CS4442/9542b Artificial Intelligence II prof. Olga Veksler

Lecture 17

Natural Language Processing

Information Retrieval

Many slides from: L. Kosseim (Concordia), Jamie Callan (CMU), C. Manning (Stanford), L. Venkata Subramaniam, Phillip Resnik

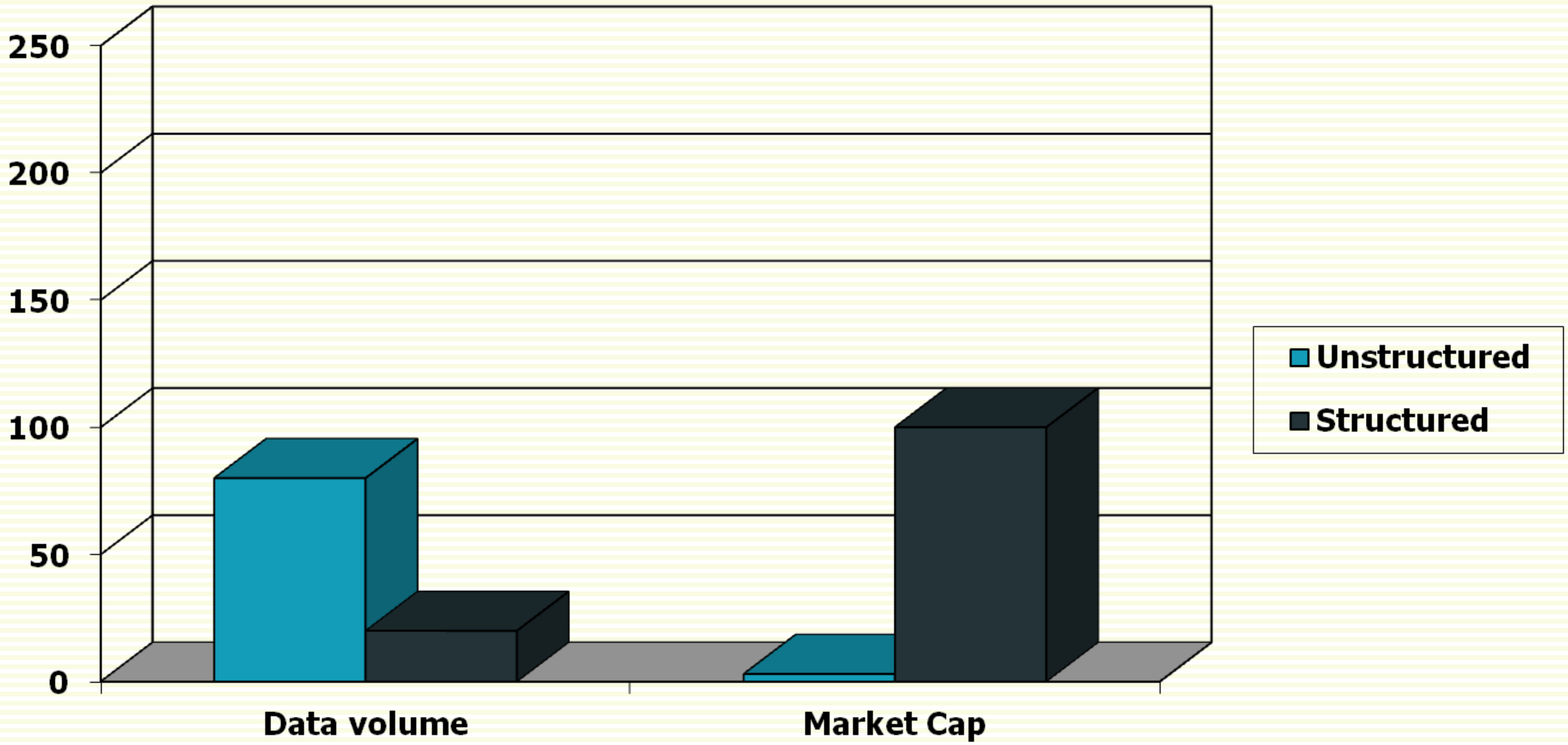
Outline

- Introduction to Information Retrieval (IR)
- Ad hoc information retrieval
    - Boolean Model
    - Vector Space Model
        - Cosine similarity measure
        - Choosing term weights
    - Performance evaluation methods
    - Improving IR system
        - Query expansion
        - Relevance feedback

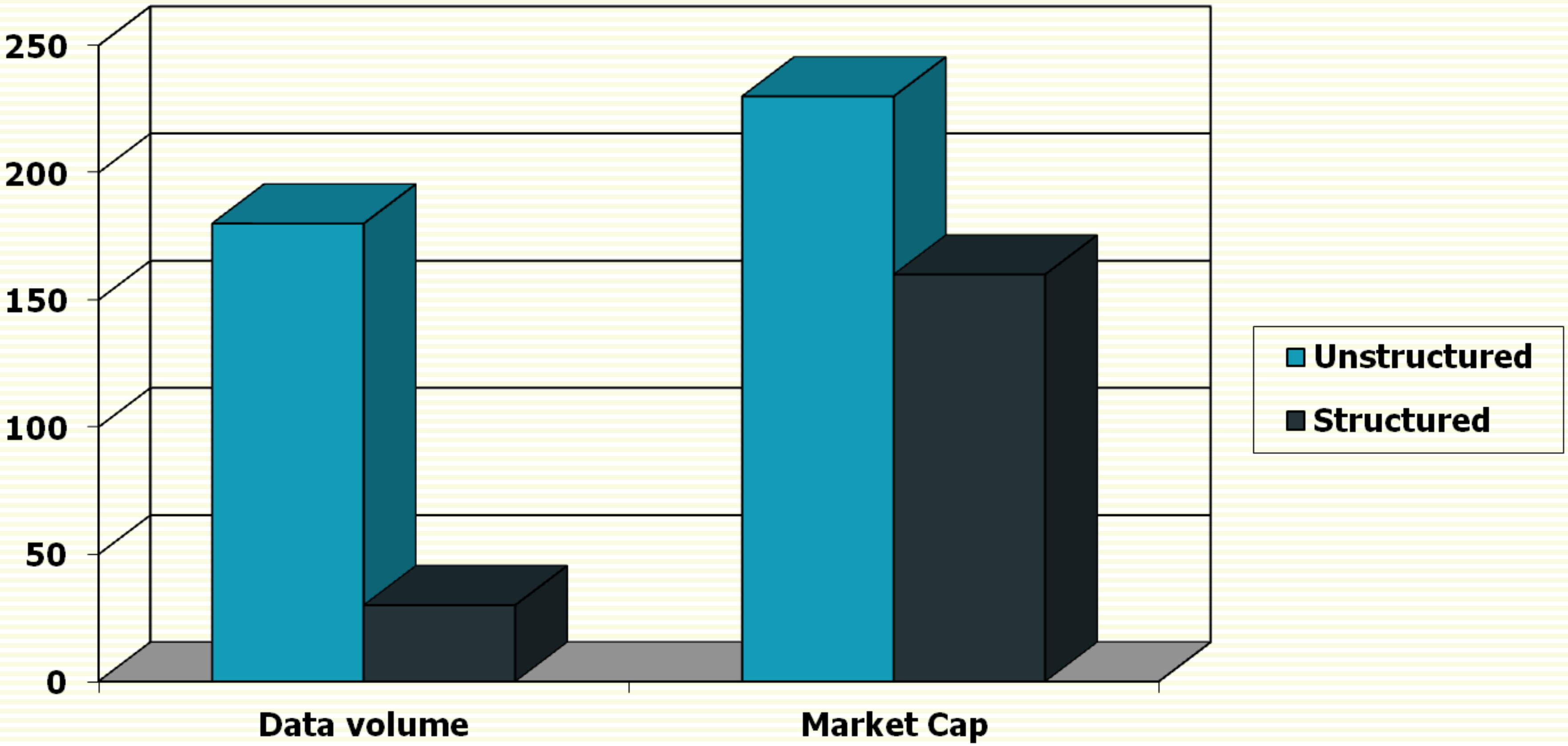
Information Retrieval (**IR**)

- Have a large collection of unstructured documents (usually text)
  - in contrast to databases, which store documents in structured from
- **IR** Goal: retrieve documents with information that is relevant to the need of the user
- Main example is web search, but also
  - E-mail search
  - Searching your laptop
  - Corporate knowledge bases
  - Legal information retrieval

90's: Unstructured vs. Structured Data



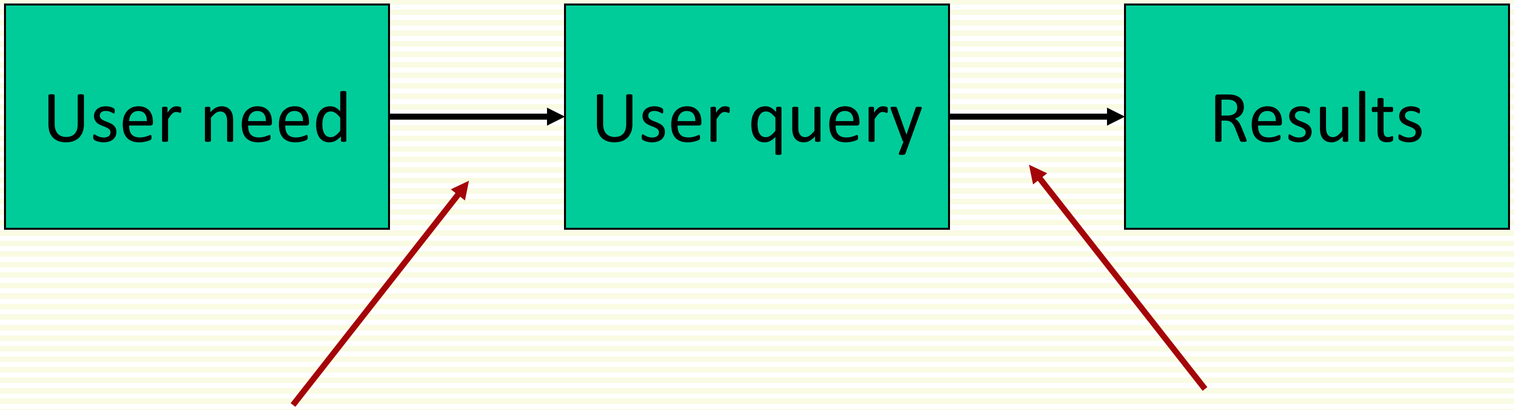
Today: Unstructured vs. Structured Data



Information Retrieval (**IR**)

- Traditionally, dealt with text documents
- More recently
  - Speech
  - Images
  - Music
  - Video

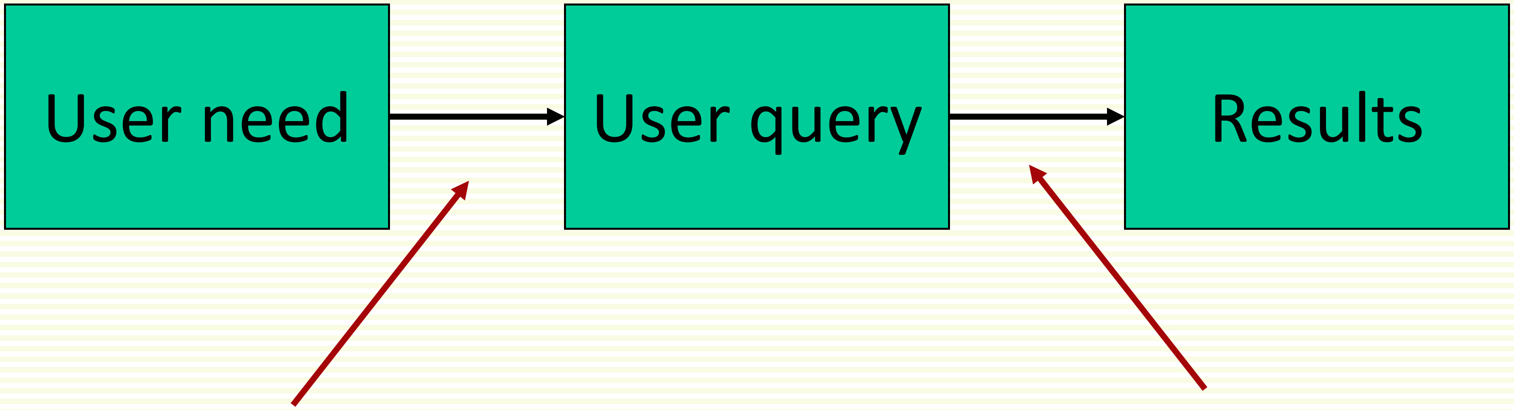
Translating User Needs: Structured data (Databases)



For databases, enlightened users know how to do this correctly, using SQL or a GUI tool

The answers coming out here are precisely what the user wanted

Translating User Needs: Unstructured Data (Text Documents)



For meanings in text, no IR-style query gives one exactly what one wants; it only hints at it

The answers coming out may be roughly what was wanted; often needs to be refined

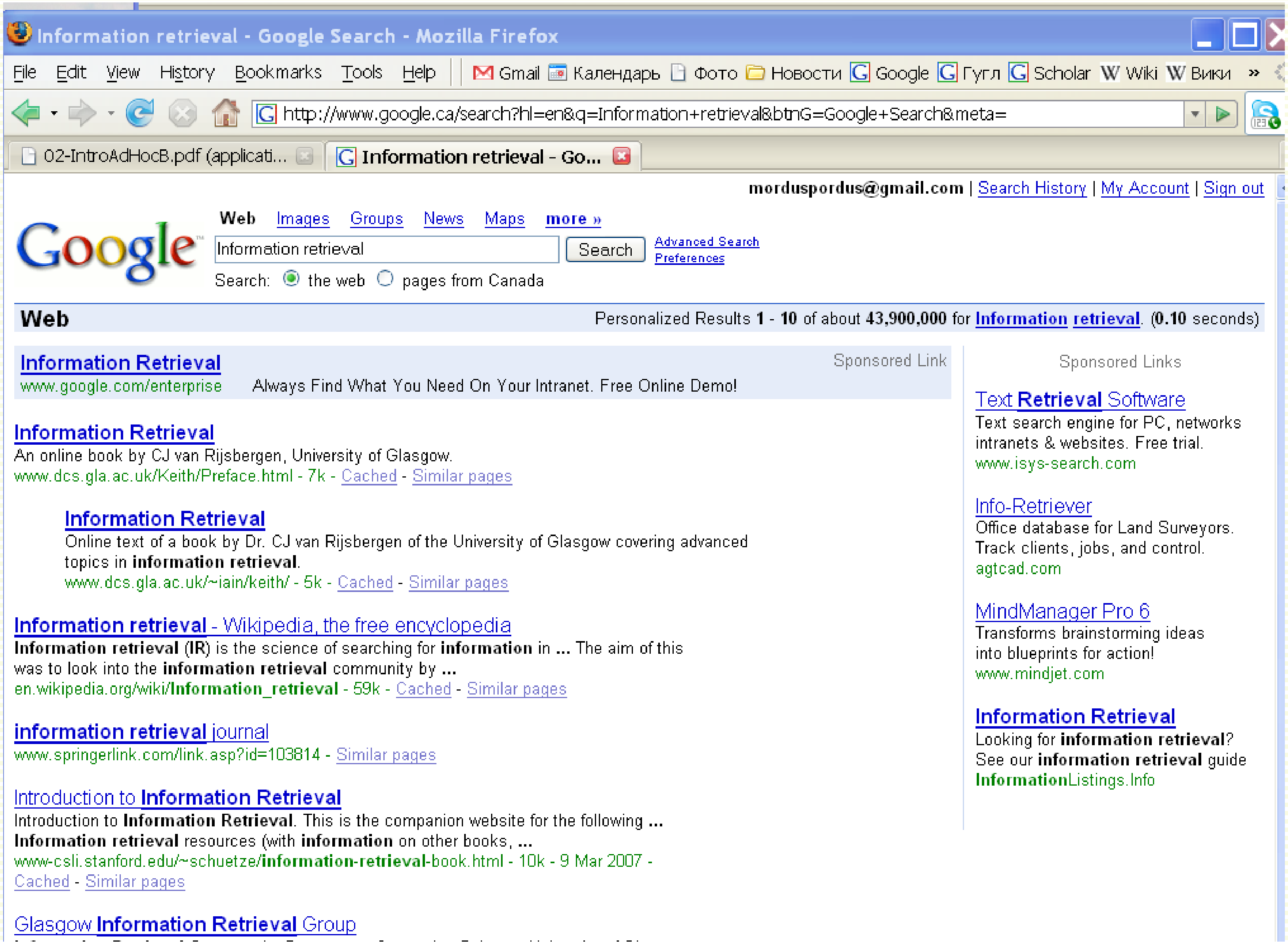
Information Retrieval Types

- Ad-hoc
  - user creates an "ad hoc" query which is not reused or saved
  - system returns a list of (hopefully) relevant documents
  - no training data is available
- Classification/categorization
  - training data is available
  - documents are classified in a pre-determined set of categories
  - Ex: corporate news (CORP-NEWS), crude oil (CRUDE), ...
  - any of machine learning techniques can be used
- Filtering/routing: special case of categorization
  - 2 categories: relevant and not-relevant
  - filtering: absolute assessment ($\mathbf{d}_1$ is relevant but $\mathbf{d}_2$ is not)
  - routing: relative ranking of documents, such as $\mathbf{d}_1$, $\mathbf{d}_2$

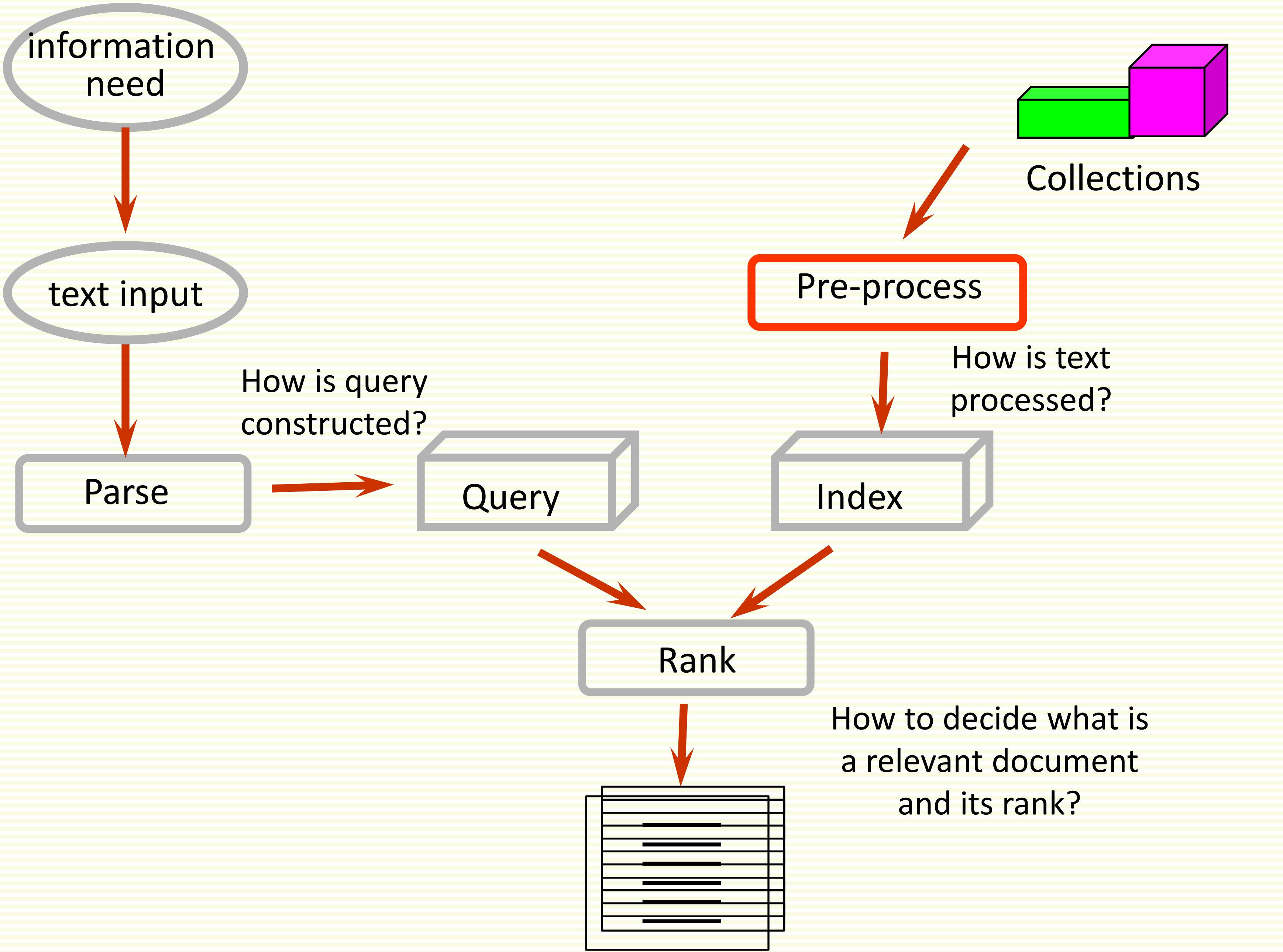
Different Types of Ad-Hoc Retrieval

- Web search
  - Massive document collection (10^8-10^9)
  - Typically high precision (most retrieved documents are relevant), low recall (not all relevant documents are retrieved)
- Commercial information providers (e.g. West, LexisNexis)
  - Large Collection (10^6-10^8) of documents
  - often high recall is essential (e.g. legal or patent search)
- Enterprise search (e.g. UWO, IBM)
  - Medium-sized to large collection (10^4-10^6) of documents
  - Opportunity to exploit domain knowledge
- Personal search (e.g. your PC)
  - Small collection (10^3-10^4) of documents
  - Good opportunity to learn a user model, do personalization

Example of Web Ad-Hoc IR



Information Retrieval Process



Relevance

- In what ways can a document be relevant to a query?
  - Answer precise question precisely
  - Partially answer question
  - Suggest a source for more information
  - Give background information
  - Remind the user of other knowledge
  - Others ...

Two Major Issues

- Indexing
  - How to represent a collection of documents to support fast search?
- Retrieval methods
  - How do we match a user query to indexed documents?

Indexing: Inverted Index

- Most IR systems use **inverted index** to represent text collection
- Inverted Index is a data structure that lists for each word all documents in the collection that contain that word
  - this list is sometimes called *posting list*
  - posting list is sorted by document number

assassination	$\{d_1, d_4, d_{95}, d_{150}, d_{190}\ldots\}$
murder	$\{d_3, d_7, d_{95}\ldots\}$
Kennedy	$\{d_{24}, d_{33}, d_{44}\ldots\}$
conspiracy	$\{d_3, d_{55}, d_{90}, d_{98}\ldots\}$

- Inverted index implemented as a dictionary which allows fast lookups based on word
  - B-trees, hash tables, etc.

Indexing: Inverted Index with Position

- Include position information, document start offset
- Enables efficient search for phrases
- example: need to find *car insurance*

car	(d_1, offset 5), (d_7, offset 10), (d_9, offset 35)
insurance	(d_2, offset 3), (d_7, offset 11), (d_8, offset 7)

$\Downarrow$

car insurance occurs in document 7

- Still primitive: *car insurance* $\neq$ *insurance for car*

Indexing: Inverted Index with Position

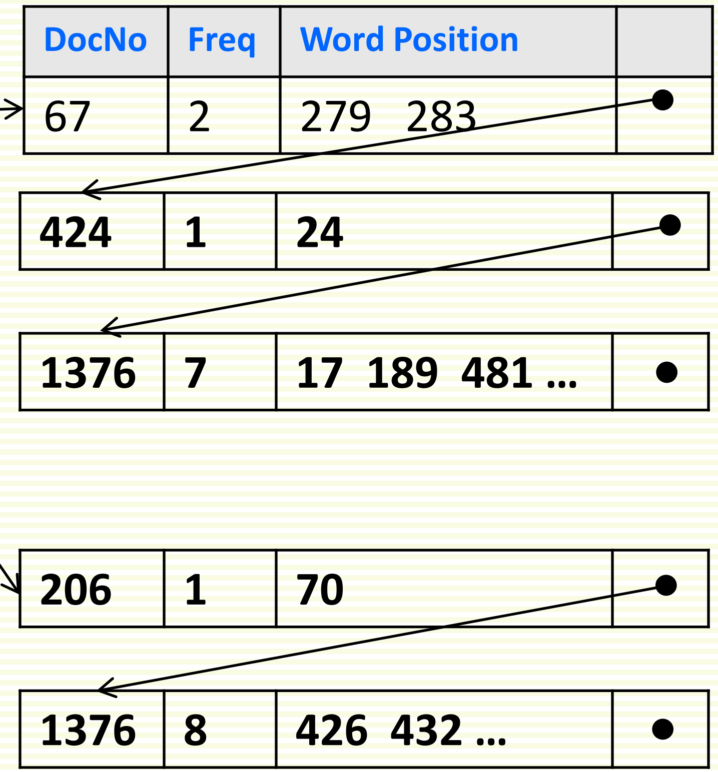
- Still primitive: *car insurance* $\neq$ *insurance for car*
- One solution: find frequent phrases and index those too

car	$\{d_1, d_7, \ldots\}$
car insurance	$\{d_1, d_4, d_{95}, d_{155}, d_{190} \ldots\}$
insurance for car	$\{d_5, d_7, d_{95}, d_{99} \ldots\}$

- Say **term** to refer to these indexed entities
  - sometimes just say word, because it's simpler

Inverted Index Example

Term	DocCnt	FreqCnt	Head
ABANDON	3	10	•
ABB	2	9	•
ABSENCE	135	185	…
ABSTRACT	7	10	…



- For each term:
  - **DocCnt:** in how many documents term occurs
  - **FreqCnt:** total number of times term occurs in all documents
- For each document
  - **Freq**: how many times term occurs in this document
  - **WordPosition**: offset where these occurrences are found in document

Choosing Terms To Index

1. **Controlled Vocabulary** Indexing, done in libraries, web directories
    - A human expert selects a set of terms
    - Pros
        - Usually controlled terms are less unambiguous
    - Cons
        - Expensive, need manual work
        - Controlled vocabularies cannot represent arbitrary detail
2. **Free Text** Indexing, done in some search engines
    - Automatically select good terms to index
3. **Full Text** Indexing, done in most search engines
    - Cons
        - Many ambiguous terms
    - Pros
        - can represent arbitrary detail
        - inexpensive and easy

Full Text Indexing

Are these terms useful?

Term	Tf	Term	Tf	Term	tf
the	78	up	8	pictures	6
to	35	for	7	red	6
i	31	have	7	digital	5
and	29	image	7	eye	5
a	19	like	7	not	5
camera	17	mode	7	on	5
is	17	much	7	or	5
in	12	software	7	shutter	5
with	11	very	7	sony	5
be	9	can	6	than	5
but	9	images	6	that	5
it	9	movies	6	after	4
of	9	my	6	also	4
this	9	no	6	: :	:

Can you tell what this document is about?

Full Text Indexing Design Issues

- To stem or not to stem
  - Stemming: *laughing, laughs, laugh* and *laughed* are stemmed to *laugh*
  - Problem: semantically different words like *gallery* and *gall* may both be truncated to *gall*
- Exclude/Include Stop words
  - Stop words make up about 50% of the text
  - excluding them makes representation more space efficient
  - But impossible to search for documents for phrases containing stop words
    - *to be or not to be*, *take over*
    - most queries are unaffected, but could be very annoying sometimes

After Stemming and Stop Word Removal

Term	Tf	Term	Tf	Term	tf
camera	18	sony	5	lag	3
image	13	after	4	last	3
like	8	any	4	lcd	3
mode	8	auto	4	mavica	3
up	8	battery	4	record	3
buy	7	flash	4	reduce	3
movie	7	problem	4	size	3
picture	7	zoom	4	15	2
software	6	include	3	2mp	2
red	6	2100	3	8x10	2
digital	5	button	3	98	2
eye	5	down	3	automatic	2
look	5	feature	3	bag	2
shutter	5	focus	3	best	2

Problems with Index Terms

- May not retrieve relevant documents that include synonymous terms
  - *restaurant* vs. *café*
  - *PRC* vs. *China*
- May retrieve irrelevant documents that include ambiguous terms
  - *bat* (baseball vs. mammal)
  - *apple* (company vs. fruit)
  - *bit* (unit of data vs. act of eating)

Retrieval Models

- We study 2 basic models:
  - boolean model
    - the oldest one, similar to what is used in database queries
  - vector-space model
    - most popular in IR
- Models vary on:
  - how they represent query and documents
  - how they calculate the relevance between the query and the documents

Boolean Model

- User gives a set of terms (keywords) that are likely to appear in relevant documents
    - *Ex: JFK Kennedy conspiracy assassination*
- Connects the terms in the query with Boolean operators (AND, OR, NOT)

```
AND(Kennedy, conspiracy, assassination)
```

- Can expand query using synonyms

```
AND (OR (Kennedy, JFK),
          (OR (conspiracy, plot),
              (OR (assassination, assassinated,
                   assassinate, murder, murdered, kill, killed)
              ) ))
```

- system returns set of documents that satisfy query **exactly**

Example

- Which of these documents will be returned for the following query :

computer AND (*information* OR *document*) AND *retrieval*

document collection:

d_1: {*computer* √ , *software, information* √, *language*} ×

d_2: {*computer* √, *document* √, *retrieval* √, *library*} √

d_3: {*computer* √, *information* √, *filtering, retrieval* √} √

Implementation With Set Operators

- Assume that the inverted index contains:

  t1-list: {d1,d2,d3,d4}   t2-list: {d1,d2}   t3-list: {d1,d2,d3}   t4-list: {d1}

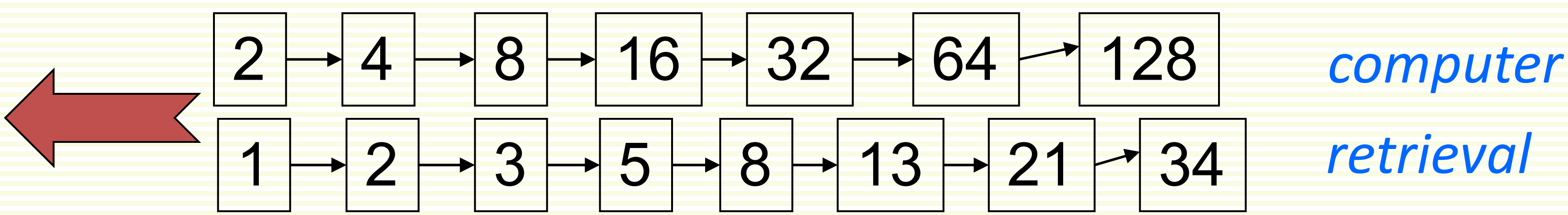
  - The query Q = (t1 **AND** t2) **OR** (t3 **AND** (**NOT** t4))
- We perform set operations:
  - to satisfy (t1 **AND** t2), we **intersect** the t1 and t2 lists
    - $\{d1,d2,d3,d4\} \cap \{d1,d2\} = \{d1,d2\}$
  - to satisfy (t3 **AND** (**NOT** t4)), we **subtract** the t4 list from the t3 list
    - $\{d1,d2,d3\} - \{d1\} = \{d2,d3\}$
  - to satisfy (t1 **AND** t2) **OR** (t3 **AND** (**NOT** t4)), we take the **union** of the two sets of documents obtained for the parts.
    - $\{d1,d2\} \cup \{d2,d3\} = \{d1,d2,d3\}$

Query processing: AND

- Consider processing the query:

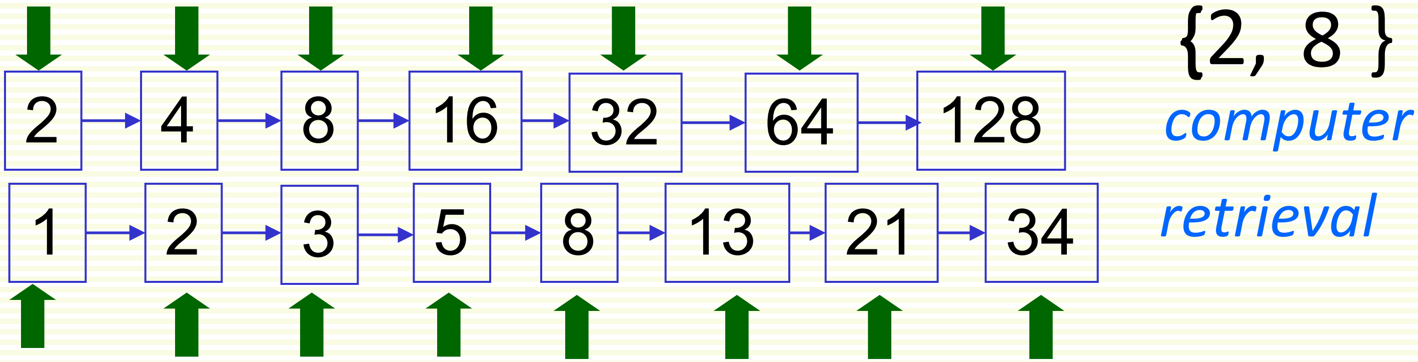
  computer AND *retrieval*

  - Locate *computer* in the Inverted Index
    - retrieve its document list
  - Locate *retrieval* in the Inverted Index
    - retrieve its document list
  - “Merge” (intersect) the document sets:



The Merge

- Crucial: lists are sorted by document ID
- Walk through two lists, in time linear in to total number of entries



- If list lengths are ***n*** and ***m***, merge takes **O**(**n**+**m**) time

Analysis of the Boolean Model

- Advantages
  - queries are expressed with Boolean operators, i.e. semantics is clearly defined
  - results are easy to explain
  - computationally efficient
  - useful for expert users
- Disadvantages
  - retrieval strategy is a binary decision (relevant or not)
  - difficult to *rank* documents in order of relevance
  - non-expert users have difficulty to express their need as Boolean expressions
    - “Feast of Famine” phenomena, people create quires that are either
      - **too strict**: few relevant documents are found
      - **too loose**: too many documents, most irrelevant, are found
    - Most boolean searches on the web either return no documents or a huge set of documents

Ranked Retrieval Models

- Rather than a set of documents **exactly** satisfying a query expression, in ranked retrieval models, the system returns an ordering over the (top) documents in the collection with respect to a query
  - large set of retrieved documents is not a problem, just show top 10 ranked documents
- Free text queries: rather than a query language of operators and expressions, the user query is just one or more words in a human language

Vector-Space Model

- Documents and queries are represented by a ***term vector***
  - each dimension corresponds to a term in the vocabulary
- Similarity between a document and a query is determined by a distance in vector space
- First system SMART
  - developed by G. Salton at Cornell 1960-1995
  - still used widely today

Gerard Salton

Term-Document Matrix

- term-by-document matrix visualizes the collection of documents

	d_1	d_2	d_3	d_4	d_5	…
$term_1$	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	
$term_2$	w_{21}	w_{22}	w_{23}	w_{24}	w_{25}	
$term_3$	w_{31}	w_{32}	w_{33}	w_{34}	w_{35}	
…						
$term_N$	w_{n1}	w_{n2}	w_{n3}	w_{n4}	w_{n5}	

- 1 column = representation of one document
- 1 row = representation of one term across all documents
- cell $\mathbf{w_{ij}}$ = weight of term **i** in document **j**
  - simplest weight $\mathbf{w_{ij}}$ is the count of times term **i** occurred in document **j**
- matrix is sparse, i.e. most weights are 0
  - Implemented with inverted index, matrix is useful just for visualization

Term-Document Count Matrix

- Consider 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
	document 1	document 2				

Bags of Words

- This representation sometimes called **bags of words**
  - the document is the **bag**
  - **bag** contains word tokens
  - Word order is ignored

    I see what I eat = I eat what I see
  - A particular word may occur more than once in the bag

document 1

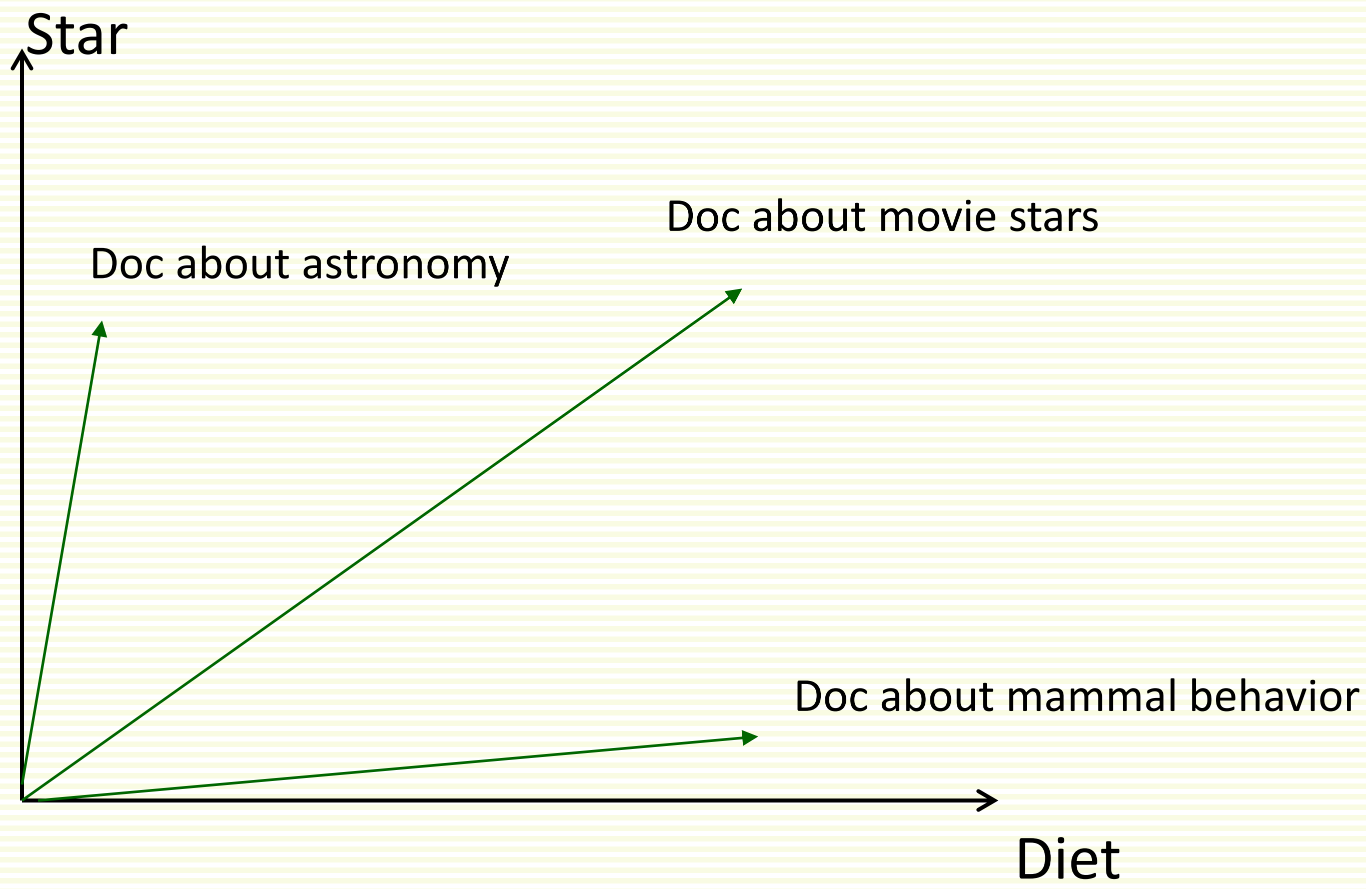
The quick brown fox jumped over the lazy dog's back.

document 2

Now is the time for all good men to come to the aid of their party.

indexed term	document 1	document 2
aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Documents as Vectors



Documents as Vectors

- $|V|$-dimensional vector space, where $|V|$ is the number of terms
- 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
- Very sparse vectors – most entries are zero

Queries as Vectors

- Key idea 1
  - represent queries also as vectors in the same vector space
- Key idea 2
  - Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Use proximity to get away from "you're-either-in-or-out" Boolean model
- Instead: rank more relevant documents higher than less relevant documents

Query Representation

- A query can also be represented as a vector, like a document

$$\mathbf{q} = (0,0,0,1,0,\ldots 1,..0,1)$$

- Size of vector corresponding to query **q** is also the number of index terms $|\mathbf{V}|$

Example

- The collection:
  - d_1 = {introduction knowledge in speech and language processing ambiguity models and algorithms language thought and understanding the state of the art and the near-term future some brief history summary}
  - d_2 = {hmms and speech recognition speech recognition architecture overview of the hidden markov models the Viterbi algorithm revisited advanced methods in decoding acoustic processing of speech computing acoustic probabilities training a speech recognizer waveform generation for speech synthesis human speech recognition summary}
  - d_3 = {language and complexity the chomsky hierarchy how to tell if a language isn’t regular the pumping lemma are English and other languages regular languages ? is natural language context-free complexity and human processing summary}
- The query:

  Q = {speech language processing}

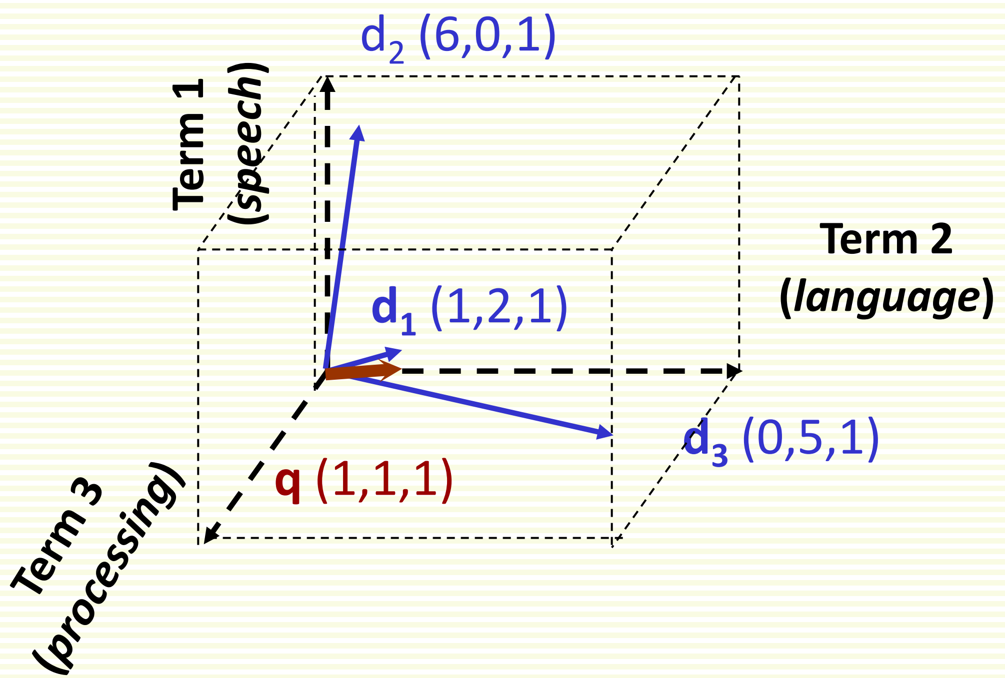
Example Continued

- The collection:
  - d_1 = {introduction knowledge in speech and language processing ambiguity models and algorithms language thought and understanding the state of the art and the near-term future some brief history summary}
  - d_2 = {hmms and speech recognition speech recognition architecture overview of the hidden markov models the viterbi algorithm revisited advanced methods in decoding acoustic processing of speech computing acoustic probabilities training a speech recognizer waveform generation for speech synthesis human speech recognition summary}
  - d_3 = {language and complexity the chomsky hierarchy how to tell if a language isn't regular the pumping lemma are English and other language regular language ? is natural language context-free complexity and human processing summary}
- The query:

  Q = {speech language processing}

Example Continued

	d_1	d_2	d_3	q
introduction	…	…	…	…
knowledge	…	…	…	…
…	…	…	…	…
speech	1	6	0	1
language	2	0	5	1
processing	1	1	1	1
…	…	…	…	…



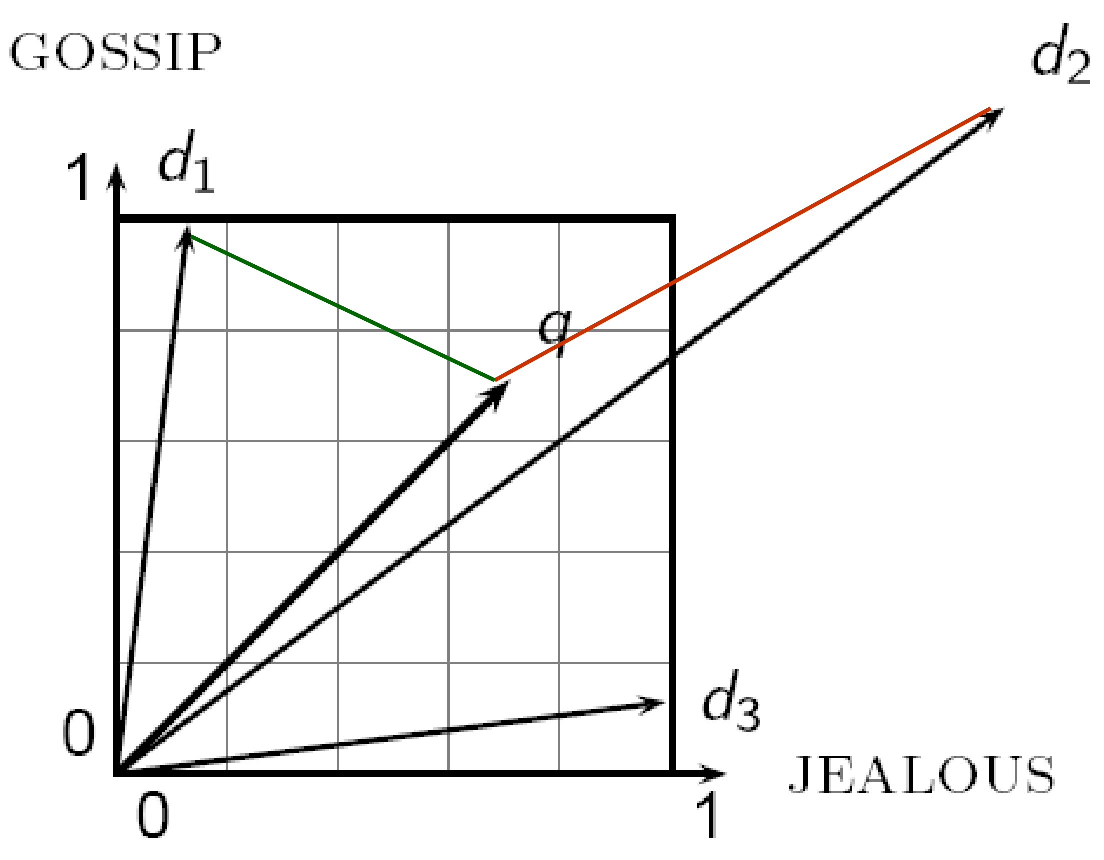
- using raw term frequencies for weights

Vector Space Proximity

- First idea: use standard Euclidean distance
  - does not work well
  - because Euclidean distance is large for vectors of different lengths
  - documents tend to vary in lengths widely

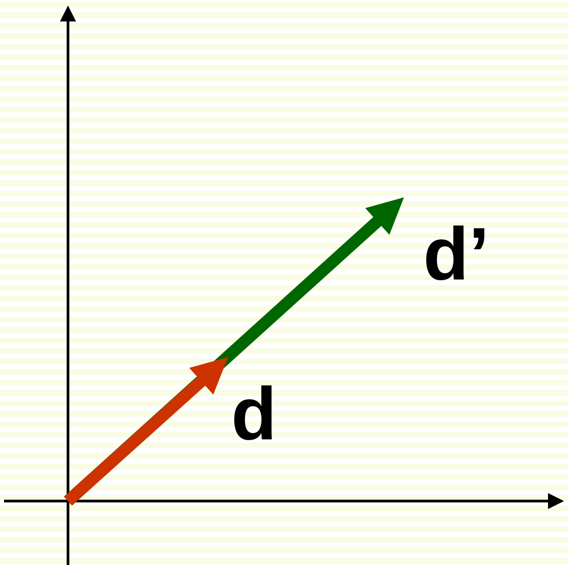
Why Eucledian Distance is a Bad Idea

- Euclidean distance between **q** and $\mathbf{d_2}$ is large even though distribution of terms in query **q** and document $\mathbf{d_2}$ are similar
- Query **q** is closer to $\mathbf{d_1}$



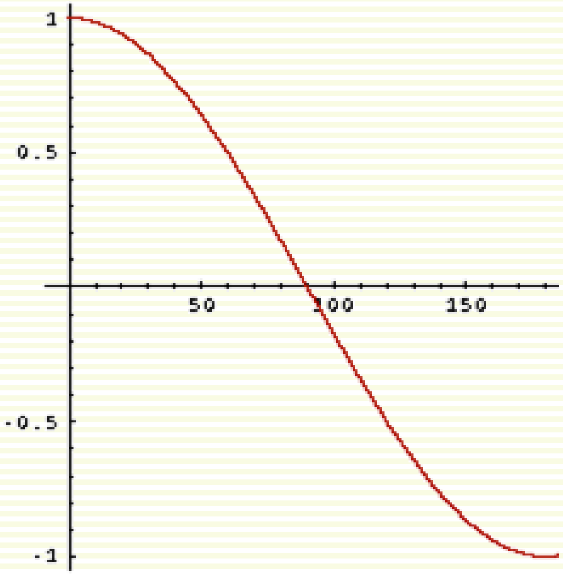
Use Angle Instead

- Thought experiment
  - take a document **d** and append it to itself
  - call this document **d’**
- Semantically **d** and **d’** have the same content
  - **d** is a short document, **d’** is a long document
- Euclidean distance between the two documents can be quite large
- Angle between the two documents is 0, corresponding to maximal similarity
- Key idea: rank documents according to the angle with the query



From Angles to Cosines

- These two are equivalent:
  - rank documents in *decreasing* order of the angle between query and document
  - rank documents in *increasing* order of **cosine**(query,document)
  - Why cosine? For efficiency
- Cosine is a monotonically decreasing function for the interval [0^o, 180^o]
- Negative between [90,180]
  - but this is not a problem



Length Normalization

- Normalize vectors by dividing each of its components by its length

$$\left\|\mathbf{x}\right\|_2 = \sqrt{\sum_i \mathbf{x}_i^2}$$

- After normalization, each vector has unit (1) length
- Let **d’** = **d+d** (**d** appended to itself)
- After normalization, **d** and **d’** are identical
- long and short documents now have comparable weight

Cosine for Length Normalized Vectors

Dot product

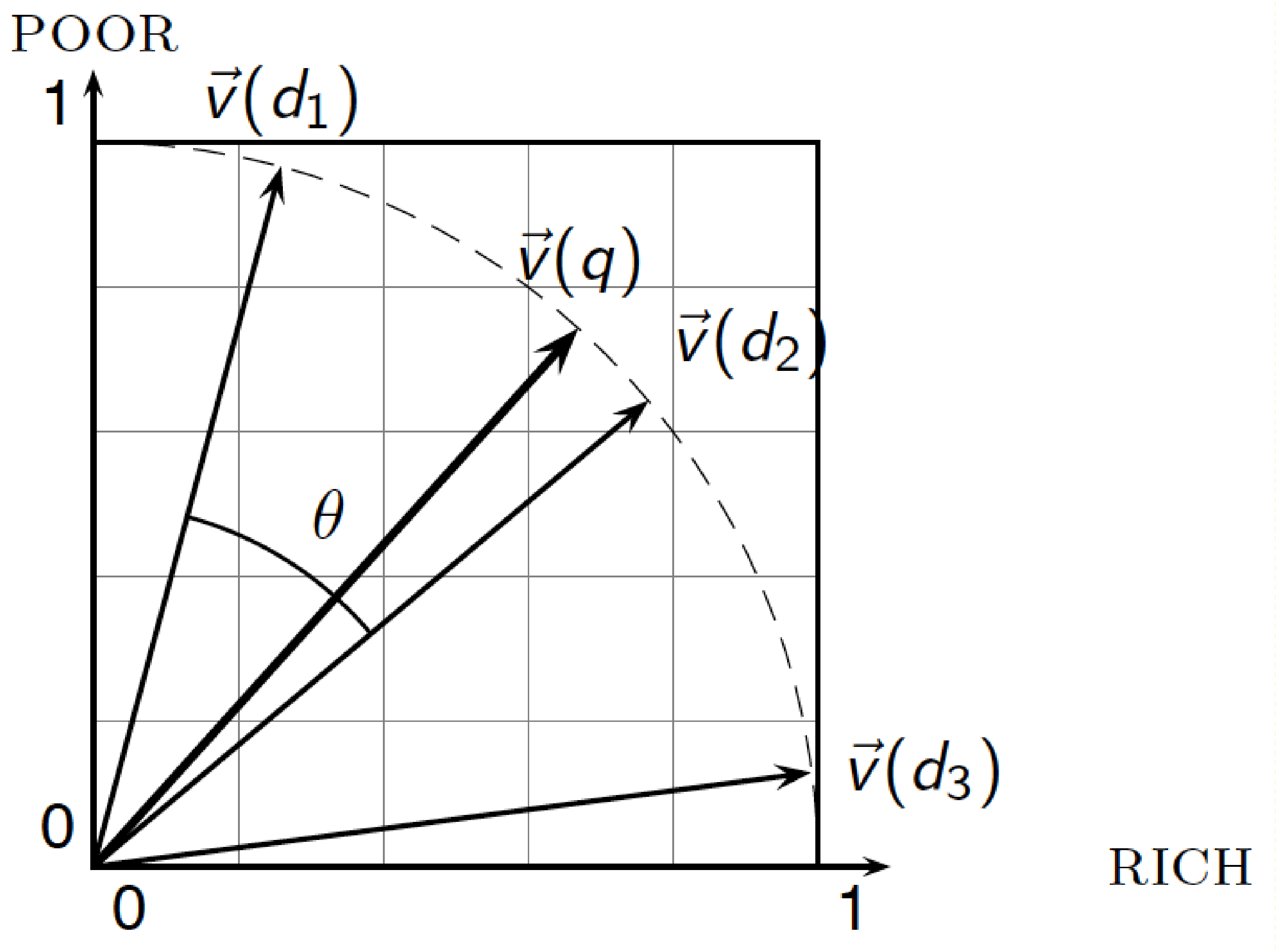
Unit vectors

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

- For length-normalized vectors, cosine similarity is simply the dot product

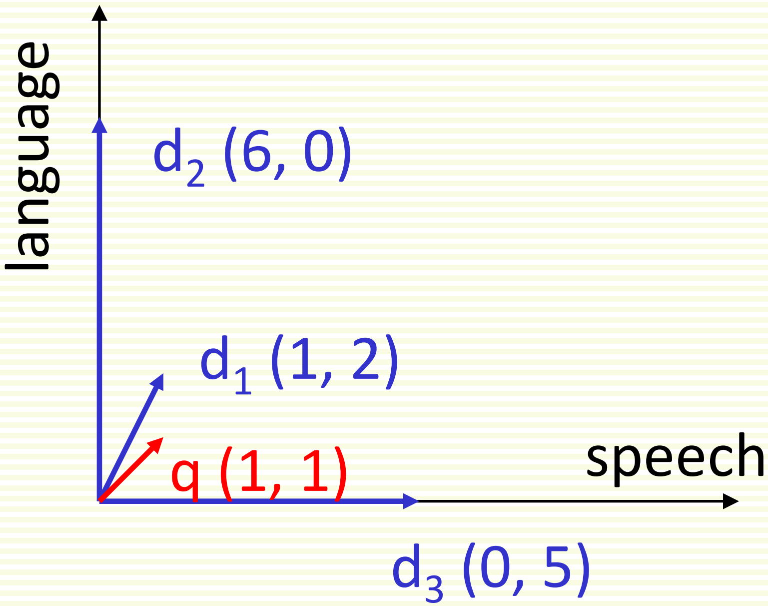
$$\mathbf{cos}(\mathbf{q},\mathbf{d}) = \mathbf{q}\cdot\mathbf{d} = \sum_{\mathbf{i}=1}^{|\mathbf{V}|}\mathbf{q}_\mathbf{i}\mathbf{d}_\mathbf{i}$$

Cosine Similarity Illustrated



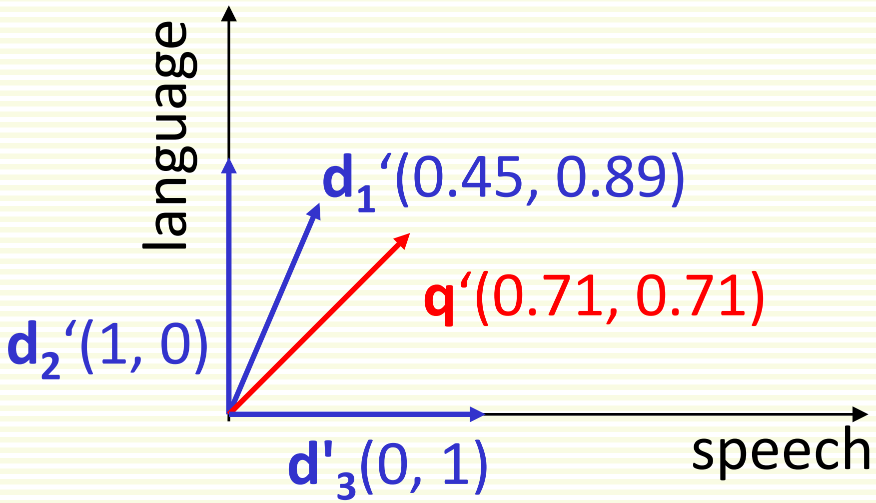
Example

- assume only two indexed terms, *speech* and *language*
- query *q* = *speech language*
- original representation



Example: Normalized vectors

- query q = *speech language*
- after normalization



$\mathbf{q}(1,1)$: $\mathbf{L} = \sqrt{\mathbf{1}^2 + \mathbf{1}^2} = \mathbf{1.41} \Rightarrow$ normalized $\mathbf{q'}$ (0.71, 0.71)

$\mathbf{d_1}(1,2)$: $\mathbf{L} = \sqrt{\mathbf{1}^2 + \mathbf{2}^2} = \mathbf{2.24} \Rightarrow$ normalized $\mathbf{d_1'}$ (0.45, 0.89)

$\mathbf{d_2}(6,0)$: $\mathbf{L} = \sqrt{\mathbf{6}^2 + \mathbf{0}^2} = \mathbf{6} \Rightarrow$ normalized $\mathbf{d_2'}$ (1, 0)

$\mathbf{d_3}(0,5)$: $\mathbf{L} = \sqrt{\mathbf{0}^2 + \mathbf{5}^2} = \mathbf{5} \Rightarrow$ normalized $\mathbf{d_3'}$ (0, 1)

Term Frequency **tf**

- Are word counts or binarized counts (bag of word) the best representation for document vectors?
- Define the number of occurrences of a term **t** in a document is **d** term frequency $\mathbf{tf}_{td}$
- Want to use **tf** when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - document with 10 occurrences of term is more relevant than document with 1 occurrence of term
  - but probably 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

$$\mathbf{w}_{\mathbf{td}} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{\mathbf{td}} & \text{if} \quad \mathrm{tf}_{\mathbf{td}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- 0 → 0
- 1 → 1
- 2 → 1.3
- 10 → 2
- 1000 → 4
- document that has 10 times more occurrences of a term is only 2 times more important with one occurrence of a term

Document Frequency

- Rare terms are more informative than frequent terms
  - recall stop words *the*, *in*, *from* ,…
- Consider a term in query that is rare in the collection
  - e.g., *arachnocentric*
- Document containing this term is very likely to be relevant to the query *arachnocentric*
- Want a higher weight for rare terms like *arachnocentric*
- The more rare the word, the higher its weight
  - word is rare if it does not occur in many documents
- Use document frequency (**df**) to capture this

idf weight

- **df_t** the document frequency of **t** is the number of documents that contain **t**
  - **df_t** is an inverse measure of the informativeness of **t**
  - **$df_t \le N$**, where **N** is the number of documents
- Define **idf** (inverse document frequency) of **t**

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

  - as before, use log (**N/df_t**) instead of **N/df_t** to dampen (lessen) the effect of **idf**
  - the base of the log is of little importance

idf Example

- Suppose **N** = 10^6

term	**df$_t$**	**idf$_t$ = log$_{10}$(N/df$_t$)**
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

Effect of idf on Ranking

- Does **idf** have an effect on ranking for one-term queries, like *iPhone*
- No effect on ranking one term queries
  - Just scales all documents by the same factor
- **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

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

- Best known weighting scheme in information retrieval
  - note: the "-" in tf-idf is a hyphen, not a minus sign
  - alternative names: tf.idf, tf x idf
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Analysis of the Vector Space Model

- advantages:
  - simple and effective
  - term-weighting scheme improves retrieval performance
  - partial matching allows for retrieval of documents that approximate the query
  - cosine ranking allows for sorting the results
- disadvantages
  - no real theoretical basis for the assumption of a term space
  - assumed independence between terms is not really true
- Note: in web search engines the weights may be calculated differently
  - heuristics on where a term occurs in the document (ex, title)
  - notion of *hub* and *authority*

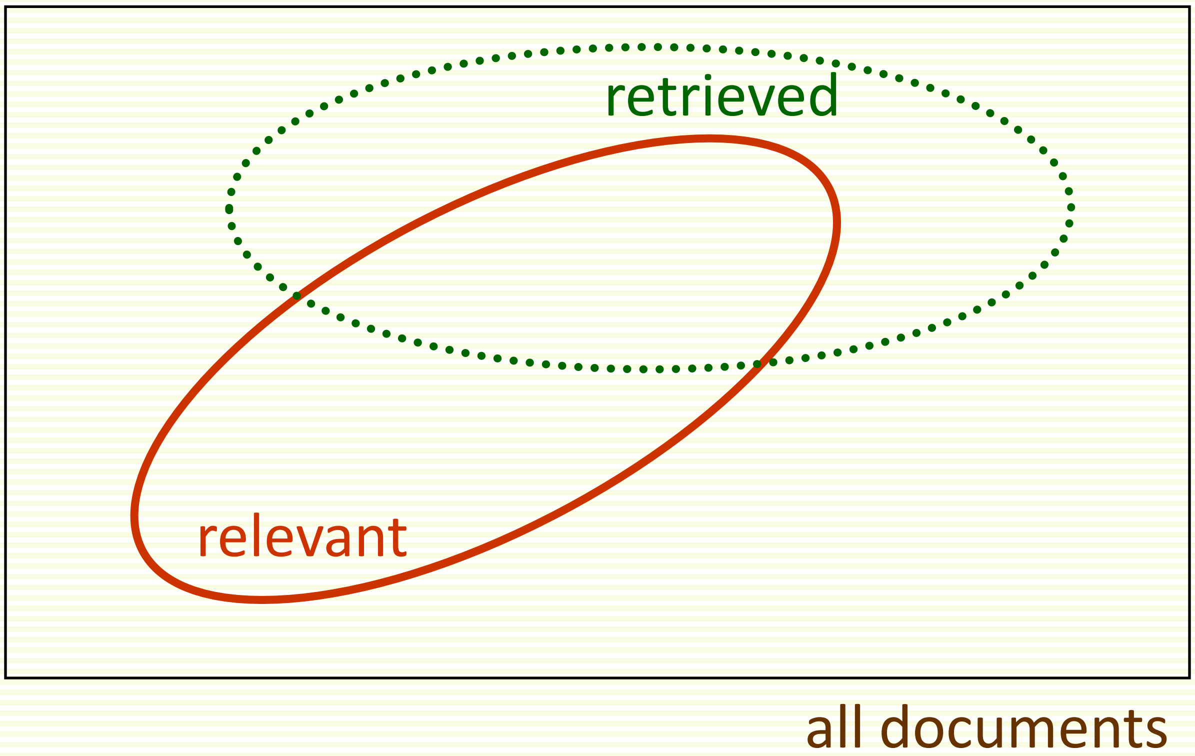
Evaluation

- Suppose have several retrieval methods
- Which one is the best?
  - for us, best = effectiveness, or the relevance of retrieved documents
  - other possible measures: ease of use, efficiency, nice interface, cost, etc.
- An **information need** is translated into a **query**
- Relevance is assessed relative to the **information need** *not* the **query**
- **Information need**: *I'm looking for information on whether drinking red wine is more effective at reducing your risk of heart attacks than white wine.*
- **Query**: ***wine red white heart attack effective***
- Evaluate whether retrieved document addresses the information need, not whether it has these words

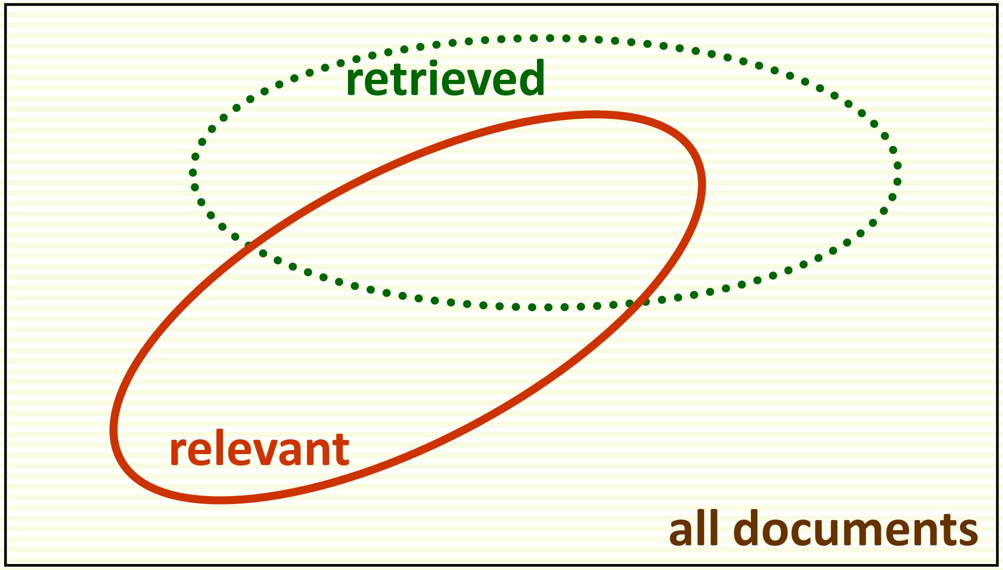
Evaluation

- To evaluate, need
  - a benchmark document collection
  - a benchmark set of queries
  - a set of relevance query/document judgments
- To compare two (or more) methods
  - Each method is used to retrieve documents for a query
  - Results are compared using some measures
  - Common measures are based on **precision** and **recall**

Relevant vs. Retrieved



Precision vs. Recall



$$\text{precision} = \frac{\text{number of relevant documents retrieved}}{\text{number of documents retrieved}} = \frac{|O \cap O|}{|O|}$$

$$\text{recall} = \frac{\text{number of relevant documents retrieved}}{\text{number of relevant documents in collection}} = \frac{|O \cap O|}{|O|}$$

Evaluation: Example of P andR

- Relevant: d_3 d_5 d_9 d_{25} d_{39} d_{44} d_{56} d_{71} d_{123} d_{389}
- System 1
  - d_{123} d_{84} d_{56}
  - Precision ?
  - Recall ?
- System 2
  - d_{123} d_{84} d_{56} d_6 d_8 d_9
  - Precision ?
  - Recall ?

Evaluation: Example of P&R

- Relevant: d_3 d_5 d_9 d_{25} d_{39} d_{44} d_{56} d_{71} d_{123} d_{389}

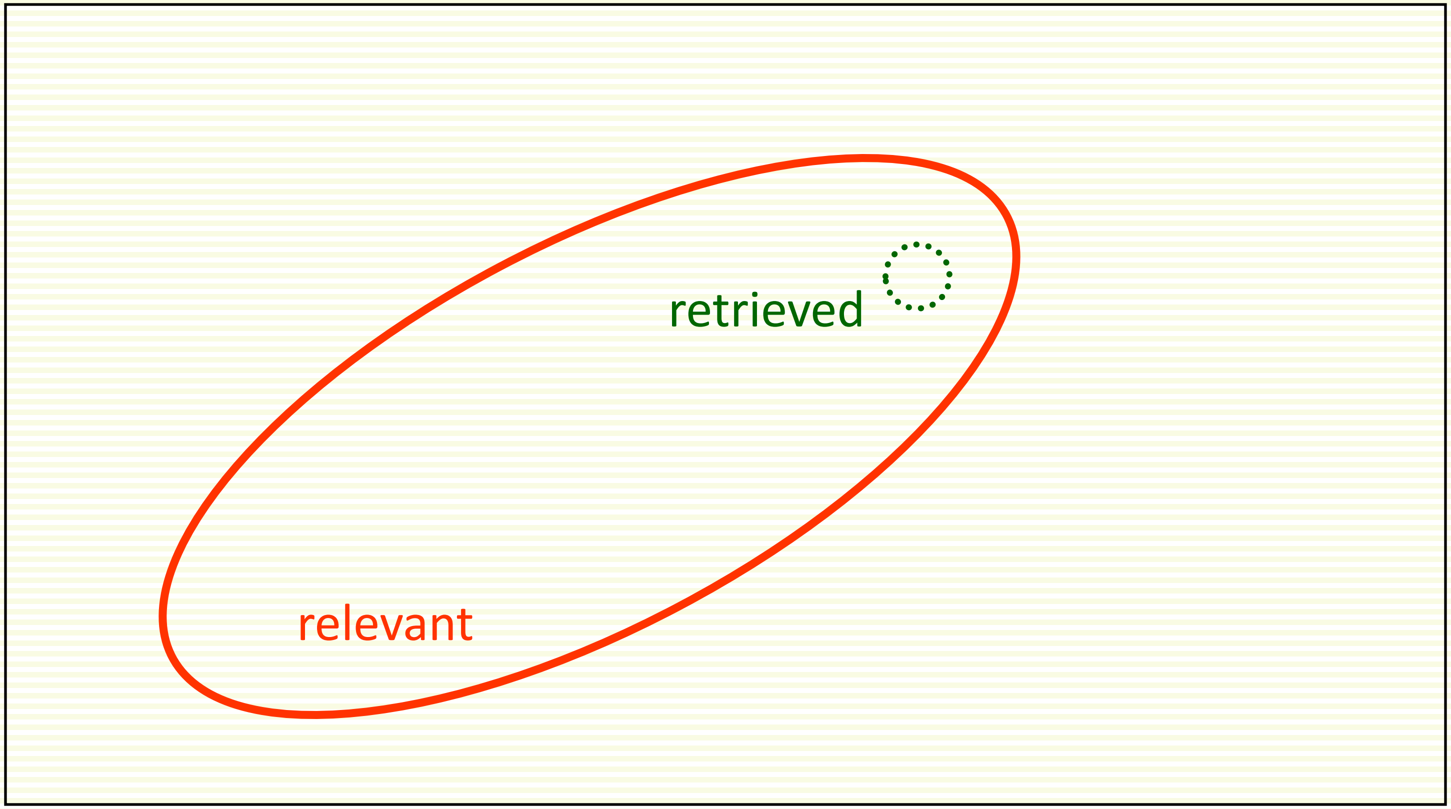
- System 1:
  - d_{123} √ d_{84} × d_{56} √
  - precision = 2/3 = 66%
  - recall = 2/10 = 20%

- System 2:
  - d_{123} √ d_{84} × d_{56} √ d_6 × d_8 × d_9 √
  - precision = 3/6 = 50%
  - recall = 3/10 = 30%

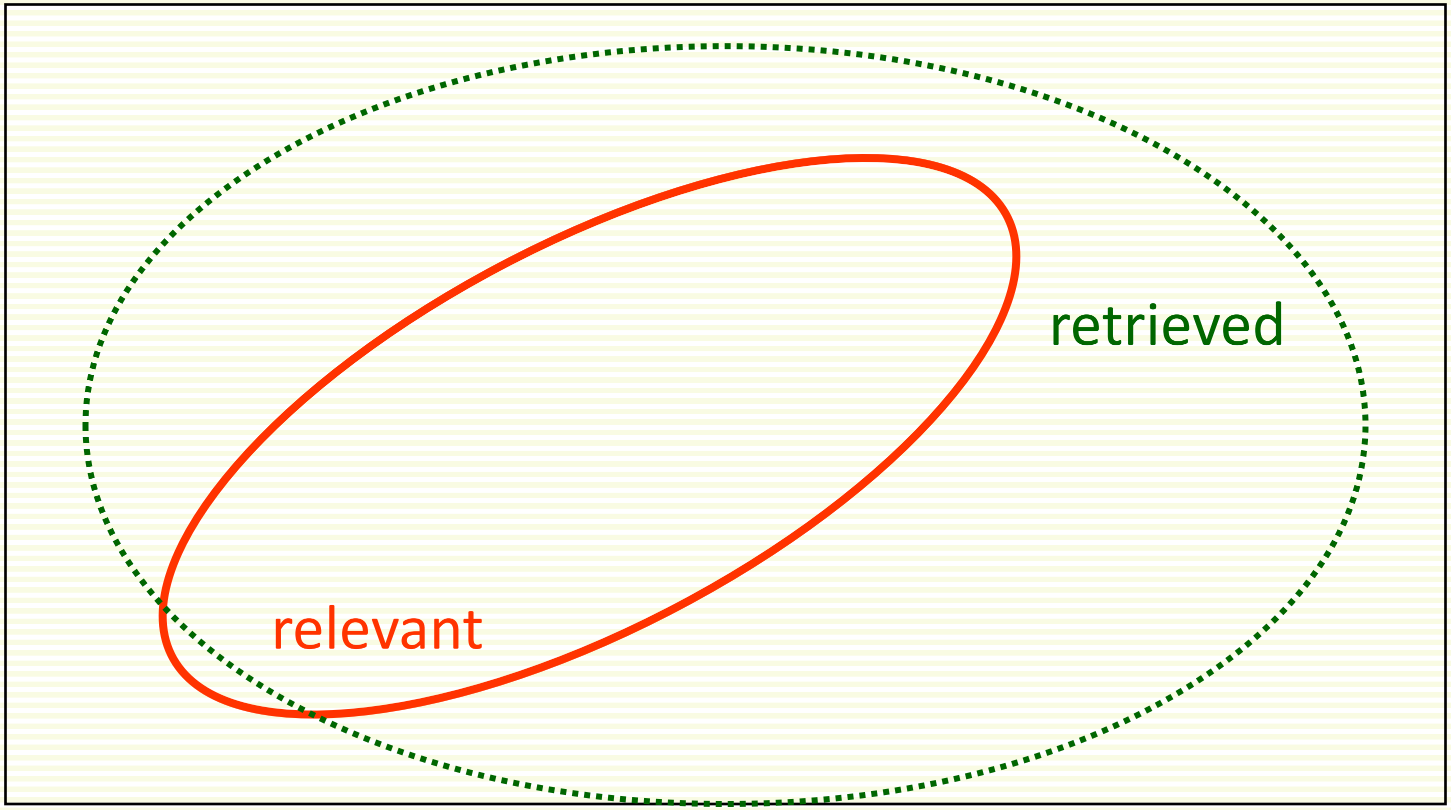
Why Precision and Recall?

- Get as much good stuff (high recall) while at the same time getting as little junk as possible (high precision)
- Easy to get either high recall or high precision
- Harder to get both high

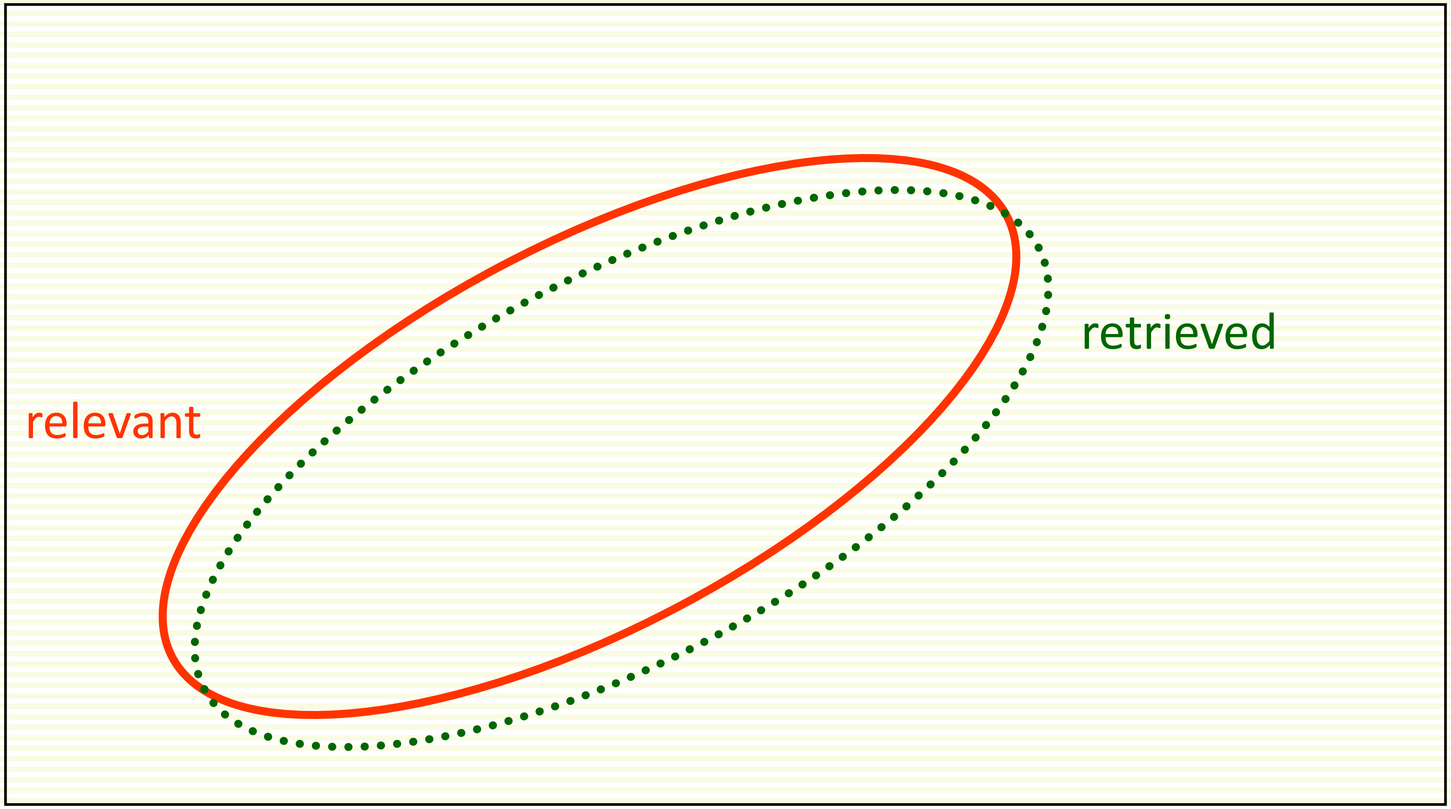
High Precision, Low Recall



High Recall, Low Precision

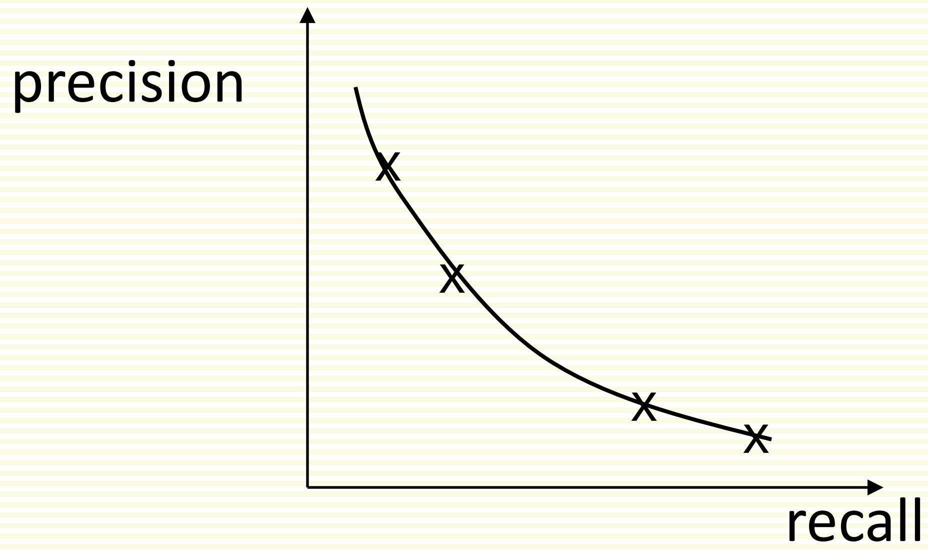


High Precision, High Recall



Precision/Recall Curves

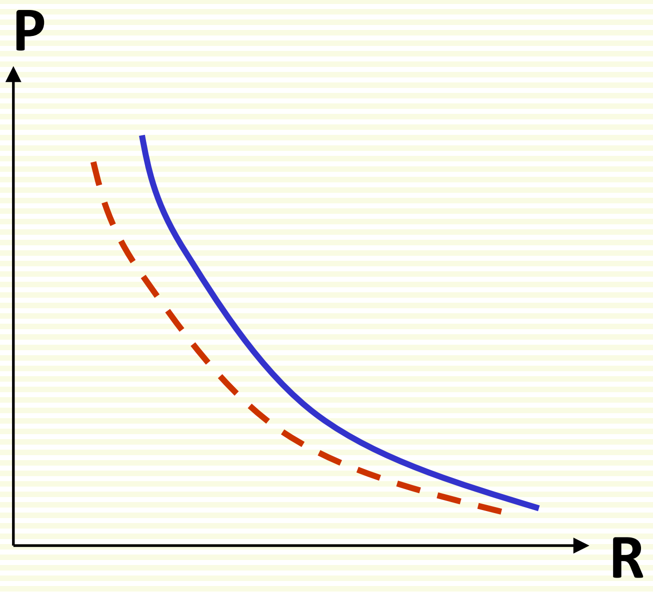
- There is a tradeoff between Precision and Recall
  - easy to get either high precision or high recall, but not both
- So measure Precision at different levels of Recall
- Note: this is an **average** over **many** queries



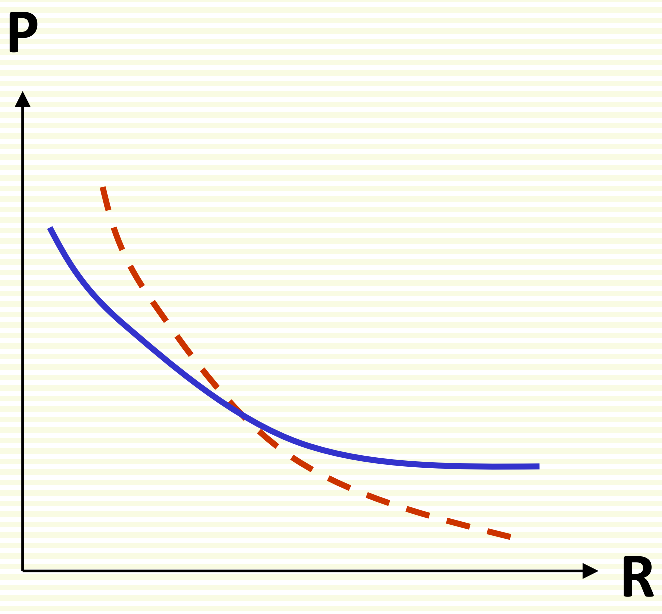
Precision/Recall Curves

- Often difficult to determine which system is better
  - Is blue method performing better than the red one?

Yes!



?



F-Measure

- Sometime only one pair of precision and recall is available
  - e.g., filtering task
- F-Measure

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}}$$

  - α > 0.5: precision is more important
  - α < 0.5: recall is more important
  - Usually α = 0.5

$$F = 2 \cdot \frac{P \cdot R}{P + R}$$

Importance of Ranking

- IR systems typically output a *ranked* list of documents
- Should take *relevance* into account when measuring performance
- The three systems have same precision/recall rates, but the method in the first column is better since it ranks the relevant documents higher

system 1	*system 2*	*system 3*
d1 √	d10 ×	d6 ×
d2 √	d9 ×	d1 √
d3 √	d8 ×	d2 √
d4 √	d7 ×	d10 ×
d5 √	d6 ×	d9 ×
d6 ×	d1 √	d3 √
d7 ×	d2 √	d5 √
d8 ×	d3 √	d4 √
d9 ×	d4 √	d7 ×
d10 ×	d5 √	d8 ×

Cutoff

- Look at precision of the top 5 (or 10, … etc) ranked documents

	system 1	*system 2*	*system 3*
	d1 √	d10 ×	d6 ×
	d2 √	d9 ×	d1 √
	d3 √	d8 ×	d2 √
	d4 √	d7 ×	d10 ×
	d5 √	d6 ×	d9 ×
	d6 ×	d1 √	d3 √
	d7 ×	d2 √	d5 √
	d8 ×	d3 √	d4 √
	d9 ×	d4 √	d7 ×
	d10 ×	d5 √	d8 ×
precision at 5	1.0	0.0	0.4
precision at 10	0.5	0.5	0.5

- How to decide on the “cut off” threshold?
  - threshold 5 is informative in this example, threshold 10 is not informative

Uninterpolated Average Precision

- Instead of using a single cut-off, average precision at many cut-offs usually at points where a relevant document is found

For system 3:

- At cutoff **d1**:
  2 retrieved, 1 relevant,
  precision ½
- At cutoff **d2**:
  3 retrieved, 2 relevant,
  precision 2/3
- …
- At cutoff **d4**:
  8 retrieved, 5 relevant,
  precision 5/8
- **Average precision 0.5726**

	system 1	*system 2*	*system 3*
	d1 √	d10 ×	d6 ×
	d2 √	d9 ×	d1 √ 1/2
	d3 √	d8 ×	d2 √ 2/3
	d4 √	d7 ×	d10 ×
	d5 √	d6 ×	d9 ×
	d6 ×	d1 √	d3 √ 3/6
	d7 ×	d2 √	d5 √ 4/7
	d8 ×	d3 √	d4 √ 5/8
	d9 ×	d4 √	d7 ×
	d10 ×	d5 √	d8 ×
precision at 5	1.0	0.0	0.4
precision at 10	0.5	0.5	0.5
aver. precision	1.0	0.3544	0.5726

IR System Improvements

- Most Queries are short
  - Web queries tend to be 2-3 keywords long
- The two big problems with short queries are:
  - Synonymy: poor **recall** results from missing documents that contain synonyms of search terms, but not the terms themselves
  - Polysemy/Homonymy: poor **precision** results from search terms that have multiple meanings leading to the retrieval of non-relevant documents

Query Expansion

- Find a way to expand a user's query to automatically include relevant terms (that they should have included themselves), in an effort to improve recall
  - Use a dictionary/thesaurus
  - Use relevance feedback

Query Expansion

- Example:
  - query: seller of email solutions for cell phones
  - document: […] Giszmotron is a leading vendor of electronic messaging services for cellular devices […]
- Solution: expand user query with related terms
  - often using a thesaurus to find related terms (synonyms, hyponyms)
  - new terms will have lower weights in the query
  - ex: expanded query: seller *vendor* phones *device* …

Relevance Feedback

- Ask the user to identify a few documents which appear to be related to their information need
- Extract terms from those documents and add them to the original query
- Run the new query and present those results to the user
- Iterate (ask the user to identify relevant documents…extract terms… add them to the query…)
  - Typically converges quickly

Blind Feedback

- Assume that first few documents returned are most relevant rather than having users identify them
- Proceed as for relevance feedback
- Tends to improve recall at the expense of precision

Additional IR Issues

- In addition to improved relevance, can improve overall information retrieval with some other factors
  - Eliminate duplicate documents
  - Provide good context
- For the web:
  - Eliminate multiple documents from one site
  - Clearly identify paid links

IR within NLP

- IR needs to process the large volumes of online text
- And (traditionally), NLP methods were not *robust* enough to work on thousands of real world texts.
- so IR:
  - not based on NLP tools (ex. syntactic/semantic analysis)
  - uses (mostly) simple (shallow) techniques
  - based mostly on word frequencies
- in IR, meaning of documents:
  - is the composition of meaning of individual words
  - ordering & constituency of words play are not taken into account
  - *bag of word* approach

    I see what I eat.

    I eat what I see.

    same meaning