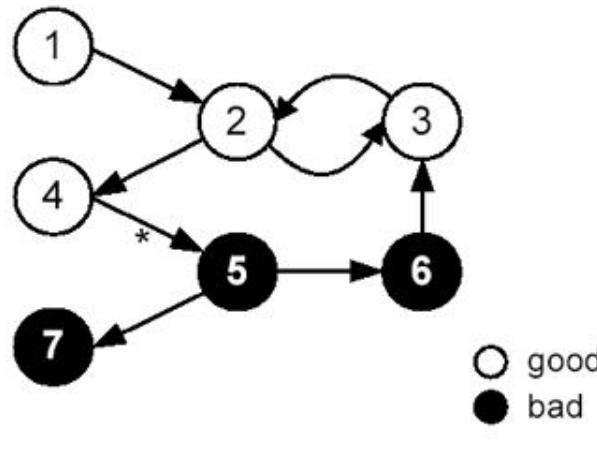
Issues

- Topic Drift
 - Off-topic pages can cause off-topic “authorities” to be returned
 - E.g., the neighborhood graph can be about a “super topic”
- Mutually Reinforcing Affiliates
 - Affiliated pages/sites can boost each others’ scores
 - Linkage between affiliated pages is not a useful signal



TrustRank

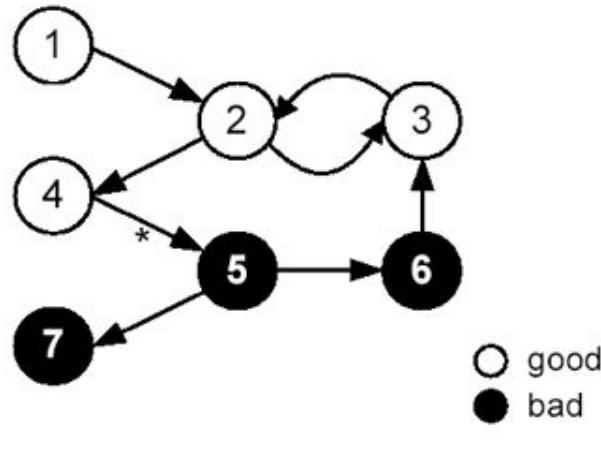
- Spamming in Web
 - Good and bad pages
- TrustRank is used to overcome Spamming
 - It proposes techniques to semi-automatically separate reputable, good pages from spam.
 - It first selects a small set of seed pages to be evaluated by an expert.
 - Once it manually identifies the reputable seed pages, then it uses the link structure of the web to discover other pages that are likely to be good



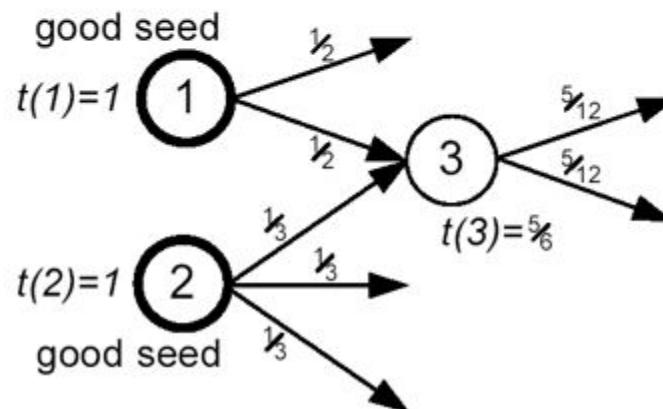
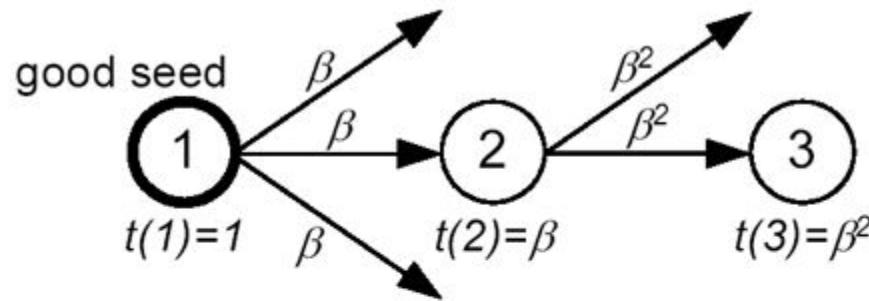
TrustRank

- It formalizes the notion of a human checking a page for spam by a binary oracle function O over all pages p :

$$O(p) = \begin{cases} 0 & \text{if } p \text{ is bad,} \\ 1 & \text{if } p \text{ is good.} \end{cases}$$



TrustRank: Trust Damping and Trust Splitting



Other Applications of Random Walk With Restart

- wikirank-2021.di.unimi.it

The Open Wikipedia Ranking²⁰²¹

Here you can browse Wikipedia pages by importance. Choose a category or write a complex query, watch the result and share your findings!

Learn more > Caveat emptor > ≈ 2020

Humans Bands Cities Paintings Women Men Pop singers Books Italians Food Countries Fashion designers Ideas Theorems

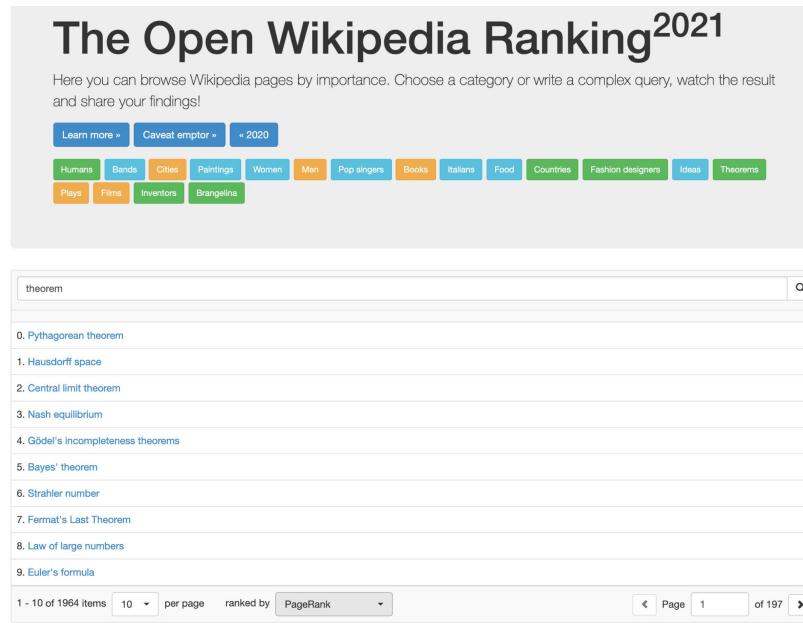
Plays Films Inventors Brangelina

theorem

0. Pythagorean theorem
1. Hausdorff space
2. Central limit theorem
3. Nash equilibrium
4. Gödel's incompleteness theorems
5. Bayes' theorem
6. Strahler number
7. Fermat's Last Theorem
8. Law of large numbers
9. Euler's formula

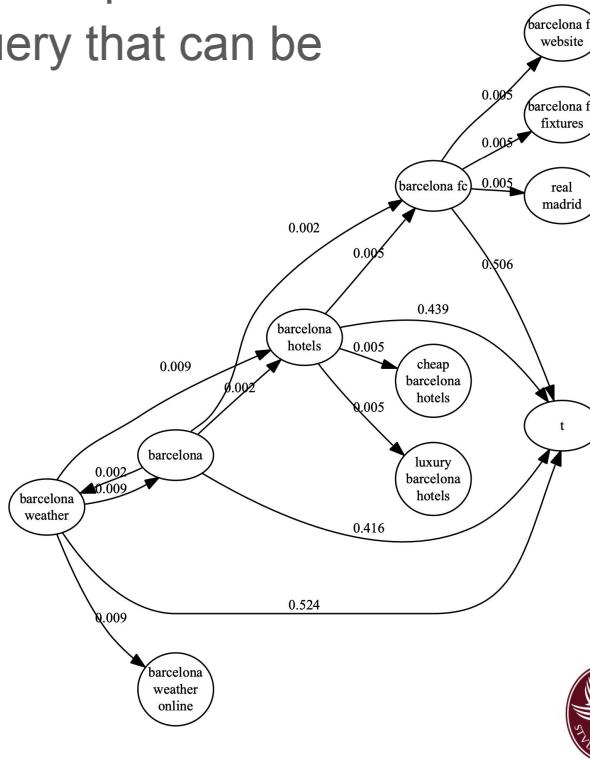
1 - 10 of 1964 items 10 per page ranked by PageRank

SHARE



Other Applications of Random Walk With Restart

- Query Suggestions and the Query-Flow Graph
- Given a query Q , find the most likely query that can be reached from Q .
 - RWR from Q



Other Applications of Random Walk With Restart

- Tourism Recommender Systems
 - A person visiting two Points of Interest (Pol) u, v induces an edge in a graph where vertices are Pols in a city
- RWR from a starting Pol can be used to recommend places to visit from that Pol.



SAPIENZA
UNIVERSITÀ DI ROMA

Other Applications of Random Walk With Restart

- Tourism Recommender Systems

	Florence	Glasgow	San Francisco
Number of PoIs	1,022	353	550
Images gathered from Flickr	124,223	176,981	937,389
Number of distinct albums (at least two photos)	2,919	1,971	4,411
Average distinct PoIs per album	3.71	4.97	3.61
Number of edges	131,238	25,486	39,372
Edges from Flickr	22,164	19,150	26,752
Edges from Wikipedia's Categories	111,778	8,644	16,038
Maximum (out)degree	415	103	263
Average (out)degree	121.86	72.20	71.59



Other Applications of Random Walk With Restart

- Tourism Recommender Systems

Starting PoIs in U	
Palazzo Vecchio	
Piazza della Signoria	

Top-10 ranked PoIs	
PoI	Probability
Ponte Vecchio	$5.9 \cdot e^{-4}$
Piazzale Michelangelo	$2.1 \cdot e^{-4}$
Palazzo Pitti	$1.9 \cdot e^{-4}$
Giotto's Campanile	$6.8 \cdot e^{-5}$
Boboli Gardens	$4.9 \cdot e^{-5}$
Loggia dei Lanzi	$4.6 \cdot e^{-5}$
Piazza Santa Croce	$4.2 \cdot e^{-5}$
Uffizi	$4.1 \cdot e^{-5}$
Basilica of Santa Croce	$3.9 \cdot e^{-5}$
Ponte alle Grazie	$3.4 \cdot e^{-5}$

Starting PoIs in U	
La Specola	
Museo Fiorentino di Preistoria	
Museo Horne	
Bargello	

Top-10 ranked PoIs	
PoI	Probability
Uffizi	$1.4 \cdot e^{-10}$
Giotto's Campanile	$1.2 \cdot e^{-10}$
Palazzo Medici Riccardi	$9.8 \cdot e^{-11}$
Vasari Corridor	$7.4 \cdot e^{-11}$
Medici Chapel	$6.5 \cdot e^{-11}$
Basilica of Santa Croce	$5.3 \cdot e^{-11}$
San Marco's National Museum	$1.3 \cdot e^{-11}$
Dante Alighieri's House	$9.6 \cdot e^{-12}$
Modern Art Gallery	$9.3 \cdot e^{-12}$
Museo Stibbert	$8.0 \cdot e^{-12}$



Other Applications of Random Walk With Restart

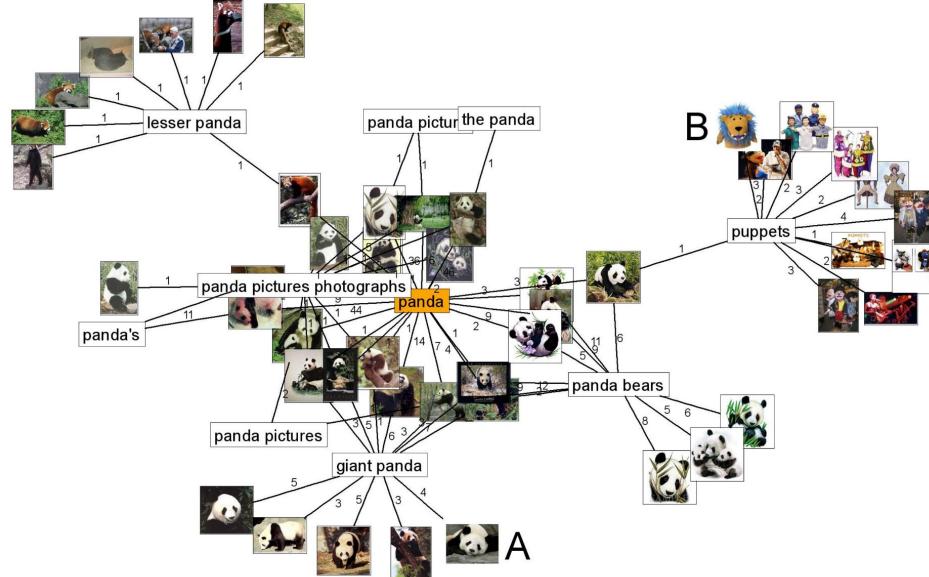
- Tourism Recommender Systems

Starting PoIs in U		Starting PoIs in U	
Top-10 ranked PoIs		Top-10 ranked PoIs	
PoI	Probability	PoI	Probability
Clyde Tunnel		Golden Gate Theatre	
Govan Subway Station		San Francisco Conservatory of Music	
Hillhead Subway Station			
Renfrew Airport			
Glasgow International Airport	$1.2 \cdot e^{-8}$	War Memorial Opera House	$1.1 \cdot e^{-5}$
Buchanan Street Subway Station	$4.2 \cdot e^{-9}$	Dolores Park	$1.0 \cdot e^{-5}$
Kelvinbridge	$6.8 \cdot e^{-10}$	Castro Theatre	$8.1 \cdot e^{-6}$
Glasgow Seaplane Terminal	$2.4 \cdot e^{-10}$	Yerba Buena Gardens	$7.8 \cdot e^{-6}$
St Enoch Subway Station	$2.0 \cdot e^{-10}$	Embarcadero Center	$7.3 \cdot e^{-6}$
Glasgow City Heliport	$2.0 \cdot e^{-10}$	Metreon	$6.3 \cdot e^{-6}$
Buchanan Bus Station	$9.5 \cdot e^{-11}$	Golden Gate Bridge	$5.5 \cdot e^{-6}$
Ibrox Subway Station	$9.5 \cdot e^{-11}$	Pacific-union Club	$4.2 \cdot e^{-6}$
Kelvinhall Subway Station	$8.3 \cdot e^{-11}$	Lake Merritt	$4.1 \cdot e^{-6}$
Cowcaddens Subway Station	$9.5 \cdot e^{-12}$	American Conservatory Theater	$3.9 \cdot e^{-6}$



Random Walking a Bipartite Graph

- Intuition: queries and clicked docs induce a bipartite graph that can be used to detect relevant documents for a given query based on co-clicks.



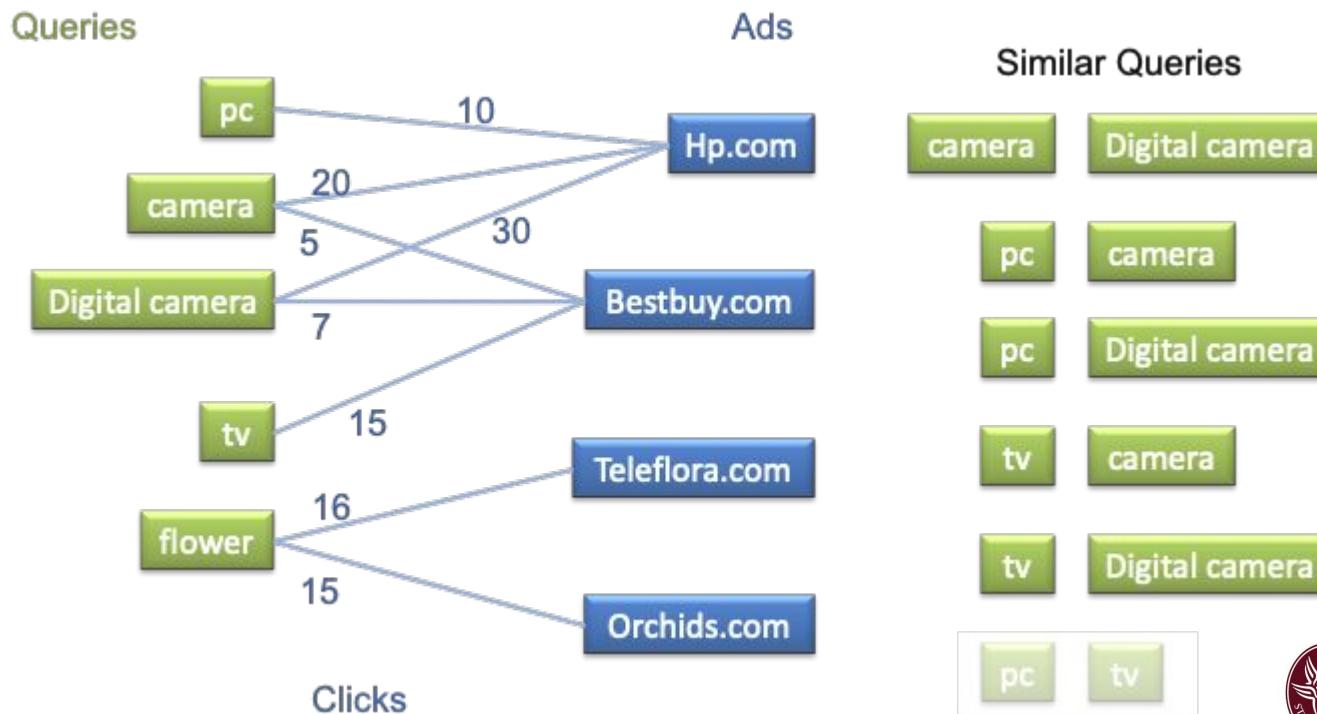
Random Walking a Bipartite Graph

- We can start a random walk from either documents or queries.
- We perform two steps at a time, this way the Markov Chain is irreducible and aperiodic.
- When we start from queries, we find similar queries on the basis of co-clicks
- When we start from documents, we find similar documents on the basis of “co-queries”.



SimRank

- Click Graph from Sponsored Search



SimRank

- Intuition:
 - “Two queries are similar if they are connected to similar ads”
 - “Two ads are similar if they are connected to similar queries”
- Iterative procedure
 - at each iteration similarity propagates through the graph



SimRank

- $N(q)$: # of ads connected to q
- $E(q)$: set of ads connected to q
- $\text{sim}_k(q, q')$: q - q' similarity at k -th iteration
- Initially $\text{sim}(q, q) = 1$, $\text{sim}(q, q') = 0$, $\text{sim}(a, a) = 1$, $\text{sim}(a, a') = 0$

$$s_k(q, q') = \frac{C}{N(q)N(q')} \sum_{i \in E(q)} \sum_{j \in E(q')} s_{k-1}(i, j)$$

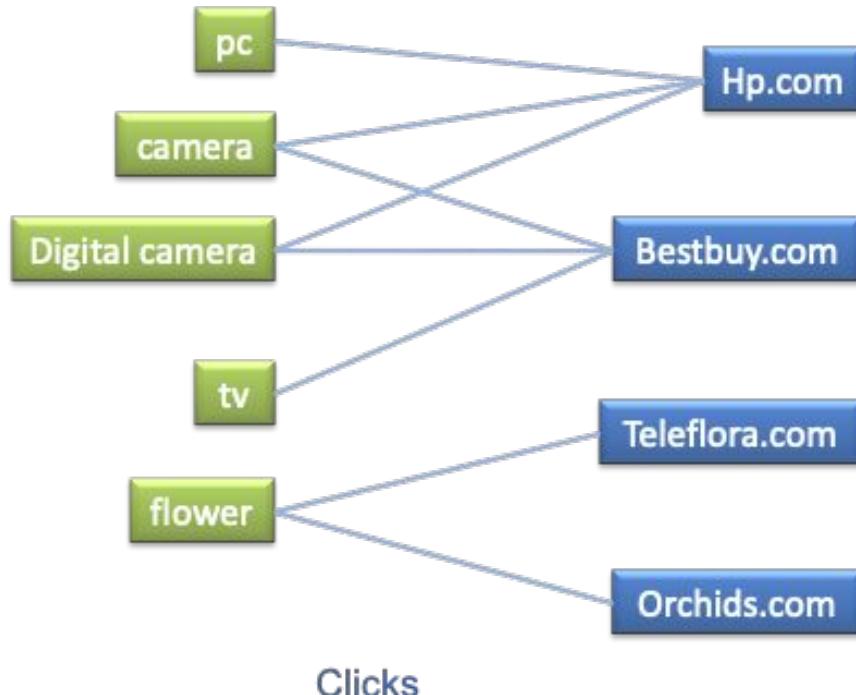
$$s_k(a, a') = \frac{C}{N(a)N(a')} \sum_{i \in E(a)} \sum_{j \in E(a')} s_{k-1}(i, j)$$

- Time: $O(n^4)$



SimRank

Queries



$$s_k(q, q') = \frac{C}{N(q)N(q')} \sum_{i \in E(q)} \sum_{j \in E(q')} s_{k-1}(i, j)$$

$$s_k(a, a') = \frac{C}{N(a)N(a')} \sum_{i \in E(a)} \sum_{j \in E(a')} s_{k-1}(i, j)$$

Two random surfer model



SAPIENZA
UNIVERSITÀ DI ROMA

Introduction to
Information Retrieval

CS276
Information Retrieval and Web Search
Chris Manning and Pandu Nayak
Personalization

Ambiguity

- Unlikely that a short query can unambiguously describe a user's information need

- For example, the query [chi] can mean
 - Calamos Convertible Opportunities & Income Fund quote
 - The city of Chicago
 - Balancing one's natural energy (or ch'i)
 - Computer-human interactions

Personalization

- Ambiguity means that a single ranking is unlikely to be optimal for all users
- Personalized ranking is the only way to bridge the gap
- Personalization can use
 - Long term behavior to identify user interests,
e.g., a long term interest in user interface research
 - Short term session to identify current task,
e.g., checking on a series of stock tickers
 - User location, e.g., MTA in New York vs Baltimore
 - Social network
 - ...

Potential for Personalization

[Teevan, Dumais, Horvitz 2010]

- How much can personalization improve ranking? How can we measure this?
- Ask raters to explicitly rate a set of queries
 - But rather than asking them to guess what a user's information need might be ...
 - ... ask which results *they would personally consider relevant*
 - Use self-generated and pre-generated queries

Computing potential for personalization

- For each query q
 - Compute average rating for each result
 - Let R_q be the optimal ranking according to the average rating
 - Compute the NDCG value of ranking R_q for the ratings of each rater i
 - Let Avg_q be the average of the NDCG values for each rater
- Let Avg be the average Avg_q over all queries
- Potential for personalization is $(1 - \text{Avg})$

Example: NDCG values for a query

Result	Rater A	Rater B	Average rating
D1	1	0	0.5
D2	1	1	1
D3	0	1	0.5
D4	0	0	0
D5	0	0	0
D6	1	0	0.5
D7	1	2	1.5
D8	0	0	0
D9	0	0	0
D10	0	0	0
NDCG	0.88	0.65	

Average NDCG for raters: 0.77

Example: NDCG values for optimal ranking for average ratings

Result	Rater A	Rater B	Average rating
D7	1	2	1.5
D2	1	1	1
D1	1	0	0.5
D3	0	1	0.5
D6	1	0	0.5
D4	0	0	0
D5	0	0	0
D8	0	0	0
D9	0	0	0
D10	0	0	0
NDCG	0.98	0.96	

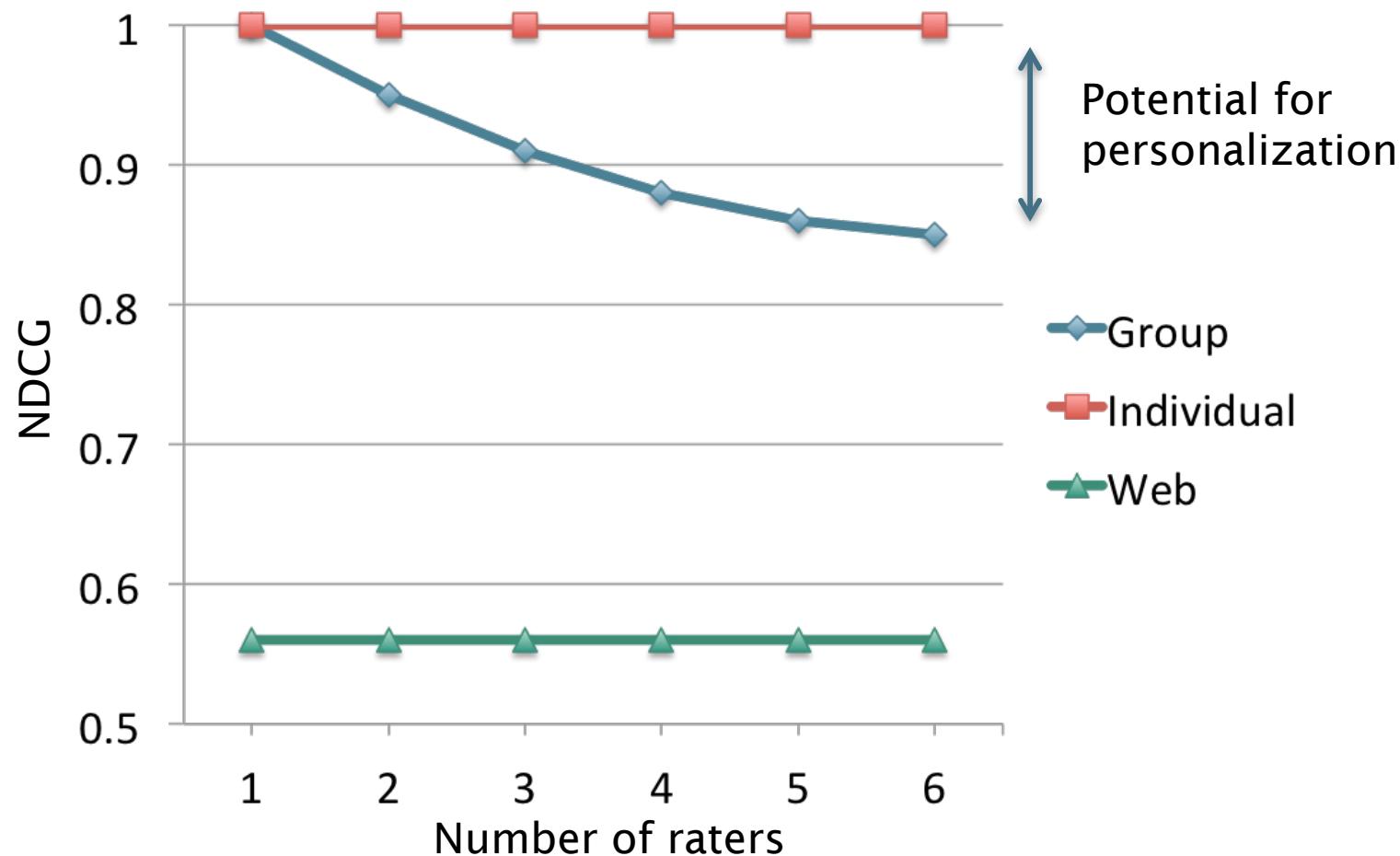
Average NDCG for raters: 0.97

Example: Potential for personalization

Result	Rater A	Rater B	Average rating
D7	1	2	1.5
D2	1	1	1
D1	1	0	0.5
D3	0	1	0.5
D6	1	0	0.5
D4	0	0	0
D5	0	0	0
D8	0	0	0
D9	0	0	0
D10	0	0	0
NDCG	0.98	0.96	

Potential for personalization: 0.03

Potential for personalization graph



PERSONALIZING SEARCH

Personalizing search

[Pitkow et al. 2002]

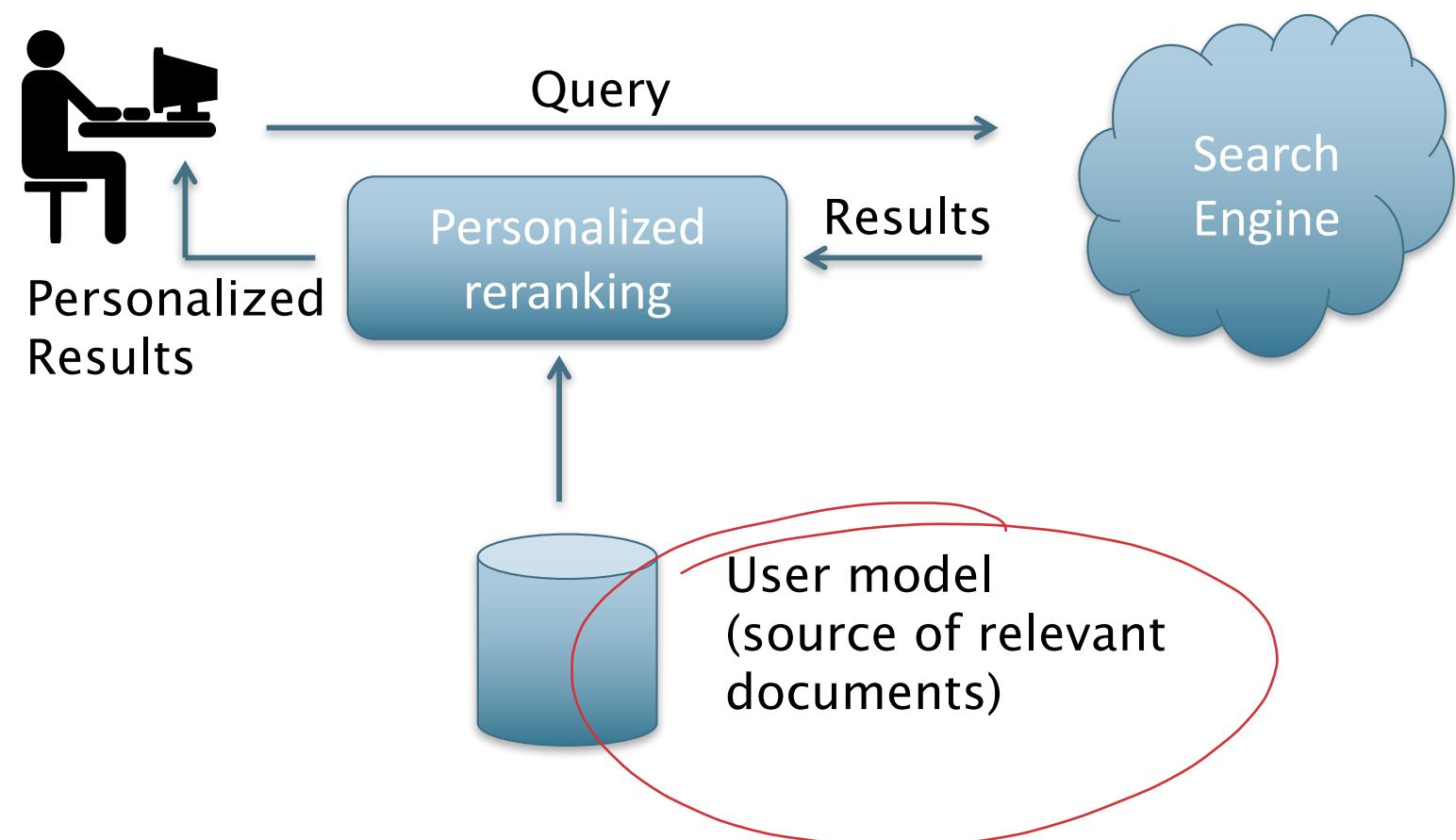
- Two general ways of personalizing search
 - **Query expansion**
 - Modify or augment user query
 - E.g., query term “IR” can be augmented with either “information retrieval” or “Ingersoll-Rand” depending on user interest
 - Ensures that there are enough personalized results
 - **Reranking**
 - Issue the same query and fetch the same results ...
 - ... but rerank the results based on a user profile
 - Allows both personalized and globally relevant results

User interests

- Explicitly provided by the user
 - Sometimes useful, particularly for new users
 - ... but generally doesn't work well
- Inferred from user behavior and content
 - Previously issued search queries
 - Previously visited Web pages
 - Personal documents
 - Emails
- Ensuring privacy and user control is very important

Relevance feedback perspective

[Teevan, Dumais, Horvitz 2005]



Binary Independence Model

- Estimating RSV coefficients in theory
- For each term i look at this table of document counts:

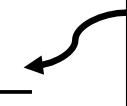
$$c_i = \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

Documents	Relevant	Non-Relevant	Total
$x_i=1$	s_i	n_i-s_i	n_i
$x_i=0$	$S-s_i$	$N-n_i-S+s_i$	$N-n_i$
Total	S	$N-S$	N

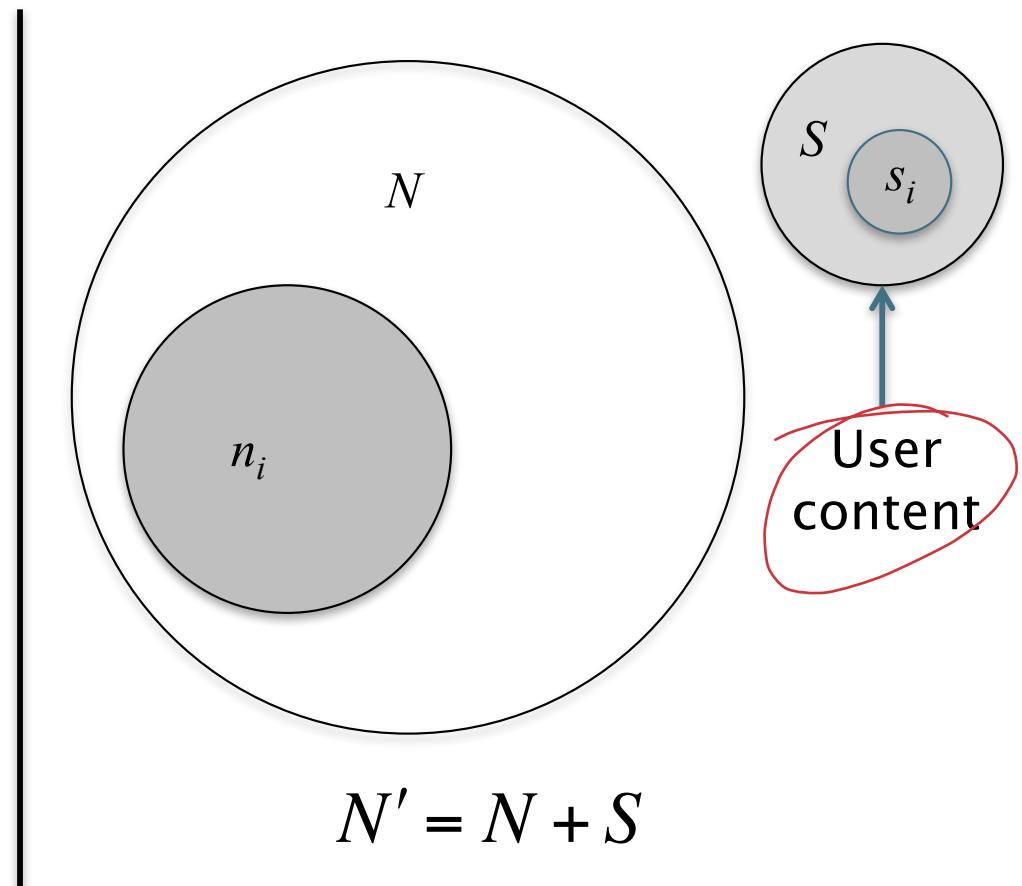
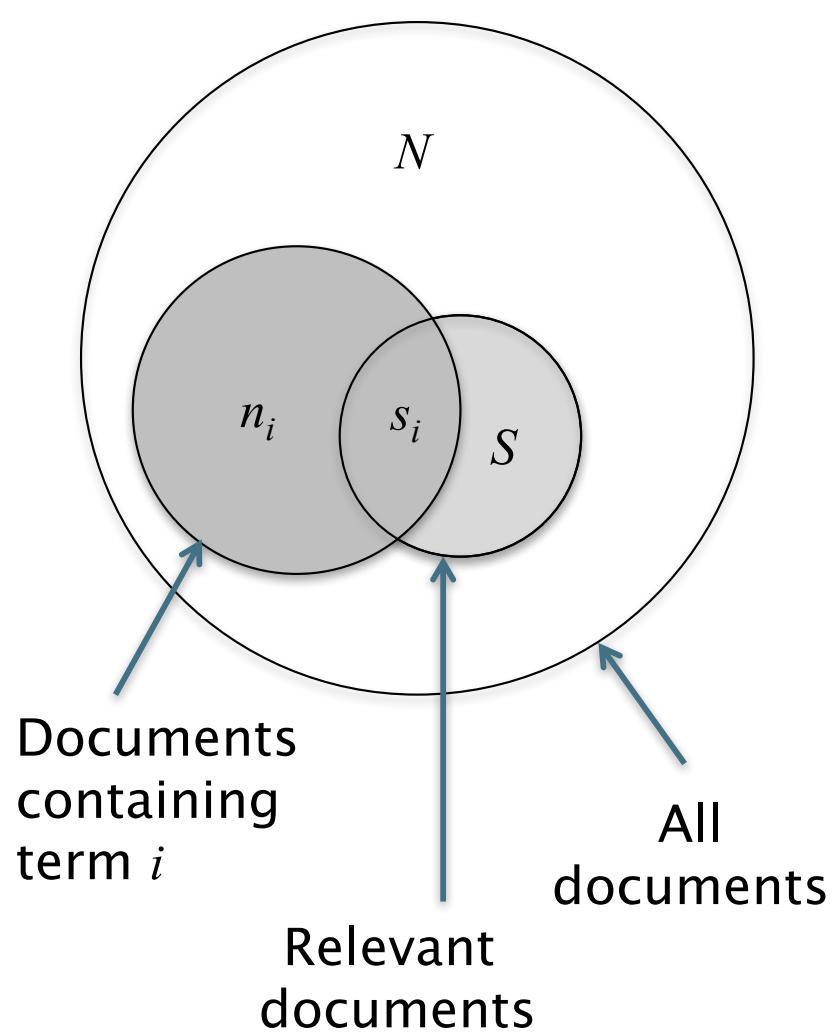
- Estimates: $p_i \approx \frac{s_i}{S}$ $r_i \approx \frac{(n_i - s_i)}{(N - S)}$

$$c_i \approx K(N, n_i, S, s_i) = \log \frac{s_i/(S - s_i)}{(n_i - s_i)/(N - n_i - S + s_i)}$$

For now,
assume no
zero terms.
See later
lecture.



Personalization as relevance feedback



$$N' = N + S$$

$$n'_i = n_i + s_i$$

Reranking

- BM25 scoring

$$\sum c_i \times tf_i$$

- Use updated weight c_i in BM25

$$c_i = \log \frac{(s_i + 0.5)}{(S - s_i + 0.5)} \frac{(N - n_i + 0.5)}{(n_i + 0.5)} \approx \log \frac{(s_i + 0.5)}{(S - s_i + 0.5)} + IDF_i$$

where we have used

$$N' = N + S$$

$$n'_i = n_i + s_i$$

Corpus representation

- Estimating N and n_i
- Many possibilities
 - N : All documents, query relevant documents, result set
 - n_i : Full text, only titles and snippets
- Practical strategy
 - Approximate corpus statistics from result set
 - ... and just the title and snippets
 - Empirically seems to work the best!

User representation

- Estimating S and s_i
- Estimated from a local search index containing
 - Web pages the user has viewed
 - Email messages that were viewed or sent
 - Calendar items
 - Documents stored on the client machine
- Best performance when
 - S is the number of local documents matching the query
 - s_i is the number that also contains term i

Document and query representation

- Document represented by the title and snippets
- Query is expanded to contain words near query terms (in titles and snippets)
 - For the query [cancer] add underlined terms

The American **Cancer** Society is dedicated to eliminating **cancer** as a major health problem by preventing **cancer**, saving lives, and diminishing suffering through ...

- This combination of corpus, user, document, and query representations seem to work well

LOCATION

User location

- User location is one of the most important features for personalization
 - Country
 - Query [football] in the US vs the UK
 - State/Metro/City
 - Queries like [zoo], [craigslist], [giants]
 - Fine-grained location
 - Queries like [pizza], [restaurants], [coffee shops]

Challenges

- Not all queries are location sensitive
 - [facebook] is not asking for the closest Facebook office
 - [seaworld] is not necessarily asking for the closest SeaWorld
- Different parts of a site may be more or less location sensitive
 - NYTimes home page vs NYTimes Local section
- Addresses on a page don't always tell us how location sensitive the page is
 - Stanford home page has address, but not location sensitive

Key idea

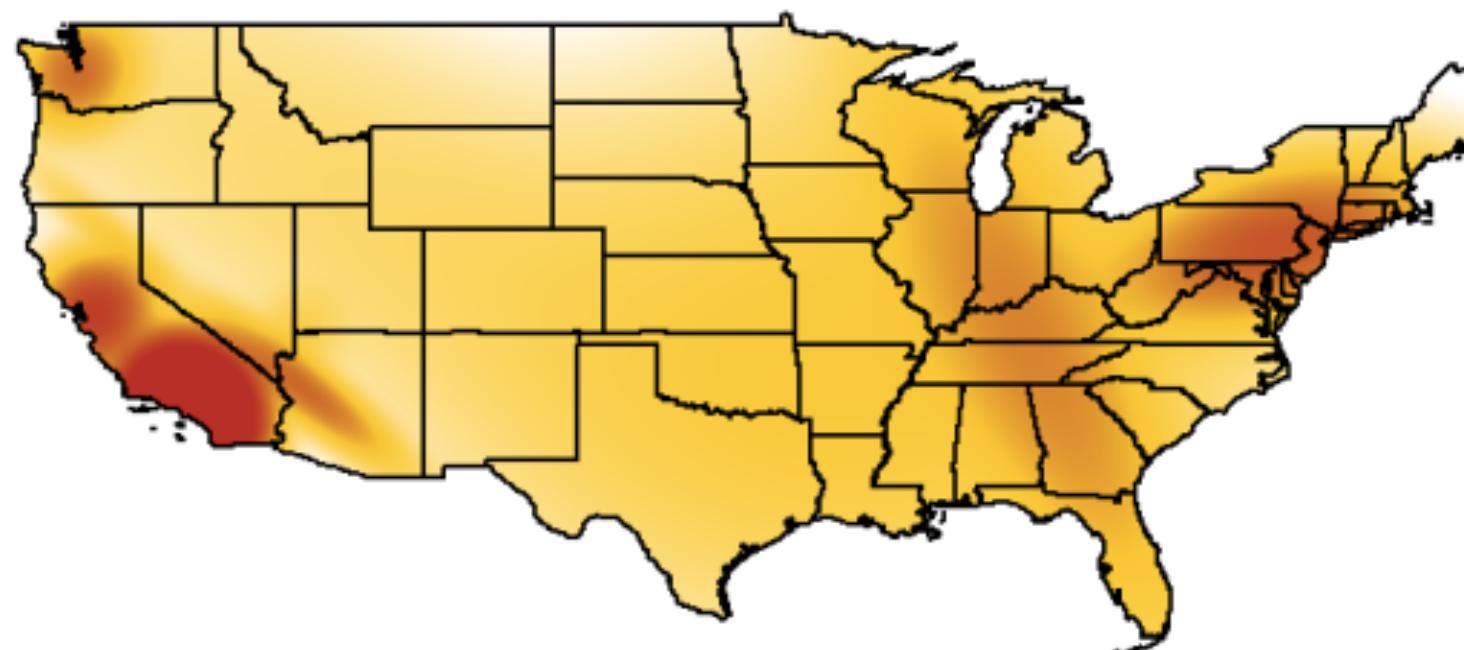
[Bennett et al. 2011]

- *Usage statistics*, rather than locations mentioned in a document, best represent where it is relevant
 - I.e., if users in a location tend to click on that document, then it is relevant in that location
- User location data is acquired from anonymized logs (with user consent, e.g., from a widely distributed browser extension)
 - User IP addresses are resolved into geographic location information

Location interest model

- Use the logs data to estimate the probability of the location of the user given they viewed this URL

$$P(\text{location} = x \mid \text{URL})$$



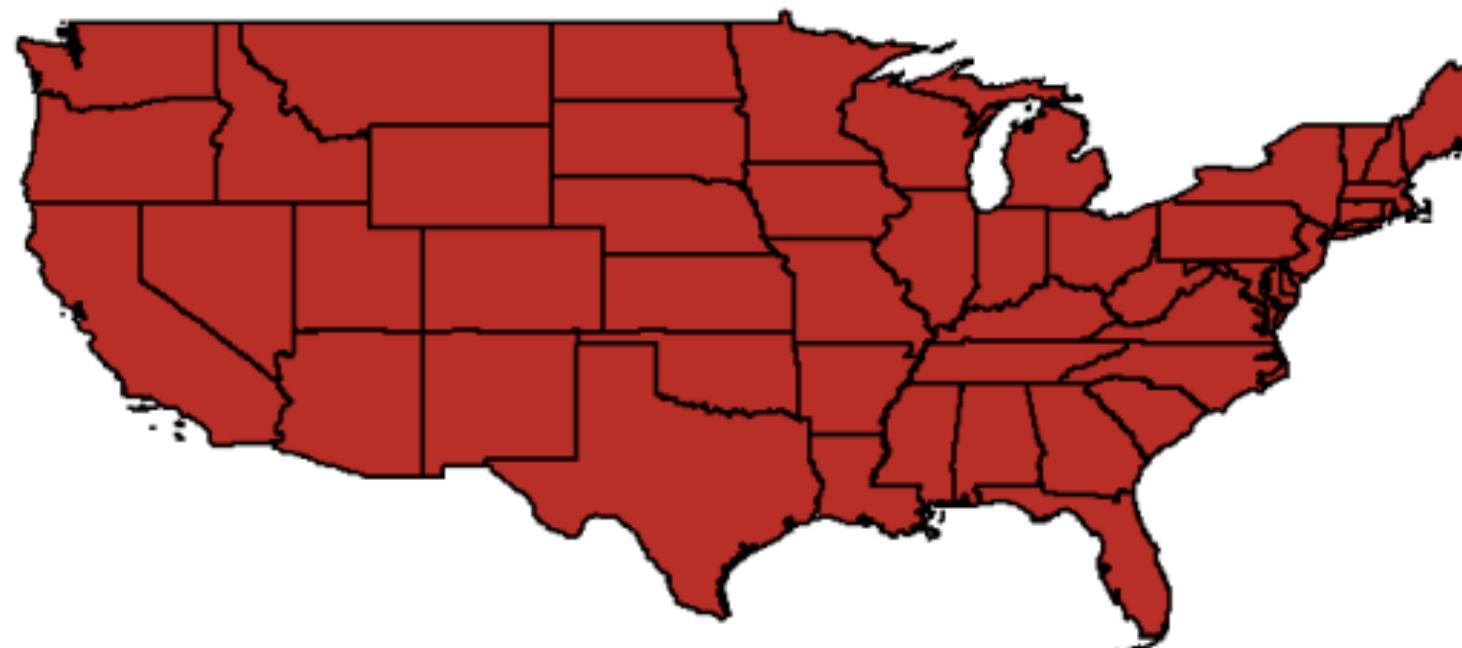
(c) Los Angeles Times: Reviews and Recommendations

<http://findlocal.latimes.com/>

Location interest model

- Use the logs data to estimate the probability of the location of the user given they viewed this URL

$$P(\text{location} = x \mid \text{URL})$$



(d) Los Angeles Times: Crossword Puzzles and Games
<http://games.latimes.com/>

Learning the location interest model

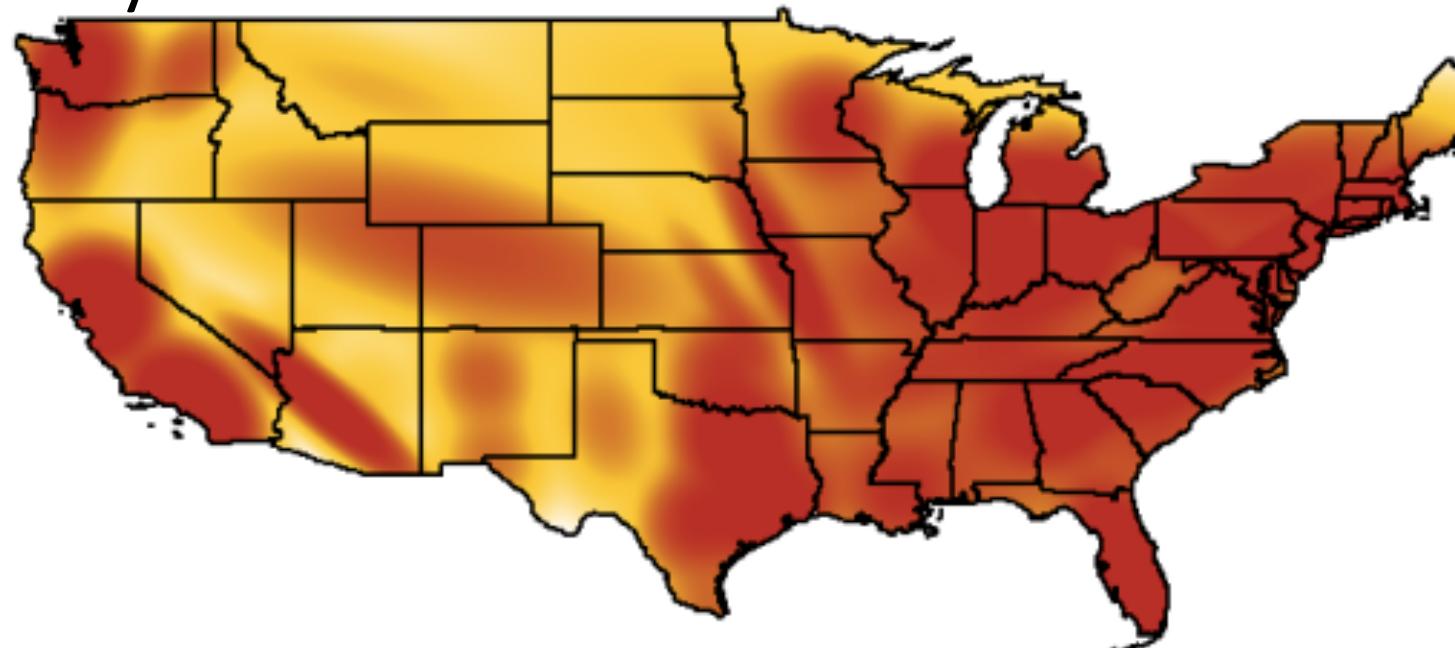
- For compactness, represent location interest model as a mixture of 5-25 2-d Gaussians (x is [lat, long])

$$\begin{aligned} P(\text{location} = x \mid URL) &= \sum_{i=1}^n w_i N(x; \mu_i, \Sigma_i) \\ &= \sum_{i=1}^n \frac{w_i}{(2\pi)^2 |\Sigma_i|^{1/2}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)} \end{aligned}$$

- Learn Gaussian mixture model using EM
 - Expectation step: Estimate probability that each point belongs to each Gaussian
 - Maximization step: Estimate most likely mean, covariance, weight

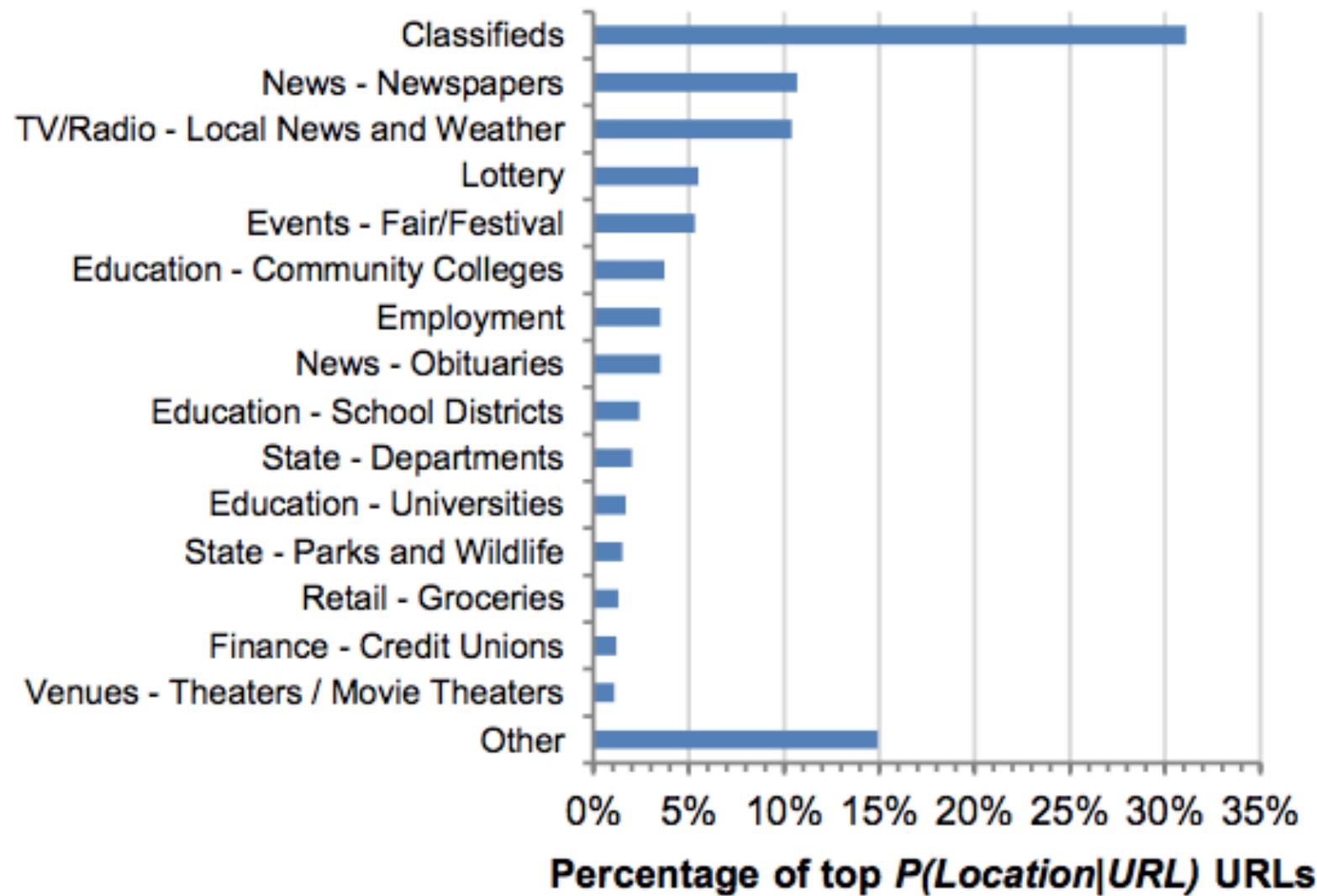
More location interest models

- Learn a location-interest model for queries
 - Using location of users who issued the query
- Learn a background model showing the overall density of users



(e) Background Model

Topics in URLs with high $P(\text{user location} \mid \text{URL})$



Location sensitive features

- Non-contextual features (user-independent)
 - Is the query location sensitive? What about the URLs?
 - Feature: Entropy of the location distribution
 - Low entropy means distribution is peaked and location is important
 - Feature: KL-divergence between location model and background model
 - High KL-divergence suggests that it is location sensitive
 - Feature: KL-divergence between query and URL models
 - Low KL-divergence suggests URL is more likely to be relevant to users issuing the query

More location sensitive features

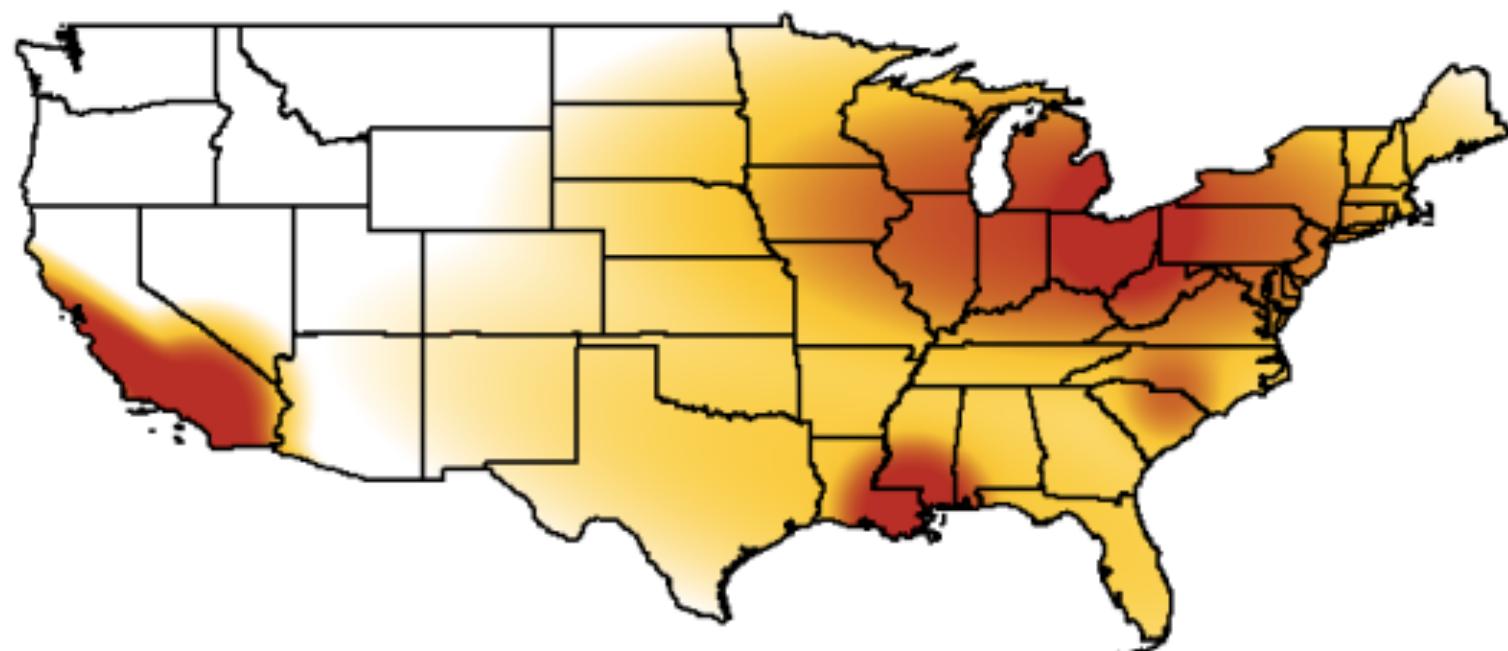
- Contextual features (user-dependent)
 - Feature: User's location (naturally!)
 - Feature: Probability of the user's location given the URL
 - Computed by evaluating URL's location model at user location
 - Feature is high when user is at a location where URL is popular
 - Downside: large population centers tend to higher probabilities for all URLs
 - Feature: Use Bayes rule to compute $P(\text{URL} \mid \text{user location})$
 - Feature: Also create a normalized version of the above feature by normalizing with the background model
 - Features: Versions of the above with query instead of URL

Learning to rank

- Add location features (in addition to standard features) for machine learned ranking
 - Training data derived from logs
 - $P(\text{URL} \mid \text{user location})$ turns out to be an important feature
 - KL divergence of the URL model from the background model also plays an important role

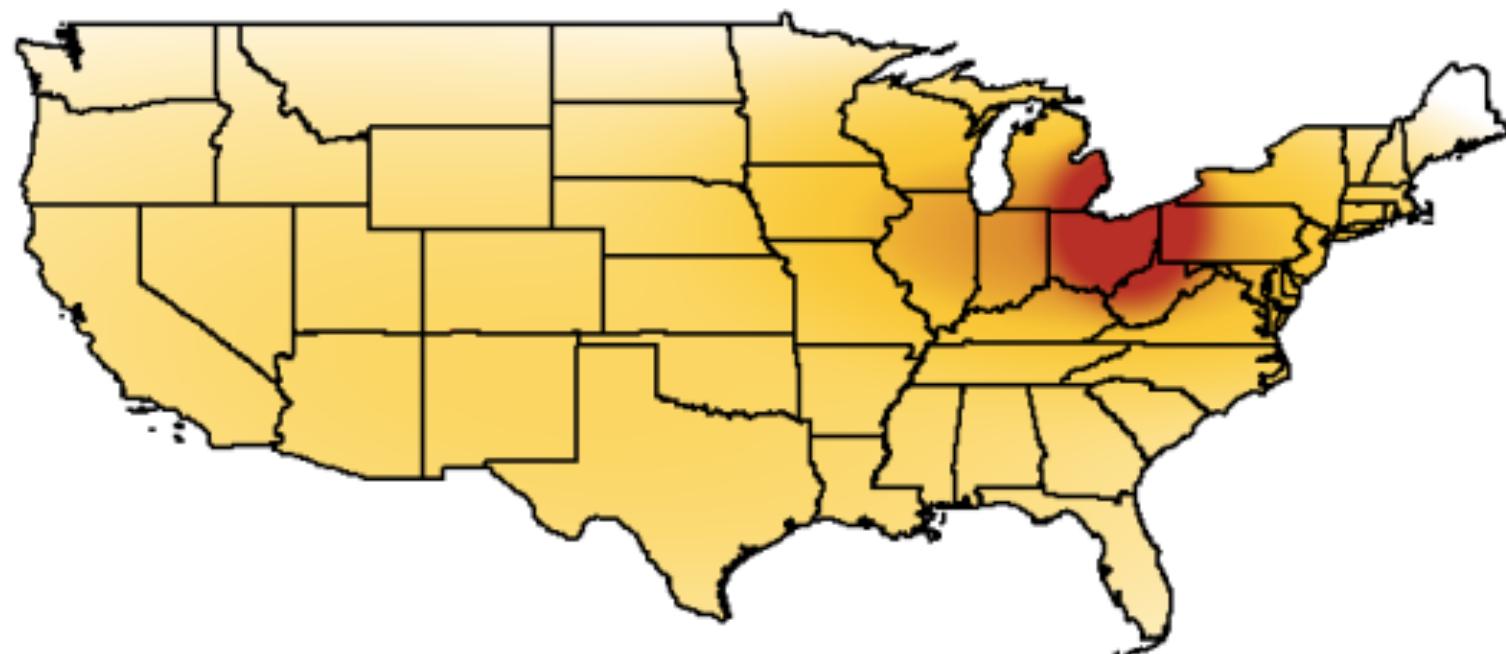
Query model for [rtb bus schedule]

User in New Orleans



URL model for top original result

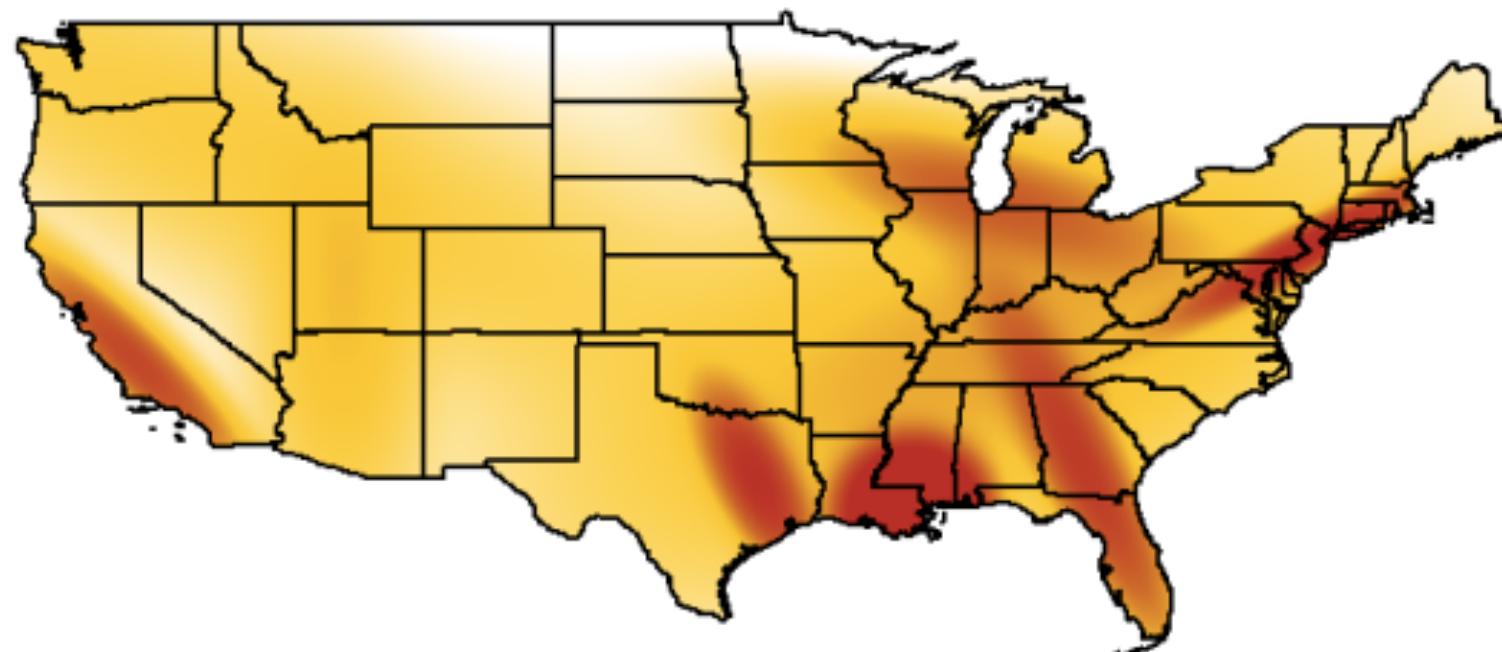
User in New Orleans



(a) <http://www.riderta.com/maps-schedules.asp>

URL model for promoted URL

User in New Orleans



(b) <http://www.norta.com/>

PERSONALIZED PAGERANK

Pagerank review

- Let \mathbf{A} be the stochastic matrix corresponding to the Web graph G over n nodes
 - No teleportation links (but assume no deadends in G)
 - If node i has o_i outlinks, and there is an edge from node i to node j , then $\mathbf{A}_{ij} = 1/o_i$
- Let \mathbf{p} be the teleportation probabilities
 - $(n \times 1)$ column vector with each entry being $1/n$
- Pagerank vector \mathbf{r} is defined by the following

$$\mathbf{r} = (1 - \alpha)\mathbf{Ar} + \alpha\mathbf{p}$$

Personalized pagerank

[Haveliwala 2003] [Jeh and Widom 2003]

- In the basic pagerank computation, teleportation probability vector \mathbf{p} is uniform over all pages
- But if the user has preferences on which pages to teleport to, that preference can be represented in \mathbf{p}
 - \mathbf{p} could be uniform over user's bookmarks
 - Or it could be non-zero on just pages on topics of interest to the user
- Pagerank would be personalized to user's interests
- But computing personalized pagerank is expensive

Linearity theorem

- For any preference vectors \mathbf{u}_1 and \mathbf{u}_2 , if \mathbf{v}_1 and \mathbf{v}_2 are the corresponding personalized pagerank vectors, then for any non-negative constants a_1 and a_2 such that $a_1 + a_2 = 1$, we have

$$a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 = (1 - \alpha) \mathbf{A} (a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2) + \alpha (a_1 \mathbf{u}_1 + a_2 \mathbf{u}_2)$$

- Proof

$$\begin{aligned} a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 &= a_1 ((1 - \alpha) \mathbf{A} \mathbf{v}_1 + \alpha \mathbf{u}_1) + a_2 ((1 - \alpha) \mathbf{A} \mathbf{v}_2 + \alpha \mathbf{u}_2) \\ &= a_1 (1 - \alpha) \mathbf{A} \mathbf{v}_1 + a_1 \alpha \mathbf{u}_1 + a_2 (1 - \alpha) \mathbf{A} \mathbf{v}_2 + a_2 \alpha \mathbf{u}_2 \\ &= (1 - \alpha) \mathbf{A} (a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2) + \alpha (a_1 \mathbf{u}_1 + a_2 \mathbf{u}_2) \end{aligned}$$

Topic-sensitive pagerank

- Compute personalized pagerank vector per topic
 - 16 top-level topics from the Open Directory Project
 - Each ODP topic has a set of pages (hand-)classified into that topic
 - Preference vector for the topic is uniform over pages in that topic, and 0 elsewhere
- Note: [Jeh and Widom 2003] provide a more general treatment

Query-time processing

- Construct a distribution over topics for the query
 - User profile can provide a distribution over topics
 - Query can be classified into the different topics
 - Any other context information can be used to inform topic distributions
- Use the topic preferences to compute a weighted linear combination of topic pagerank vectors to use in place of pagerank

SOCIAL NETWORKS

Unicorn

[Curtiss et al 2013]

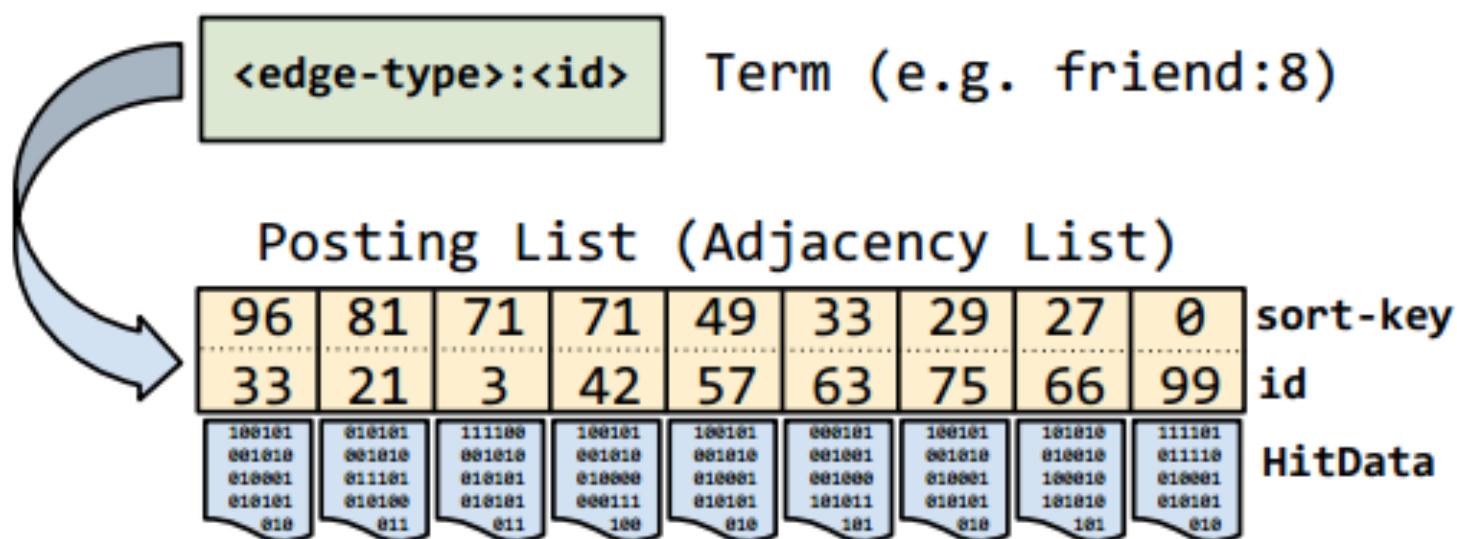
- Primary backend for Facebook Graph Search

- Facebook social graph

- Nodes represent people and things (entities)
- Each entity has a unique 64-bit **id**
- Edges represent relationships between nodes
- There are many thousands of edge-types
 - Examples: friend, likes, likers, ...

Data model

- Billions of nodes, but graph is sparse
 - Represent graph using adjacency list
 - Postings sorted by sort-key (importance) and then id
 - Index sharded by result-id



Basic set operations

- Query language includes basic set operations
 - and, or, difference
- Friends of either Jon Jones (id 5) and Lea Lin (id 6)
`(or(friend:5 friend:6))`
- Female friends of Jon Jones who are not friend of Lea Lin
`(difference (and friend:5 gender:1) friend:6)`

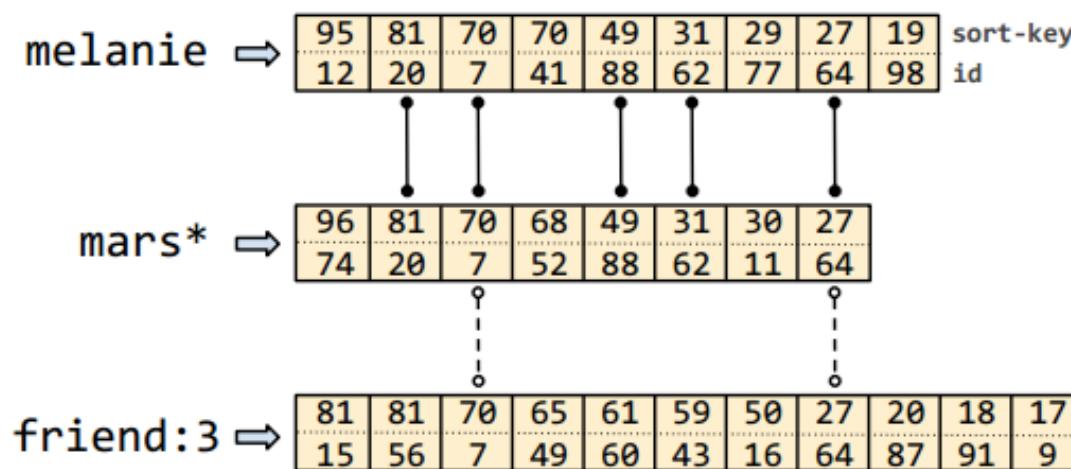
Typeahead

- Find users by typing first few characters of their name
- Index servers contain postings lists for every name prefix up to a predefined character limit
 - Simple typeahead implementation would simply return ids in the corresponding postings lists
- Simple solution doesn't ensure social relevance
- Alternate solution: Use a conjunctive query
`(and mel* friend:3)`
 - Misses people who are not friends
 - Issuing two queries is expensive

WeakAnd operator

- Provides a mechanism for some fraction of results to possess a trait without requiring trait for all results
- WeakAnd** allows missing terms from some results
 - These optional terms can have an optional count or weight
 - Once the optional count is met, the term is required

(weak-and (term friend:3 :optional-hits 2) (term melanie) (term mars*))



Graph Search

- Graph Search results are often more than one edge away from source nodes
 - Example: Pages liked by friends of Melanie who like Emacs
- Unicorn provides additional operators to support Graph Search
 - **Apply**
`(apply likes: (and friend:7 likers:42))`
 - **Extract**
 - Extract and return (denormalized) ids stored in HitData

References

- J. Teevan, S. Dumais, E. Horvitz. Potential for personalization. 2010
- J. Pitkow et al. Personalized search. 2002
- J. Teevan, S. Dumais, E. Horvitz. Personalizing search via automated analysis of interests and activities. 2005
- P. Bennett et al. Inferring and using location metadata to personalize Web search. 2011
- T. Haveliwala. Topic-sensitive pagerank. 2002.
- G. Jeh and J. Widom. Scaling personalized Web search. 2003
- M. Curtiss et al. Unicorn: A system for searching the social graph. 2013

Web Information Retrieval

Crash notes on Personalized Pagerank

Iterative computation of Personalized Pagerank

Assume you want to compute personalized pagerank with respect to some personalization vector \mathbf{p} . For example, \mathbf{p} might be:

$$\mathbf{p}_i = \begin{cases} \frac{1}{|S|}, & i \in S, \\ 0 & \text{otherwise} \end{cases}$$

where S denotes a seed set. Note that \mathbf{p} might in general be any probability distribution over the n pages in of the Web graph, not necessarily uniform over a given seed set.

For any personalization vector \mathbf{p} , we know that for a generic page i we have:

$$\pi_i(t) = (1 - \alpha)\mathbf{p}_i + \alpha \sum_{j \rightarrow i} \frac{\pi_j(t-1)}{d_j},$$

where $1 - \alpha$ is the teleportation probability and d_j denotes j 's out-degree (we assume all sinks were previously removed by linking each of them to each node in S).

A compact and equivalent way of writing this equation is:

$$\pi(t)^T = \pi(t-1)^T ((1 - \alpha)\mathbf{1}\mathbf{p}^T + \alpha M),$$

where M is the transition matrix of the underlying graph (after removal of sinks). Of course, this is exactly the original, iterative formulation of pagerank, with the only difference that we changed the vector of teleportation probabilities from being uniform over the entire set of Web pages to being defined by some distribution \mathbf{p} . As a consequence, the very same version of the power method using pagerank leaking can be applied, thus exploiting the sparseness of M . Note that, in general, the following happens: i) the personalized pagerank exists, since the Markov chain we obtain is still irreducible and aperiodic; ii) the stationary distribution will in general differ from the original pagerank. You should elaborate on why this is the case.

Composability of Personalized Pagerank

Next, let

$$P = (1 - \alpha)\mathbf{1}\mathbf{p}^T + \alpha M$$

Of course we have $\pi(t)^T = \mathbf{p}^T P^t$, given that $\pi(0) = \mathbf{p}$.

Building personalization vectors for single users

Assume we have personalization vectors $\mathbf{q}^1, \dots, \mathbf{q}^{(k)}$ for some number k of "broad" topics. These might for example be "sports", "news", "movies" etc. Note that each $\mathbf{q}^{(r)}$ is a probability distribution over the entire set pages of Web pages, possibly placing 0 mass on some (or even the majority) of them. So, each such vector has n entries, with n the number of Web pages. For the sake of exposition, we call *broad personalization vectors* the $\mathbf{q}^{(r)}$'s and we call *broad personalized pagerank vectors* the k corresponding pagerank vectors that correspond to them in the remainder of these notes. Assume now that some user's interests are succinctly described by a distribution \mathbf{s}

over the set of k topics. The ℓ -th entry of vector \mathbf{s} can for instance be estimated from the fraction of times the user clicked on pages that are relevant for the ℓ -th topic when returned result sets corresponding to her queries. It should be emphasized that \mathbf{s} can be computed in many ways, more or less refined, depending on users' profile information available to the search engine. Next, let

$$Q = (\mathbf{q}^1 \dots \mathbf{q}^{(k)})$$

I.e., Q is an $n \times k$ matrix whose columns are the broad personalization vectors. A personalization vector \mathbf{p} for the user under consideration can then be obtained as $\mathbf{p} = Q\mathbf{s} = \sum_{r=1}^k \mathbf{s}_r \mathbf{q}^{(r)}$. Note that \mathbf{p} is still a distribution over the set of the n Web pages (*you are warmly invited to check this yourself*). On the other hand, if we consider the pagerank vector $\pi(t)$ after t steps with personalization vector \mathbf{p} we obtain:

$$\pi(t)^T = \mathbf{p}^T P^t = \left(\sum_{r=1}^k \mathbf{s}_r \mathbf{q}^{(r)} \right)^T P^t = \sum_{r=1}^k \mathbf{s}_r (\mathbf{q}^{(r)})^T P^t = \sum_{r=1}^k \mathbf{s}_r (\pi^{(r)}(t))^T,$$

where $\pi^{(r)}$ denotes the personalized pagerank corresponding to personalization vector $\mathbf{q}^{(r)}$ after t steps, while the last equality follows immediately from the linearity of matrix transposition. Since this equality holds for every value of t , it also holds for the limiting, stationary distribution, i.e., the personalized pagerank corresponding to personalization vector \mathbf{p} (the argument is *almost* formal and this will be enough for these notes).

Meaning and advantages

In practice, the derivation above implies that we do not need to explicitly compute personalized pagerank vectors (using the power method) for every user, whenever personalization vectors reflect users' interests with respect to a set of k given "broad" topics. Rather, it is enough to compute, once and for all, k personalized pagerank vectors, one for each of the aforementioned topics. At this point, if the personalization vector of a user is expressed as a linear combination of the k broad personalization vectors, his/her personalized pagerank is a linear combination of the corresponding broad personalized pagerank vectors, according to the same coefficients. This makes it possible to compute personalized pagerank vectors for different user profiles (e.g., reflecting different user categories) on the fly, without having to run the power method every time.

Introduction to **Information Retrieval**

CS276

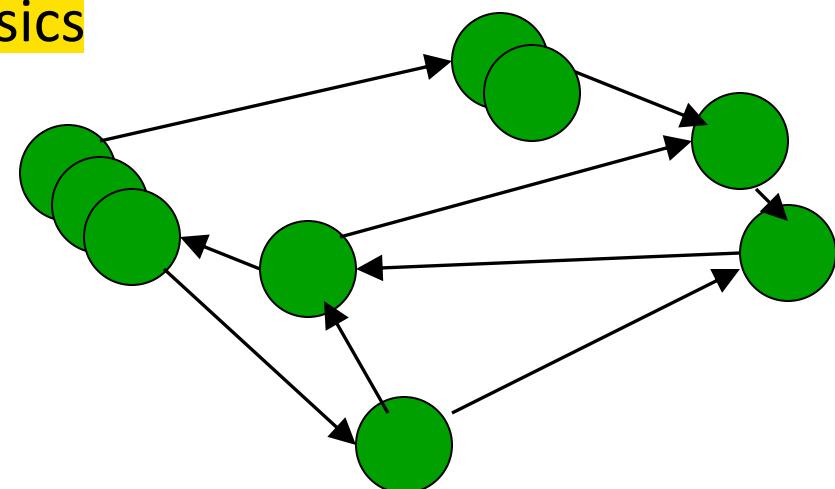
Information Retrieval and Web Search

Chris Manning and Pandu Nayak

Link analysis

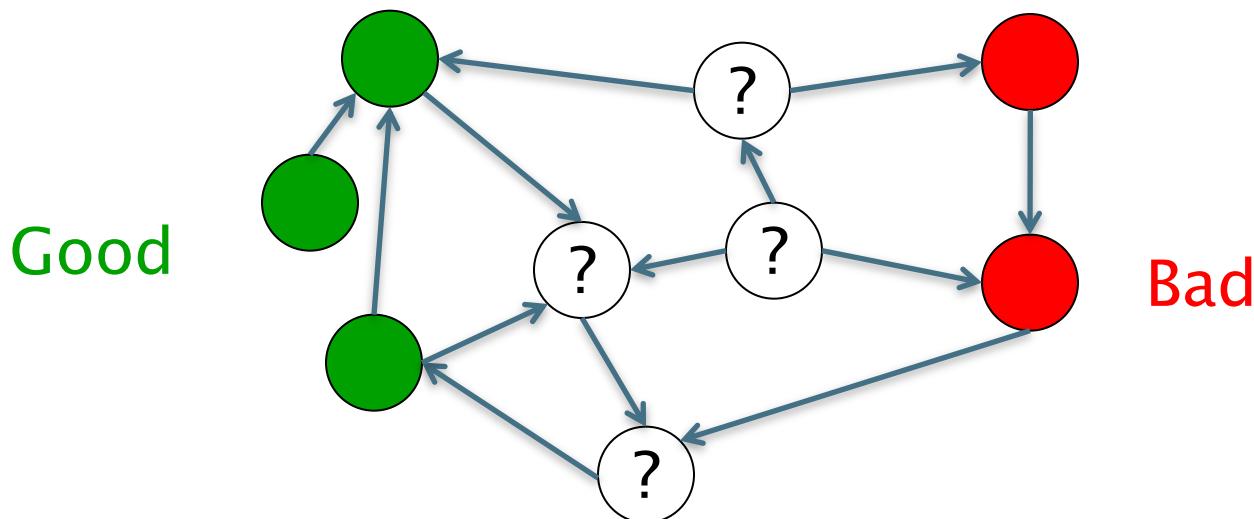
Today's lecture – hypertext and links

- We look beyond the *content* of documents
 - We begin to look at the hyperlinks between them
- Address questions like
 - Do the links represent a conferral of authority to some pages? Is this useful for ranking?
 - How likely is it that a page pointed to by the CERN home page is about high energy physics
- Big application areas
 - The Web
 - Email
 - Social networks



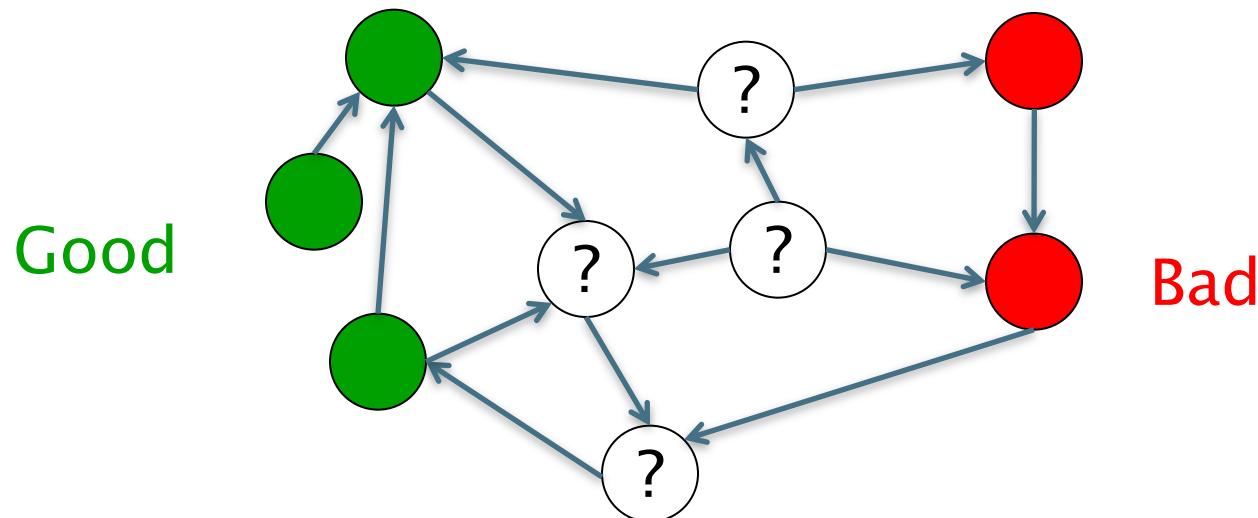
Links are everywhere

- Powerful sources of authenticity and authority
 - Mail spam – which email accounts are spammers?
 - Host quality – which hosts are “bad”?
 - Phone call logs
- The **Good**, The **Bad** and The Unknown



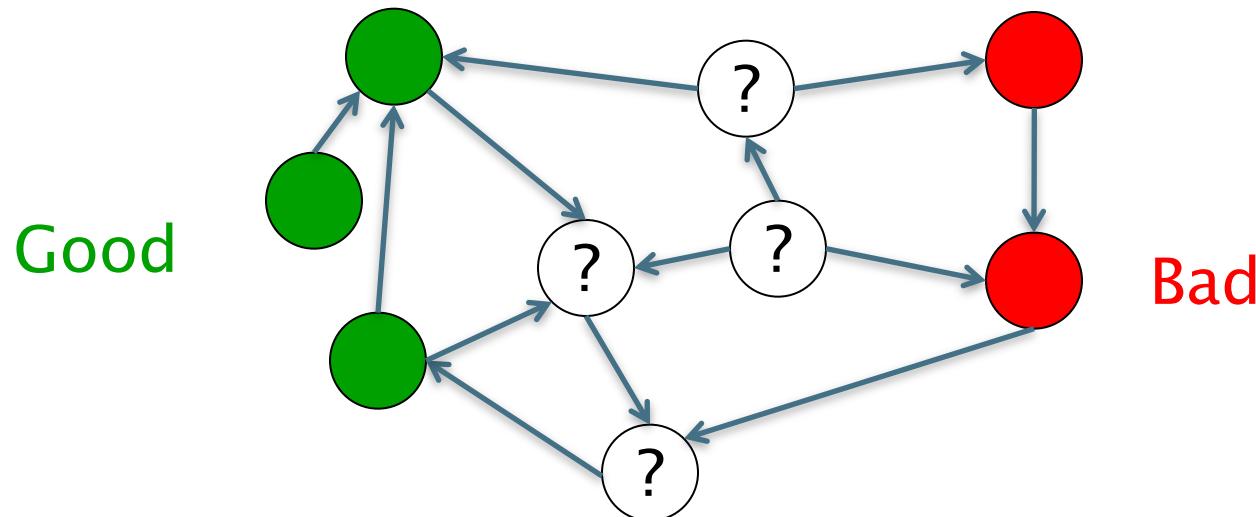
Example 1: Good/Bad/Unknown

- The **Good**, The **Bad** and The **Unknown**
 - **Good** nodes won't point to **Bad** nodes
 - All other combinations plausible



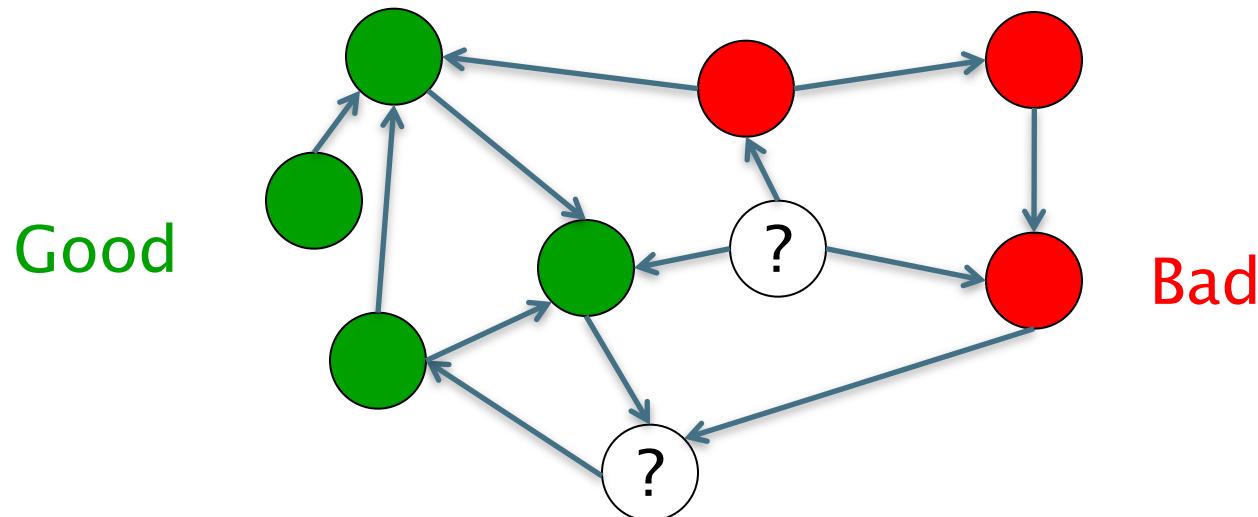
Simple iterative logic

- **Good** nodes won't point to **Bad** nodes
 - If you point to a **Bad** node, you're **Bad**
 - If a **Good** node points to you, you're **Good**



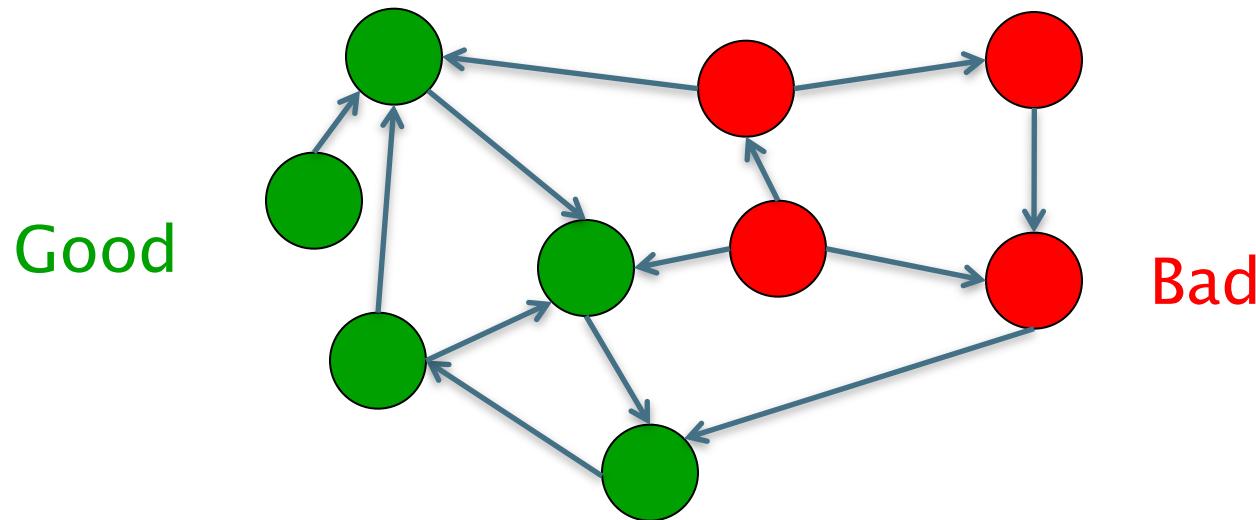
Simple iterative logic

- **Good** nodes won't point to **Bad** nodes
 - If you point to a **Bad** node, you're **Bad**
 - If a **Good** node points to you, you're **Good**



Simple iterative logic

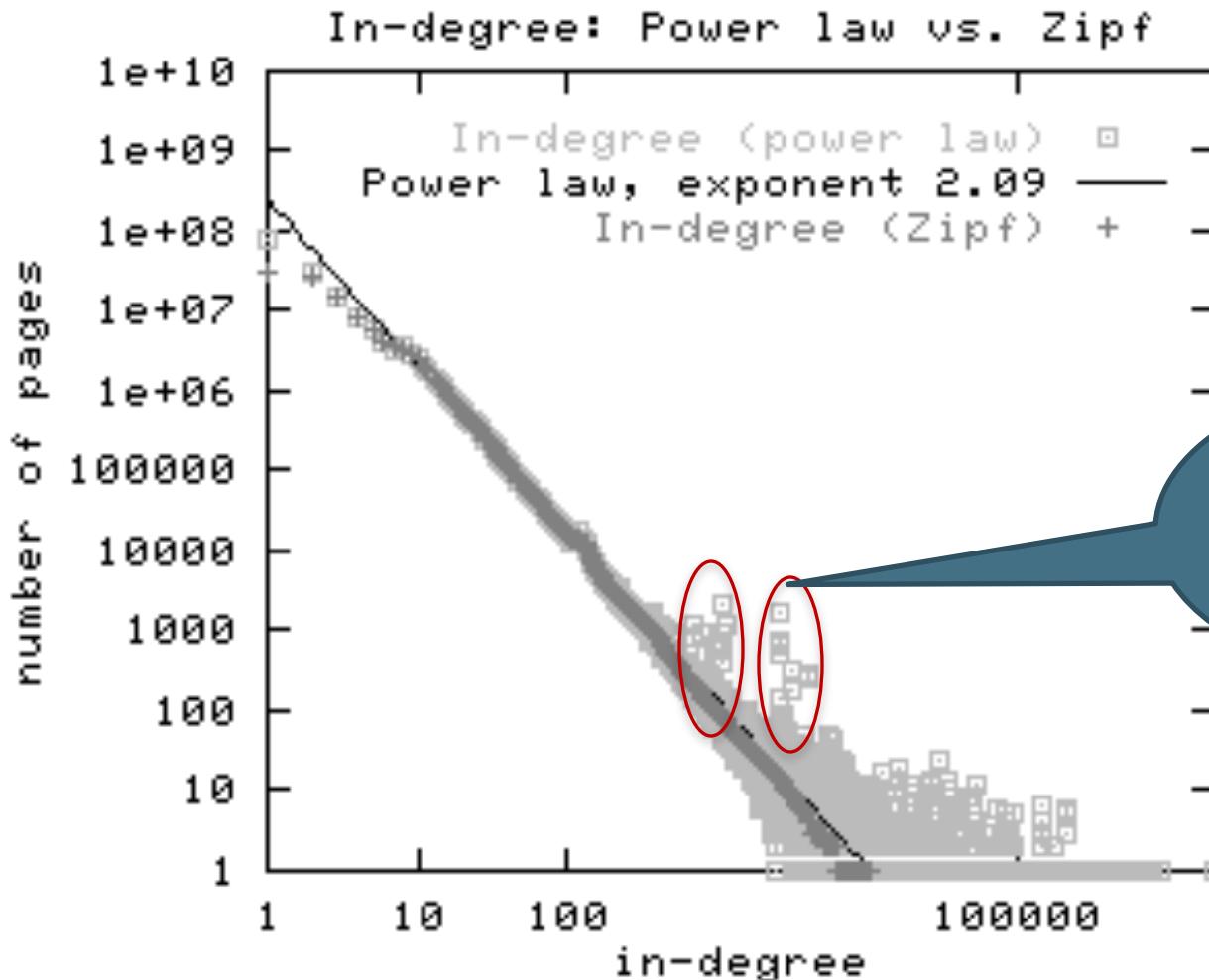
- **Good** nodes won't point to **Bad** nodes
 - If you point to a **Bad** node, you're **Bad**
 - If a **Good** node points to you, you're **Good**



Sometimes need probabilistic analogs – e.g., mail spam

Example 2:

In-links to pages – unusual patterns 😊



Spammers
violating
power laws!

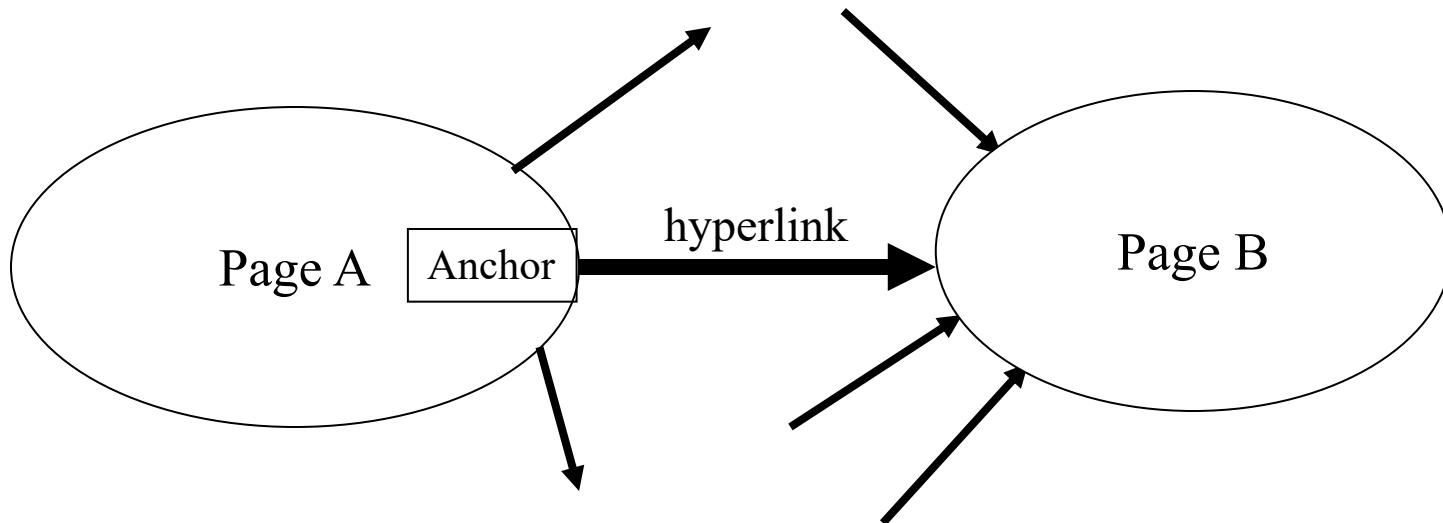
Many other examples of link analysis

- Social networks are a rich source of grouping behavior
- E.g., Shoppers' affinity – Goel+Goldstein 2010
 - Consumers whose friends spend a lot, spend a lot themselves
- <http://www.cs.cornell.edu/home/kleinber/networks-book/>
- See cs224w

Our primary interest in this course

- Link analysis additions to IR functionality thus far based purely on text
 - Scoring and ranking
 - Link-based clustering – topical structure from links
 - Links as features in classification – documents that link to one another are likely to be on the same subject
- Crawling
 - Based on the links seen, where do we crawl next?

The Web as a Directed Graph

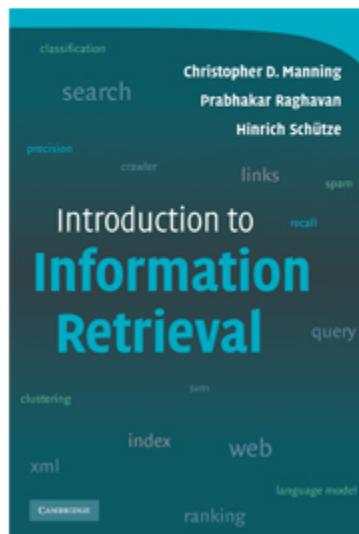


Hypothesis 1: A hyperlink between pages denotes
a conferral of authority (quality signal)

Hypothesis 2: The text in the anchor of a hyperlink on page
A describes the target page B

Assumption 1: reputed sites

Introduction to Information Retrieval



This is the companion website for the following book.

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*

You can order this book at [CUP](#) at your local bookstore or on the internet. The best search

The book aims to provide a modern approach to information retrieval from a computer science perspective. It is available at the [University of Stuttgart](#).

We'd be pleased to get feedback about how this book works out as a textbook, what is missing, and other comments to: [informationretrieval\(at\)yahoogroups\(dot\)com](mailto:informationretrieval(at)yahoogroups(dot)com)

Assumption 2: annotation of target

The image illustrates the evolution of a university's internationalization strategy through its website's language interface. It shows two versions of the Tohoku University homepage side-by-side.

Original Version (Top): Features the university's logo and name in Japanese (東北大学 TOHOKU UNIVERSITY). The language menu includes "中文" (Chinese), "한국어" (Korean), "English" (which is circled in red), and "日本語" (Japanese). Below the menu are five navigation tabs: "大学概要" (About the University), "学部・大学院・研究所" (Faculties, Schools, and Institutes), "教育・学生支援" (Education and Student Support), "国際交流" (International Exchange), and "研究・産学連携" (Research and Industry Cooperation).

Modified Version (Bottom): Shows the same layout but with changes in the language menu. The "English" link has been removed, replaced by "Chinese" and "Korean". The "Search" bar is also present here. Below the menu, there are seven navigation tabs: "About Tohoku University", "Faculties, Schools and Institutes", "Campus Life", "International Exchange", "Research and Cooperation", "Disclosure and Public Information", and "Entrance Exam Information".

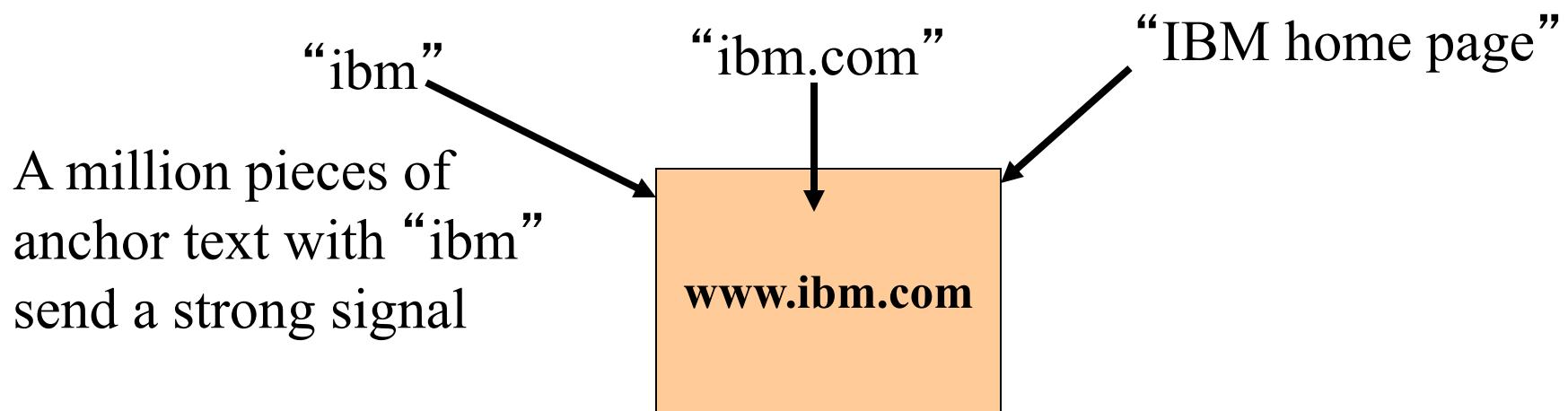
Left Sidebar: A vertical sidebar on the left contains links for different user groups: Prospective Students, General Public, Corporations, Alumni, Current Students, and Faculty and Staff (Internal use).

Image and Text: A photograph of students at an orientation ceremony is displayed with the caption "東北大学入学式(平成23年5月)". To the right, a promotional banner for a new video channel features the university's logo and a call-to-action button labeled "Click Here".

Anchor Text

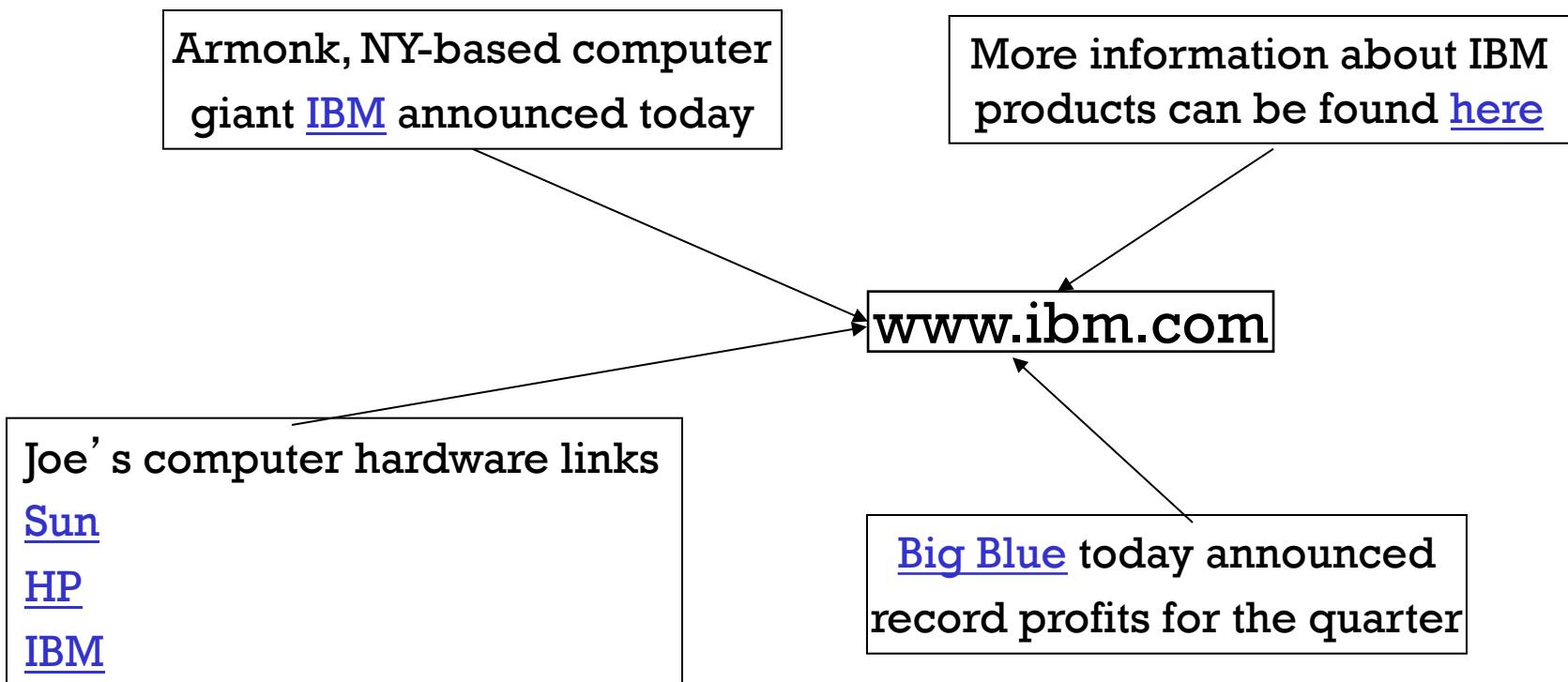
WWW Worm - McBryan [Mcbr94]

- For ***ibm*** how to distinguish between:
 - IBM's home page (mostly graphical)
 - IBM's copyright page (high term freq. for 'ibm')
 - Rival's spam page (arbitrarily high term freq.)



Indexing anchor text

- When indexing a document D , include (with some weight) anchor text (and perhaps nearby surrounding text) from links pointing to D .



Indexing anchor text

- Can sometimes have unexpected effects, e.g., spam, **miserable failure**
- **Can score anchor text with weight depending on the authority of the anchor page's website**
 - E.g., if we were to assume that content from cnn.com or yahoo.com is authoritative, then trust (more) the anchor text from them
 - **Increase the weight of off-site anchors (non-nepotistic scoring)**

Connectivity servers

Getting at all that link information
inexpensively

Connectivity Server

- Support for fast queries on the web graph
 - Which URLs point to a given URL?
 - Which URLs does a given URL point to?

Stores mappings in memory from

- URL to outlinks, URL to inlinks

- Applications
 - Link analysis
 - Web graph analysis
 - Connectivity, crawl optimization
 - Crawl control

Boldi and Vigna 2004

- <http://www2004.org/proceedings/docs/1p595.pdf>
- Webgraph – set of algorithms and a java implementation
- Fundamental goal – maintain node adjacency lists in memory
 - For this, compressing the adjacency lists is the critical component

Adjacency lists

- The set of neighbors of a node
- Assume each URL represented by an integer
- E.g., for a 4 billion page web, need 32 bits per node ... and now there are definitely $> 4B$ pages
- Naively, this demands 64 bits to represent each hyperlink
- Boldi/Vigna get down to an average of ~ 3 bits/link
 - Further work achieves 2 bits/link

Adjacency list compression

- Properties exploited in compression:
 - Similarity (between lists)
 - Locality (many links from a page go to “nearby” pages)
 - Use gap encoding in sorted lists
 - Distribution of gap values

Main ideas of Boldi/Vigna

- Consider lexicographically ordered list of all URLs,
e.g.,
 - www.stanford.edu/alchemy
 - www.stanford.edu/biology
 - www.stanford.edu/biology/plant
 - www.stanford.edu/biology/plant/copyright
 - www.stanford.edu/biology/plant/people
 - www.stanford.edu/chemistry

Boldi/Vigna

- Each of these URLs has an adjacency list
- Main idea: due to templates, the adjacency list of a node is similar to one of the 7 preceding URLs in the lexicographic ordering ... or else encoded anew
- Express adjacency list in terms of one of these
- E.g., consider these adjacency lists
 - 1, 2, 4, 8, 16, 32, 64
 - 1, 4, 9, 16, 25, 36, 49, 64
 - 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
 - 1, 4, 8, 16, 25, 36, 49, 64

Encode as (-2), remove 9, add 8

Gap encodings

- Given a sorted list of integers x, y, z, \dots , represent by $x, y-x, z-y, \dots$
- Compress each integer using a code
 - γ code - Number of bits = $1 + 2 \lfloor \lg x \rfloor$
 - δ code: ...
 - Information theoretic bound: $1 + \lfloor \lg x \rfloor$ bits
 - ζ code: Works well for integers from a power law [Boldi, Vigna: Data Compression Conf. 2004]

Main advantages of BV

- Depends only on locality in a canonical ordering
 - Lexicographic ordering works well for the web
- Adjacency queries can be answered very efficiently
 - To fetch out-neighbors, trace back the chain of prototypes
 - This chain is typically short in practice (since similarity is mostly intra-host)
 - Can also explicitly limit the length of the chain during encoding
- Easy to implement one-pass algorithm

Link analysis: Pagerank

Citation Analysis

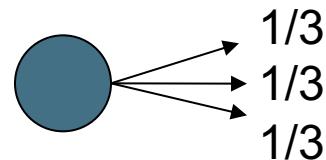
- Citation frequency
- Bibliographic coupling frequency
 - Articles that co-cite the same articles are related
- Citation indexing
 - Who is this author cited by? (Garfield 1972)
- Pagerank preview: Pinski and Narin '60s
 - Asked: which journals are authoritative?

The web isn't scholarly citation

- Millions of participants, each with self interests
- Spamming is widespread
- Once search engines began to use links for ranking (roughly 1998), link spam grew
 - You can join a *link farm* – a group of websites that heavily link to one another

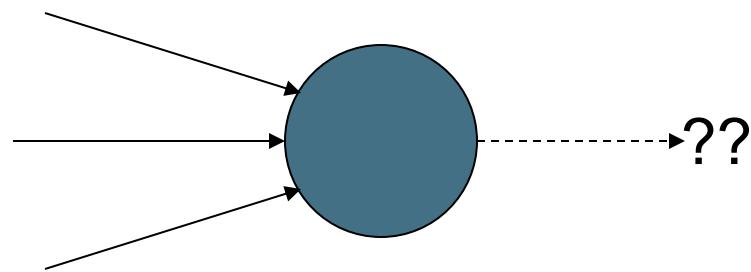
Pagerank scoring

- Imagine a user doing a random walk on web pages:
 - Start at a random page
 - At each step, go out of the current page along one of the links on that page, equiprobably
- “In the long run” each page has a long-term visit rate
 - use this as the page’s score
- Variant: rather than equiprobable, use text and link information to have probability of following a link: intelligent surfer [Richardson and Domingos 2001]



Not quite enough

- The web is full of dead-ends.
 - Random walk can get stuck in dead-ends.
 - Makes no sense to talk about long-term visit rates.



Teleporting

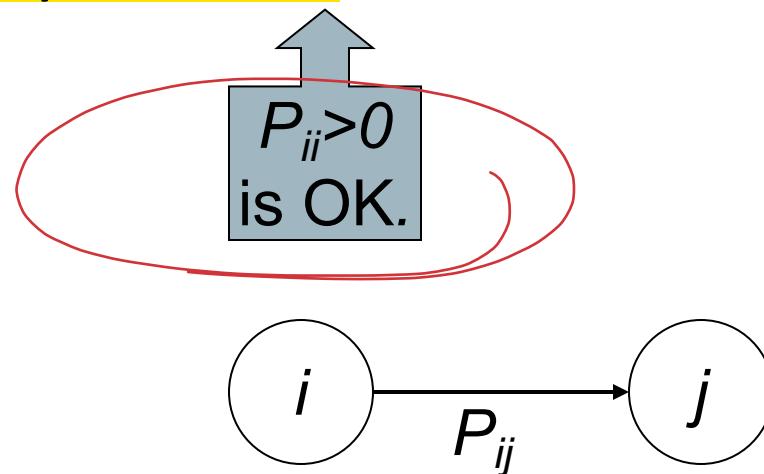
- At a dead end, jump to a random web page.
- At any non-dead end, with probability 10%, jump to a random web page.
 - With remaining probability (90%), go out on a random link.
 - 10% - a parameter.
 - “Teleportation” probability
 - Simulates a web users going somewhere else
 - Solves linear algebra problems....

Result of teleporting

- Now cannot get stuck locally.
- There is a long-term rate at which any page is visited (not obvious, will show this).
- How do we compute this visit rate?

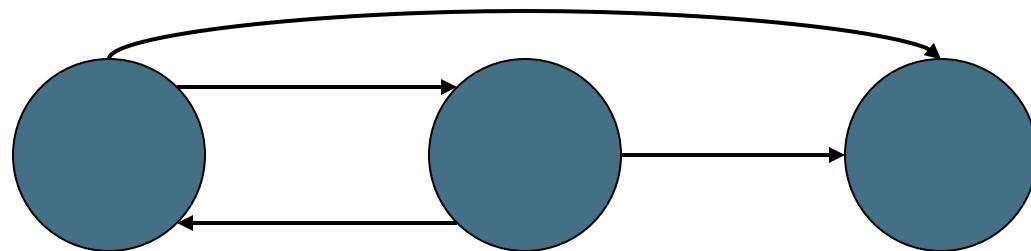
Markov chains

- A Markov chain consists of n states, plus an $n \times n$ transition probability matrix \mathbf{P} .
- At each step, we are in one of the states.
- For $1 \leq i, j \leq n$, the matrix entry P_{ij} tells us the probability of j being the next state, given we are currently in state i .



Markov chains

- Clearly, for all i , $\sum_{j=1}^n P_{ij} = 1$.
- Markov chains are abstractions of random walks.
- *Exercise:* represent the teleporting random walk from 3 slides ago as a Markov chain, for this case:



Ergodic Markov chains

- For any *ergodic* Markov chain, there is a unique long-term visit rate for each state.
 - *Steady-state probability distribution.*
- Over a long time-period, we visit each state in proportion to this rate.
- It doesn't matter where we start.
- Ergodic: no periodic patterns
 - Teleportation ensures ergodicity

Probability vectors

- A probability (row) vector $\mathbf{x} = (x_1, \dots x_n)$ tells us where the walk is at any point.
- E.g., $(000\dots 1\dots 000)$ means we're in state i .

1 i n

More generally, the vector $\mathbf{x} = (x_1, \dots x_n)$ means the walk is in state i with probability x_i .

$$\sum_{i=1}^n x_i = 1.$$

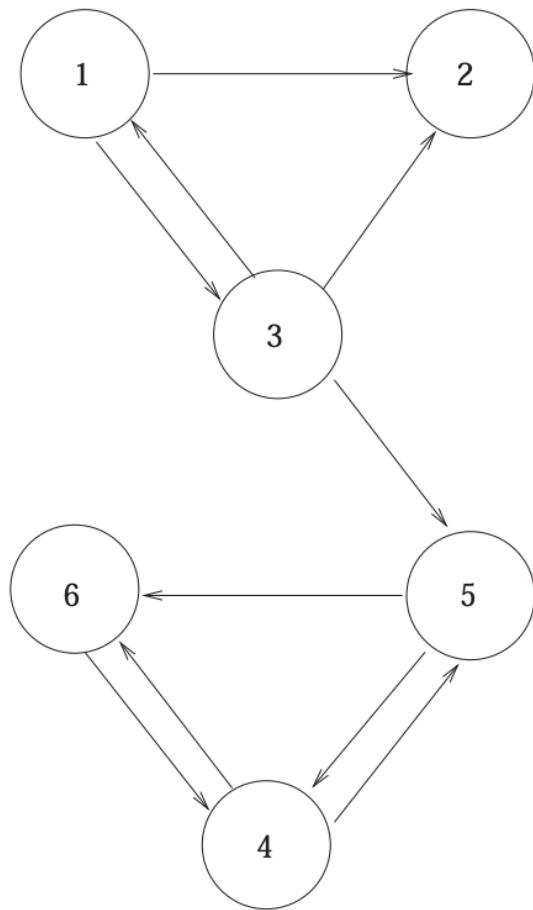
Change in probability vector

- If the probability vector is $\mathbf{x} = (x_1, \dots x_n)$ at this step, what is it at the next step?
- Recall that row i of the transition prob. matrix \mathbf{P} tells us where we go next from state i .
- So from \mathbf{x} , our next state is distributed as $\mathbf{x}\mathbf{P}$
 - The one after that is $\mathbf{x}\mathbf{P}^2$, then $\mathbf{x}\mathbf{P}^3$, etc.
 - (Where) Does this converge?
 - Running this and finding out is “the power method”
 - It’s actually the method of choice, done with sparse \mathbf{P}

How do we compute this vector?

- Let $\mathbf{a} = (a_1, \dots, a_n)$ denote the row vector of steady-state probabilities.
- If our current position is described by \mathbf{a} , then the next step is distributed as $\mathbf{a}\mathbf{P}$.
- But \mathbf{a} is the steady state, so $\mathbf{a}=\mathbf{a}\mathbf{P}$.
- Solving this matrix equation gives us \mathbf{a} .
 - So \mathbf{a} is the (left) eigenvector for \mathbf{P} .
 - Corresponds to the “principal” eigenvector of \mathbf{P} with the largest eigenvalue. (See: Perron-Frobenius theorem.)
 - Transition probability matrices always have largest eigenvalue 1.

Example: Mini web graph



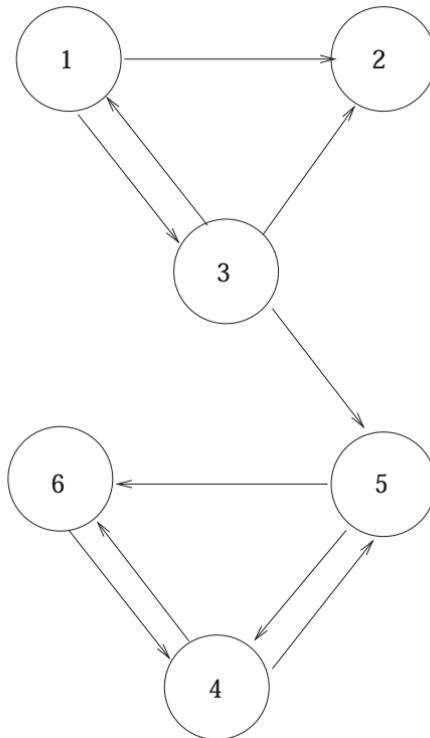
$$\mathbf{P} = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 4 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 5 & 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 6 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Example: Fixing sinks and teleporting

$$\bar{\mathbf{P}} = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

$$\bar{\mathbf{P}} = \alpha \bar{\mathbf{P}} + (1 - \alpha) \mathbf{e} \mathbf{e}^T / n = \begin{pmatrix} 1/60 & 7/15 & 7/15 & 1/60 & 1/60 & 1/60 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 19/60 & 19/60 & 1/60 & 1/60 & 19/60 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 7/15 & 7/15 \\ 1/60 & 1/60 & 1/60 & 7/15 & 1/60 & 7/15 \\ 1/60 & 1/60 & 1/60 & 11/12 & 1/60 & 1/60 \end{pmatrix}$$

Example: Doing power iteration



```

import numpy as np

x0 = np.matrix([1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

P = np.matrix([[1/60, 7/15, 7/15, 1/60, 1/60, 1/60],
               [1/6, 1/6, 1/6, 1/6, 1/6, 1/6],
               [19/60, 19/60, 1/60, 1/60, 19/60, 1/60],
               [1/60, 1/60, 1/60, 1/60, 7/15, 7/15],
               [1/60, 1/60, 1/60, 7/15, 1/60, 7/15],
               [1/60, 1/60, 1/60, 11/12, 1/60, 1/60]])

print(x0 * P)
[[0.09166667 0.16666667 0.11666667 0.26666667 0.16666667 0.19166667]]

print(x0 * P * P)
[[0.07666667 0.11791667 0.08291667 0.28916667 0.19666667 0.23666667]]

print(x0 * P * P * P * P * P)
[[0.05138229 0.07803542 0.05737917 0.34361667 0.20251667 0.26706979]]

print(x0 * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P)
[[0.0391419 0.05730065 0.04374176 0.37100521 0.20527182 0.28353866]]

print(x0 * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P * P)
[[0.03724891 0.05402154 0.04154868 0.37500616 0.20598094 0.28619378]]
  
```

Link analysis: HITS Kleinberg (1999)

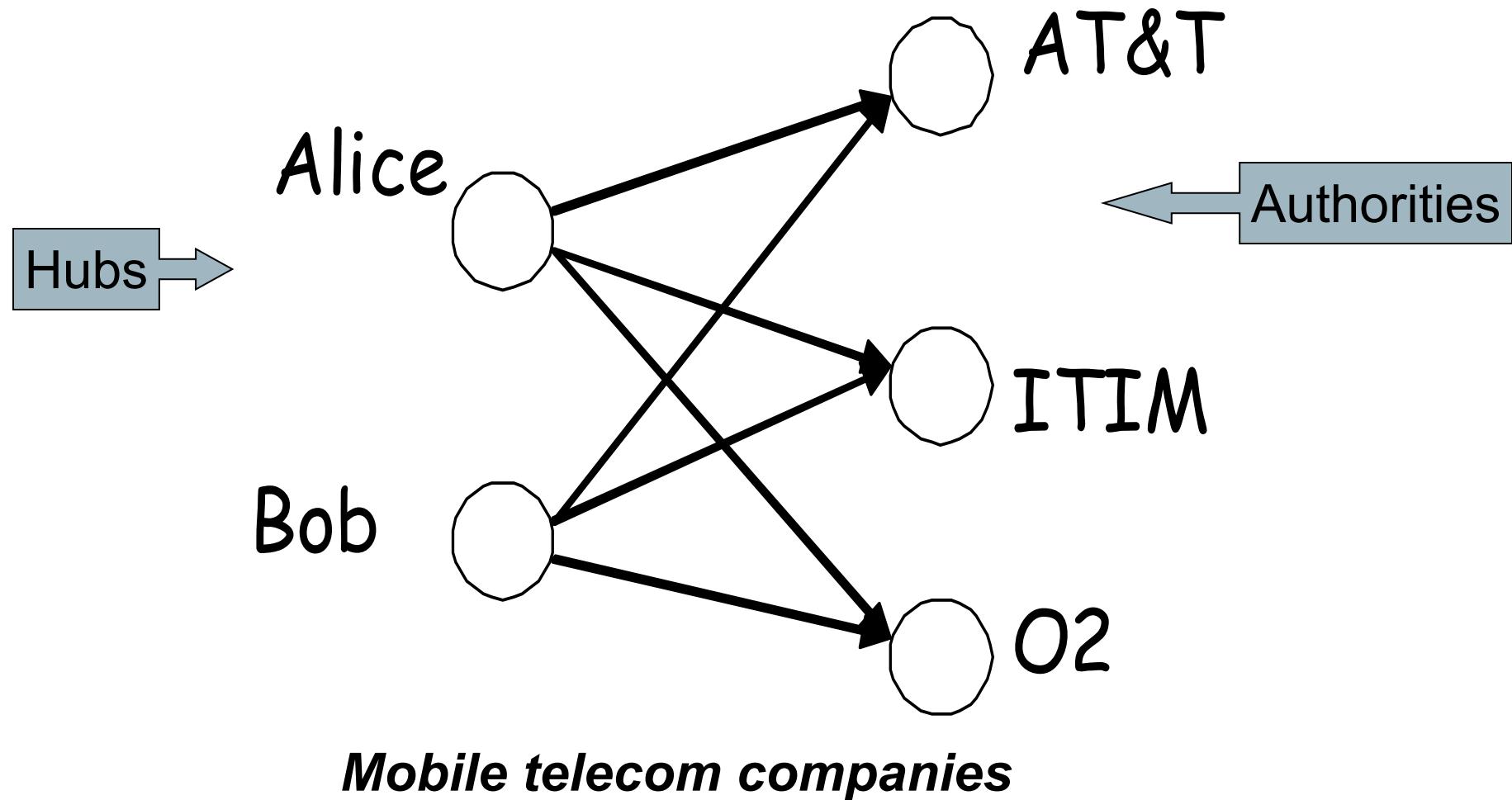
Hyperlink-Induced Topic Search (HITS)

- In response to a query, instead of an ordered list of pages each meeting the query, find two sets of inter-related pages:
 - *Hub pages* are good lists of links on a subject
 - e.g., “Bob’s list of cancer-related links.”
 - *Authority pages* occur recurrently on good hubs for the subject
- Best suited for “broad topic” queries rather than for page-finding queries
- Gets at a broader slice of common *opinion*

Hubs and Authorities

- Thus, a good hub page for a topic *points* to many authoritative pages for that topic.
- A good authority page for a topic is *pointed* to by many good hubs for that topic.
- Circular definition – will turn this into an iterative computation.

The hope



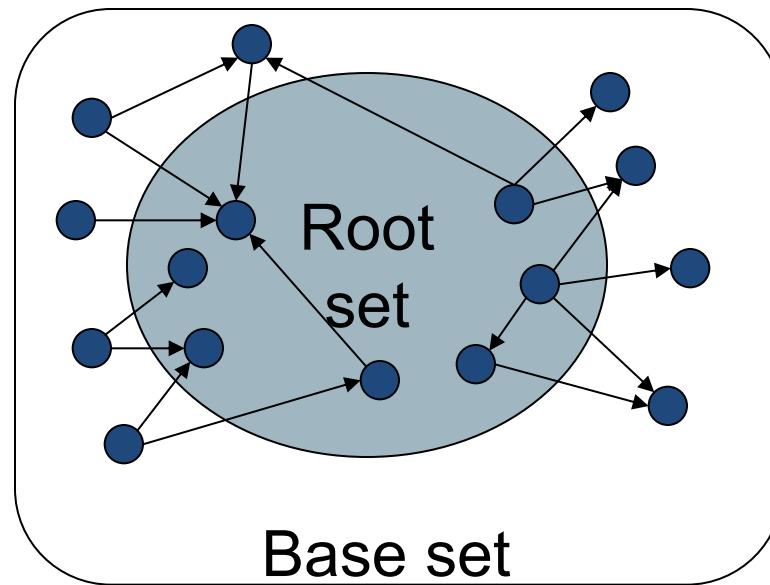
High-level scheme

- Extract from the web a base set of pages that *could* be good hubs or authorities.
- From these, identify a small set of top hub and authority pages;
→iterative algorithm.

Base set

- Given text query (say ***browser***), use a text index to get all pages containing ***browser***.
 - Call this the root set of pages.
- Add in any page that either
 - points to a page in the root set, or
 - is pointed to by a page in the root set.
- Call this the base set.

Visualization



Get in-links (and out-links) from a *connectivity server*

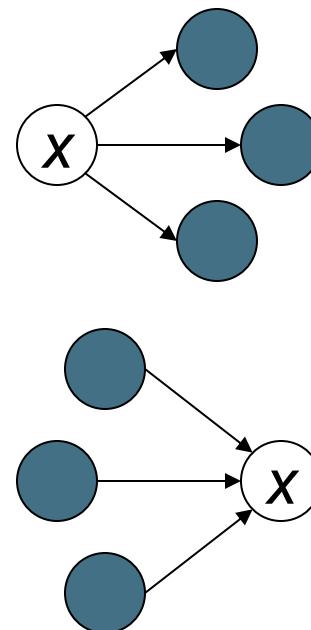
Distilling hubs and authorities

- Compute, for each page x in the base set, a hub score $h(x)$ and an authority score $a(x)$.
- Initialize: for all x , $h(x) \leftarrow 1$; $a(x) \leftarrow 1$;
- Iteratively update all $h(x)$, $a(x)$;  Key
- After iterations
 - output pages with highest $h()$ scores as top hubs
 - highest $a()$ scores as top authorities.

Iterative update

- Repeat the following updates, for all x :

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$



$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$

Scaling

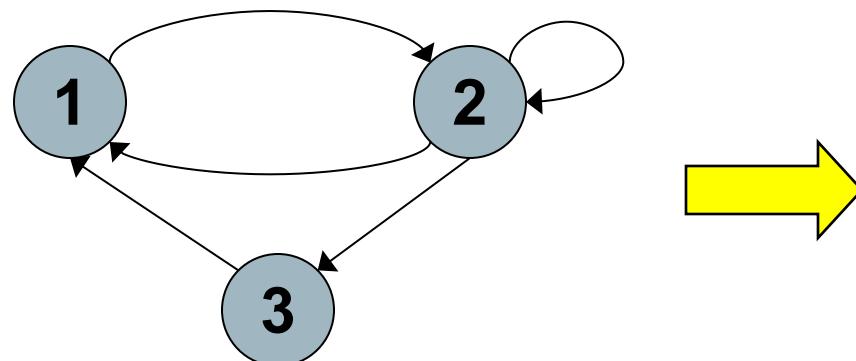
- To prevent the $h()$ and $a()$ values from getting too big, can scale down after each iteration.
- Scaling factor doesn't really matter:
 - we only care about the *relative* values of the scores.

How many iterations?

- Claim: relative values of scores will converge after a few iterations:
 - in fact, suitably scaled, $h()$ and $a()$ scores settle into a steady state!
 - proof of this comes later.
- In practice, ~5 iterations get you close to stability.

Proof of convergence

- $n \times n$ adjacency matrix A:
 - each of the n pages in the base set has a row and column in the matrix.
 - Entry $A_{ij} = 1$ if page i links to page j , else = 0.



	1	2	3
1	0	1	0
2	1	1	1
3	1	0	0

Hub/authority vectors

- View the hub scores $h()$ and the authority scores $a()$ as vectors with n components.
- Recall the iterative updates

$$h(x) \leftarrow \sum_{x \mapsto y} a(y)$$

$$a(x) \leftarrow \sum_{y \mapsto x} h(y)$$

Rewrite in matrix form

- $\mathbf{h} = \mathbf{A}\mathbf{a}$.
- $\mathbf{a} = \mathbf{A}^T\mathbf{h}$.

Recall \mathbf{A}^T is the transpose of \mathbf{A} .

Substituting, $\mathbf{h} = \mathbf{A}\mathbf{A}^T\mathbf{h}$ and $\mathbf{a} = \mathbf{A}^T\mathbf{A}\mathbf{a}$.

Thus, \mathbf{h} is an eigenvector of $\mathbf{A}\mathbf{A}^T$ and \mathbf{a} is an eigenvector of $\mathbf{A}^T\mathbf{A}$.

Further, our algorithm is a particular, known algorithm for computing eigenvectors: again, the *power iteration* method.

Guaranteed to converge.

Example authorities found

- (java) Authorities
 - .328 http://www.gamelan.com/ Gamelan
 - .251 http://java.sun.com/ JavaSoft Home Page
 - .190 http://www.digitalfocus.com/... The Java Developer: How Do I ...
 - .190 http://lightyear.ncsa.uiuc.edu/;srp/java/ javabooks.html
 - .183 http://sunsite.unc.edu/javafaq/javafaq.html comp.lang.java FAQ
- (censorship) Authorities
 - .378 http://www.eff.org/ EFFweb—The Electronic Frontier Foundation
 - .344 http://www.eff.org/blueribbon.html The Blue Ribbon Campaign for Online Free Speech
 - .238 http://www.cdt.org/ The Center for Democracy and Technology
 - .235 http://www.vtw.org/ Voters Telecommunications Watch
 - .218 http://www.aclu.org/ ACLU: American Civil Liberties Union

Issues

- Topic Drift
 - Off-topic pages can cause off-topic “authorities” to be returned
 - E.g., the neighborhood graph can be about a “super topic”
- Mutually Reinforcing Affiliates
 - Affiliated pages/sites can boost each others’ scores
 - Linkage between affiliated pages is not a useful signal

Resources

- IIR Chap 21
- Kleinberg, Jon (1999). [Authoritative sources in a hyperlinked environment. Journal of the ACM.](#) **46** (5): 604–632.
- <http://www2004.org/proceedings/docs/1p309.pdf>
- <http://www2004.org/proceedings/docs/1p595.pdf>
- <http://www2003.org/cdrom/papers/refereed/p270/kamvar-270-xhtml/index.html>
- <http://www2003.org/cdrom/papers/refereed/p641/xhtml/p641-mccurley.html>
- [The WebGraph framework I: Compression techniques \(Boldi et al. 2004\)](#)

Web Information Retrieval

Prof. Luca Becchetti: (very) rough notes on Markov Chains and Random Walks

While reviewing a first draft of this notes, I came across the [following tutorial](#) on Markov chains @[Towards Data Science](#), which I found extremely accurate and well done. I encourage you to take a look, it contains further points and nice examples that are worth reflecting upon.

Basics

Consider the process described by the following picture:

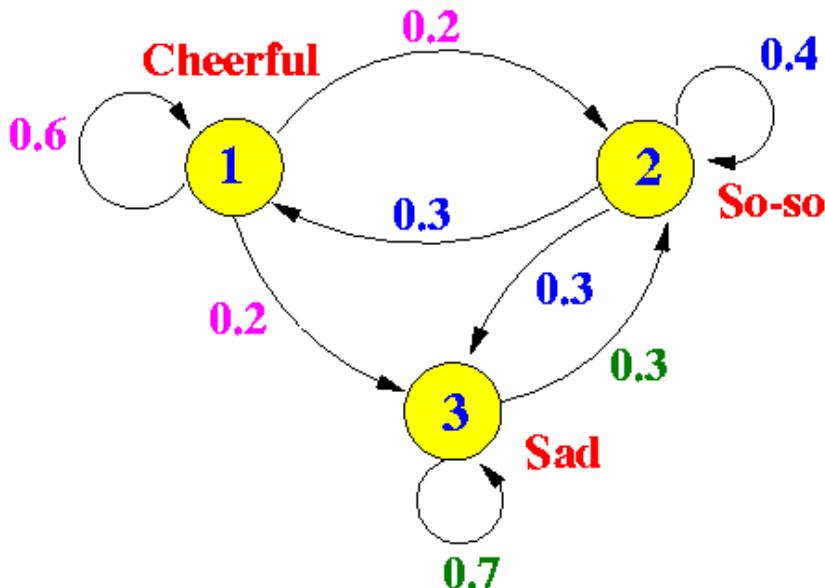


Image source: <http://www.mathcs.emory.edu/~cheung/>

Above we have a (directed) network $G = (V, E)$. Nodes represent **states**, directed links represent possible **transitions** between them. Each link (i, j) (note that order is important!) is labelled with P_{ij} , the probability that, if in state i at any round t , we "move" to state j in round $t + 1$. For example, $P_{23} = 0.3$ in the picture above.

Process. The process starts in some initial state at round 0 and it performs a transition to a different (possibly the same) state in each round, following an outgoing link of the current state with the probability associated to that link. Such a process is called a *Markov Chain* (often abbreviated as MC in the remainder of these notes).

Remark. To be precise, the one given above is an example of a *discrete* (transitions occur in discrete time steps), *homogeneous* (transition probabilities do not change over time), *finite-state* (we have a finite number of states, 3 in the example above) MC. As you may suspect, a number of variants are possible (and studied). In the remainder, we restrict to discrete, homogeneous, finite-state MCs, even though I just write MC for brevity.

Notation. We use $X(t)$ to denote the state of the Markov chain after t steps. We use P to denote the $n \times n$ matrix, whose entry (i, j) is P_{ij} . P is called the *transition matrix* of the MC.

Markov property. As the picture above also suggests, a Markov Chain (hence, MC) exhibits the following, crucial property:

$$\mathbf{P}(X(t) = j | X(t-1) = a_{t-1}, \dots, X(0) = a_0) = \mathbf{P}(X(t) = j | X(t-1) = a_{t-1}),$$

where a_r denotes the state in which the MC found itself at the end of round r , $r = 1, \dots, t-1$. Stated informally, what is going to happen next only depends on what is happening now. Note that

$$\boxed{\mathbf{P}(X(t) = j | X(t-1) = i) = P_{ij}}$$

Properties of \mathbf{P} . We assume henceforth that each state i has at least one outgoing link and that the matrix is stochastic, i.e., $\sum_{j:i \rightsquigarrow j} P_{ij} = 1$, where $i \rightsquigarrow j$ means that a link from i to j exists. Note that, for Pagerank, we enforce this property by adding n links from each dead end j , each pointing to a different state (including j itself) and with associated transition probability $1/n$.

MC evolution

After t steps, the MC is in each state with some probability (possibly, 0). This corresponds to a probability distribution $\mathbf{p}(t)$. Here, $\mathbf{p}(t)$ is an n -dimensional vector, such that its j -th entry is $\mathbf{p}_j(t) = \mathbf{P}(X(t) = j)$, i.e., the unconditional probability that the MC ends up in state j in round t . Since the MC has to be in some state at the end of each round (recall that we have no dead ends), we necessarily have $\sum_{j=1}^n \mathbf{p}_j(t) = 1$, for every t . There is an obvious relationship between $\mathbf{p}(t)$ and $\mathbf{p}(t-1)$. Namely, $\mathbf{p}_j(t)$ depends on $\mathbf{p}(t)$ as follows:

$$\mathbf{p}_j(t) = \sum_{i \rightsquigarrow j} \mathbf{p}_i(t-1) P_{ij}.$$

You should convince yourself that this relationship can be expressed in compact form as:

$$\mathbf{p}(t)^T = \mathbf{p}(t-1)^T P \quad (1)$$

Iterating over different rounds we finally obtain:

$$\mathbf{p}(t)^T = \mathbf{p}(0)^T P^t.$$

Rows or columns? If you transpose both sides of Eq. (1) you immediately obtain:

$$\mathbf{p}(t) = (P^T)^t \mathbf{p}(0).$$

Now, $\mathbf{p}(t)$ and $\mathbf{p}(0)$ are column vectors and P^T has columns (not rows) summing to 1, but the two forms are completely equivalent for all that matters. People often consider the first form when studying Markov chains, while a column representation is mostly preferred in the related literature on Spectral Graph Theory, a topic well beyond the scope of this lecture.

Ergodic Markov chains

A number of questions are of interest, the most important being:

1. What happens to $\mathbf{p}(t)$ as $t \rightarrow \infty$?
2. Does $\lim_{t \rightarrow \infty} \mathbf{p}(t)$ exist? Is it unique?

The answer to the questions above depends on the structure of the MC, i.e., the topology of the underlying, directed graph. In the remainder, we focus on the class of ergodic MCs, for which the answer to the above questions is always yes. We emphasize that the MC defining Pagerank is ergodic.

Definition. We say the state j is *accessible* from state i if a directed path from i to j exists in the MC.

Note that this implies that, for all $i, j \in V$: P_{ij}^k for some $k \geq 0$.

Definition. We say the i and j *communicate* if and only if j is accessible from i and viceversa.

Note that communicating relation is, mathematically speaking, an *equivalence relation*, namely, it satisfies the *reflexive*, *symmetric* and *transitive* properties. You are warmly invited to review the notion of equivalence relation, if needed.

Definition. A MC is *irreducible* if and only if it has a single communicating class. Otherwise, the MC is said *reducible*.

Remark. In practice, this means that the (directed graph) representing the MC is *strongly connected*, i.e., for every i and j there are directed paths connecting i to j and viceversa.

When a MC is reducible, $\lim_{t \rightarrow \infty} \mathbf{p}(t)$, provided it exists, in general depends on $\mathbf{p}(0)$.

Definition. A state j of a MC is *periodic* if an integer $\Delta > 1$ exists, such that

$P(X(t+s) = j | X(t) = j) = 0$, unless s is divisible by Δ . A MC is periodic if it contains at least one periodic state, otherwise it is said *aperiodic*.

Fact. In an irreducible Markov Chain, the presence of a single aperiodic state implies that all states are aperiodic, i.e., the chain is aperiodic.

Irreducible and aperiodic MCs identify the class of *ergodic* MCs. Ergodic MCs exhibit nice properties. First of all, if P is the transition matrix of an ergodic MC, the following holds:

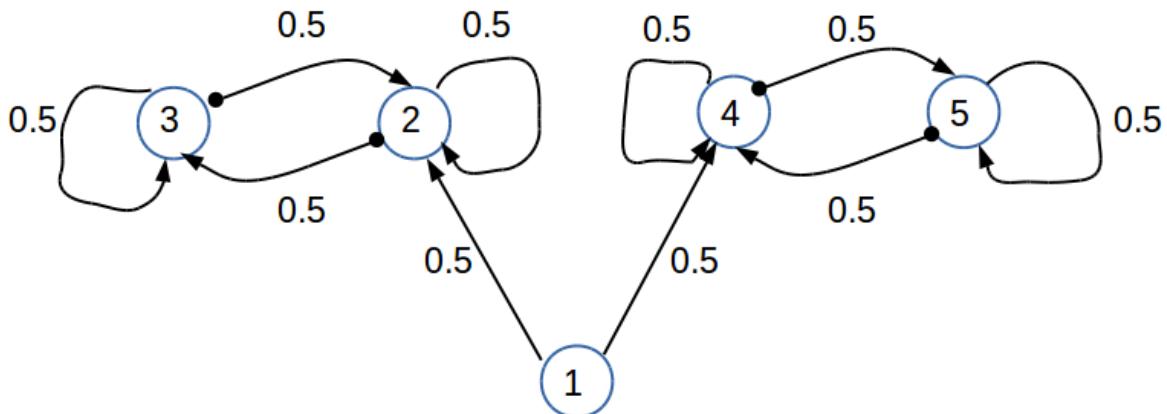
Property 1. If P is the transition matrix of an ergodic MC, an integer $k \geq 1$ exists, such that $P_{ij}^t > 0$, for every pair i, j of states, whenever $t \geq k$.

The behaviour of $\lim_{t \rightarrow \infty} \mathbf{p}(t)$ is interesting in ergodic MCs. Before we discuss this, we need the notion of stationary distribution.

Definition. A *stationary* (or *equilibrium*) distribution of a MC with transition matrix P is a probability distribution π , such that $\pi^T = \pi^T P$.

Note that, by definition, if a MC reaches a stationary distribution, it maintains that distribution forever.

Remark. In general, one can have multiple, stationary distributions. Consider the following example:



The transition matrix of this Markov chain is:

$$P = \begin{pmatrix} 0 & 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 1/2 \end{pmatrix}$$

It is easy to see that this MC has multiple, possible stationary distributions. One is $(\pi^{(1)})^T = (0, 1/2, 1/2, 0, 0)$, another one is $(\pi^{(2)})^T = (0, 0, 0, 1/2, 1/2)$. To see this, just observe that $(\pi^{(1)})^T P = \pi^{(1)}$ and $(\pi^{(2)})^T P = \pi^{(2)}$. Moreover, which stationary distribution is reached depends on the initial distribution $\mathbf{p}(0)$. For example, if $\mathbf{p}(0) = (0, 1, 0, 0, 0)$ the MC will converge to $\pi^{(1)}$ with probability 1 while, if $\mathbf{p}(0) = (1, 0, 0, 0, 0)$, the stationary distribution will be one between $\pi^{(1)}$ and $\pi^{(2)}$ with equal chances, i.e., the stationary distribution will be $\frac{1}{2}\pi^{(1)} + \frac{1}{2}\pi^{(2)}$ in this case (you can verify yourselves that this is yet another stationary distribution).

Ergodic MCs have interesting properties with respect to stationary distributions. In particular, we have the following, fundamental theorem:

Theorem. Any finite, ergodic MC has the following properties:

- The chain has a *unique* stationary distribution $\pi^T = (\pi_1, \dots, \pi_n)$.
- For every pair i, j , the limit $\lim_{t \rightarrow \infty} P_{ij}^t$ exists and is *independent* of i .
- For every j : $\lim_{t \rightarrow \infty} P_{ij}^t = \pi_j$.

Remarks. The theorem above has a number of important implications. The first consequence (actually, just a restatement of the third claim of the theorem) is that

$$\lim_{t \rightarrow \infty} \mathbf{p}(0) P^t = \pi^T,$$

independently of $\mathbf{p}(0)$. This implies that, picking any arbitrary initial distribution (for example, $\mathbf{p}(0) = \mathbf{e}^i$, with \mathbf{e}^i the i -th canonical vector), and applying the *power method*, i.e.,

$$\mathbf{p}(t)^T = \mathbf{p}(t-1)^T P, \text{ for } t > 0,$$

we have that $\mathbf{p}(t)$ eventually converges to the stationary distribution, whenever P is the transition matrix of an ergodic MC.

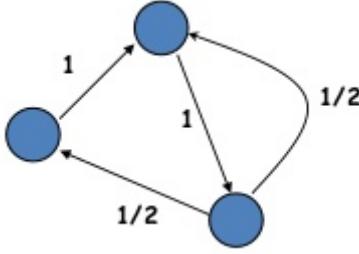
A second important consequence (an intuitive one, but one that would need a proof) is that

$$\lim_{t \rightarrow \infty} \frac{\sum_{k=0}^t \mathbf{p}_i(k)}{t} = \pi_i, \text{ for every } i$$

If one considers that the numerator is the expected number of times that i is visited over t consecutive rounds, the above equality simply states that the expected fraction of time spent by a MC in a state over a (very) long time span is very close to the stationary probability for that state.

Random walks and Markov chains

Given a (directed or undirected) graph $G = (V, E)$, a walk on G is a (not necessarily simple) path on G , i.e., a sequence of vertices (possibly with repetitions), such that v_2 may occur right after v_1 in the sequence if and only if the (possibly directed) edge (v_1, v_2) exists. In a random walk, the next edge that is traversed is chosen randomly: if at vertex u , in the next step the walker crosses one of the outgoing edges of u with the same probability $1/d_u$, where d_u denotes the out-degree of u (which of course is just the degree if G is undirected). This is the simplest version of random walks, more general notions are possible (e.g., in weighted graphs). Consider a random walk on a graph $G = (V, E)$, such as the following:



The random walk may be described by the following transition matrix P (or one obtained permuting the rows of P , depending on how we number vertices of the graph):

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1/2 & 1/2 & 0 \end{pmatrix}$$

A random walk is always a MC. If the graph is strongly connected, the resulting transition matrix is always stochastic (no dead ends) and thus it corresponds to a MC. In this case, the entries on row i correspond to outgoing links from i and they can take on two values (d_i is the out-degree of i):

$$P_{ij} = \begin{cases} \frac{1}{d_i}, & \text{if } (i, j) \in E, \\ 0, & \text{if } (i, j) \notin E \end{cases}$$

In fact, we can also regard a MC as a random walk, with the caveat that, in this case, jumps from one node of the graph to its neighbours do not occur with the same probability.

Important. Note that an undirected, connected graph is also strongly connected, hence a random walk is always defined on any undirected, connected graph.

Question: what happens if an undirected graph consists of two or more connected components? Is a random walk defined? If yes, is it ergodic?

Pagerank's random walk

Assume we have a Web graph $G = (V, E)$ with n vertices. In theory, we would like to score vertices in V using scores that reflects their centrality in G . The basic intuition is using a random walk on G to achieve this goal. Unfortunately, in most cases, we cannot directly use G to perform a random walk. The reason is that G may not be strongly connected, since it can have dead ends, periodic nodes and so on. In general, a random walk in G does not correspond to an ergodic MC (it may not even be defined, as is the case if we have dead ends). To obtain an ergodic MC that strongly depends on G 's underlying topology, we proceed as follows.

Step 1: removing dead ends. Assume \hat{P} is G 's transition matrix. If G contains a dead end i , the corresponding row in \hat{P} will be a row of 0's. Such rows are removed simply by replacing each of them with the vector $\frac{1}{n}\mathbf{1}^T$, which corresponds to adding n links from i to each distinct vertex in G (including i itself).

Henceforth, we consider the matrix $P = \hat{P} + \frac{1}{n}\mathbf{a}\mathbf{1}^T$, where \mathbf{a} is a column vector, such that $\mathbf{a}_i = 1$ if i is a dead end, $\mathbf{a}_i = 0$ otherwise. Note that P is now a stochastic matrix and that the corresponding MC might still be reducible and/or periodic.

Step 2: teleportation.

The random walk described by P is modified as follows. When at a generic vertex i of G (i.e., its possibly modified version in which dead ends were removed):

- With probability α :
 - Follow one of i 's outgoing links uniformly at random.
- With probability $1 - \alpha$:
 - Jump to any vertex (including i) uniformly at random (hence, with probability $1/n$).

This corresponds to a MC/random walk described by the following equation for the generic vertex i :

$$\mathbf{p}_i(t) = \alpha \sum_{j:(j,i) \in E} \mathbf{p}_j(t-1) P_{ji} + \frac{1-\alpha}{n} = \alpha \sum_{j:(j,i) \in E} \frac{\mathbf{p}_j(t-1)}{d_j} + \frac{1-\alpha}{n}$$

Written in matrix form for all vertices, this equation becomes:

$$\mathbf{p}(t)^T = \alpha \mathbf{p}(t-1)^T P + \frac{1-\alpha}{n} \mathbf{1}^T = \mathbf{p}(t-1)^T \left(\alpha P + \frac{1-\alpha}{n} \mathbf{1}\mathbf{1}^T \right),$$

where the second equality follows since $\mathbf{p}(t-1)^T \mathbf{1} = 1$ (recall that $\mathbf{p}(t-1)^T$ is a probability distribution). You should convince yourself that steps 1 and 2 enforce ergodicity of the corresponding MC. To this end, note that i) each vertex can be reached from each other vertex (possibly, with tiny probability) and ii) teleportation removes any form of periodicity (in principle, we can reach any other vertex in one step, with probability at least $1/n$).

References

This is a very succinct excerpt of material that can be found in many textbooks about linear algebra and/or probability and algorithms. For example, you find a thorough introduction in Chapter 7 of the following book:

Michael Mitzenmacher and Eli Upfal. Probability and Computing, 2nd edition. Cambridge University Press, 2017.

Evaluation of search results

Chapter 8 - IIR



SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

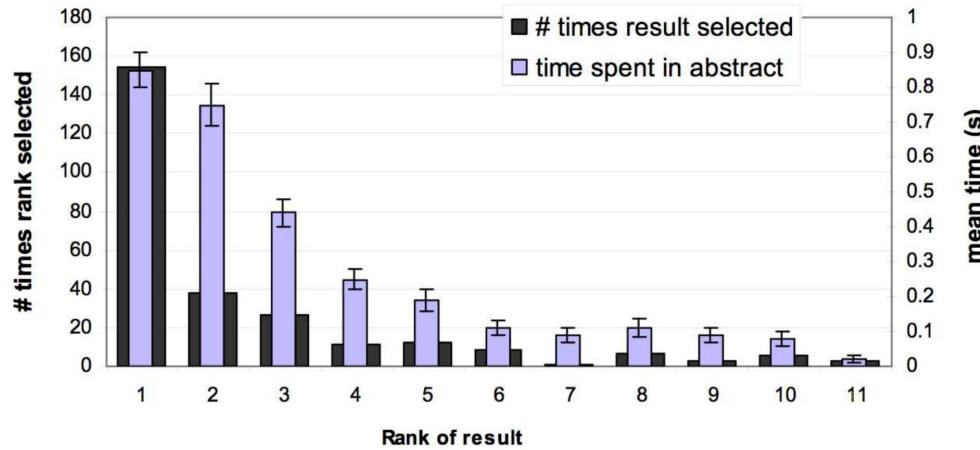
Quick Recap



SAPIENZA
UNIVERSITÀ DI ROMA

Users' Behavior

Looking vs. Clicking



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

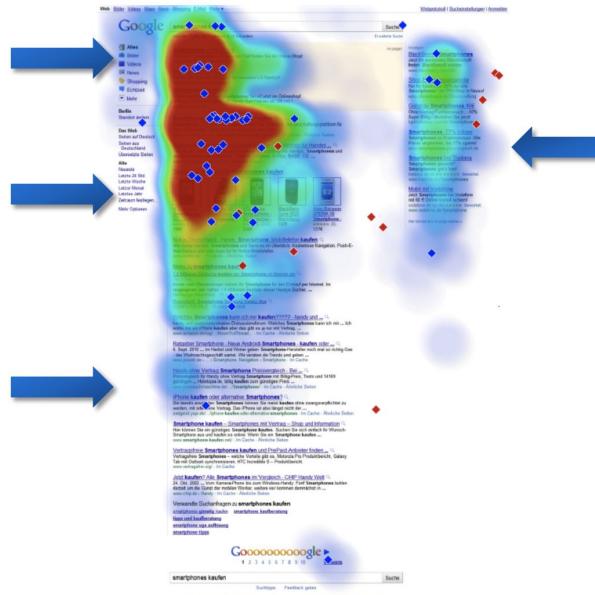
Attention of Users

Desktop search engine result page

Strong focus on the AdWords results

Rich Media elements move the attention/fixation points down the site

Nearly no focus on the organic results below the fold



People look more intensely on the right hand side than on the organic results below the fold

thinkinsights
with Google

Source: Eye Square Eye Tracking Study, 2011
Base: Respondents with contact to the stationary advertising on Google (n=38 Stationary)
Info: Aggregation over three brands

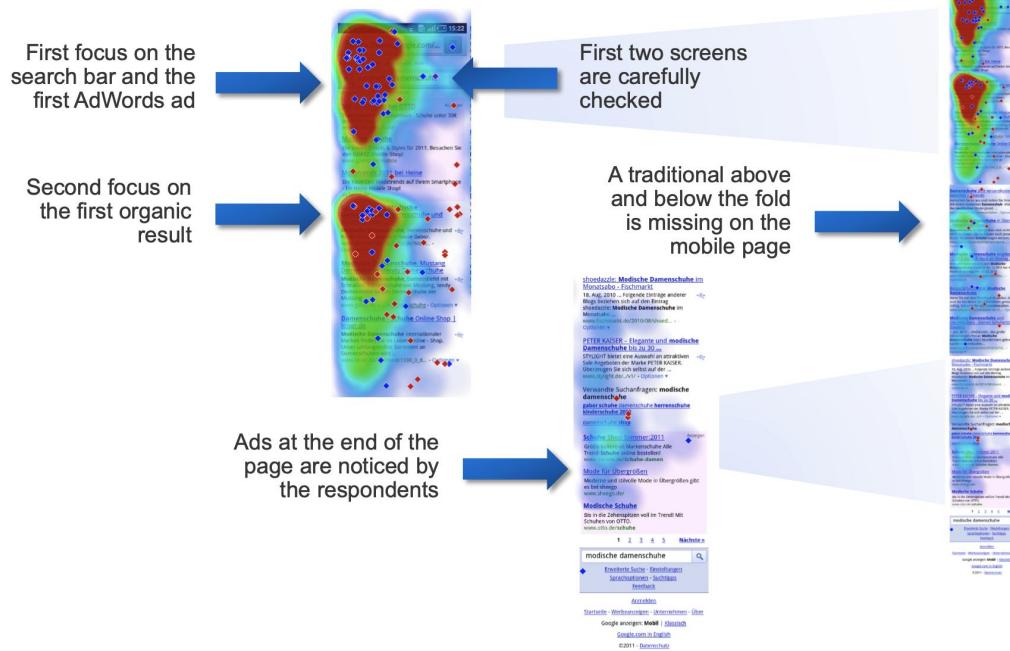
Google Think Insights – Eye Tracking Study, July 2011 7



SAPIENZA
UNIVERSITÀ DI ROMA

Attention of Users

Mobile search engine results page



Source: Eye Square Eye Tracking Study, 2011
Base: Respondents with contact to the mobile advertising on Google (n=50 mobile)
Info: Aggregation over three brands

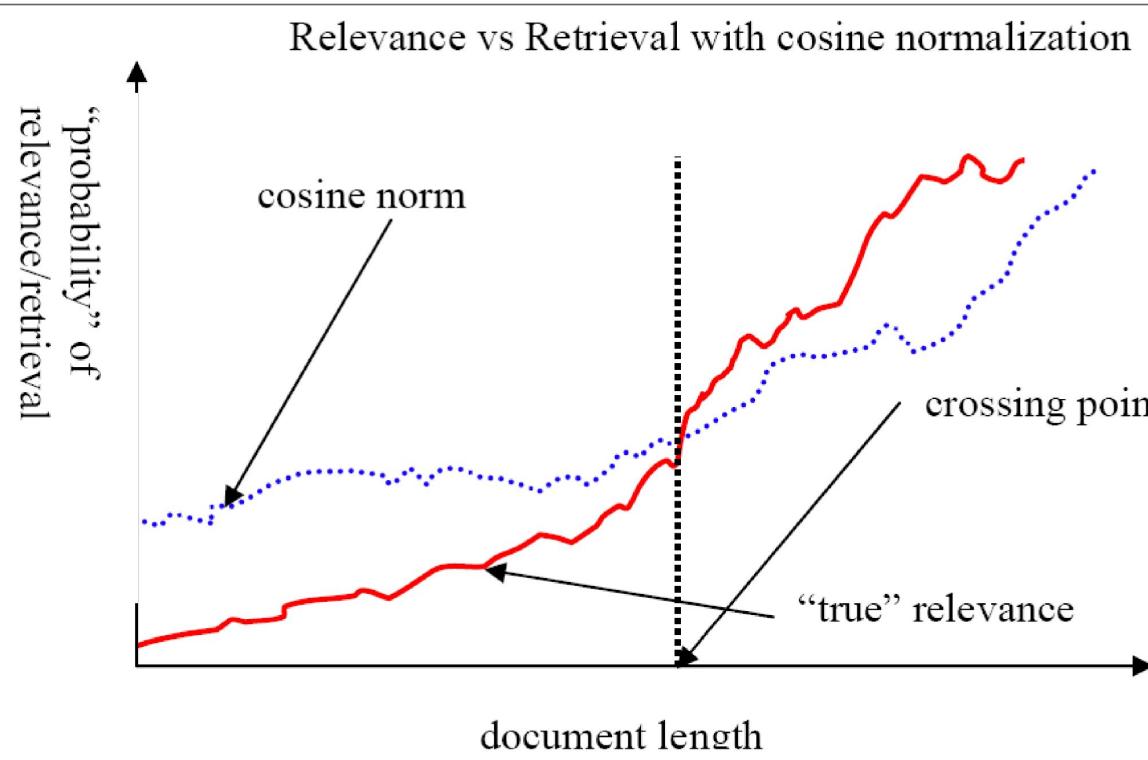
Google Think Insights – Eye Tracking Study, July 2011 8

thinkinsights
with Google



SAPIENZA
UNIVERSITÀ DI ROMA

Predicted and true probability of relevance



Effect on Effectiveness: Amit Singhal's experiments

Cosine	Pivoted Cosine Normalization				
	Slope				
	0.60	0.65	0.70	0.75	0.80
6,526	6,342	6,458	6,574	6,629	6,671
0.2840	0.3024	0.3097	0.3144	0.3171	0.3162
Improvement	+ 6.5%	+ 9.0%	+10.7%	+11.7%	+11.3%

- (relevant documents retrieved and (change in) average precision)



Evaluation of Search Results



SAPIENZA
UNIVERSITÀ DI ROMA

Outline

- Introduction to evaluation: Measures of an IR system
- Evaluation of unranked and ranked retrieval
- Evaluation benchmarks
- Result summaries

What to measure

- How fast does it index?
 - e.g., number of bytes per hour
- How fast does it search?
 - e.g., latency as a function of queries per second
- What is the cost per query?
 - in dollars



What to measure

- All of the preceding criteria are measurable: we can quantify speed / size / money
- However, the key measure for a search engine is **user happiness**.
- What is user happiness?
- Factors include:
 - Speed of response
 - Size of index
 - Uncluttered UI
 - Most important: **relevance**
 - (actually, maybe even more important: it's free)
- Note that none of these is sufficient: blindingly fast, but useless answers won't make a user happy.
- **How can we quantify user happiness?**



Who is the user?

- Who is the user we are trying to make happy?
- Web search engine: searcher. Success: Searcher finds what she was looking for. **Measure: rate of return to this search engine.**
- Web search engine: advertiser. **Success: Searcher clicks on ad. Measure: clickthrough rate.**
- Ecommerce: buyer. Success: Buyer buys something. **Measures: time to purchase, fraction of “conversions” of searchers to buyers.**
- Ecommerce: seller. Success: Seller sells something. **Measure: profit per item sold.**
- Enterprise: CEO. Success: Employees are more productive (because of effective search). **Measure: profit of the company.**



Most common definition of user happiness: Relevance

- User happiness is equated with the relevance of search results to the query.
- But how do you measure relevance?
- Standard methodology in information retrieval consists of three elements.
 - A benchmark document collection
 - A benchmark suite of queries
 - An assessment of the relevance of each query-document pair



Relevance: query vs. information need

- Relevance to what?
- First take: relevance to the query
- “Relevance to the query” is very problematic.
- Information need i: “I am looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.”
- This is an information need, not a query.
- Query q: [red wine white wine heart attack]
- Consider document d': At the heart of his speech was an attack on the wine industry lobby for downplaying the role of red and white wine in drunk driving.
- d' is an excellent match for query q . . .
- d' is not relevant to the information need i.



Relevance: query vs. information need

- User happiness can only be measured by relevance to an information need, not by relevance to queries.
- Our terminology is sloppy in these slides and in IIR: we talk about query-document relevance judgments even though we mean information-need-document relevance judgments.



Relevance: query vs. information need

- User happiness can only be measured by relevance to an information need, not by relevance to queries.
- Our terminology is sloppy in these slides and in IIR: we talk about query-document relevance judgments even though we mean information-need-document relevance judgments.



Unranked Evaluation



SAPIENZA
UNIVERSITÀ DI ROMA

Precision and Recall

- Precision (P) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#\text{(relevant items retrieved)}}{\#\text{(retrieved items)}} = P(\text{relevant}|\text{retrieved})$$

- Recall (R) is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#\text{(relevant items retrieved)}}{\#\text{(relevant items)}} = P(\text{retrieved}|\text{relevant})$$



Precision and Recall

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = TP / (TP + FP)$$

$$R = TP / (TP + FN)$$



Precision and Recall Tradeoff

- You can increase recall by returning more docs.
- Recall is a non-decreasing function of the number of docs retrieved.
 - A system that returns all docs has 100% recall!
- The converse is also true (usually): It's easy to get high precision for very low recall.
- Suppose the document with the largest score is relevant. How can we maximize precision?



F-Measure

- F allows us to trade off precision against recall.

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \quad \text{where} \quad \beta^2 = \frac{1 - \alpha}{\alpha}$$

- $\alpha \in [0,1]$ and thus $\beta^2 \in [0, \infty]$
- Most frequently used: balanced F with $\beta = 1$ or $\alpha = 0.5$
- This is the harmonic mean of P and R: $\frac{1}{F} = \frac{1}{2} \left(\frac{1}{P} + \frac{1}{R} \right)$
- What value range of β weights recall higher than precision?



Example of P, R, F1-Measure

	relevant	not relevant	
retrieved	20	40	60
not retrieved	60	1,000,000	1,000,060
	80	1,000,040	1,000,120

- $P = 20 / (20 + 40) = \frac{1}{3}$
- $R = 20 / (20 + 60) = \frac{1}{4}$
- $F1 = 2 / ((1 / \frac{1}{3}) + (1 / \frac{1}{4}))$



Accuracy

- Why do we use complex measures like precision, recall, and F?
- Why not something simple like accuracy?
- Accuracy is the fraction of decisions (relevant/nonrelevant) that are correct.
- In terms of the contingency table above,
$$\text{accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{FP} + \text{FN} + \text{TN}).$$



Example

- Compute precision, recall and F1 for this result set:

	relevant	not relevant
retrieved	18	2
not retrieved	82	1,000,000,000

- The snoogle search engine below always returns 0 results (“0 matching results found”), regardless of the query. Why does snoogle demonstrate that accuracy is not a useful measure in IR?



Why accuracy is a useless measure in IR

- Simple trick to maximize accuracy in IR: always say no and return nothing
- You then get 99.99% accuracy on most queries.
- Searchers on the web (and in IR in general) want to find something and have a certain tolerance for junk.
- It's better to return some bad hits as long as you return something.
 - → We use precision, recall, and F for evaluation, not accuracy.



F: Why harmonic mean?

- Why don't we use a different mean of P and R as a measure?
 - e.g., the arithmetic mean
- The simple (arithmetic) mean is close to 50% for snooglevsearch engine – which is too high.
- Desideratum: Punish really bad performance on either precision or recall.
- Taking the minimum achieves this.
- But minimum is not smooth and hard to weight.
- F (harmonic mean) is a kind of smooth minimum.



Difficulties in using precision, recall and F

- We need relevance judgments for information-need-document pairs – but they are expensive to produce.
- For alternatives to using precision/recall and having to produce relevance judgments – see end of this lecture.



Ranked Evaluation



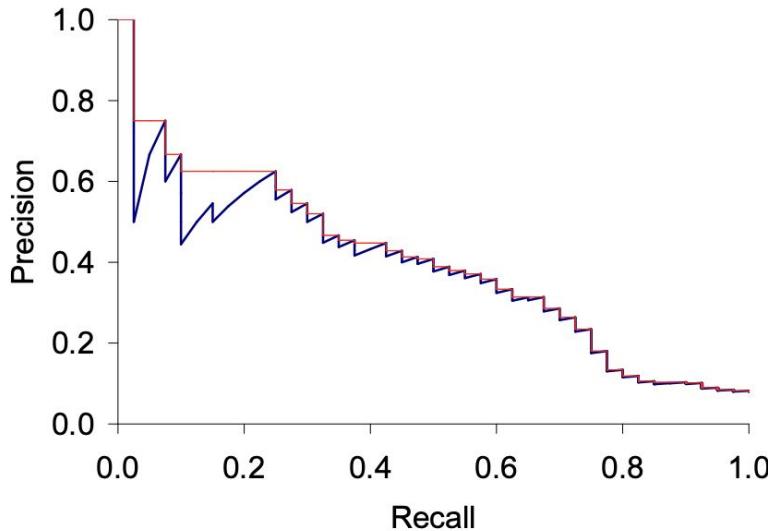
SAPIENZA
UNIVERSITÀ DI ROMA

Precision Recall (PR) Curve

- Precision/recall/F are measures for **unranked sets**.
- We can easily turn set measures into measures of **ranked lists**.
- Just compute the set measure for each “prefix”: the top 1, top 2, top 3, top 4 etc results
- Doing this for precision and recall gives you a **precision-recall curve**.



Precision Recall (PR) Curve



- Each point corresponds to a result for the top k ranked hits ($k = 1, 2, 3, \dots$).
- Interpolation (in red): Take maximum of all future points
- Rationale for interpolation: The user is willing to look at more stuff if both precision and recall get better.



11-point interpolated average precision

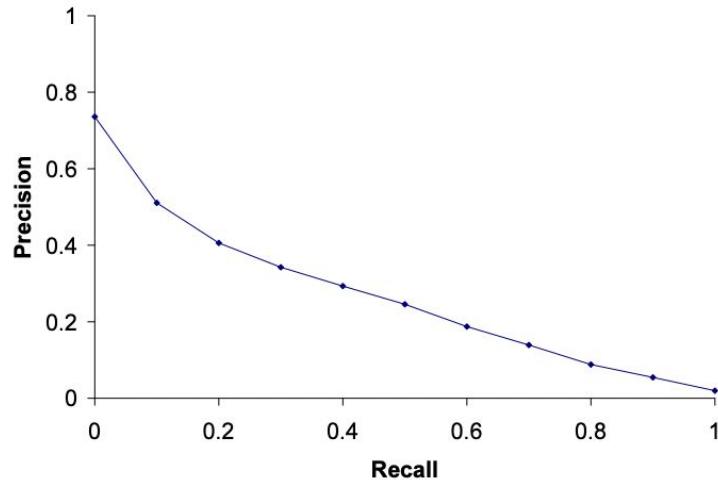
Recall	Interpolated Precision
0.0	1.00
0.1	0.67
0.2	0.63
0.3	0.55
0.4	0.45
0.5	0.41
0.6	0.36
0.7	0.29
0.8	0.13
0.9	0.10
1.0	0.08

11-point average: \approx
0.425

How can precision
at 0.0 be > 0 ?



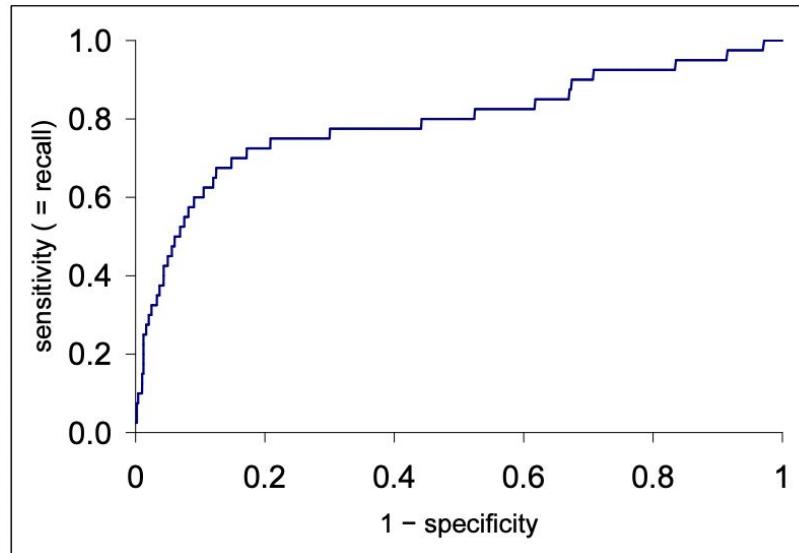
11-point interpolated average precision



- Compute interpolated precision at recall levels 0.0, 0.1, 0.2, . . .
- Do this for each of the queries in the evaluation benchmark
- Average over queries
- This measure measures performance at all recall levels.
- The curve is typical of performance levels at TREC.
- Note that performance is not very good!



ROC Curve



- Similar to precision-recall graph
- But we are only interested in the small area in the lower left corner.
- Precision-recall graph “blows up” this area.



Variance of measures like precision/recall

- For a test collection, it is usual that a system does badly on some information needs (e.g., $P = 0.2$ at $R = 0.1$) and really well on others (e.g., $P = 0.95$ at $R = 0.1$).
- Indeed, it is usually the case that the variance of the same system across queries is much greater than the variance of different systems on the same query.
- That is, there are easy information needs and hard ones.



Variance of measures like precision/recall

- For a test collection, it is usual that a system does badly on some information needs (e.g., $P = 0.2$ at $R = 0.1$) and really well on others (e.g., $P = 0.95$ at $R = 0.1$).
- Indeed, it is usually the case that the variance of the same system across queries is much greater than the variance of different systems on the same query.
- That is, there are easy information needs and hard ones.



Precision@K

- Set a rank threshold K
- Compute % relevant in top K
- Ignores documents ranked lower than K
- Ex: 
 - Prec@3 of 2/3
 - Prec@4 of 2/4
 - Prec@5 of 3/5

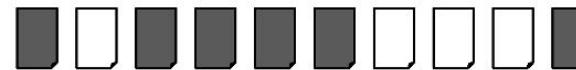


Average Precision



= the relevant documents

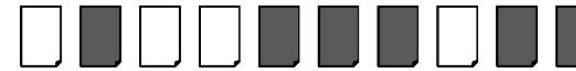
Ranking #1



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

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

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

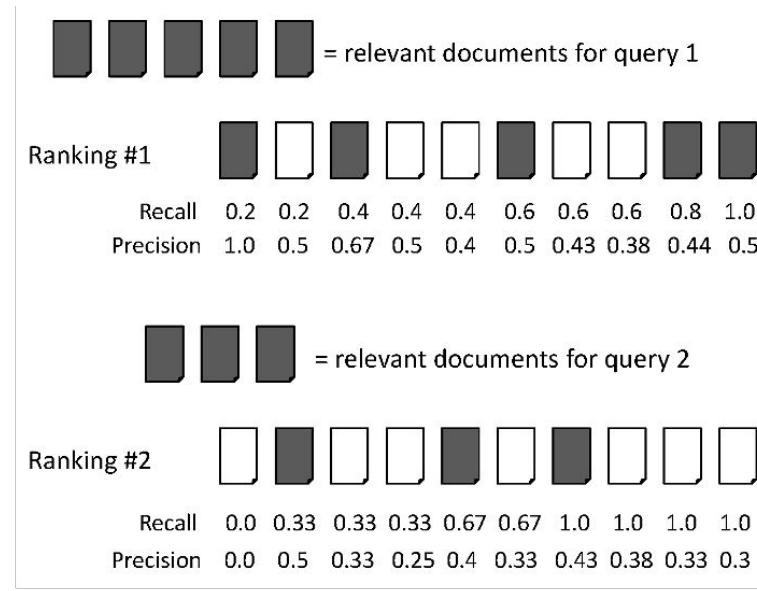


Mean Average Precision (MAP)

- Consider rank position of each relevant doc
 - K₁, K₂, ... K_R
- Compute Precision@K for each K₁, K₂, ... 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



Mean Average Precision (MAP)



$$\text{average precision query 1} = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$

$$\text{average precision query 2} = (0.5 + 0.4 + 0.43)/3 = 0.44$$

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



Mean Average Precision (MAP)

- If a relevant document never gets retrieved, we assume the precision corresponding to that relevant doc to be zero
- MAP is macro-averaging: each query counts equally
- Now perhaps most commonly used measure in research papers
- Good for web search?
- MAP assumes user is interested in finding many relevant documents for each query
- MAP requires many relevance judgments in text collection



The case of a single relevant doc

- Scenarios:
 - known-item search
 - navigational queries
 - looking for a fact
- Search Length = Rank of the answer
 - measures a user's effort



Mean Reciprocal Rank (MRR)

- Consider rank position, K, of first relevant doc
- Reciprocal Rank score = $1 / K$
- MRR is the mean RR across multiple queries



Critique of pure relevance

- Relevance vs Marginal Relevance
 - A document can be redundant even if it is highly relevant
 - Duplicates
 - The same information from different sources
 - Marginal relevance is a better measure of utility for the user
 - But harder to create evaluation set
- Using facts/entities as evaluation unit can more directly measure true recall
- Also related is seeking diversity in first page results
- See also recent paper on evaluation by N. Fuhr:
https://www.is.inf.uni-due.de/bib/pdf/ir/Fuhr_17b.pdf





artificial intelligence news

X |

All News Videos Images Maps More Settings Tools

About 601,000,000 results (0.64 seconds)

<https://www.sciencedaily.com> > news > computers_math

Artificial Intelligence News -- ScienceDaily

Artificial Intelligence News. Everything on AI including futuristic robots with artificial intelligence, computer models of human intelligence and more.

[Artificial emotional intelligence - Photonics for artificial...](https://www.sciencedaily.com) · 'Audeo' teaches artificial...

Fair

<https://artificialintelligence-news.com>

AI News - Artificial Intelligence News

Artificial Intelligence News provides the latest AI news and trends. Explore industry research and reports from the frontline of AI technology news.

[News](#) · [Featured: AI News' list of...](#) · [British intelligence agency...](#) · [Engagement](#)

Good

<https://news.mit.edu> > topic > [artificial-intelligence2](#)

Artificial intelligence | MIT News | Massachusetts Institute of ...

Artificial intelligence. Download RSS feed: [News Articles](#) / In the Media. Displaying 1 - 15 of 706 news articles related to this topic. Show: [News Articles](#).

Fair

<https://www.businessinsider.com> > [artificial-intelligence](#)

Artificial Intelligence News: Latest Advancements in AI ...

Artificial Intelligence (AI), or machine intelligence, is the field developing computers and robots capable of parsing data contextually to provide requested ...

What is Artificial Intelligence (AI)?



How does Artificial Intelligence work?



What are the different types of Artificial Intelligence?



[▼ Show more](#)

People also ask

What is the current status of artificial intelligence in the world?



What is the most advanced AI right now?



What is AI being used for today?



[Feedback](#)

<https://www.bbc.co.uk> > news > topics > [artificial-intelligence](#)...

Artificial intelligence - BBC News

All the latest news about Artificial Intelligence from the BBC. ... An outright ban on some AI systems, such as "social scoring" by governments, is proposed for the ...



SAPIENZA
UNIVERSITÀ DI ROMA

Discounted Cumulative Gain

- Popular measure for evaluating web search and related tasks
- Two assumptions:
 - Highly relevant documents are more useful than marginally relevant document
 - the lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined



Discounted Cumulative Gain

- Uses graded relevance as a measure of usefulness, or gain, from examining a document
- Gain is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks
- Typical discount is $1/\log(\text{rank})$
 - With base 2, the discount at rank 4 is $1/2$, and at rank 8 it is $1/3$



Summarize a Ranking: DCG

- What if relevance judgments are in a scale of $[0,r]$? $r>2$
- Cumulative Gain (CG) at rank n
 - Let the ratings of the n documents be r_1, r_2, \dots, r_n (in ranked order)
 - $CG = r_1 + r_2 + \dots + r_n$
- Discounted Cumulative Gain (DCG) at rank n
 - $DCG = r_1 + r_2/\log_2 2 + r_3/\log_2 3 + \dots + r_n/\log_2 n$
 - We may use any base for the logarithm, e.g., base=b



Discounted Cumulative Gain

- DCG is the total gain accumulated at a particular rank p:

$$DCG_p = rel_1 + \sum_{i=2}^p \frac{rel_i}{\log_2 i}$$

- Alternative formulation:

$$\underline{DCG_p = \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log(1+i)}}$$

- used by some web search companies
- emphasis on retrieving highly relevant documents



DCG Example

- 10 ranked documents judged on 0-3 relevance scale:
3, 2, 3, 0, 0, 1, 2, 2, 3, 0
- discounted gain:
3, $2/1$, $3/1.59$, 0, 0, $1/2.59$, $2/2.81$, $2/3$, $3/3.17$, 0
 $= 3, 2, 1.89, 0, 0, 0.39, 0.71, 0.67, 0.95, 0$
- DCG:
3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61



Summarize a Ranking: NDCG

- Normalized Cumulative Gain (NDCG) at rank n
 - Normalize DCG at rank n by the DCG value at rank n of the ideal ranking
 - The ideal ranking would first return the documents with the highest relevance level, then the next highest relevance level, etc
 - Compute the precision (at rank) where each (new) relevant document is retrieved => $p(1), \dots, p(k)$, if we have k rel. Docs
- NDCG is now quite popular in evaluating Web search



NDCG Example

4 documents: d_1, d_2, d_3, d_4

i	Ground Truth		Ranking Function ₁		Ranking Function ₂	
	Document Order	r_i	Document Order	r_i	Document Order	r_i
1	d_4	2	d_3	2	d_3	2
2	d_3	2	d_4	2	d_2	1
3	d_2	1	d_2	1	d_4	2
4	d_1	0	d_1	0	d_1	0
	$NDCG_{GT}=1.00$		$NDCG_{RF1}=1.00$		$NDCG_{RF2}=0.9203$	

$$DCG_{GT} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

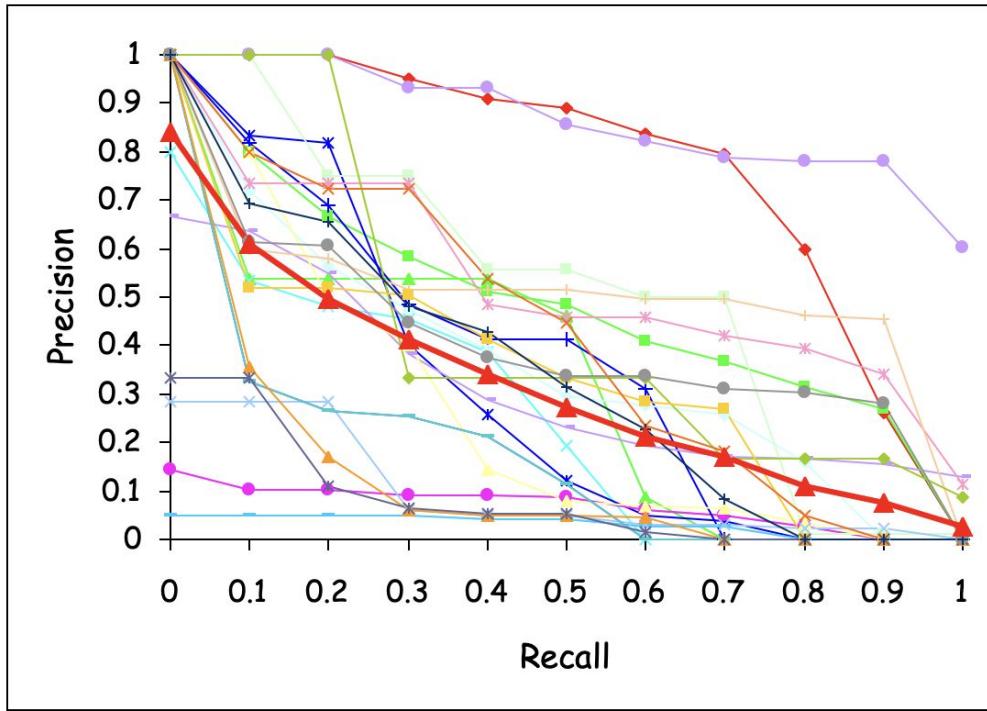
$$DCG_{RF1} = 2 + \left(\frac{2}{\log_2 2} + \frac{1}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.6309$$

$$DCG_{RF2} = 2 + \left(\frac{1}{\log_2 2} + \frac{2}{\log_2 3} + \frac{0}{\log_2 4} \right) = 4.2619$$

$$MaxDCG = DCG_{GT} = 4.6309$$



What Query Averaging Hides



Slide from Doug Oard's presentation, originally from Ellen Voorhees' presentation



Benchmarks



SAPIENZA
UNIVERSITÀ DI ROMA

What we need for a benchmarks

- A collection of documents
 - Documents should be representative of the documents we expect to see in reality.
- A collection of information needs (often incorrectly called queries)
 - Information needs should be representative of the information needs we expect to see in reality.
- Human relevance assessments
 - We need to hire/pay “judges” or assessors to do this.
 - Expensive, time-consuming
 - Judges should be representative of the users we expect to see in reality.



First standard relevance benchmark: Cranfield

- Pioneering: first testbed allowing precise quantitative measures of information retrieval effectiveness
- Late 1950s, UK
- 1398 abstracts of aerodynamics journal articles, a set of 225 queries, exhaustive relevance judgments of all query-document-pairs
- Too small, too untypical for serious IR evaluation today



Second-generation relevance benchmark: TREC

- TREC = Text Retrieval Conference (TREC)
- Organized by the U.S. National Institute of Standards and Technology (NIST)
- TREC is actually a set of several different relevance benchmarks.
- Best known: TREC Ad Hoc, used for first 8 TREC evaluations between 1992 and 1999
- 1.89 million documents, mainly newswire articles, 450 information needs
- No exhaustive relevance judgments – too expensive
- Rather, NIST assessors' relevance judgments are available only for the documents that were among the top k returned for some system which was entered in the TREC evaluation for which the information need was developed.



Example of more recent benchmark: ClueWeb09

- 1 billion web pages
- 25 terabytes (compressed: 5 terabyte)
- Collected January/February 2009
- 10 languages
- Unique URLs: 4,780,950,903 (325 GB uncompressed, 105 GB compressed)
- Total Outlinks: 7,944,351,835 (71 GB uncompressed, 24 GB compressed)



Validity of relevance assessments

- Relevance assessments are only usable if they are consistent.
- If they are not consistent, then there is no “truth” and experiments are not repeatable.
- How can we measure this consistency or agreement among judges?
- → Kappa measure



Kappa measure

- Kappa is measure of how much judges agree or disagree.
- Designed for categorical judgments
- Corrects for chance agreement
- $P(A)$ = proportion of time judges agree
- $P(E)$ = what agreement would we get by chance

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)}$$

- $\kappa = ?$ for (i) chance agreement (ii) total agreement



Kappa measure

- Values of κ in the interval $[2/3, 1.0]$ are seen as acceptable.
- With smaller values: need to redesign relevance assessment methodology used etc.



Calculating the Kappa statistics

		Judge 2 Relevance			Observed proportion of the times the judges agreed
		Yes	No	Total	
Judge 1	Yes	300	20	320	
	No	10	70	80	
Total	310	90	400		

the times the judges agreed

$$P(A) = (300 + 70)/400 = 370/400 = 0.925$$

Pooled marginals

$$P(\text{nonrelevant}) = (80 + 90)/(400 + 400) = 170/800 = 0.2125$$

$$P(\text{relevant}) = (320 + 310)/(400 + 400) = 630/800 = 0.7878$$

Probability that the two judges agreed by chance $P(E) =$

$$P(\text{nonrelevant})^2 + P(\text{relevant})^2 = 0.2125^2 + 0.7878^2 = 0.665$$

Kappa statistic $\kappa = (P(A) - P(E))/(1 - P(E)) =$

$$(0.925 - 0.665)/(1 - 0.665) = 0.776 \text{ (still in acceptable range)}$$



Interjudge agreement at TREC

information need	number of docs judged	disagreements
51	211	6
62	400	157
67	400	68
95	400	110
127	400	106



Impact of interjudge disagreement

- Judges disagree a lot. Does that mean that the results of information retrieval experiments are meaningless?
 - No.
- Large impact on absolute performance numbers
- Virtually no impact on ranking of systems
- Supposes we want to know if algorithm A is better than algorithm B
- An information retrieval experiment will give us a reliable answer to this question . . .
 - . . . even if there is a lot of disagreement between judges.



Evaluation at Large Search Engines

- Recall is difficult to measure on the web
- Search engines often use precision at top k, e.g., $k = 10 \dots$
 - \dots or use measures that reward you more for getting rank 1 right than for getting rank 10 right.
- Search engines also use non-relevance-based measures.
 - Example 1: clickthrough on first result
 - Not very reliable if you look at a single clickthrough (you may realize after clicking that the summary was misleading and the document is nonrelevant) \dots
 - \dots but pretty reliable in the aggregate.
 - Example 2: Ongoing studies of user behavior in the lab – recall last lecture
 - Example 3: A/B testing



A/B testing

- Purpose: Test a single innovation
- Prerequisite: You have a large search engine up and running.
- Have most users use old system
- Divert a small proportion of traffic (e.g., 1%) to the new system that includes the innovation
- Evaluate with an “automatic” measure like clickthrough on first result
- Now we can directly see if the innovation does improve user happiness.
- Probably the evaluation methodology that large search engines trust most



Results Summary



SAPIENZA
UNIVERSITÀ DI ROMA

How do we present results to users?

- Originally, as a list – aka “10 blue links”
- How should each document in the list be described?
- This description is crucial.
- The user often can identify good hits (= relevant hits) based on the description.
- No need to actually view any document



Each result is

- Most commonly: doc title, url, some metadata . . .
- . . . and a summary
- How do we “compute” the summary?



Summaries / Snippets

- Two basic kinds: (i) static (ii) dynamic
- A **static summary** of a document is always the same, regardless of the query that was issued by the user.
- **Dynamic summaries** are **query-dependent**. They attempt to explain why the document was retrieved for the query at hand

[https://en.wikipedia.org › wiki › Tower_of_fields ▾](https://en.wikipedia.org/wiki/Tower_of_fields)

Tower of fields - Wikipedia

In mathematics, a **tower of fields** is a sequence of **field** extensions $F_0 \subseteq F_1 \subseteq \dots \subseteq F_n \subseteq \dots$ The name comes from such sequences often being written in the form. A **tower of fields** may be finite or infinite.



Static Summaries

- In typical systems, the static summary is a subset of the document.
- Simplest heuristic: the first 50 or so words of the document
- More sophisticated: extract from each document a set of “key” sentences
 - Simple NLP heuristics to score each sentence
 - Summary is made up of top-scoring sentences.
 - Machine learning approach
- Most sophisticated: complex NLP to synthesize/generate a summary
 - For most IR applications: not quite ready for prime time yet

<https://en.wikipedia.org/wiki/Volkswagen> ▾

Volkswagen - Wikipedia

(listen)), is a German motor vehicle manufacturer founded in 1937 by the German Labour Front, known for the iconic Beetle and headquartered in Wolfsburg. It is ...

Founder: [German Labour Front](#)

Founded: 1937; 84 years ago

Key people: Ralf Brandstaetter (brand CEO)

Industry: [Automotive](#)



Dynamic Summaries

- Present one or more “windows” or snippets within the document that contain several of the query terms.
- Prefer snippets in which query terms occurred as a phrase
- Prefer snippets in which query terms occurred jointly in a small window
- The summary that is computed this way gives the entire content of the window – all terms, not just the query terms.



Generating Dynamic Summaries

- Where do we get these other terms in the snippet from?
- We cannot construct a dynamic summary from the positional inverted index – at least not efficiently.
- We need to cache documents.
- The positional index tells us: query term occurs at position 4378 in the document.
- Byte offset or word offset?
- Note that the cached copy can be outdated
- Don't cache very long documents – just cache a short prefix



Dynamic Summaries

- Real estate on the search result page is limited → snippets must be short . . .
 - . . . but snippets must be long enough to be meaningful.
- Snippets should communicate whether and how the document answers the query.
- Ideally: linguistically well-formed snippets
- Ideally: the snippet should answer the query, so we don't have to look at the document.
- Dynamic summaries are a big part of user happiness because . . .
 - . . . we can quickly scan them to find the relevant document we then click on.
 - . . . in many cases, we don't have to click at all and save time.



Main Takeaways

- Introduction to evaluation: Measures of an IR system
- Evaluation of unranked and ranked retrieval
- Evaluation benchmarks
- Result summaries



Relevance Feedback and Query Expansion

Chapter 9 - IIR



SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Quick Recap



SAPIENZA
UNIVERSITÀ DI ROMA

Relevance

- We will evaluate the quality of an information retrieval system and, in particular, its ranking algorithm with respect to relevance.
- A document is relevant if it gives the user the information she was looking for.
- To evaluate relevance, we need an evaluation benchmark with three elements:
 - A benchmark document collection
 - A benchmark suite of queries
 - An assessment of the relevance of each query-document pair



Query vs. Information Need

- The notion of “relevance to the query” is very problematic.
- **Information need i:** You are looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.
- **Query q:** wine and red and white and heart and attack
- Consider document d':
 - He then launched into the heart of his speech and attacked the wine industry lobby for downplaying the role of red and white wine in drunk driving.
- d' is relevant to the query q, but d' is **not** relevant to the information need i.
- User happiness/satisfaction (i.e., how well our ranking algorithm works) can only be measured by **relevance to information needs, not by relevance to queries.**



Precision and Recall

- Precision (P) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#\text{(relevant items retrieved)}}{\#\text{(retrieved items)}} = P(\text{relevant}|\text{retrieved})$$

- Recall (R) is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#\text{(relevant items retrieved)}}{\#\text{(relevant items)}} = P(\text{retrieved}|\text{relevant})$$



Motivation



SAPIENZA
UNIVERSITÀ DI ROMA

Improving Recall

- Main topic today: two ways of improving recall: relevance feedback and query expansion
- As an example consider query q : [aircraft] . . .
- . . . and document d containing “plane”, but not containing “aircraft”
- A simple IR system will not return d for q .
- Even if d is the most relevant document for q !
- We want to change this:
 - Return relevant documents even if there is no term match with the (original) query



In fact... a lousier definition of Recall

- Loose definition of recall in this lecture: “increasing the number of relevant documents returned to user”
- This may actually decrease recall on some measures, e.g., when expanding “jaguar” to “jaguar AND panthera”
 - . . . which eliminates some relevant documents, but increases relevant documents returned on top pages



Options for improving recall

- Local: Do a “local”, on-demand analysis for a user query
Main local method:
relevance feedback
 - Part 1
- Global: Do a global analysis once (e.g., of collection) to produce thesaurus
Use thesaurus for query expansion
 - Part 2



Options for improving recall

- Local: Do a “local”, on-demand analysis for a user query
Main local method:
relevance feedback
 - Part 1
- Global: Do a global analysis once (e.g., of collection) to produce thesaurus
Use thesaurus for query expansion
 - Part 2



Relevance Feedback: Basics



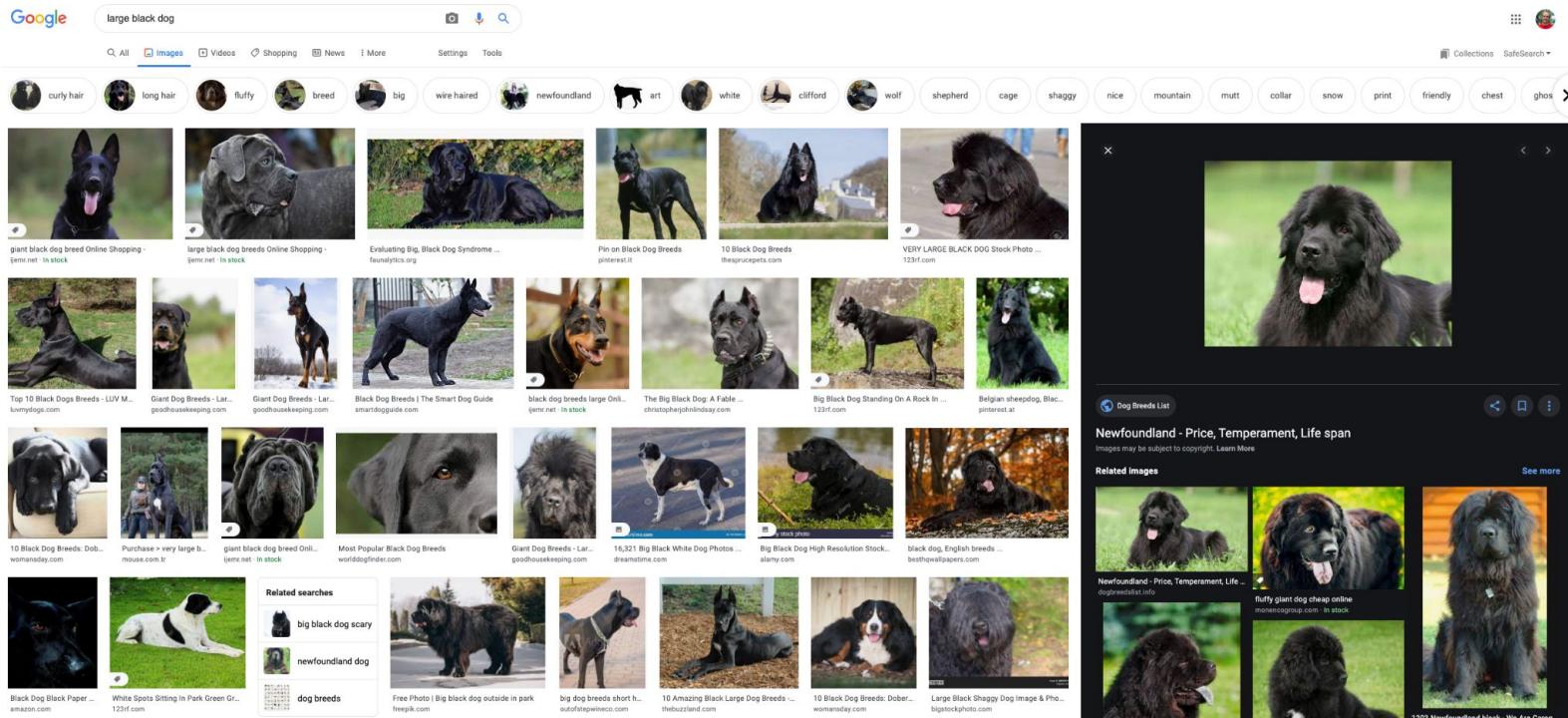
SAPIENZA
UNIVERSITÀ DI ROMA

Relevance feedback: Basic idea

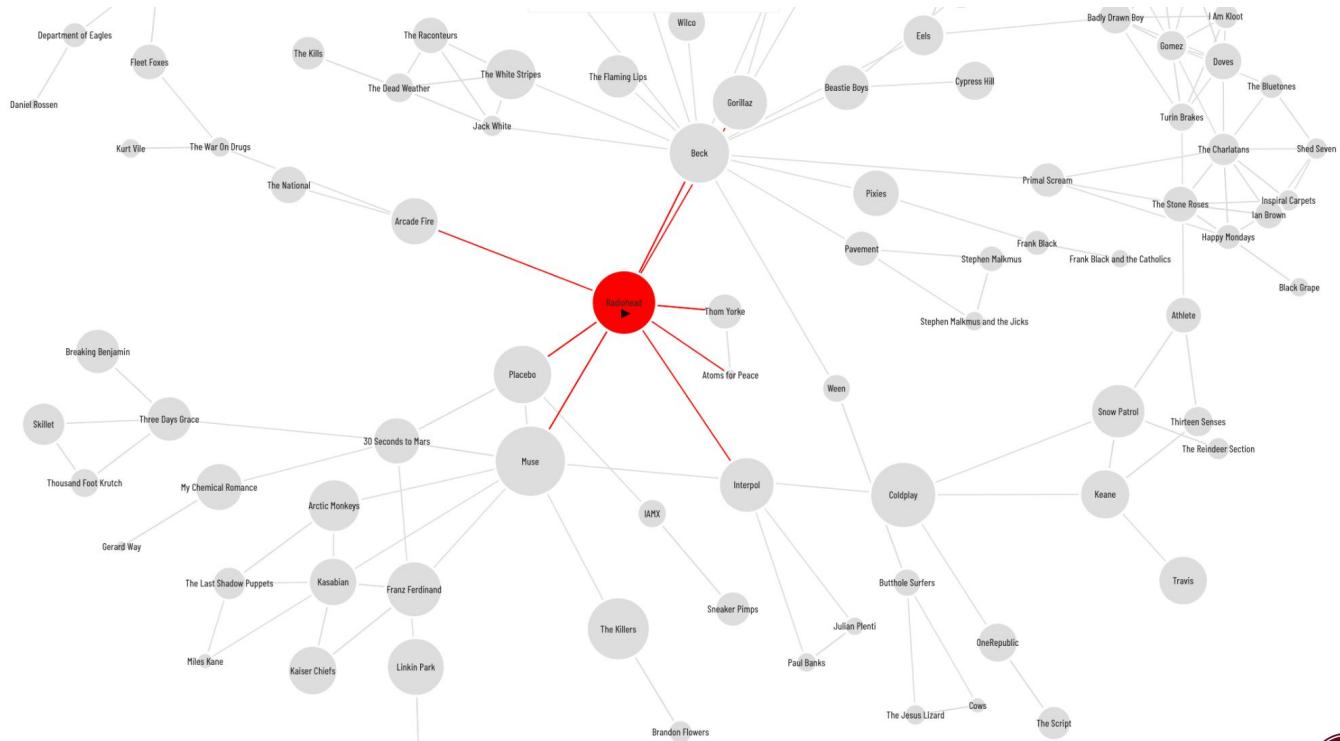
- The user issues a (short, simple) query.
- The search engine returns a set of documents.
- User marks some docs as relevant, some as nonrelevant.
- Search engine computes a new representation of the information need. Hope: better than the initial query.
- Search engine runs new query and returns new results.
- New results have (hopefully) better recall.
- We will use the term ad hoc retrieval to refer to regular retrieval without relevance feedback.



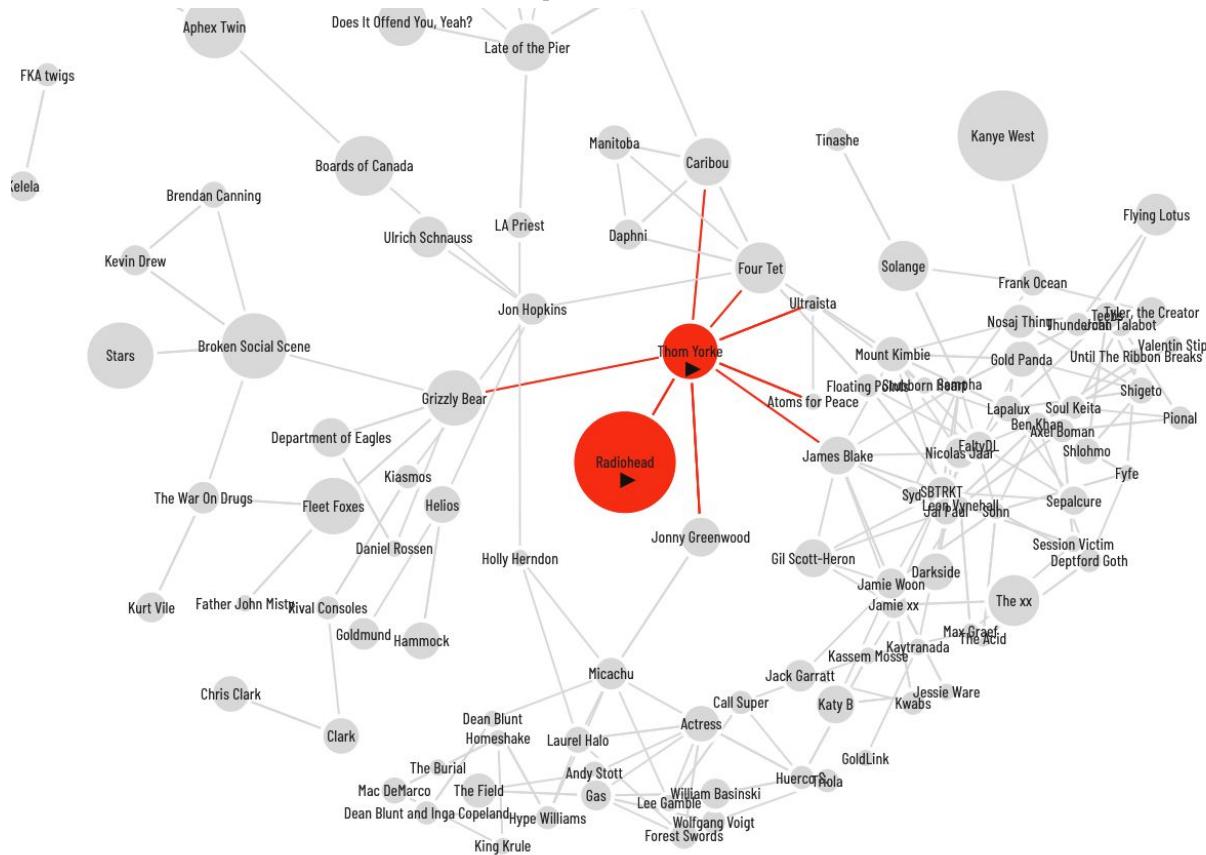
Relevance feedback: Example



Relevance feedback: Liveplasma.com



Relevance feedback: Liveplasma.com



Relevance Feedback: Details



SAPIENZA
UNIVERSITÀ DI ROMA

Key Concepts: The Centroid

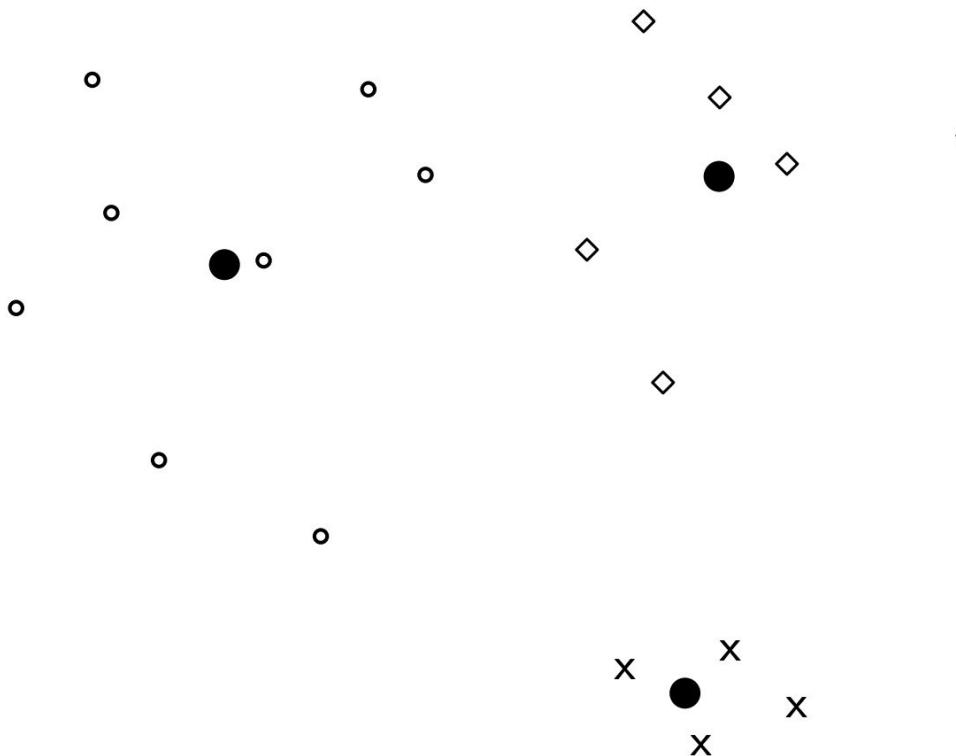
- The centroid is the center of mass of a set of points.
- Recall that we represent documents as points in a high-dimensional space.
- Thus: we can compute centroids of documents.
- Definition:

$$\vec{\mu}(D) = \frac{1}{|D|} \sum_{d \in D} \vec{v}(d)$$

- where D is a set of documents and v(d) is the vector we use to represent document d.



Centroids



SAPIENZA
UNIVERSITÀ DI ROMA

Rocchio

- The Rocchio algorithm implements relevance feedback in the vector space model.
- Rocchio chooses the query q_{opt} that maximizes

$$\vec{q}_{opt} = \arg \max_{\vec{q}} [\text{sim}(\vec{q}, \mu(D_r)) - \text{sim}(\vec{q}, \mu(D_{nr}))]$$

- D_r : set of relevant docs; D_{nr} : set of nonrelevant docs
- Intent: q_{opt} is the vector that separates relevant and nonrelevant docs maximally.
- Making some additional assumptions (sim is cosine similarity), we can rewrite q_{opt} as: $\vec{q}_{opt} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$



Optimal Query Vector

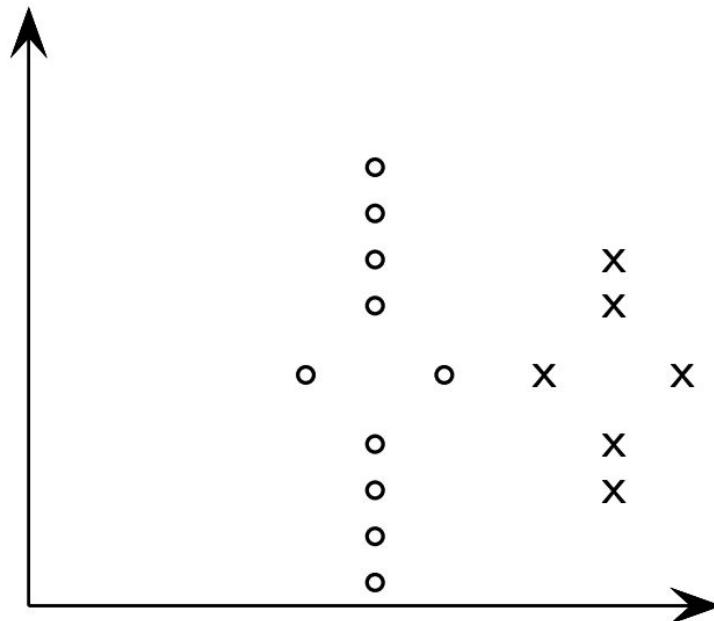
- The optimal query vector is:

$$\begin{aligned}\vec{q}_{opt} &= \mu(D_r) + [\mu(D_r) - \mu(D_{nr})] \\ &= \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j + \left[\frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j \right]\end{aligned}$$

- We move the centroid of the relevant documents by the difference between the two centroids.



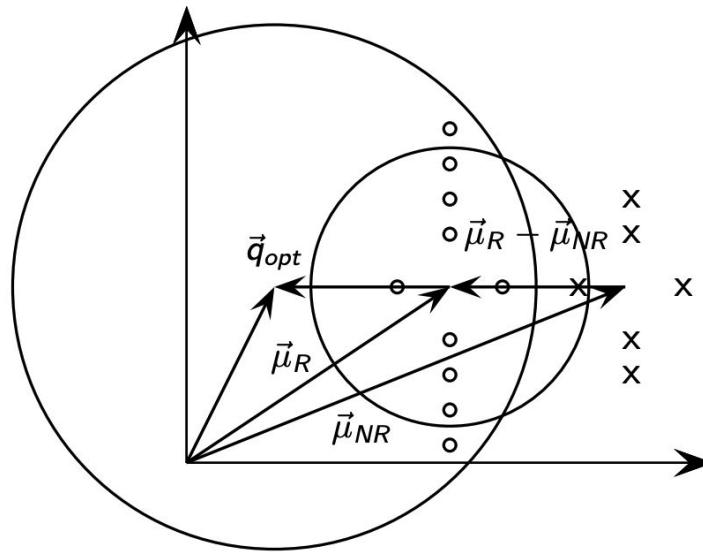
Example: Compute Rocchio Vector



- O's relevant docs, X's nonrelevant docs
- Compute $\vec{q}_{opt} = \mu(D_r) + [\mu(D_r) - \mu(D_{nr})]$



Example: Compute Rocchio Vector



- Os: relevant documents, Xs: nonrelevant documents
- $\mu(R)$: centroid of relevant documents $\mu(R)$ does not separate relevant/nonrelevant.
- $\mu(NR)$: centroid of nonrelevant documents; $\mu(R) - \mu(NR)$: difference vector
- Add difference vector to $\mu(R)$ to get q_{opt} , which separates relevant/nonrelevant perfectly.



Rocchio as it is usually implemented

- Used in practice:

$$\begin{aligned}\vec{q}_m &= \alpha \vec{q}_0 + \beta \mu(D_r) - \gamma \mu(D_{nr}) \\ &= \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j\end{aligned}$$

\vec{q}_m : modified query vector; \vec{q}_0 : original query vector; D_r and D_{nr} : sets of known relevant and nonrelevant documents respectively; α , β , and γ : weights

- New query moves towards relevant documents and away from nonrelevant documents.
- Tradeoff α vs. β/γ : If we have a lot of judged documents, we want a higher β/γ .
- Set negative term weights to 0.
- “Negative weight” for a term doesn’t make sense in the vector space model.

Rocchio as it is usually implemented

- Used in practice:

$$\begin{aligned}\vec{q}_m &= \alpha \vec{q}_0 + \beta \mu(D_r) - \gamma \mu(D_{nr}) \\ &= \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d}_j \in D_r} \vec{d}_j - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d}_j \in D_{nr}} \vec{d}_j\end{aligned}$$

\vec{q}_m : modified query vector; \vec{q}_0 : original query vector; D_r and D_{nr} : sets of known relevant and nonrelevant documents respectively; α , β , and γ : weights

- New query moves towards relevant documents and away from nonrelevant documents.
- Tradeoff α vs. β/γ : If we have a lot of judged documents, we want a higher β/γ .
- Set negative term weights to 0.
- “Negative weight” for a term doesn’t make sense in the vector space model.

Positive vs. Negative Feedback

- Positive feedback is more valuable than negative feedback.
- For example, set $\beta = 0.75$, $\gamma = 0.25$ to give higher weight to positive feedback.
- Many systems only allow positive feedback.



Relevance Feedback: Assumptions

- When can relevance feedback enhance recall?
- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Assumption A2: Relevant documents contain similar terms (so I can “hop” from one relevant document to a different one when giving relevance feedback).



Violations of A1

- Assumption A1: The user knows the terms in the collection well enough for an initial query.
- Violation: Mismatch of searcher's vocabulary and collection vocabulary
- Example: cosmonaut / astronaut



Violations of A2

- Assumption A2: Relevant documents are similar.
- Example for violation: [contradictory government policies]
- Several unrelated “prototypes”
 - Subsidies for tobacco farmers vs. anti-smoking campaigns
 - Aid for developing countries vs. high tariffs on imports from developing countries
- Relevance feedback on tobacco docs will not help with finding docs on developing countries.



Evaluating Relevance Feedback

- Pick an evaluation measure, e.g., precision in top 10: P@10
- Compute P@10 for original query q0
- Compute P@10 for modified relevance feedback query q1
- In most cases: q1 is spectacularly better than q0!
- Is this a fair evaluation?



Evaluating Relevance Feedback

- Fair evaluation must be on “residual” collection: docs not yet judged by user.
- Studies have shown that relevance feedback is successful when evaluated this way.
- Empirically, one round of relevance feedback is often very useful. Two rounds are marginally useful.



Evaluating Relevance: Caveat Emptor

- True evaluation of usefulness must compare to other methods taking the same amount of time.
- Alternative to relevance feedback: User revises and resubmits query.
- Users may prefer revision/resubmission to having to judge relevance of documents.
- There is no clear evidence that relevance feedback is the “best use” of the user’s time.



Issues with Relevance Feedback

- Relevance feedback is expensive.
- Relevance feedback creates long modified queries.
- Long queries are expensive to process.
- Users are reluctant to provide explicit feedback.
- It's often hard to understand why a particular document was retrieved after applying relevance feedback.



Pseudo Relevance Feedback

- Pseudo-relevance feedback automates the “manual” part of true relevance feedback.
- Pseudo-relevance feedback algorithm:
 - Retrieve a ranked list of hits for the user’s query
 - Assume that the top k documents are relevant.
 - Do relevance feedback (e.g., Rocchio)
- Works very well on average
- But can go horribly wrong for some queries.
- Because of **query drift**
 - If you do several iterations of pseudo-relevance feedback, then
 - you will get query drift for a large proportion of queries.



Query Expansion



SAPIENZA
UNIVERSITÀ DI ROMA

What is Query Expansion

 **palm plant**

 **palm hxh**

 **palm phone 2**

 **palm pda**

 **palm device**

 **palm meaning**



What is Query Expansion

- Query expansion is another method for increasing recall.
- We use “global query expansion” to refer to “global methods for query reformulation”.
- In global query expansion, the query is modified based on some global resource, i.e. a resource that is not query-dependent.
- Main information we use: (near-)synonymy



“Global” resources used for query expansion

- A publication or database that collects (near-)synonyms is called a thesaurus.
- Manual thesaurus (maintained by editors, e.g., PubMed)
- Automatically derived thesaurus (e.g., based on co-occurrence statistics)
- Query-equivalence based on query log mining (common on the web as in the “palm” example)



Thesaurus-based Query Expansion

- For each term t in the query, expand the query with words the thesaurus lists as semantically related with t .
- Example from earlier: hospital → medical
- Generally increases recall
- May significantly decrease precision, particularly with ambiguous terms
 - interest rate → interest rate fascinate
- Widely used in specialized search engines for science and engineering
- It's very expensive to create a manual thesaurus and to maintain it over time.



Automatic Thesaurus Generation

- Attempt to generate a thesaurus automatically by analyzing the distribution of words in documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are **similar if they co-occur with similar words.**
 - “car” ≈ “motorcycle” because both occur with “road”, “gas” and “license”, so they must be similar.
- Definition 2: Two words are **similar if they occur in a given grammatical relation with the same words.**
 - You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence is more robust, grammatical relations are more accurate.



Co-occurrence-based thesaurus: Examples

Word	Nearest neighbors
absolutely	absurd whatsoever totally exactly nothing
bottomed	dip copper drops topped slide trimmed
captivating	shimmer stunningly superbly plucky witty
doghouse	dog porch crawling beside downstairs
makeup	repellent lotion glossy sunscreen skin gel
mediating	reconciliation negotiate case conciliation
keeping	hoping bring wiping could some would
lithographs	drawings Picasso Dali sculptures Gauguin
pathogens	toxins bacteria organisms bacterial parasite
senses	grasp psyche truly clumsy naive innate



Query Expansion at Search Engines

- Main source of query expansion at search engines: query logs
- Example 1: After issuing the query [herbs], users frequently search for [herbal remedies].
 - → “herbal remedies” is potential expansion of “herb”.
- Example 2: Users searching for [flower pix] frequently click on the URL photobucket.com/flower. Users searching for [flower clipart] frequently click on the same URL.
 - → “flower clipart” and “flower pix” are potential expansions of each other.



Main Takeaways

- [Interactive relevance feedback](#): improve initial retrieval results by telling the IR system which docs are relevant / nonrelevant
- Best known relevance feedback method: [Rocchio feedback](#)
- [Query expansion](#): improve retrieval results by adding synonyms / related terms to the query
- [Sources for related terms](#): Manual thesauri, automatic thesauri, query logs



Clustering

IIR secs 16



SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Recap and Quick Intro



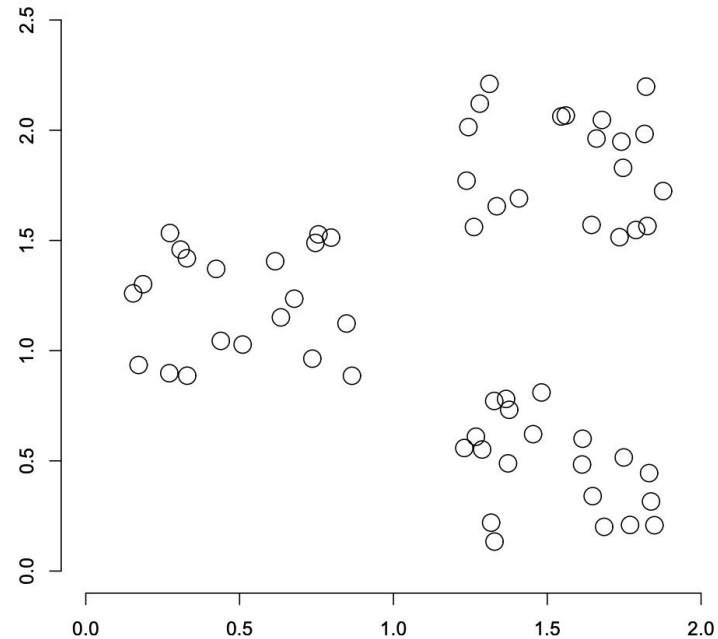
SAPIENZA
UNIVERSITÀ DI ROMA

Learning to Rank

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.
- In principle, any method learning a classifier (including least squares regression) can be used to find this line.
- Big advantage of learning to rank: we can avoid hand-tuning scoring functions and simply learn them from training data.
- Bottleneck of learning to rank: the cost of maintaining a representative set of training examples whose relevance assessments must be made by humans.



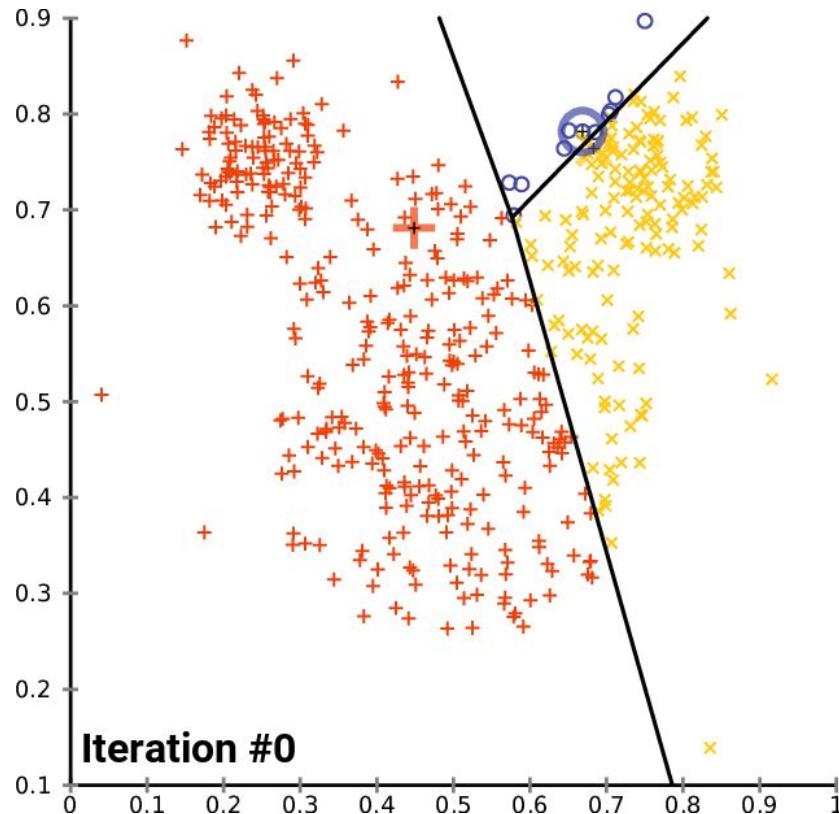
An Example



Propose
algorithm
for finding
the cluster
structure in
this
example



What's Clustering?



Clustering



SAPIENZA
UNIVERSITÀ DI ROMA

Clustering: Definitions

- The problem of making a binary relevant/nonrelevant judgment is cast as a classification or regression problem, based on a training set of query-document pairs and associated relevance judgments.
- In principle, any method learning a classifier (including least squares regression) can be used to find this line.
- Big advantage of learning to rank: we can avoid hand-tuning scoring functions and simply learn them from training data.
- Bottleneck of learning to rank: the cost of maintaining a representative set of training examples whose relevance assessments must be made by humans.



Classification Vs. Clustering

- **Classification**: supervised learning
- **Clustering**: unsupervised learning
- **Classification**: Classes are human-defined and part of the input to the learning algorithm.
- **Clustering**: Clusters are inferred from the data without human input.

However, there are many ways of influencing the outcome of clustering: number of clusters, similarity measure, representation of documents, . . .



Clustering in IR



SAPIENZA
UNIVERSITÀ DI ROMA

Classification Vs. Clustering

Cluster hypothesis.

Documents in the same cluster behave similarly with respect to relevance to information needs.

All applications of clustering in IR are based (directly or indirectly) on the cluster hypothesis. Van Rijsbergen's original wording (1979): "closely associated documents tend to be relevant to the same requests".



Applications in IR

application	what is clustered?	benefit
search result clustering	search results	more effective information presentation to user
Scatter-Gather	(subsets of) collection	alternative user interface: “search without typing”
collection clustering	collection	effective information presentation for exploratory browsing
cluster-based retrieval	collection	higher efficiency: faster search



Applications in IR

Google cat

All Images Videos Maps News More Settings Tools

kitten anime baby wallpaper white fluffy drawing kawaii tabby black beautiful orange

Romeow Cat Bistro Roma ...
facebook.com

Van cat - Wikipedia
en.wikipedia.org

Thinking of getting a cat ...
icatcare.org

5 things that scare and stress your c...
timesofindia.indiatimes.com

Coronavirus: Cat owners fear pets will ...
bbc.com

Cat infected with COVID-19 from owner ...
livescience.com

The cat's m...
humanesoc...



Applications in IR

dmoz About Become an Editor Suggest a Site Help Login

Important Notice
Welcome to our archive of dmoz.org.

Visit resource-zone to stay in touch with the community.

#OrganizeTheWeb

+ Search DMOZ

 Arts Movies, Television, Music...	 Business Jobs, Real Estate, Investing...	 Computers Internet, Software, Hardware...
 Games Video Games, RPGs, Gambling...	 Health Fitness, Medicine, Alternative...	 Home Family, Consumers, Cooking...
 News Media, Newspapers, Weather...	 Recreation Travel, Food, Outdoors, Humor...	 Reference Maps, Education, Libraries...
 Regional US, Canada, UK, Europe...	 Science Biology, Psychology, Physics...	 Shopping Clothing, Food, Gifts...
 Society People, Religion, Issues...	 Sports Baseball, Soccer, Basketball...	 Kids & Teens Directory Arts, School Time, Teen Life...
 DMOZ around the World Deutsch, Français, 日本語, Italiano, Español, Русский, Nederlands, Polski, Türkçe, Dansk, 简体中文, ...		



How do you build directories?

- DMOZ is manually built...
- Can you do it automatically?
 - E.g. Google News

The screenshot shows the Google News interface. At the top, there's a navigation bar with 'Top stories' (highlighted in blue), 'For you', 'Following', and 'Saved searches'. Below this is a search bar and a 'COVID-19' filter. On the left, a sidebar lists categories: U.S. (highlighted with a yellow box), World, Your local news, Business, Technology, Entertainment, Sports, Science, and Health.

The main content area is titled 'Headlines' and features a section for 'COVID-19 news'. It includes a summary: 'Biden Backs Suspending Patents on Covid Vaccines' (The New York Times, 1 hour ago) and several bullet points about vaccine patent waivers from Pfizer, Moderna, and US companies. There's a link to 'View Full Coverage'.

Below this is another news item: 'Jim Jordan: 'The votes are there to oust Liz Cheney from House GOP post'' (New York Post, 10 hours ago). It includes a bullet point about Cheney's ouster and a video thumbnail of Jim Jordan.

To the right of the news feed, there's a 'More Headlines' section and a weather forecast for Rome. The weather is sunny at 69°F. It shows forecasts for the next five days: Today (70°F, 55°F), Fri (72°F, 55°F), Sat (76°F, 55°F), Sun (77°F, 55°F), and Mon (79°F, 59°F). There's also a 'Fact check' section with a link to FactCheck.org about a meme featuring DeSantis.

At the bottom, there's a note: 'COVID-19 vaccines do not make people' followed by the Sapienza University logo.

Can Clustering Improve Recall?

- To improve search recall:
 - Cluster docs in collection a priori
- When a query matches a doc d, also return other docs in the cluster containing d
- Hope: if we do this: the query “car” will also return docs containing “automobile”
- Because the clustering algorithm groups together docs containing “car” with those containing “automobile”.
- Both types of documents contain words like “parts”, “dealer”, “mercedes”, “road trip”.



Let's go Deeper



SAPIENZA
UNIVERSITÀ DI ROMA

What clustering should do?

- General goal: put related docs in the same cluster, put unrelated docs in different clusters.
 - We'll see different ways of formalizing this.
- The number of clusters should be appropriate for the data set we are clustering.
 - Initially, we will assume the number of clusters K is given.
 - Later: Semiautomatic methods for determining K
- Secondary goals in clustering
 - Avoid very small and very large clusters
 - Define clusters that are easy to explain to the user
 - Many others . . .



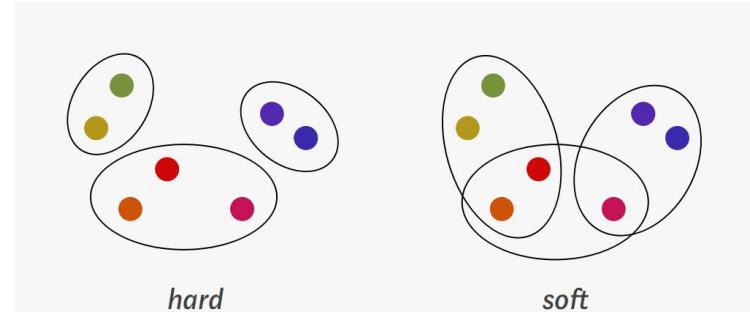
Flat vs. Hierarchical Clustering

- Flat algorithms
 - Usually start with a random (partial) partitioning of docs into groups
 - Refine iteratively
 - Main algorithm: K-means
- Hierarchical algorithms
 - Create a hierarchy
 - Bottom-up, agglomerative
 - Top-down, divisive



Hard vs. Soft clustering

- Hard clustering: Each document belongs to exactly one cluster.
 - More common and easier to do
- Soft clustering: A document can belong to more than one cluster.
 - Makes more sense for applications like creating browsable hierarchies
 - You may want to put sneakers in two clusters: sports apparel, and shoes
 - You can only do that with a soft clustering approach.



Flat Clustering

- Flat algorithms compute a partition of N documents into a set of K clusters.
- Given: a set of documents and the number K
- Find: a partition into K clusters that optimizes the chosen partitioning criterion
- Global optimization: exhaustively enumerate partitions, pick optimal one
 - Not tractable
- Effective heuristic method: K-means algorithm



Axioms for Clustering

- Invariance
 - Clustering should not depend on how we measure distances (e.g., feet, metres, inches, etc.)
- Richness
 - Clustering induces a partition on the set of objects we are partitioning. A good clustering algorithm should not rule out any partition a-priori.
- Consistency
 - Once objects are partitioned into clusters, reducing the distance between objects in a cluster or increasing the distance between objects in different clusters should not impact the clustering result



Impossibility Theorem for Clustering

- John Kleinberg. “An Impossibility Theorem for Clustering”. NIPS 2015

Theorem 2.1 *For each $n \geq 2$, there is no clustering function f that satisfies Scale-Invariance, Richness, and Consistency.*



K-Means



SAPIENZA
UNIVERSITÀ DI ROMA

What's K-Means?

- Perhaps the best known clustering algorithm
- Simple, works well in many cases
- Use as default / baseline for clustering documents



How are documents represented?

- Vector space model
- As in vector space classification, we measure relatedness between vectors by Euclidean distance . . .
 - . . . which is “almost” equivalent to cosine similarity.
- Almost: centroids are not length-normalized.



K-Means: Basic Idea

- Each cluster in K-means is defined by a centroid.
- Objective/partitioning criterion: minimize the average squared difference from the centroid
- Recall definition of centroid:

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega}$$

cluster

- We try to find the minimum average squared difference by iterating two steps:
 - reassignment: assign each vector to its closest centroid
 - recomputation: recompute each centroid as the average of the vectors that were assigned to it in reassignment



K-Means: pseudocode (μ_k is centroid of ω_k)

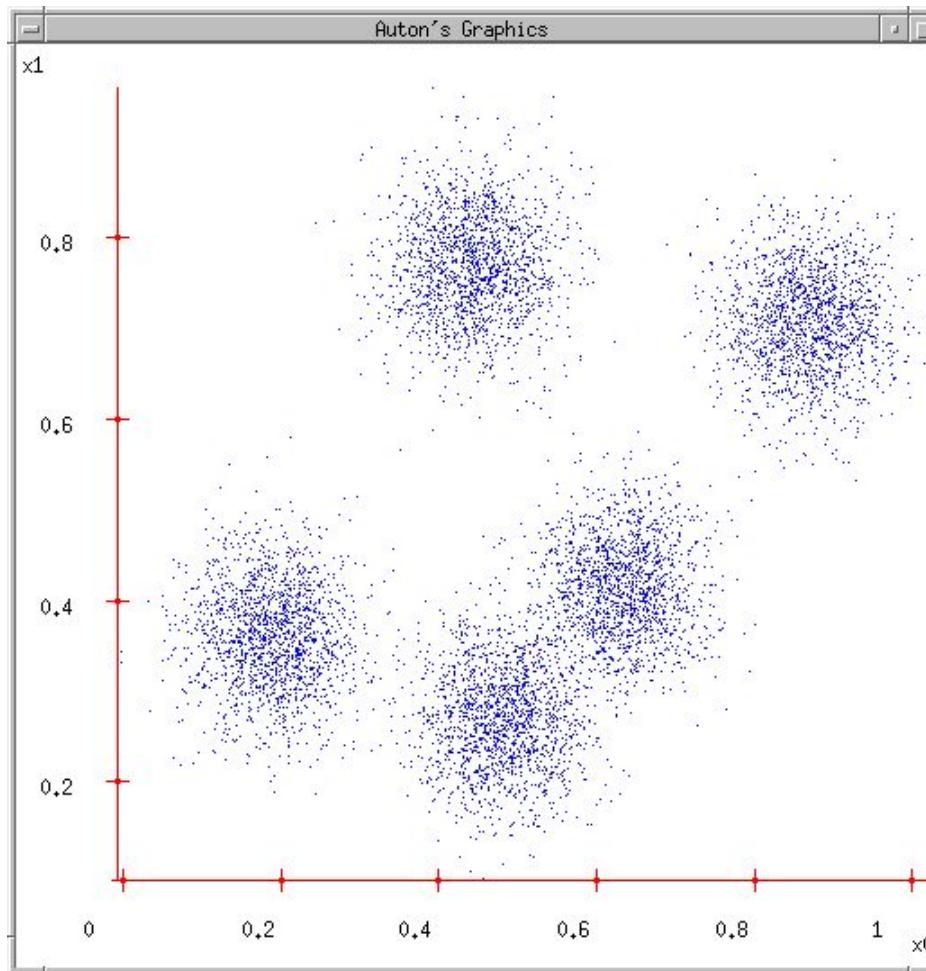
K-MEANS($\{\vec{x}_1, \dots, \vec{x}_N\}$, K)

```
1   $(\vec{s}_1, \vec{s}_2, \dots, \vec{s}_K) \leftarrow \text{SELECTRANDOMSEEDS}(\{\vec{x}_1, \dots, \vec{x}_N\}, K)$ 
2  for  $k \leftarrow 1$  to  $K$ 
3  do  $\vec{\mu}_k \leftarrow \vec{s}_k$ 
4  while stopping criterion has not been met
5  do for  $k \leftarrow 1$  to  $K$ 
6    do  $\omega_k \leftarrow \{\}$ 
7    for  $n \leftarrow 1$  to  $N$ 
8    do  $j \leftarrow \arg \min_{j'} |\vec{\mu}_{j'} - \vec{x}_n|$ 
9     $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$  (reassignment of vectors)
10   for  $k \leftarrow 1$  to  $K$ 
11   do  $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$  (recomputation of centroids)
12 return  $\{\vec{\mu}_1, \dots, \vec{\mu}_K\}$ 
```



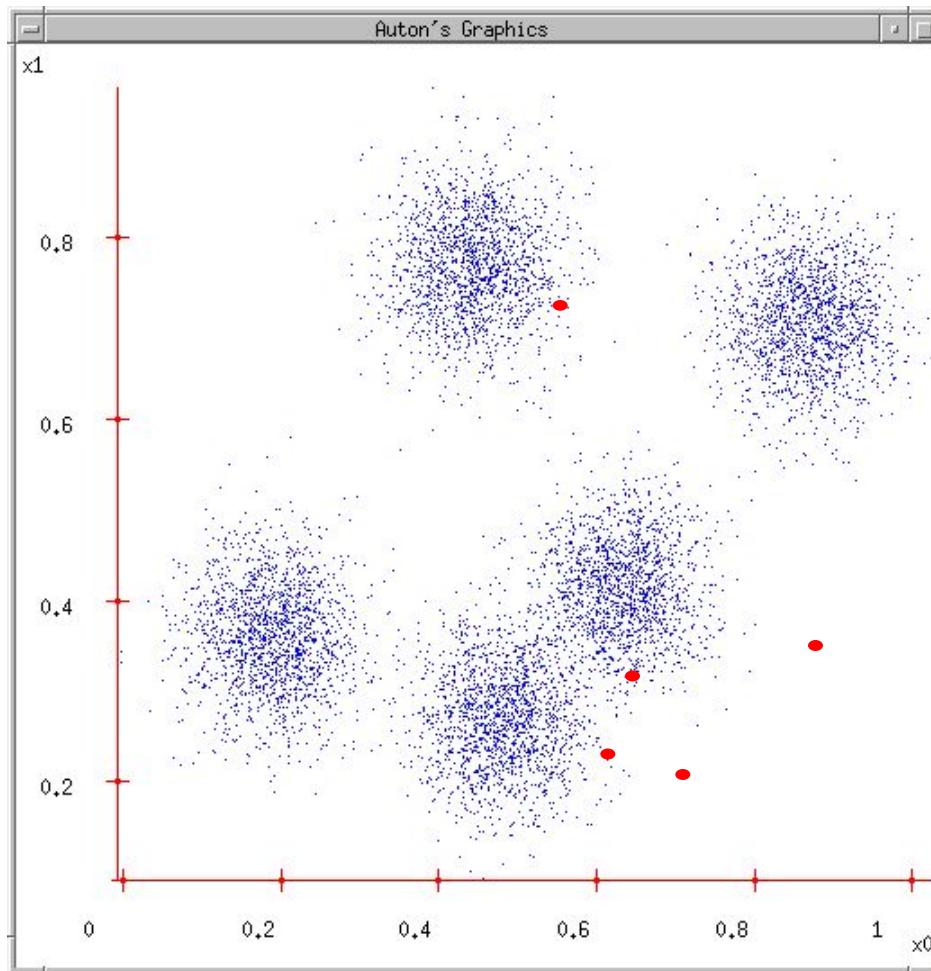
K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)



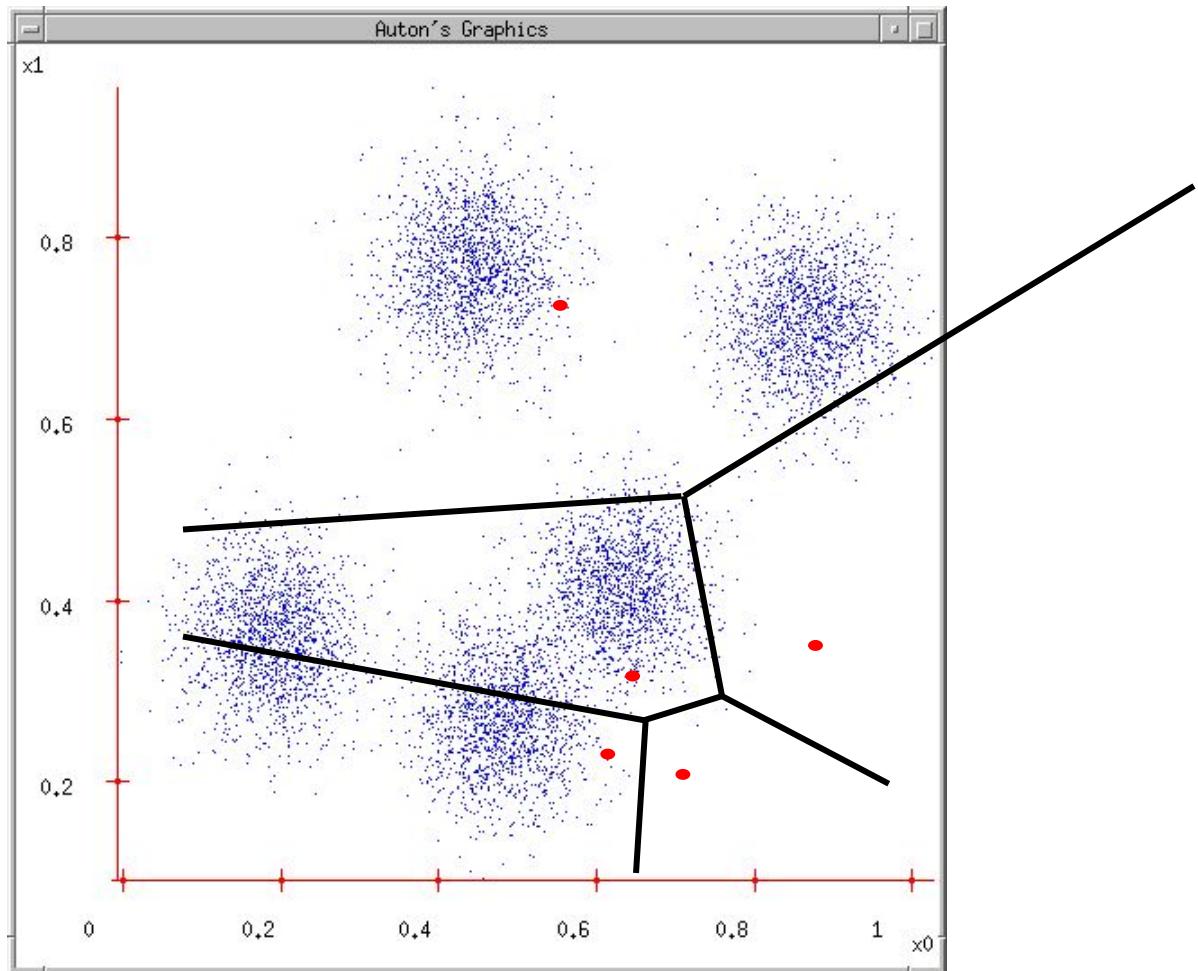
K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)
2. Randomly guess k cluster Center locations



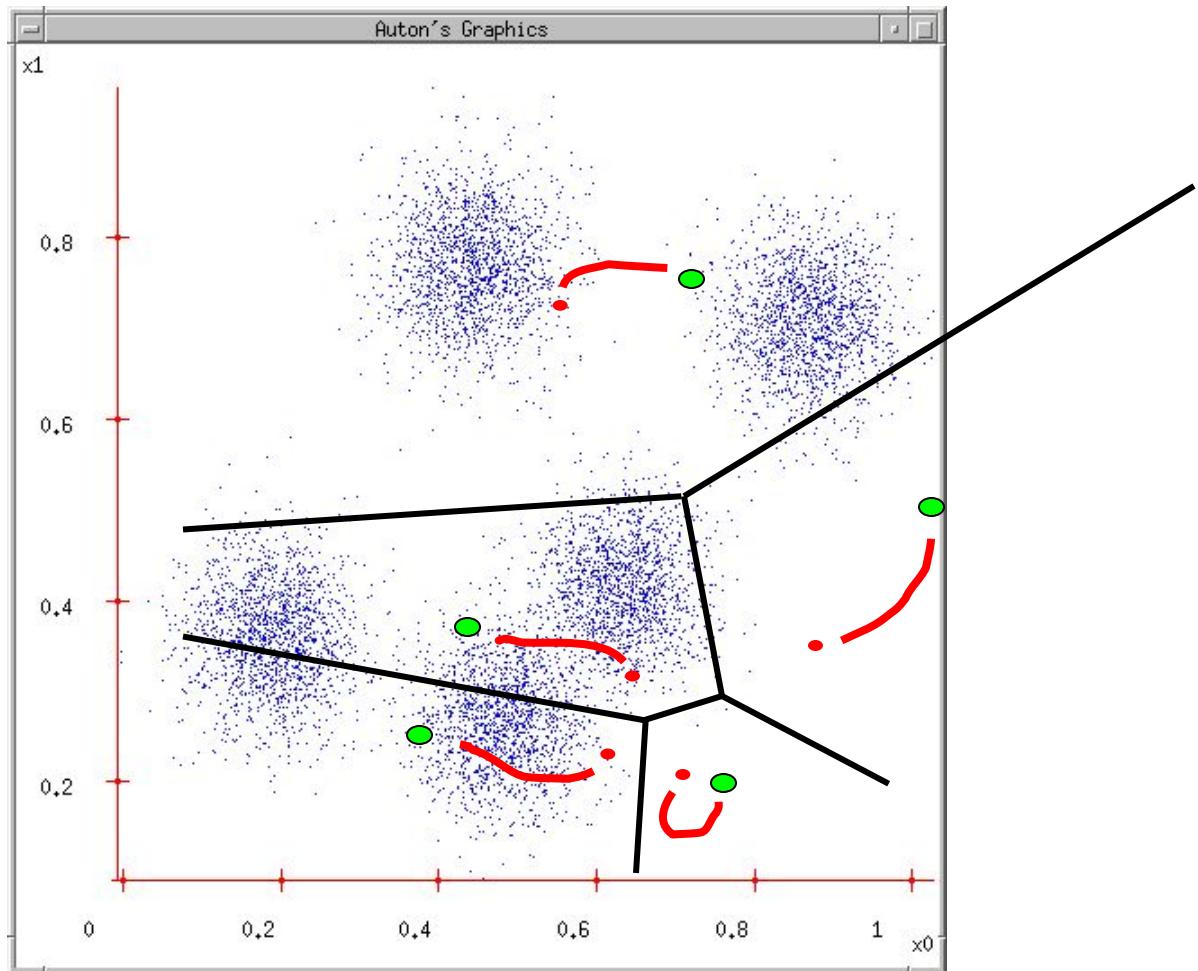
K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



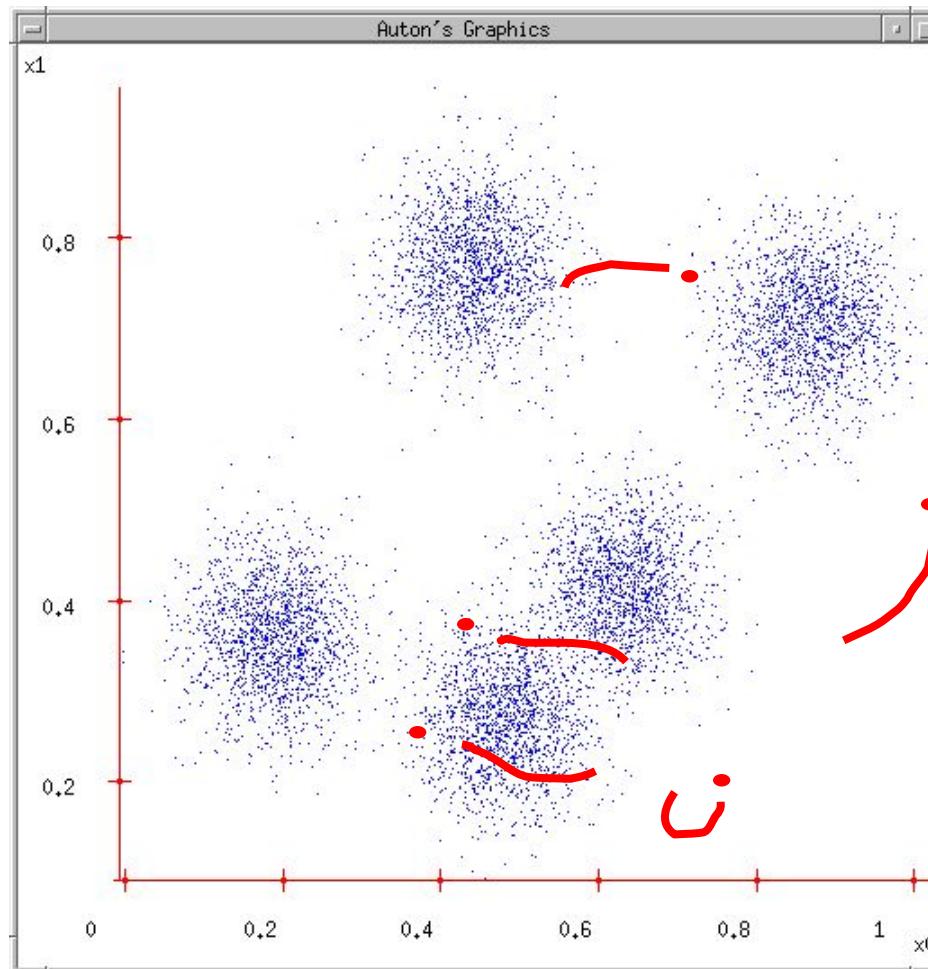
K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

1. Ask user how many clusters they'd like. (e.g. $k=5$)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!



K-Means is guaranteed to converge

- RSS = sum of all squared distances between document vector and closest centroid
- RSS decreases during each reassignment step.
 - because each vector is moved to a closer centroid
- RSS decreases during each recomputation step.
 - see next slide
- There is only a finite number of clusterings.
- Thus: We must reach a fixed point.
- Assumption: Ties are broken consistently.
- Finite set & monotonically decreasing → convergence



Recomputation decreases average distance

$\text{RSS} = \sum_{k=1}^K \text{RSS}_k$ – the residual sum of squares (the “goodness” measure)

$$\text{RSS}_k(\vec{v}) = \sum_{\vec{x} \in \omega_k} \|\vec{v} - \vec{x}\|^2 = \sum_{\vec{x} \in \omega_k} \sum_{m=1}^M (v_m - x_m)^2$$

$$\frac{\partial \text{RSS}_k(\vec{v})}{\partial v_m} = \sum_{\vec{x} \in \omega_k} 2(v_m - x_m) = 0$$

$$v_m = \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} x_m$$

The last line is the componentwise definition of the centroid! We minimize RSS_k when the old centroid is replaced with the new centroid. RSS, the sum of the RSS_k , must then also decrease during recomputation.



K-Means is guaranteed to converge, but ...

- ... we don't know how long convergence will take!
 - If we don't care about a few docs switching back and forth, then convergence is usually fast (< 10-20 iterations).
 - However, complete convergence can take many more iterations.



Optimality of K-Means

- Convergence \neq optimality
- Convergence does not mean that we converge to the optimal clustering!
- This is the great weakness of K-means.
- If we start with a bad set of seeds, the resulting clustering can be horrible.



Initialization of centroids

- Random seed selection is just one of many ways K-means can be initialized.
- Random seed selection is not very robust: It's easy to get a suboptimal clustering.
- Better ways of computing initial centroids:
 - Select seeds not randomly, but using some heuristic (e.g., filter out outliers or find a set of seeds that has “good coverage” of the document space)
 - Use hierarchical clustering to find good seeds
 - Select i (e.g., $i = 10$) different random sets of seeds, do a K-means clustering for each, select the clustering with lowest RSS



Complexity of K-Means

- Computing one distance of two vectors is $O(M)$.
- Reassignment step: $O(KNM)$ (we need to compute KN document-centroid distances)
- Recomputation step: $O(NM)$ (we need to add each of the document's $< M$ values to one of the centroids)
- Assume number of iterations bounded by I
- Overall complexity: $O(IKNM)$ – linear in all important dimensions
- However: This is not a real worst-case analysis.
 - In pathological cases, complexity can be worse than linear.



Evaluation



SAPIENZA
UNIVERSITÀ DI ROMA

What is a good clustering?

- Internal criteria
 - Example of an internal criterion: RSS in K-means
- But an internal criterion often does not evaluate the actual utility of a clustering in the application.
- Alternative: External criteria
 - Evaluate with respect to a human-defined classification



External Criteria for Clustering Evaluation

- Based on a gold standard data set, e.g., the Reuters collection we also used for the evaluation of classification
- Goal: Clustering should reproduce the classes in the gold standard
- (But we only want to reproduce how documents are divided into groups, not the class labels.)
- First measure for how well we were able to reproduce the classes: **purity**

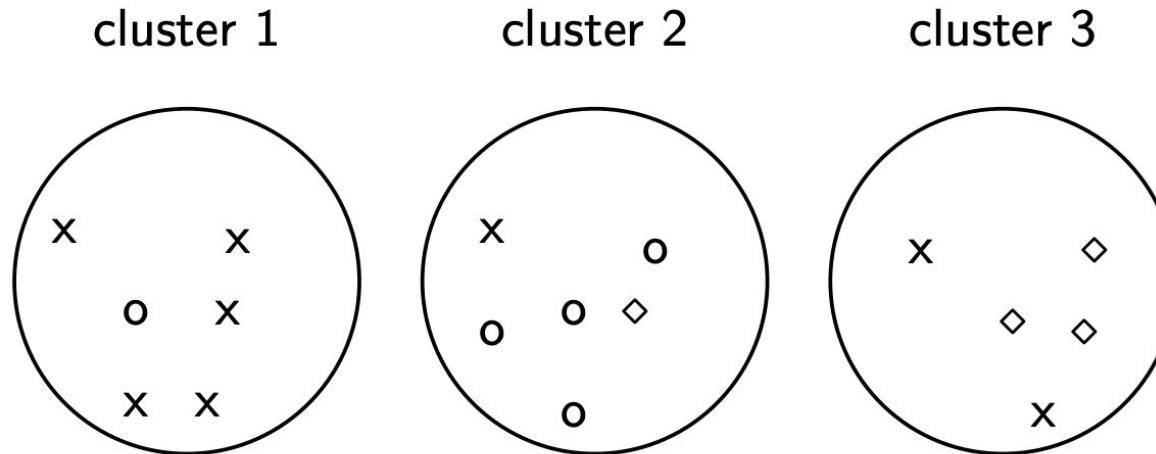


External Criteria: Purity

- $\Omega = \{\omega_1, \omega_2, \dots, \omega_K\}$ is the set of clusters and $C = \{c_1, c_2, \dots, c_J\}$ is the set of classes.
- For each cluster ω_k : find class c_j with most members n_{kj} in ω_k
- Sum all n_{kj} and divide by total number of points



Computing Purity: An Example



To compute

purity: $5 = \max_j |\omega_1 \cap c_j|$ (class x, cluster 1); $4 = \max_j |\omega_2 \cap c_j|$ (class o, cluster 2); and $3 = \max_j |\omega_3 \cap c_j|$ (class \diamond , cluster 3).
Purity is $(1/17) \times (5 + 4 + 3) \approx 0.71$.



External Criteria: Rand

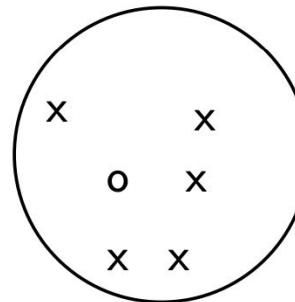
- Purity can be increased easily by increasing K – a measure that does not have this problem: [Rand index](#).
- Definition: $RI = \frac{TP+TN}{TP+FP+FN+TN}$
- Based on 2x2 contingency table of all pairs of documents:

	same cluster	different clusters
same class	true positives (TP)	false negatives (FN)
different classes	false positives (FP)	true negatives (TN)

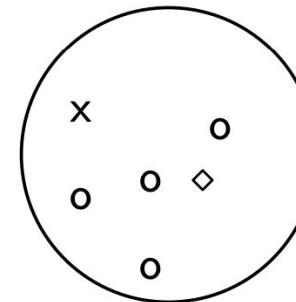
- $TP+FN+FP+TN$ is the total number of pairs.
- $TP+FN+FP+TN = \text{choose}(N, 2)$ for N documents
-
- Each pair is either positive or negative (the clustering puts the two documents in the same or in different clusters) . . .
 - . . . and either “true” (correct) or “false” (incorrect): the clustering decision is correct or incorrect

Rand Index: Example

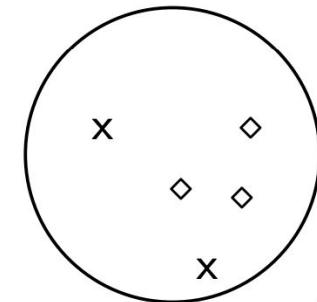
cluster 1



cluster 2



cluster 3



As an example, we compute RI for the o/◊/x example. We first compute TP + FP. The three clusters contain 6, 6, and 5 points, respectively, so the total number of “positives” or pairs of documents that are in the same cluster is:

$$TP + FP = \binom{6}{2} + \binom{6}{2} + \binom{5}{2} = 40$$

Of these, the x pairs in cluster 1, the o pairs in cluster 2, the ◊ pairs in cluster 3, and the x pair in cluster 3 are true positives:

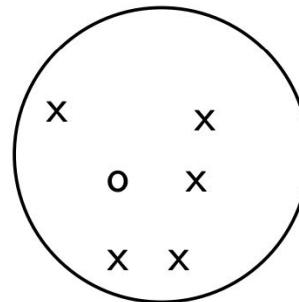
$$TP = \binom{5}{2} + \binom{4}{2} + \binom{3}{2} + \binom{2}{2} = 20$$

Thus, $FP = 40 - 20 = 20$. FN and TN are computed similarly

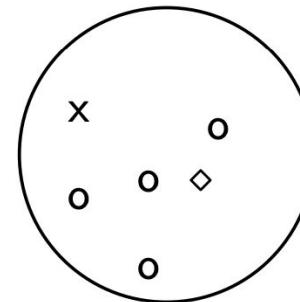


Rand Index: Example

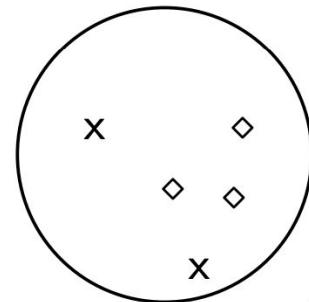
cluster 1



cluster 2



cluster 3



	same cluster	different clusters	
same class	TP = 20	FN = 24	RI is then
different classes	FP = 20	TN = 72	

$$(20 + 72) / (20 + 20 + 24 + 72) \approx 0.68.$$



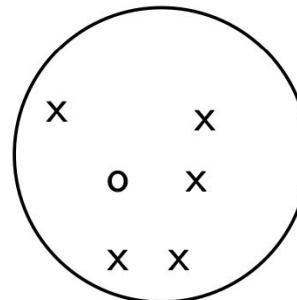
Two other external evaluation measures

- Normalized mutual information (NMI)
 - How much information does the clustering contain about the classification?
 - Singleton clusters (number of clusters = number of docs) have maximum MI
 - Therefore: normalize by entropy of clusters and classes
- F measure
 - Like Rand, but “precision” and “recall” can be weighted

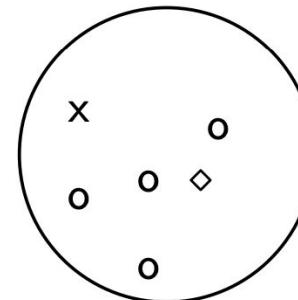


Evaluation

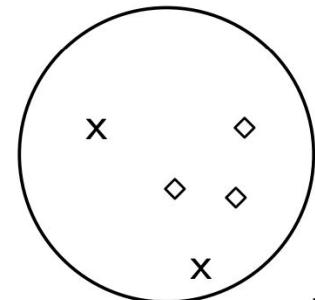
cluster 1



cluster 2



cluster 3



	purity	NMI	RI	F_5
lower bound	0.0	0.0	0.0	0.0
maximum	1.0	1.0	1.0	1.0
value for example	0.71	0.36	0.68	0.46

All four measures range from 0 (really bad clustering) to 1 (perfect clustering).



How many clusters



SAPIENZA
UNIVERSITÀ DI ROMA

How many clusters?

- Number of clusters K is given in many applications.
 - E.g., there may be an external constraint on K . Example: In the case of clustering of results 10 clusters are enough.
- What if there is no external constraint? Is there a “right” number of clusters?
- One way to go: define an optimization criterion
 - Given docs, find K for which the optimum is reached.
 - What optimization criterion can we use?
 - We can't use RSS or average squared distance from centroid
 - as criterion: always chooses $K = N$ clusters.



How many clusters?

- Start with 1 cluster ($K = 1$)
- Keep adding clusters (= keep increasing K)
- Add a penalty for each new cluster
- Then trade off cluster penalties against average squared distance from centroid
- Choose the value of K with the best tradeoff

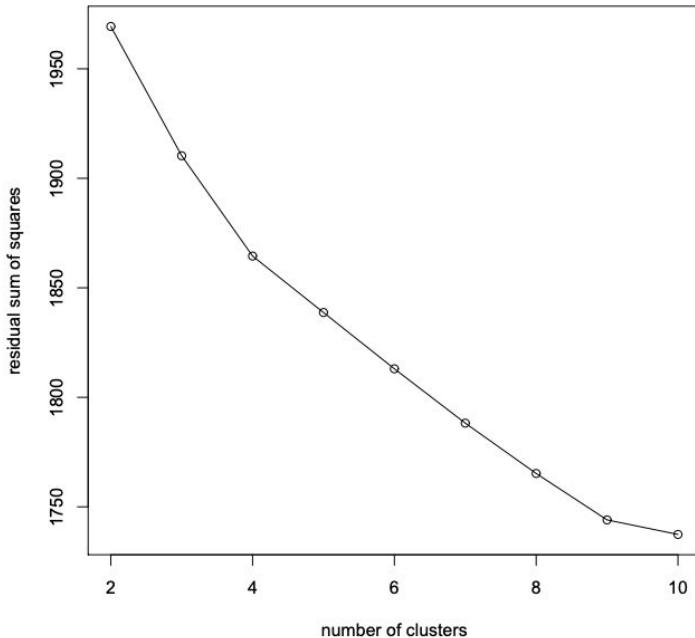


How many clusters?

- Given a clustering, define the cost for a document as (squared) distance to centroid
- Define total distortion $RSS(K)$ as sum of all individual document costs (corresponds to average distance)
- Then: penalize each cluster with a cost λ
- Thus for a clustering with K clusters, total cluster penalty is $K\lambda$
- Define the total cost of a clustering as distortion plus total cluster penalty: $RSS(K) + K\lambda$
- Select K that minimizes $(RSS(K) + K\lambda)$
- Still need to determine good value for λ . . .



Finding the knee on the curve



Pick the number of clusters where curve “flattens”. Here: 4 or 9.



Takeaway Messages

- What is clustering?
- Applications of clustering in information retrieval
- K-means algorithm
- Evaluation of clustering
- How many clusters?



MLR: Machine Learning Ranking

IIR secs 6.1.2–3 and 15.4



SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Quick Intro



SAPIENZA
UNIVERSITÀ DI ROMA

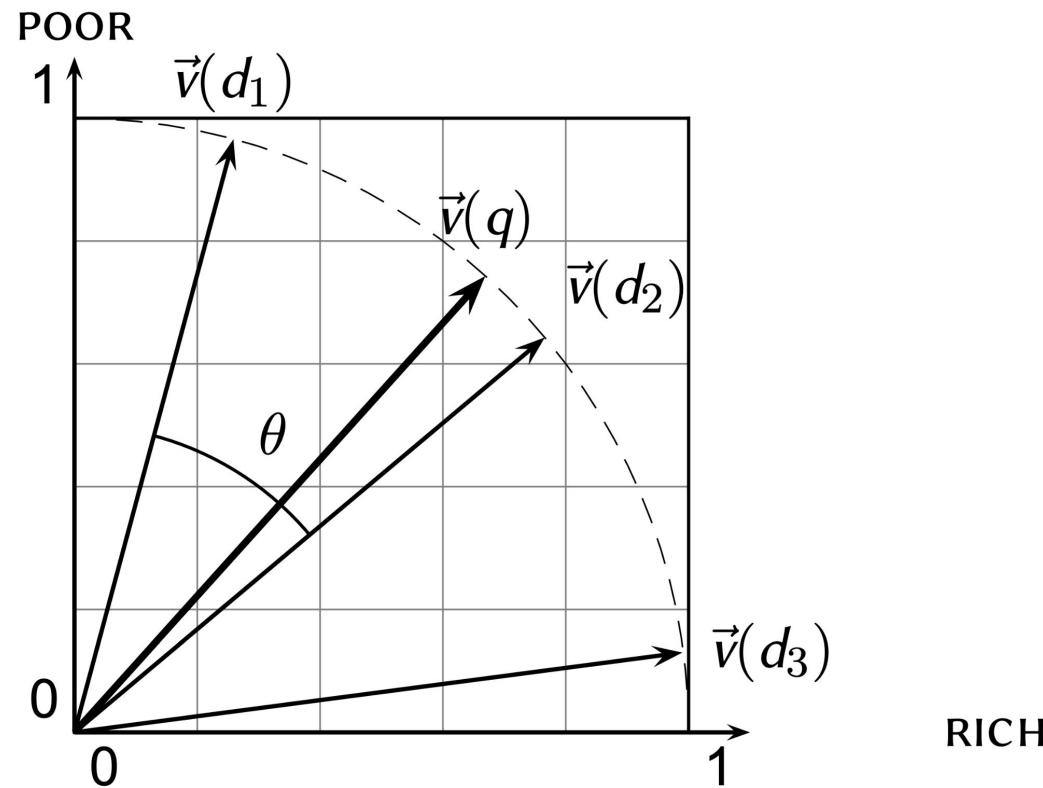
Cosine Similarity between Query and Document

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

- 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 vectors \vec{q} and \vec{d} , respectively.
- $\vec{q}/|\vec{q}|$ and $\vec{d}/|\vec{d}|$ are length-1 vectors (= normalized)



Computing Scores



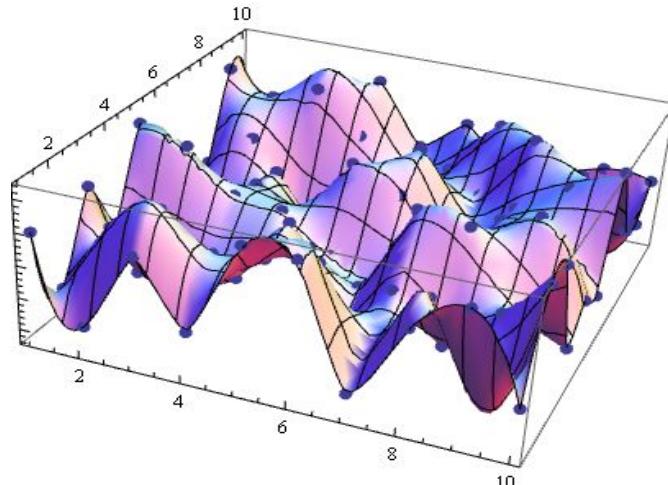
Other factors affecting scores

- Query-Click pairs
- Different sections of a document have different importance
- Date, time of query affects score
- Location of the searcher
- Demographic information
- History of queries
- History of clicks
- History of visited pages
- ...



Why cosine similarity?

- BM25 (a refined form of cosine similarity) arises from well defined principles.
- But... score = $f(\text{query}, \text{document})$
 - What is query? And document?
 - What is the functional form of f ?
- Interpolation



Can we do more?



SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning to Rank Results

- We've looked at methods for ranking documents in IR
 - Cosine similarity, inverse document frequency, BM25, proximity, pivoted document length normalization, (will look at) Pagerank, ...
- We've looked at methods for classifying documents using supervised machine learning classifiers
 - Naive Bayes.
- Surely we can also use machine learning to rank the documents displayed in search results?
 - Sounds like a good idea
 - Known as “machine-learned relevance” or “learning to rank”





Senior Software Engineer, Machine Learning

Google · Mountain View, CA

Posted 6 days ago · 585 views

...
...

Apply

Save

See how you compare to 48 applicants

Try Premium Free for 1 Month

Job	Company	Connections
<ul style="list-style-type: none">48 applicantsFull-time	<ul style="list-style-type: none">1000+ employeesInternet	 103 connections  20 company alumni

Note: By applying to this position your application is automatically submitted to the following locations: **Mountain View, CA, USA; Los Angeles, CA, USA**

Minimum qualifications:

- Bachelor's degree or equivalent practical experience.
- 7 years of software development experience, or 5 years with an advanced degree.
- Experience in applied machine learning or artificial intelligence.
- Experience with one or more general purpose programming languages including but not limited to: Java, C/C++, or Python.

Preferred qualifications:

- Master's or PhD degree in Computer Science, Artificial Intelligence, Machine Learning, or related technical field.
- Experience with one or more of the following: Natural Language Processing, text understanding, classification, pattern recognition, recommendation systems, targeting systems, ranking systems or similar.
- Experience with relevant technologies (e.g., Tensorflow, Flume, machine learning libraries).
- Relevant professional experience with applied data analytics and predictive modeling.
- Ability to speak and write in English fluently and idiomatically.



SAPIENZA
UNIVERSITÀ DI ROMA

Major Search Engines (even FB's one) use MLR

Embedding-based Retrieval in Facebook Search

Jui-Ting Huang
juiting@fb.com
Facebook Inc.

Li Xia
xiali824@fb.com
Facebook Inc.

Janani Padmanabhan
jananip@fb.com
Facebook Inc.

Ashish Sharma
ashishsharma@fb.com
Facebook Inc.

David Zhang
shihaoz@fb.com
Facebook Inc.

Giuseppe Ottaviano
ott@fb.com
Facebook Inc.

Shuying Sun
shuyingsun@fb.com
Facebook Inc.

Philip Pronin
philipp@fb.com
Facebook Inc.

Linjun Yang^{*}
yang.linjun@microsoft.com
Microsoft



SAPIENZA
UNIVERSITÀ DI ROMA



Official Blog

Insights from Googlers into our products,
technology, and the Google culture

Introduction to Google Search Quality

May 20, 2008

Posted by Udi Manber, VP Engineering, Search Quality

Search Quality is the name of the team responsible for the ranking of Google search results. Our job is clear: A few hundreds of millions of times a day people will ask Google questions, and within a fraction of a second Google needs to decide which among the billions of pages on the web to show them – and in what order. Lately, we have been doing other things as well. But more on that later.

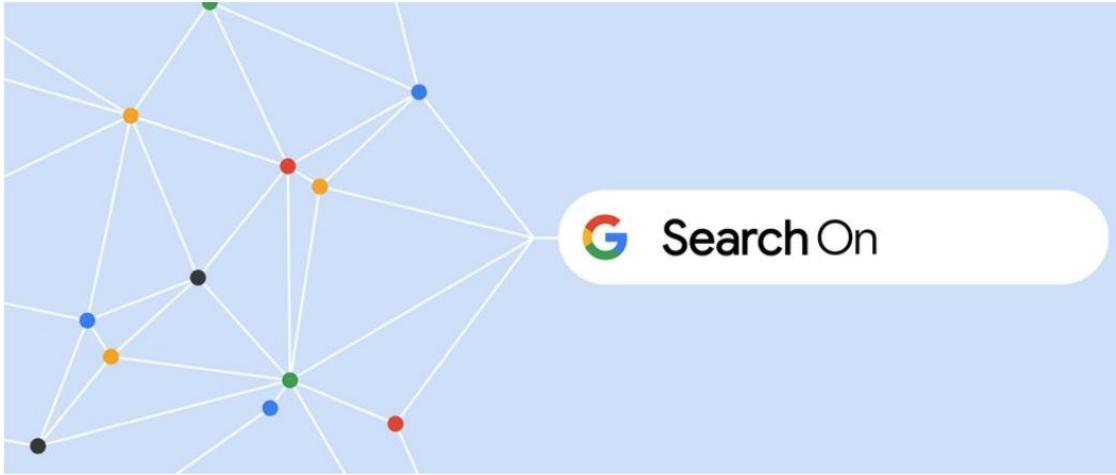
For something that is used so often by so many people, surprisingly little is known about ranking at Google. This is entirely our fault, and it is by design. We are, to be honest, quite secretive about what we do. There are two reasons for it: competition and abuse. Competition is pretty straightforward. No company wants to share its secret recipes with its competitors. As for abuse, if we make our ranking formulas too accessible, we make it easier for people to game the system. Security by obscurity is never the strongest measure, and we do not rely on it exclusively, but it does prevent a lot of abuse.

<https://googleblog.blogspot.com/2008/05/introduction-to-google-search-quality.html>



SAPIENZA
UNIVERSITÀ DI ROMA

How AI is powering a more helpful Google



Prabhakar Raghavan
Senior Vice President, Search & Assistant, Geo, Ads, Commerce, Payments & NBU

Published Oct 15, 2020

When I first came across the web as a computer scientist in the mid-90s, I was struck by the sheer volume of information online, in contrast with how hard it was to find what you were looking for. It was then that I first started thinking about search, and I've been fascinated by the problem ever since.

We've made tremendous progress over the past 22 years, making Google Search work better for you every day. With recent advancements in AI, we're making bigger leaps forward in improvements to Google than we've seen over the last decade, so it's even easier for you to find just what you're looking for. Today during our [Search On](#) livestream, we shared how we're bringing the most advanced AI into our products to further our mission to organize the world's information and make it universally accessible and useful.

<https://blog.google/products/search/search-on/>



SAPIENZA
UNIVERSITÀ DI ROMA

ML for Ranking in IR

- This “good idea” has been actively researched – and actively deployed by major web search engines – in the last 20 years
- Why didn’t it happen earlier?
- Modern supervised ML has been around for about 35 years...
- Naïve Bayes has been around for about 70 years...



Truth to be told...

- There's some truth to the fact that the IR community wasn't very connected to the ML community
- But there were a whole bunch of precursors:
 - Wong, S.K. et al. 1988. Linear structure in information retrieval. SIGIR 1988.
 - Fuhr, N. 1992. Probabilistic methods in information retrieval. Computer Journal.
 - Gey, F. C. 1994. Inferring probability of relevance using the method of logistic regression. SIGIR 1994.
 - Herbrich, R. et al. 2000. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers.



Why weren't early attempts very successful/influential?

- Sometimes an idea just takes time to be appreciated...
- Limited training data
 - Especially for real world use (as opposed to writing academic papers), it was very hard to gather test collection queries and relevance judgments that are representative of real user needs and judgments on documents returned
 - This has changed, both in academia and industry
- Poor machine learning techniques
- Insufficient customization to IR problem
- Not enough features for ML to show value



Why wasn't ML much needed?

- Traditional ranking functions in IR used a very small number of features, e.g.,
 - Term frequency
 - Inverse document frequency
 - Document length
- It was easy/possible to tune weighting coefficients by hand
 - And people did (Google's Amit Singhal)

While RankBrain was initially approved by Singhal, he was hesitant to more widely incorporate machine learning technology into Google Search due to the potential unpredictability and lack of transparency in how the AI actually functions.

<https://www.csmonitor.com/Technology/2016/0204/After-losing-a-pioneering-leader-where-is-Google-s-AI-search-going>



SAPIENZA
UNIVERSITÀ DI ROMA

... Nowadays

- Modern (web) systems 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 loading speed
- The New York Times in 2008-06-03 quoted Amit Singhal as saying Google was using over 200 such features (“signals”) – so it’s sure to be over 500 today.



Simple example: 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)
 - Query-Document pair is represented by a feature vector
 - $x = (\alpha, \omega) \rightarrow \alpha$ 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 an 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	<i>relevant</i>
Φ_2	37	penguin logo	0.02	4	<i>nonrelevant</i>
Φ_3	238	operating system	0.043	2	<i>relevant</i>
Φ_4	238	runtime environment	0.004	2	<i>nonrelevant</i>
Φ_5	1741	kernel layer	0.022	3	<i>relevant</i>
Φ_6	2094	device driver	0.03	2	<i>relevant</i>
Φ_7	3191	device driver	0.027	5	<i>nonrelevant</i>



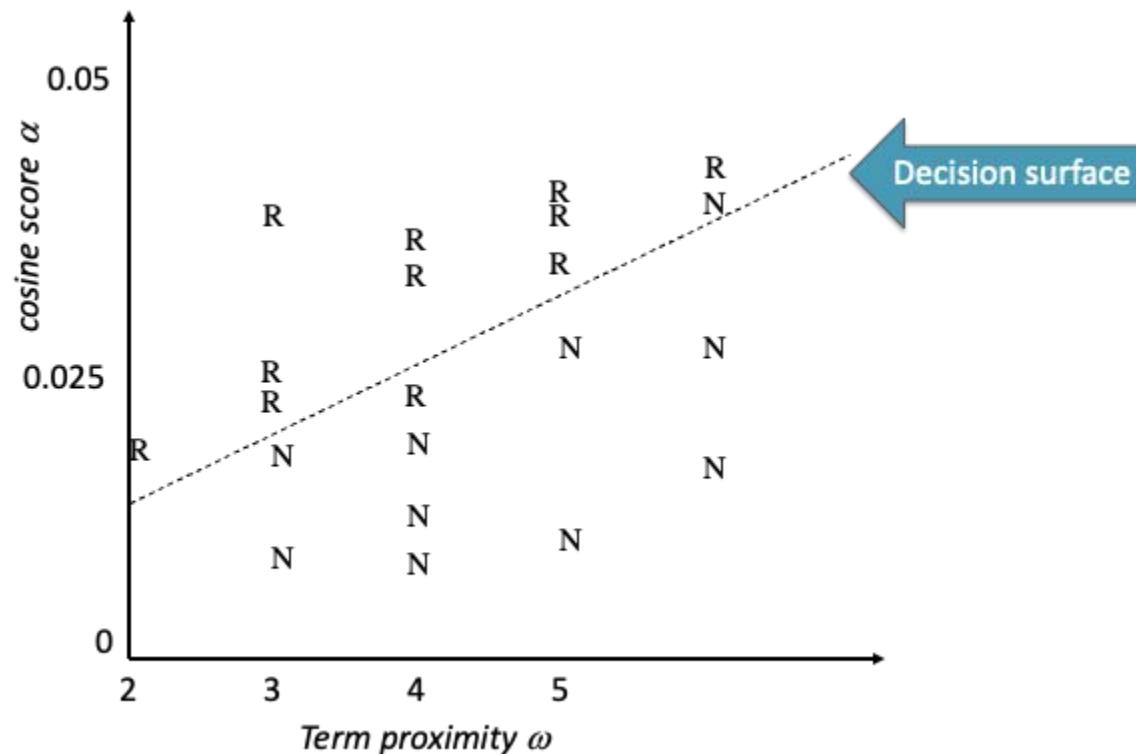
Simple example: Classification for ad hoc IR

- A linear score function is then
 - $\text{Score}(d, q) = \text{Score}(\alpha, \omega) = a\alpha + b\omega + c$
- And the linear classifier is
 - Decide relevant if $\text{Score}(d, q) > \theta$

... just like when we were doing text classification



Simple example: Classification for ad hoc IR



Using classification for search ranking

- We can generalize this to classifier functions over more features
- We can use other methods for learning the linear classifier weights



An SVM classifier for information retrieval

- Let relevance score $g(r|d,q) = wf(d,q) + b$
- Uses SVM: want $g(r|d,q) \leq -1$ for nonrelevant documents and $g(r|d,q) \geq 1$ for relevant documents
- SVM testing: decide relevant iff $g(r|d,q) \geq 0$
- Features are not word presence features (how would you deal with query words not in your training data?) but scores like the summed (log) tf of all query terms
- Unbalanced data (which can result in trivial always-say-nonrelevant classifiers) is dealt with by undersampling nonrelevant documents during training (just take some at random)



An SVM classifier for information retrieval

Train \ Test		Disk 3	Disk 4-5	WT10G (web)
TREC Disk 3	Lemur	0.1785	0.2503	0.2666
	SVM	0.1728	0.2432	0.2750
Disk 4-5	Lemur	0.1773	0.2516	0.2656
	SVM	0.1646	0.2355	0.2675

- At best the results are about equal to Lemur
 - Actually a little bit below
- Paper's advertisement: Easy to add more features
 - This is illustrated on a homepage finding task on WT10G:
 - Baseline Lemur 52% success@10, baseline SVM 58%
 - SVM with URL-depth, and in-link features: 78% success@10



Is it really learning to “rank”?

- Classification probably isn't the right way to think about approaching ad hoc IR:
 - Classification problems: Map to an unordered set of classes
 - Regression problems: Map to a real value
 - Ordinal regression (or “ranking”) problems: Map to an ordered set of classes
- This formulation gives extra power:
 - Relations between relevance levels are modeled
 - **Documents are good versus other documents for a query given collection;** not an absolute scale of goodness



Learning to Rank

- Assume a number of categories C of relevance exist
 - These are totally ordered: $c_1 < c_2 < \dots < c_J$
 - This is the ordinal regression setup
- Assume training data is available consisting of document-query pairs (d, q) represented as feature vectors x_i with relevance ranking c_i



Learning to Rank in Search

- Support Vector Machines (Vapnik, 1995)
 - Adapted to ranking: Ranking SVM (Joachims 2002)
- Neural Nets:
 - RankNet (Burges et al., 2006)
 - Unified Embedding Framework (Huang et al., 2020)
- Tree Ensembles
 - Random Forests (Breiman and Schapire, 2001)
 - Boosted Decision Trees
 - Multiple Additive Regression Trees (Friedman, 1999)
 - Gradient-boosted decision trees: LambdaMART (Burges, 2010)
 - Used by all search engines? AltaVista, Yahoo!, Bing, Yandex, Google, ...
- All top teams in the 2010 Yahoo! Learning to Rank Challenge used combinations with Tree Ensembles!



Yahoo! Learning to Rank Challenge

(Chapelle and Chang, 2011)

- Yahoo! Webscope dataset : 36,251 queries, 883k documents, 700 features, 5 ranking levels
 - Ratings: Perfect (navigational), Excellent, Good, Fair, Bad
 - Real web data from U.S. and “an Asian country”
 - set-1: 473,134 feature vectors; 519 features; 19,944 queries
 - set-2: 34,815 feature vectors; 596 features; 1,266 queries
- Winner (Burges et al.) was linear combo of 12 models:
 - 8 Tree Ensembles (LambdaMART)
 - 2 LambdaRank Neural Nets
 - 2 Logistic regression models



Regression Trees



SAPIENZA
UNIVERSITÀ DI ROMA

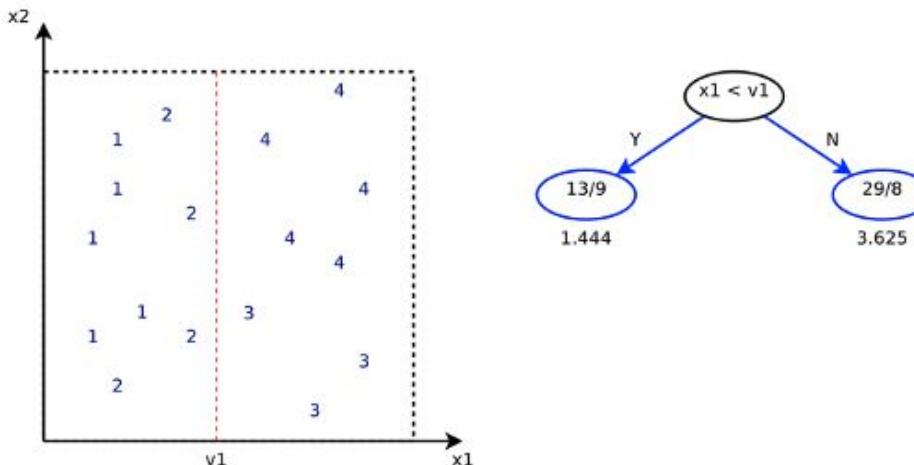
Regression trees

- Decision trees can predict a real value
 - They're then often called "regression trees"
- The value of a leaf node is the mean of all instances at the leaf
 - $\gamma_k = f(x_i) = \text{AVG}(x_i)$
- Splitting criterion: Standard Deviation Reduction
 - Choose split value to minimize the variance (standard deviation SD) of the values in each subset S_i of S induced by split A (normally just a binary split for easy search):
 - $SDR(A, S) = SD(S) - \sum_i \frac{|S_i|}{|S|} SD(S_i)$
 - $SD = \sum_i (y_i - f(x_i))^2$
- Termination: cutoff on SD or #examples or tree depth



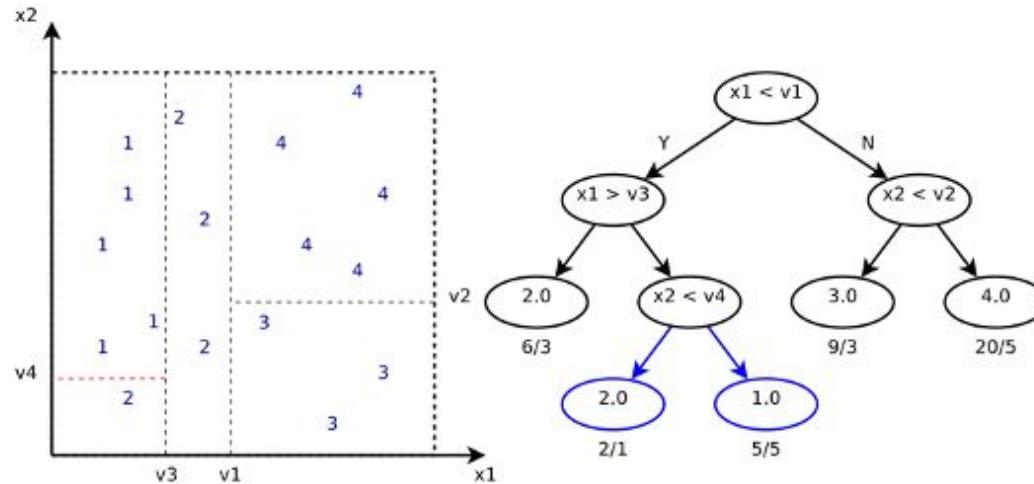
Training Regression trees

- The algorithm searches for split variables and split points, x_1 and v_1 so as to minimize the predicted error, i.e., $\sum_i (y_i - f(x_i))^2$



Training Regression trees

- You can grow tree till 0 error (if no identical points with different scores)



What's boosting, anyway?

- Motivating question:
 - Can we use individually weak machine learning classifiers to build a high-accuracy classification system?
- Classic approach (AdaBoost)
 - Learn a small decision tree (often a 1-split decision stump)
 - It will get the biggest split in the data right
 - Repeat:
 - Upweight examples it gets wrong;
 - Downweight examples it gets right
 - Learn another small decision tree on that reweighted data
- Classify with weighted vote of all trees
 - Weight trees by individual accuracy



Gradient Boosting

- Want: a function $F^*(x)$ that maps \mathbf{x} to y , s.t. the expected value of some loss function $L(y, F(\mathbf{x}))$ is minimized:
 - $F^*(x) = \operatorname{argmin}_{F(x)} \mathbb{E}_{(y,x)} L(y, F(\mathbf{x}))$
- Boosting approximates $F^*(x)$ by an additive expansion
 - $F(x) = \sum [\beta_m h(\mathbf{x}; a_m)]$ for m in $[1, M]$
- where $h(\mathbf{x}; a)$ are simple functions of \mathbf{x} with parameters $a = \{a_1, a_2, \dots, a_n\}$ defining the function h , and the β are weighting coefficients



Fitting Parameters

- Function parameters are iteratively fit to the training data:
 - Set $F_0(x) = \text{initial guess (or zero)}$
 - For each $m = 1, 2, \dots, M$
 - $a_m = \operatorname{argmin}_a \sum_i L(y_i, F_{(m-1)}(\mathbf{x}_i) + \beta h(\mathbf{x}_i, a))$
 - $F_m(x) = F_{(m-1)}(\mathbf{x}_i) + \beta h(\mathbf{x}_i, a_m)$
- You successively estimate and add a new tree to the sum
- You never go back to revisit past decisions



Fitting Parameters

- Gradient boosting approximately achieves this for any differentiable loss function
 - Fit the function $h(x; a)$ by least squares
 - $a_m = \operatorname{argmin}_a \sum_i [(\tilde{y}_{im} - h(x_i, a)]^2$
 - to the “pseudo-residuals” (deviation from desired scores)
 - $\tilde{y}_{im} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)}$
- Whatever the loss function, gradient boosting simplifies the problem to least squares estimation!!!
 - We can take a gradient (Newton) step to improve model



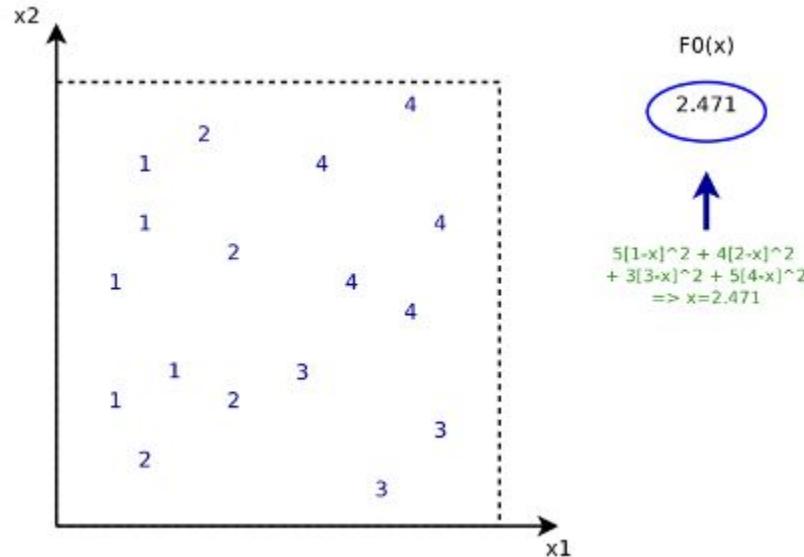
Gradient Tree Boosting

- Gradient tree boosting applies this approach on functions $h(x; a)$ which are small regression trees
 - The trees used normally have 1–8 splits only
 - Sometimes stumps do best!
 - The allowed depth of the tree controls the feature interaction order of model (do you allow feature pair conjunctions, feature triple conjunctions, etc.?)



Gradient Tree Boosting: Learning

- First, learn the simplest predictor that predicts a constant value that minimizes the error on the training data



Gradient Tree Boosting: Learning

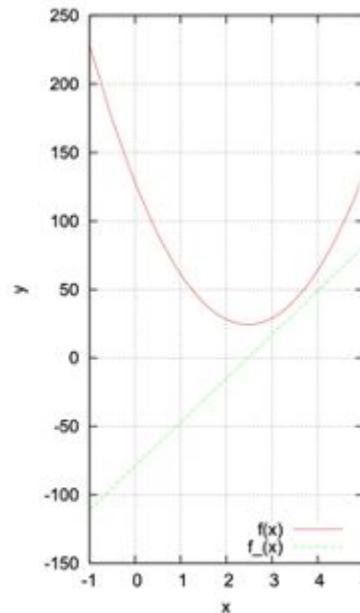
- We want to find value γ_{km} for root node of tree

Quadratic loss for the leaf (red):

$$\begin{aligned}f(x) = & 5 \cdot (1-x)^2 + 4 \cdot (2-x)^2 \\& + 3 \cdot (3-x)^2 + 5 \cdot (4-x)^2\end{aligned}$$

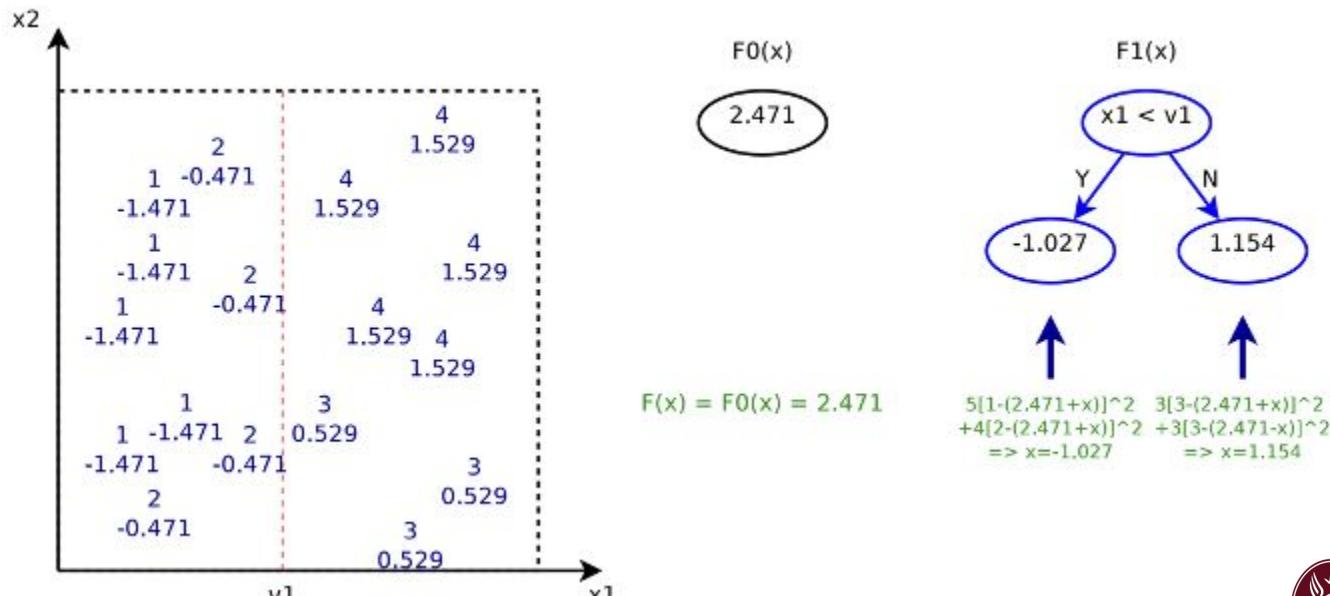
$f(x)$ is quadratic, *convex*
⇒ Optimum at $f'(x) = 0$ (green)

$$\begin{aligned}\frac{\partial f(x)}{\partial x} = & 5 \cdot (-2+2x) + 4 \cdot (-4+2x)^2 \\& + 3 \cdot (-6+2x)^2 + 5 \cdot (-8+2x)^2 \\= & -84 + 34x = 34(x - 2.471)\end{aligned}$$



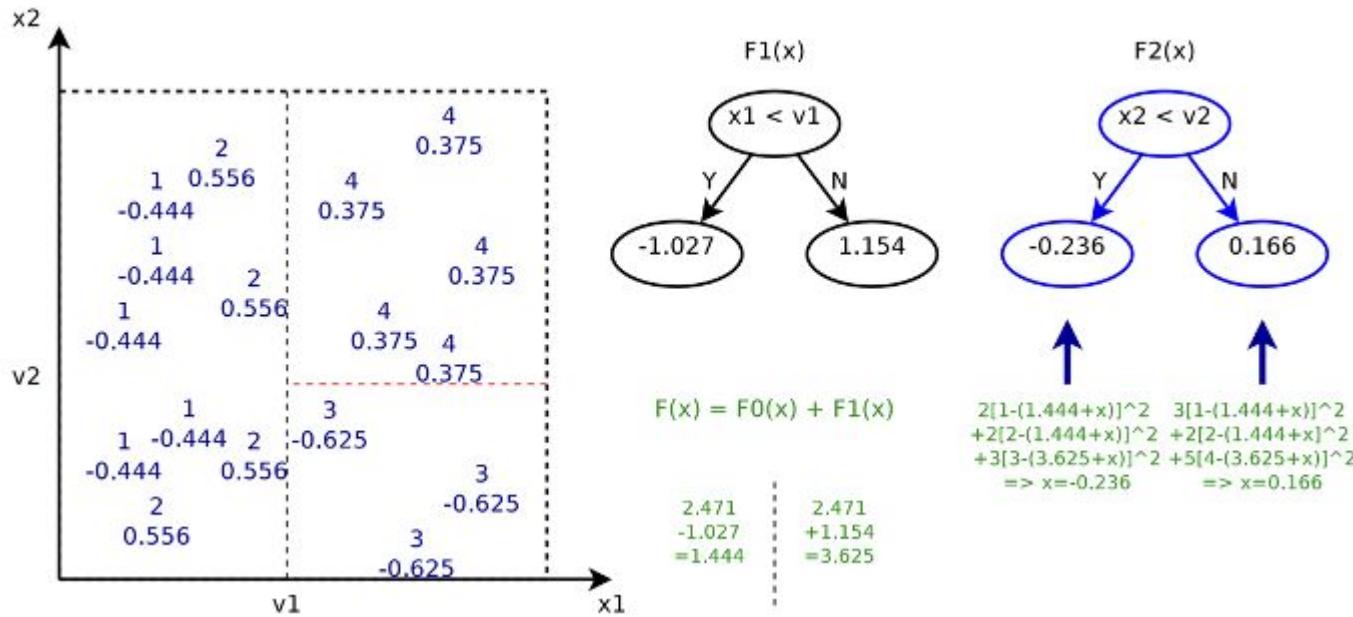
Gradient Tree Boosting: Learning

- We split root node based on least squares criterion and build a tree predicting “pseudo-residuals”



Gradient Tree Boosting: Learning

- Then another tree is added to fit the actual “pseudo-residuals” of the first tree



Multiple Additive Regression Trees (MART)

Algorithm 1 Multiple Additive Regression Trees.

```
1: Initialize  $F_0(\mathbf{x}) = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, \gamma)$ 
2: for  $m = 1, \dots, M$  do
3:   for  $i = 1, \dots, N$  do
4:      $\tilde{y}_{im} = - \left[ \frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)} \right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$ 
5:   end for
6:    $\{R_{km}\}_{k=1}^K$  // Fit a regression tree to targets  $\tilde{y}_{im}$ 
7:   for  $k = 1, \dots, K_m$  do
8:      $\gamma_{km} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$ 
9:   end for
10:   $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \eta \sum_{k=1}^{K_m} \gamma_{km} \mathbf{1}(\mathbf{x}_i \in R_{km})$ 
11: end for
12: Return  $F_M(\mathbf{x})$ 
```



Historical Path to LambdaMART via RankNet (NN)

- Have differentiable function with model parameters w:
 - $x_i \rightarrow f(x; w) = s_i$
- For query q, learn probability of different ranking class for documents $d_i > d_j$ via:
 - $P_{ij} = P(d_i > d_j) = 1/(1 + e^{(-\sigma(s_i - s_j))})$
- Cost function calculates cross entropy loss:
 - $C = -P_{ij} \log P_{ij} - (1 - P_{ij}) \log(1 - P_{ij})$
- Where \underline{P}_{ij} is the model probability; \overline{P}_{ij} the actual probability (0 or 1 for categorical judgments)



Historical Path to LambdaMART via RankNet (NN)

- Combining these equations gives
- $C = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log(1 + e^{(-\sigma(s_i - s_j))})$
- where, for a given query, $S_{ij} \in \{0, +1, -1\}$
1 if d_i is more relevant than d_j ; -1 if the reverse, and 0 if they have the same label

$$\frac{\partial C}{\partial s_i} = \sigma \left(\frac{1}{2} (1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}} \right) = -\frac{\partial C}{\partial s_j}$$

$$\begin{aligned}\frac{\partial C}{\partial w_k} &= \frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k} = \sigma \left(\frac{1}{2} (1 - S_{ij}) - \frac{1}{1 + e^{\sigma(s_i - s_j)}} \right) \left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k} \right) \\ &= \lambda_{ij} \left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k} \right)\end{aligned}$$



Historical Path to LambdaMART via RankNet (NN)

- The crucial part of the update is

$$\frac{\partial C}{\partial w_k} = \frac{\partial C}{\partial s_i} \frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j} \frac{\partial s_j}{\partial w_k} = \lambda_{ij} \left(\frac{\partial s_i}{\partial w_k} - \frac{\partial s_j}{\partial w_k} \right)$$

- λ_{ij} describes the desired change of scores for the pair of documents d_i and d_j
- The sum of all λ_{ij} 's and λ_{ji} 's of a query-doc vector x_i w.r.t. all other differently labelled documents for q is

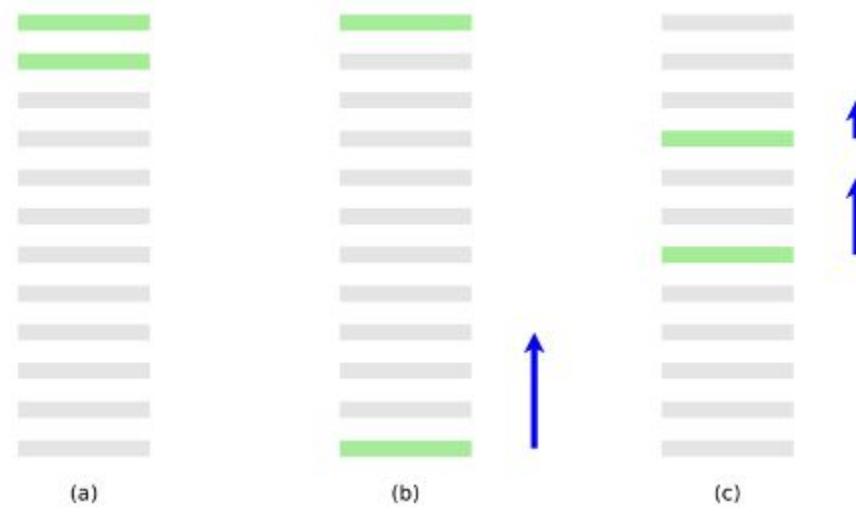
$$\lambda_i = \sum_{j:\{i,j\} \in I} \lambda_{ij} - \sum_{k:\{k,i\} \in I} \lambda_{ki}$$

- λ_i is (sort of) a gradient of the pairwise loss of vector x_i



RankNet lambdas

- (a) is the perfect ranking, (b) is a ranking with 10 pairwise errors, (c) is a ranking with 8 pairwise errors. Each blue arrow represents the λ_i for each query-document vector x_i



RankNet lambdas

- Problem: RankNet is based on pairwise error, while modern IR measures emphasize higher ranking positions. Red arrows show better λ 's for modern IR, esp. web search.



From RankNet to LambdaRank

- Rather than working with pairwise ranking errors, scale by effect a change has on NDCG
- Idea: Multiply λ 's by $|\Delta Z|$, the difference of an IR measure when d_i and d_j are swapped
- E.g. $|\Delta \text{NDCG}|$ is the change in NDCG when swapping d_i and d_j giving

$$\lambda_{ij} = \frac{\partial C(s_i - s_j)}{\partial s_i} = \frac{-\sigma}{1 + e^{\sigma(s_i - s_j)}} |\Delta \text{NDCG}|$$

- Burges et al. “prove” (partly theory, partly empirical) that this change is sufficient for model to optimize NDCG



From RankNet to LambdaRank

- LambdaRank models gradients
- MART can be trained with gradients (“gradient boosting”)
- Combine both to get LambdaMART
 - MART with specified gradients and optimization step



LambdaRank Algorithm

set number of trees N , number of training samples m , number of leaves per tree L , learning rate η

for $i = 0$ to m **do**

$F_0(x_i) = \text{BaseModel}(x_i)$ //If BaseModel is empty, set $F_0(x_i) = 0$

end for

for $k = 1$ to N **do**

for $i = 0$ to m **do**

$$y_i = \lambda_i$$

$$w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$$

end for

$\{R_{lk}\}_{l=1}^L$ // Create L leaf tree on $\{x_i, y_i\}_{i=1}^m$ R_{lk} is data items at leaf node l

$\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$ // Assign leaf values based on Newton step.

$F_k(x_i) = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$ // Take step with learning rate η .

end for



Yahoo! Learning to rank challenge

- Goal was to validate learning to rank methods on a large, “real” web search problem
 - Previous work was mainly driven by LETOR datasets
 - Great as first public learning-to-rank data
 - Small: 10s of features, 100s of queries, 10k's of docs
- Only feature vectors released
 - Not URLs, queries, nor feature descriptions
 - Wanting to keep privacy and proprietary info safe
 - But included web graph features, click features, page freshness and page classification features as well as text match features



Takeaway Messages

- The idea of learning ranking functions has been around for about 30 years
- ML knowledge, availability of training datasets, a rich space of features, and massive computation came together to make this a hot research area
- Typically LambdaMART outperforms other methods in real search ranking tasks
- MLR over many features now easily beats traditional hand-designed ranking functions in comparative evaluations
 - [in part by using the hand-designed functions as features!]
- Next Step: [NeuralIR!](#)



Text Classification & Naive Bayes

Chapter 13 - IIR



SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

Quick Intro



SAPIENZA
UNIVERSITÀ DI ROMA

A text classification task: Email spam filtering

From: ' ' <takworl1d@hotmail.com>
Subject: real estate is the only way... gem oalvgkay
Anyone can buy real estate with no money down
Stop paying rent TODAY !
There is no need to spend hundreds or even thousands for similar courses
I am 22 years old and I have already purchased 6 properties using the
methods outlined in this truly INCREDIBLE ebook.
Change your life NOW !

=====

Click Below to order:

<http://www.wholesaledaily.com/sales/nmd.htm>

=====

- How would you proceed to decide whether this text is spam or ham?



Text Classification: a (more) formal definition

- Given:
 - A document space X
 - Documents are represented in this space – typically some type of high-dimensional space.
 - A fixed set of classes $C = \{c_1, c_2, \dots, c_J\}$
 - The classes are human-defined for the needs of an application (e.g., spam vs. nonspam).
 - A training set D of labeled documents.
 - Each labeled document $(d, c) \in X \times C$

Using a learning method or learning algorithm, we then wish to learn a classifier f that maps documents to classes:

$$f : X \rightarrow C$$

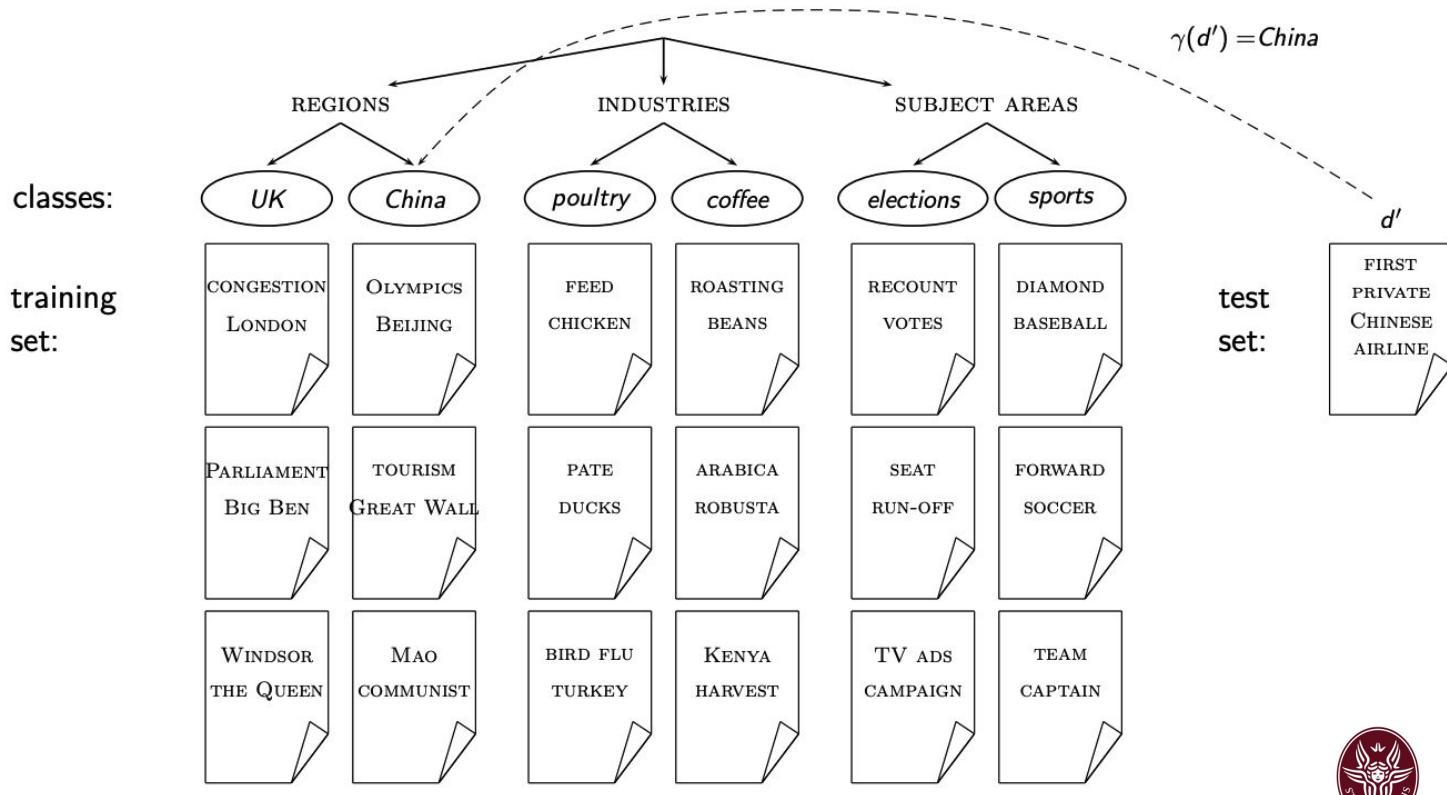


Text Classification: Inference

- Given a representation for a document $d \in X$, determine the most appropriate class using the learnt function f
- In other words $C = f(d)$



An example: Topic Classification



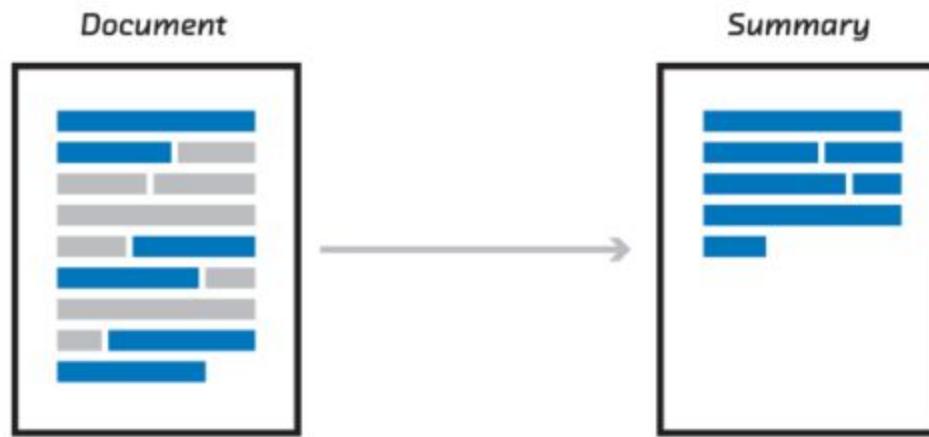
An example: Sentiment Analysis



An example: Misinformation Classification



An example: Automatic Summarization



An example: Language Detection

A screenshot of a Google search results page. The search bar at the top contains the query "language \"ciao\"". Below the search bar are navigation links for All, Images, Videos, News, Maps, and More, with "All" being the selected category. To the right are links for Settings and Tools. A message indicates "About 20,700,000 results (0.58 seconds)". The main content area features a large, bold title "Italian". Below it is a detailed description of the word "Ciao": "Ciao (/tʃaʊ/; **Italian** pronunciation: [tʃa:o]) is an informal salutation in the **Italian language** that is used for both "hello" and "goodbye". Originally from the **Venetian language**, it has entered the vocabulary of **English** and of many **other languages** around the world." At the bottom, there is a link to the Wikipedia article: "https://en.wikipedia.org › wiki › Ciao".

Italian

Ciao (/tʃaʊ/; **Italian** pronunciation: [tʃa:o]) is an informal salutation in the **Italian language** that is used for both "hello" and "goodbye". Originally from the **Venetian language**, it has entered the vocabulary of **English** and of many **other languages** around the world.

<https://en.wikipedia.org › wiki › Ciao>

[Ciao - Wikipedia](#)



SAPIENZA
UNIVERSITÀ DI ROMA

An example: Question Detection

Google X |

All Images News Videos Shopping More Settings Tools

About 450,000,000 results (0.60 seconds)

7 feet, 1 inch

Peppa's height is **7 feet, 1 inch**.

[https://peppafanon.fandom.com/wiki/Peppa_Pig_\(char...](https://peppafanon.fandom.com/wiki/Peppa_Pig_(character))

Peppa Pig (character) | Peppa Pig Fanon Wiki | Fandom

About featured snippets • Feedback

People also ask

Is Peppa Pig 7 ft tall? ▾

How tall is Peppa Pig in real life? ▾

How tall is George the Pig? ▾

How did Peppa Pig die? ▾

Feedback



SAPIENZA
UNIVERSITÀ DI ROMA

An example: Vertical Selection

mineral water

About 417,000,000 results (1.07 seconds)

[All](#) [Images](#) [Videos](#) [Shopping](#) [Maps](#) [More](#) [Settings](#) [Tools](#)

Mineral water is water from a mineral spring that contains various minerals, such as salts and sulfur compounds. Mineral water may usually be still or sparkling (carbonated/effervescent) according to the presence or absence of added gases.

https://en.wikipedia.org/wiki/Mineral_water

Mineral water - Wikipedia

People also ask

- Is it good to drink mineral water?
- What is the best mineral water to drink?
- Can you make your own mineral water?
- Is mineral water the same as distilled water?

Feedback

Find results on

- Indiamart
- Karpacz
- Nature
- Influence of a

<https://www.healthline.com/nutrition/mineral-water-...>

Does Mineral Water Have Health Benefits? - Healthline

Sep 4, 2019 — As its name suggests, mineral water can contain high amounts of minerals and other naturally occurring compounds, including magnesium, ...
What it is · Benefits · Bottom line · Mineral

vs.

johnson lindenstrauss

About 79,600 results (0.44 seconds)

[All](#) [Images](#) [News](#) [Shopping](#) [Videos](#) [More](#) [Settings](#) [Tools](#)

https://en.wikipedia.org/wiki/Johnson–Lindenstrauss_lemma ▾

Johnson–Lindenstrauss lemma - Wikipedia

In mathematics, the Johnson–Lindenstrauss lemma is a result named after William B. Johnson and Joram Lindenstrauss concerning low-distortion embeddings ...

<https://home.ttic.edu/~gregory/courses/lectures/> ▾ PDF

Random Projections 1 The Johnson–Lindenstrauss ... - TTIC

1 The Johnson–Lindenstrauss lemma. Theorem 1.1. (Johnson–Lindenstrauss) Let $\epsilon \in (0, 1/2)$. Let $Q \subset \mathbb{R}^d$ be a set of n points and $k = 20 \log n / \epsilon^2$. There,

<https://cs.stanford.edu/Lectures/lecture1> ▾ PDF

The Johnson–Lindenstrauss Lemma - Stanford Computer ...

Sep 23, 2009 — The Johnson–Lindenstrauss Lemma states that any n points in high dimensional euclidian space can be mapped onto k dimensions where $k \geq O(\log n/\epsilon^2)$ without distorting the euclidian distance between any two points. more than a factor of $1 \pm \epsilon$.

<https://www.cantorsparadise.com/the-johnson-lindenst...> ▾

The Johnson–Lindenstrauss Lemma. Why you don't always ...

Apr 7, 2020 — Johnson (1944-) and Joram Lindenstrauss (1936–2012). Informally, the lemma says that, given N points with N coordinates each (i.e., these ...

<http://cseweb.ucsd.edu/~dakane/derandomizedJL> ▾ PDF

Almost Optimal Explicit Johnson–Lindenstrauss ... - UCSD CSE

by DM Kane · Cited by 57 · Abstract. The Johnson–Lindenstrauss lemma is a fundamental result in probability with several applications in the design and analysis of algorithms.



Some Text Classification Methods



SAPIENZA
UNIVERSITÀ DI ROMA

Rule Based

- E.g., Google Alerts is rule-based classification.
- There are IDE-type development environments for writing very complex rules efficiently. (e.g., Verity)
- Often: Boolean combinations (as in Google Alerts)
- Accuracy is very high if a rule has been carefully refined over time by a subject expert.
- Building and maintaining rule-based classification systems is cumbersome and expensive.



Statistical Modeling

- This was our definition of the classification problem – text classification as a learning problem
 - (i) Supervised learning of a classification function f and
 - (ii) application of f to classifying new documents
- We will look at two methods for doing this: Naive Bayes and SVMs
- No free lunch: requires hand-classified training data
 - But this manual classification can be done by non-experts.



Naïve Bayes



SAPIENZA
UNIVERSITÀ DI ROMA

What is a Naïve Bayes Classifier

- The Naive Bayes classifier is a probabilistic classifier.
- We compute the probability of a document d being in a class c as follows:

$$P(c|d) \propto P(c) \prod_{1 \leq k \leq n_d} P(t_k|c)$$

- n_d is the length of the document. (number of tokens)
- $P(t_k | c)$ is the conditional probability of term t_k occurring in a document of class c
- $P(t_k | c)$ as a measure of **how much evidence** t_k contributes that c is the correct class.
- $P(c)$ is the prior probability of c.
- If a document's terms do not provide clear evidence for one class vs. another, we choose the c with highest $P(c)$.



Maximum A Posteriori (MAP) Classifier

- Our goal in Naive Bayes classification is to find the “best” class.
- The best class is the most likely or Maximum A Posteriori (MAP) class c_{map} :

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg \max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)$$



Actually...

- Multiplying lots of small probabilities can result in floating point underflow.
- Since $\log(xy) = \log(x) + \log(y)$, we can sum log probabilities instead of multiplying probabilities.
- Since log is a monotonic function, the class with the highest score does not change.
- So what we usually compute in practice is:

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k | c)]$$



Summarizing: Naïve Bayes Classifier

$$c_{\text{map}} = \arg \max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \leq k \leq n_d} \log \hat{P}(t_k | c)]$$

- Each conditional parameter $\log \hat{P}(t_k | c)$ is a weight that indicates how good an indicator t_k is for c .
- The prior $\log \hat{P}(c)$ is a weight that indicates the relative frequency of c .
- The sum of log prior and term weights is then a measure of how much evidence there is for the document being in the class.
- We select the class with the most evidence.



Parameter Estimation: MLE

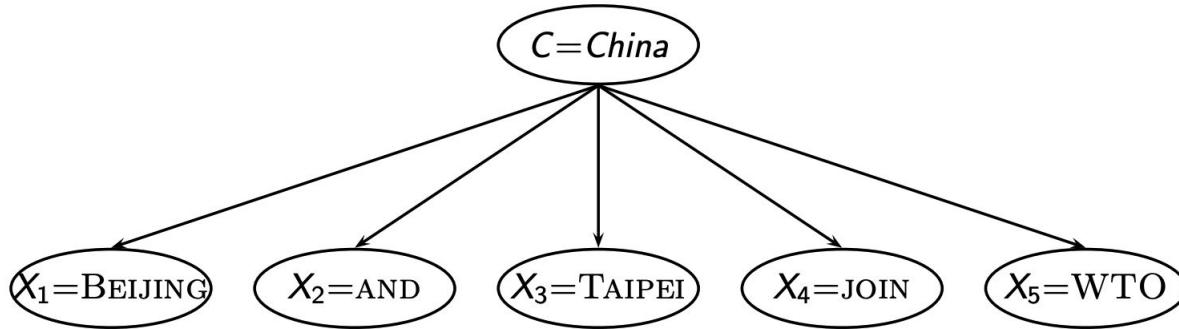
- Estimate parameters $P(c)$ and $P(t_k | c)$ from train data: How?
- Prior: $P(c) = N_c / N$
 - N_c : number of docs in class c ; N : total number of docs
- Conditional probabilities:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- T_{ct} is the number of tokens of t in training documents from class c (includes multiple occurrences)
- We've made a **Naive Bayes independence assumption** here:
 $P(t_k | c) = P(t_k | c)$, independent of position.



How to deal with zeros?



$$P(China|d) \propto P(China) \cdot P(BEIJING|China) \cdot P(AND|China) \\ \cdot P(TAIPEI|China) \cdot P(JOIN|China) \cdot P(WTO|China)$$

We will get $P(China|d) = 0$ for any document that contains WTO!

...er occurs in class China in the train set:

$$\hat{P}(WTO|China) = \frac{T_{China,WTO}}{\sum_{t' \in V} T_{China,t'}} = \frac{0}{\sum_{t' \in V} T_{China,t'}} = 0$$



Add Smoothing

- Before:

$$\hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

- Now: Add one to each count to avoid zeros:

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

- B is the number of bins – in this case the number of different words or the size of the vocabulary $|V| = M$

Naïve Bayes: Summary

- Estimate parameters from the training corpus using add-one smoothing
- For a new document, for each class, compute sum of (i) log of prior and (ii) logs of conditional probabilities of the terms
- Assign the document to the class with the largest score



Naïve Bayes: Training

TRAINMULTINOMIALNB(C, \mathbb{D})

- 1 $V \leftarrow \text{EXTRACTVOCABULARY}(\mathbb{D})$
- 2 $N \leftarrow \text{COUNTDocs}(\mathbb{D})$
- 3 **for each** $c \in C$
- 4 **do** $N_c \leftarrow \text{COUNTDOCSINCLASS}(\mathbb{D}, c)$
- 5 $prior[c] \leftarrow N_c / N$
- 6 $text_c \leftarrow \text{CONCATENATETEXTOفالDOCSINCLASS}(\mathbb{D}, c)$
- 7 **for each** $t \in V$
- 8 **do** $T_{ct} \leftarrow \text{COUNTTOKENSOFTERM}(text_c, t)$
- 9 **for each** $t \in V$
- 10 **do** $condprob[t][c] \leftarrow \frac{T_{ct} + 1}{\sum_{t'}(T_{ct'} + 1)}$
- 11 **return** $V, prior, condprob$



Naïve Bayes: Testing

APPLYMULTINOMIALNB($\mathbb{C}, V, prior, condprob, d$)

- 1 $W \leftarrow \text{EXTRACTTOKENSFROMDOC}(V, d)$
- 2 **for each** $c \in \mathbb{C}$
- 3 **do** $score[c] \leftarrow \log prior[c]$
- 4 **for each** $t \in W$
- 5 **do** $score[c] += \log condprob[t][c]$
- 6 **return** $\arg \max_{c \in \mathbb{C}} score[c]$



Exercise: Parameter Estimation

	docID	words in document	in $c = China$?
training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$\hat{P}(c) = \frac{N_c}{N}$$

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V} (T_{ct'} + 1)} = \frac{T_{ct} + 1}{(\sum_{t' \in V} T_{ct'}) + B}$$

(B is the number of bins – in this case the number of different words or the size of the vocabulary $|V| = M$)

$$c_{\text{map}} = \arg \max_{c \in \mathcal{C}} [\hat{P}(c) \cdot \prod_{1 \leq k \leq n_d} \hat{P}(t_k|c)]$$



Solution

Priors: $\hat{P}(c) = 3/4$ and $\hat{P}(\bar{c}) = 1/4$ Conditional probabilities:

$$\hat{P}(\text{CHINESE}|c) = (5 + 1)/(8 + 6) = 6/14 = 3/7$$

$$\hat{P}(\text{TOKYO}|c) = \hat{P}(\text{JAPAN}|c) = (0 + 1)/(8 + 6) = 1/14$$

$$\hat{P}(\text{CHINESE}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

$$\hat{P}(\text{TOKYO}|\bar{c}) = \hat{P}(\text{JAPAN}|\bar{c}) = (1 + 1)/(3 + 6) = 2/9$$

The denominators are $(8 + 6)$ and $(3 + 6)$ because the lengths of text_c and $\text{text}_{\bar{c}}$ are 8 and 3, respectively, and because the constant B is 6 as the vocabulary consists of six terms.



Solution

$$\hat{P}(c|d_5) \propto 3/4 \cdot (3/7)^3 \cdot 1/14 \cdot 1/14 \approx 0.0003$$

$$\hat{P}(\bar{c}|d_5) \propto 1/4 \cdot (2/9)^3 \cdot 2/9 \cdot 2/9 \approx 0.0001$$

Thus, the classifier assigns the test document to $c = China$. The reason for this classification decision is that the three occurrences of the positive indicator CHINESE in d_5 outweigh the occurrences of the two negative indicators JAPAN and TOKYO.



Implementing NB

- [NB from scratch](#)
- [Using scikit-learn](#)



Naïve Bayes: Computational Complexity

mode	time complexity
training	$\Theta(\mathbb{D} L_{ave} + \mathbb{C} V)$
testing	$\Theta(L_a + \mathbb{C} M_a) = \Theta(\mathbb{C} M_a)$

- L_{ave} : average length of a training doc, L_a : length of the test doc, M_a : number of distinct terms in the test doc, \mathbb{D} : training set, V : vocabulary, \mathbb{C} : set of classes
- $\Theta(|\mathbb{D}|L_{ave})$ is the time it takes to compute all counts.
- $\Theta(|\mathbb{C}||V|)$ is the time it takes to compute the parameters from the counts.
- Generally: $|\mathbb{C}||V| < |\mathbb{D}|L_{ave}$
- Test time is also linear (in the length of the test document).
- Thus: Naive Bayes is linear in the size of the training set (training) and the test document (testing). This is optimal.



Theoretical Analysis of Naïve Bayes



SAPIENZA
UNIVERSITÀ DI ROMA

Properties of NB

- Now we want to gain a better understanding of the properties of Naive Bayes.
- We will formally derive the classification rule . . .
- . . . and make our assumptions explicit.



Derivation of NB

- We want to find the class that is most likely given the document:

$$c_{\text{map}} = \arg \max_{c \in \mathcal{C}} P(c|d)$$

- Apply Bayes rule $P(c|d) = (1/P(d)) * P(d|c)P(c)$

$$c_{\text{map}} = \arg \max_{c \in \mathcal{C}} \frac{P(d|c)P(c)}{P(d)}$$

- Drop denominator since $P(d)$ is the same for all classes:

$$c_{\text{map}} = \arg \max_{c \in \mathcal{C}} P(d|c)P(c)$$



Data sparsity

$$\begin{aligned}c_{\text{map}} &= \arg \max_{c \in \mathcal{C}} P(d|c)P(c) \\&= \arg \max_{c \in \mathcal{C}} P(\langle t_1, \dots, t_k, \dots, t_{n_d} \rangle | c)P(c)\end{aligned}$$

- There are too many parameters $P(\langle t_1, \dots, t_k, \dots, t_n \rangle | c)$, one for each unique combination of a class and a sequence of words.
- We would need a very, very large number of training examples to estimate that many parameters.
- This is the problem of **data sparseness**.



NB: conditional independence assumption

- To reduce the number of parameters to a manageable size, we make the Naive Bayes conditional independence assumption:

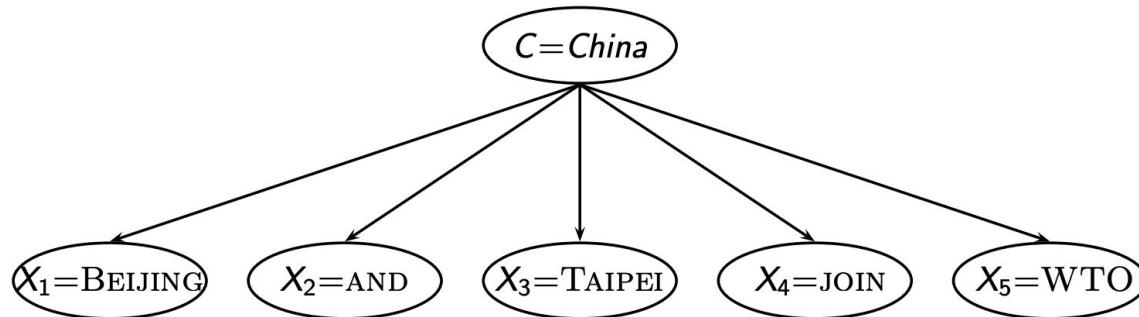
$$P(d|c) = P(\langle t_1, \dots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

- We assume that the probability of observing the conjunction of attributes is equal to the product of the individual probabilities $P(X_k = t_k | c)$. Recall from earlier the estimates for these conditional probabilities:

$$\hat{P}(t|c) = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'})+B}$$



NB as a Generative Model



$$P(c|d) \propto P(c) \prod_{1 < k < n_d} P(t_k|c)$$

- Generate a class with probability $P(c)$
- Generate each of the words (in their respective positions), conditional on the class, but independent of each other, with probability $P(t_k|c)$
- To classify docs, we “reengineer” this process and find the class that is most likely to have generated the doc.

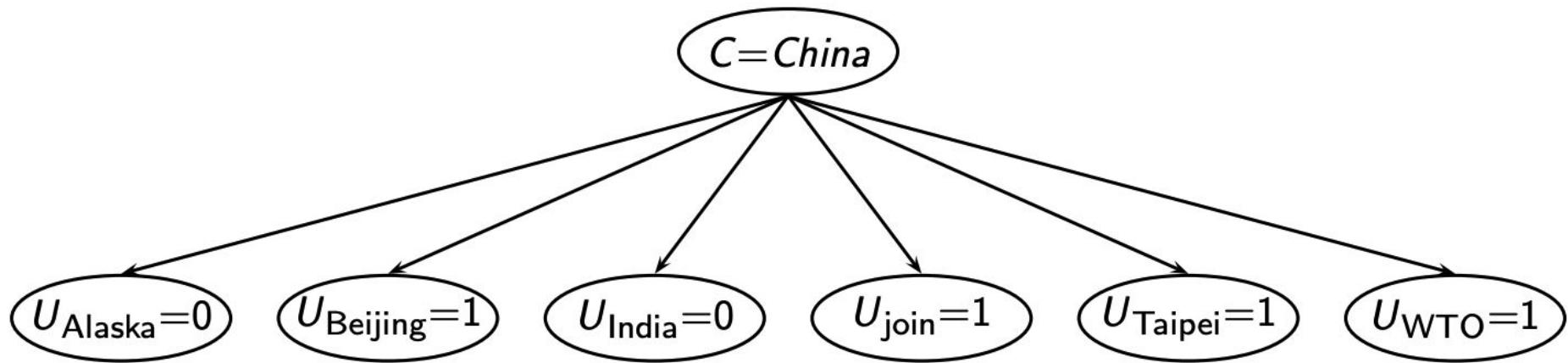


Naïve Bayes: The Second Independence Assumption

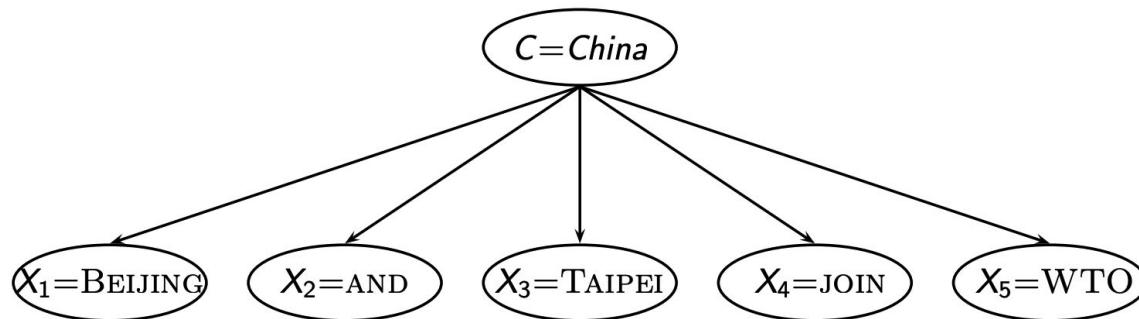
- $\hat{P}(X_{k_1} = t|c) = \hat{P}(X_{k_2} = t|c)$
- For example, for a document in the class UK, the probability of generating queen in the first position of the document is the same as generating it in the last position.
- The two independence assumptions amount to the bag of words model.



Bernoulli NB



Gaussian NB



$$P(c|d) \propto P(c) \prod_{1 < k < n_d} P(t_k|c)$$

- Each $P(t_k|c) \sim N(m, s)$
 - Normally distributed with parameter m and s estimated on the training set



Violations of the Independence Assumptions

- Conditional independence:

- $$P(\langle t_1, \dots, t_{n_d} \rangle | c) = \prod_{1 \leq k \leq n_d} P(X_k = t_k | c)$$

- Positional independence:

- $$\hat{P}(X_{k_1} = t | c) = \hat{P}(X_{k_2} = t | c)$$

- The independence assumptions do not really hold of documents written in natural language.

- Exercise

- Examples for why conditional independence assumption is not really true?
- Examples for why positional independence assumption is not really true?

- How can Naive Bayes work if it makes such inappropriate assumptions?



Why does NB work?

- Naive Bayes can work well even though conditional independence assumptions are badly violated.
- Example:

	c_1	c_2	class selected
true probability $P(c d)$	0.6	0.4	c_1
$\hat{P}(c) \prod_{1 \leq k \leq n_d} \hat{P}(t_k c)$	0.00099	0.00001	
NB estimate $\hat{P}(c d)$	0.99	0.01	c_1

- Double counting of evidence causes underestimation (0.01) and overestimation (0.99).
- Classification is about predicting the correct class and not about accurately estimating probabilities.
- Naive Bayes is terrible for correct estimation . . .
 - but it often performs well at accurate prediction (choosing the correct class).



NB is it so Naive?

- Naive Bayes has won some bakeoffs (e.g., KDD-CUP 97)
- More robust to nonrelevant features than some more complex learning methods
- More robust to concept drift (changing of definition of class over time) than some more complex learning methods
- Better than methods like decision trees when we have **many equally important features**
- A good dependable baseline for text classification (but not the best)
- Optimal if independence assumptions hold (never true for text, but true for some domains)
- Very fast
- Low storage requirements

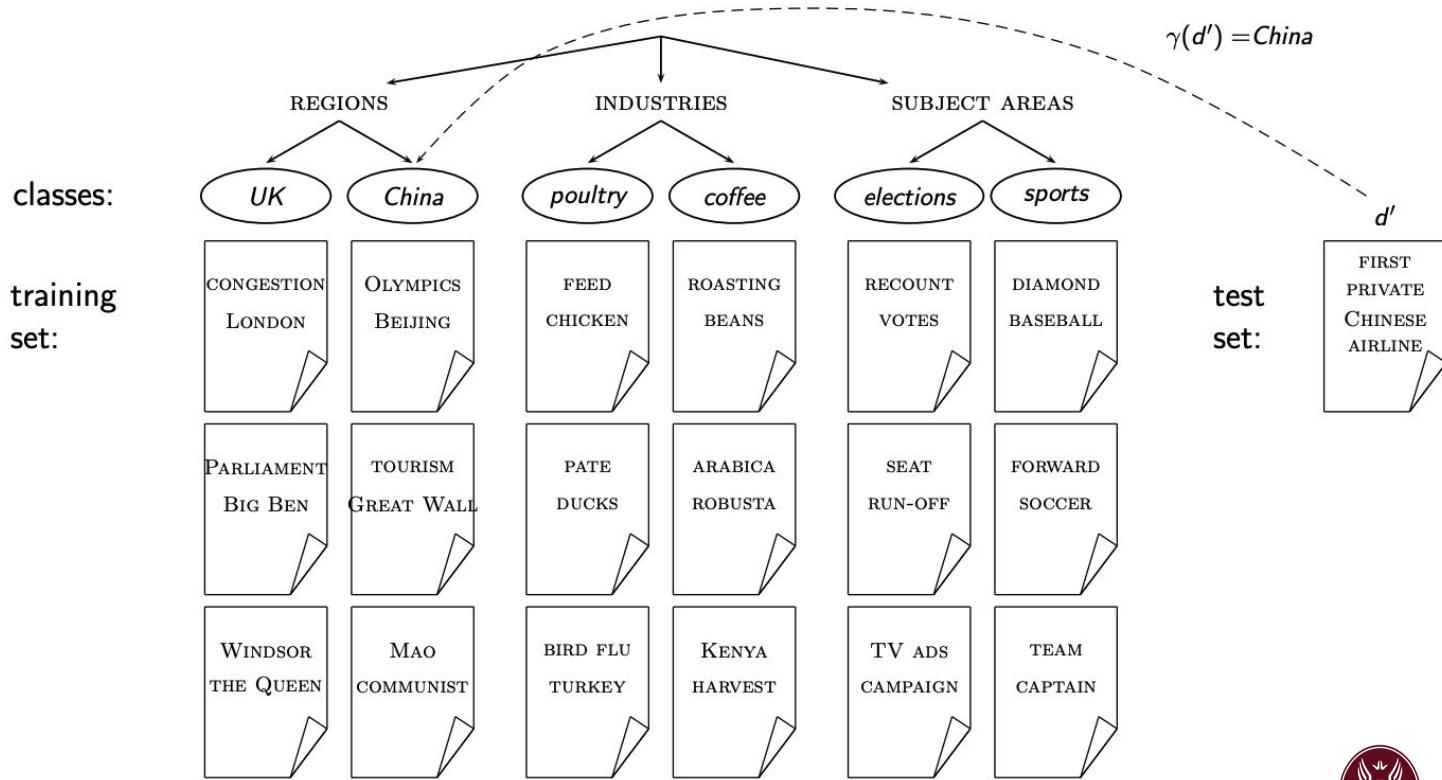


Evaluation



SAPIENZA
UNIVERSITÀ DI ROMA

Reuters Collection



The Reuters Collection

symbol	statistic	value
N	documents	800,000
L	avg. # word tokens per document	200
M	word types	400,000

type of class	number	examples
region	366	UK, China
industry	870	poultry, coffee
subject area	126	elections, sports



An example of a Reuters Document

May 4, 2021
2:17 PM CEST

India

'Last resort': Desperate for oxygen, Indian hospitals go to court

4 minute read

Aditya Kalra

[Facebook](#) [Twitter](#) [Email](#)



Patients suffering from the coronavirus disease (COVID-19) get treatment at the casualty ward at the Lok Nayak Jai Prakash

A court in India's capital New Delhi has become the last hope for many hospitals [struggling to get oxygen for COVID-19 patients](#) as supplies run dangerously short while government officials bicker over who is responsible.



SAPIENZA
UNIVERSITÀ DI ROMA

Evaluating Classification

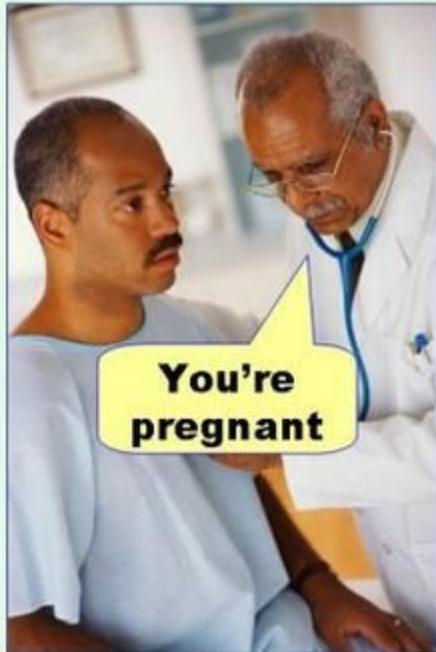
- Evaluation must be done on test data that are independent of the training data, i.e., training and test sets are disjoint.
- It's easy to get good performance on a test set that was available to the learner during training (e.g., just memorize the test set).
- Measures: Precision, recall, F_1 , classification accuracy



The Confidence Interval

Actual
Value

Type I error
(false positive)



Type II error
(false negative)



\overline{N})



SAPIENZA
UNIVERSITÀ DI ROMA

F_1 Measure

- F_1 allows us to trade off precision against recall.

$$F_1 = \frac{1}{\frac{\frac{1}{2}\frac{1}{P}}{R} + \frac{\frac{1}{2}\frac{1}{R}}{P}} = \frac{2PR}{P+R}$$

- This is the harmonic mean of P and R : $\frac{1}{F} = \frac{1}{2}(\frac{1}{P} + \frac{1}{R})$



Averaging: Micro vs. Macro

- We now have an evaluation measure (F_1) for one class.
- But we also want a single number that measures the aggregate performance over all classes in the collection.
- Macroaveraging
 - Compute F_1 for each of the C classes
 - Average these C numbers
- Microaveraging
 - Compute TP, FP, FN for each of the C classes
 - Sum these C numbers (e.g., all TP to get aggregate TP)
 - Compute $F1$ for aggregate TP, FP, FN



Takeaway Messages

- Text classification: definition & relevance to information retrieval
- Naive Bayes: simple baseline text classifier
- Theory: derivation of Naive Bayes classification rule & analysis
- Evaluation of text classification: how do we know it worked / didn't work?



(Modern) Language Models

Fabrizio Silvestri

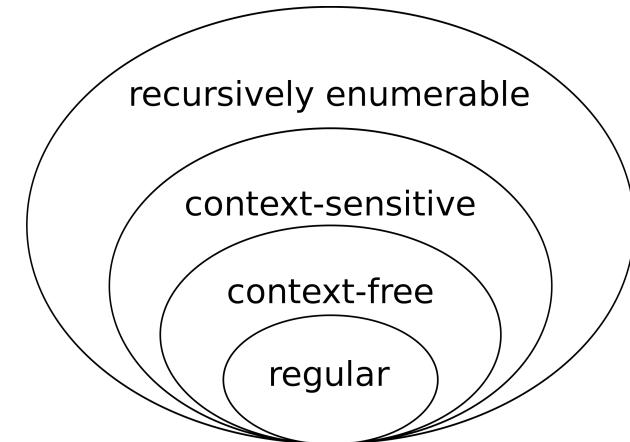


SAPIENZA
UNIVERSITÀ DI ROMA

Fabrizio Silvestri

How to Model a Language?

- You may recall from early classes:
 - Chomsky' Hierarchy
 - Formally defined
 - In many cases automata can recognize a language
 - FSAs
 - Push-down Automata
 - Turing Machines
- In general NLP makes use of **Statistical Language Models (LMs)**
 - Better suited for natural language



$$P(w_1 w_2 \dots w_T) = \prod_{i=1}^n P(w_i | w_1 w_2 \dots w_{i-1})$$



Applications of LMs

- Information Retrieval
 - Each document is generated by a different LM
 - Queries are generated by a LM
 - Which of the documents' LMs are most similar to the query's LM?
- Speech Recognition
 - "I ate eight apples" more likely than "Aye eight ate apples"?
- Spell Correction
 - "Google" more likely than "Googel"?
- Natural Language Generation
 - What's the most likely continuation of this <X>?
 - Question, Sentence, Phrase, Paragraph, ...
- Translation
 - ES → "El cafe negro me gusta mucho"
 - IT → "Il caffè nero mi piace molto"
 - EN → "I like black coffee very much" vs "The coffee black me pleases much"
- ...



Formal LM Definition

- Goal:
 - Define a probability distribution over “tokens” to be able to measure likelihood of sequences s of tokens $p(s) = p(t_1, t_2, \dots, t_n)$
- This way NLP tasks can be seen as computing a probability score for components of tasks
 - Information Retrieval
 - Probability of a document given a query: $p(d|q) = p(q|d) \frac{p(d)}{p(q)}$
 - Translation
 - Noisy channel: $p(d_{eng}|d_{ita}) \propto p(d_{eng}, d_{ita}) = p(d_{ita}|d_{eng}) p(d_{eng})$



K-Gram Language Models

- Relative Frequency Estimate

$$p(\text{Computers are useless, they can only give you answers}) = \frac{\text{Count}(\text{Computers are useless, they can only give you answers})}{\text{Count}(\text{all sentences ever spoken or written})}$$

- Estimator is unbiased but, can we use it?
 - “*Count(all sentences ever spoken or written)*” is extremely large.
 - Even assuming $|V| = 10^5$ and $\max |x| = 10$ we have that $\text{Count}(\dots)$ is 10^{50} .
- Let's consider the probability of the sequence of tokens: $p(s) = p(t_1, t_2, \dots, t_n)$
 - Which formally is: $p(t_1, \dots, t_n) = p(t_1)p(t_2|t_1)p(t_3|t_2, t_1), \dots, (t_n|t_1, t_2, \dots, t_{n-1})$
- We simplify by considering only subsequences of length k :

$$p(t_m|t_{m-1}, \dots, t_1) \sim p(t_m| \underbrace{t_{m-1}, \dots, t_{m-k+1}}_{k\text{-gram}})$$



Berkeley Restaurant Project (BRP) sentences

- can you tell me about any good cantonese restaurants close by
- mid priced thai food is what i'm looking for
- tell me about chez panisse
- can you give me a listing of the kinds of food that are available
- i'm looking for a good place to eat breakfast
- when is caffe venezia open during the day



BRP - Raw Bigram Counts (out of 9,222 sentences)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0



BRP - Raw Bigram Probabilities

Unigram Counts

	i	want	to	eat	chinese	food	lunch	spend
	2533	927	2417	746	158	1093	341	278

Bigram probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0



Bigram estimates of sentence probabilities

- $P(< s > | \text{I want english food } < /s >) =$
 - $P(\text{I} | < s >) *$
 - $P(\text{want} | \text{I}) *$
 - $P(\text{english} | \text{want}) *$
 - $P(\text{food} | \text{english}) *$
 - $P(< /s > | \text{food}) =$
- 0.00031

$$P(\text{lunch} | \text{want}) = P(\text{chinese} | \text{want}) = 0$$

I want lunch

I want ~~chinese~~ food



K-Gram Language Models

```
def main():
    nltk.download('reuters')
    nltk.download('punkt')
    text_sentences = reuters.sents()

    model = init_model()
    model = count_cooccurrences(model)
    model = generate_probabilities(model)

    sentence_to_be_continued = 'market analysis'

    current_sentence = sentence_to_be_continued
    n_continuations = 20
    for iteration in range(n_continuations):
        current_sentence = continue_sentence_greedy(model, current_sentence)
    print('Greedy:\t{}'.format(current_sentence))

    current_sentence = sentence_to_be_continued
    n_continuations = 20
    k = 5
    for iteration in range(n_continuations):
        current_sentence = continue_sentence_top_k(model, current_sentence, k)
    print('top-{}:\t{}'.format(k, current_sentence))
```



K-Gram Language Models

```
def count_cooccurrences(model):
    # Count frequency of co-occurrence
    for sentence in reuters.sents():
        for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
            model[(w1, w2)][w3] += 1

    return model


def generate_probabilities(model):
    # Let's transform the counts to probabilities
    for w1_w2 in model:
        total_count = float(sum(model[w1_w2].values()))
        for w3 in model[w1_w2]:
            model[w1_w2][w3] /= total_count

    return model
```



Greedy NL Generation

```
def continue_sentence_greedy(model, prefix):
    # Use a greedy approach to generate sentences
    new_sentence = prefix
    last_bigram = prefix.lower().split()[-2:]
    last_bigram = (last_bigram[0], last_bigram[1])
    alternatives = dict(model[last_bigram])
    if len(alternatives) > 0:
        continuation = max(alternatives.items(), key=operator.itemgetter(1))[0]
        if continuation:
            new_sentence = ' '.join([new_sentence, continuation])

    return new_sentence
```



Top-K NL Generation

```
def continue_sentence_top_k(model, prefix, k):
    # Use a top-k approach to generate sentences
    new_sentence = prefix
    last_bigram = prefix.lower().split()[-2:]
    last_bigram = (last_bigram[0], last_bigram[1])
    alternatives = dict(model[last_bigram])
    if len(alternatives) > 0:
        k = min(len(alternatives), k)
        continuations = nlargest(k, alternatives, key=alternatives.get)
        continuation = random.choice(continuations)
        if continuation:
            new_sentence = ' '.join([new_sentence, continuation])

    return new_sentence
```



Examples of Text Generated by N-Gram LMs

- ‘today they’
 - Greedy: **today they** found the United states since the beginning of the company ' s & It ; BP
 - top-5: **today they** found the United states will make no contribution to this process .
- ‘the beginning’
 - Greedy: **the beginning** of the company ' s & It ; BP
 - top-5: **the beginning** of a share for the current level , despite massive central bank , the company .
- ‘news articles’
 - Greedy: **news articles**
 - top-5: **news articles**
- ‘the base’
 - Greedy: **the base** rate cut , and the U
 - top-5: **the base** in data storage subsystems for the current level ' is rather confident currency stability will benefit all foreign meat processing



Bigram estimates of sentence probabilities

- $P(< s > | \text{I want english food } < /s >) =$
 - $P(\text{I} | < s >) *$
 - $P(\text{want} | \text{I}) *$
 - $P(\text{english} | \text{want}) *$
 - $P(\text{food} | \text{english}) *$
 - $P(< /s > | \text{food}) =$
- 0.00031

$$P(\text{lunch} | \text{want}) = P(\text{chinese} | \text{want}) = 0$$

I want lunch

I want ~~chinese~~ food



Smoothing

- What if $p(w) = 0$?
- **Smoothing** → add an imaginary “pseudo-count” to avoid situations where the probability of a token is zero
- Lidstone smoothing:

$$p_{\text{smooth}}(w_m | w_{m-1}) = \frac{\text{count}(w_{m-1}, w_m) + \alpha}{\sum_{w' \in \mathcal{V}} \text{count}(w_{m-1}, w') + |\mathcal{V}| \alpha}$$

- Laplace's smoothing (add one) → $\alpha = 1$
- Jeffreys-Perks law → $\alpha = 0.5$
- Kneser-Ney smoothing → considers $p_{\text{continuations}}$



BRP - Laplace Smoothing

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	1	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1



BRP - Laplace Smoothed Bigram Probabilities

Unigram Counts

	i	want	to	eat	chinese	food	lunch	spend
	2534	928	2418	747	159	1094	342	279

Bigram probabilities

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0022	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00083	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0063	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.014	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0059	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0036	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058



Reconstituted Counts

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.19



	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.19

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0



Backoff and Interpolation

- Backoff
 - Use n-grams if you have enough evidence
 - Otherwise → Use (n-1)-grams if you have enough evidence of this kind
 - ...
 - Finally → Use unigrams if you have enough evidence of this kind
- Katz Backoff
- Interpolation
 - Mix n-grams, (n-1)-grams, ..., unigrams

} Stupid Backoff

Interpolation works better



SAPIENZA
UNIVERSITÀ DI ROMA

Linear Interpolation

- Simple. E.g., tri-gram
 - $P'(w_n | w_{n-1}, w_{n-2}) = \lambda_1 P(w_n | w_{n-1}, w_{n-2}) + \lambda_2 P(w_n | w_{n-1}) + \lambda_3 P(w_n)$
 - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
- Lambdas conditional on context:
 - $P'(w_n | w_{n-1}, w_{n-2}) = \lambda_1(w_{n-1}, w_{n-2})P(w_n | w_{n-1}, w_{n-2}) + \lambda_2(w_{n-1}, w_{n-2})P(w_n | w_{n-1}) + \lambda_3(w_{n-1}, w_{n-2})P(w_n)$
- To set λ 's one can use a held-out dataset



OOV: Out Of Vocabulary

- If we know all the words in advanced
 - Vocabulary V is fixed
 - Closed vocabulary task
- Often we don't know this
 - Out Of Vocabulary (OOV) words
 - Open vocabulary task
- Instead: create an unknown word token <UNK>
 - Training of <UNK> probabilities
 - Create a fixed vocabulary V
 - At text normalization phase, any training word not in V changed to <UNK>
 - Treat <UNK> like a normal word
 - At inference time use UNK probabilities for any word not in V



Language Models in IR: The Query Likelihood LM

- Given a query q the goal is to compute $P(d|q)$ for each d in a collection
- Each document d is represented by a language model M_d
- Bayes rules is used to compute $P(d|q)$ as $P(d)/P(q) * P(q|d)$
 - $P(d)$ is a prior on document. We can consider it as uniform, i.e., constant
 - $P(q)$ is the same for all the document
- We can conclude that $P(d|q) \sim P(q|d)$
 - Generative model for queries given a document



Estimating $P(q|M_d)$

- Given the LM for d , the formula then becomes $P(q|M_d)$ and we use multimodal unigram LM
 - $P(q|M_d) = K_q \prod_{t \in V} P(t|M_d)$
 - K_q is a normalizing factor constant for each query q , and we can ignore it
- $P(t|M_d)$ can be estimated using MLE
 - $\prod_{t \in V} P(t|M_d) = \prod_{t \in q} P(t|M_d) = \prod_{t \in q} tf(t, d) / \text{len}(d)$
- Smoothing can be applied to reduce the impact of 0 probabilities, or...
 - $P(t|d) = \lambda P_{MLE}(t|M_d) + (1 - \lambda)P_{MLE}(t|M_c);$
 - M_c being the language model associated with the whole collection



Is that it?



SAPIENZA
UNIVERSITÀ DI ROMA

Neural Language Models



SAPIENZA
UNIVERSITÀ DI ROMA

The Key Idea

- Model the problem as a discriminative learning task
 - $P(\text{word} | \text{context}; \theta)$
- Predict the occurrence of a word w_i given a set of words C_j , as

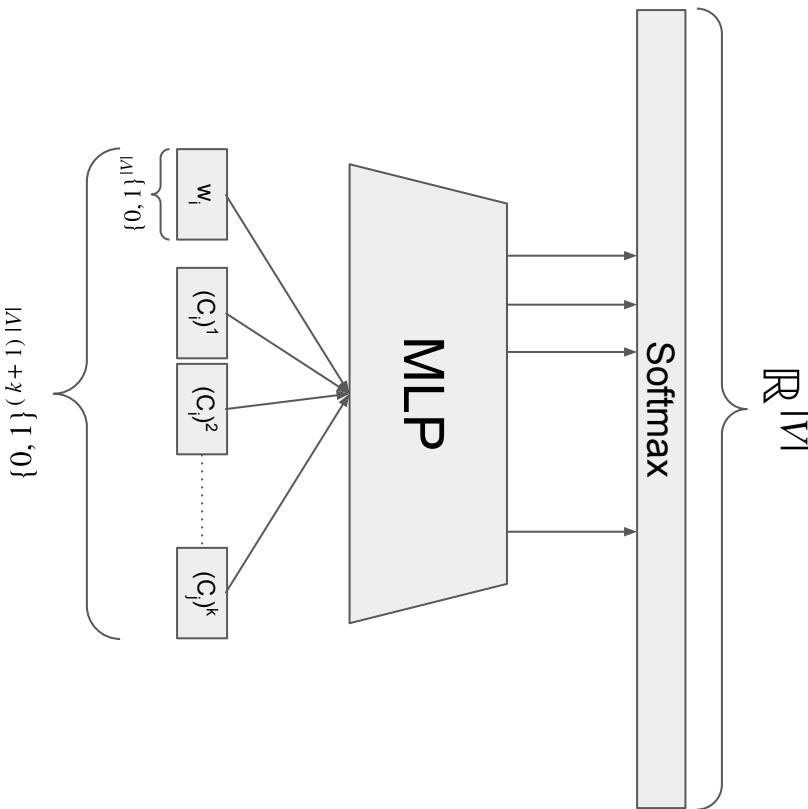
$$\circ \quad P(w_i | C_j) = \frac{e^{\vartheta_{w_i}} \cdot \vartheta_{C_j}}{\sum_{w' \in V} e^{\vartheta_{w'}} \cdot \vartheta_{C_j}}$$

$\text{softmax}(\vec{z})_i = \frac{e^{z_i}}{\sum_{1 \leq j \leq K} e^{z_j}}$

$$\circ \quad P(- | C_j) = \text{softmax}(\vartheta_{w_1} \cdot \vartheta_{C_j}, \vartheta_{w_2} \cdot \vartheta_{C_j}, \dots, \vartheta_{w_{|V|}} \cdot \vartheta_{C_j})$$



Simple Idea: Use an MLP



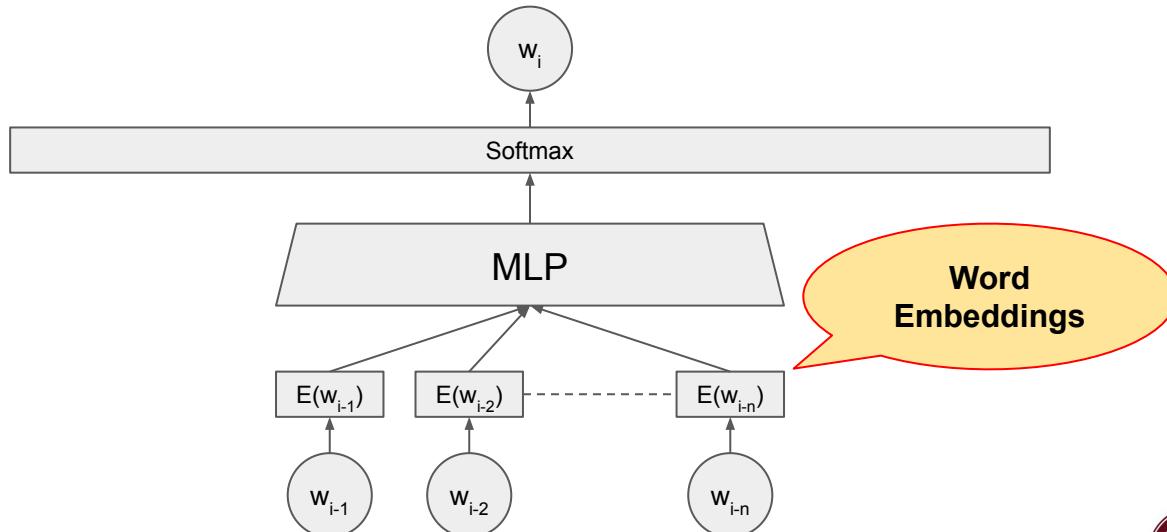
- Input is a concatenation of w with each context word $(C_j)^j$
 - 1-hot encoding: overall $(k+1)|V|$ (bits)
- Output is $|V|$ real numbers
- The MLP, then, uses a matrix of $(k+1)|V|^2 + |V|$ parameters.
 - E.g., for a (very small) vocabulary of 10^3 words, and a context of 10 words we will need $11*10^6+10^3 \sim 10^7$ parameters for a single layer MLP.
 - If $|V| = 10^5$ (more realistic), number of parameters is $\sim 10^{11}$



Word Embeddings (a quick intro)

Let C, the context, be the “sequence” of n preceding tokens to a word w_i
We want to compute $P(w_i | w_{i-1}, w_{i-2}, \dots, w_{i-n})$

```
52 def forward(self, input):
53     emb = self.embedding(input)
54     x = emb.reshape(emb.size()[0], self.embedding_dim * self.context_size)
55     x = self.mlp(x)
56
57     return x
```

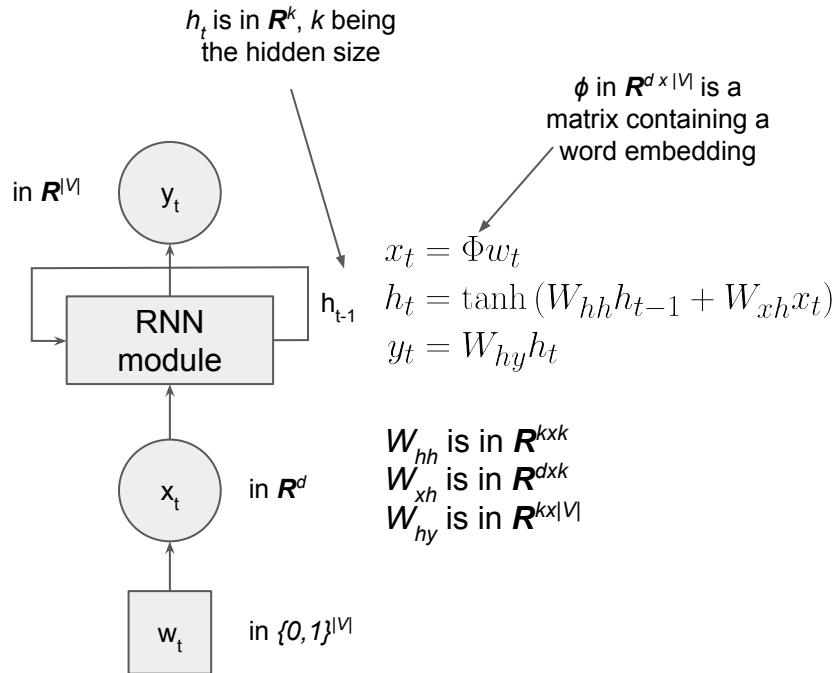


Recurrent Neural Network LMs



SAPIENZA
UNIVERSITÀ DI ROMA

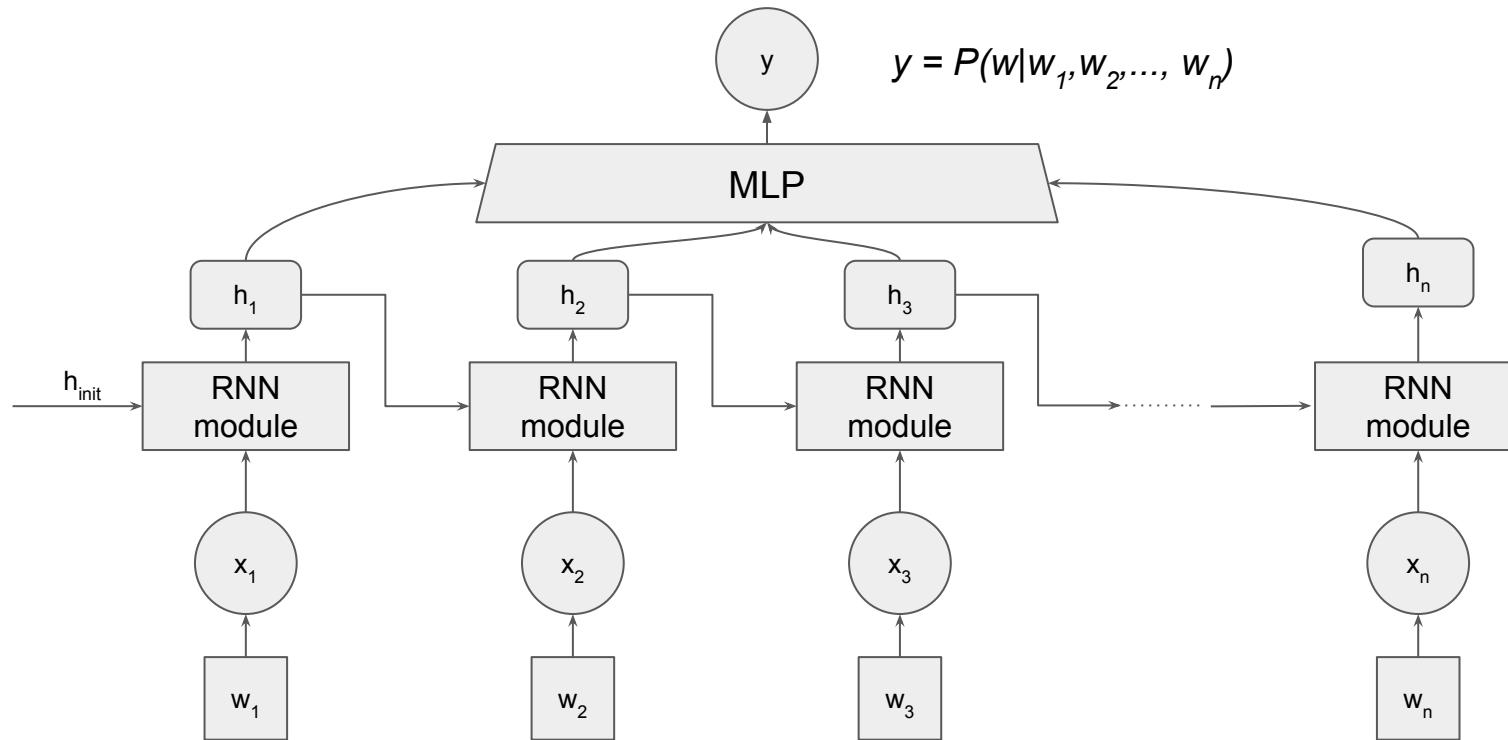
What is an RNN?



- Total number of parameters:
 - $d|V|+dk+k^2+k|V| = (d+k)|V| + (d+k)k = (d+k)(|V|+k)$
- E.g., $|V| = 10^5$, $d=10^2$, $k=10^2$
 - Total parameters = $10^7 \ll 10^{11}$



“Unrolling” an RNN: Training an LM



Backpropagation Through Time
(BTT)



SAPIENZA
UNIVERSITÀ DI ROMA

PyTorch (Lightning) Model

```
1 class RNNModel(pl.LightningModule):
2     def __init__(self, num_tokens, config_params):
3         super(RNNModel, self).__init__()
4         self.config_params = config_params
5
6         self.embedding = nn.Embedding(
7             num_embeddings=num_tokens,
8             embedding_dim=self.config_params.embedding_dim,
9             _weight=self.init_embedding(
10                 num_tokens,
11                 self.config_params.embedding_dim
12             )
13         )
14
15         self.rnn = nn.RNN(
16             self.config_params.embedding_dim,
17             self.config_params.rnn_hidden_dim,
18             self.config_params.num_layers,
19             batch_first=True
20         )
21         self.mlp = nn.Sequential(
22             nn.Linear(
23                 self.config_params.rnn_hidden_dim * self.config_params.num_layers,
24                 self.config_params.hidden1_dim
25             ),
26             nn.ReLU(),
27             nn.Dropout(self.config_params.dropout_probability),
28             nn.Linear(self.config_params.hidden1_dim, num_tokens)
29         )
30
31     def forward(self, input):
32         h = self.init_hidden(input.size(0))
33
34         x = self.embedding(input)
35         out, h = self.rnn(x, h)
36         x = h.transpose(0, 1)
37         x = x.reshape(
38             x.size()[0],
39             self.config_params.rnn_hidden_dim * self.config_params.num_layers
40         )
41         x = self.mlp(x)
42
43         return x
44
```



Examples

- [MLP Language Model](#)
- [RNN Language Model](#)

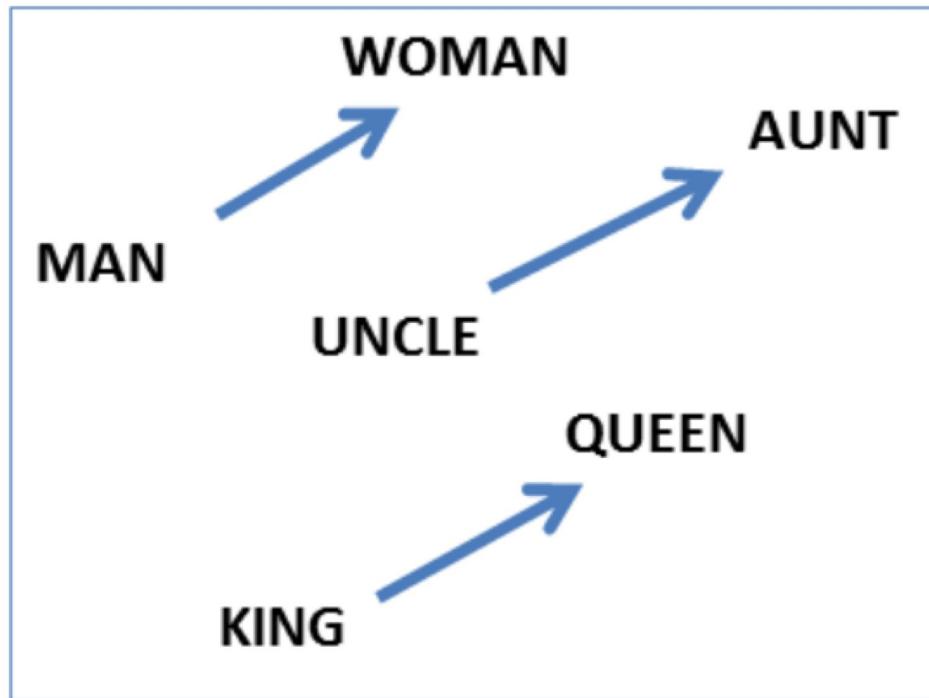


Word2Vec

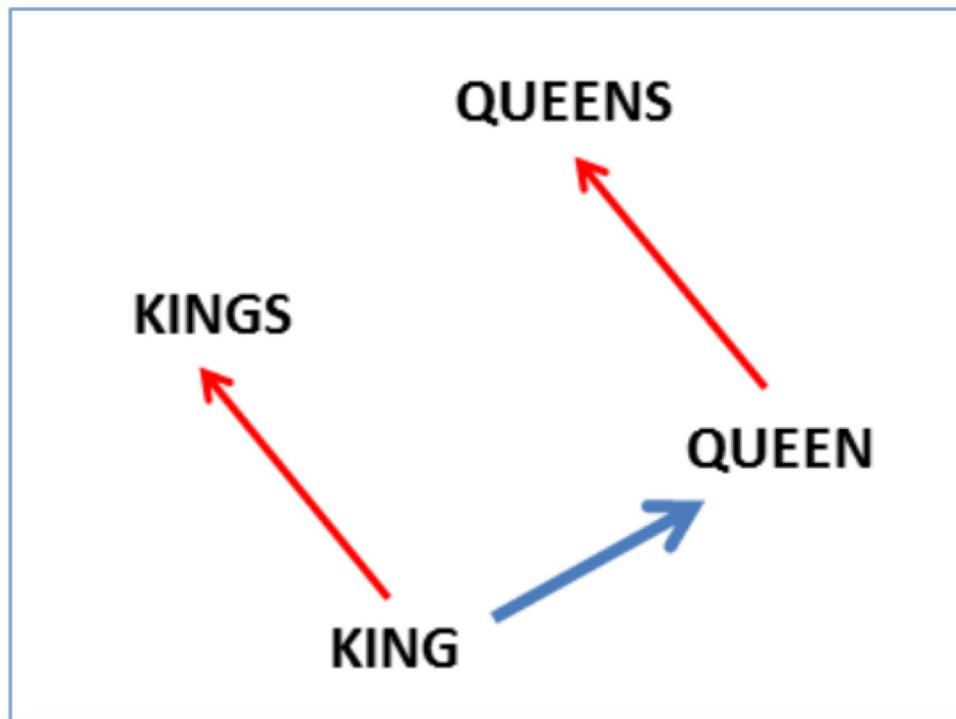
- Computing softmax for large vocabularies is expensive (typically $O(|V|^2)$)
- Use Noise Contrastive Estimation (or Negative Sampling) to “cast” the problem into a classification problem $P(w_i | \text{context}_i)$ into $P(D=1 | w_i, \text{context}_i)$
- How to pick D=0 cases?
 - Randomly sample $w_i, \text{context}_i$ pairs according to a given distribution
 - For each positive, sample 5~10 negatives
 - Based on: Gutmann and Hyvärinen. [Noise-contrastive estimation: A new estimation principle for unnormalized statistical models](#). JMLR, 13:307-361, 2012.
 - Dyer’s [notes](#) on differences between NCE and NS.



Word2Vec: Some Nice Properties



Word2Vec: Some Nice Properties



Word2Vec: Some Nice Properties

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza



Word2Vec: Some Nice Properties

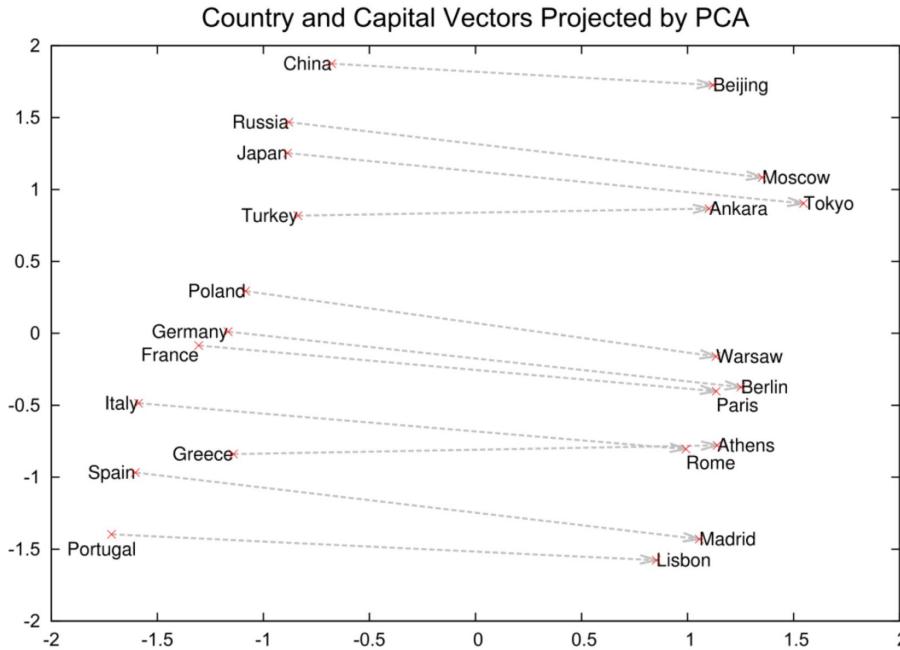


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.



Word2Vec: Some Nice Properties

monitor
laptop xbox
airplane keyboard tv stand
teddy sink
dinosaur range hood
snake hand
ant stool chair
spider plant tent
bird flower pot lamp
person vase bathtub
dolphin pliers toilet
octopus fish night stand
crab bowl dresser shelf
mantel desk bench
cone bottle glass box
radio cup door stairs
piano guitar wardrobe spectacle
curtain



After Word2Vec

- GloVe
- FastText
- Misspelling Oblivious Embeddings
- ...



Evaluating Language Models

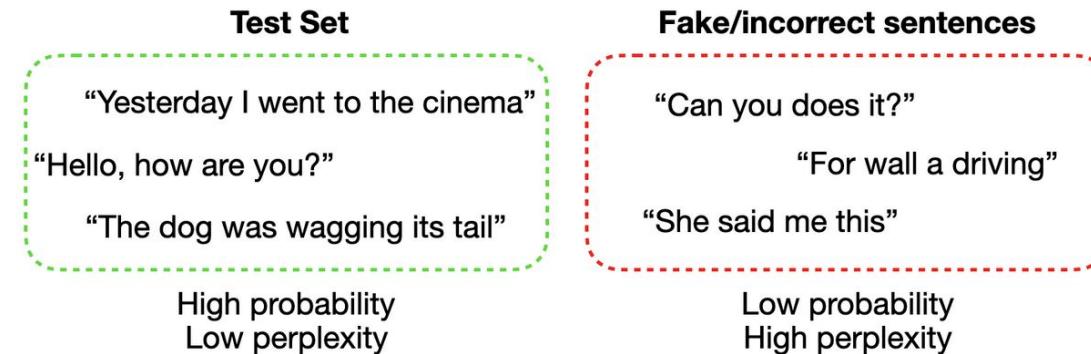
- *Intrinsic Evaluation*
 - Metrics to evaluate LMs in isolation, not taking into account the specific tasks it's going to be used for.
- *Extrinsic Evaluation*
 - Employing LMs in actual tasks (such as machine translation) and looking at their final loss/accuracy.



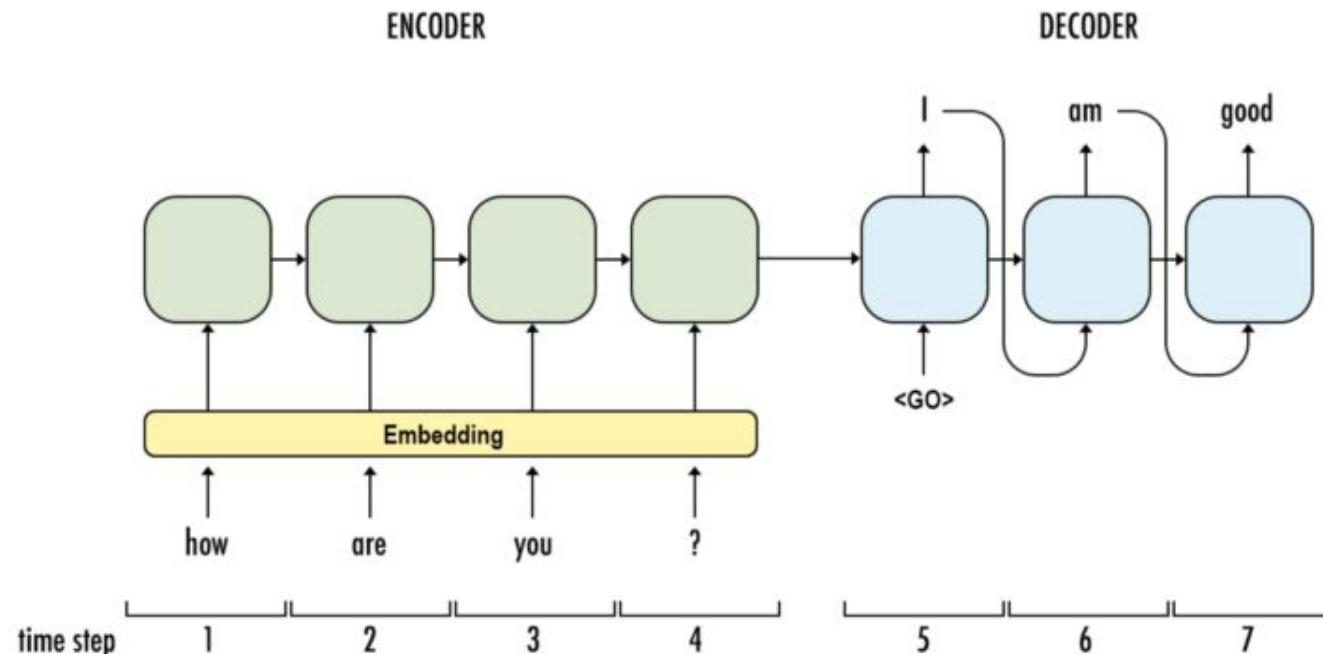
Intrinsic Metric: Perplexity

- It evaluates the normalised inverse probability of the test set

$$PP(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, \dots, w_N)}}$$



Encoder Decoder Architectures



Modern Language Models: Transformers

- Vaswani, Ashish, et al. "Attention is All you Need." NIPS. 2017.

```
class EncoderDecoder(nn.Module):
    """
    A standard Encoder-Decoder architecture. Base for this and many
    other models.
    """
    def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
        super(EncoderDecoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.src_embed = src_embed
        self.tgt_embed = tgt_embed
        self.generator = generator

    def forward(self, src, tgt, src_mask, tgt_mask):
        "Take in and process masked src and target sequences."
        return self.decode(self.encode(src, src_mask), src_mask,
                           tgt, tgt_mask)

    def encode(self, src, src_mask):
        return self.encoder(self.src_embed(src), src_mask)

    def decode(self, memory, src_mask, tgt, tgt_mask):
        return self.decoder(self.tgt_embed(tgt), memory, src_mask, tgt_mask)
```

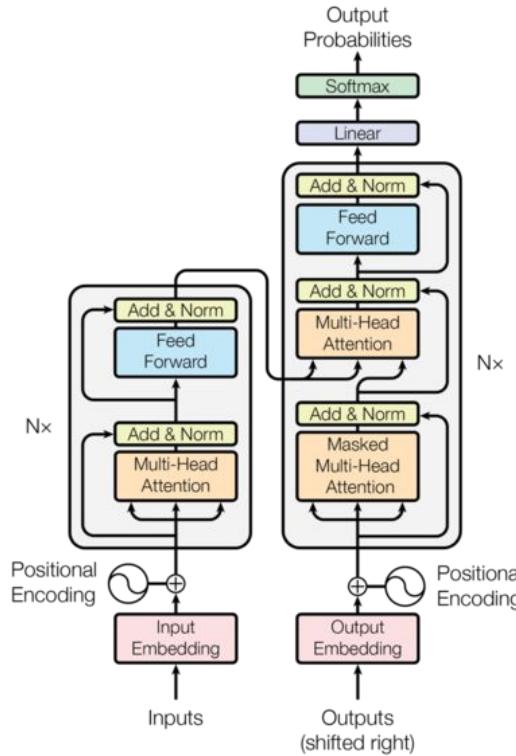
```
class Generator(nn.Module):
    "Define standard linear + softmax generation step."
    def __init__(self, d_model, vocab):
        super(Generator, self).__init__()
        self.proj = nn.Linear(d_model, vocab)

    def forward(self, x):
        return F.log_softmax(self.proj(x), dim=-1)
```



Modern Language Models: Transformers

- Vaswani, Ashish, et al. "Attention is All you Need." NIPS. 2017.



Transformers: Encoder

```
def clones(module, N):
    "Produce N identical layers."
    return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
```

```
class Encoder(nn.Module):
    "Core encoder is a stack of N layers"
    def __init__(self, layer, N):
        super(Encoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, mask):
        "Pass the input (and mask) through each layer in turn."
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)
```

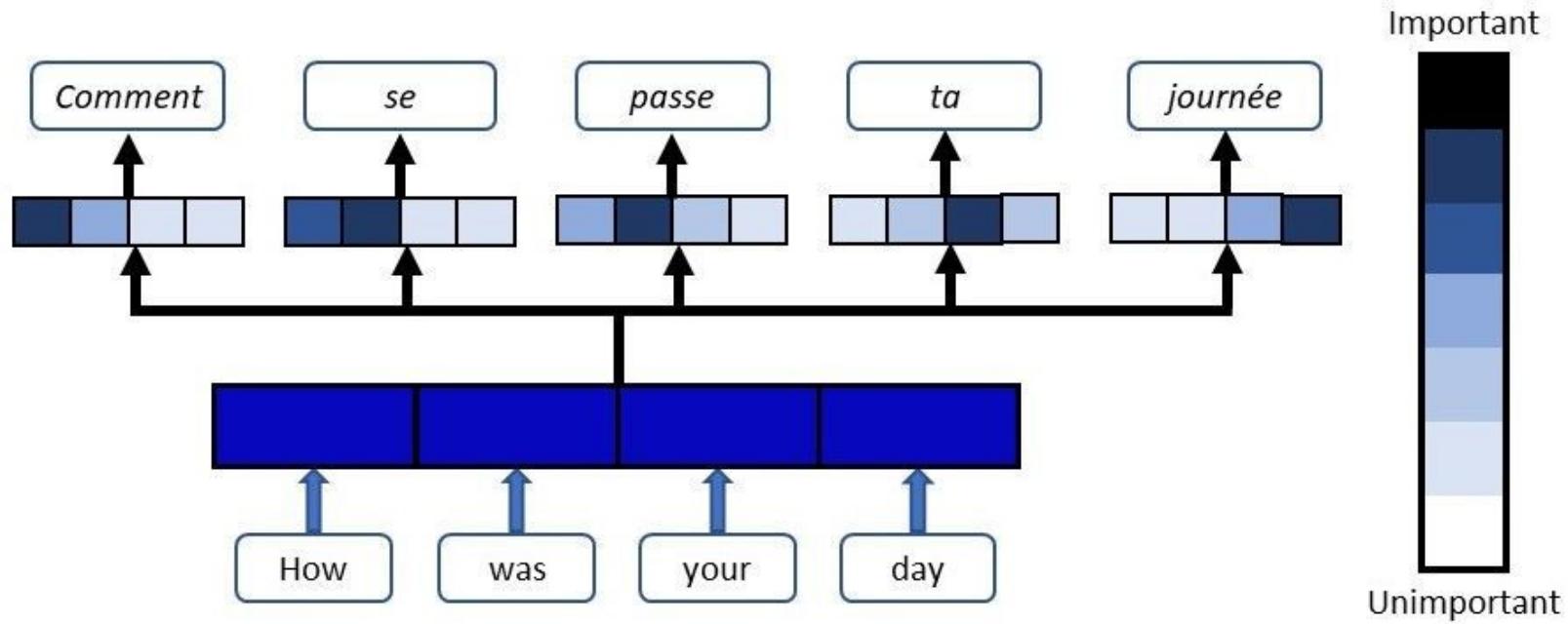
We employ a residual connection ([cite](#)) around each of the two sub-layers, followed by layer normalization ([cite](#)).

```
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

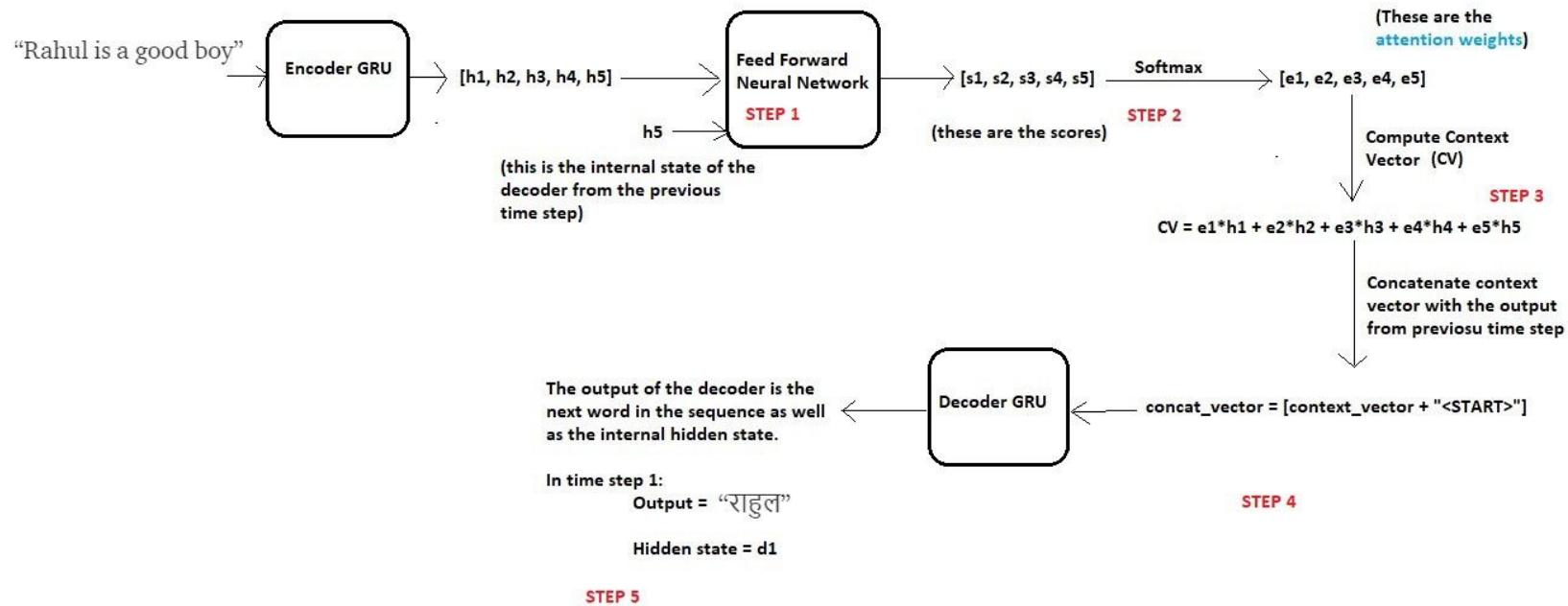
    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```



Attention Mechanism



Attention Mechanism



Transformers: Encoder

- That is, the output of each sub-layer is $\text{LayerNorm}(x + \text{SubLayer}(x))$
 - SubLayer(x) implements the sub-layer that we are about to describe

```
class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """

    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))
```



Transformers: Encoder

- Each layer has two sub-layers:
 - multi-head self-attention mechanism
 - position-wise fully connected feed-forward network

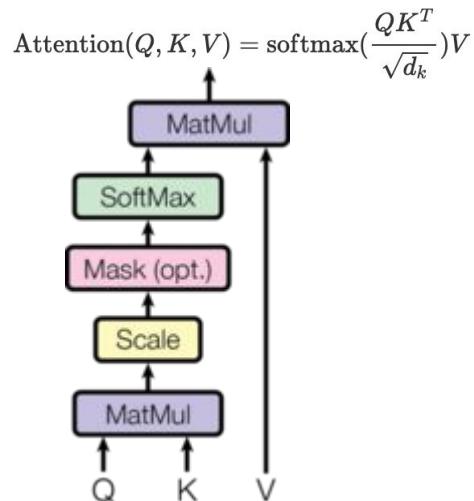
```
class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"
    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)
```



Transformers: Attention

- It maps a query and a set of key-value pairs to an output
 - where the query, keys, values, and output are all vectors.
- The output is computed as a weighted sum of the values
 - the weight assigned to each value is computed by a **compatibility** function of the query with the corresponding key.



```
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
        / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = F.softmax(scores, dim = -1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```



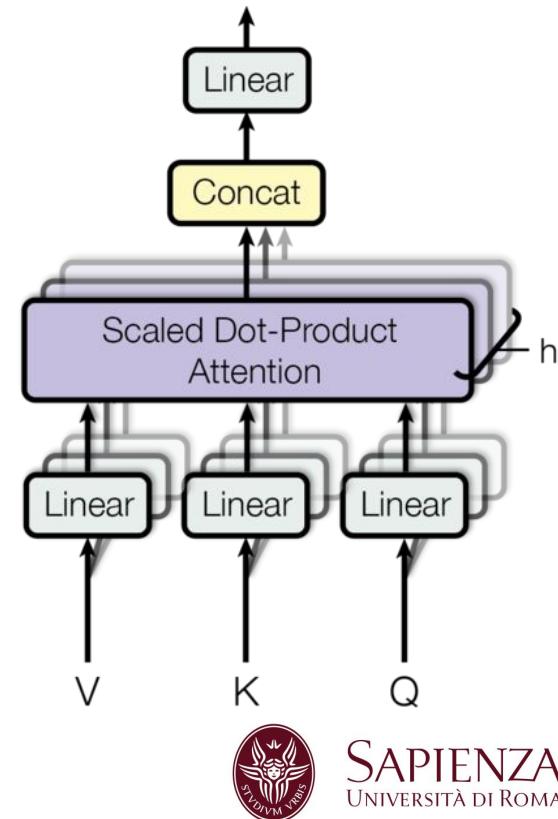
Transformers: Multi-Head Attention

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions.
- With a single attention head, averaging inhibits this.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where } \text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v} \text{ and } W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$$



Multi-Head Attention in PyTorch

```
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        "Implements Figure 2"
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)

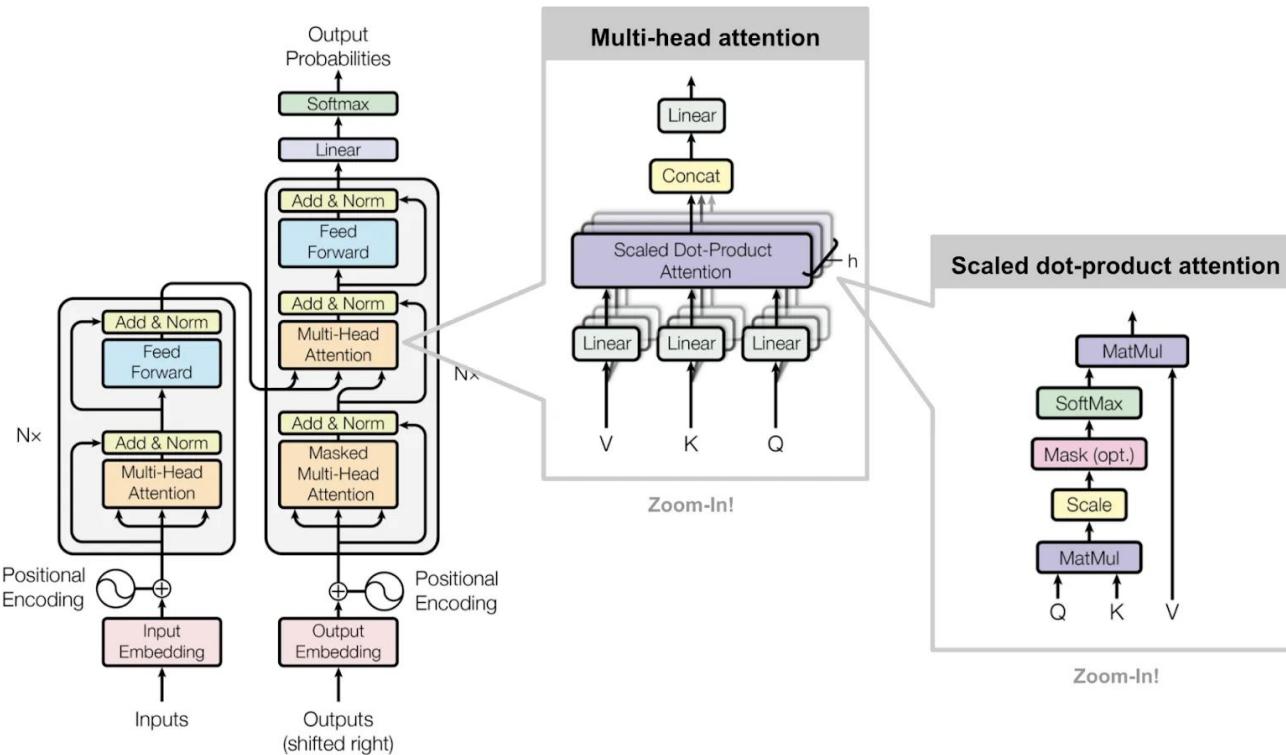
        # 1) Do all the linear projections in batch from d_model => h x d_k
        query, key, value = \
            [l(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
             for l, x in zip(self.linears, (query, key, value))]

        # 2) Apply attention on all the projected vectors in batch.
        x, self.attn = attention(query, key, value, mask=mask,
                                 dropout=self.dropout)

        # 3) "Concat" using a view and apply a final linear.
        x = x.transpose(1, 2).contiguous() \
            .view(nbatches, -1, self.h * self.d_k)
        return self.linears[-1](x)
```



The Full Picture

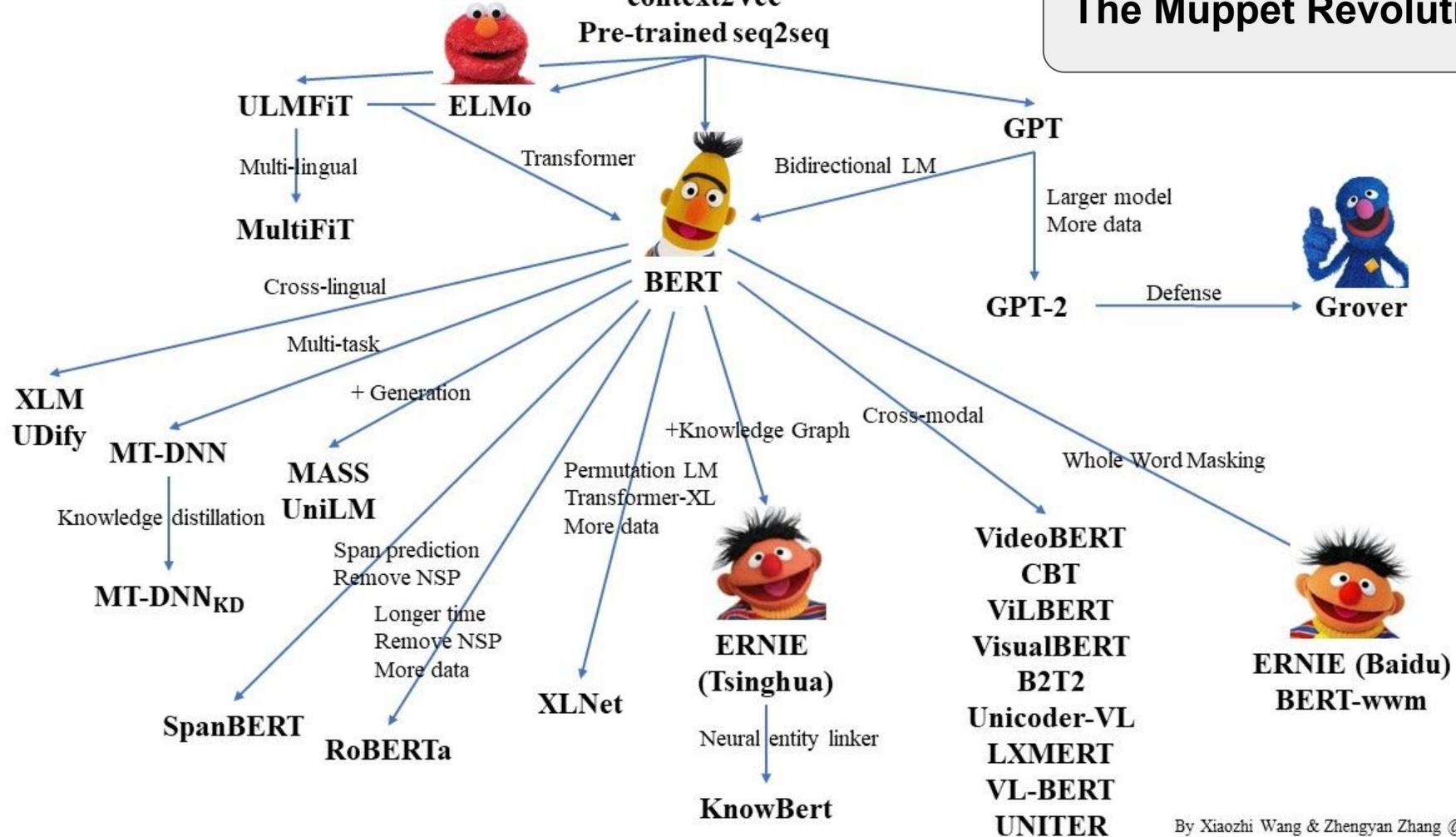


Semi-supervised Sequence Learning

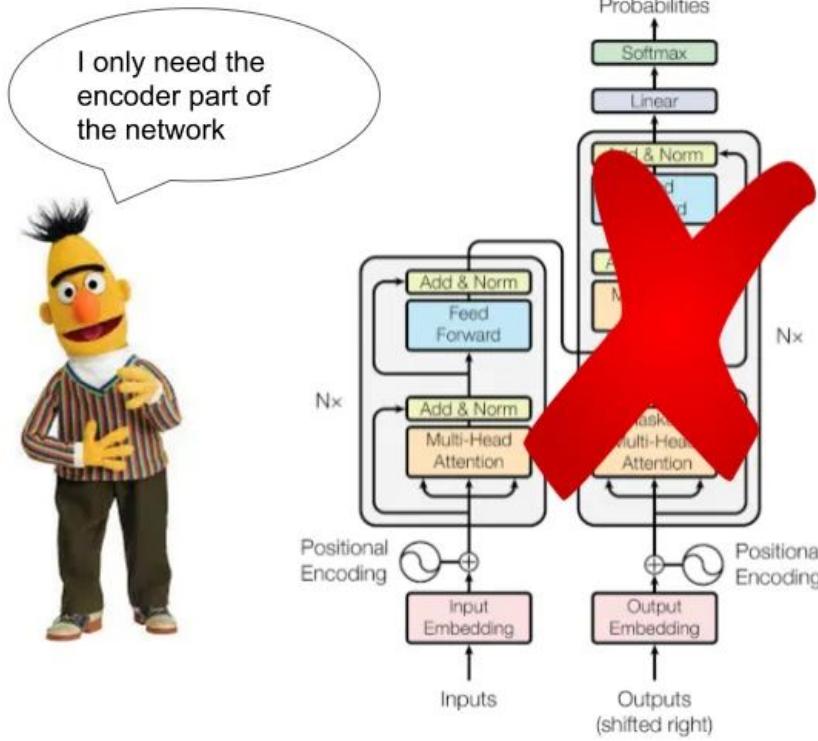
context2Vec

Pre-trained seq2seq

The Muppet Revolution



BERT: Bidirectional Encoder Representations from Transformers



I only need the
encoder part of
the network



SAPIENZA
UNIVERSITÀ DI ROMA

A Colab Example

1. <https://colab.research.google.com/drive/1XqA6oMGaJvmHdC4Mlyz1gUs10cUY-w-f>
2. <https://colab.research.google.com/drive/12NWHoUjmZjVtxYD4RAipU7b5Qx8yHaHp>

Introduction to **Information Retrieval**

Probabilistic Information Retrieval
Christopher Manning and Pandu Nayak

From Boolean to Ranked Retrieval

1. Why ranked retrieval?
2. Introduction to the classical probabilistic retrieval model and the probability ranking principle
3. The Binary Independence Model: BIM
4. Relevance feedback, briefly
5. The vector space model (VSM) (quick cameo)
6. BM25 model
7. Ranking with features: BM25F (if time allows ...)

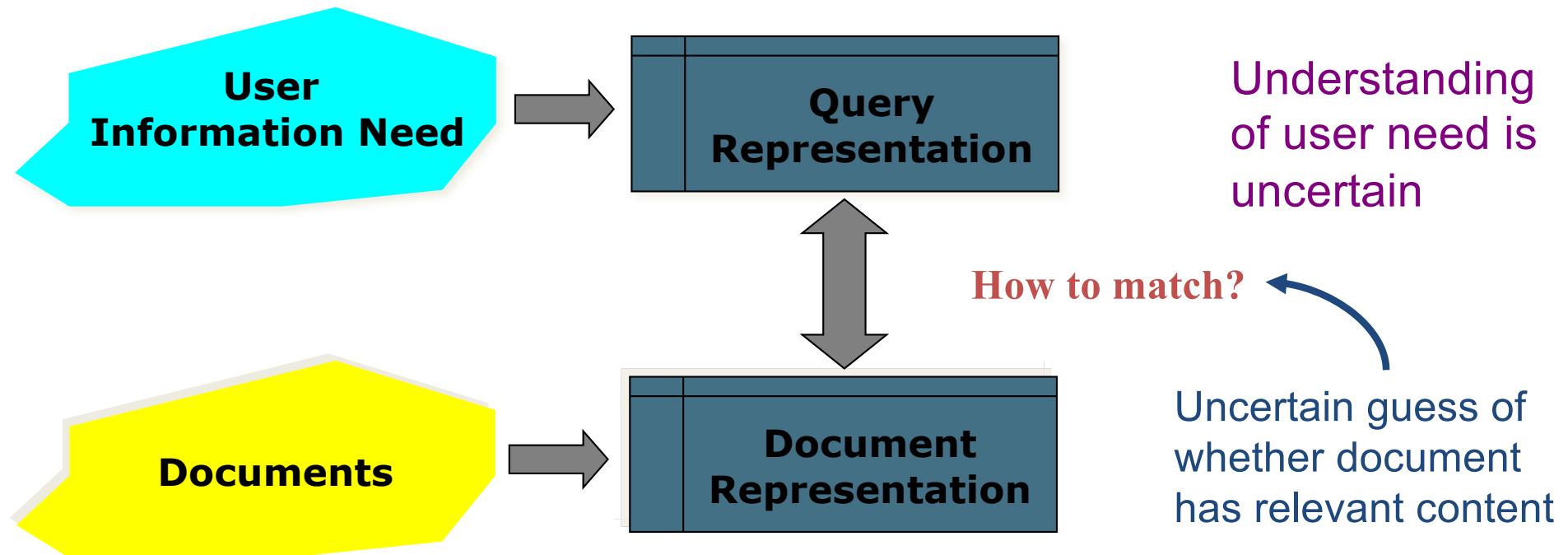
1. Ranked retrieval

- Thus far, our queries have all been Boolean
 - Documents either match or don't
- Can be good for expert users with precise understanding of their needs and the collection
 - Can also be 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*”: 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
- Suggested solution:
 - Rank documents by goodness – a sort of clever “soft AND”

2. Why probabilities in IR?



In traditional IR systems, matching between each document and query is attempted in a semantically imprecise space of index terms.

Probabilities provide a principled foundation for uncertain reasoning.
Can we use probabilities to quantify our search uncertainties?

Probabilistic IR topics

1. Classical probabilistic retrieval model

- Probability ranking principle, etc.
- Binary independence model (\approx Naïve Bayes text cat)
- (Okapi) BM25

2. Bayesian networks for text retrieval

3. Language model approach to IR (IIR ch. 12)

- An important development in 2000s IR

Probabilistic methods are one of the oldest but also one of the currently hot topics in IR

- *Traditionally: neat ideas, but didn't win on performance*
- *It seems to be different now*

Who are these people?



Karen Spärck Jones



Stephen Robertson



Keith van Rijsbergen

The document ranking problem

- We have a collection of documents
- User issues a query
- A list of documents needs to be returned
- **Ranking method is the core of modern IR systems:**
 - **In what order do we present documents to the user?**
 - We want the “best” document to be first, second best second, etc.
- **Idea: Rank by probability of relevance of the document w.r.t. information need**
 - $P(R=1 | \text{document}_i, \text{query})$

The Probability Ranking Principle (PRP)

“If a ~~reference~~ retrieval system’s response to each ~~request~~ is a ranking of the documents ~~in the collection~~ in order of decreasing probability of relevance to the user ~~who submitted the request~~, where the probabilities are estimated ~~as accurately as possible~~ on ~~the basis of~~ whatever data ~~have been made available to the system~~ for this purpose, the overall effectiveness of ~~the system to its user~~ will be the best that is obtainable on the basis of those data.”

- [1960s/1970s] S. Robertson, W.S. Cooper, M.E. Maron; van Rijsbergen (1979:113); Manning & Schütze (1999:538)

Recall a few probability basics

- For events A and B :

$$p(A, B) = p(A \cap B) = p(A | B)p(B) = p(B | A)p(A)$$

- Bayes' Rule

$$p(A | B) = \frac{p(B | A)p(A)}{p(B)} = \frac{p(B | A)p(A)}{\sum_{X=A, \bar{A}} p(B | X)p(X)}$$

↑ Posterior ↑ Prior

- Odds:

$$O(A) = \frac{p(A)}{p(\bar{A})} = \frac{p(A)}{1 - p(A)}$$

The Probability Ranking Principle (PRP)

Let x represent a document in the collection.

Let R represent relevance of a document w.r.t. given (fixed) query and let $R=1$ represent relevant and $R=0$ not relevant.

Need to find $p(R=1|x)$ – probability that a document x is relevant.

$$p(R=1|x) = \frac{p(x|R=1)p(R=1)}{p(x)}$$

$$p(R=0|x) = \frac{p(x|R=0)p(R=0)}{p(x)}$$

$$p(R=0|x) + p(R=1|x) = 1$$

$p(R=1), p(R=0)$ - prior probability of retrieving a relevant or non-relevant document at random

$p(x|R=1), p(x|R=0)$ - probability that if a relevant (not relevant) document is retrieved, it is x .

Probabilistic Retrieval Strategy

- First, estimate how each term contributes to relevance
 - How do other things like term frequency and document length influence your judgments about document relevance?
 - Not at all in BM1
 - A more nuanced answer is given by BM25
- Combine to find document relevance probability
- Order documents by decreasing probability
- Theorem: Using the PRP is optimal, in that it minimizes the loss (Bayes risk) under 1/0 loss
 - Provable if all probabilities correct, etc. [e.g., Ripley 1996]

3. Binary Independence Model

- Traditionally used in conjunction with PRP
- “**Binary**” = **Boolean**: documents are represented as binary incidence vectors of terms (cf. IIR Chapter 1):
 - $\vec{x} = (x_1, \dots, x_n)$
 - $x_i = 1$ iff term i is present in document x .
- “**Independence**”: terms occur in documents independently
- Different documents can be modeled as the same vector ?

Binary Independence Model

- Queries: binary term incidence vectors
- Given query q ,
 - for each document d need to compute $p(R|q,d)$
 - replace with computing $p(R|q,x)$ where x is binary term incidence vector representing d
 - Interested only in ranking
- Will use odds and Bayes' Rule:

$$O(R|q, \vec{x}) = \frac{p(R=1|q, \vec{x})}{p(R=0|q, \vec{x})} = \frac{\frac{p(R=1|q)p(\vec{x}|R=1, q)}{p(\vec{x}|q)}}{\frac{p(R=0|q)p(\vec{x}|R=0, q)}{p(\vec{x}|q)}}$$

Binary Independence Model

$$O(R \mid q, \vec{x}) = \frac{p(R = 1 \mid q, \vec{x})}{p(R = 0 \mid q, \vec{x})} = \frac{p(R = 1 \mid q)}{p(R = 0 \mid q)} \cdot \frac{p(\vec{x} \mid R = 1, q)}{p(\vec{x} \mid R = 0, q)}$$

Constant for a
given query

Needs estimation

- Using **Independence Assumption**:

$$\frac{p(\vec{x} \mid R = 1, q)}{p(\vec{x} \mid R = 0, q)} = \prod_{i=1}^n \frac{p(x_i \mid R = 1, q)}{p(x_i \mid R = 0, q)}$$

$$O(R \mid q, \vec{x}) = O(R \mid q) \cdot \prod_{i=1}^n \frac{p(x_i \mid R=1, q)}{p(x_i \mid R=0, q)}$$

Binary Independence Model

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{i=1}^n \frac{p(x_i | R = 1, q)}{p(x_i | R = 0, q)}$$

- Since x_i is either 0 or 1:

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{x_i=1} \frac{p(x_i = 1 | R = 1, q)}{p(x_i = 1 | R = 0, q)} \cdot \prod_{x_i=0} \frac{p(x_i = 0 | R = 1, q)}{p(x_i = 0 | R = 0, q)}$$

- Let $p_i = p(x_i = 1 | R = 1, q)$; $r_i = p(x_i = 1 | R = 0, q)$;

- Assume, for all terms not occurring in the query ($q_i = 0$) $p_i = r_i$

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{\substack{x_i=1 \\ q_i=1}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i=0 \\ q_i=1}} \frac{(1 - p_i)}{(1 - r_i)}$$

	document	relevant (R=1)	not relevant (R=0)
term present	$x_i = 1$	p_i	r_i
term absent	$x_i = 0$	$(1 - p_i)$	$(1 - r_i)$

Binary Independence Model

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{\substack{x_i = q_i = 1 \\ \text{All matching terms}}} \frac{p_i}{r_i} \cdot \prod_{\substack{x_i = 0 \\ q_i = 1}} \frac{1 - p_i}{1 - r_i}$$

All matching terms

$x_i = q_i = 1$

$x_i = 0$
 $q_i = 1$

Non-matching query terms

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{\substack{x_i = 1 \\ q_i = 1}} \frac{p_i}{r_i} \cdot \prod_{x_i = 1} \left(\frac{1 - r_i}{1 - p_i} \cdot \frac{1 - p_i}{1 - r_i} \right) \prod_{\substack{x_i = 0 \\ q_i = 1}} \frac{1 - p_i}{1 - r_i}$$

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{\substack{x_i = q_i = 1 \\ \text{All matching terms}}} \frac{p_i(1 - r_i)}{r_i(1 - p_i)} \cdot \prod_{q_i = 1} \frac{1 - p_i}{1 - r_i}$$

All matching terms

$x_i = q_i = 1$

$q_i = 1$

All query terms

Binary Independence Model

$$O(R | q, \vec{x}) = O(R | q) \cdot \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} \cdot \prod_{q_i=1} \frac{1-p_i}{1-r_i}$$

Constant for each query

Only quantity to be estimated for rankings

Retrieval Status Value:

$$RSV = \log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} = \sum_{x_i=q_i=1} \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

Binary Independence Model

[Robertson & Spärck-Jones 1976]

All boils down to computing RSV.

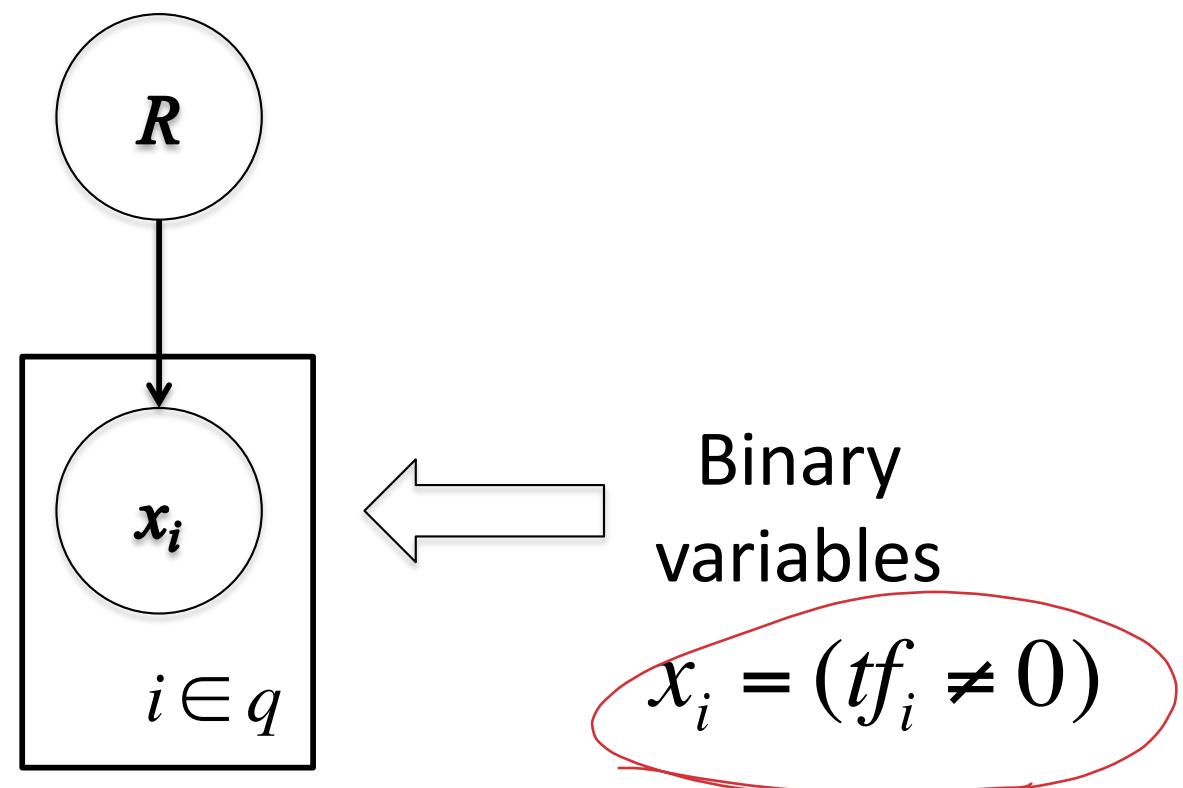
$$RSV = \log \prod_{x_i=q_i=1} \frac{p_i(1-r_i)}{r_i(1-p_i)} = \sum_{x_i=q_i=1} \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

$$RSV = \sum_{x_i=q_i=1} c_i; \quad c_i = \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

The c_i are **log odds ratios** (of contingency table a few slides back)
They function as the term weights in this model

So, how do we compute c_i 's from our data?

Graphical model for BIM – Bernoulli NB



Binary Independence Model

- Estimating RSV coefficients in theory
- For each term i look at this table of document counts:

$$c_i = \log \frac{p_i(1-r_i)}{r_i(1-p_i)}$$

Documents	Relevant	Non-Relevant	Total
$x_i=1$	s	$n-s$	n
$x_i=0$	$S-s$	$N-n-S+s$	$N-n$
Total	S	$N-S$	N

- Estimates:

$$p_i \approx \frac{s}{S} \quad r_i \approx \frac{(n-s)}{(N-S)}$$

$$c_i \approx K(N, n, S, s) = \log \frac{s/(S-s)}{(n-s)/(N-n-S+s)}$$

For now,
assume no
zero terms.
Remember
smoothing.

Estimation – key challenge

- If non-relevant documents are approximated by the whole collection, then r_i (prob. of occurrence in non-relevant documents for query) is n/N and

$$\log \frac{1 - r_i}{r_i} = \log \frac{N - n - S + s}{n - s} \approx \log \frac{N - n}{n} \approx \log \frac{N}{n} = IDF!$$

- Inverse Document Frequency (IDF)
 - Spärck-Jones (1972)
 - A key, still-important term weighting concept

Collection vs. Document frequency

- Collection frequency of t is the total number of occurrences of t in the collection (incl. multiples)
- Document frequency is number of docs t is in
- 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)?

Estimation – key challenge

- p_i (probability of occurrence in relevant documents) cannot be approximated as easily
- p_i can be estimated in various ways:
 - from relevant documents if you know some
 - Relevance weighting can be used in a feedback loop
 - constant (Croft and Harper combination match) – then just get idf weighting of terms (with $p_i=0.5$)
$$RSV = \sum_{x_i=q_i=1} \log \frac{N}{n_i}$$
 - proportional to prob. of occurrence in collection
 - Greiff (SIGIR 1998) argues for $1/3 + 2/3 df_i/N$

4. Probabilistic Relevance Feedback

1. Guess a preliminary probabilistic description of $R=1$ documents; use it to retrieve a set of documents
2. Interact with the user to refine the description: learn some definite members with $R = 1$ and $R = 0$
3. Re-estimate p_i and r_i on the basis of these
 - If i appears in V_i within set of documents V : $p_i = |V_i|/|V|$
 - Or can combine new information with original guess (use Bayesian prior):
$$p_i^{(2)} = \frac{|V_i| + \kappa p_i^{(1)}}{|V| + \kappa}$$
4. Repeat, thus generating a succession of approximations to relevant documents

κ is prior weight

Pseudo-relevance feedback (iteratively auto-estimate p_i and r_i)

1. Assume that p_i is constant over all x_i in query and r_i as before
 - $p_i = 0.5$ (even odds) for any given doc
2. Determine guess of relevant document set:
 - V is fixed size set of highest ranked documents on this model
3. We need to improve our guesses for p_i and r_i , so
 - Use distribution of x_i in docs in V . Let V_i be set of documents containing x_i
 - $p_i = |V_i| / |V|$
 - Assume if not retrieved then not relevant
 - $r_i = (n_i - |V_i|) / (N - |V|)$
4. Go to 2. until converges then return ranking

PRP and BIM

- It is possible to reasonably approximate probabilities
 - But either require partial relevance information or need to make do with somewhat inferior term weights
- Requires restrictive assumptions:
 - “Relevance” of each document is independent of others
 - Really, it’s bad to keep on returning **duplicates**
 - Term independence
 - Terms not in query don’t affect the outcome
 - Boolean representation of documents/queries
 - Boolean notion of relevance
- Some of these assumptions can be removed

Removing term independence

- In general, index terms aren't independent
 - "Hong Kong"
- Dependencies can be complex
- van Rijsbergen (1979) proposed simple model of dependencies as a tree
- Each term dependent on one other
 - Exactly Friedman and Goldszmidt's Tree Augmented Naive Bayes (AAAI 13, 1996)
- In 1970s, estimation problems held back success of this model

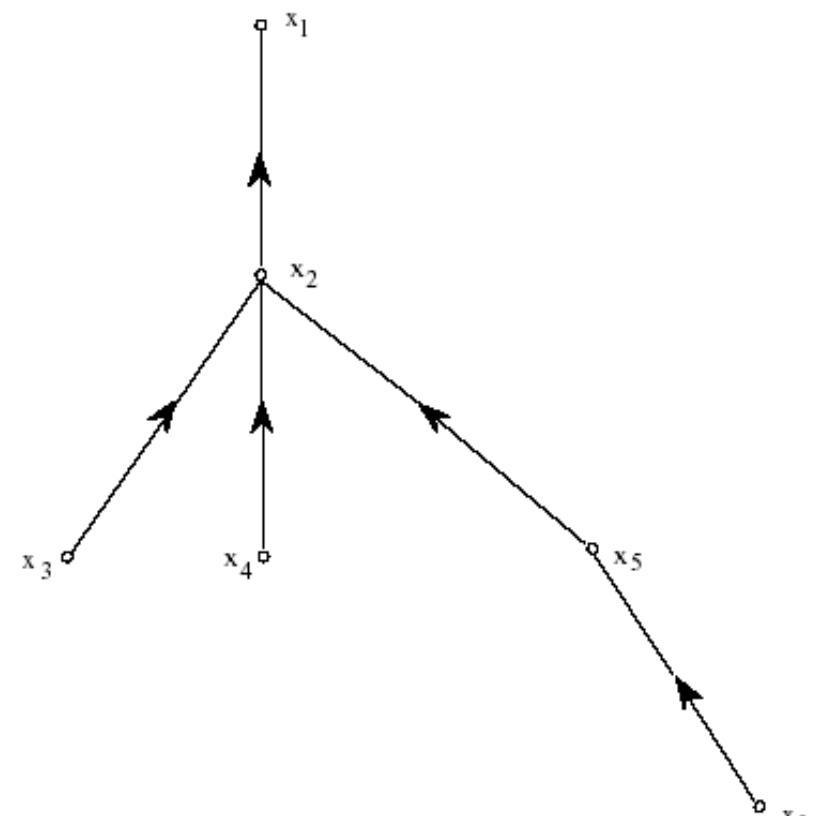


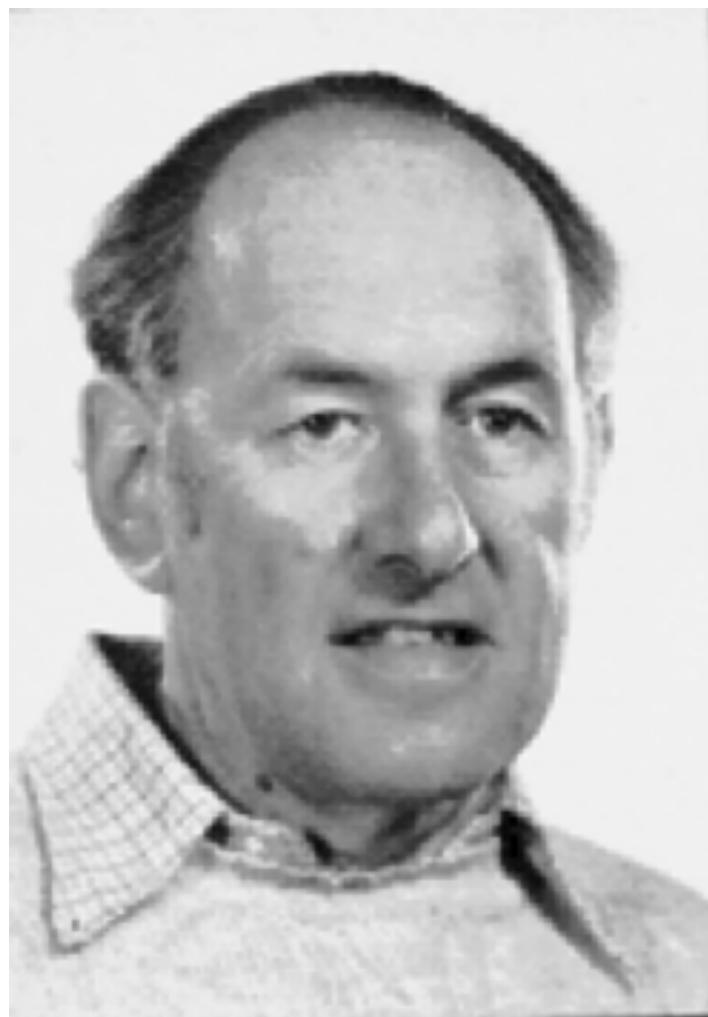
Figure 6.1.

5. Term frequency and the VSM

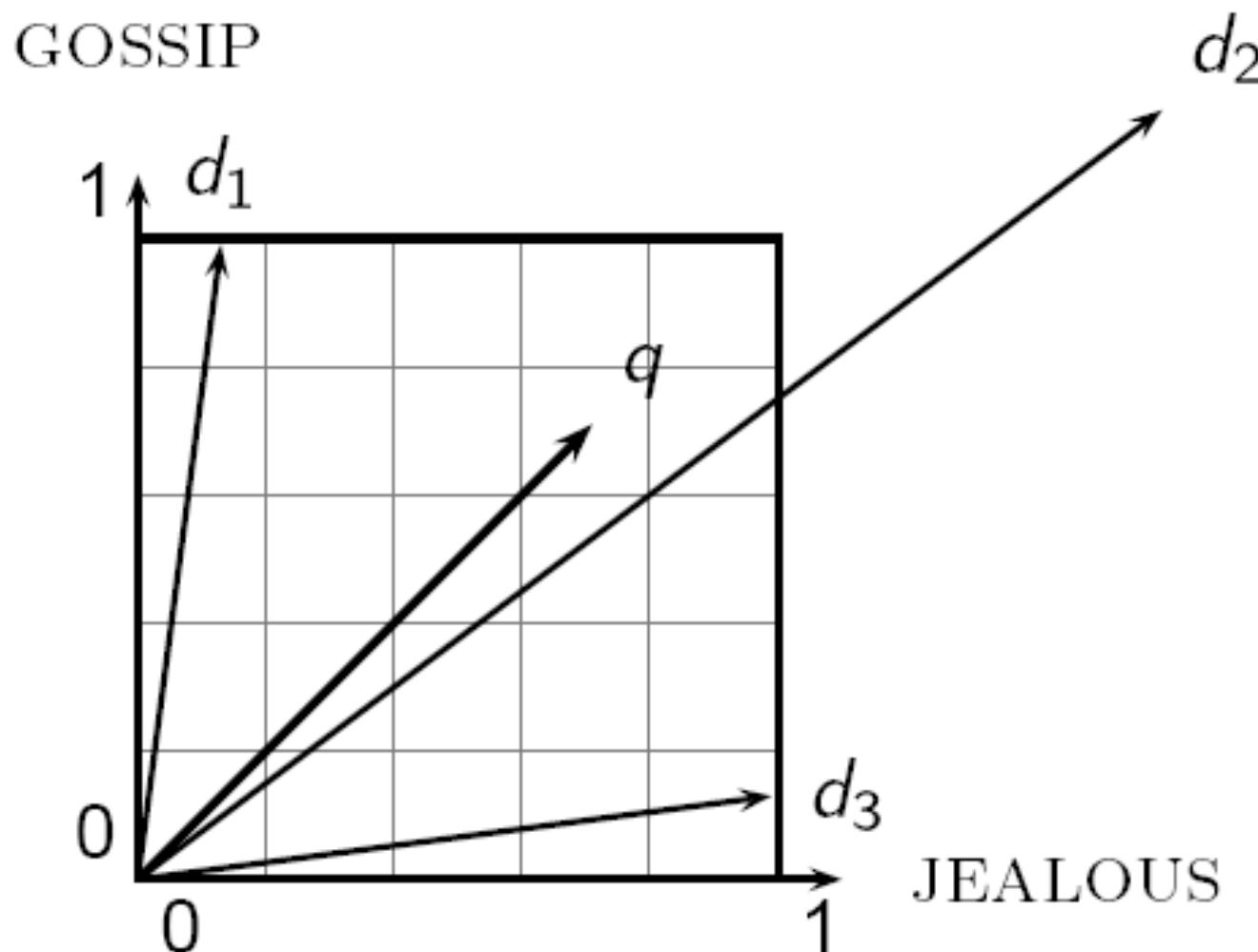
- Right in the first lecture, we said that a page should rank higher if it mentions a word more
 - Perhaps modulated by things like page length
- Why not in BIM? Much of early IR was designed for titles or abstracts, and not for modern full text search
- We now want a model with term frequency in it
- We'll mainly look at a probabilistic model (BM25)
- First, a quick summary of vector space model

Summary – vector space ranking (ch. 6)

- Represent the query as a weighted term frequency/inverse document frequency (tf-idf) vector
 - (0, 0, 0, 0, 2.3, 0, 0, 0, 1.78, 0, 0, 0, ..., 0, 8.17, 0, 0)
- Represent each document as a weighted tf-idf vector
 - (1.2, 0, 3.7, 1.5, 2.0, 0, 1.3, 0, 3.7, 1.4, 0, 0, ..., 3.5, 5.1, 0, 0)
- 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



Cosine similarity



tf-idf weighting has many variants

Term frequency	Document frequency	Normalization
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
I (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) $\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$		

6. BM25



BM25 The Next Generation of Lucene Relevance

Doug Turnbull – October 16, 2015

There's something new cooking in how Lucene scores text. Instead of the traditional "TF*IDF," Lucene just switched to something called BM25 in trunk. That means a new scoring formula for Solr ([Solr 6](#)) and Elasticsearch down the line.

Sounds cool, but what does it all mean? In this article I want to give you an overview of how the switch might be a boon to your Solr and Elasticsearch applications. What was the original TF*IDF? How did it work? What does the new BM25 do better? How do you tune it? Is BM25 right for everything?

Okapi BM25

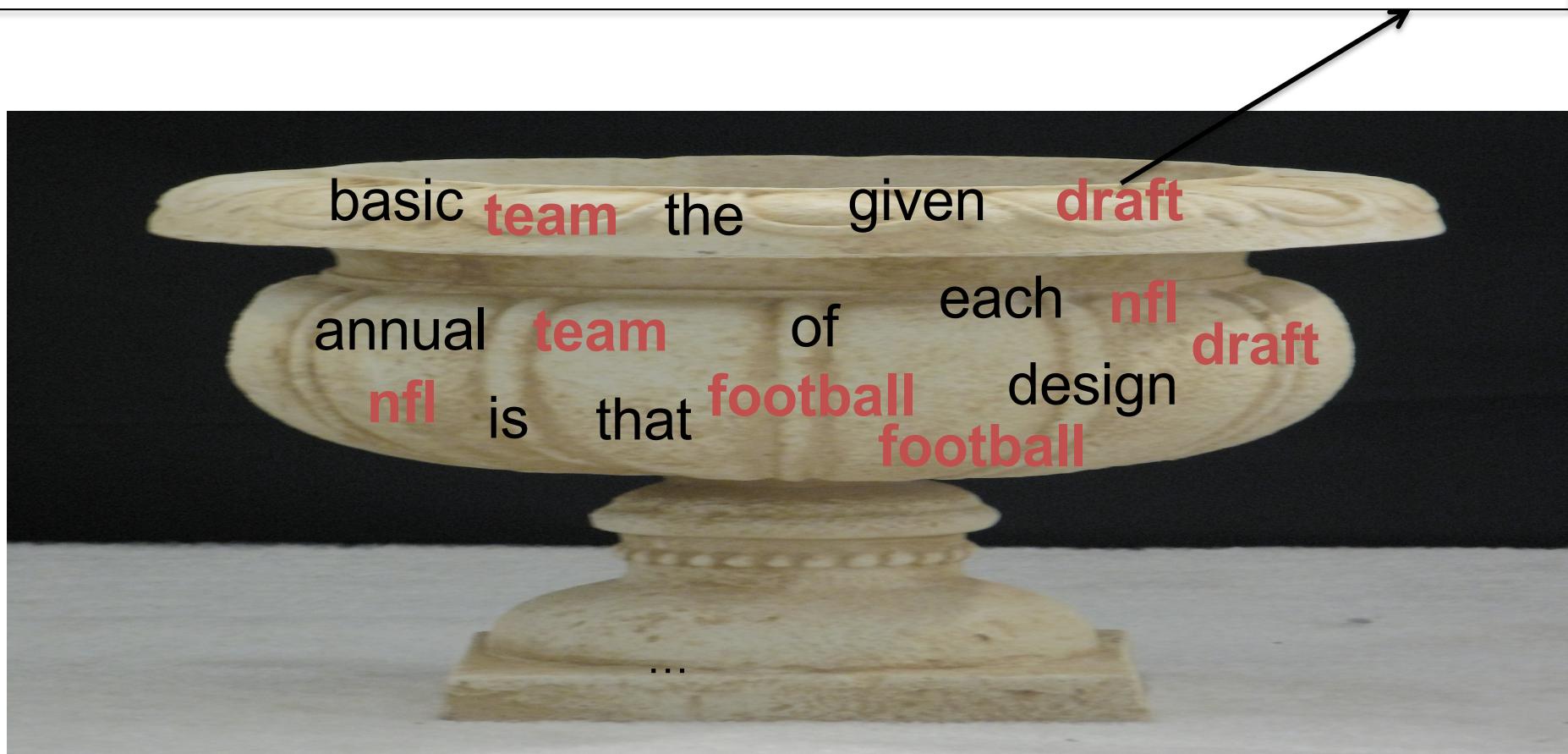
[Robertson et al. 1994, TREC City U.]

- BM25 “Best Match 25” (they had a bunch of tries!)
 - Developed in the context of the Okapi system
 - Started to be increasingly adopted by other teams during the TREC competitions
 - It works well
- Goal: be sensitive to term frequency and document length while not adding too many parameters
 - (Robertson and Zaragoza 2009; Spärck Jones et al. 2000)

Generative model for documents

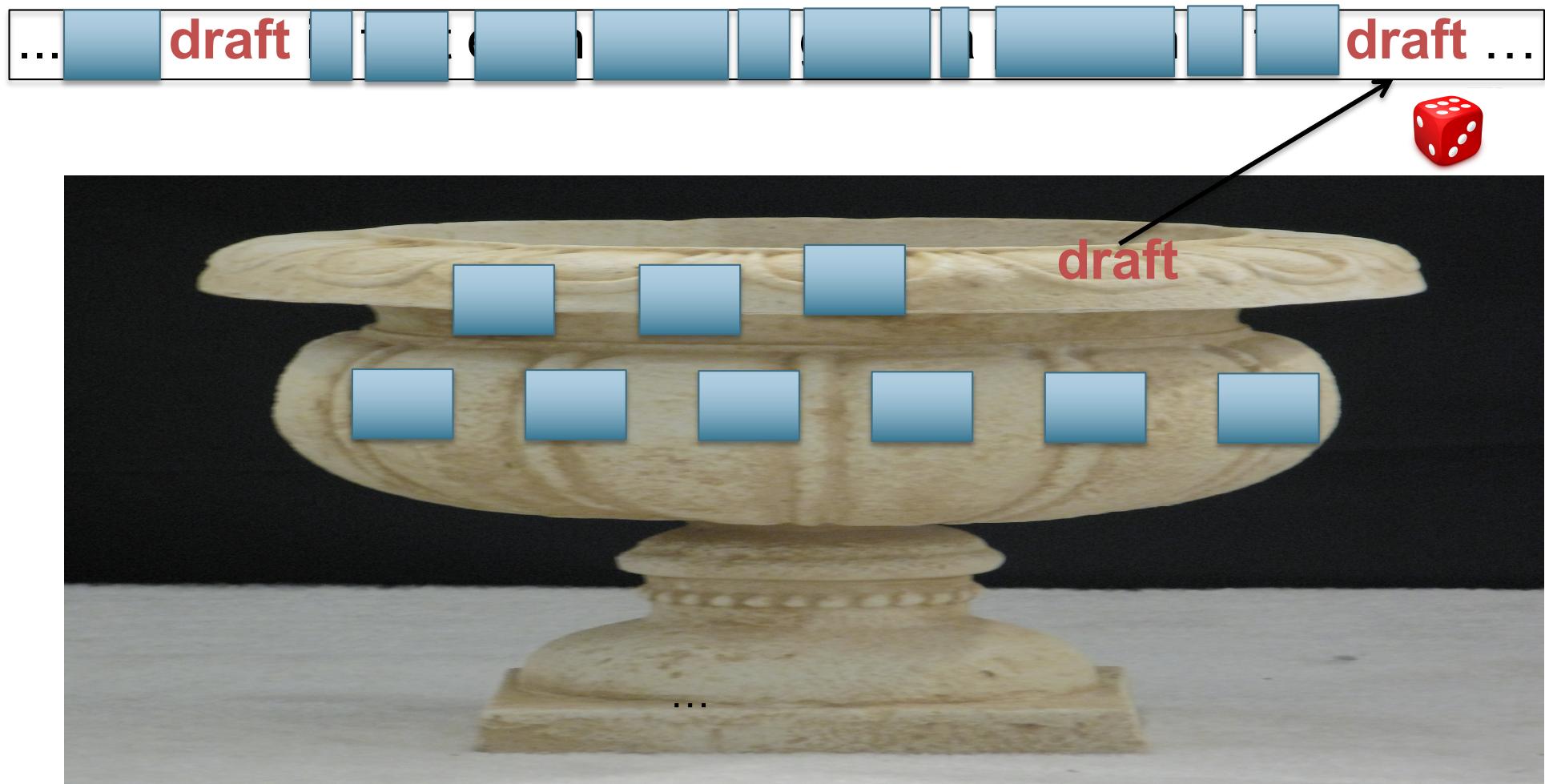
- Words are drawn independently from the vocabulary using a multinomial distribution

... the **draft** is that each **team** is given a position in the **draft** ...



Generative model for documents

- Distribution of term frequencies (tf) follows a binomial distribution – approximated by a Poisson



Poisson distribution

- The Poisson distribution models the probability of k , the number of events occurring in a fixed interval of time/space, with known average rate λ ($= \text{cf}/T$), independent of the last event

$$p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

- Examples
 - Number of cars arriving at a toll booth per minute
 - Number of typos on a page

Poisson distribution

- If T is large and p is small, we can approximate a binomial distribution with a Poisson where $\lambda = Tp$

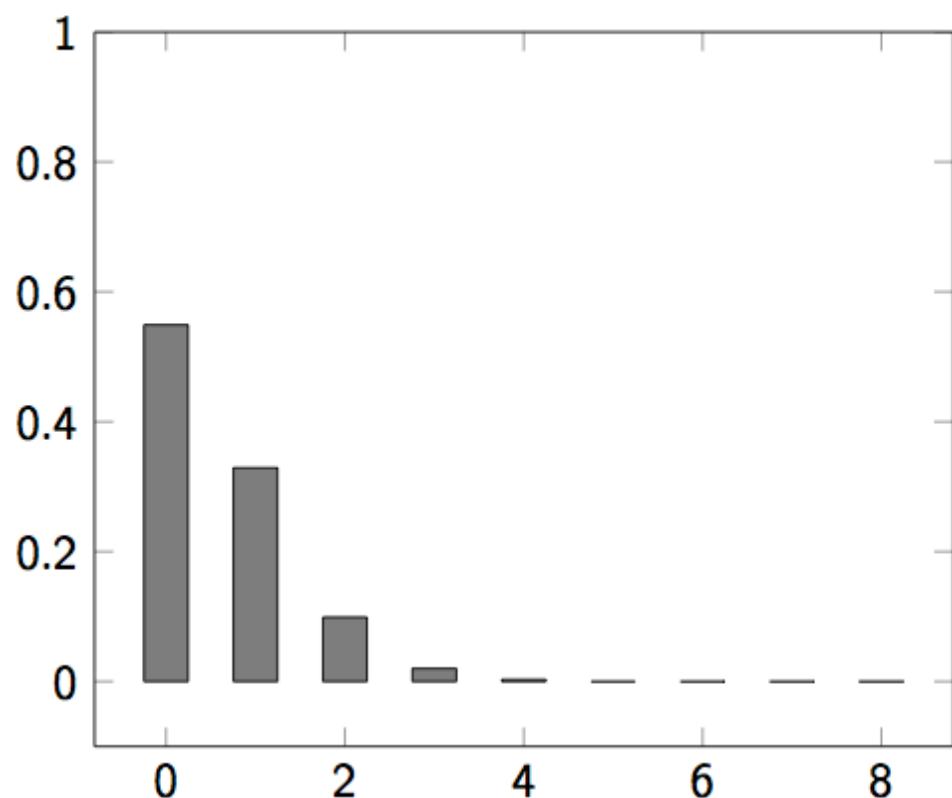
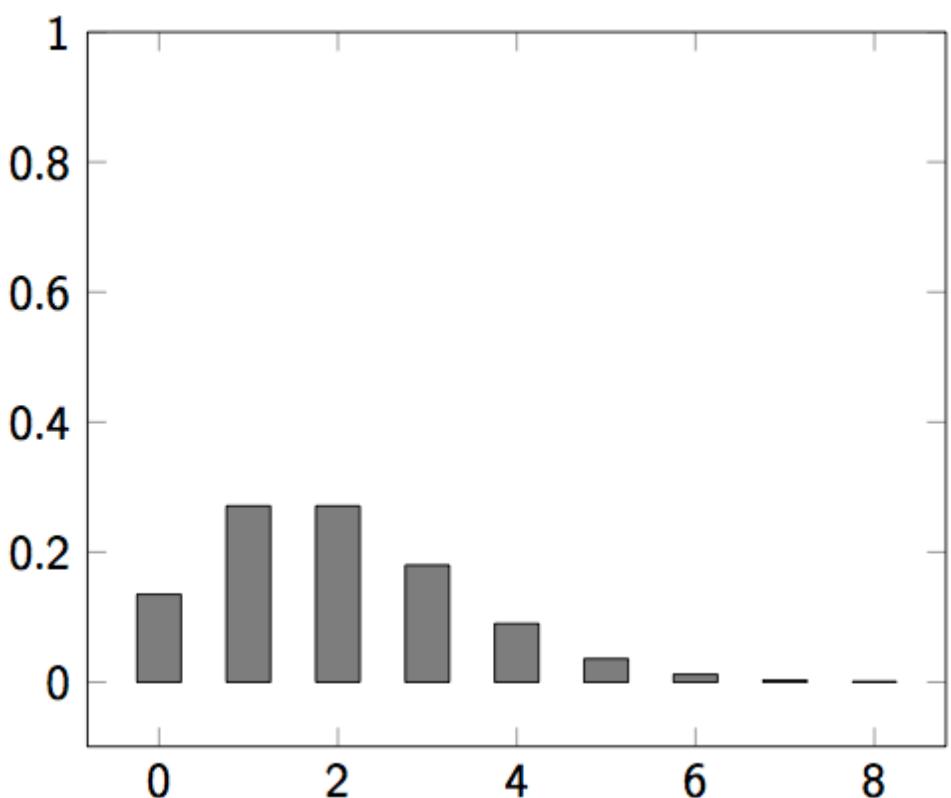
$$p(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

- Mean = Variance = $\lambda = Tp$.
- Example $p = 0.08$, $T = 20$. Chance of 1 occurrence is:
 - Binomial $P(1) = \binom{20}{1} (.08)^1 (.92)^{19} = .3282$
 - Poisson $P(1) = \frac{[(20)(.08)]^1}{1!} e^{-(20)(.08)} = \frac{1.6}{1} e^{-1.6} = 0.3230$... already close

Poisson model

- Assume that term frequencies in a document (tf_i) follow a Poisson distribution
 - “Fixed interval” implies fixed document length ... think roughly constant-sized document abstracts
 - ... will fix later

Poisson distributions

 $\lambda = 0.6$  $\lambda = 2$ 

(One) Poisson Model flaw

- Is a reasonable fit for “general” words
- Is a poor fit for topic-specific words
 - get higher $p(k)$ than predicted too often

		Documents containing k occurrences of word ($\lambda = 53/650$)												
Freq	Word	0	1	2	3	4	5	6	7	8	9	10	11	12
53	expected	599	49	2										
52	based	600	48	2										
53	conditions	604	39	7										
55	cathexis	619	22	3	2	1	2	0	1					
51	comic	642	3	0	1	0	0	0	0	0	0	1	1	2

Harter, “A Probabilistic Approach to Automatic Keyword Indexing”, JASIST, 1975

Eliteness (“aboutness”)

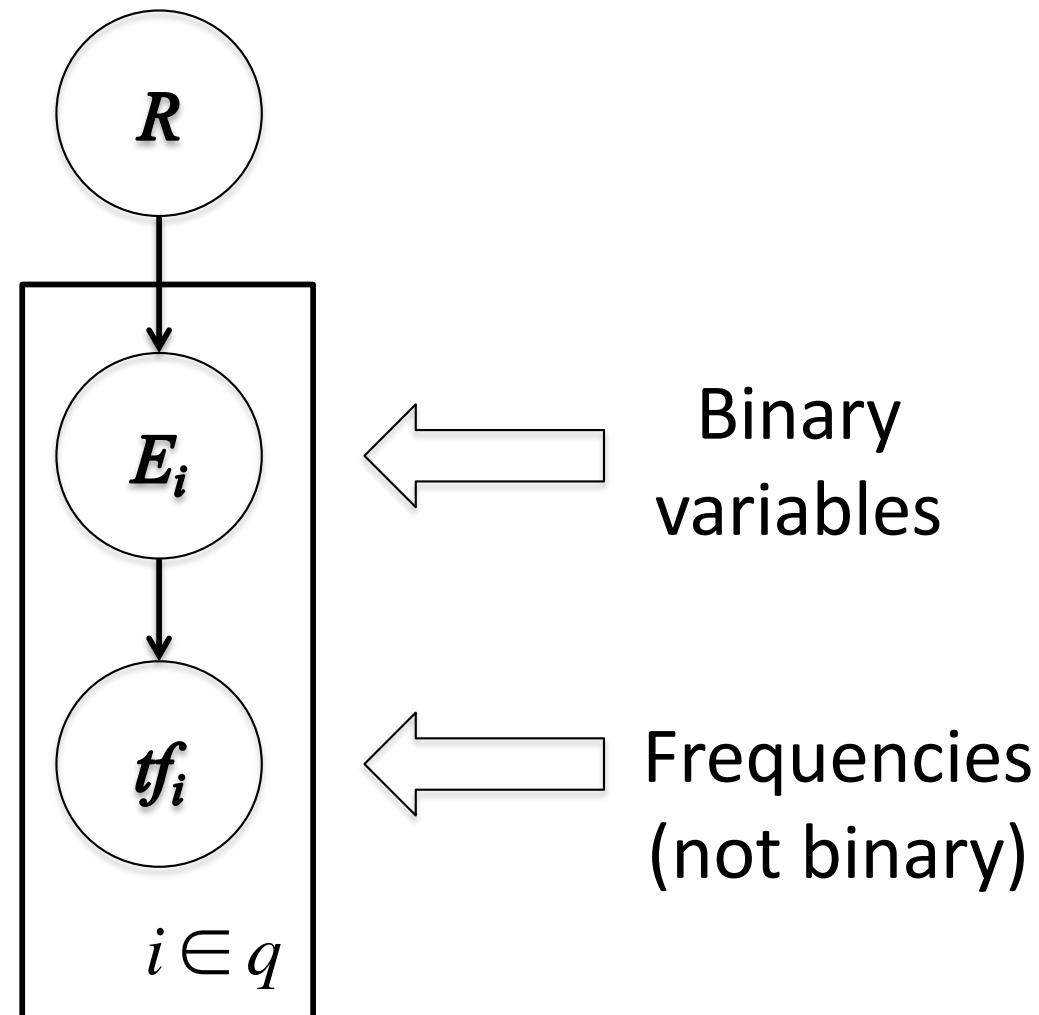
- Model term frequencies using *eliteness*
- What is eliteness?
 - Hidden variable for each document-term pair, denoted as E_i for term i
 - Represents *aboutness*: a term is elite in a document if, in some sense, the document is about the concept denoted by the term
 - Eliteness is binary
 - Term occurrences depend only on eliteness...
 - ... but eliteness depends on relevance

Elite terms

Text from the Wikipedia page on the NFL draft showing
elite terms

The **National Football League Draft** is an annual event in which the **National Football League (NFL)** teams select eligible college football players. It serves as the league's most common source of player recruitment. The basic design of the **draft** is that each **team** is given a **position** in the **draft order** in **reverse order** relative to its **record** ...

Graphical model with eliteness



Retrieval Status Value

- Similar to the BIM derivation, we have

$$RSV^{elite} = \sum_{i \in q, tf_i > 0} c_i^{elite}(tf_i);$$

where

$$c_i^{elite}(tf_i) = \log \frac{p(TF_i = tf_i | R = 1)p(TF_i = 0 | R = 0)}{p(TF_i = 0 | R = 1)p(TF_i = tf_i | R = 0)}$$

and using eliteness, we have:

$$\begin{aligned} p(TF_i = tf_i | R) &= p(TF_i = tf_i | E_i = elite)p(E_i = elite | R) \\ &\quad + p(TF_i = tf_i | E_i = \overline{elite})(1 - p(E_i = elite | R)) \end{aligned}$$

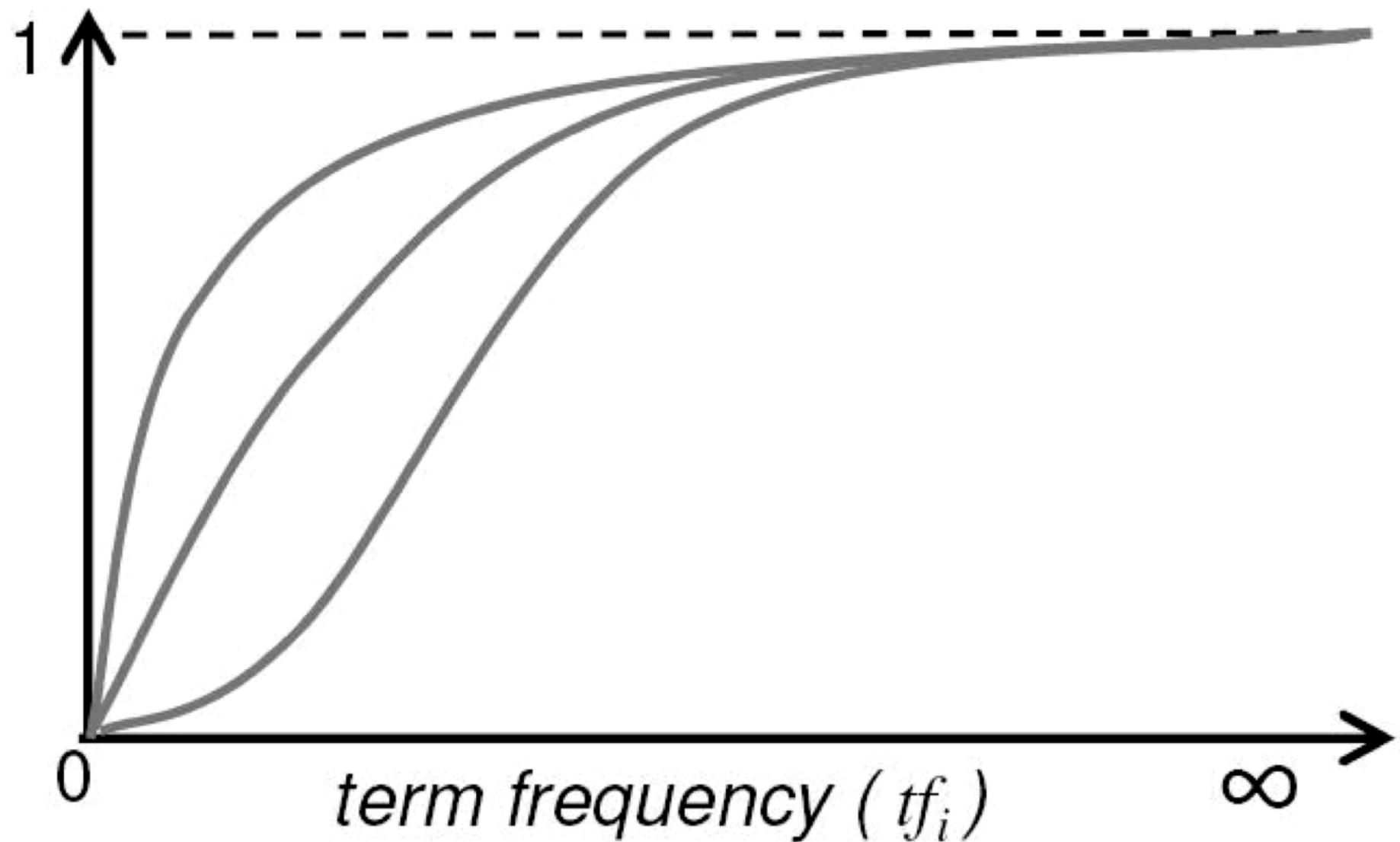
2-Poisson model

- The problems with the 1-Poisson model suggests fitting two Poisson distributions
- In the “2-Poisson model”, the distribution is different depending on whether the term is elite or not

$$p(TF_i = k_i | R) = \pi \frac{\lambda^k}{k!} e^{-\lambda} + (1 - \pi) \frac{\mu^k}{k!} e^{-\mu}$$

- where π is probability that document is elite for term
- but, unfortunately, we don't know π, λ, μ

Let's get an idea: Graphing $c_i^{elite}(tf_i)$ for different parameter values of the 2-Poisson



Qualitative properties

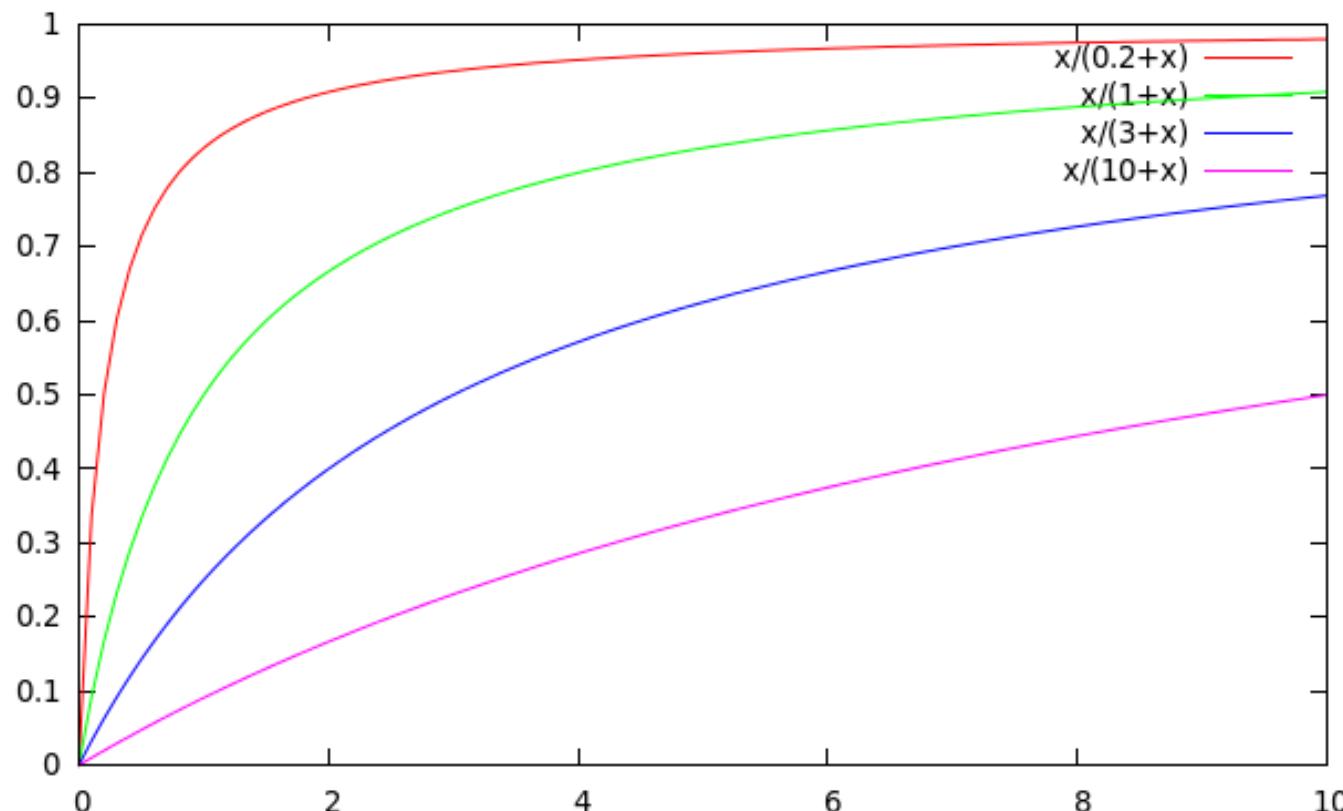
- $c_i^{elite}(0) = 0$
 - $c_i^{elite}(tf_i)$ increases monotonically with tf_i
 - ... but asymptotically approaches a maximum value as $tf_i \rightarrow \infty$ [not true for simple scaling of tf]
 - ... with the asymptotic limit being c_i^{BIM}
- Weight of
eliteness
feature

Approximating the saturation function

- Estimating parameters for the 2-Poisson model is not easy
- ... So approximate it with a simple parametric curve that has the same qualitative properties

$$\frac{tf}{k_1 + tf}$$

Saturation function



- For high values of k_1 , increments in tf_i continue to contribute significantly to the score
- Contributions tail off quickly for low values of k_1

“Early” versions of BM25

- Version 1: using the saturation function

$$c_i^{BM25v1}(tf_i) = c_i^{BIM} \frac{tf_i}{k_1 + tf_i}$$

- Version 2: BIM simplification to IDF

$$c_i^{BM25v2}(tf_i) = \log \frac{N}{df_i} \times \frac{(k_1 + 1)tf_i}{k_1 + tf_i}$$

- $(k_1 + 1)$ factor doesn't change ranking, but makes term score 1 when $tf_i = 1$
- Similar to $tf-idf$, but term scores are bounded

Document length normalization

- Longer documents are likely to have larger tf_i values
- Why might documents be longer?
 - Verbosity: suggests observed tf_i too high
 - Larger scope: suggests observed tf_i may be right
- A real document collection probably has both effects
- ... so should apply some kind of partial normalization

Document length normalization

- Document length:

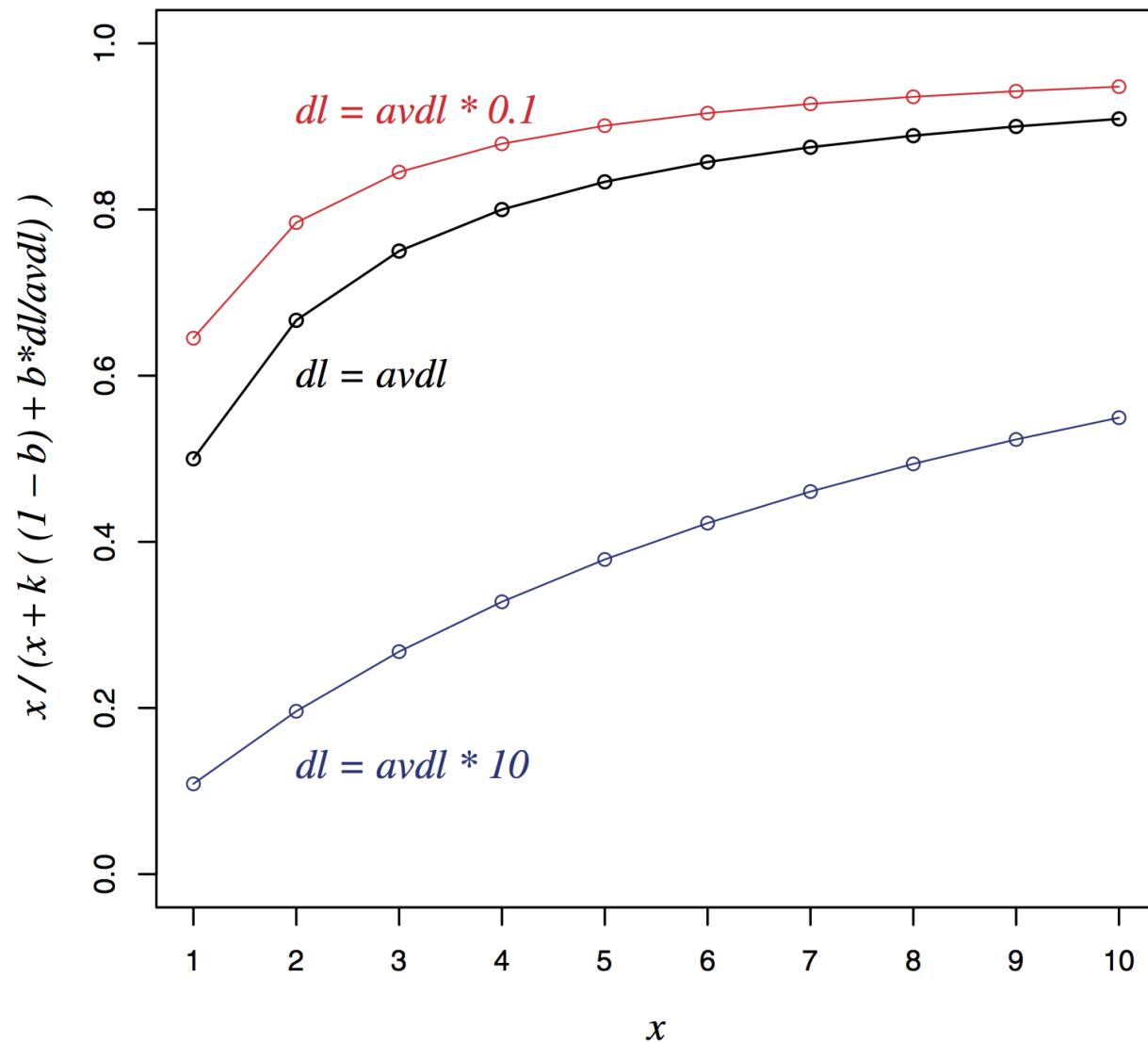
$$dl = \sum_{i \in V} tf_i$$

- $avdl$: Average document length over collection
- Length normalization component

$$B = \left((1 - b) + b \frac{dl}{avdl} \right), \quad 0 \leq b \leq 1$$

- $b = 1$ full document length normalization
- $b = 0$ no document length normalization

Document length normalization



Okapi BM25

- Normalize tf using document length

$$tf'_i = \frac{tf_i}{B}$$

$$\begin{aligned} c_i^{BM25}(tf_i) &= \log \frac{N}{df_i} \times \frac{(k_1 + 1)tf'_i}{k_1 + tf'_i} \\ &= \log \frac{N}{df_i} \times \frac{(k_1 + 1)tf_i}{k_1((1 - b) + b \frac{dl}{avdl}) + tf_i} \end{aligned}$$

- BM25 ranking function

$$RSV^{BM25} = \sum_{i \in q} c_i^{BM25}(tf_i);$$

Okapi BM25

$$RSV^{BM25} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1 + 1)tf_i}{k_1((1 - b) + b \frac{dl}{avdl}) + tf_i}$$

- k_1 controls term frequency scaling
 - $k_1 = 0$ is binary model; k_1 large is raw term frequency
- b controls document length normalization
 - $b = 0$ is no length normalization; $b = 1$ is relative frequency (fully scale by document length)
- Typically, k_1 is set around 1.2–2 and b around 0.75
- IIR sec. 11.4.3 discusses incorporating query term weighting and (pseudo) relevance feedback

Why is BM25 better than VSM tf-idf?

- Suppose your query is [machine learning]
- Suppose you have 2 documents with term counts:
 - doc1: learning 1024; machine 1
 - doc2: learning 16; machine 8
- tf-idf: $\log_2 \text{tf} * \log_2 (\text{N}/\text{df})$
 - doc1: $11 * 7 + 1 * 10 = 87$
 - doc2: $5 * 7 + 4 * 10 = 75$
- BM25: $k_1 = 2$
 - doc1: $7 * 3 + 10 * 1 = 31$
 - doc2: $7 * 2.67 + 10 * 2.4 = 42.7$

7. Ranking with features

- Textual features
 - Zones: Title, author, abstract, body, anchors, ...
 - Proximity
 - ...
- Non-textual features
 - File type
 - File age
 - Page rank
 - ...

Ranking with zones

- Straightforward idea:
 - Apply your favorite ranking function (BM25) to each zone separately
 - Combine zone scores using a weighted linear combination
- But that seems to imply that the eliteness properties of different zones are different and independent of each other
 - ...which seems unreasonable

Ranking with zones

- Alternate idea
 - Assume eliteness is a term/document property shared across zones
 - ... but the relationship between eliteness and term frequencies are zone-dependent
 - e.g., denser use of elite topic words in title
- Consequence
 - First combine evidence across zones for each term
 - Then combine evidence across terms

BM25F with zones

- Calculate a weighted variant of total term frequency
- ... and a weighted variant of document length

$$\tilde{tf}_i = \sum_{z=1}^Z v_z tf_{zi} \quad \tilde{dl} = \sum_{z=1}^Z v_z len_z \quad avd\tilde{l} = \text{Average } \tilde{dl} \text{ across all documents}$$

where

v_z is zone weight

tf_{zi} is term frequency in zone z

len_z is length of zone z

Z is the number of zones

Simple BM25F with zones

$$RSV^{SimpleBM25F} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1 + 1)\tilde{tf}_i}{k_1((1 - b) + b \frac{\tilde{dl}}{avd\tilde{l}}) + \tilde{tf}_i}$$

- Simple interpretation: zone z is “replicated” v_z times
- But we may want zone-specific parameters (k_1 , b , IDF)

BM25F

- Empirically, zone-specific length normalization (i.e., zone-specific b) has been found to be useful

$$\tilde{tf}_i = \sum_{z=1}^Z v_z \frac{tf_{zi}}{B_z}$$

$$B_z = \left((1 - b_z) + b_z \frac{len_z}{avlen_z} \right), \quad 0 \leq b_z \leq 1$$

$$RSV^{BM25F} = \sum_{i \in q} \log \frac{N}{df_i} \cdot \frac{(k_1 + 1)\tilde{tf}_i}{k_1 + \tilde{tf}_i}$$

See Robertson and Zaragoza (2009: 364)

Ranking with non-textual features

- Assumptions
 - Usual independence assumption
 - Independent of each other and of the textual features
 - Allows us to factor out $\frac{p(F_j = f_j | R = 1)}{p(F_j = f_j | R = 0)}$ in BIM-style derivation
 - Relevance information is ***query independent***
 - Usually true for features like page rank, age, type, ...
 - Allows us to keep all non-textual features in the BIM-style derivation where we drop non-query terms

Ranking with non-textual features

$$RSV = \sum_{i \in q} c_i(tf_i) + \sum_{j=1}^F \lambda_j V_j(f_j)$$

where

$$V_j(f_j) = \log \frac{p(F_j = f_j | R = 1)}{p(F_j = f_j | R = 0)}$$

and λ_j is an artificially added free parameter to account for rescalings in the approximations

- Care must be taken in selecting V_j depending on F_j . E.g.

$$\log(\lambda'_j + f_j) \quad \frac{f_j}{\lambda'_j + f_j} \quad \frac{1}{\lambda'_j + \exp(-f_j \lambda''_j)}$$

- Explains why $RSV^{BM25} + \log(\text{pagerank})$ works well

Resources

- S. E. Robertson and K. Spärck Jones. 1976. Relevance Weighting of Search Terms. *Journal of the American Society for Information Sciences* 27(3): 129–146.
- C. J. van Rijsbergen. 1979. *Information Retrieval*. 2nd ed. London: Butterworths, chapter 6. <http://www.dcs.gla.ac.uk/Keith/Preface.html>
- K. Spärck Jones, S. Walker, and S. E. Robertson. 2000. A probabilistic model of information retrieval: Development and comparative experiments. Part 1. *Information Processing and Management* 779–808.
- S. E. Robertson and H. Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Foundations and Trends in Information Retrieval* 3(4): 333-389.

Introduction to
Information Retrieval

CS276
Information Retrieval and Web Search

Chris Manning and Pandu Nayak

Systems issues

Background

- Score computation is a large (10s of %) fraction of the CPU work on a query
 - Generally, we have a tight budget on latency (say, 250ms)
 - CPU provisioning doesn't permit exhaustively scoring every document on every query
- Today we'll look at ways of cutting CPU usage for scoring, without compromising the quality of results (much)
- Basic idea: avoid scoring docs that won't make it into the top K

Safe vs non-safe ranking

- The terminology “safe ranking” is used for methods that guarantee that the K docs returned are the K absolute highest scoring documents
- Is it ok to be non-safe?

Ranking function is only a proxy

- User has a task and a query formulation
- Ranking function matches docs to query
- Thus the ranking function is anyway a proxy for user happiness
- If we get a list of K docs “close” to the top K by the ranking function measure, should be ok

Recap: 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, measured by cosine similarity

Efficient cosine ranking

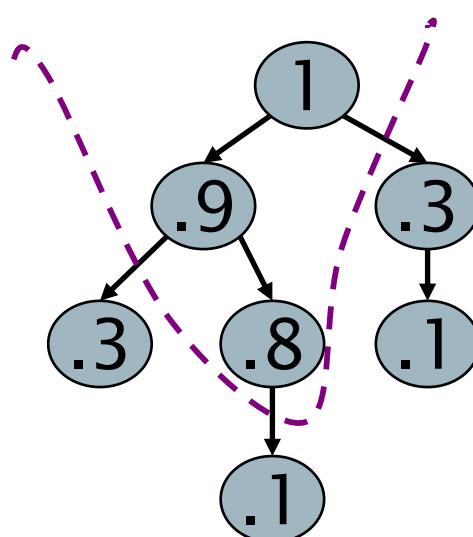
- Find the K docs in the collection “nearest” to the query $\Rightarrow K$ largest query-doc cosines.
- Efficient ranking:
 - Computing a single cosine efficiently.
 - Choosing the K largest cosine values efficiently.
 - Can we do this without computing all N cosines?

Computing the K largest cosines: selection vs. sorting

- Typically we want to retrieve the top K docs (in the cosine ranking for the query)
 - not to totally order all docs in the collection
- Can we pick off docs with K highest cosines?
- Let J = number of docs with nonzero cosines
 - We seek the K best of these J

Use heap for selecting top K

- Binary tree in which each node's value > the values of children
- Takes $2J$ operations to construct, then each of K “winners” read off in $2\log J$ steps.
- For $J=1M$, $K=100$, this is about 10% of the cost of sorting.



Bottlenecks

- Primary computational bottleneck in scoring: cosine computation
- **Can we avoid all this computation?**
- Yes, but may sometimes get it wrong
 - a doc *not* in the top K may creep into the list of K output docs
 - As noted earlier, this may not be a bad thing

SPEEDING COSINE COMPUTATION BY PRUNING

Generic approach

- Find a set A of *contenders*, with $K < |A| \ll N$
 - A does not necessarily contain the top K , but has many docs from among the top K
 - Return the top K docs in A
- Think of A as pruning non-contenders
- The same approach is also used for other (non-cosine) scoring functions
- Will look at several schemes following this approach

Index elimination

- Basic cosine computation algorithm only considers docs containing at least one query term
- Take this further:
 - Only consider high-idf query terms
 - Only consider docs containing many query terms

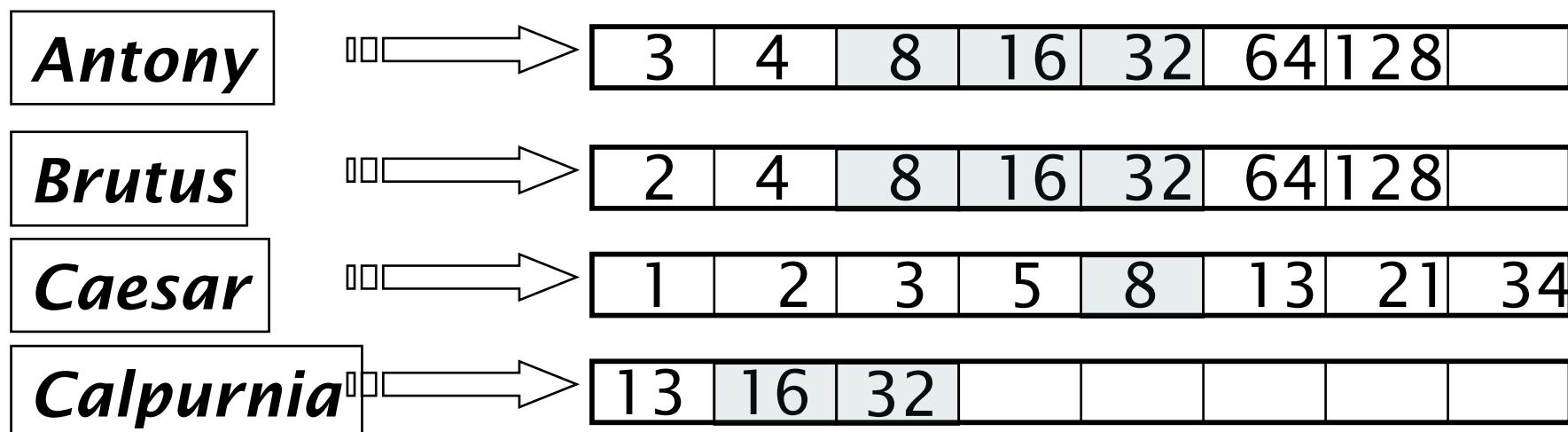
High-idf query terms only

- For a query such as *catcher in the rye*
- Only accumulate scores from *catcher* and *rye*
- Intuition: ***in*** and ***the*** contribute little to the scores and so don't alter rank-ordering much
- Benefit:
 - Postings of low-idf terms have many docs → these (many) docs get eliminated from set A of contenders

Docs containing many query terms

- Any doc with at least one query term is a candidate for the top K output list
- For multi-term queries, only compute scores for docs containing several of the query terms
 - Say, at least 3 out of 4
 - Imposes a “soft conjunction” on queries seen on web search engines (early Google)
- Easy to implement in postings traversal

3 of 4 query terms



Scores only computed for docs 8, 16 and 32.

Champion lists

- Precompute for each dictionary term t , the r docs of highest weight in t 's postings
 - Call this the champion list for t
 - (aka fancy list or top docs for t)
- Note that r has to be chosen at index build time
 - Thus, it's possible that $r < K$
- At query time, only compute scores for docs in the champion list of some query term
 - Pick the K top-scoring docs from amongst these

Exercises

- How can Champion Lists be implemented in an inverted index?

QUERY-INDEPENDENT DOCUMENT SCORES

Static quality scores

- We want top-ranking documents to be both *relevant* and *authoritative*
- *Relevance* is being modeled by cosine scores
- *Authority* is typically a query-independent property of a document
- Examples of authority signals
 - Wikipedia among websites
 - Articles in certain newspapers
 - A paper with many citations
 - Many bitlys, likes, or bookmarks
 - Pagerank

Quantitative

Modeling authority

- Assign to each document a *query-independent quality score* in $[0,1]$ to each document d
 - Denote this by $g(d)$
- Thus, a quantity like the number of citations is scaled into $[0,1]$
 - Exercise: suggest a formula for this.

Net score

- Consider a simple total score combining cosine relevance and authority
- $\text{net-score}(q, d) = g(d) + \cosine(q, d)$
 - Can use some other linear combination
 - Indeed, any function of the two “signals” of user happiness
- Now we seek the top K docs by net score

Top K by net score – fast methods

- First idea: Order all postings by $g(d)$
- Key: this is a common ordering for all postings
- Thus, can concurrently traverse query terms' postings for
 - Postings intersection
 - Cosine score computation
- Exercise: write pseudocode for cosine score computation if postings are ordered by $g(d)$

Why order postings by $g(d)$?

- Under $g(d)$ -ordering, top-scoring docs likely to appear early in postings traversal
- In time-bound applications (say, we have to return whatever search results we can in 50 ms), this allows us to stop postings traversal early
 - Short of computing scores for all docs in postings

Champion lists in $g(d)$ -ordering

- Can combine champion lists with $g(d)$ -ordering
- Maintain for each term a champion list of the r docs with highest $g(d) + \text{tf-idf}_{td}$
- Seek top- K results from only the docs in these champion lists

CLUSTER PRUNING

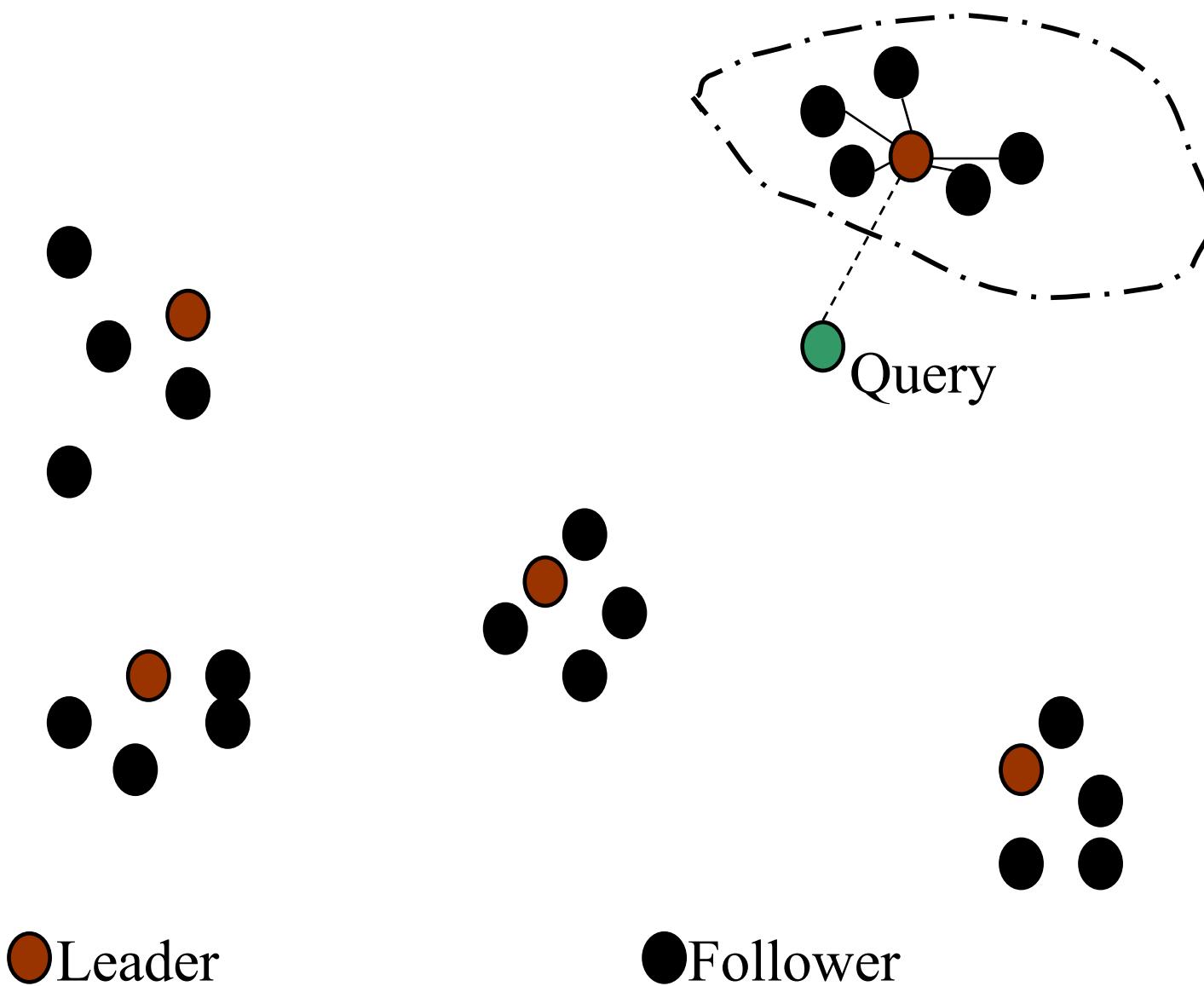
Cluster pruning: preprocessing

- Pick \sqrt{N} docs at random: call these *leaders*
- For every other doc, pre-compute nearest leader
 - Docs attached to a leader: its *followers*;
 - Likely: each leader has $\sim \sqrt{N}$ followers.

Cluster pruning: query processing

- Process a query as follows:
 - Given query Q , find its nearest *leader* L .
 - Seek K nearest docs from among L 's followers.

Visualization



Why use random sampling

- Fast
- Leaders reflect data distribution

General variants

- Have each follower attached to $b1=3$ (say) nearest leaders.
- From query, find $b2=4$ (say) nearest leaders and their followers.
- Can recurse on leader/follower construction.

TIERED INDEXES

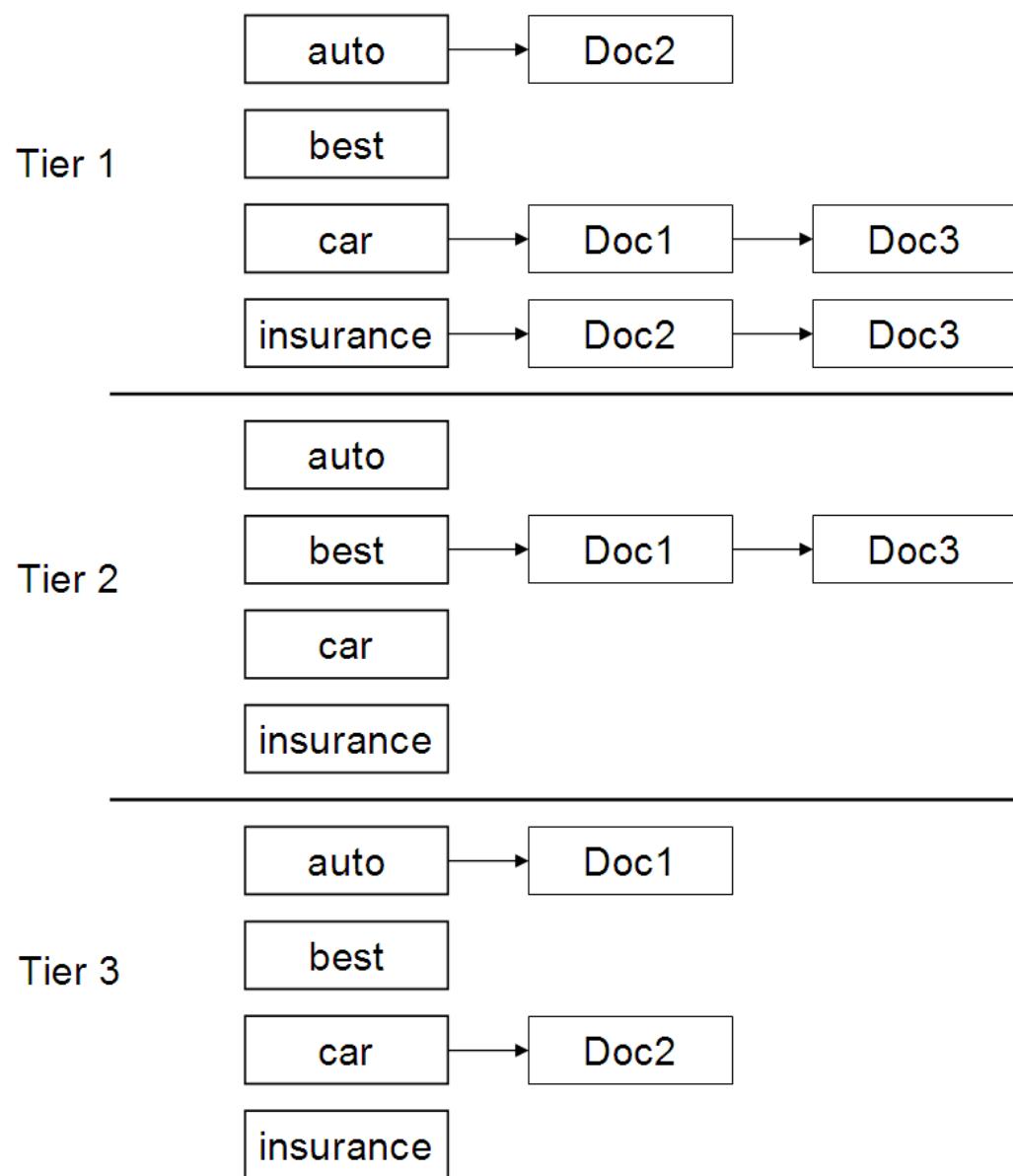
High and low lists

- For each term, we maintain two postings lists called *high* and *low*
 - Think of *high* as the champion list
- When traversing postings on a query, only traverse *high* lists first
 - If we get more than K docs, select the top K and stop
 - Else proceed to get docs from the *low* lists
- Can be used even for simple cosine scores, without global quality $g(d)$
- A means for segmenting index into two tiers

Tiered indexes

- Break postings up into a hierarchy of lists
 - Most important
 - ...
 - Least important
- Can be done by $g(d)$ or another measure
- Inverted index thus broken up into tiers of decreasing importance
- At query time use top tier unless it fails to yield K docs
 - If so drop to lower tiers

Example tiered index



Impact-ordered postings

- We only want to compute scores for docs for which $wf_{t,d}$ is high enough
- We sort each postings list by $wf_{t,d}$
- Now: not all postings in a common order!
- How do we compute scores in order to pick off top K ?
 - Two ideas follow

1. Early termination

- When traversing t 's postings, stop early after either
 - a fixed number of r docs
 - $wf_{t,d}$ drops below some threshold
- Take the union of the resulting sets of docs
 - One from the postings of each query term
- Compute only the scores for docs in this union

2. idf-ordered terms

- When considering the postings of query terms
- Look at them in order of decreasing idf
 - High idf terms likely to contribute most to score
- As we update score contribution from each query term
 - Stop if doc scores relatively unchanged
- Can apply to cosine or some other net scores

SAFE RANKING

Safe vs non-safe ranking

- The terminology “safe ranking” is used for methods that guarantee that the K docs returned are the K absolute highest scoring documents
 - (Not necessarily just under cosine similarity)

Safe ranking

- When we output the top K docs, we have a proof that these are indeed the top K
- Does this imply we always have to compute all N cosines?
 - We'll look at pruning methods
 - So we only fully score some J documents

WAND scoring

- An instance of DAAT scoring
- Basic idea reminiscent of branch and bound
 - We maintain a running *threshold* score – e.g., the K^{th} highest score computed so far
 - We prune away all docs whose cosine scores are guaranteed to be below the threshold
 - We compute exact cosine scores for only the un-pruned docs

Broder et al. Efficient Query Evaluation using a Two-Level Retrieval Process.

Index structure for WAND

- Postings ordered by docID
- Assume a special iterator on the postings of the form “go to the first docID greater than or equal to X ”
- Typical state: we have a “finger” at some docID in the postings of each query term
 - Each finger moves only to the right, to larger docIDs
- Invariant – all docIDs lower than any finger have already been *processed*, meaning
 - These docIDs are either pruned away or
 - Their cosine scores have been computed

Upper bounds

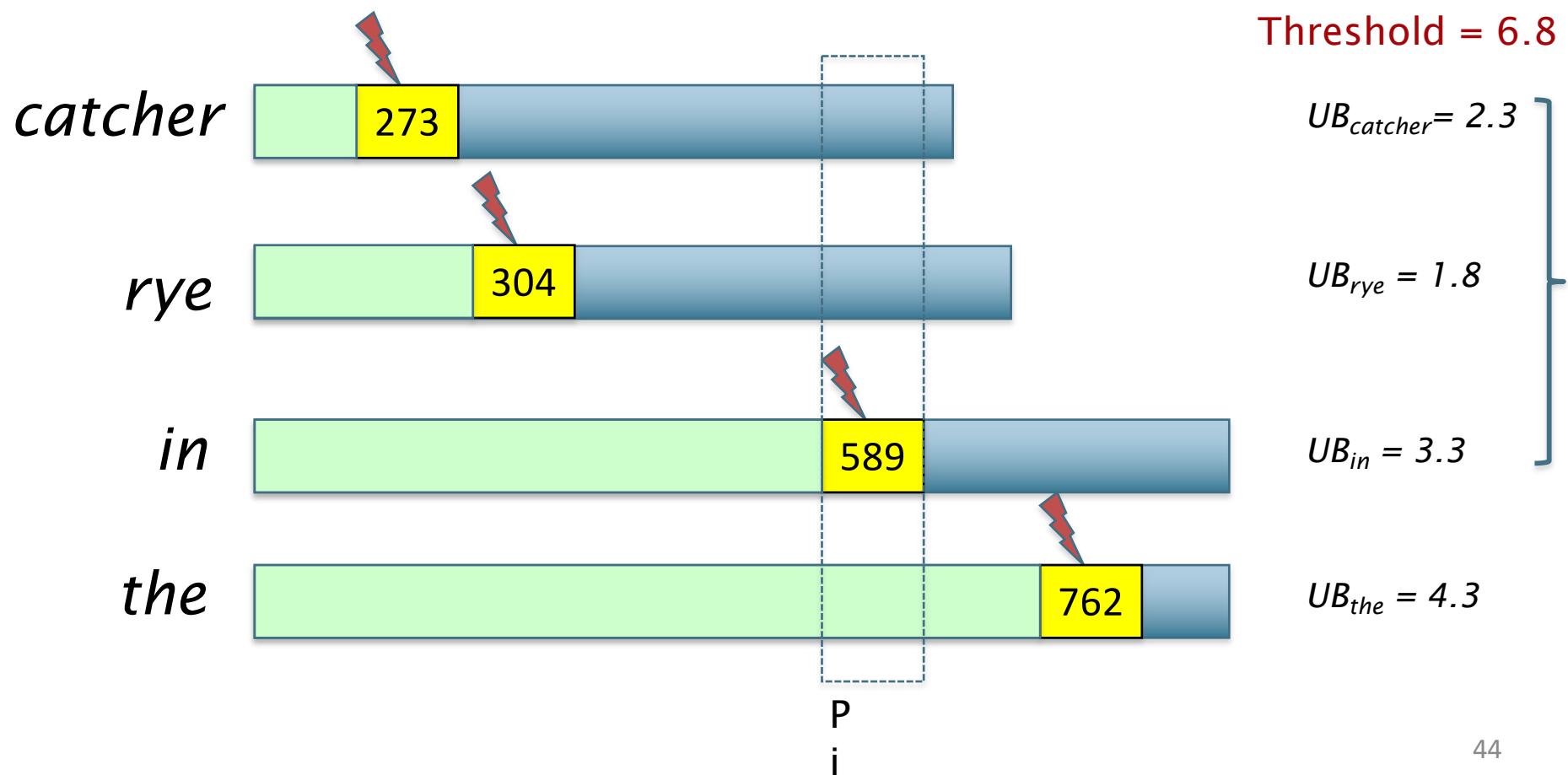
- At all times for each query term t , we maintain an *upper bound* UB_t on the score contribution of any doc to the right of the finger
 - Max (over docs remaining in t 's postings) of $w_t(\text{doc})$



As finger moves right, UB drops

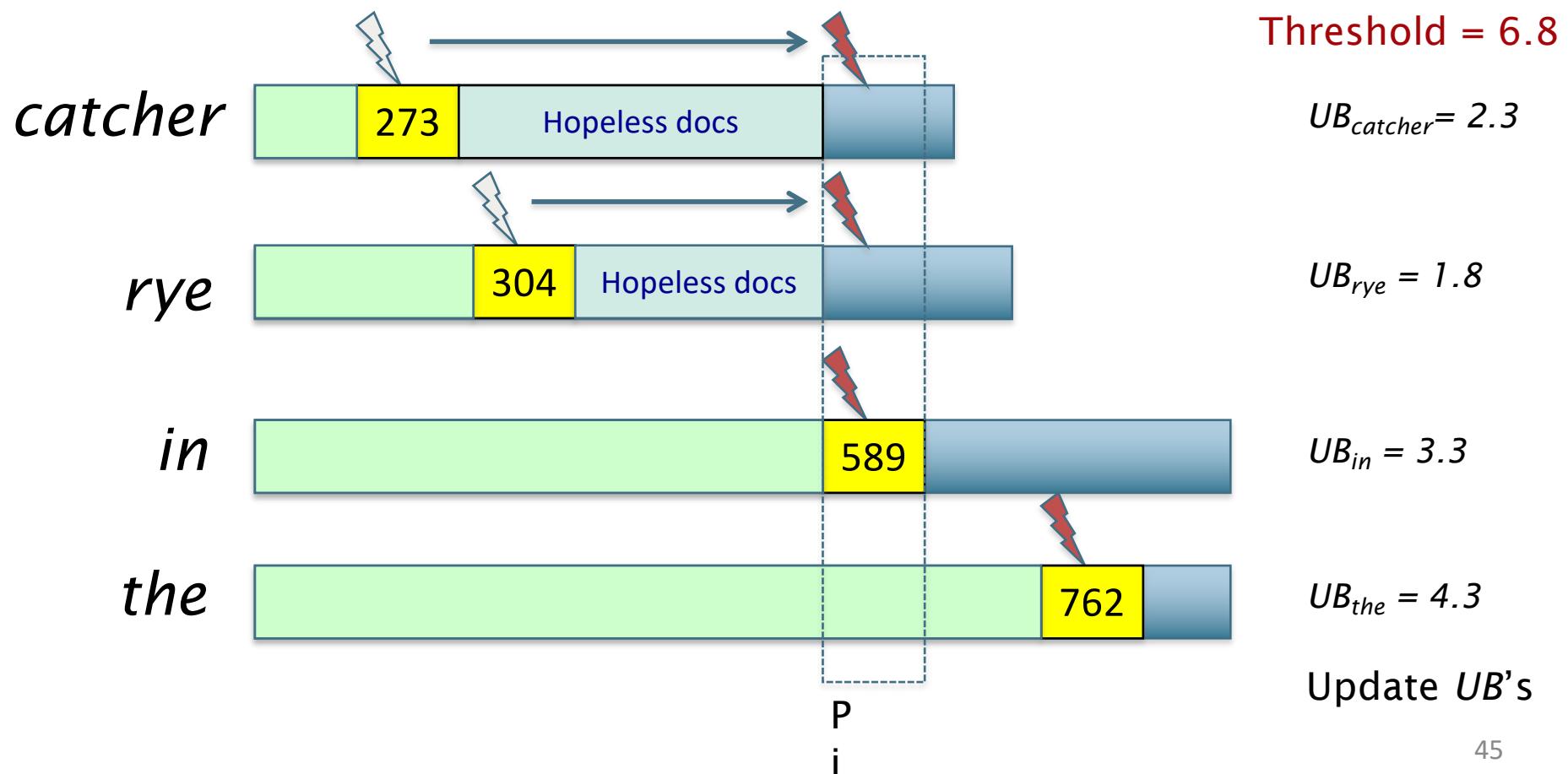
Pivoting

- Query: *catcher in the rye*
- Let's say the current finger positions are as below



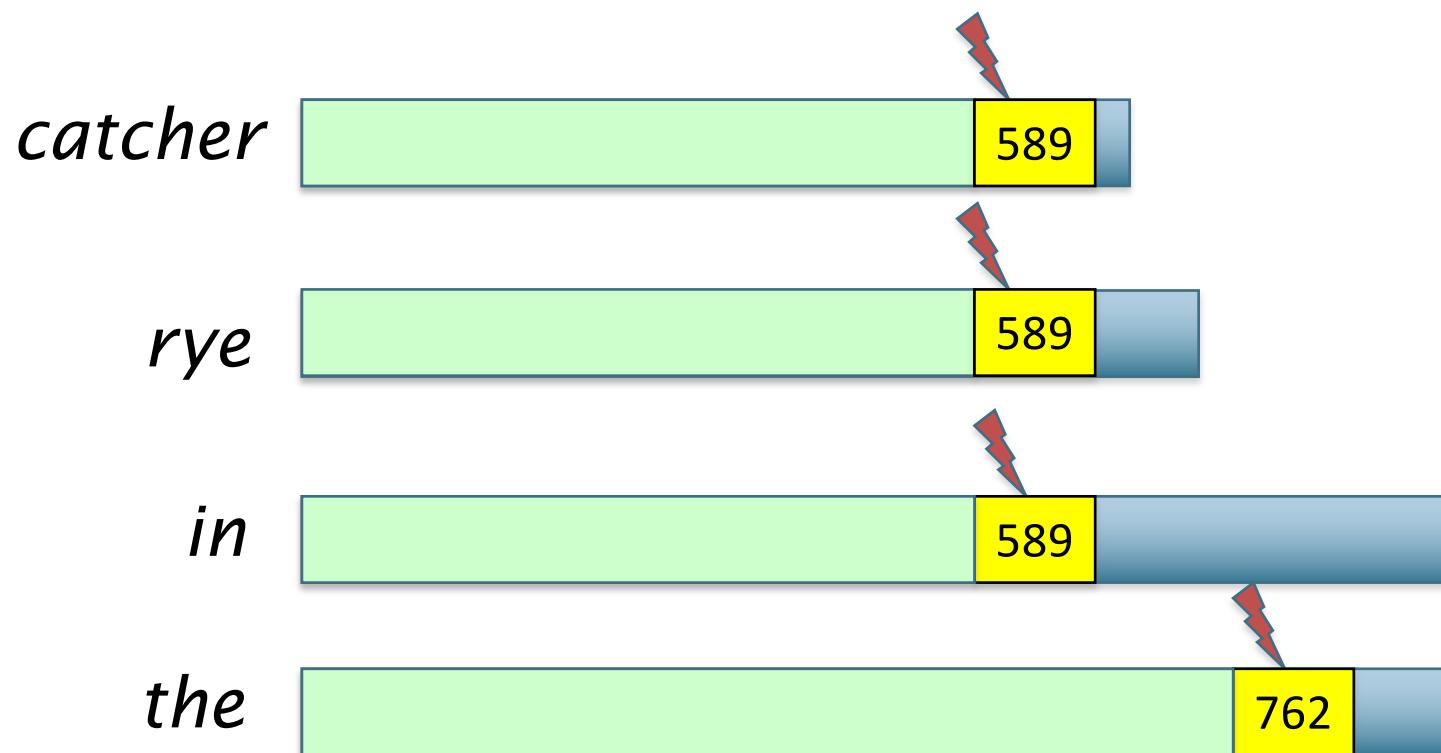
Prune docs that have no hope

- Terms sorted in order of finger positions
- Move fingers to 589 or right



Compute 589's score if need be

- If 589 is present in enough postings, compute its full cosine score – else some fingers to right of 589
- Pivot again ...



WAND summary

- In tests, WAND leads to a 90+% reduction in score computation
 - Better gains on longer queries
- Nothing we did was specific to cosine ranking
 - We need scoring to be *additive* by term
- WAND and variants give us safe ranking
 - Possible to devise “careless” variants that are a bit faster but not safe (see summary in Ding+Suel 2011)
 - Ideas combine some of the non-safe scoring we considered

FINISHING TOUCHES FOR A COMPLETE SCORING SYSTEM

Query term proximity

- Free text queries: just a set of terms typed into the query box – common on the web
- Users prefer docs in which query terms occur within close proximity of each other
- Let w be the smallest window in a doc containing all query terms, e.g.,
- For the query *strained mercy* the smallest window in the doc *The quality of mercy is not strained* is 4 (words)
- Would like scoring function to take this into account – how?

Query parsers

- Free text query from user may in fact spawn one or more queries to the indexes, e.g., query *rising interest rates*
 - Run the query as a phrase query
 - If $<K$ docs contain the phrase *rising interest rates*, run the two phrase queries *rising interest* and *interest rates*
 - If we still have $<K$ docs, run the vector space query *rising interest rates*
 - Rank matching docs by vector space scoring
- This sequence is issued by a query parser

Aggregate scores

- We've seen that score functions can combine cosine, static quality, proximity, etc.
- **How do we know the best combination?**
- Some applications – expert-tuned
- Increasingly common: machine-learned

Putting it all together

