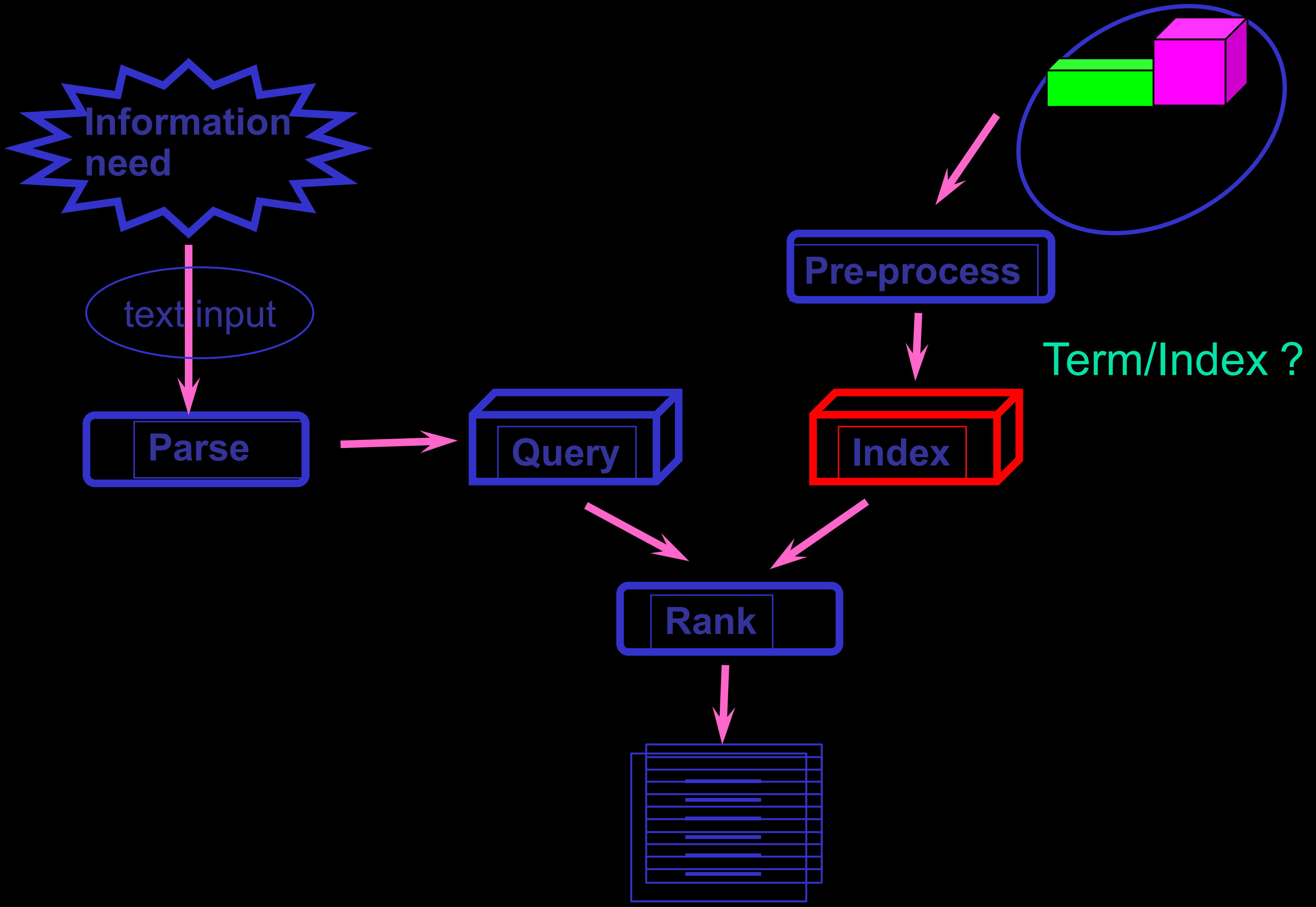
Biomedical Information Retrieval

Lecture 2: Term vocabulary and posting lists

Major subjects for this lecture

- Preprocessing to form the "term vocabulary"
  - Documents
  - Tokenization
  - What *terms* do we put in the index?


Information need
text input
Parse
Query
Pre-process
Index
Term/Index ?
Rank

Content Analysis

- Automated Transformation of raw text into a form that represent some aspect(s) of its meaning
- Including, but not limited to:
  - Automated Thesaurus Generation
  - Phrase Detection
  - Categorization
  - Clustering
  - Summarization

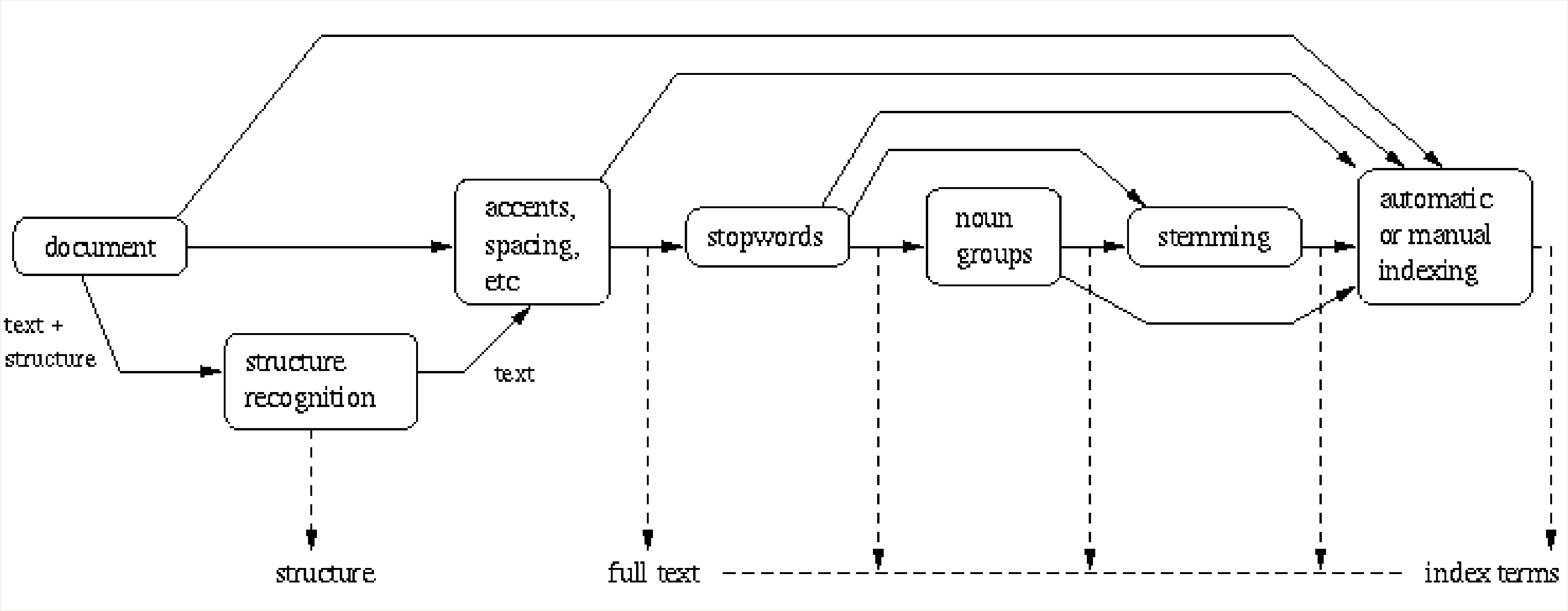
Techniques for Content Analysis

- Statistical
  - Single Document
  - Full Collection
- Linguistic
  - Syntactic
  - Semantic
  - Pragmatic
- Knowledge-Based (Artificial Intelligence)
- Hybrid (Combinations)

Text Processing

- Standard Steps:
  - Recognize document structure
    - titles, sections, paragraphs, etc.
  - Break into tokens
    - usually space and punctuation delineated
    - special issues with Asian languages
  - Stemming/morphological analysis
  - Store in inverted index (to be discussed later)

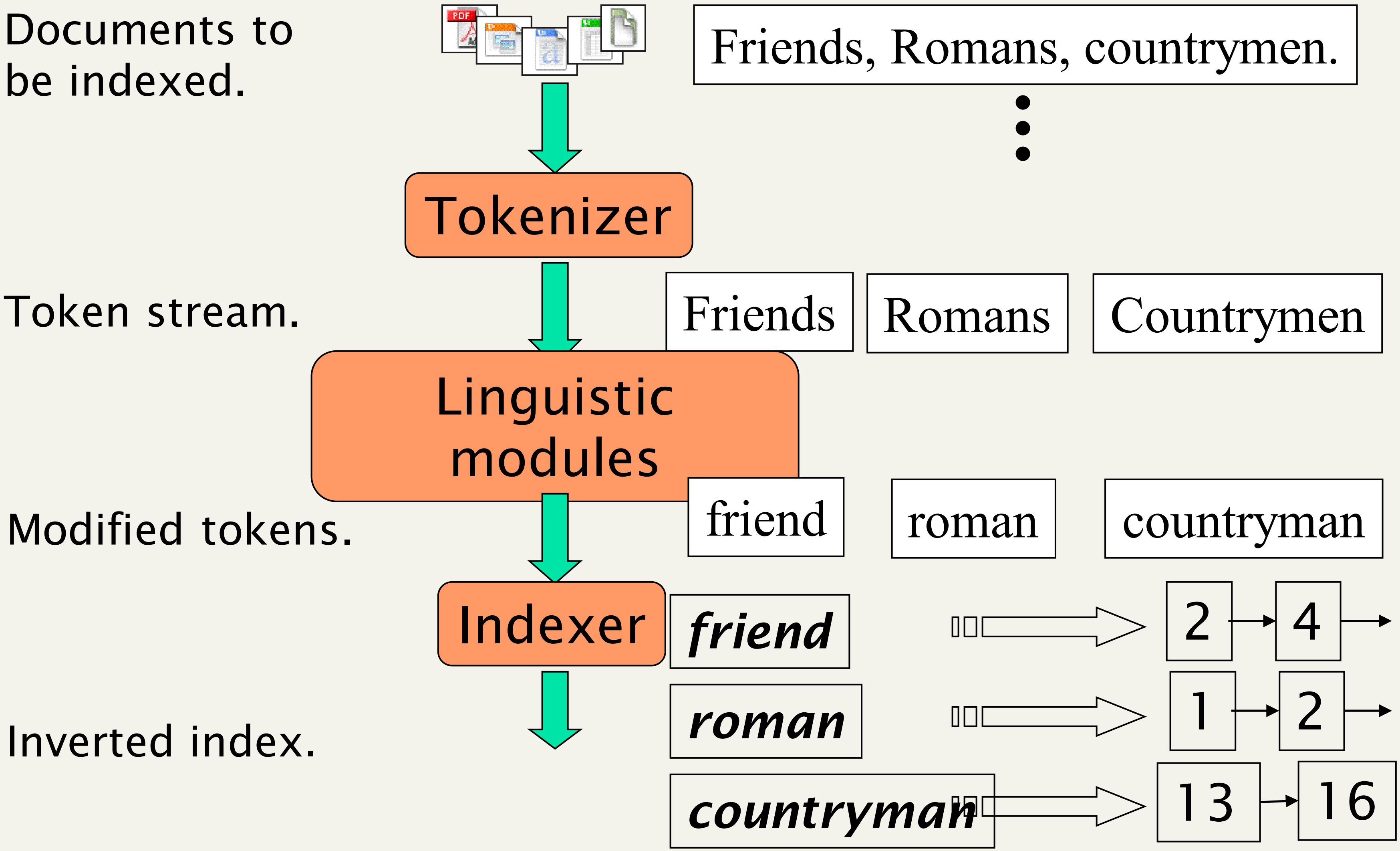
Document Processing Steps



Stemming and Morphological Analysis

- Goal: “normalize” similar words
- Morphology (“form” of words)
  - Inflectional Morphology
    - E.g,. inflect verb endings and noun number
    - Never change grammatical class
      - *dog, dogs*
      - *tengo, tienes, tiene, tenemos, tienen*
  - Derivational Morphology
    - Derive one word from another,
    - Often change grammatical class
      - *build, building; health, healthy*

Recall basic indexing pipeline



Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

Each of these is a classification problem, which we will study later in the course.

But these tasks are often done heuristically …

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***

Complications: Format/language

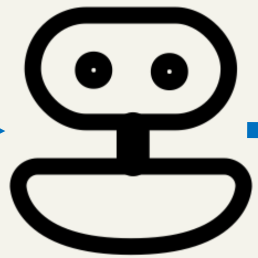
- Documents being indexed can include docs from many different languages
  - A single index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
- What is a unit document?
  - A file?
  - An email? (Perhaps one of many in an mbox.)
  - An email with 5 attachments?
  - A group of files (PPT or LaTeX as HTML pages)

Tokens and Terms

字、詞、字串、符號、代碼...

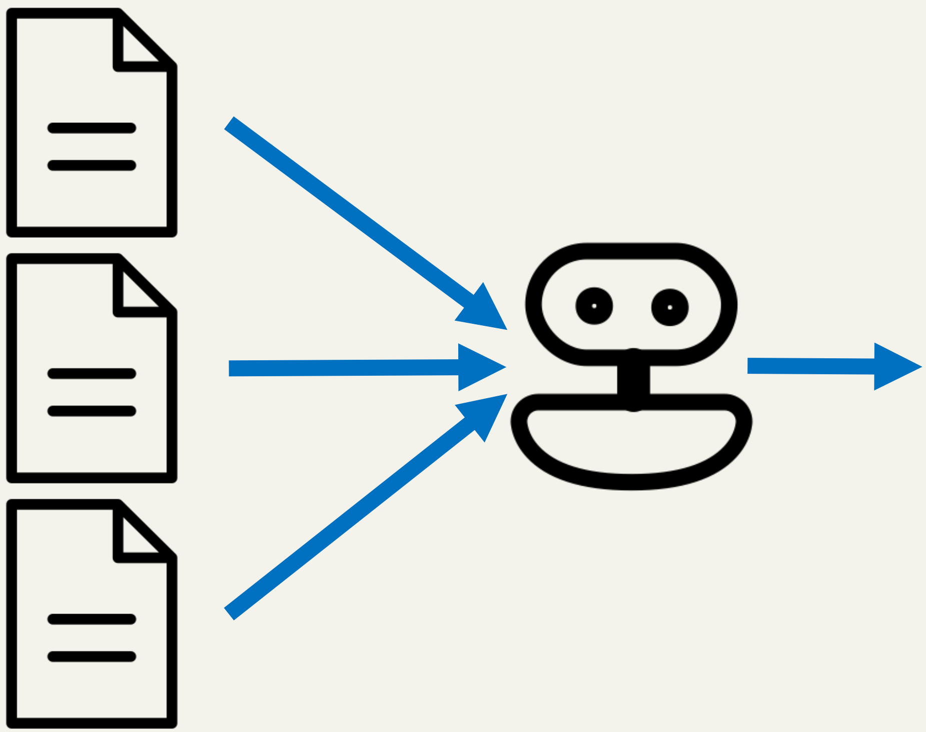
Bag of Words

I love dogs



	I	Love	Dogs
Doc 1	1	1	1

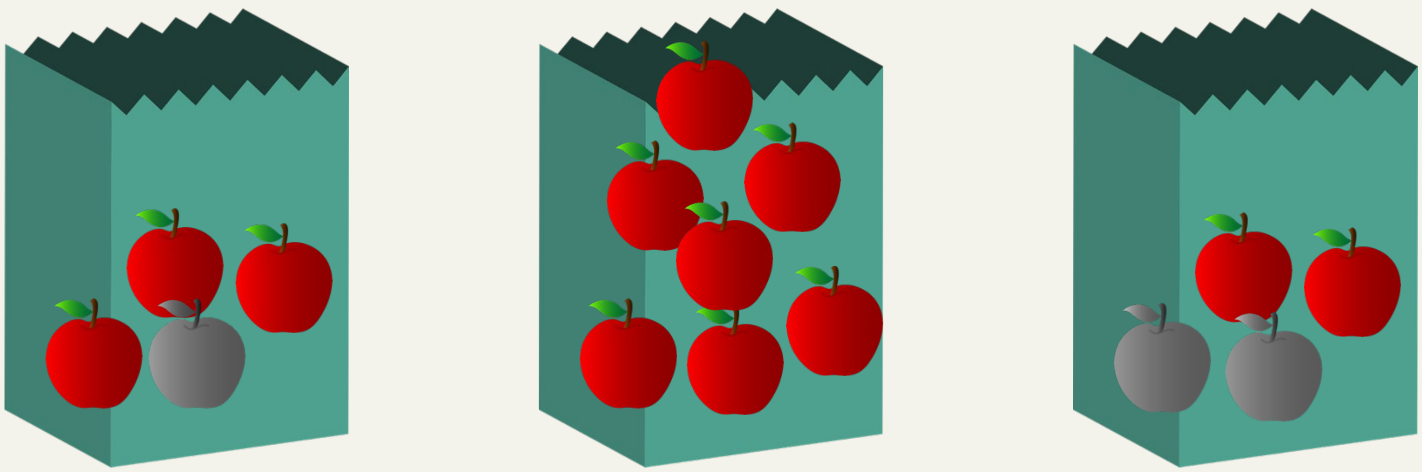
Bag of Words



	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

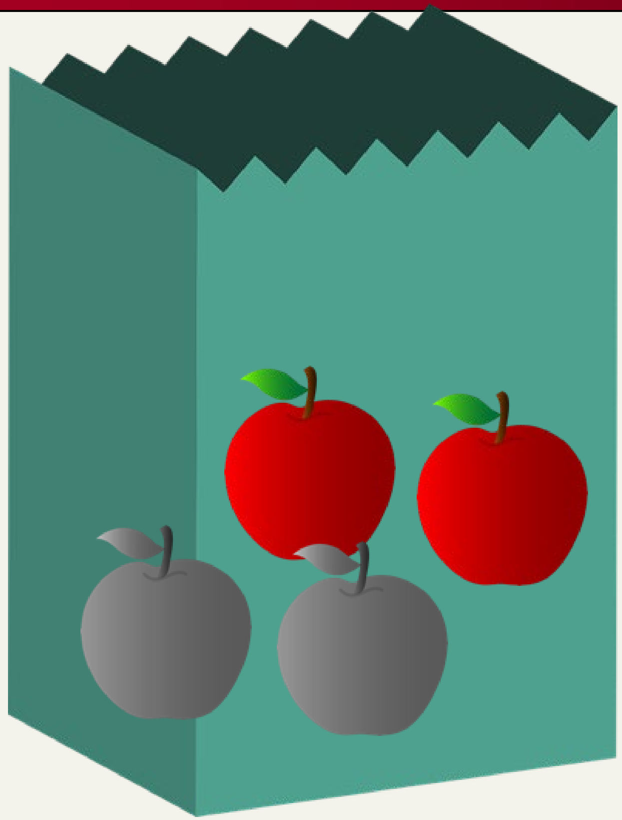
	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	5	1				
Doc 3					1	1	1	2	1	1

紅蘋果

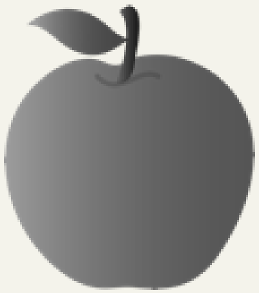


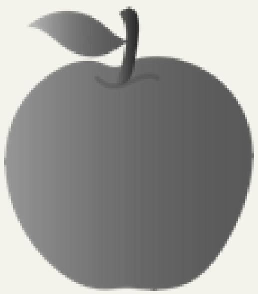
金蘋果

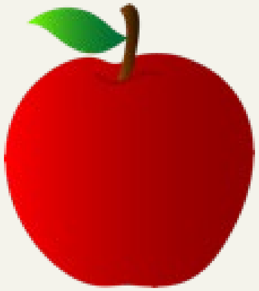
銀蘋果

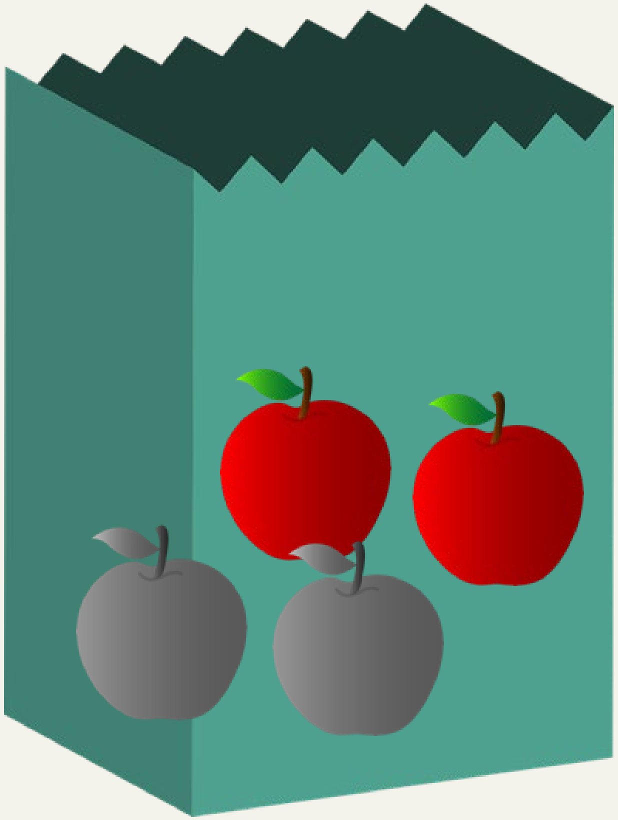


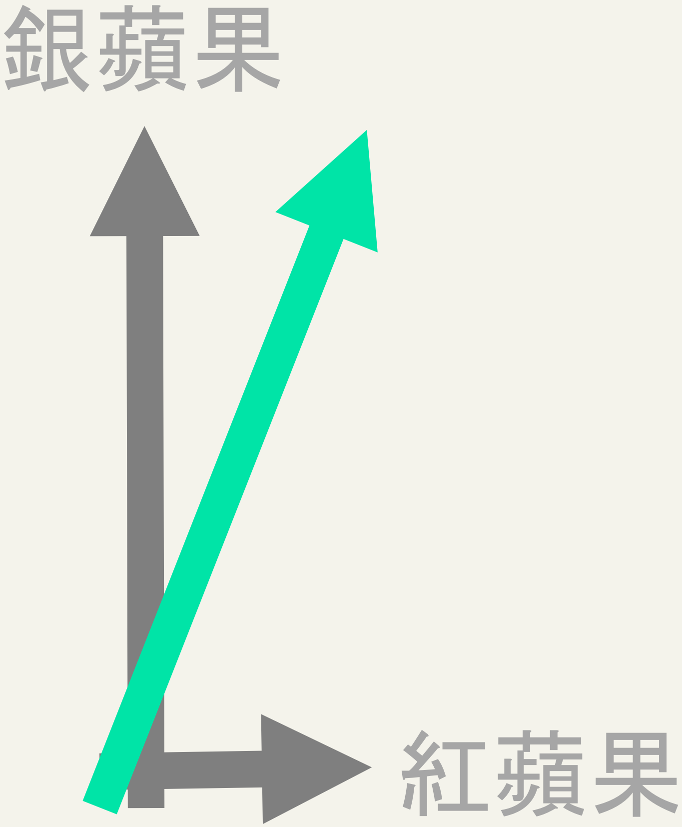
銀蘋果
對於這袋子
有多重要？

金蘋果  1/10000

銀蘋果  1/1000

紅蘋果  998.9/1000




銀蘋果
紅蘋果

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

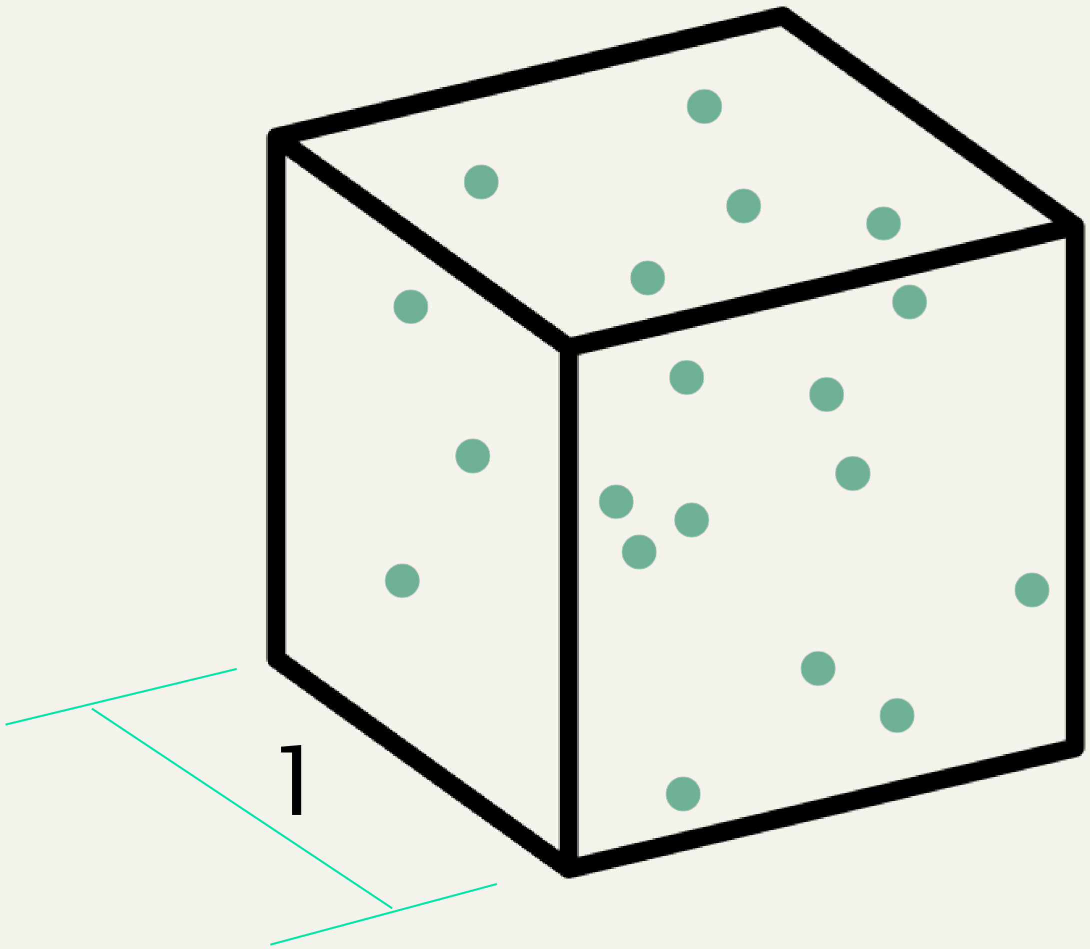
$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

WORD EMBEDDING

keras.layers.Embedding

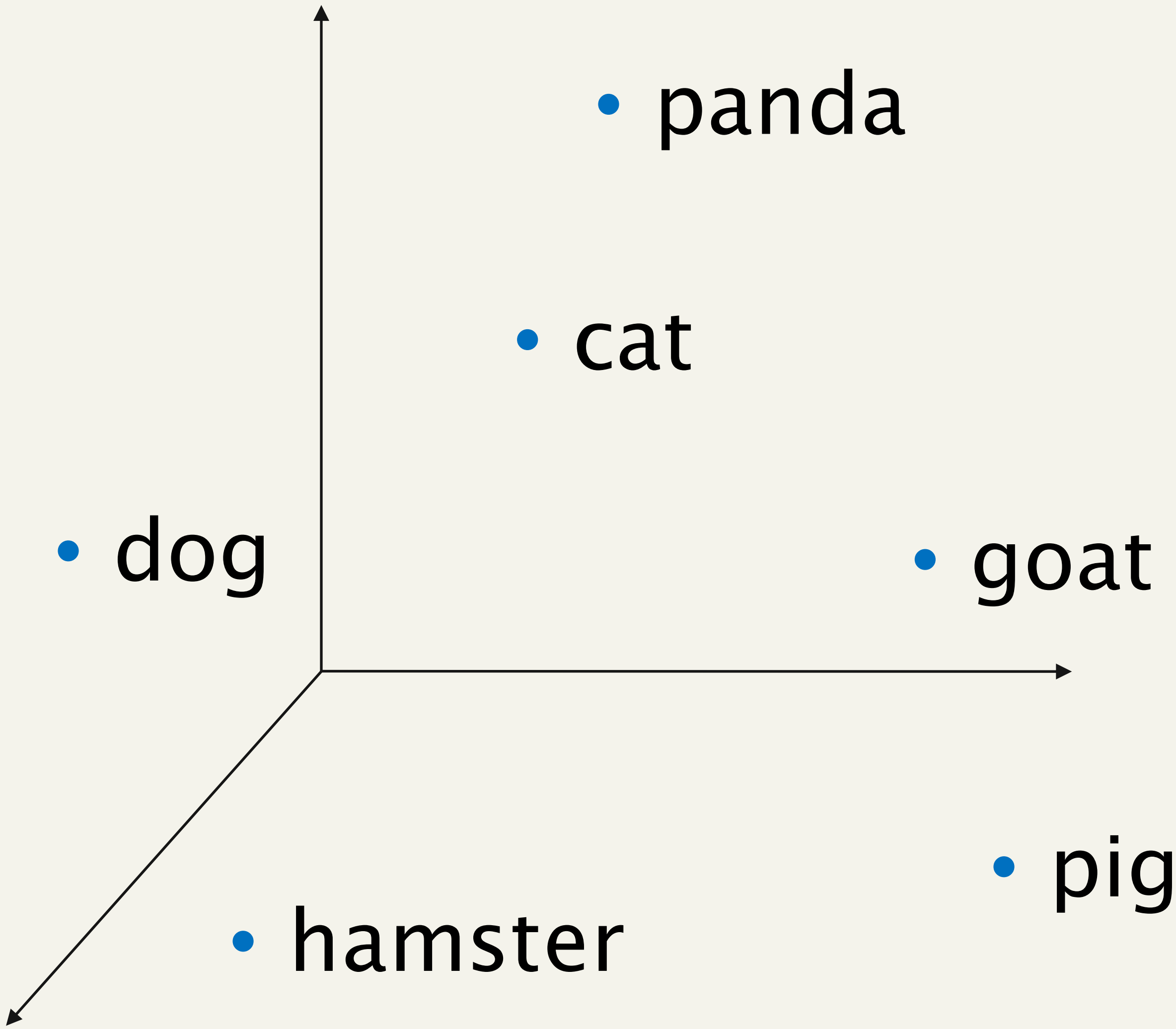


passion

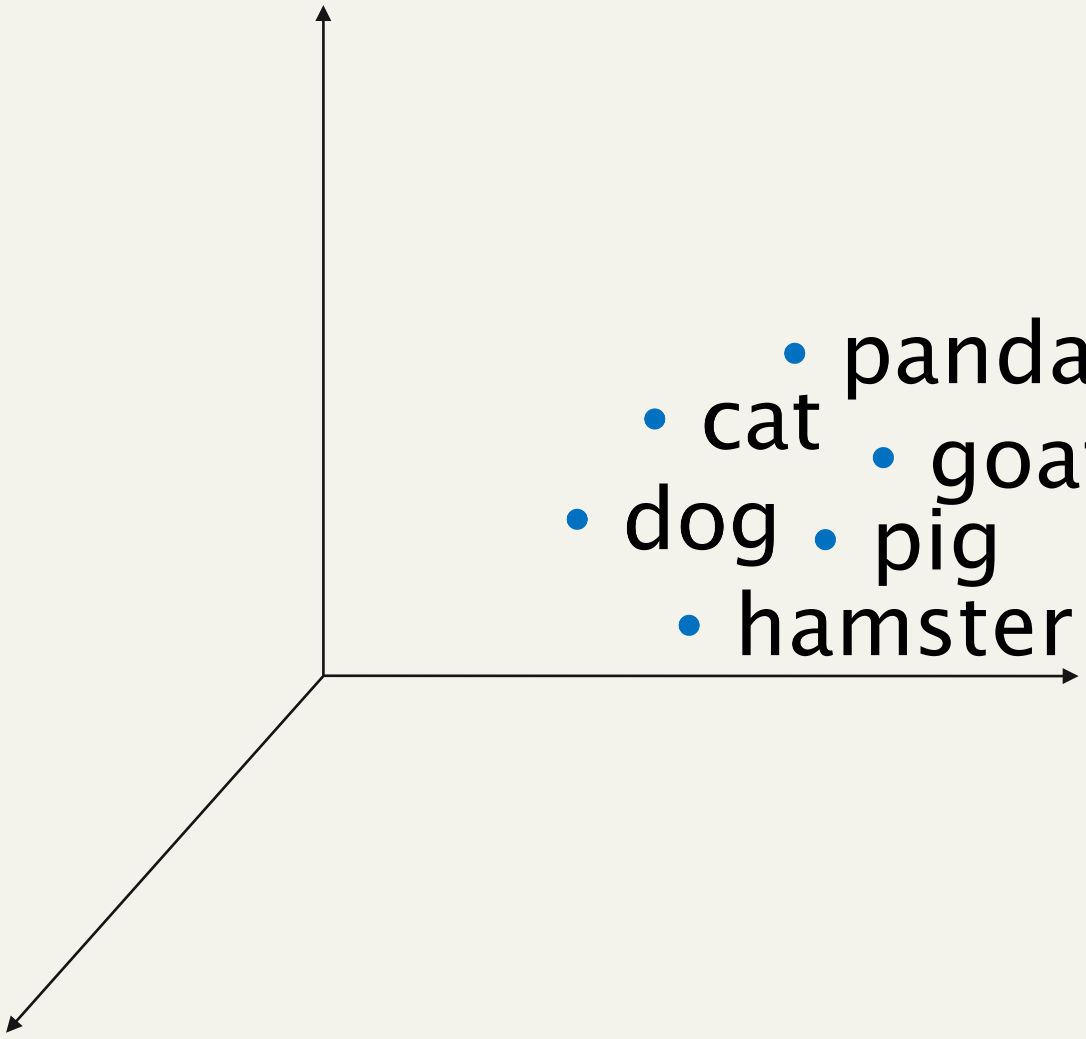
x: 0.119

y: 0.212

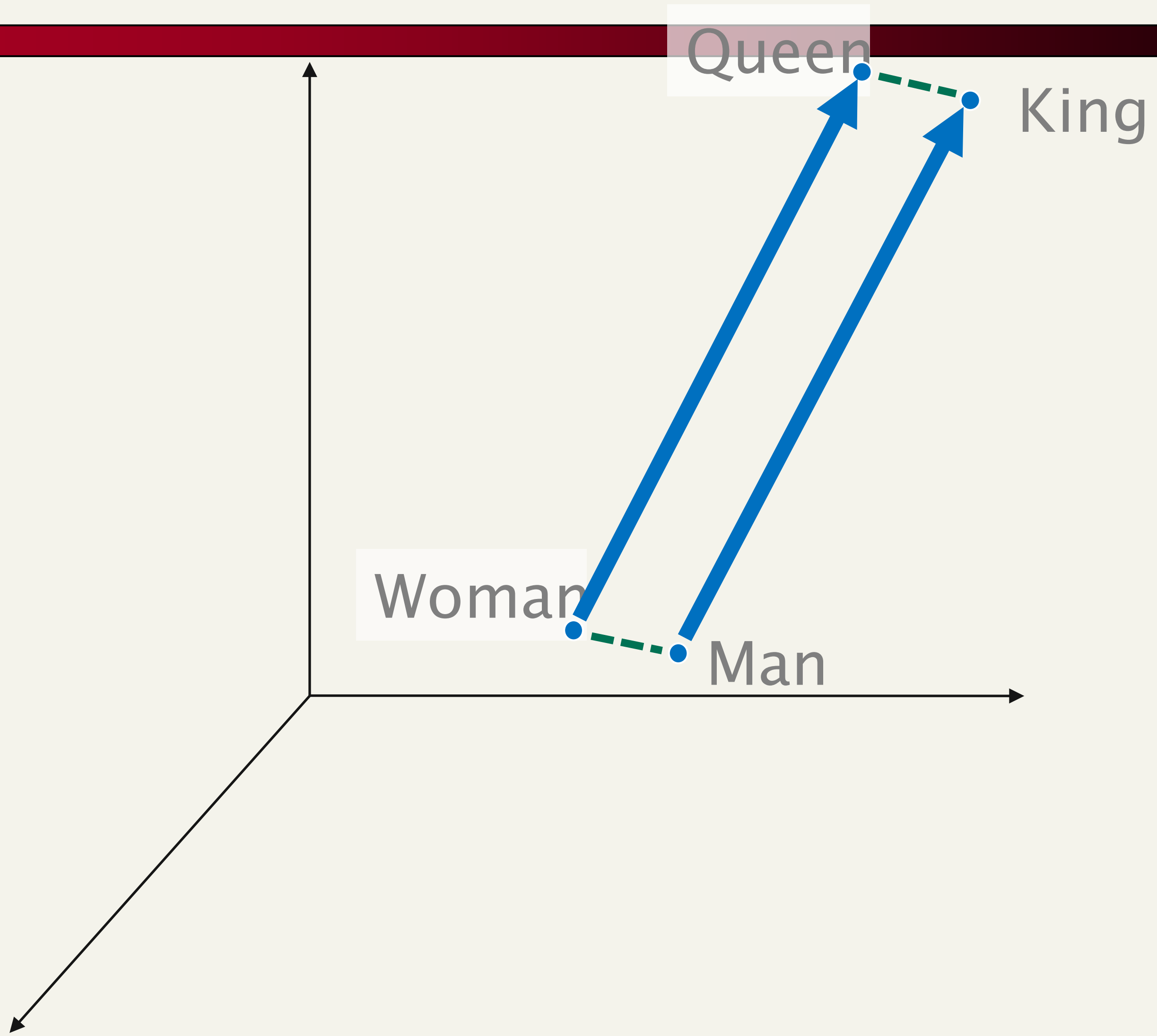
z: 0.010


panda
cat
dog
goat
pig
hamster

Word2Vec

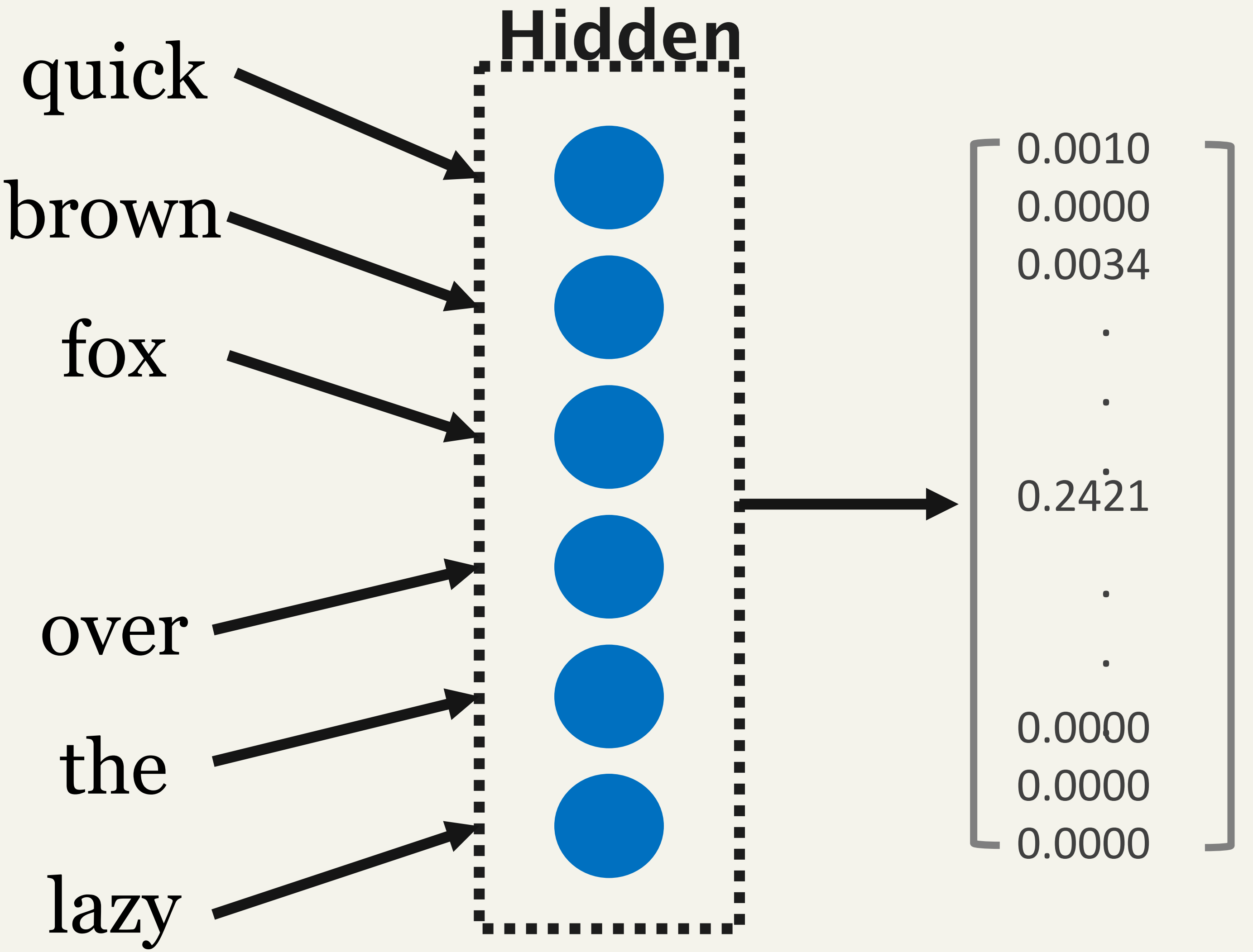


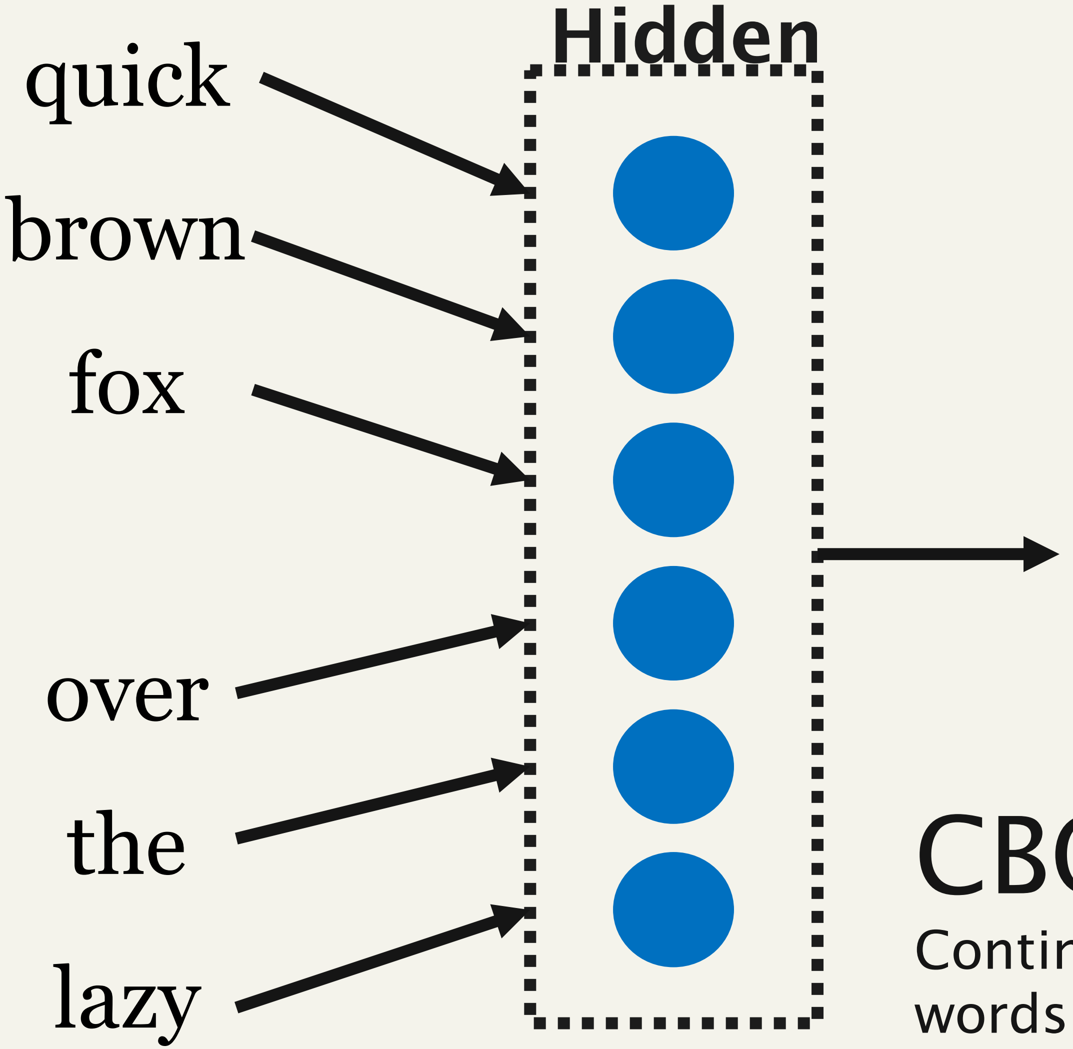
Word2Vec



" The quick brown fox ____?____ over the lazy dog "

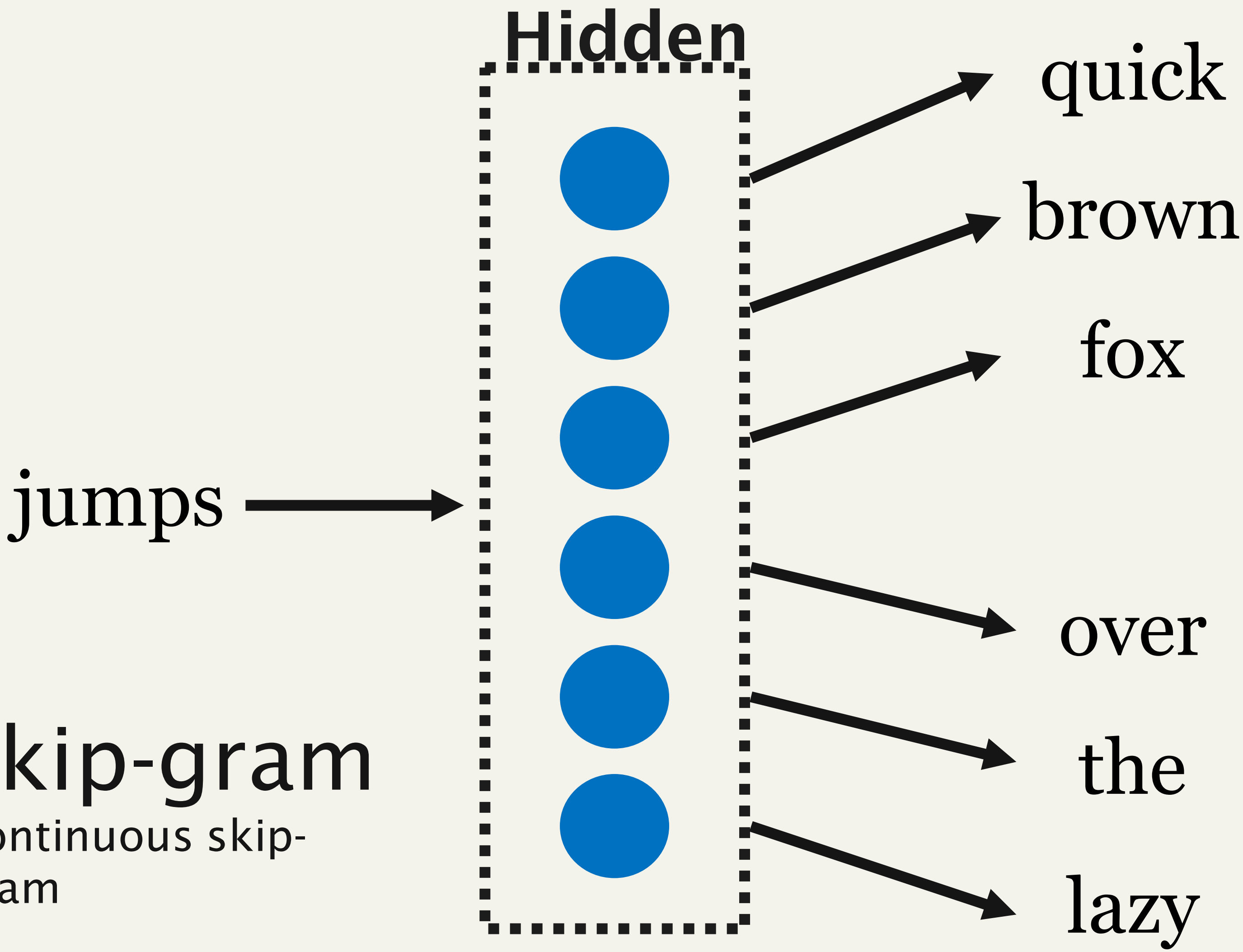
“ The quick brown fox ________ over the lazy dog ”


Hidden
quick
brown
fox
over
the
lazy
0.0010
0.0000
0.0034
.
.
.
0.2421
.
.
0.0000
0.0000
0.0000



CBOW

Continuous bag-of-words



Skip-gram

Continuous skip-gram

Dimension coordination

Probability list

$$\begin{bmatrix} 0.0010 \\ 0.0000 \\ 0.0034 \\ \cdot \\ \cdot \\ \cdot \\ 0.2421 \\ \cdot \\ \cdot \\ \cdot \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}$$

“ The quick brown fox _______ over the lazy dog ”

grammatical, semantical similarity

Word Embedding Choices

1. Learnable embedding
2. Word2Vec
3. GloVe
4. FastText

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?

Tokenization

- 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.
    - ***co-education***
    - ***lowercase***, ***lower-case***, ***lower case*** ?
    - It's effective to get the user to put in possible hyphens
  - ***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***
- ***(800) 234-2333***
  - Often have embedded spaces
  - 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: Lecture 3)
  - Will often index “meta-data” separately
    - Creation date, format, etc.

Tokenization: language issues

- French
  - ***L'ensemble*** → one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
    - Want ***l'ensemble*** to match with ***un ensemble***
- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module

Tokenization: language issues

- Chinese 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 in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

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)

Katakana Hiragana Kanji Romaji

End-user can express query entirely in hiragana!

Tokenization: language issues

- Arabic (or Hebrew) is 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.

- ← → ← → ← start
- ‘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

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*
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques (lecture 5) 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”

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***
- We most commonly implicitly 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, window***
  - Enter: ***Windows*** Search: ***Windows***
- Potentially more powerful, but less efficient

Normalization: other languages

- 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

- 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 …

Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***
  - Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization…

- Aug 2005 Google example:
  - ***C.A.T.*** → Cat Fanciers website *not* Caterpiller Inc.

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
  - Invented for the US Census
  - E.g., ***chebyshev*** $\rightarrow$ ***tchebycheff***
- More on this in the next lecture

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *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

Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Typical rules in Porter

- *sses* $\rightarrow$ *ss*
- *ies* $\rightarrow$ *i*
- *ational* $\rightarrow$ *ate*
- *tional* $\rightarrow$ *tion*

- Weight of word sensitive rules
- *(m>1) EMENT* $\rightarrow$
  - *replacement* $\rightarrow$ *replac*
  - *cement* $\rightarrow$ *cement*

Other stemmers

- Other stemmers exist, e.g., Lovins stemmer
  http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Single-pass, longest suffix removal (about 250 rules)
- Full morphological analysis – at most modest benefits for retrieval
- Do stemming and other normalizations help?
  - English: very mixed results. Helps recall for some queries but harms precision on others
    - E.g., operative (dentistry) ⇒ oper
  - Definitely useful for Spanish, German, Finnish, …

Language-specificity

- Many of the above features embody transformations that are
  - Language-specific and
  - Often, application-specific
- These are “plug-in” addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

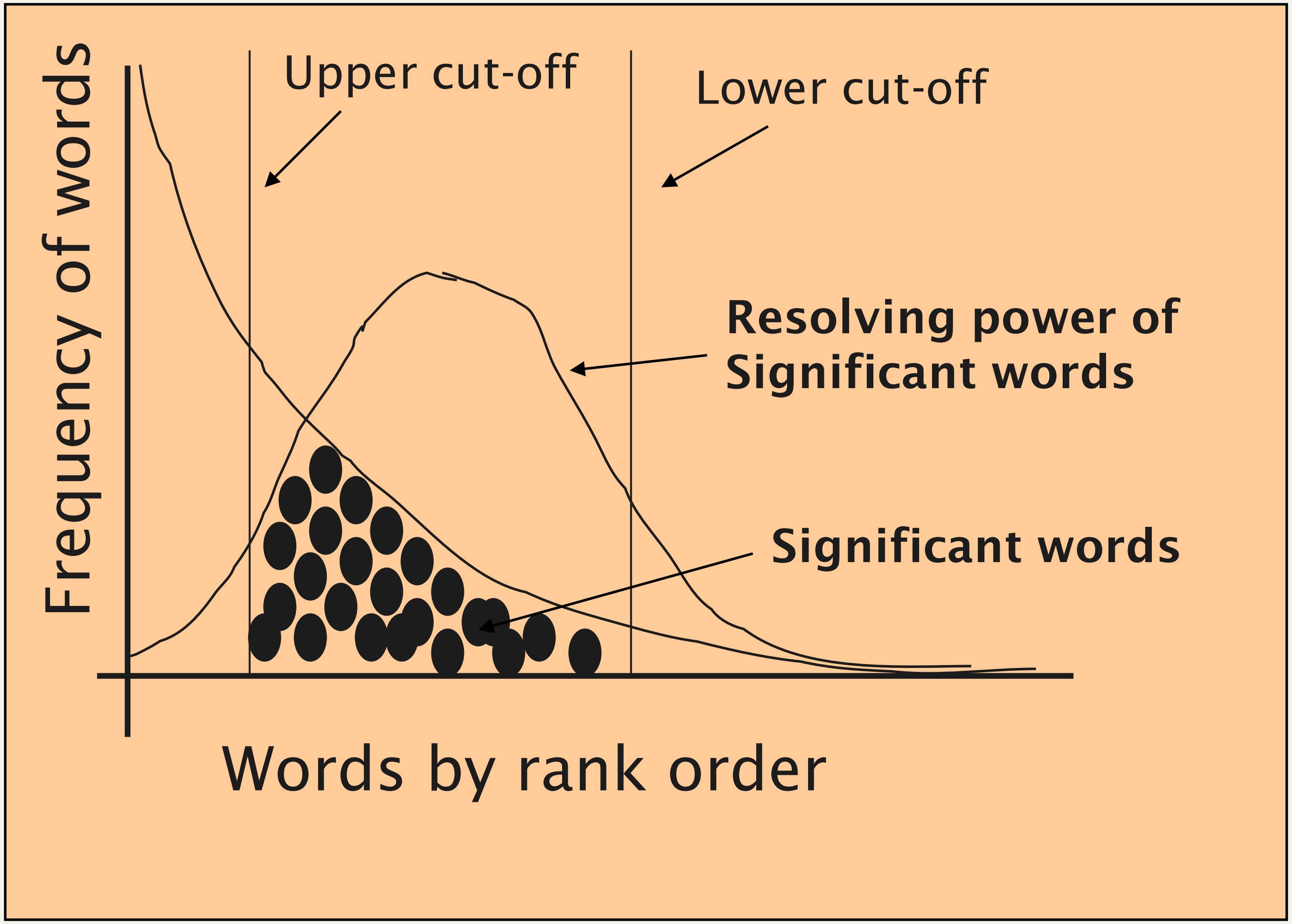
Dictionary entries – first cut

ensemble.french
時間.chinese
MIT.english
mit.german
guaranteed.english
entries.english
sometimes.english
tokenization.english

These may be grouped by language (or not…).
More on this in ranking/query processing.

Word Frequency vs. Resolving Power (from van Rijsbergen 79)

The most frequent words are *not* the most descriptive

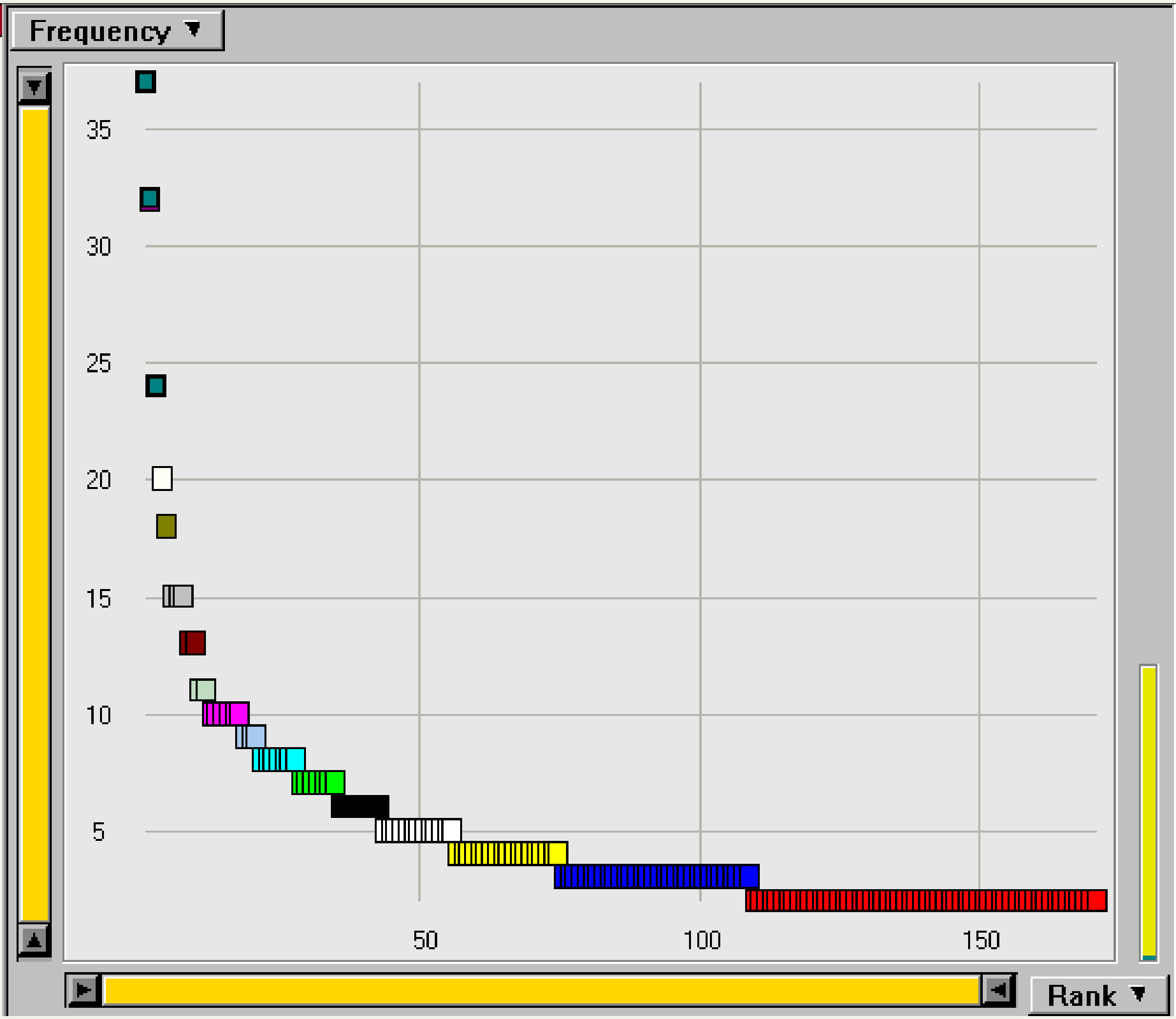


Plotting Word Frequency by Rank

- Say for a text with 100 tokens
- Count
  - How many tokens occur 1 time (50)
  - How many tokens occur 2 times (20) …
  - How many tokens occur 7 times (10) …
  - How many tokens occur 12 times (1)
  - How many tokens occur 14 times (1)
- So things that occur the most times have the highest rank (rank 1).
- Things that occur the fewest times have the lowest rank (rank n).

The Corresponding Zipf Curve

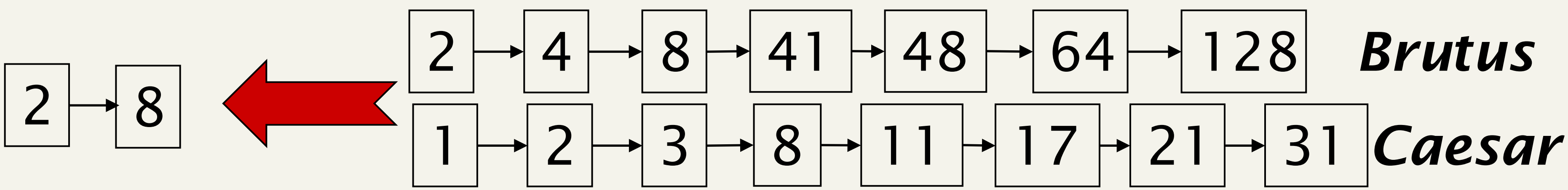
Rank	Freq	
1	37	system
2	32	knowledg
3	24	base
4	20	problem
5	18	abstract
6	15	model
7	15	languag
8	15	implem
9	13	reason
10	13	inform
11	11	expert
12	11	analysi
13	10	rule
14	10	program
15	10	oper
16	10	evalu
17	10	comput
18	10	case
19	9	gener
20	9	form



Faster postings merges:
Skip pointers/Skip lists

Recall basic merge

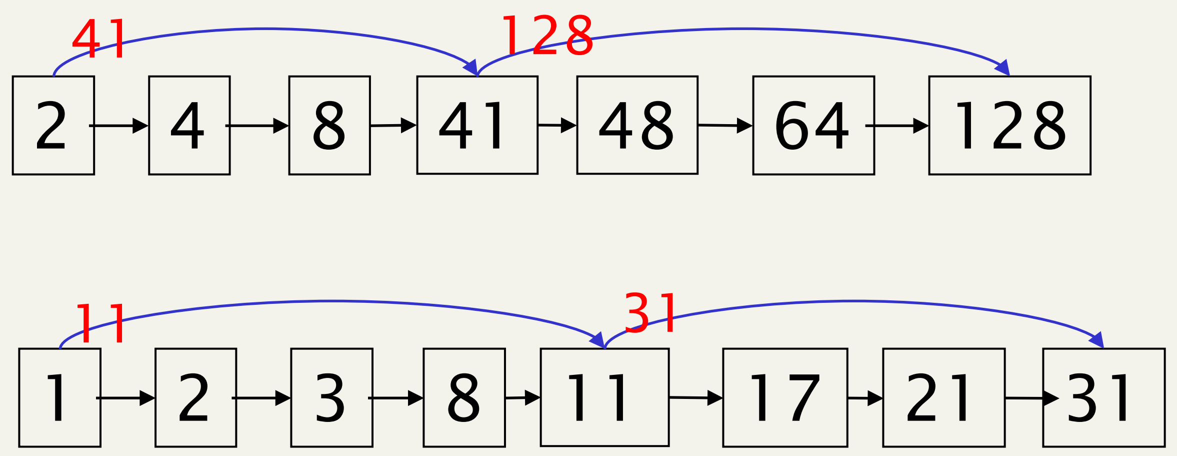
- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *m* and *n*, the merge takes O(*m*+*n*) operations.

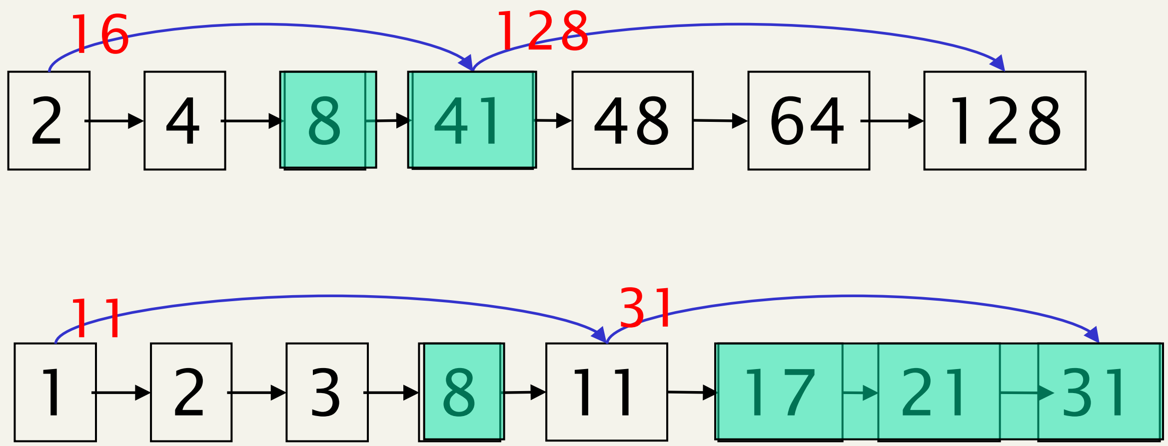
Can we do better?
Yes (if index isn't changing too fast).

Augment postings with skip pointers (at indexing time)



- Why?
- <u>To skip postings that will not figure in the search results.</u>
- How?
- Where do we place skip pointers?

Query processing with skip pointers



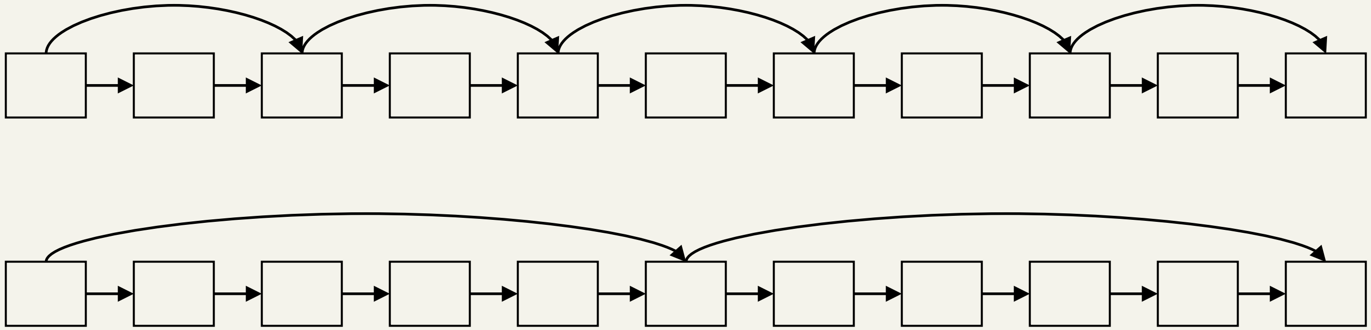
Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.

We then have **41** and **11** on the lower. **11** is smaller.

But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

Where do we place skips?

- Tradeoff:
  - More skips $\rightarrow$ shorter skip spans $\Rightarrow$ more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips $\rightarrow$ few pointer comparison, but then long skip spans $\Rightarrow$ few successful skips.



Placing skips

- Simple heuristic: for postings of length L, use $\sqrt{L}$ evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if L keeps changing because of updates.
- This definitely used to help; with modern hardware it may not (Bahle et al. 2002)
  - The I/O cost of loading a bigger postings list can outweigh the gains from quicker in memory merging!

Phrase queries and positional indexes

Phrase queries

- 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
  <*term* : *docs*> 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 are processed as we did with wild-cards:
- ***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!

Extended biwords

- Parse the indexed text and perform part-of-speech-tagging (POST).
- Bucket the terms into (say) Nouns (N) and articles/prepositions (X).
- Now deem any string of terms of the form NX*N to be an <u>extended biword</u>.
  - Each such extended biword is now made a term in the dictionary.
- Example: ***catcher in the rye***

  N X X N
- Query processing: parse it into N's and X's
  - Segment query into enhanced biwords
  - Look up index

Issues for biword indexes

- False positives, as noted before
- Index blowup due to bigger dictionary

- For extended biword index, parsing longer queries into conjunctions:
  - E.g., the query ***tangerine trees and marmalade skies*** is parsed into
  - ***tangerine trees*** *AND* ***trees and marmalade*** *AND* ***marmalade skies***

- Not standard solution (for all biwords)

Solution 2: Positional indexes

- In the postings, store, for each ***term***, entries of the form:

  <***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***"?

- 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
  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
  - There's likely to be a problem on it!

Positional index size

- You can compress position values/offsets: we'll talk about that in lecture 5
- Nevertheless, a positional index expands postings storage *substantially*
- 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

Why?

  - 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

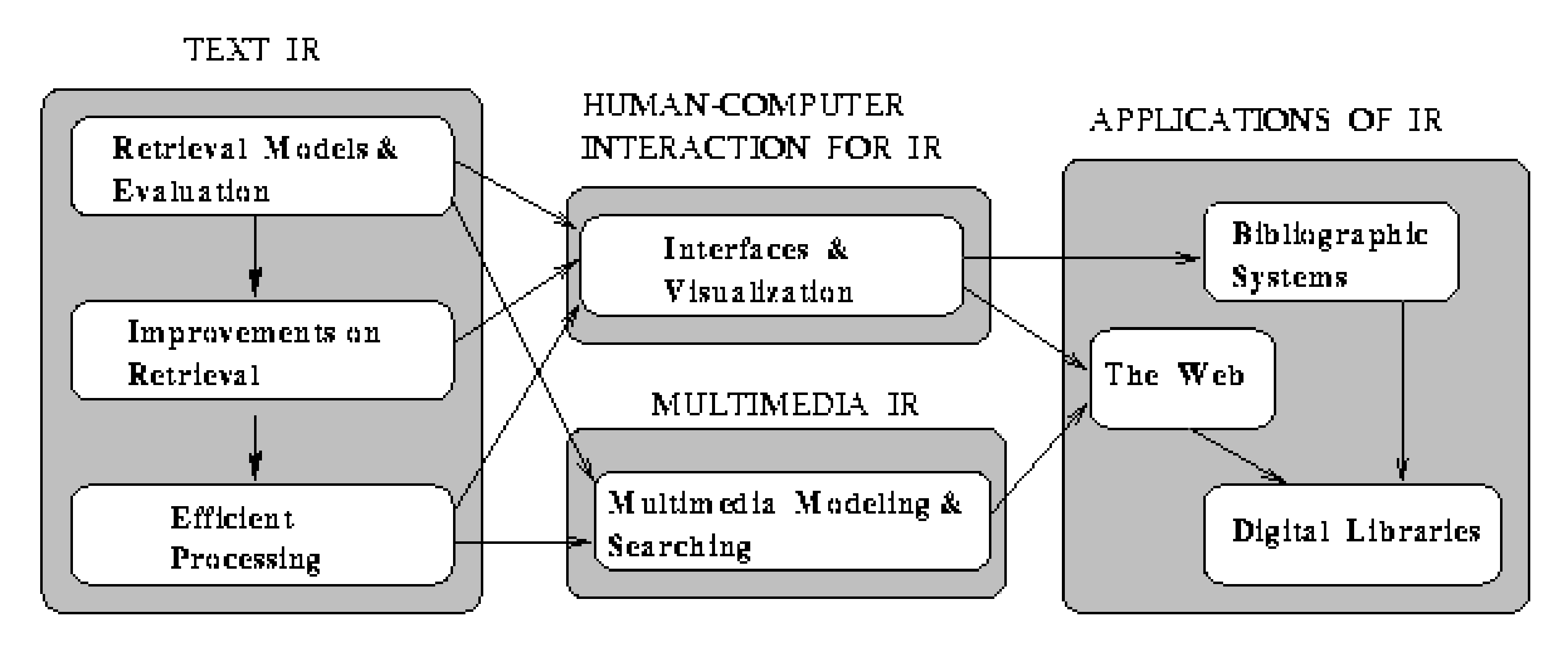
Rules of thumb

- 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

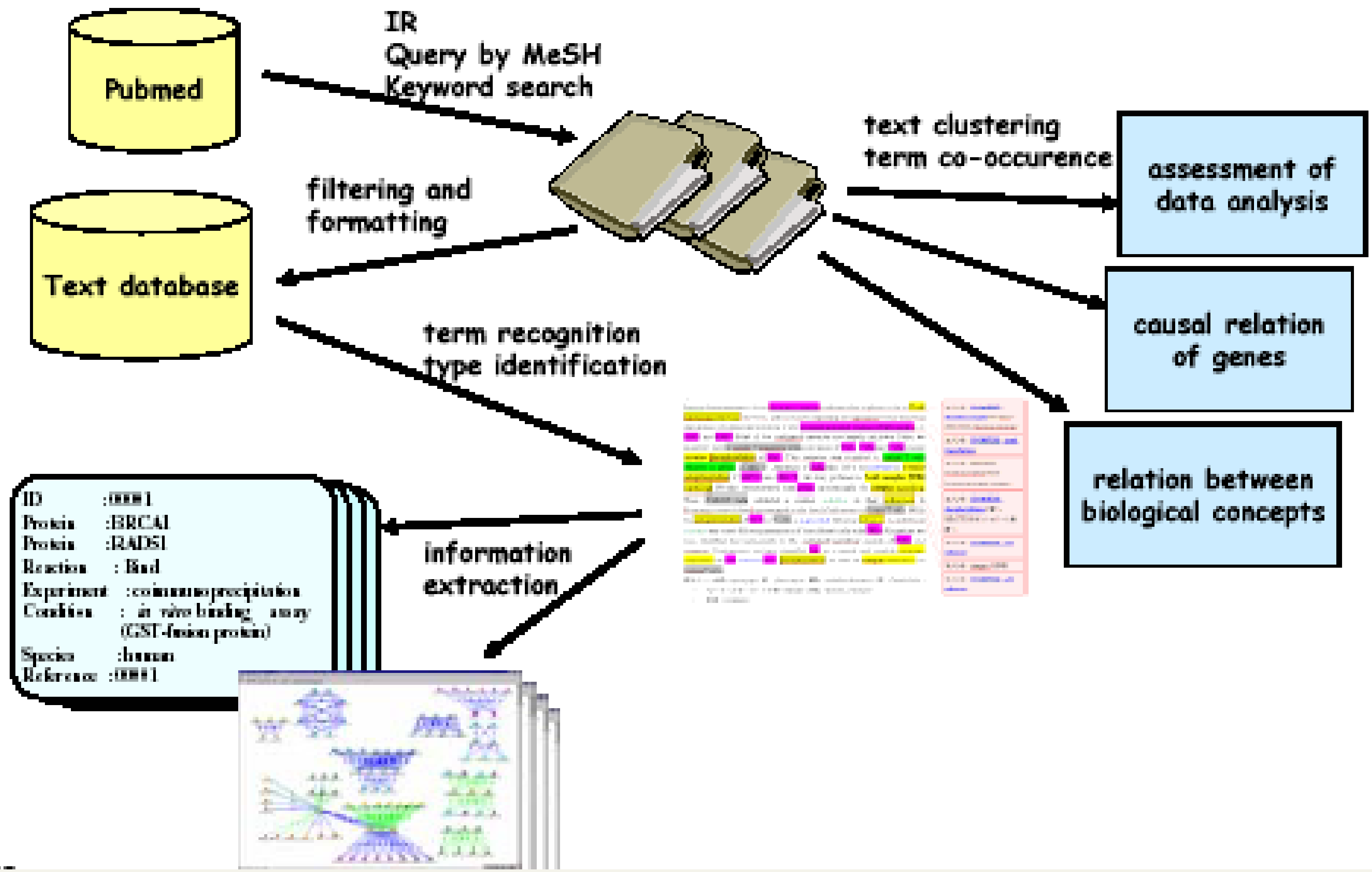
Combination schemes

- 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 ¼ of the time of using just a positional index
  - It required 26% more space than having a positional index alone

Research Topics of IR

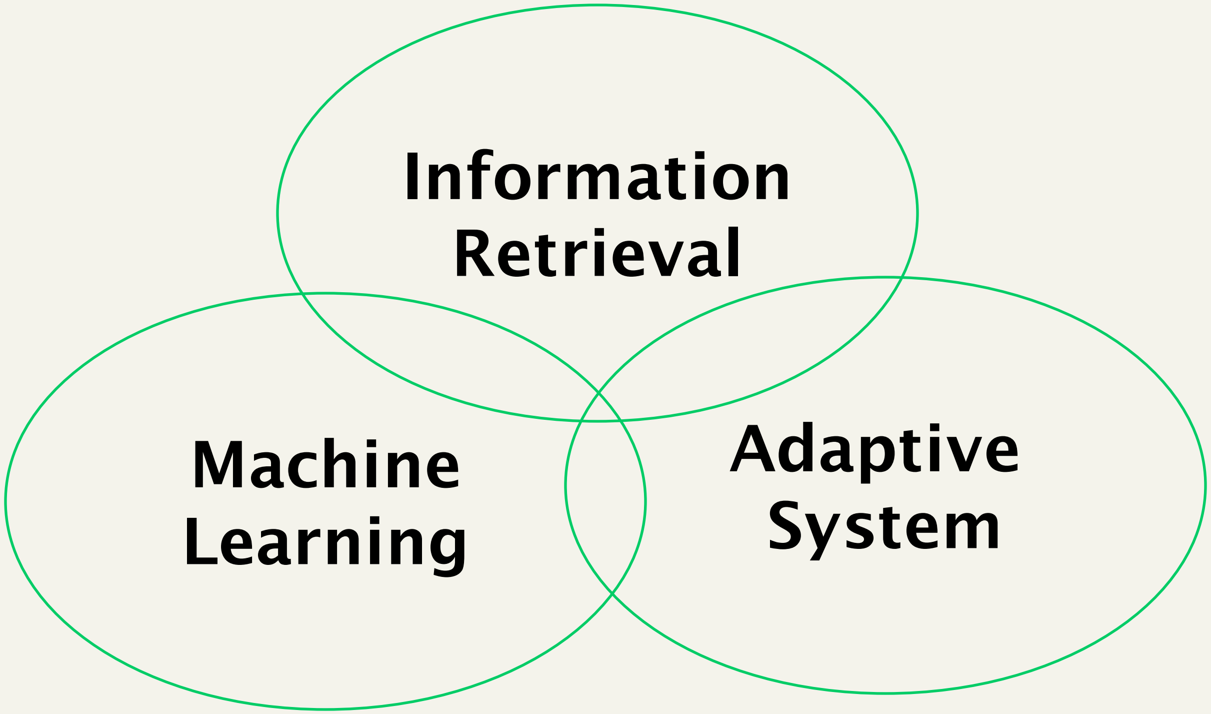


Overview of Text Processing in Biology



Intelligent Information Retrieval

Intelligent Information Retrieval (IIR)



Some Issues in IIR

- Document Clustering
- Automatic Text Categorization
- Feature Selection
- Topic Detection and Tracking
- New Information Detection

Document Clustering

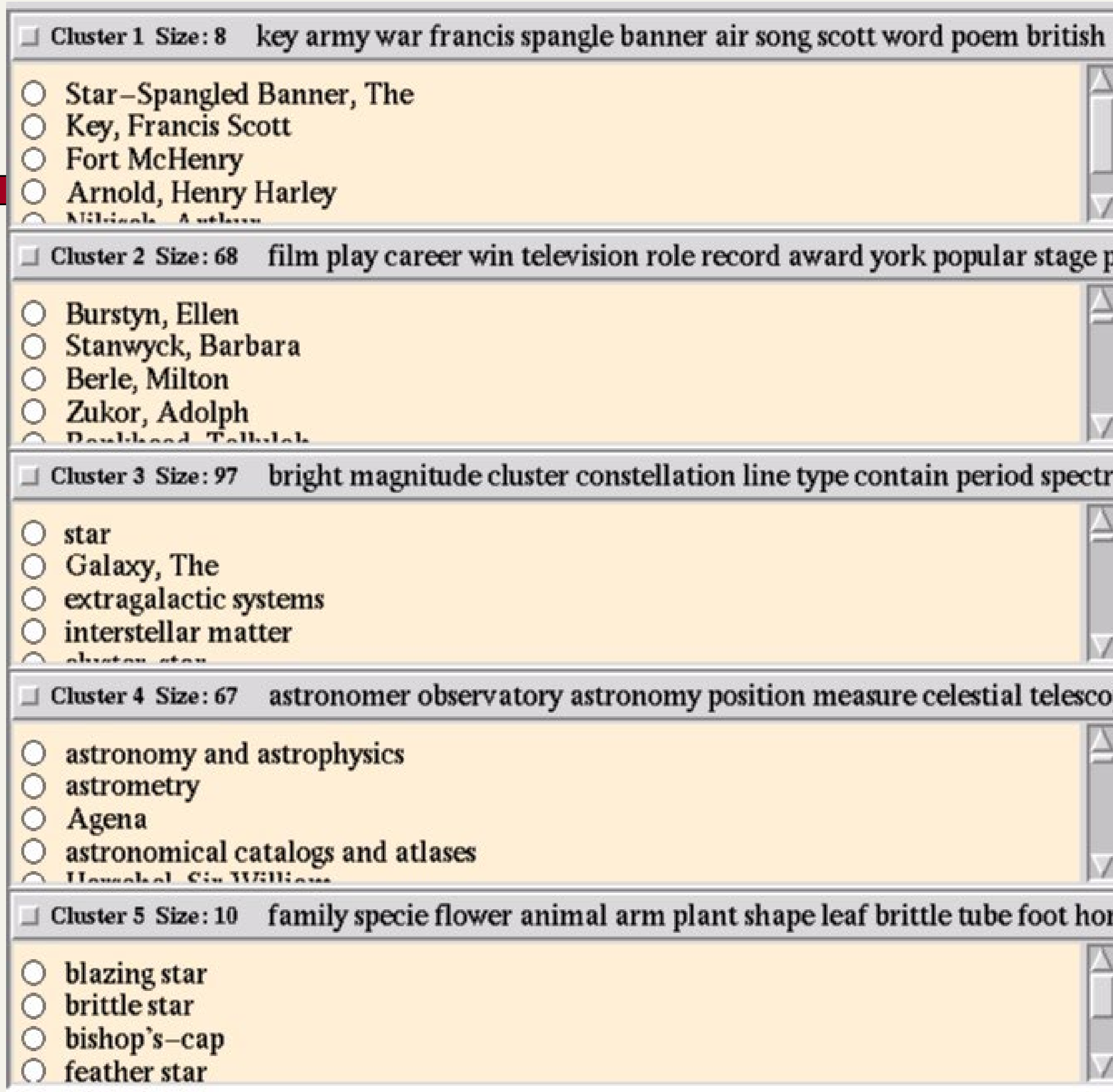
- Technique for analyzing structures and relations in data
- No classes to be identified prior to process
- Intensive literature on
  - medical data
  - census and survey data
  - literature citations
  - document retrieval

Document Clustering

- Web browsing ("Scatter/Gather")
- Taxonomy creation (Yahoo!)
- Term thesaurus development (WordNet)
- Query-log analysis on the web
- User grouping for email routing
- Summarization

Text Clustering

- Finds overall similarities among groups of documents
- Finds overall similarities among groups of tokens
- Picks out some themes, ignores others


Cluster 1 Size: 8 key army war francis spangle banner air song scott word poem british
Star–Spangled Banner, The
Key, Francis Scott
Fort McHenry
Arnold, Henry Harley
Cluster 2 Size: 68 film play career win television role record award york popular stage
Burstyn, Ellen
Stanwyck, Barbara
Berle, Milton
Zukor, Adolph
Cluster 3 Size: 97 bright magnitude cluster constellation line type contain period
star
Galaxy, The
extragalactic systems
interstellar matter
Cluster 4 Size: 67 astronomer observatory astronomy position measure celestial
astronomy and astrophysics
astrometry
Agena
astronomical catalogs and atlases
Cluster 5 Size: 10 family specie flower animal arm plant shape leaf brittle tube foot
blazing star
brittle star
bishop's–cap
feather star

Clustering as Document Ranking

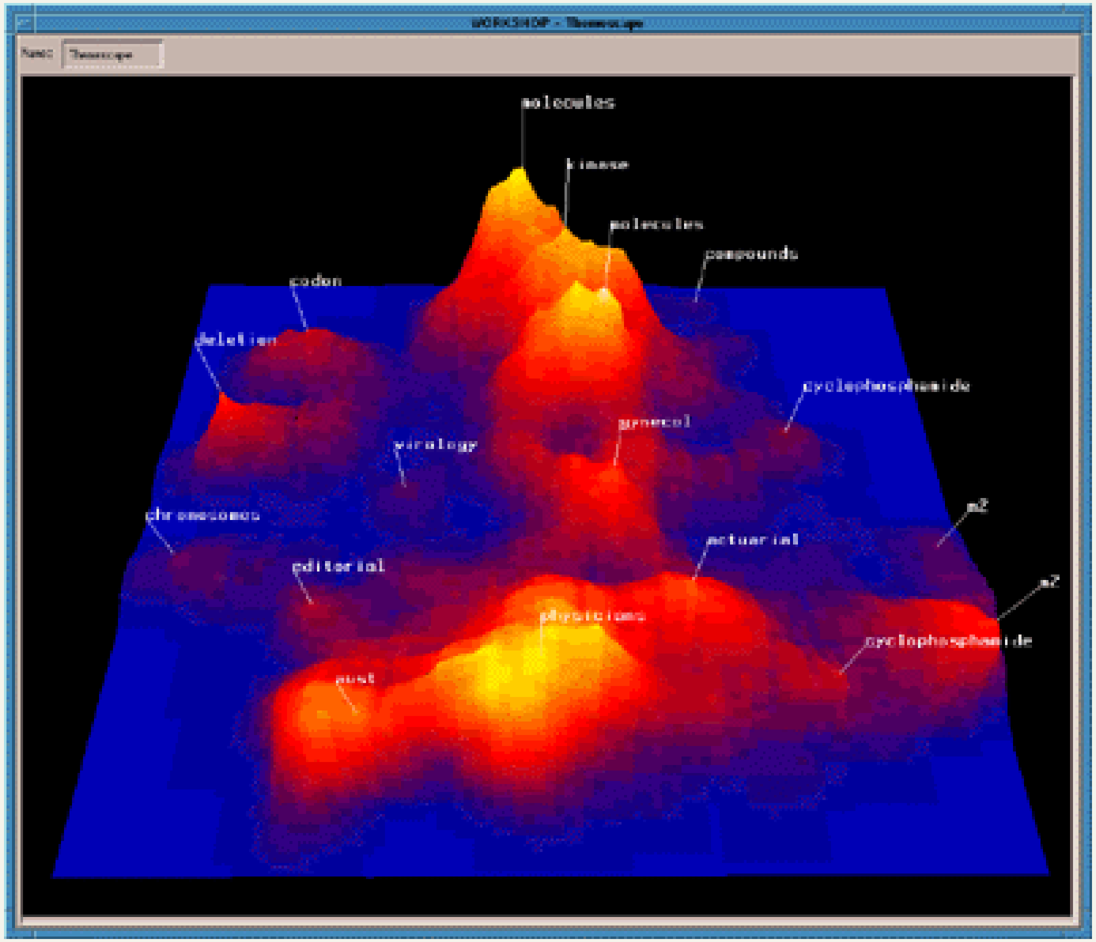
- Cluster entire collection
- Find cluster centroid that best matches the query
- This has been explored extensively
  - it is expensive
  - it doesn't work well

Two Queries: Two Clusterings

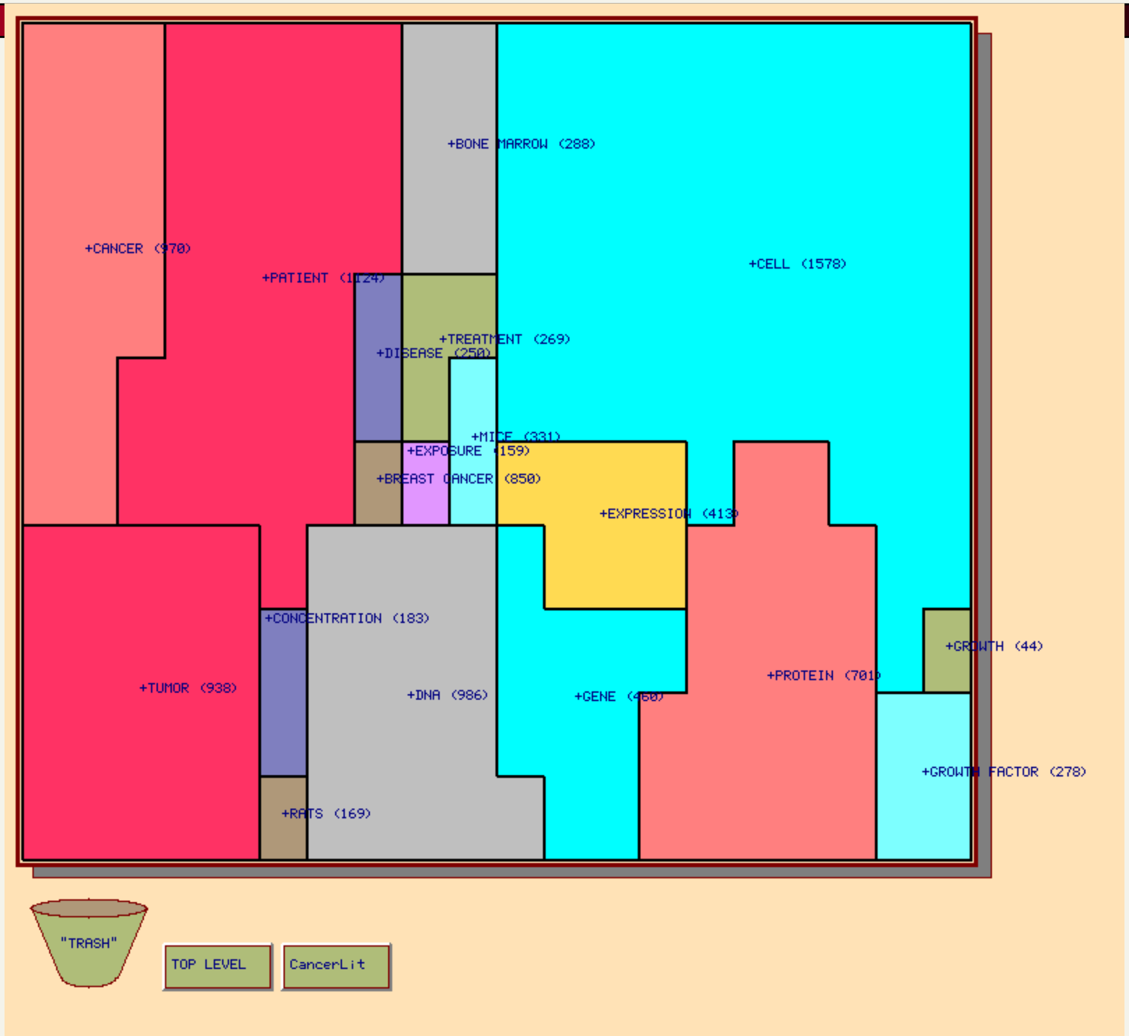
AUTO, CAR, ELECTRIC	AUTO, CAR, SAFETY
8 control drive accident …	6 control inventory integrate …
25 battery california technology …	10 investigation washington …
48 import j. rate honda toyota …	12 study fuel death bag air …
16 export international unit japan	61 sale domestic truck import …
3 service employee automatic …	11 japan export defect unite …

The main differences are the clusters that are central to the query

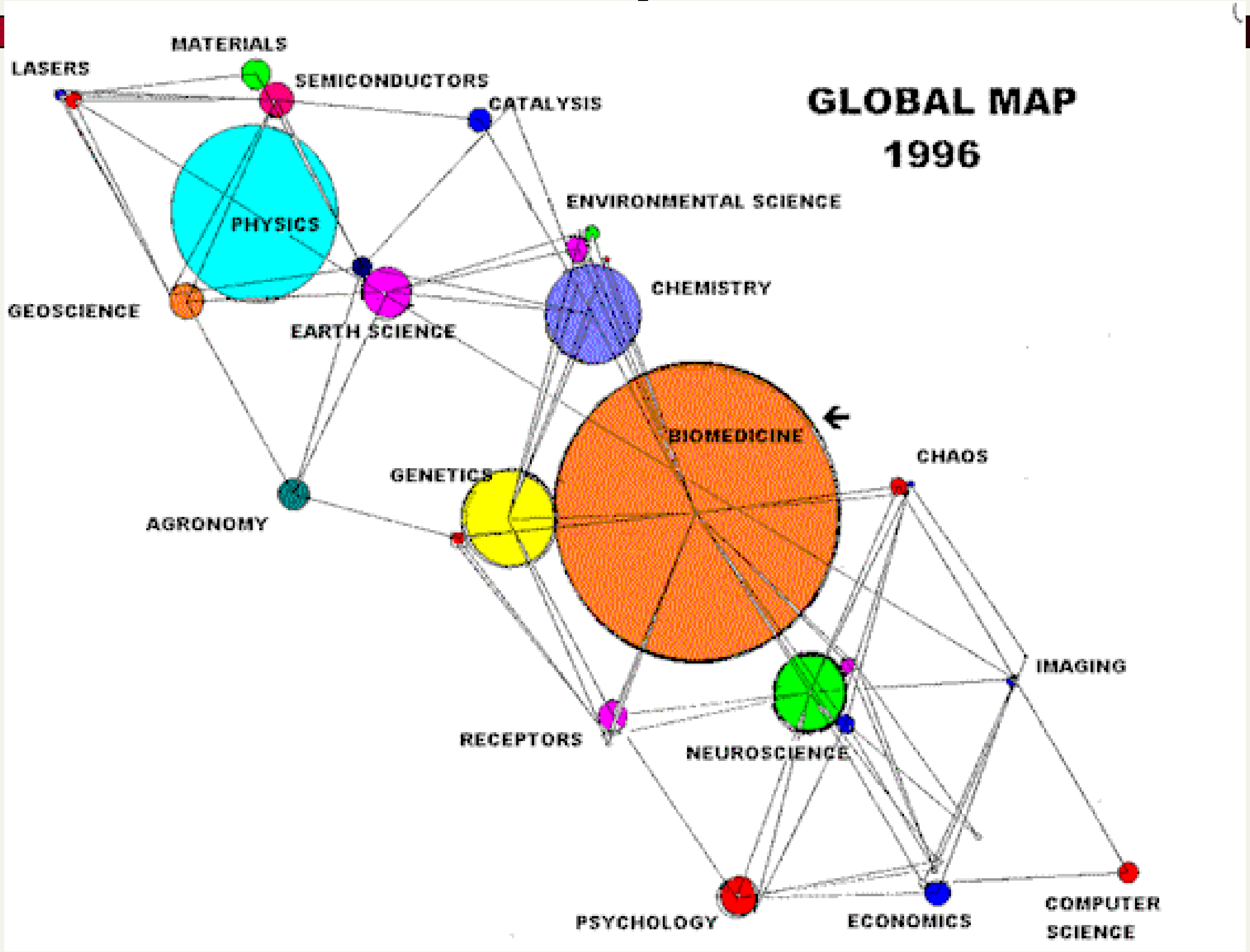
Clustering Multi-Dimensional Document Space
(image from Wise et al 95)



Kohonen Feature Maps on Text (from Chen et al., JASIS 49(7))



Co-citation analysis (From Garfield 98)



Co-citation analysis (From Garfield 98)