# ECE8813
# Statistical Natural Language Processing

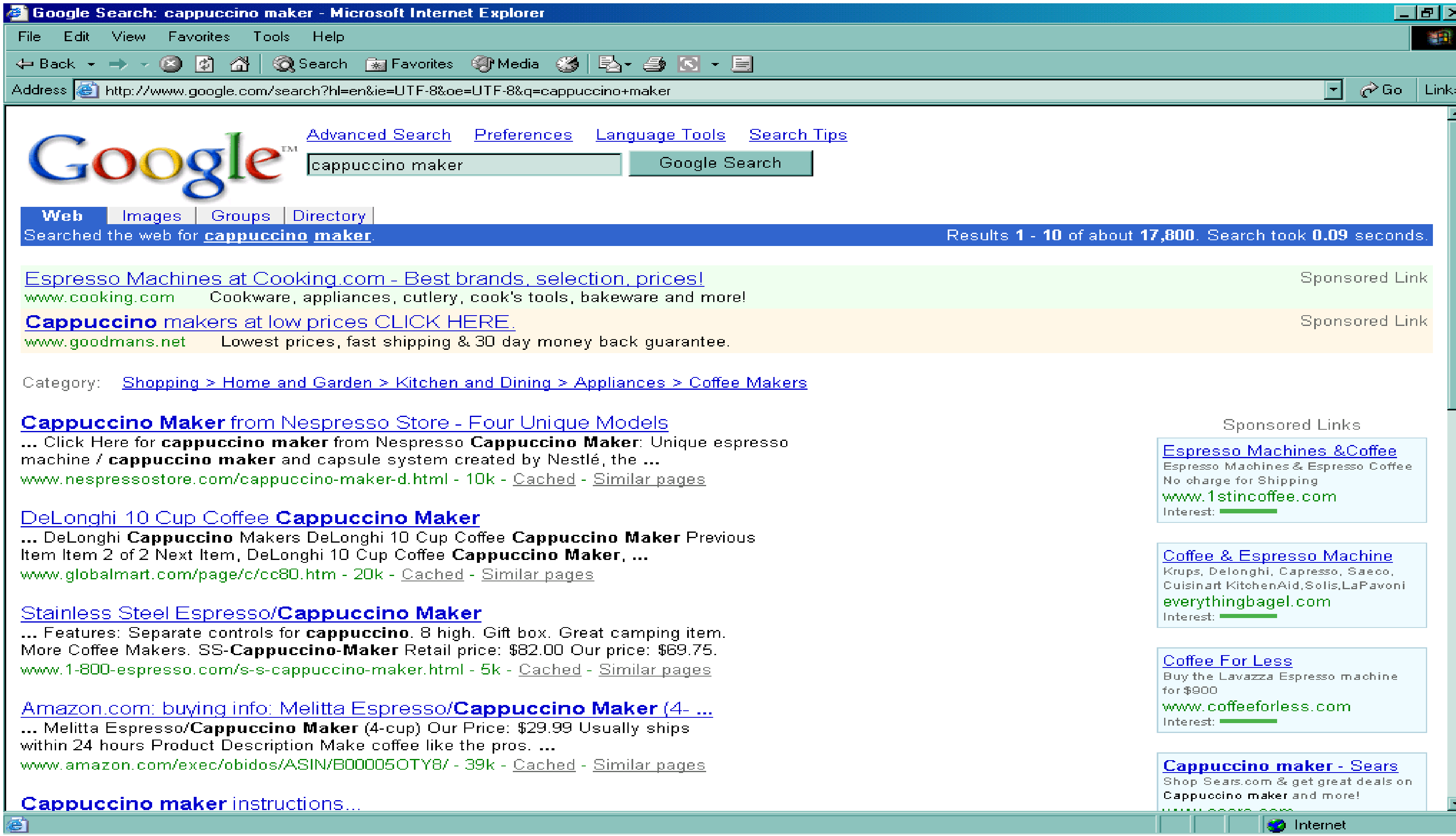
## Lectures 21-22: Information Retrieval

*Chin-Hui Lee*
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332, USA
chl@ece.gatech.edu

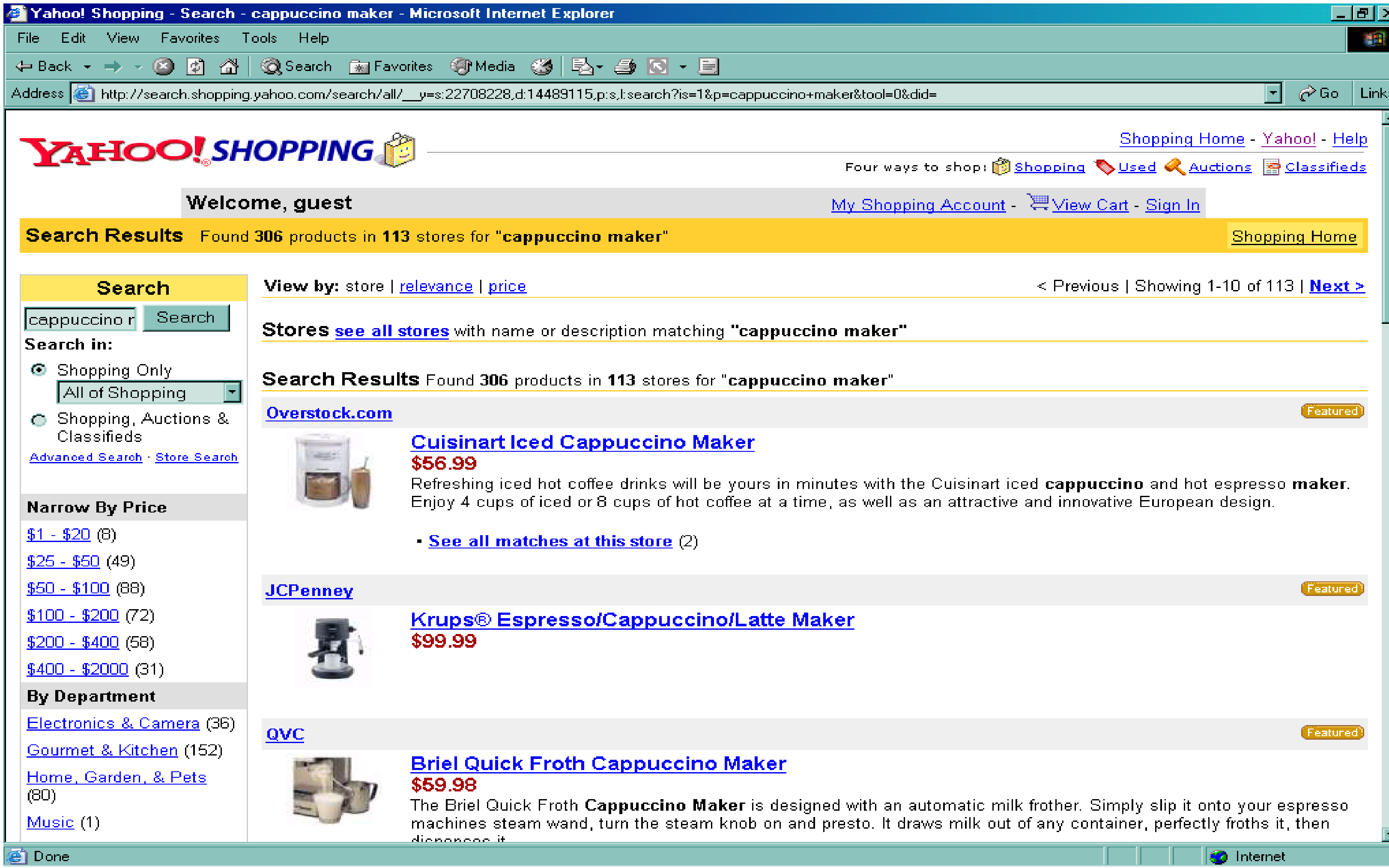
# A Text Categorization Scenario

- Suppose you want to buy a cappuccino maker as a gift on the web
  - try Google for “cappuccino maker”
  - try “Yahoo! Shopping” for “cappuccino maker”

# Google Search Results



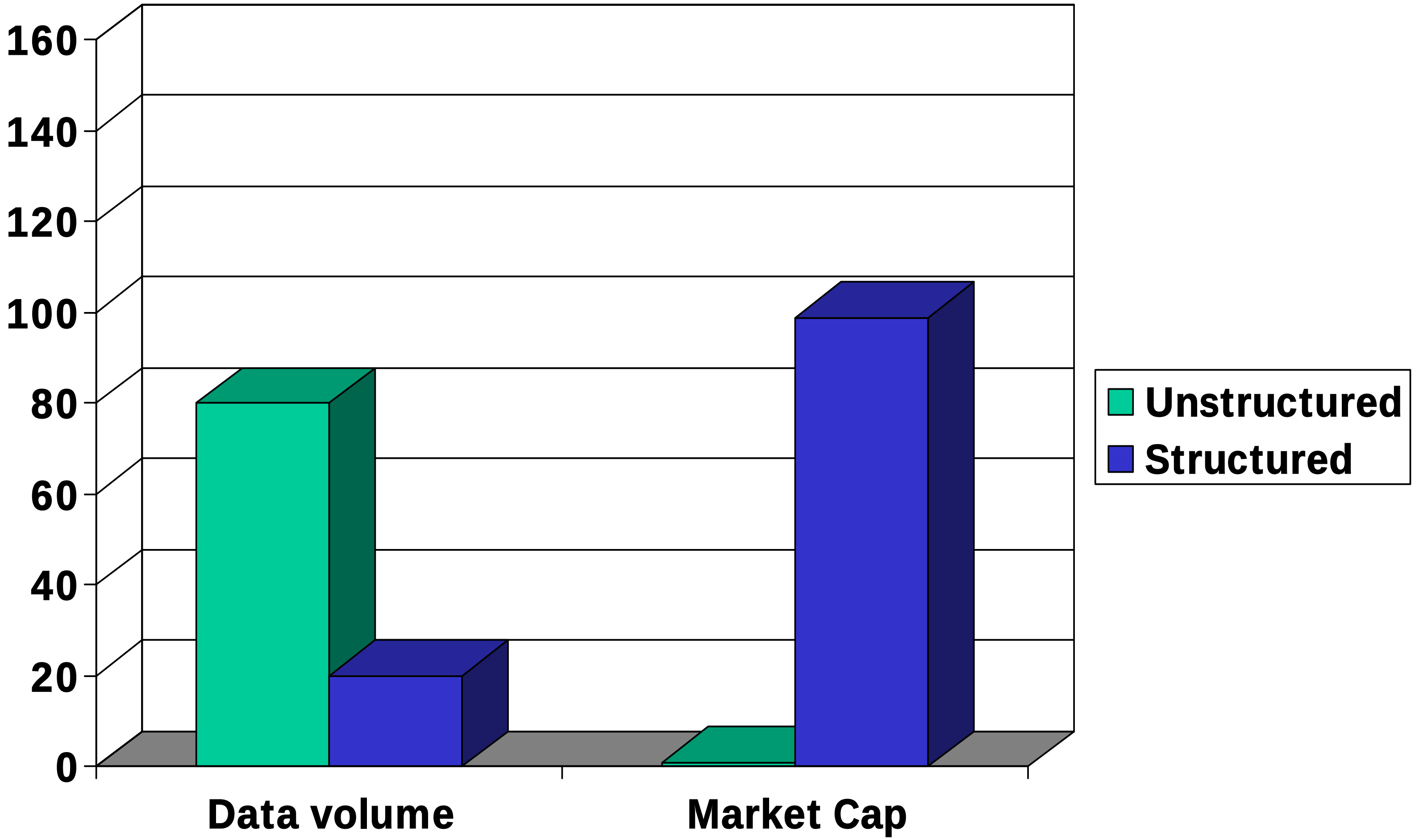
# Yahoo Search Results



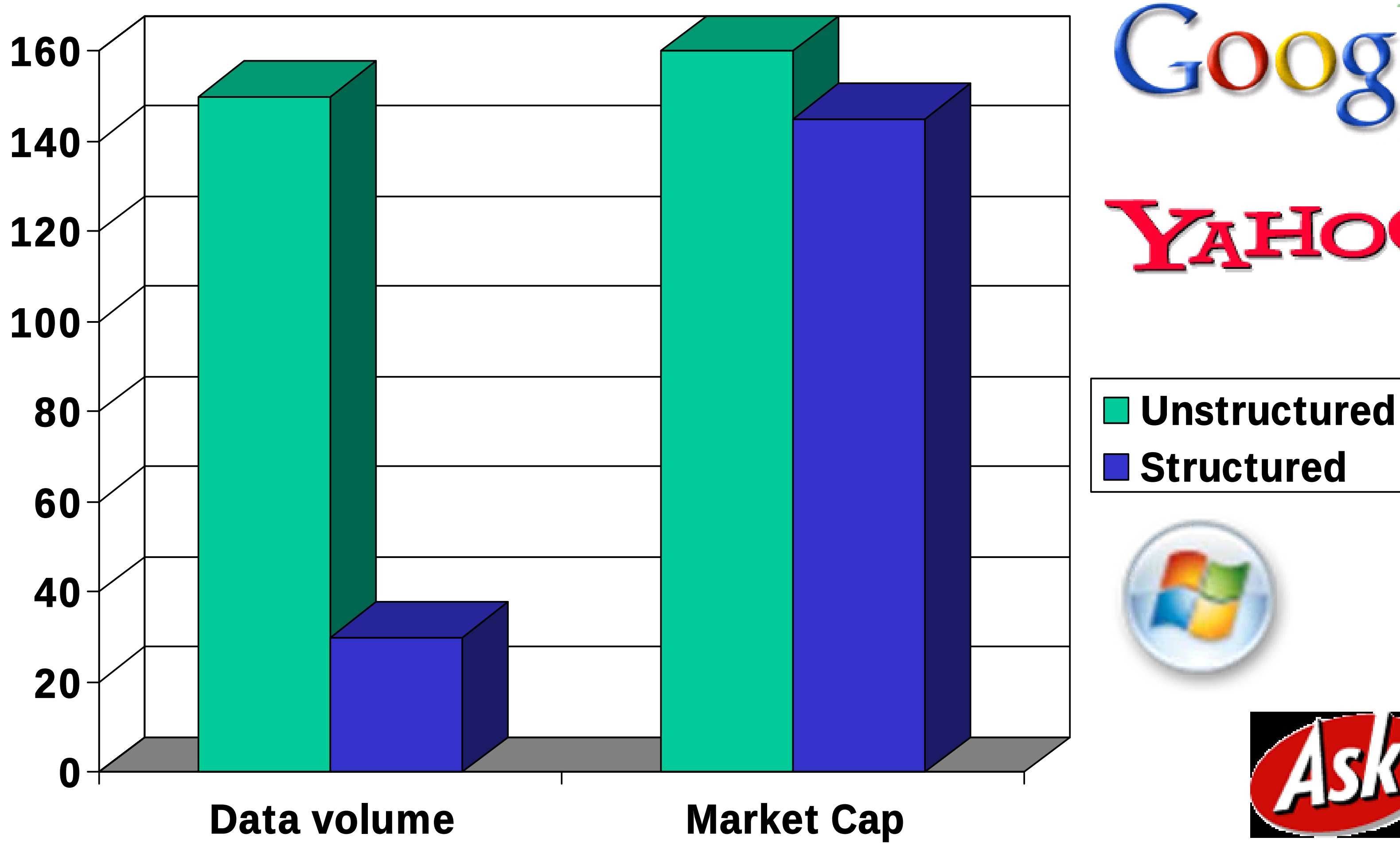
# Observations

- Broad indexing & speedy search alone are not enough
- Organizational view of data is critical for effective retrieval
- Categorized data are easy for user to browse
- Category taxonomies become most central in well-known web sites (Yahoo!, Lycos, ...)

# Unstructured vs. Structured Data (1996)



# Unstructured vs. Structured Data (2006)



# Background on IR

- Suggested reading:
  - M. Berry and M Browne, *Understanding Search Engines: Mathematical Modeling and Text Retrieval*, Chapter 3, SIAM, 1999.
- Retrieve textual information from document repositories
  - User enters a query describing the desired information
  - The system returns a list of documents – exact match, ranked list

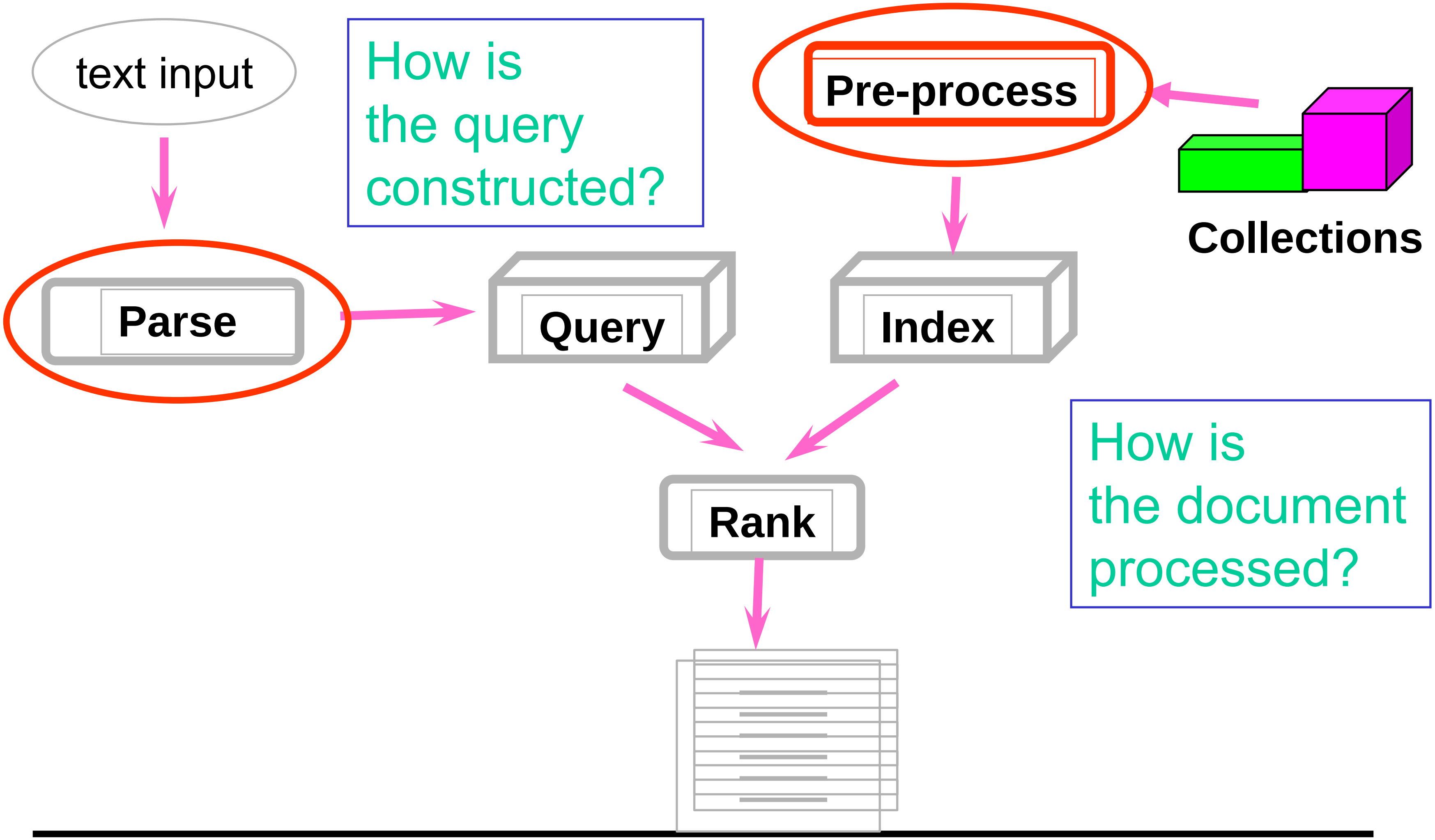
# Text Categorization

- Attempt to assign documents to two or more pre-defined categories
  - **Routing**: Ranking of documents according to relevance. Training information in the form of relevance labels is available
  - **Filtering**: Absolute assessment of relevance

# Discourse Segmentation

- Break documents into topically coherent multi-paragraph subparts
  - Detect topic shifts within document
  - Search for vocabulary shifts in subtopics
- TextTiling (Hearst and Plaunt, 1993)
  - Divide text into fixed size blocks (20 words)
  - Look for topic shifts in-between these blocks
    - Cohesion scorer: measures the topic continuity at each gap (point between two block)
    - Depth scorer: at a gap determine how low the cohesion score is compared to surrounding gaps
    - Boundary selector: looks at the depth scores & selects the gaps that are the best segmentation points

# Information Retrieval Process





*Center of Signal and Image Processing*
*Georgia Institute of Technology*

# Example: Information Needs

- Sometimes very specific
  - <title> Falkland petroleum exploration
  - <desc> Description: What information is available on petroleum exploration in the South Atlantic near the Falkland Islands?
  - <narr> Narrative: Any document discussing petroleum exploration in the South Atlantic near the Falkland Islands is considered relevant. Documents discussing petroleum exploration in continental South America are not relevant
- Sometimes very vague
  - I am going to Kyoto, Japan for a conference in two months. What should I know?

# Relevance

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

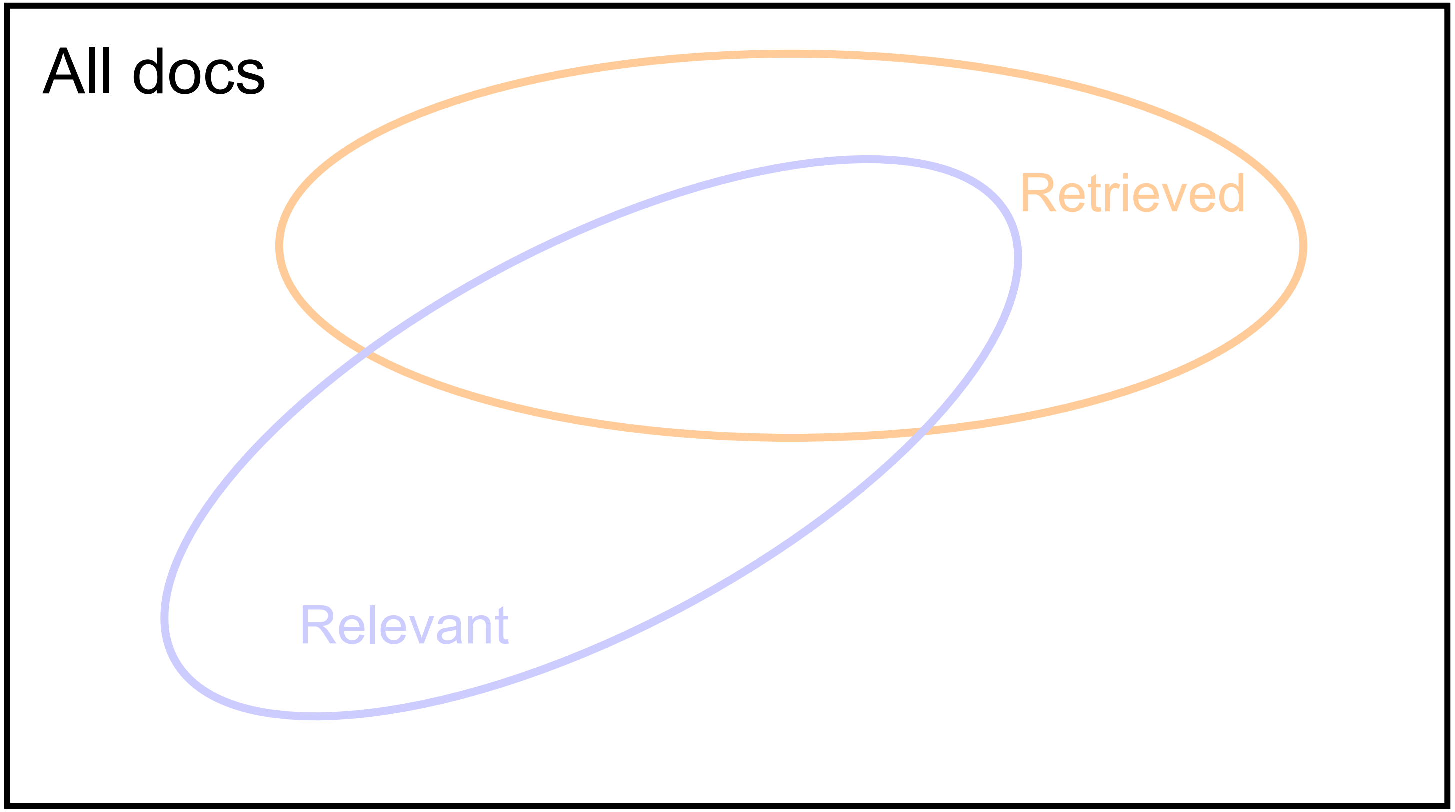
# Qualifying Relevance

- How relevant is the document
  - for this user for this information need
- Subjective, but measurable to some extent
  - How often do people agree that a document is relevant to a query
- How well does it answer the question?
  - Complete answer?  Partial?
  - Background Information?
  - Hints for further exploration?

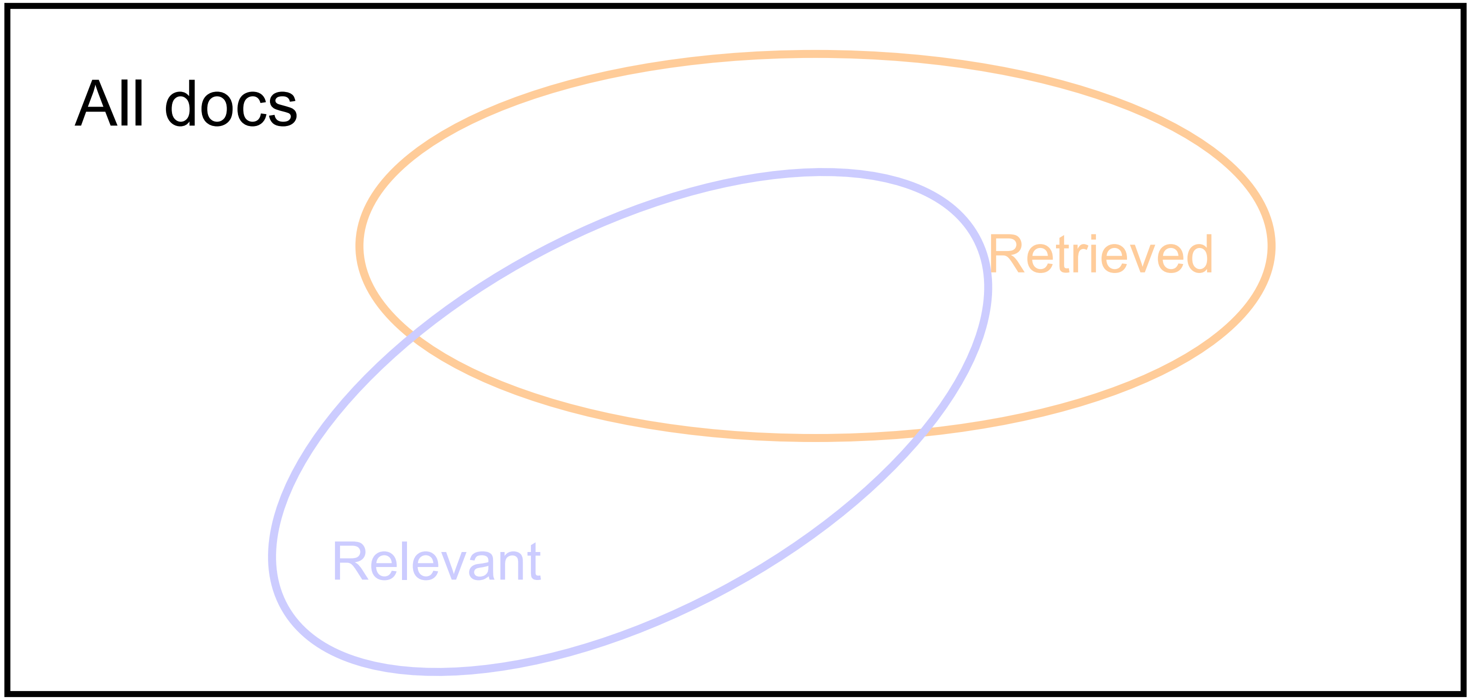
# Evaluation Measures

- Precision: Percentage of relevant items returned
- Recall: Percentage of all relevant documents in the collection that is in the returned set
- Combine precision and recall:
  - Cutoff
  - Un-interpolated average precision
  - Interpolated average precision
  - Precision-recall curves
  - F measure
  - Categorization accuracy and error

# Quality Metrics: Relevant vs. Retrieved



# Precision vs. Recall

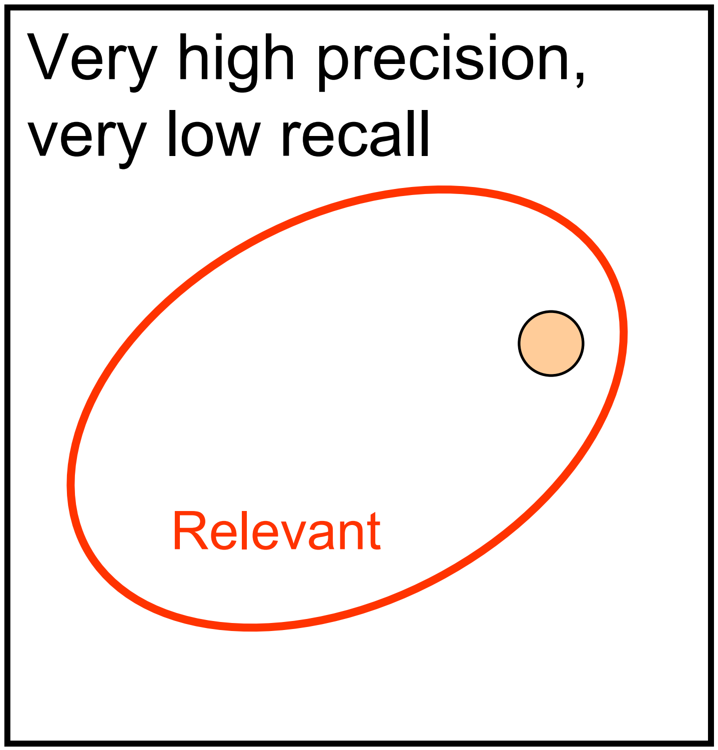


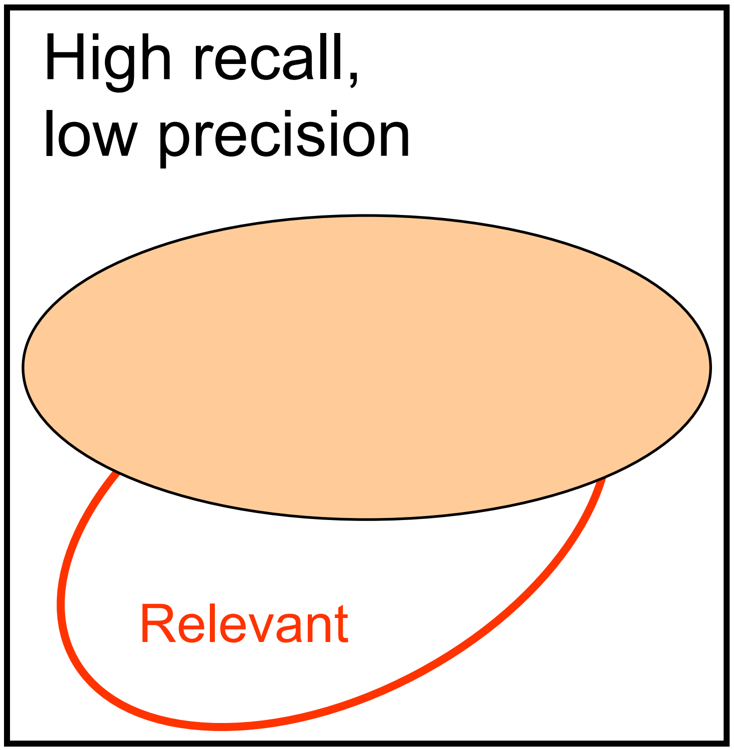

$$\text{Recall} = \frac{|\text{RelRetrieved}|}{|\text{Rel in Collection}|}$$

$$\text{Precision} = \frac{|\text{RelRetrieved}|}{|\text{Retrieved}|}$$

# Why Precision and Recall?

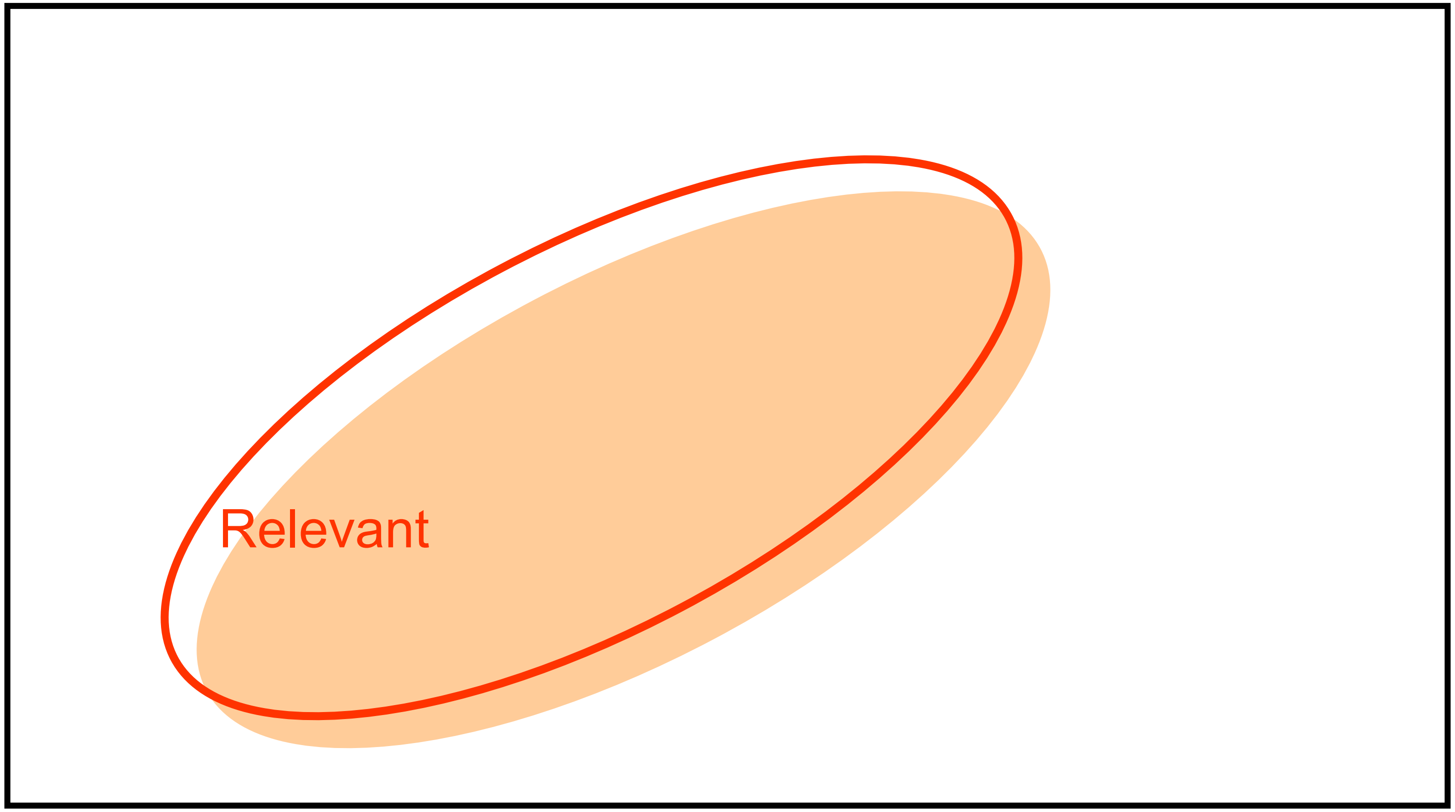
Get as much good stuff while at the same time getting as little junk as possible



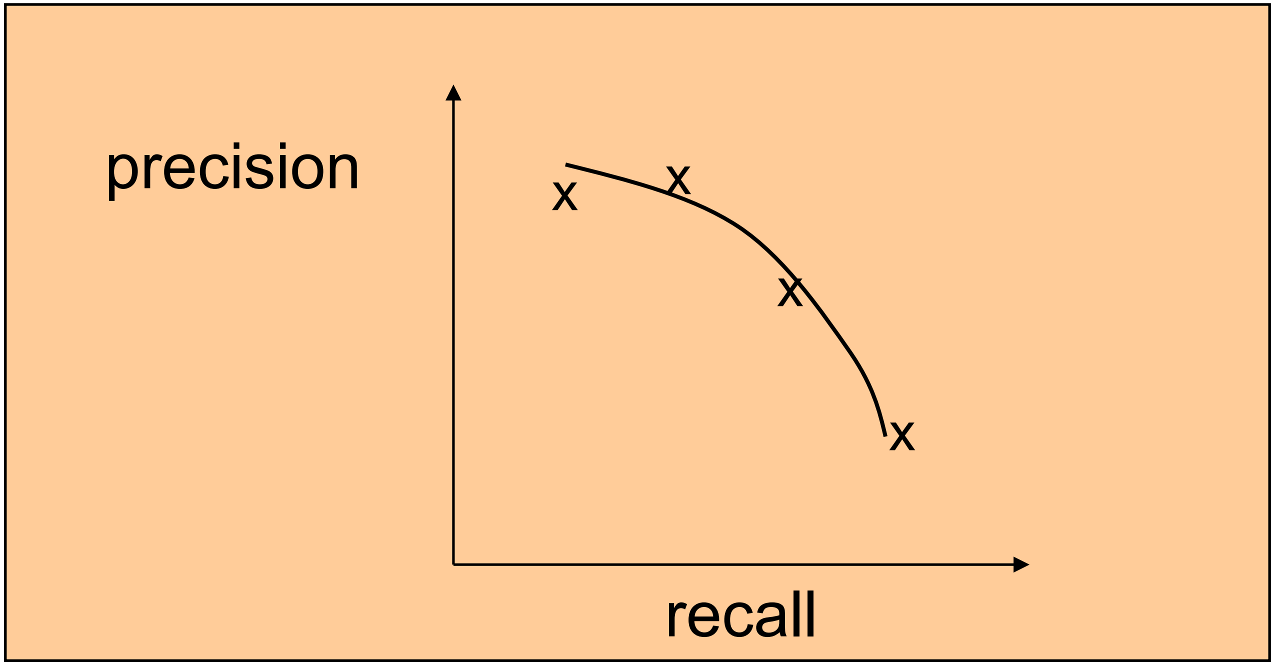
# Retrieved vs. Relevant Documents

High precision, high recall (at last!)



# Precision/Recall Curves

- There is a tradeoff between Precision and Recall
- So measure Precision at different levels of Recall
- Note: this is an AVERAGE over MANY queries



# Average Precision

- IR systems typically output a ranked list of documents
- For each relevant document, compute the precision up to that point
- Average over all precision values computed this way

# Interpolated Average Precision

- Precision may go up when going down the ranked list
- Intuitively, this should only go down
- Interpolated Average Precision
  - for each recall level in 0%, 10%, 20%, …
    - compute the highest precision after recall reached that point
  - take the average of the max precision scores

# 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>1: precision is more important
  - α<1: recall is more important
  - Usually α=1 ($F_1$)

# Probability Ranking Principle (PRP)

- Ranking documents in order of decreasing probability of relevance
  - View retrieval as a greedy search that aims to identify the most valuable document
- Assumptions of PRP:
  - Documents are independent
  - Complex information need is broken into a number of queries which are each optimized in isolation
  - Probability of relevance is only estimated

# Document Representation

- Information needs and documents are usually represented as sets/bags of terms
  - Bag: allow multiple instances of the same element
- Terms: words, phrases
  - To stem or not to stem
  - Annotation with location information: title, heading

# Types of Queries

- Boolean Query
  - Does the document satisfy the Boolean expression?
    - “java” AND “compilers” AND (“unix” OR “linux”)
- Vector Query
  - How similar is the document to the query?
    - [(java 3) (compiler 2) (unix 1) (linus 1)]
- Probabilistic Query
  - What is the probability that the document is generated by the query?

# Boolean Model of Retrieval

- Pros
  - Easy to understand/clear semantics
    - AND means ‘all’, OR means ‘any’
  - Usually computationally efficient
- Cons
  - Difficult to rank results
  - Rigid: either get too much or too little
    - AND means ‘all’, OR means ‘any’
    - When the information need is complex, it is hard to formulate it as a Boolean query

# The Vector Space Model

- Measure closeness between query and document
  - Queries and documents represented as *n*-dim vectors
  - Each dimension corresponds to a word.
  - Advantages: Conceptual simplicity and use of spatial proximity for semantic proximity

$$
\begin{array}{c}
\begin{array}{ccccc} & T_1 & T_2 & \cdots & T_t \end{array} \\
\left(\begin{array}{ccccc}
D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\
D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\
\vdots & \vdots & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & \vdots \\
D_n & w_{1n} & w_{2n} & \cdots & w_{tn}
\end{array}\right)
\end{array}
$$

# Vector Similarity

- *d* = The man said that a space age man appeared *d'* = Those men appeared to say their age

| | $\vec{d}$ | $\vec{d}'$ |
|---|---|---|
| age | 1 | 1 |
| appeared | 1 | 1 |
| man | 2 | 0 |
| men | 0 | 1 |
| said | 1 | 0 |
| say | 0 | 1 |
| space | 1 | 0 |

# Cosine Vector Similarity

- cosine measure or normalized correlation coefficient

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}||\vec{y}|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

Euclidean Distance:

$$|\vec{x} - \vec{y}| = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

# Term Weights

- The weight $w_{ij}$ reflects the importance of the term $T_i$ in document $D_j$
- Intuitions:
  - A term that appears in many documents is not important: e.g., *the*, *going*, *come*, …
  - If a term is frequent in a document, it is probably important in that document


*Center of Signal and Image Processing*
*Georgia Institute of Technology*

# Assigning Weights to Terms

- Binary weights
- Raw term frequency
- tf x idf (TDIDF)
  - Recall the Zipf law
  - Want to weight terms highly if they are
    - frequent in relevant documents … BUT
    - infrequent in the collection as a whole
- Pointwise mutual information
- Term distribution models

# Binary Weights

- Only the presence (1) or absence (0) of a term is included in the vector

| *docs* | *t1* | *t2* | *t3* |
|---|---|---|---|
| D1 | 1 | 0 | 1 |
| D2 | 1 | 0 | 0 |
| D3 | 0 | 1 | 1 |
| D4 | 1 | 0 | 0 |
| D5 | 1 | 1 | 1 |
| D6 | 1 | 1 | 0 |
| D7 | 0 | 1 | 0 |
| D8 | 0 | 1 | 0 |
| D9 | 0 | 0 | 1 |
| D10 | 0 | 1 | 1 |
| D11 | 1 | 0 | 1 |

# Raw Term Weights

- The frequency of occurrence for the term in each document is included in the vector

| *docs* | *t1* | *t2* | *t3* |
|---|---|---|---|
| D1 | 2 | 0 | 3 |
| D2 | 1 | 0 | 0 |
| D3 | 0 | 4 | 7 |
| D4 | 3 | 0 | 0 |
| D5 | 1 | 6 | 3 |
| D6 | 3 | 5 | 0 |
| D7 | 0 | 8 | 0 |
| D8 | 0 | 10 | 0 |
| D9 | 0 | 0 | 1 |
| D10 | 0 | 3 | 5 |
| D11 | 4 | 0 | 1 |

# Inverse Document Frequency

- IDF provides high values for rare words and low values for common words

For a collection of 10000 documents

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

# Term Weights: tf x idf

- Term frequency (tf)
  - the frequency count of a term in a document
- Inverse document frequency (idf)
  - The amount of information contained in the statement "Document X contains the term $T_i$"
- Assign a tf * idf weight to each term in each document

# TDIDF

$$w_{ik} = tf_{ik} * \log(N / n_k)$$

$T_k$ = term $k$ in document $D_i$

$tf_{ik}$ = frequency of term $\mathrm{T}_k$ in document $D_i$

$idf_k$ = inverse document frequency of term $\mathrm{T}_k$ in $C$

$N$ = total number of documents in the collection $C$

$n_k$ = the number of documents in $C$ that contain $\mathrm{T}_k$

$$idf_k = \log\left(\frac{N}{n_k}\right)$$

# Pointwise Mutual Information

- Pointwise Mutual Information measures the strength of association between two elements (a document and a term)
  – Observed frequency vs. expected frequency

$$w_{ij} = MI(T_i, D_j) = \log\left(\frac{P(T_i, D_j)}{P(T_i) \times P(D_j)}\right)$$

- MI weight is insensitive to stemming and the use of stop word list [Pantel and Lin, 2002]

# Term Distribution Models

- Develop a model for the distribution of a word and use this model to characterize its importance for retrieval
- Estimate $p_i(k)$
  - $p_i(k)$ : proportion of times that word $w_i$ appears $k$ times in a document
- Poisson, two-Poisson and $K$-mixture
  - We can derive the IDF from term distribution models

# The Poisson Distribution

$$p(k;\lambda_i) = e^{-\lambda_i} \frac{\lambda_i^k}{k!} \text{ for some } \lambda_i > 0$$

- the parameter is the average number of occurrences of $w_i$ per document

$$\lambda_i = \frac{\mathrm{cf_i}}{\mathrm{N}}$$

- We are interested in the frequency of occurrence of a particular word $w_i$ in a document
- Poisson distribution is good for estimating non-content words

# The Two-Poisson Model

- Better fit to the frequency distribution
  - Mixture of two poissons
    - Non-privileged class: Low average # of occurrences
      - Occurrences are accidental
    - Privileged class: High average # of occurrences
      - Central content word

$$p(k;\pi,\lambda_1,\lambda_2) = \pi e^{-\lambda_1}\frac{\lambda_1^k}{k!} + (1-\pi)e^{-\lambda_2}\frac{\lambda_2^k}{k!}$$

$\pi$ : probability of a document being in the privileged class

$1-\pi$ : probability of a document being in the non-privileged class

$\lambda_1$, $\lambda_2$ : average number of occurrence of word $w_i$ in each class

# Latent Semantic Indexing

- Projects queries and documents into a space with “latent” semantic dimensions
- Dimensionality reduction: the latent semantic space that we project into has fewer dimensions than the original space
- Exploits co-occurrence: the fact that two or more terms occur in the same document more often than chance
- Similarity metric: Co-occurring terms are projected onto the same dimensions

# LSA Mathematical Framework

- LSA Matrix (also known as Routing Matrix) *C*

  $$c_{ij} = (1-\varepsilon_i) n_{ij} / n_{\cdot j} \text{ (scaling and normalization)}$$

  – number of times word $w_i$ occurs in $A_j$ : $n_{ij}$
  – total number of words present in $A_j$ : $n_{\cdot j}$ (column sum)
  – total number of $w_i$ occurs in *A* : $n_{i\cdot}$ (row sum)
  – “indexing” power of $w_i$ in corpus *A* : $\eta_i = 1 - \varepsilon_i$
  – normalized entropy:

$$\varepsilon_i = -\frac{1}{\log N} \sum_{j=1}^{N} \frac{n_{ij}}{n_{i\cdot}} \log \frac{n_{ij}}{n_{i\cdot}} \quad 0 \le \varepsilon_i \le 1$$

$$\begin{cases} \varepsilon_i = 0 \quad \text{if } n_{ij} = n_{i\cdot} \quad \text{maximum indexing power} \\ \varepsilon_i = 1 \quad \text{if } n_{ij} = \frac{n_{i\cdot}}{N} \quad \text{no power (equally probable)} \end{cases}$$

# Singular Value Decomposition

- SVD takes a document-by-term matrix A in n-dim space and projects it to $\hat{A}$ in a lower dimensional space k (n>>k). The 2-norm (distance) between the two matrices is minimized:

$$\Delta = \left\| A - \hat{A} \right\|_2$$

# SVD (Cont.)

- SVD projection: $A_{t\times d} = T_{t\times n} S_{n\times n} (D_{d\times n})^T$
  - $A_{txd}$ – document-by-term matrix
  - $T_{txn}$ – Terms in new space
  - $S_{nxn}$ – Singular values of $A$ in descending order
  - $D_{dxn}$ – document matrix in new space
  - $N$ = min $(t,d)$
  - $T$, $D$ have orthonormal columns
- LSI in IR
  - Encode terms and documents using factors derived from SVD
  - Rank similarity of terms and docs to query via Euclidean distances or cosines

# Bigger Corpora: Real-World Problems

- Consider $N$ = 1M documents, each with about 1K terms
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents
- Say there are $m$ = 500K *distinct* terms among these

# Design Features of IR Systems

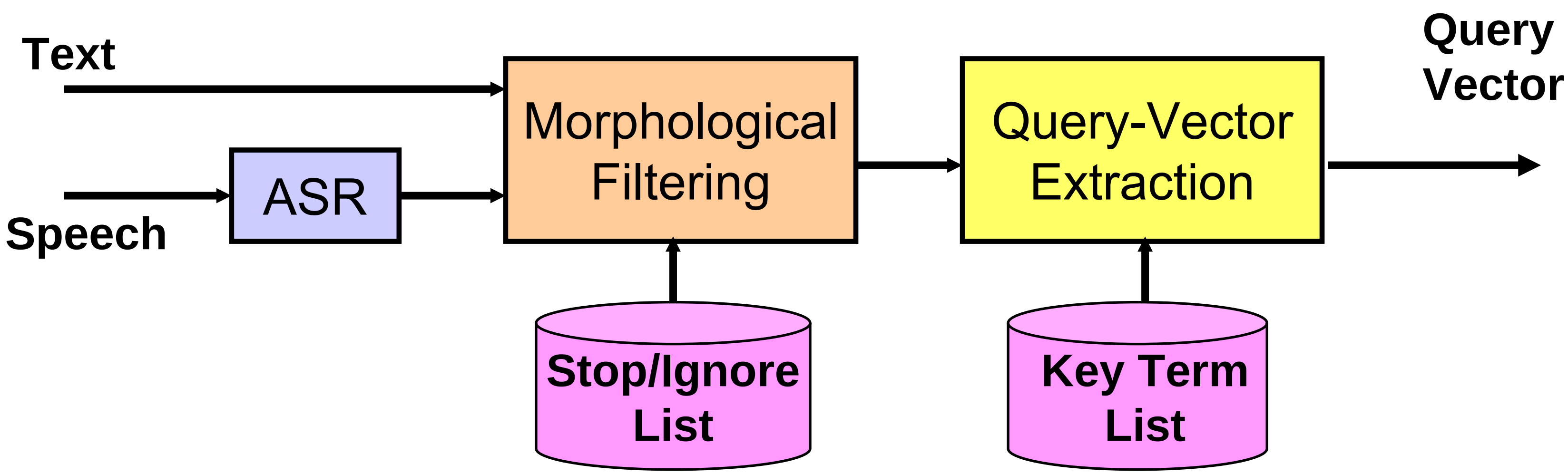
- **Inverted Index**:
  - Primary data structure of IR systems
  - Data structure that lists each word and its frequency in all documents
  - Including the position information allows us to search for phrases
- **Stop List (Function Words)**:
  - Lists words unlikely to be useful for searching
  - Examples: *the, from, to* ….
  - Excluding this reduces the size of the inverted index

# Design Features (Cont.)

- **Stemming**:
  - Simplified form of morphological analysis consisting simply of truncating a word
  - For example *laughing, laughs, laugh* and *laughed* are all stemmed to *laugh*
  - The problem is semantically different words like *gallery* and *gall* may both be truncated to *gall* making the stems unintelligible to users
  - Levins and Porter Stemmer
- **Thesaurus**:
  - Widen search to include documents using related terms

# Query Vector Extraction

- Text Pre-processing (SMART, Salton, 1971)
  - Extract root form of a word
  - Delete ignore words, e.g. *um, uh*
  - Remove stop words, e.g. *I would like to*



# Sparse Term-Document 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

Why?

# Inverted Files for Multiple Documents

## LEXICON

| WORD | NDOCS | PTR |
|---|---|---|
| jezebel | 20 | |
| jezer | 3 | |
| jezerit | 1 | |
| jeziah | 1 | |
| jeziel | 1 | |
| jezliah | 1 | |
| jezoar | 1 | |
| jezrahliah | 1 | |
| jezreel | 39 | |

“jezebel” occurs
6 times in document 34,
3 times in document 44,
4 times in document 56 . . .

| DOCID | OCCUR | POS 1 | POS 2 | . . . | | | |
|---|---|---|---|---|---|---|---|
| 34 | 6 | 1 | 118 | 2087 | 3922 | 3981 | 5002 |
| 44 | 3 | 215 | 2291 | 3010 | | | |
| 56 | 4 | 5 | 22 | 134 | 992 | | |

…

| | | | | |
|---|---|---|---|---|
| 566 | 3 | 203 | 245 | 287 |

| | | |
|---|---|---|
| 67 | 1 | 132 |

. . .

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 107 | 4 | 322 | 354 | 381 | 405 | | |
| 232 | 6 | 15 | 195 | 248 | 1897 | 1951 | 2192 |
| 677 | 1 | 481 | | | | | |
| 713 | 3 | 42 | 312 | 802 | | | |

## OCCURENCE INDEX

# Inverted Index

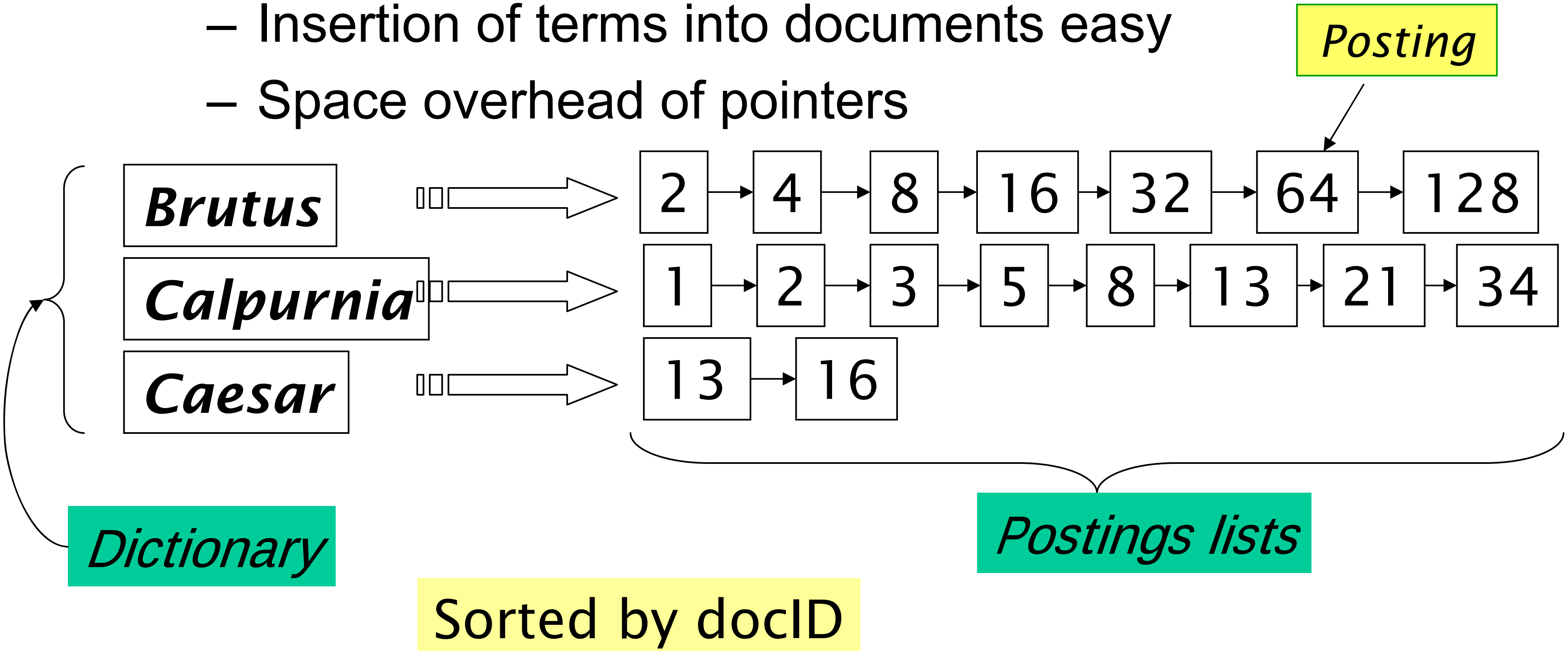
- For each term *T*, we must store a list of all documents that contain *T*
- Do we use an array or a list for this?

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

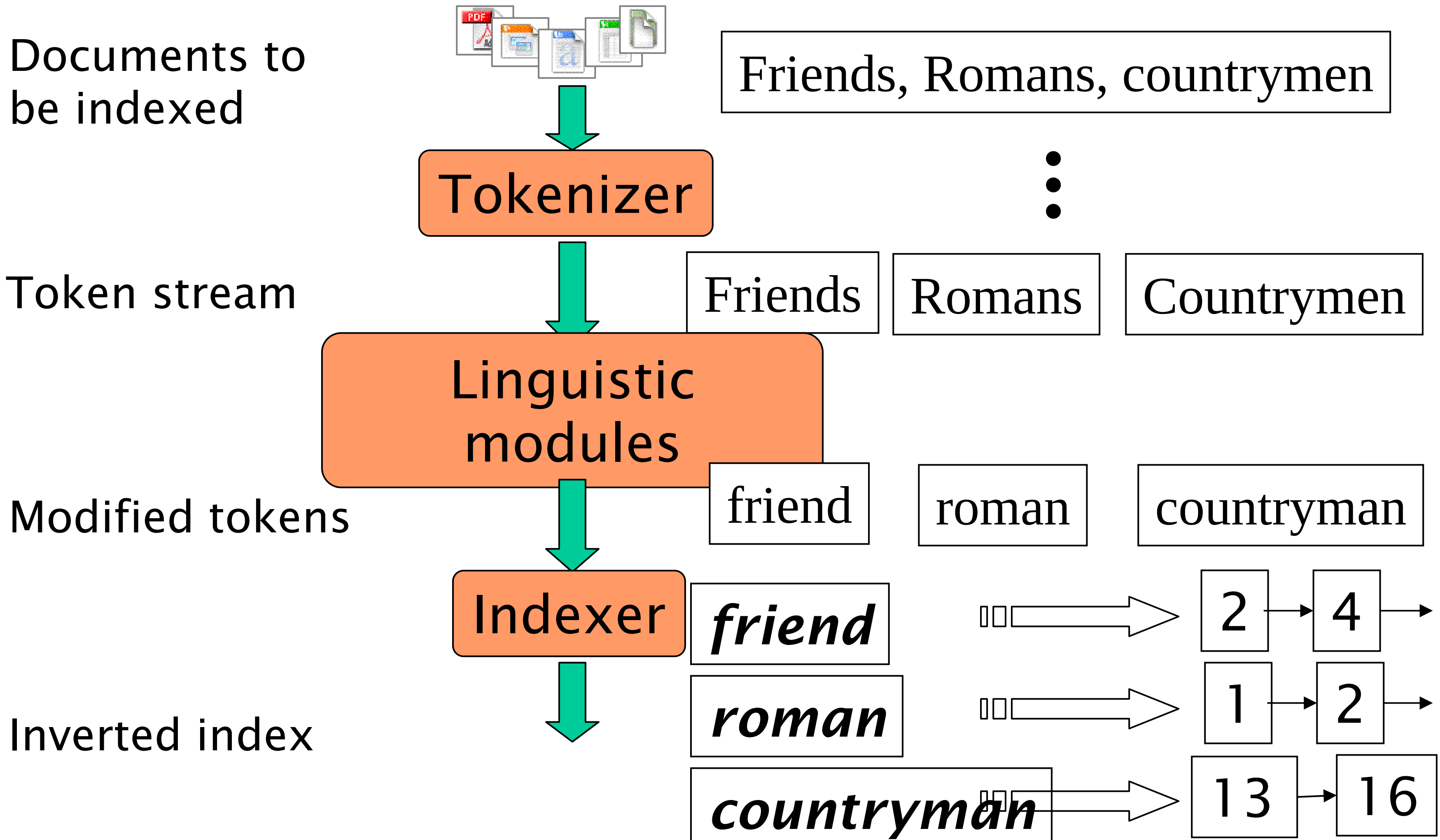
- What happens if the word ***Caesar*** is added to document 14?

# Inverted Index

- Linked lists generally preferred to arrays
  - Dynamic space allocation
  - Insertion of terms into documents easy
  - Space overhead of pointers



# Inverted Index Construction



# Indexer Steps

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 | 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 |

# Term Sorting

Sort by terms.

**Core indexing step.**

| 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 |

| Term | Doc # |
|---|---|
| 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 Merging

- Multiple term entries in a single document are merged
- Frequency information is added

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

# Dictionary & Postings Split

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

| Term | N docs | Coll freq |
|---|---|---|
| ambitious | 1 | 1 |
| be | 1 | 1 |
| brutus | 2 | 2 |
| capitol | 1 | 1 |
| caesar | 2 | 3 |
| did | 1 | 1 |
| enact | 1 | 1 |
| hath | 1 | 1 |
| I | 1 | 2 |
| i' | 1 | 1 |
| it | 1 | 1 |
| julius | 1 | 1 |
| killed | 1 | 2 |
| let | 1 | 1 |
| me | 1 | 1 |
| noble | 1 | 1 |
| so | 1 | 1 |
| the | 2 | 2 |
| told | 1 | 1 |
| you | 1 | 1 |
| was | 2 | 2 |
| with | 1 | 1 |

| Doc # | Freq |
|---|---|
| 2 | 1 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 1 | 1 |
| 1 | 1 |
| 2 | 1 |
| 1 | 2 |
| 1 | 1 |
| 2 | 1 |
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 2 | 1 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 2 | 1 |

# Where Do We Pay in Storage?

Terms

| Term | N docs | Coll freq |
|---|---|---|
| ambitious | 1 | 1 |
| be | 1 | 1 |
| brutus | 2 | 2 |
| capitol | 1 | 1 |
| caesar | 2 | 3 |
| did | 1 | 1 |
| enact | 1 | 1 |
| hath | 1 | 1 |
| I | 1 | 2 |
| i' | 1 | 1 |
| it | 1 | 1 |
| julius | 1 | 1 |
| killed | 1 | 2 |
| let | 1 | 1 |
| me | 1 | 1 |
| noble | 1 | 1 |
| so | 1 | 1 |
| the | 2 | 2 |
| told | 1 | 1 |
| you | 1 | 1 |
| was | 2 | 2 |
| with | 1 | 1 |
| | | |

Pointers

| Doc # | Freq |
|---|---|
| 2 | 1 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 1 | 1 |
| 1 | 1 |
| 2 | 2 |
| 1 | 1 |
| 1 | 1 |
| 2 | 1 |
| 1 | 2 |
| 1 | 1 |
| 2 | 1 |
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 2 | 1 |
| 2 | 1 |
| 1 | 1 |
| 2 | 1 |
| 2 | 1 |
| | |
| | |

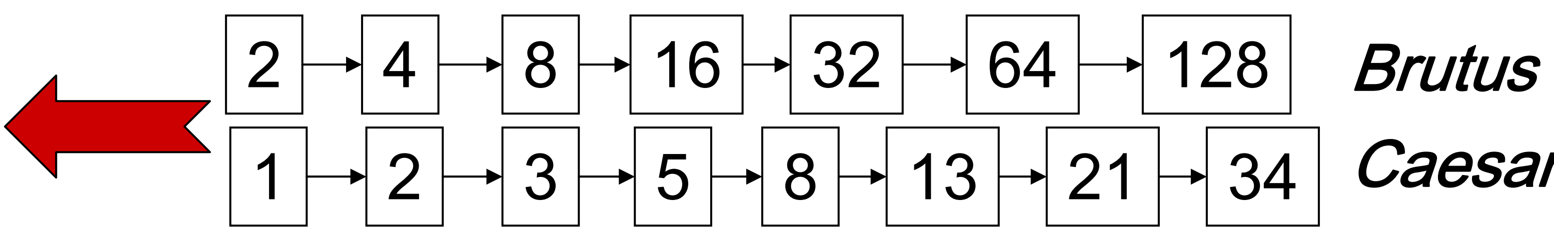
# The Index We Just Built

- How do we process a query?

- Later - what kinds of queries can we process?

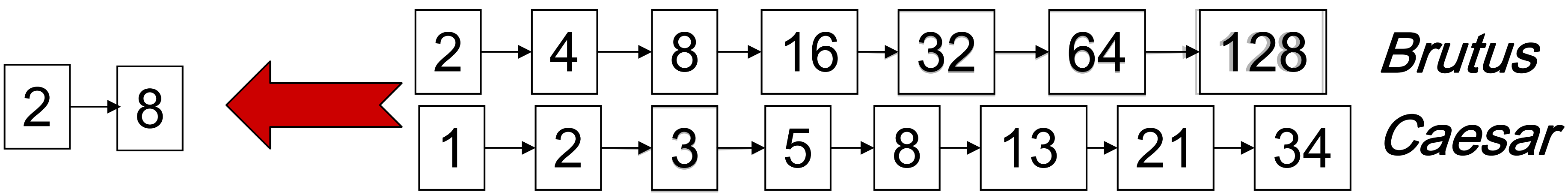
# Query Processing: AND

- 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:



# 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.
<u>Crucial</u>: postings sorted by docID

# 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 <u>set</u> of words
    - Is precise: document matches condition or not
- Primary commercial retrieval tool for 3 decades
- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you're getting

# Example: WestLaw http://www.westlaw.com/

- Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)
- 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
- Professional searchers often like Boolean search:
  - Precision, transparency and control
- But that doesn't mean they actually work 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($x$+$y$) or 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 *t* terms.
- For each of the *t* terms, get its postings, then *AND* them together.

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

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 freq in dictionary

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

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

# Query Processing Exercises

- 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.

# Evidence Accumulation

- 1 vs. 0 occurrence of a search term
  - 2 vs. 1 occurrence
  - 3 vs. 2 occurrences, etc.
  - Usually more seems better
- Need term frequency information in docs

# Ranking Search Results

- Boolean queries give inclusion or exclusion of docs
- Often we want to rank/group results
  - Need to measure proximity from query to each doc
  - Need to decide whether docs presented to user are singletons, or a group of docs covering various aspects of the query

# IR vs. Structured Databases

- Structured data tends to refer to information in "tables"

| Employee | Manager | Salary |
|---|---|---|
| Smith | Jones | 50000 |
| Chang | Smith | 60000 |
| Ivy | Smith | 50000 |

- Typically allows numerical range and exact match (for text) queries, e.g.,
- *Salary < 60000 AND Manager = Smith*.

# Unstructured Data

- Typically refers to free text
- Allows
  - Keyword queries including operators
  - More sophisticated “concept” queries e.g.,
    - find all web pages dealing with *drug abuse*
- Classic model for searching text documents

# Semi-Structured Data

- In fact almost no data is “unstructured”
- E.g., this slide has distinctly identified zones such as the *Title* and *Bullets*
- Facilitates “semi-structured” search such as
  - *Title* contains <u>data</u> AND *Bullets* contain <u>search</u>

    - … to say nothing of linguistic structure

# More Sophisticated Semi-Structured Search

- *Title* is about <u>Object Oriented Programming</u> AND *Author* something like <u>stro*rup</u>
- where * is the wild-card operator
- Issues:
  – how do you process “about”?
  – how do you rank results?
- The focus of XML search

# Clustering and Classification

- Given a set of docs, group them into clusters based on their contents
- Given a set of topics, plus a new doc *D*, decide which topic(s) *D* belongs to


Center of Signal and Image Processing
Georgia Institute of Technology

# The Web and Its Challenges

- Unusual and diverse documents
- Unusual and diverse users, queries, information needs
- Beyond terms, exploit ideas from social networks
  - link analysis, clickstreams ...

- How do search engines work? And how can we make them better?

# More Sophisticated *Information* Retrieval

- Cross-language information retrieval
- Question answering
- Summarization
- Text mining
- …

# Documents and Terms Revisited

- What's a document?
  - Depends on your target user and application
- What's a term and how do we find them?
  - Tokenizing
  - Stop lists
  - Stemming
  - Multi-word units

# Tokenization

Input: "*Friends, Romans and Countrymen*"

Output: Tokens

***Friends***

***Romans***

***Countrymen***

- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?


Center of Signal and Image Processing
Georgia Institute of Technology

# Tokenization (Cont.)

- Issues in tokenization:
  - *Finland's capital* →
    *Finland? Finlands? Finland's*?
  - *Hewlett-Packard* → *Hewlett* and *Packard* as two tokens?
    - ***State-of-the-art***: break up hyphenated sequence
      - sometimes
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

# Numbers

***3/12/91*** ***Mar. 12, 1991***

***55 B.C.***

***B-52***

***My PGP key is 324a3df234cb23e***

***100.2.86.144***

- Often, don't index as text.
  - But often very useful: think about things like looking up error codes/stacktraces on the web (One answer is using n-grams)

# Tokenization: Language Issues (1)

- ***L'ensemble*** $\rightarrow$ one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***
- German noun compounds
  - Lebensversicherungsgesellschaftsangestellter
  - 'life insurance company employee'

# Tokenization: Language Issues (2)

- Chinese, Korean and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated when alphabets can intermingle

***フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)***

# Tokenization: Language Issues (3)

- Arabic and Hebrew are basically written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
- استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي. ('Algeria achieved its independence in 1962 after 132 years of French occupation.')
- With Unicode, the surface presentation is complex, but the stored form is straightforward

# Normalization

- Need to “normalize” terms in indexed text as well as query terms into the same form
  - We want to match ***U.S.A.*** and ***USA***
- Most commonly define equivalence classes of terms
  - e.g., by deleting periods in a term
- Alternative is to do asymmetric expansion:
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

# Normalization: Other Languages (1)

- Accents: ***résumé*** vs. ***resume***.
- Most important criterion:
  - How are your users like to write their queries for these words?

- Even in languages that standardly have accents, users often may not type them

- German: Tuebingen vs. Tübingen
  - Should be equivalent

# Normalization: Other Languages (2)

- Need to "normalize" indexed text as well as query terms into the same form

  ***7月30日 vs. 7/30***

- Character-level alphabet detection and conversion
  - Tokenization not separable from this.
  - Sometimes ambiguous:

    ***Morgen will ich in MIT*** …

Is this German "mit"?

# Stop Words

- With a stop list, you exclude from dictionary entirely the commonest words. Intuition:
  - They have little semantic content: *the, a, and, to, be*
  - They take a lot of space: ~30% of postings for top 30
- But the trend is away from doing this:
  - Good index compression techniques means the space for including stopwords in a system is very small
  - Good query optimization techniques mean you pay little at query time for including stop words.
  - You need them for:
    - Phrase queries: “King of Denmark”
    - Various song titles, etc.: “Let it be”, “To be or not to be”
    - “Relational” queries: “flights to London” vs. “flights from London”

# Thesauri and Soundex

- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., ***car*** = ***automobile***
    - ***color*** = ***colour***
- Rewrite to form equivalence classes
- Index such equivalences
  - When the document contains ***automobile***, index it under ***car*** as well (usually, also vice-versa)
- Or expand query?
  - When the query contains ***automobile***, look under ***car*** as well

# Soundex

- Traditional class of heuristics to expand a query into phonetic equivalents
  - Language specific – mainly for names
  - e.g., ***chebyshev*** $\rightarrow$ ***tchebycheff***
- Critical for voice-based search applications

# Lemmatization

- Reduce inflectional/variant forms to base form
- For example:
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's*, *cars'* $\rightarrow$ *car*

  *“the boy's cars are different colors”* $\rightarrow$ *“the boy car be different color”*
- Lemmatization implies doing “proper” reduction to dictionary headword form

# Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggest crude affix chopping
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***

***for example compressed and compression are both accepted as equivalent to compress***.

for exampl compress and compress ar both accept as equival to compress

# Summary

- Today's Class
  - Information retrieval
- Next Classes
  - Project presentation on 4/16
  - Probabilistic context free grammar
  - Lab 6 assigned
- Reading Assignments
  - Manning and Schutze, Chapters 14-16