Complex Data Mining & Workflow Mining

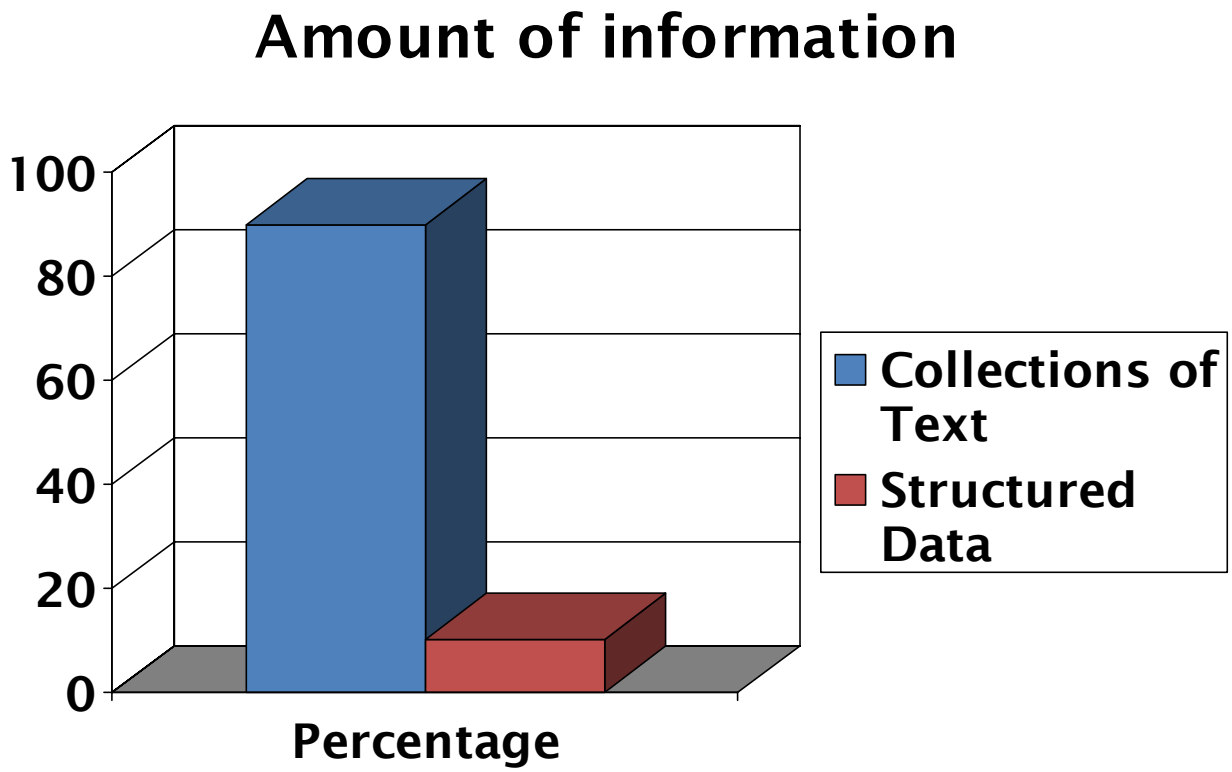
Introduzione al text mining

ICAR CNR
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Outline

- Introduzione e concetti di base
  - Motivazioni, applicazioni
  - Concetti di base nell'analisi dei dati complessi
- Text/Web Mining
  - Concetti di base sul Text Mining
  - Tecniche di data mining su dati testuali
- Graph Mining
  - Introduzione alla graph theory
  - Principali tecniche e applicazioni
- Workflow Mining
  - I workflow: grafi con vincoli
  - Frequent pattern mining su workflow: motivazioni, metodi, applicazioni
- Multi-Relational data mining
  - Motivazioni: da singole tabelle a strutture complesse
  - Alcune delle tecniche principali

The Reason for Text Mining…



Corporate Knowledge “Ore”

- Email
- Insurance claims
- News articles
- Web pages
- Patent portfolios
- IRC
- Scientific articles
- Customer complaint letters
- Contracts
- Transcripts of phone calls with customers
- Technical documents

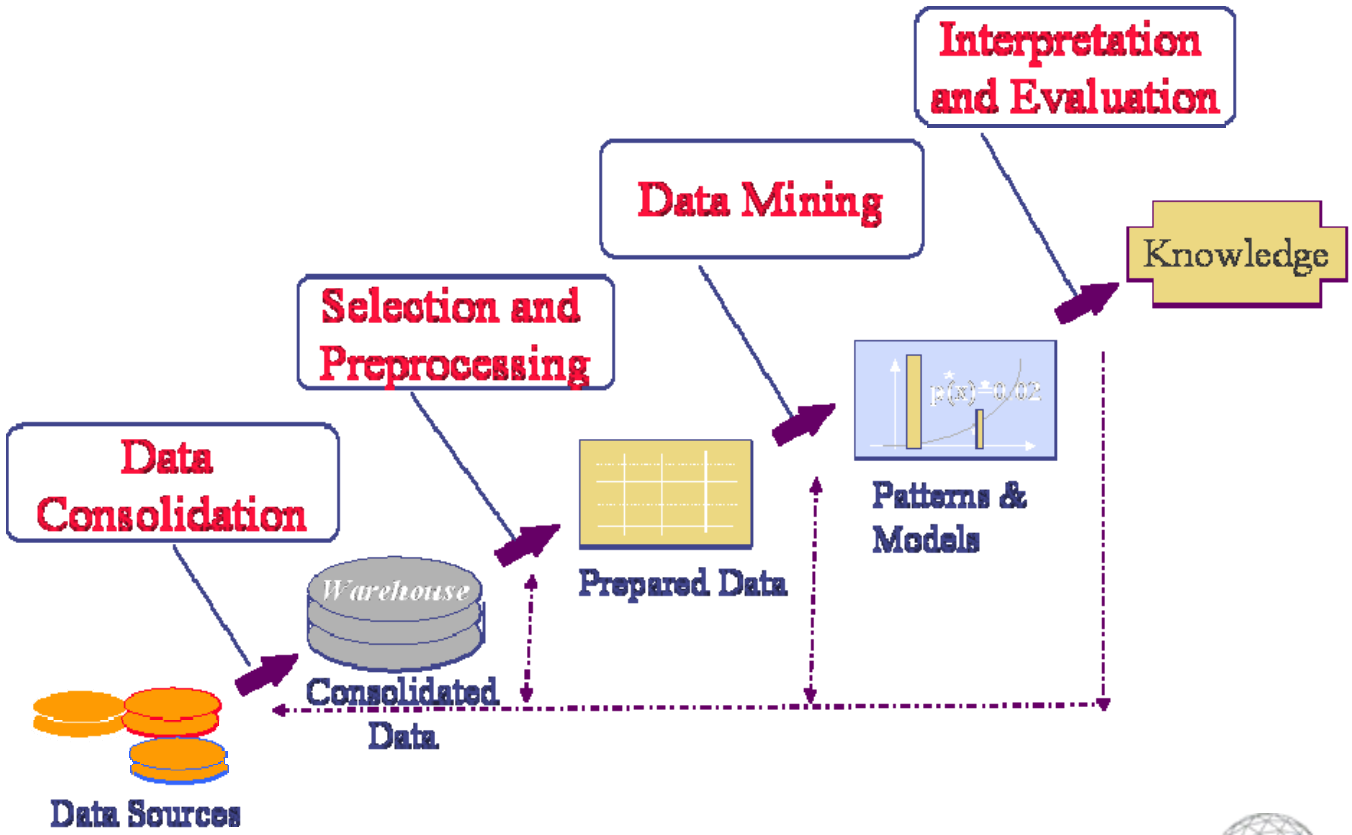
Problems with textual data (I)

- Known KDD challenges extend to textual data
  - Large (textual) data collections
  - High dimensionality
  - Overfitting
  - Changing data and knowledge
  - Noisy data
  - Understandability of mined patterns
  - Etc.

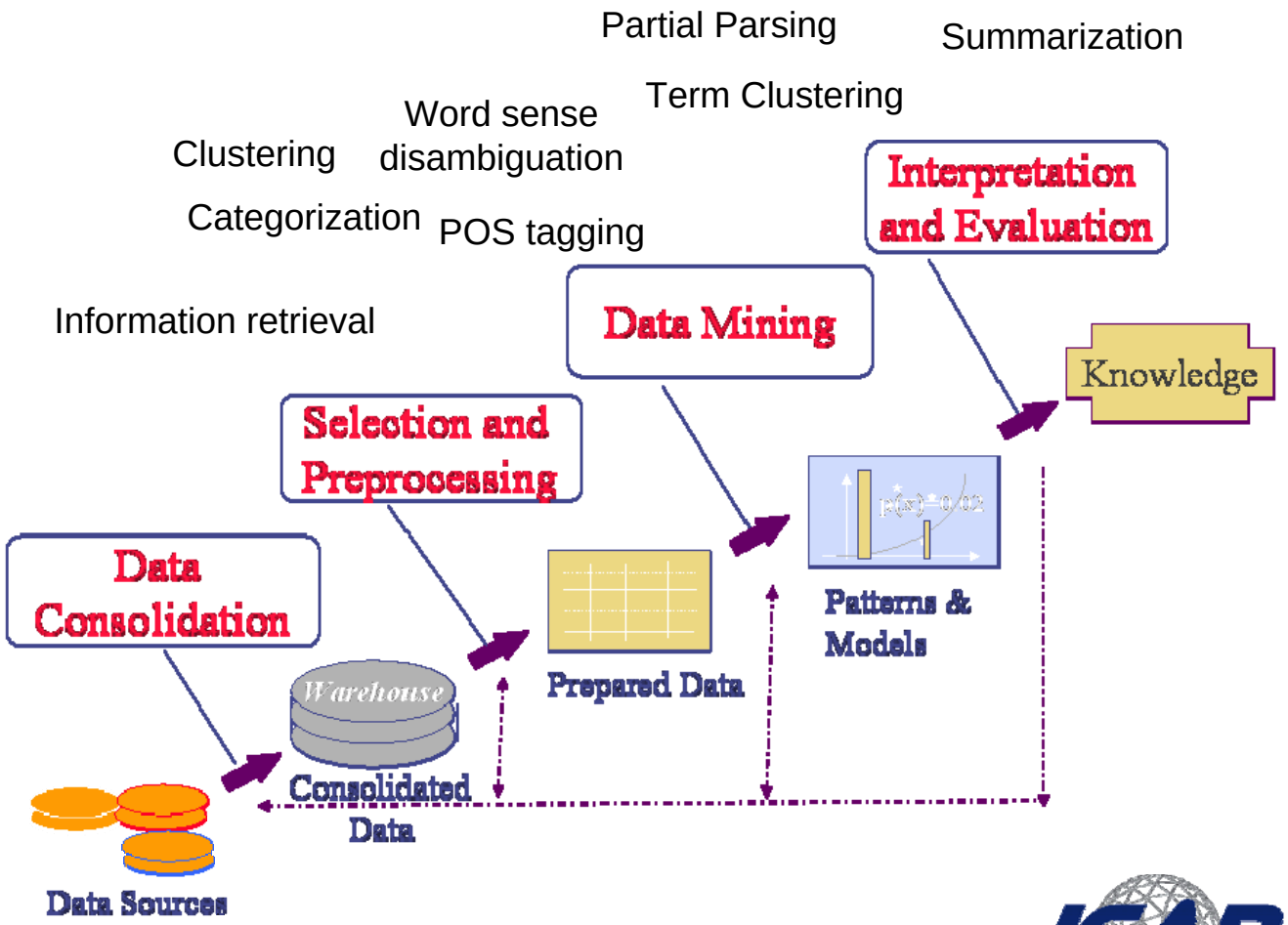
Problems with textual data (II)

- But there are new problems
  - Text is not designed to be used by computers
  - Complex and poorly defined structure and semantics
  - But much harder, ambiguity
    - In speech, morphology, syntax, semantics, pragmatics
      - Plan (pianta, piano)
      - vehicle, car
  - Multilingualism
    - Lack of reliable and general translation tools

The KDD process



The KDD Process specialized for Text Data



Real Text Mining Example: Don Swanson's Medical Work (1991)

- Given
  - medical titles and abstracts
  - a problem (incurable rare disease)
  - some medical expertise
- find causal links among titles
  - symptoms
  - drugs
  - diseases
- E.g.: Magnesium deficiency related to migraine
  - This was found by extracting features from medical literature on migraines and nutrition

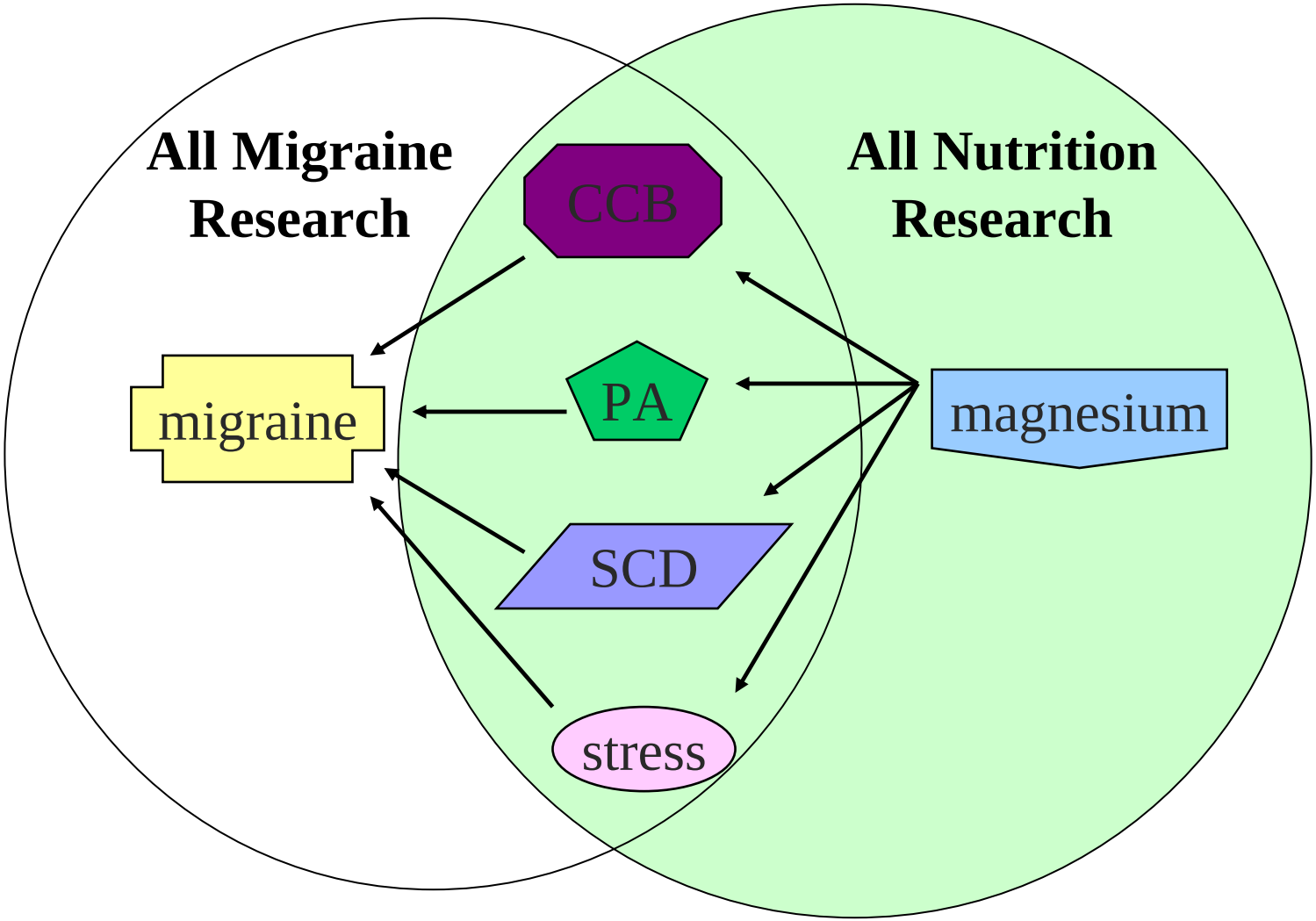
Swanson Example

- Results for Migraine headaches
  - Stress is associated with migraines;
  - Stress can lead to a loss of magnesium;
  - calcium channel blockers (CCB) prevent some migraines
  - Magnesium is a natural calcium channel blocker;
  - Spreading cortical depression (SCD) is implicated in some migraines;
  - High levels of magnesium inhibit SCD;
  - Migraine patients have high platelet aggregability (PA);
  - Magnesium can suppress platelet aggregability.
- All extracted from medical journal titles

Swanson's TDM

- Two of his hypotheses have received some experimental verification.
- His technique
  - Only partially automated
  - Required medical expertise
- Few people are working on this kind of information aggregation problem.

Gathering Evidence



Basics from Information Retrieval

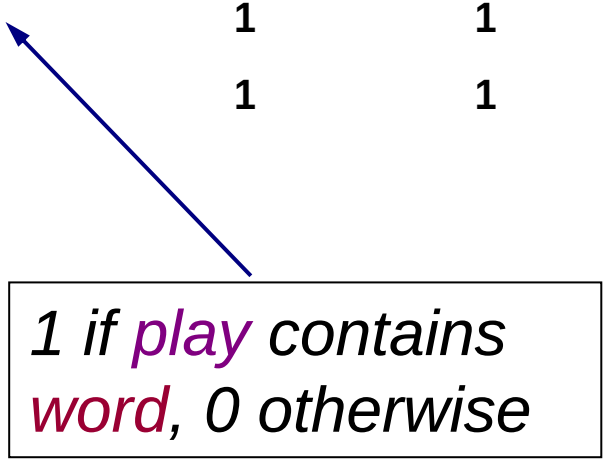
ICAR CNR
Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

The starting point: querying text

- Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calpurnia*?
- Could grep all of Shakespeare's plays for *Brutus* and *Caesar* then strip out lines containing *Calpurnia*?
  - Slow (for large corpora)
  - *NOT* is non-trivial
  - Other operations (e.g., find the phrase *Romans and countrymen*) not feasible

Term-document incidence

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



Incidence vectors

- We have a 0/1 vector for each document
- To answer query:
  - take the vectors for *Brutus, Caesar* and *Calpurnia* (complemented)
  - Compute bitwise *AND* over them all
    - 110100 *AND* 110111 *AND* 101111 = 100100

Bigger corpora

- Example
  - 1M documents, each with about 1K terms
    - Avg 6 bytes/term including spaces/punctuation
    - 6GB of data
  - Say there are *m* = 500K *distinct* terms among these
- Term-doc matrix
  - 500K x 1M matrix has half-a-trillion 0's and 1's.
  - But it has no more than one billion 1's (WHY?)
    - The matrix is extremely sparse
- What's a better representation?

Inverted Index

- Stores the associations of terms with documents
  - Dictionary: gathers all (releavant) index terms
  - Posting lists: for each term, a list of the documents it occurs within

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

Doc 1

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

Doc 2

Term	N docs	Tot 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

Inverted index construction

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

Doc 1

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

Doc 2

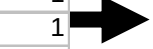
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

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

Inverted index construction (II)

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

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

Inverted index construction (III)

- Multiple term entries in a single document are merged and frequency information 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 #	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

Inverted index construction (IV)

- The file is commonly split into a *Dictionary* and a *Postings* file

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	Tot 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

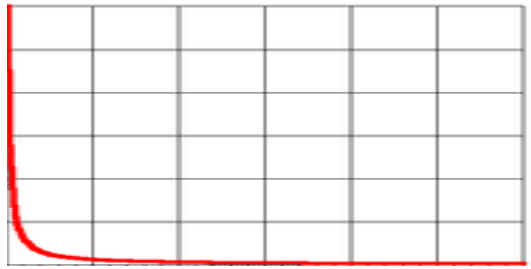
Issues with index we just built

- How do we process a query?
- What terms in a doc do we index?
  - All words or only “important” ones?
- Stopword list: terms that are so common that they’re ignored for indexing.
  - *e.g.*, *the, a, an, of, to* …
  - language-specific.

Zipf's law

- The most frequent word will occur approximately twice as often as the second most frequent word

  …. which occurs twice as often as the fourth most frequent word, etc. …

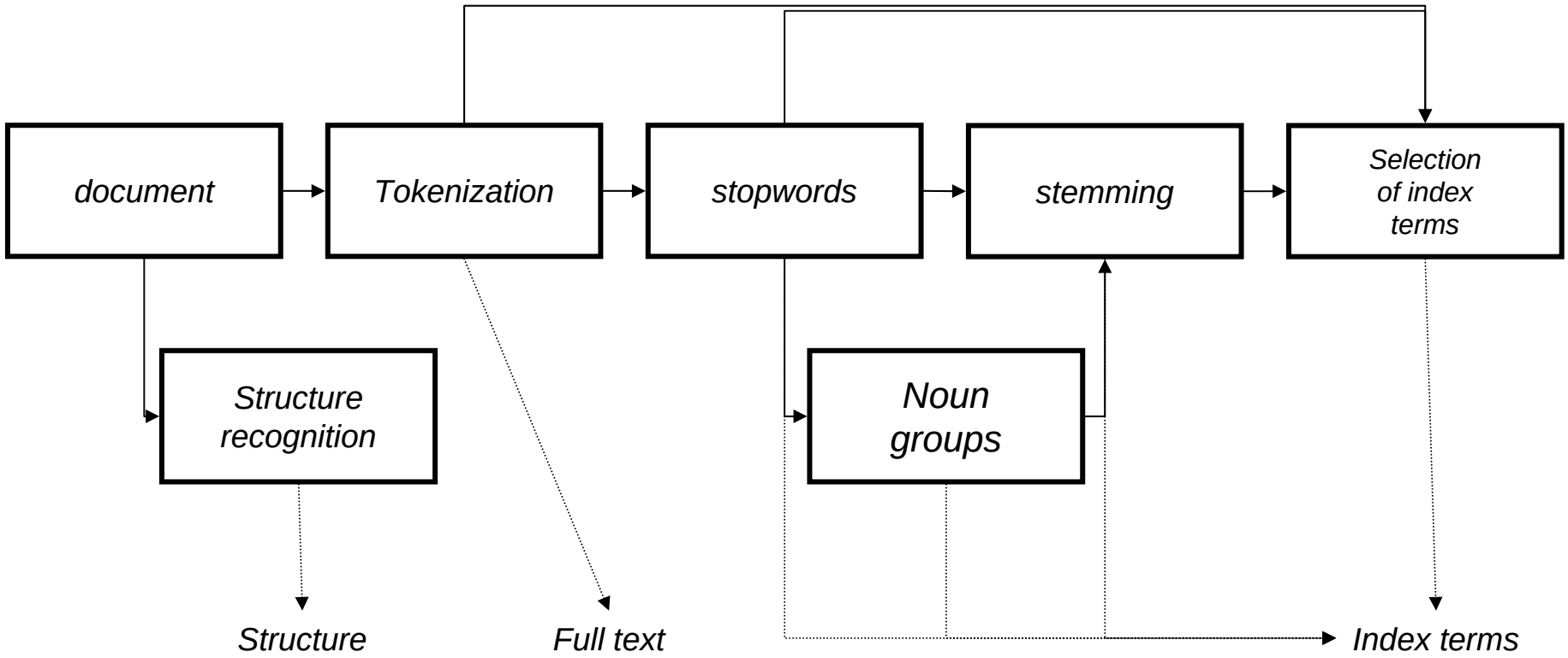


- Consequences
  - High-frequency terms are not so meaningful
    - *stop words*
  - Low-frequency terms a very common
    - They may be specific to the document
    - They may be typos

Some issues to be aware

- Different languages
- Typos
- Syntax
- Grammar
- Zipf’ law

Text processing



Tokenization

- Language dependent
- Identify words (also know as tokens)
- Basic units of text
- Take care of delimiters
- ‘ is a ‘s or a delimiter? What about - ?
- Other elements .,:<>()?!

Stopwords: a small list in English

a	also	an	and	as	at	be	but
by	can	could	do	for	from	go	
have	he	her	here	his	how		
i	if	in	into	it	its		
my	of	on	or	our	say	she	
that	the	their	there	therefore	they		
this	these	those	through	to	until		
we	what	when	where	which	while	who	with
would	you	your					

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*

Stemming

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

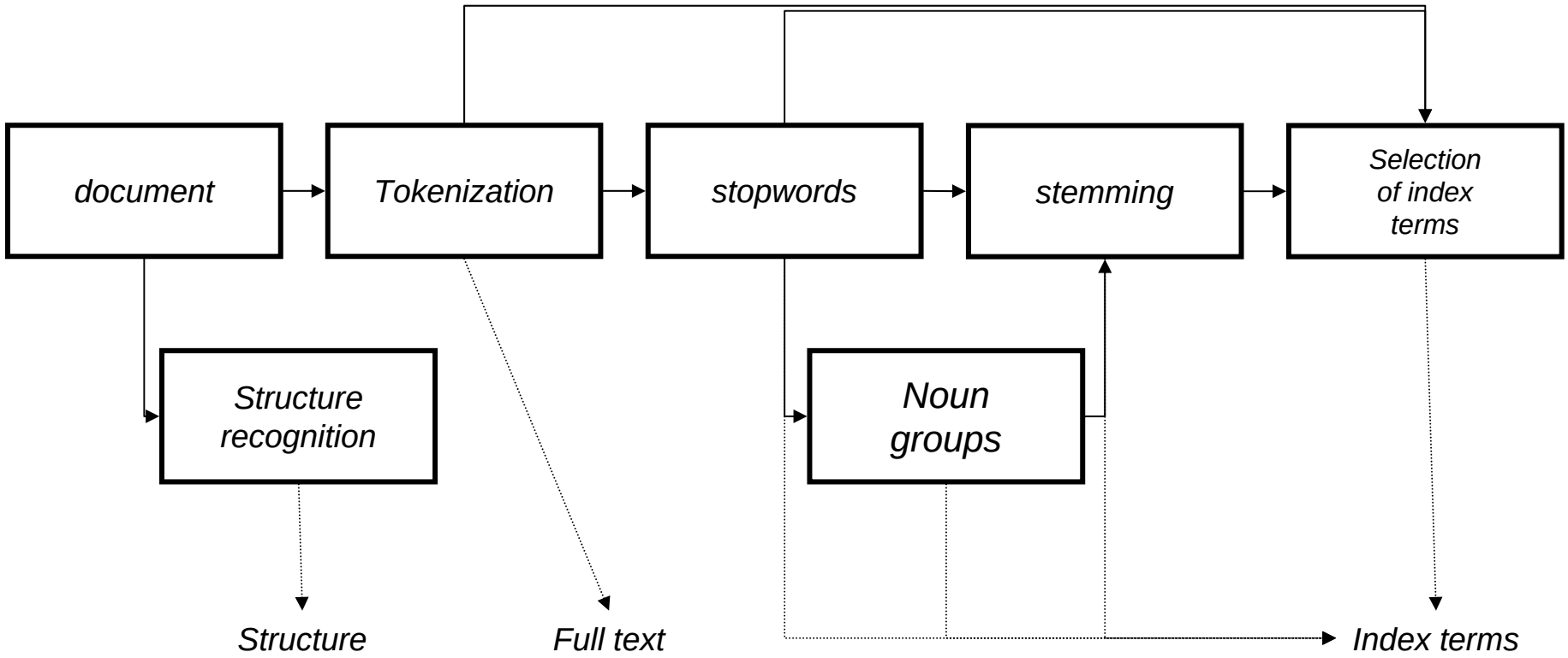
for example compressed and compression are both accepted as equivalent to compress.

for exampl compres and compres are both accept as equival to compres.

Exercise

- Stem the following words
  - Automobile
  - Automotive
  - Cars
  - Information
  - Informative

Summary of text processing



Boolean model: Exact match

- An algebra of queries using AND, OR and NOT together with query words
  - What we used in examples in the first class
  - Uses “set of words” document representation
  - Precise: document matches condition or not
- Primary commercial retrieval tool for 3 decades
  - Researchers had long argued superiority of ranked IR systems, but not much used in practice until spread of web search engines
  - Professional searchers still like boolean queries: you know exactly what you’re getting
    - Cf. Google’s boolean AND criterion

Boolean Models – Problems

- Very rigid: AND means all; OR means any.
- Difficult to express complex user requests.
- Difficult to control the number of documents retrieved.
  - *All* matched documents will be returned.
- Difficult to rank output.
  - *All* matched documents logically satisfy the query.
- Difficult to perform relevance feedback.
  - If a document is identified by the user as relevant or irrelevant, how should the query be modified?

Evidence accumulation

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

Relevance Ranking: Binary term presence matrices

- Record whether a document contains a word: document is binary vector in $\{0,1\}^v$
  - What we have mainly assumed so far
- Idea: Query satisfaction = overlap measure:

$$|X \cap Y|$$

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

Overlap matching

- What are the problems with the overlap measure?
- It doesn't consider:
  - Term frequency in document
  - Term scarcity in collection (document mention frequency)
  - Length of documents

Overlap matching

- One can normalize in different ways:
  - Jaccard coefficient:

$$|X \cap Y| / |X \cup Y|$$

  - Cosine measure:

$$|X \cap Y| / \sqrt{|X| \times |Y|}$$

- What documents would score best using Jaccard against a typical query?
  - Does the cosine measure fix this problem?

Count term-document matrices

- We haven't considered frequency of a word
- Count of a word in a document:
  - Bag of words model
  - Document is a vector in $\mathbb{N}^v$

	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

Weighting term frequency: tf

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear: but it seems that more is better, but a lot isn't necessarily better than a few
  - Can just use raw score
  - Another option commonly used in practice:

$$tf_{t,d} > 0 \; ? \; 1 + \log tf_{t,d} : 0$$

Dot product matching

- Match is dot product of query and document

$$q \cdot d = \sum_i tf_{i,q} \times tf_{i,d}$$

- [Note: 0 if orthogonal (no words in common)]
- Rank by match

- It still doesn't consider:
  - Term scarcity in collection (document mention frequency)
  - Length of documents and queries
    - Not normalized

Weighting should depend on the term overall

- Which of these tells you more about a doc?
  - 10 occurrences of *hernia*?
  - 10 occurrences of *the*?
- Suggest looking at collection frequency (cf)
- But document frequency (df) may be better:

Word	cf	df
try	10422	8760
insurance	10440	3997

- Document frequency weighting is only possible in known (static) collection.

tf x idf term weights

- tf x idf measure combines:
  - term frequency (tf)
    - measure of term density in a doc
  - inverse document frequency (idf)
    - measure of informativeness of term: its rarity across the whole corpus
    - could just be raw count of number of documents the term occurs in ($idf_i = 1/df_i$)
    - but by far the most commonly used version is:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

Summary: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term *i* in each document *d*

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$tf_{i,d}$ = frequency of term *i* in document *j*

n = total number of documents

df_i = the number of documents that contain term *i*

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

Real-valued term-document matrices

- Function (scaling) of count of a word in a document:
  - Bag of words model
  - Each is a vector in $\mathbb{R}^V$
  - Here log scaled tf.idf

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

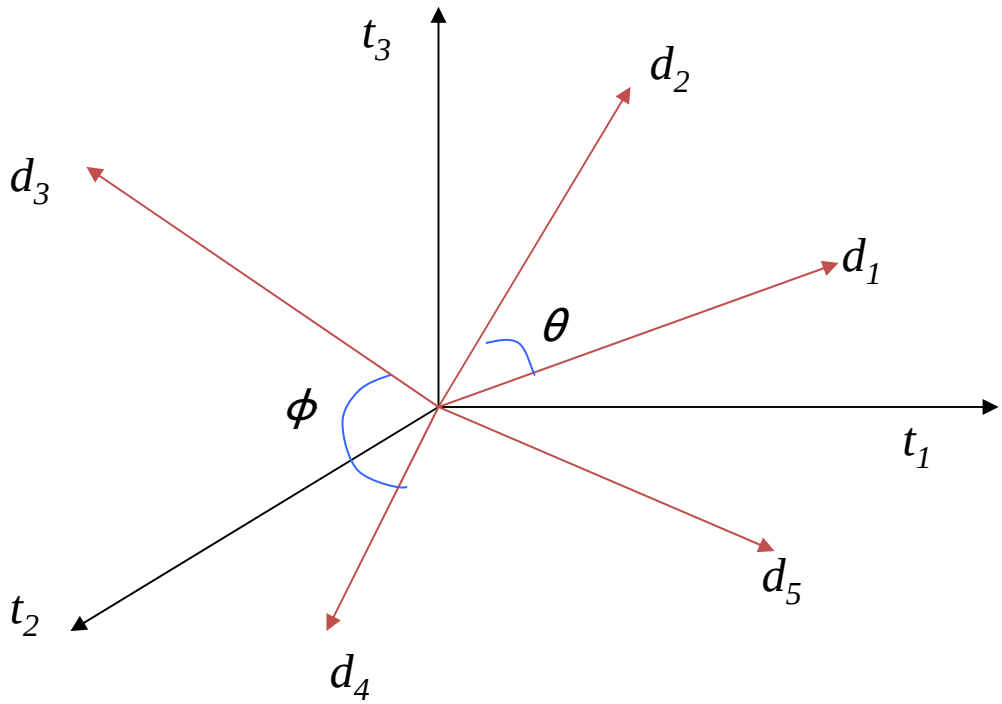
Documents as vectors

- Each doc *j* can now be viewed as a vector of *tf*×*idf* values, one component for each term
- So we have a vector space
  - terms are axes
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions
- (The corpus of documents gives us a matrix, which we could also view as a vector space in which words live – transposable data)

Why turn docs into vectors?

- First application: Query-by-example
  - Given a doc *d*, find others “like” it.
  - Now that *d* is a vector, find vectors (docs) “near” it.
- Higher-level applications: clustering, classification

Intuition



Postulate: Documents that are "close together" in vector space talk about the same things.

The vector space model

Query as vector:

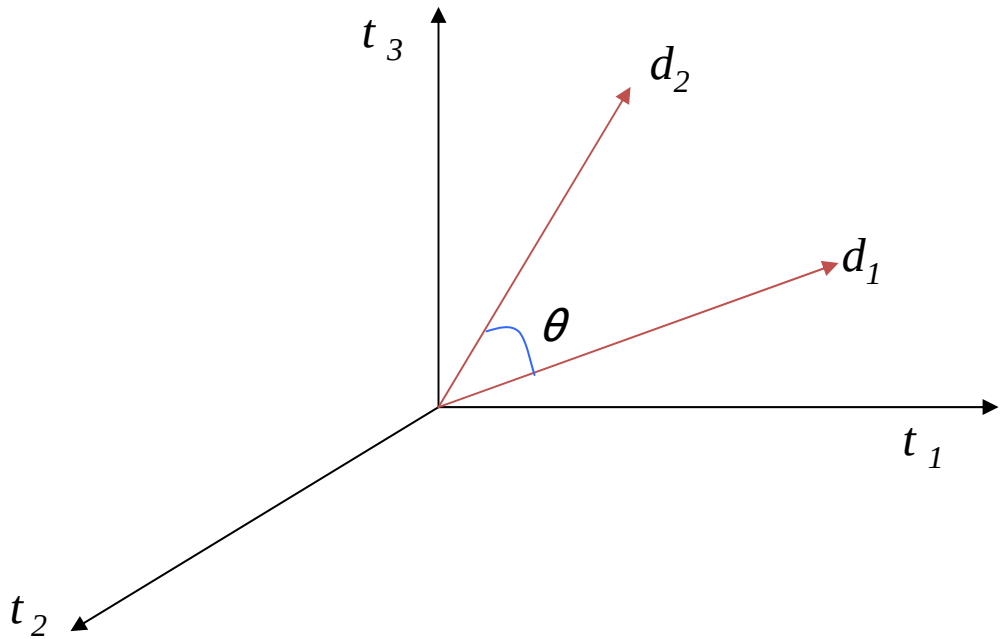
- We regard query as short document
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector.

How to measure proximity

- Euclidean distance
  - Distance between vectors d_1 and d_2 is the length of the vector $|d_1 - d_2|$.
  - Why is this not a great idea?
- We still haven't dealt with the issue of length normalization
  - Long documents would be more similar to each other by virtue of length, not topic
- However, we can implicitly normalize by looking at *angles* instead

Cosine similarity

- Distance between vectors d_1 and d_2 *captured* by the cosine of the angle *x* between them.
- Note – this is *similarity*, not distance



Cosine similarity

$$sim(d_j, d_k) = \frac{d_j \cdot d_k}{|d_j||d_k|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

- Cosine of angle between two vectors
- The denominator involves the lengths of the vectors
- So the cosine measure is also known as the *normalized inner product*

$$\text{Length } |d_j| = \sqrt{\sum_{i=1}^{n} w_{i,j}^2}$$

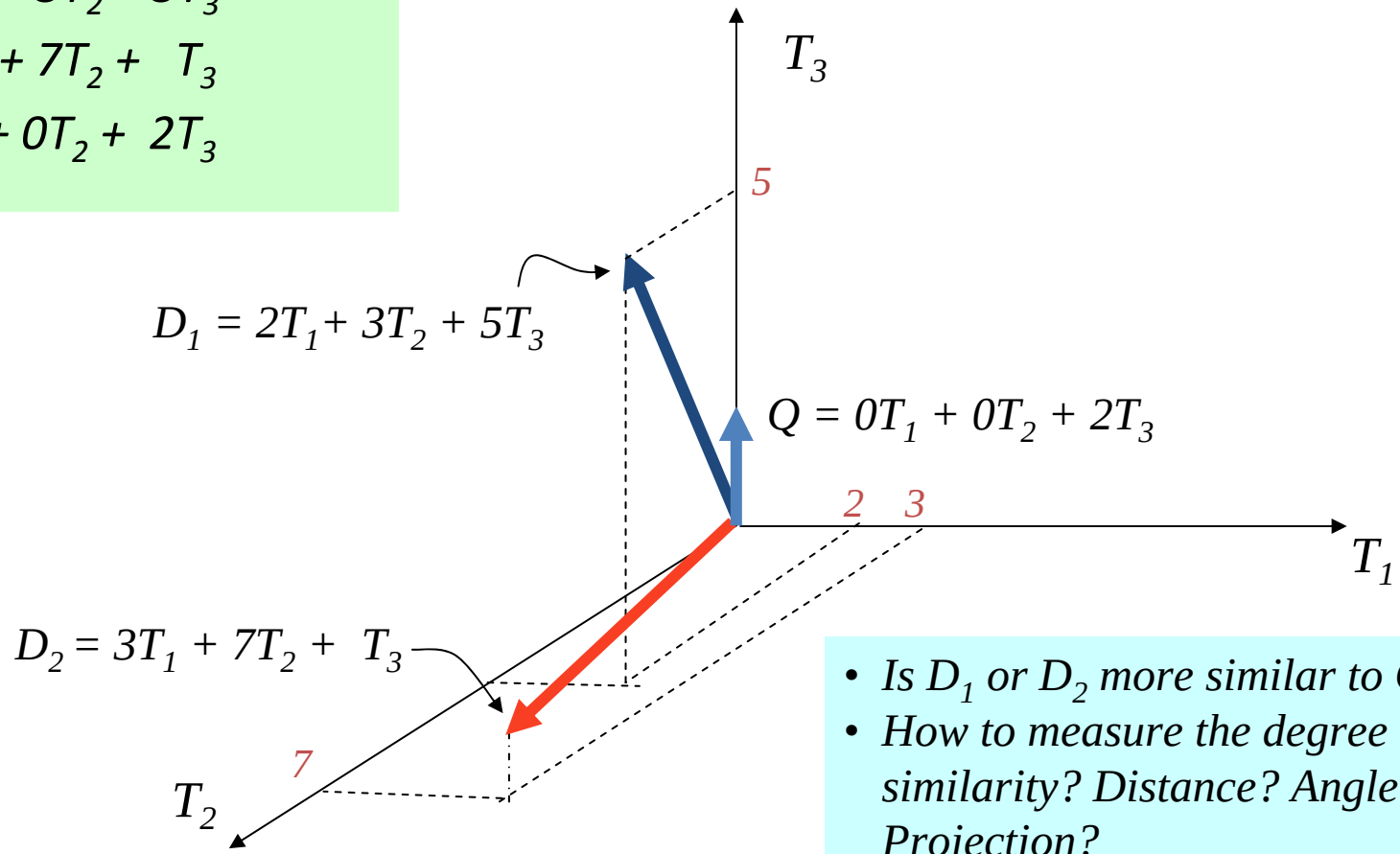
Graphic Representation

Example:

$D_1 = 2T_1 + 3T_2 + 5T_3$

$D_2 = 3T_1 + 7T_2 + T_3$

$Q = 0T_1 + 0T_2 + 2T_3$



- *Is D_1 or D_2 more similar to Q?*
- *How to measure the degree of similarity? Distance? Angle? Projection?*

Cosine similarity exercises

- *Exercise: Rank the following by decreasing cosine similarity:*
  - Two docs that have only frequent words *(the, a, an, of)* in common.
  - Two docs that have no words in common.
  - Two docs that have many rare words in common *(wingspan, tailfin).*

Normalized vectors

- A vector can be normalized (given a length of 1) by dividing each of its components by the vector's length
- This maps vectors onto the unit circle:
- Then, $\left|d_j\right| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1$
- Longer documents don't get more weight
- For normalized vectors, the cosine is simply the dot product:

$$\cos(d_j, d_k) = d_j \cdot d_k$$

Example

- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights*

	SaS	PaP	WH
affection	**115**	**58**	**20**
jealous	**10**	**7**	**11**
gossip	**2**	**0**	**6**

	SaS	PaP	WH
affection	**0.996**	**0.993**	**0.847**
jealous	**0.087**	**0.120**	**0.466**
gossip	**0.017**	**0.000**	**0.254**

- cos(SAS, PAP) = .996 x .993 + .087 x .120 + .017 x 0.0 = 0.999
- cos(SAS, WH) = .996 x .847 + .087 x .466 + .017 x .254 = 0.929

Summary of vector space model

- Docs and queries are modelled as vectors
  - Key: A user's query is a short document
  - We can measure doc's proximity to the query
- Natural measure of scores/ranking – no longer Boolean.
- Provides partial matching and ranked results.
- Allows efficient implementation for large document collections

Problems with Vector Space Model

- Missing semantic information (e.g. word sense).
- Missing syntactic information (e.g. phrase structure, word order, proximity information).
- Assumption of term independence (e.g. ignores synonomy).
- Lacks the control of a Boolean model (e.g., *requiring* a term to appear in a document).
  - Given a two-term query "A B", may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.

Clustering documents

Text Clustering

- Term clustering
  - Query expansion
  - Thesaurus construction
- Document clustering
  - Topic maps
  - Clustering of retrieval results

Why cluster documents?

- For improving recall in search applications
- For speeding up vector space retrieval
- Corpus analysis/navigation
  - Sense disambiguation in search results

Improving search recall (automatic query expansion)

- *Cluster hypothesis* - Documents with similar text are related
- Ergo, to improve search recall:
  - Cluster docs in corpus a priori
  - When a query matches a doc *D*, also return other docs in the cluster containing *D*
- Hope: docs containing *automobile* returned on a query for *car* because
  - clustering grouped together docs containing *car* with those containing *automobile.*

Speeding up vector space retrieval

- In vector space retrieval, must find nearest doc vectors to query vector
  - This would entail finding the similarity of the query to every doc - slow!
- By clustering docs in corpus a priori
  - find nearest docs in cluster(s) close to query
  - inexact but avoids exhaustive similarity computation

Corpus analysis/navigation

- Partition a corpus it into groups of related docs
  - Recursively, can induce a tree of topics
  - Allows user to browse through corpus to home in on information
  - Crucial need: meaningful labels for topic nodes

Navigating search results

- Given the results of a search (say *jaguar*), partition into groups of related docs
  - sense disambiguation
  - See for instance vivisimo.com

- *Cluster 1:*
  - *Jaguar Motor Cars' home page*
  - *Mike's XJS resource page*
  - *Vermont Jaguar owners' club*
- *Cluster 2:*
  - *Big cats*
  - *My summer safari trip*
  - *Pictures of jaguars, leopards and lions*
- *Cluster 3:*
  - *Jacksonville Jaguars' Home Page*
  - *AFC East Football Teams*

What makes docs “related”?

- Ideal: semantic similarity.
- Practical: statistical similarity
  - We will use cosine similarity.
  - Docs as vectors.
  - For many algorithms, easier to think in terms of a *distance* (rather than similarity) between docs.
  - We will describe algorithms in terms of cosine similarity.

Recall: doc as vector

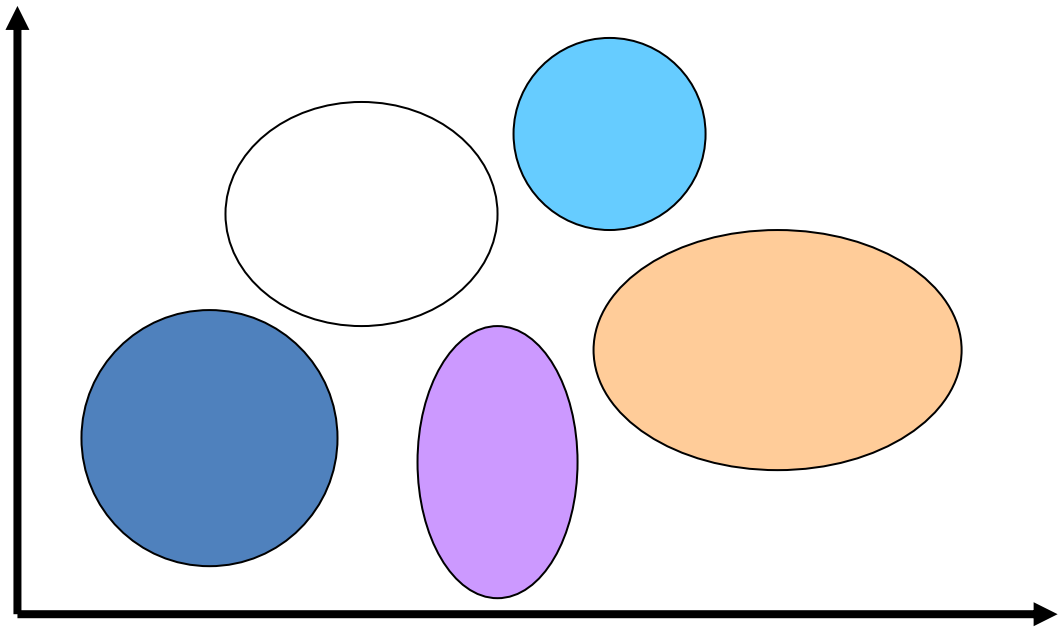
- Each doc *j* is a vector of *tf*×*idf* values, one component for each term.
- Can normalize to unit length.
- So we have a vector space
  - terms are axes - aka *features*
  - *n* docs live in this space
  - even with stemming, may have 10000+ dimensions
  - do we really want to use all terms?

Two flavors of clustering

- Given *n* docs and a positive integer *k*, partition docs into *k* (disjoint) subsets.
- Given docs, partition into an "appropriate" number of subsets.
  - E.g., for query results - ideal value of *k* not known up front - though UI may impose limits.
- Can usually take an algorithm for one flavor and convert to the other.

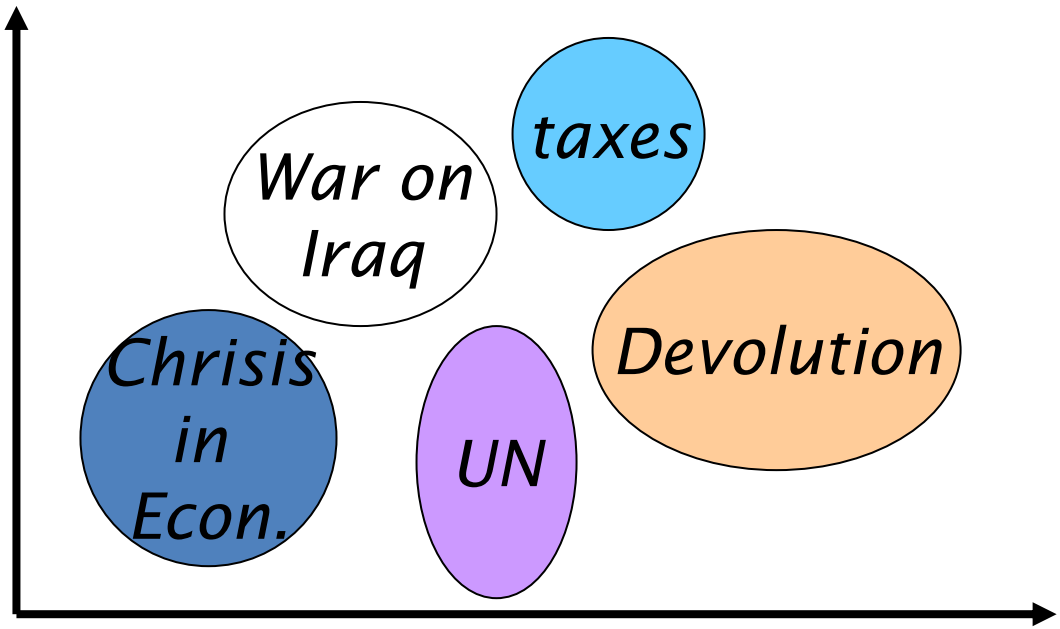
Thought experiment

- Consider clustering a large set of politics documents
  - what do you expect to see in the vector space?



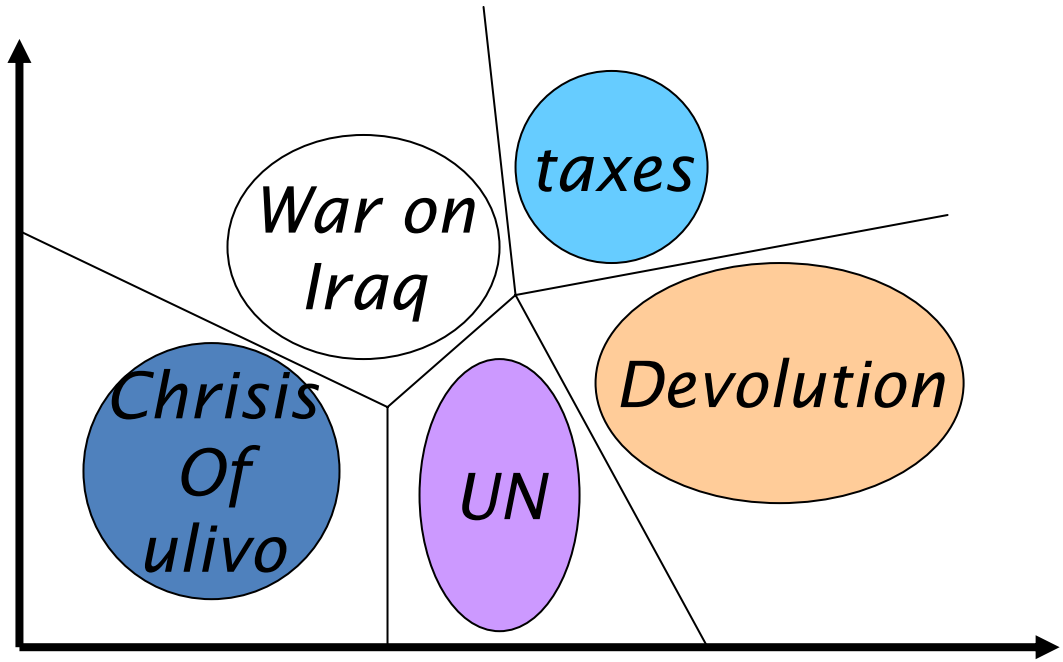
Thought experiment

- Consider clustering a large set of politics documents
  - what do you expect to see in the vector space?



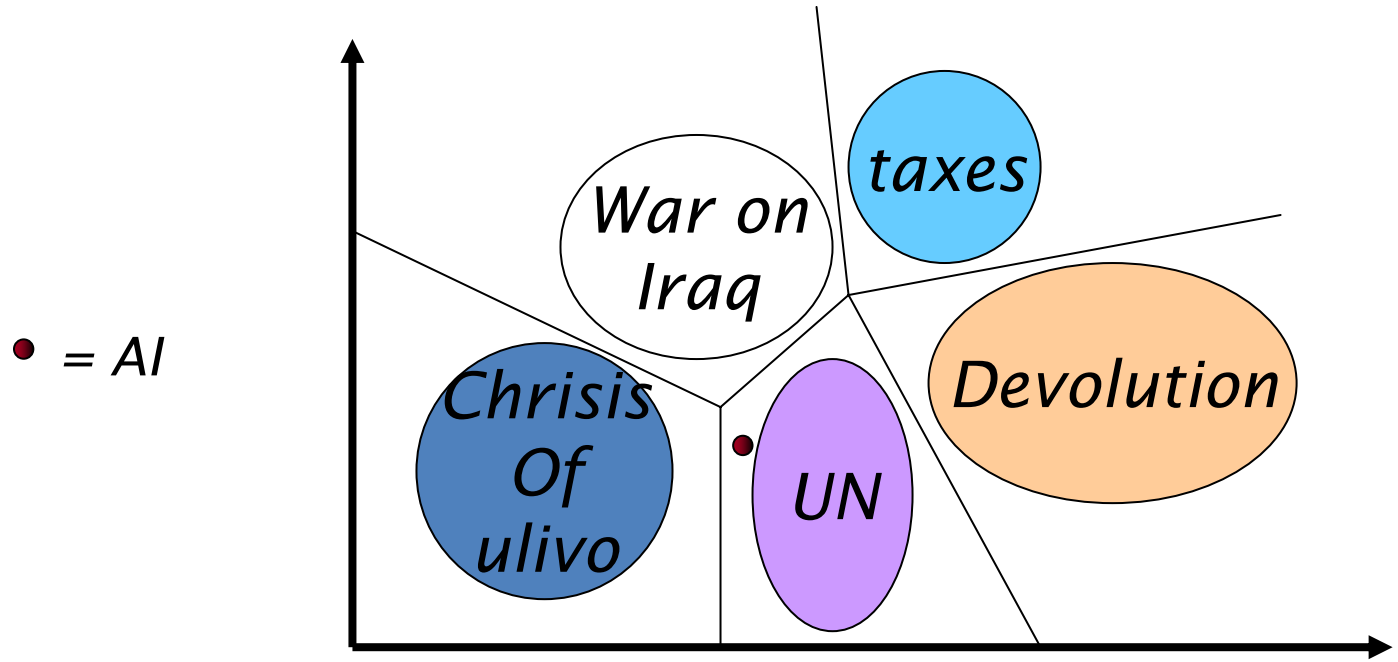
Decision boundaries

- Could we use these blobs to infer the subject of a new document?



Deciding what a new doc is about

- Check which region the new doc falls into
  - can output “softer” decisions as well.



Setup

- Given “training” docs for each category
  - Devolution, UN, War on Iraq, etc.
- Cast them into a decision space
  - generally a vector space with each doc viewed as a bag of words
- Build a classifier that will classify new docs
  - Essentially, partition the decision space
- Given a new doc, figure out which partition it falls into

Clustering algorithms

- Centroid-Based approaches
- Hierarchical approaches
- Model-based approaches (not considered here)

Key notion: *cluster representative*

- In the algorithms to follow, will generally need a notion of a representative point in a cluster
- Representative should be some sort of "typical" or central point in the cluster, e.g.,
  - smallest squared distances, etc.
  - point that is the "average" of all docs in the cluster
- Need not be a document

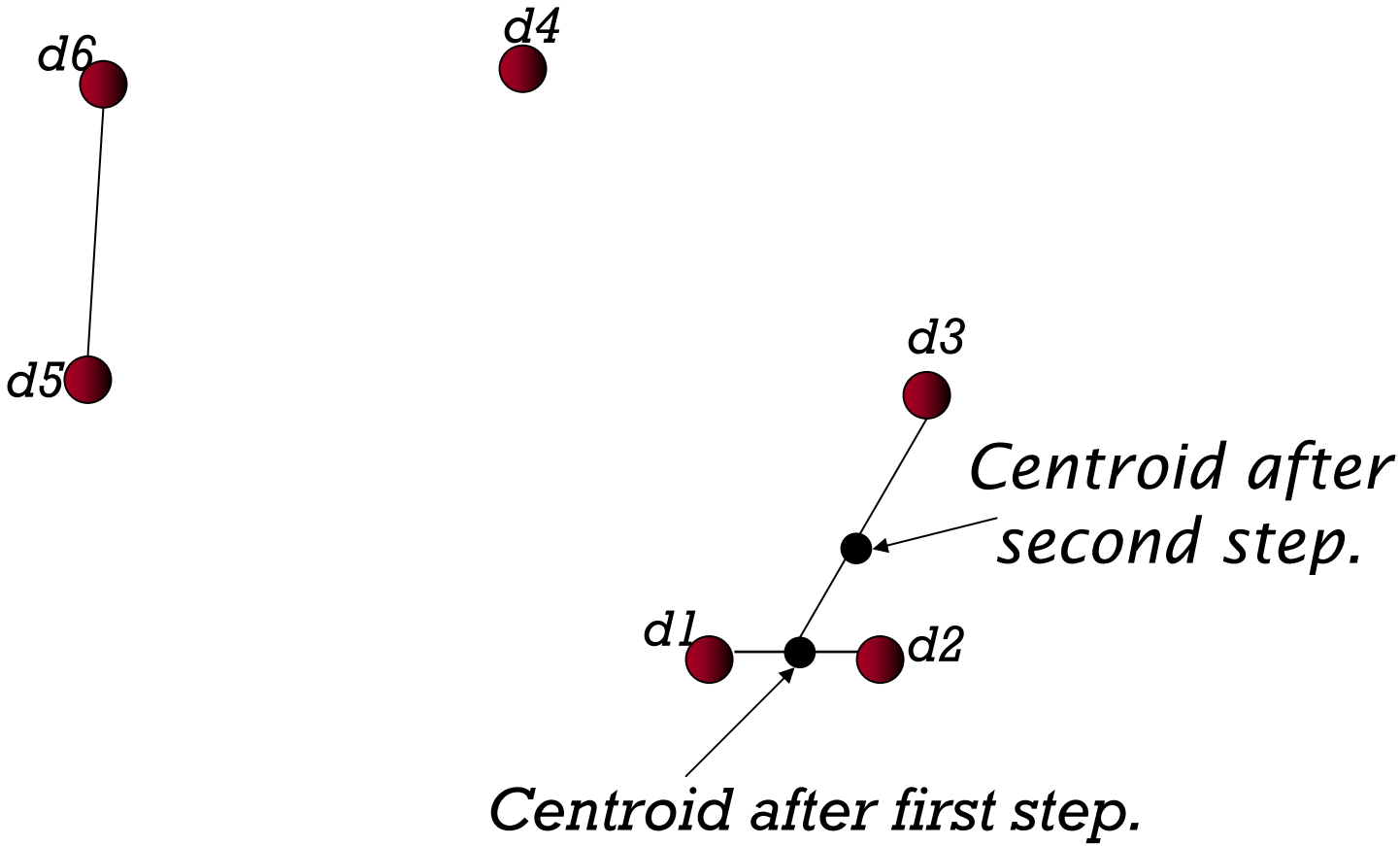
Key notion: cluster centroid

- Centroid of a cluster = component-wise average of vectors in a cluster - is a vector.
  - Need not be a doc.
- Centroid of (1,2,3); (4,5,6); (7,2,6) is (4,3,5).

Agglomerative clustering

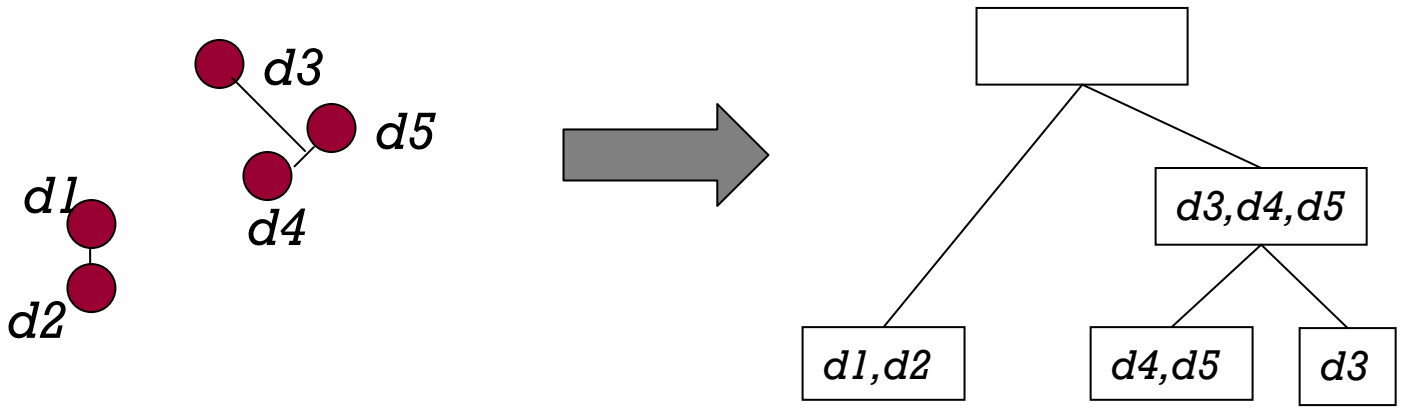
- Given target number of clusters *k*.
- Initially, each doc viewed as a cluster
  - start with *n* clusters;
- Repeat:
  - while there are > *k* clusters, find the "closest pair" of clusters and merge them
- Many variants to defining ***closest pair*** of clusters
  - Clusters whose centroids are the most cosine-similar
  - … whose "closest" points are the most cosine-similar
  - … whose "furthest" points are the most cosine-similar

Example: *n=6, k=3,* closest pair of centroids



Hierarchical clustering

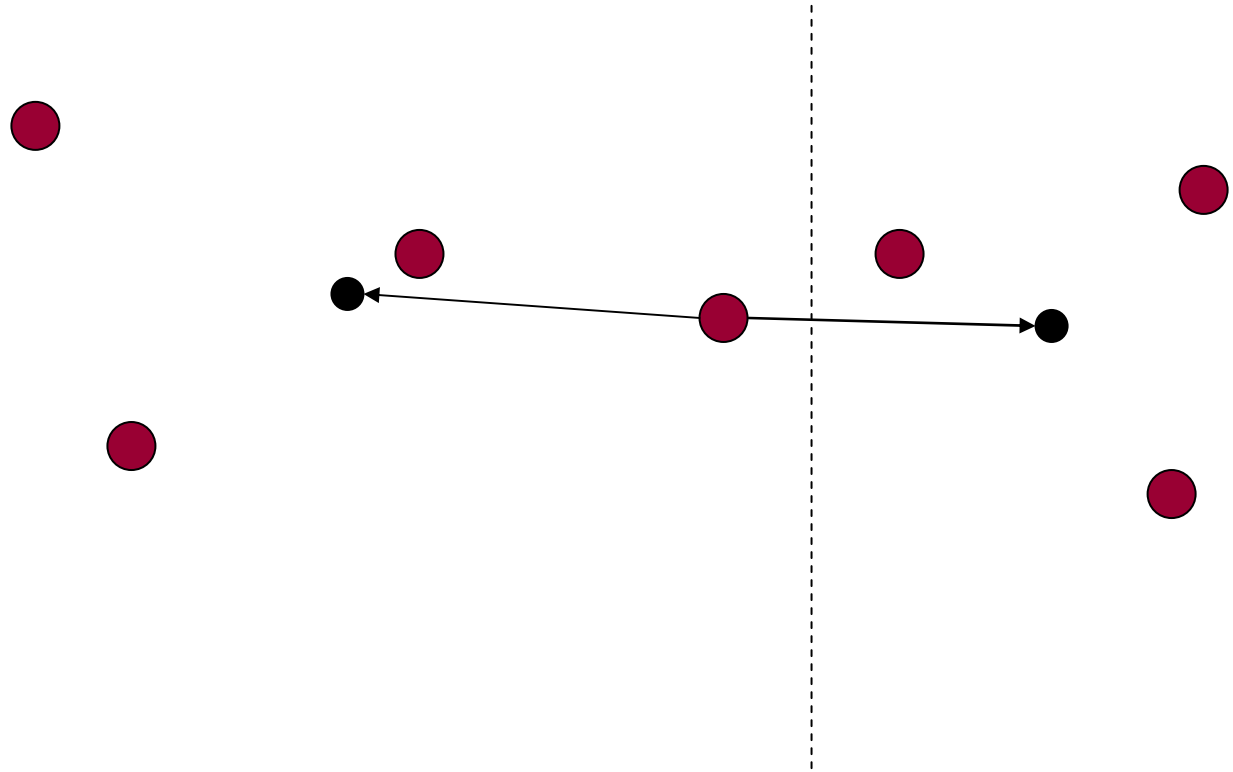
- As clusters *agglomerate*, docs likely to fall into a hierarchy of “topics” or concepts.



Different algorithm: *k*-means

- Given *k* - the number of clusters desired.
- Basic scheme:
  - At the start of the iteration, we have *k* centroids.
  - Each doc assigned to the nearest centroid.
  - All docs assigned to the same centroid are averaged to compute a new centroid;
    - thus have *k* new centroids.
- More locality within each iteration.
- Hard to get good bounds on the number of iterations.

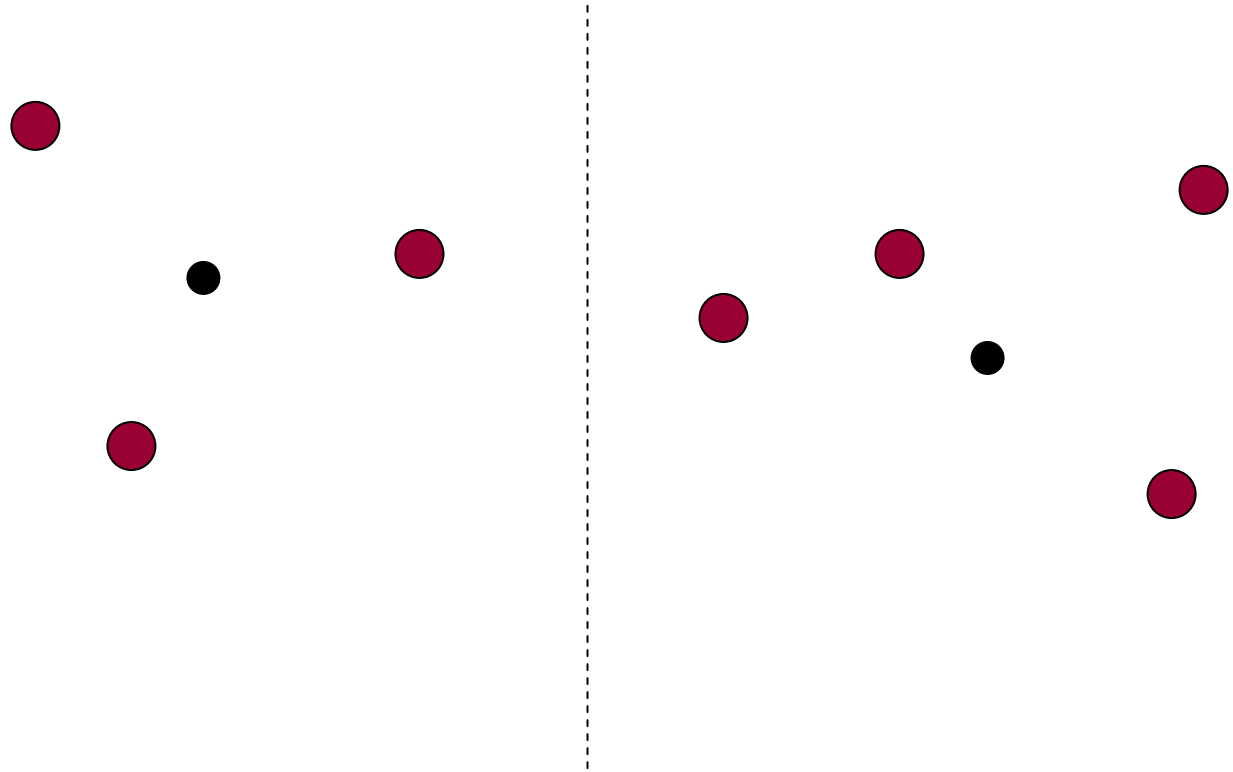
Iteration example



● *Docs*

● *Current centroids*

Iteration example



● *Docs*

● *New centroids*

k-means clustering

- Begin with *k* docs as centroids
  - could be any *k* docs, but *k* random docs are better.
- Repeat the Basic Scheme until some termination condition is satisfied, e.g.:
  - A fixed number of iterations.
  - Doc partition unchanged.
  - Centroid positions don't change

Text clustering:
More issues/applications

List of issues/applications

- Term vs. document space clustering
- Multi-lingual docs
- Feature selection
- Clustering to speed-up scoring
- Building navigation structures
  - "Automatic taxonomy induction"
- Labeling

Term vs. document space

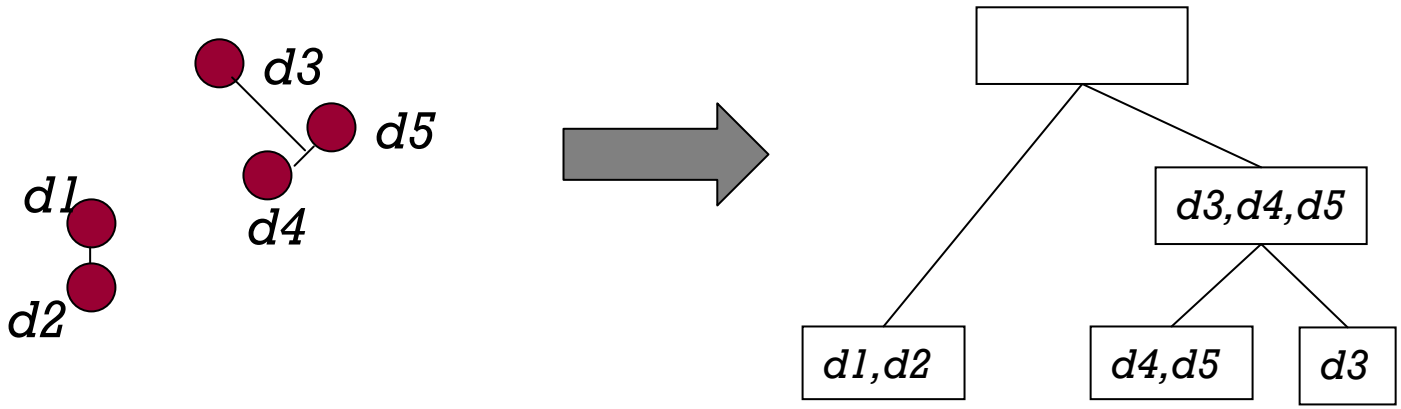
- Thus far, we clustered docs based on their similarities in terms space
- For some applications, e.g., topic analysis for inducing navigation structures, can “dualize”:
  - use docs as axes
  - represent (some) terms as vectors
  - proximity based on co-occurrence of terms in docs
  - now clustering terms, *not* docs

Term Clustering

- Clustering of words or phrases based on the document texts in which they occur
  - Identify term relationships
  - Assumption: words that are contextually related (i.e., often co-occur in the same sentence/paragraph/document) are semantically related and hence should be put in the same class
- General process
  - Selection of the document set and the dictionary
    - Term by document matrix
  - Computation of association or similarity matrix
  - Clustering of highly related terms
- Applications
  - Query expansion
  - Thesaurus constructions

Navigation structure

- Given a corpus, agglomerate into a hierarchy
- Throw away lower layers so you don't have *n* leaf topics each having a single doc



Major issue - labeling

- After clustering algorithm finds clusters - how can they be useful to the end user?
- Need label for each cluster
  - In search results, say “Football” or “Car” in the *jaguar* example.
  - In topic trees, need navigational cues.

How to Label Clusters

- Show titles of typical documents
  - Titles are easy to scan
  - Authors create them for quick scanning!
  - But you can only show a few titles which may not fully represent cluster
- Show words/phrases prominent in cluster
  - More likely to fully represent cluster
  - Use distinguishing words/phrases
  - But harder to scan

Labeling

- Common heuristics - list 5-10 most frequent terms in the centroid vector.
  - Drop stop-words; stem.
- Differential labeling by frequent terms
  - Within the cluster “Computers”, child clusters all have the word *computer* as frequent terms.

Clustering as dimensionality reduction

- Clustering can be viewed as a form of data compression
  - the given data is recast as consisting of a “small” number of clusters
  - each cluster typified by its representative “centroid”
- Recall LSI
  - extracts “principal components” of data
    - attributes that best explain segmentation
  - ignores features of either
    - low statistical presence, or
    - low discriminating power

Feature selection

- Which terms to use as axes for vector space?
- IDF is a form of feature selection
  - can exaggerate noise e.g., mis-spellings
- Pseudo-linguistic heuristics, e.g.,
  - drop stop-words
  - stemming/lemmatization
  - use only nouns/noun phrases
- Good clustering should “figure out” some of these

Text Categorization

Is this spam?

From: "" <takworlld@hotmail.com>
Subject: real estate is the only way... gem oalvgkay

Anyone can buy real estate with no money down

Stop paying rent TODAY !

There is no need to spend hundreds or even thousands for similar courses

I am 22 years old and I have already purchased 6 properties using the methods outlined in this truly INCREDIBLE ebook.

Change your life NOW !

==========
Click Below to order:
http://www.wholesaledaily.com/sales/nmd.htm
==========

Categorization/Classification

- Given:
  - A description of an instance, $x \in X$, where X is the *instance language* or *instance space*.
    - Issue: how to represent text documents.
  - A fixed set of categories:

    $C = \{c_1, c_2, \ldots, c_n\}$
- Determine:
  - The category of x: $c(x) \in C$, where $c(x)$ is a *categorization function* whose domain is X and whose range is C.
    - We want to know how to build categorization functions ("classifiers").

Text Categorization Examples

Assign labels to each document or web-page:

- Labels are most often topics such as Yahoo-categories
  e.g., "finance," "sports," "news>world>asia>business"
- Labels may be genres
  e.g., "editorials" "movie-reviews" "news“
- Labels may be opinion
  e.g., “like”, “hate”, “neutral”
- Labels may be domain-specific binary
  e.g., "interesting-to-me" : "not-interesting-to-me”
  e.g., “spam” : “not-spam”
  e.g., “is a toner cartridge ad” :“isn’t”

Methods

- Supervised learning of document-label assignment function
- Many new systems rely on machine learning
  - k-Nearest Neighbors (simple, powerful)
  - Naive Bayes (simple, common method)
  - Support-vector machines (new, more powerful)
  - … plus many other methods
  - No free lunch: requires hand-classified training data
    - Recent advances: semi-supervised learning

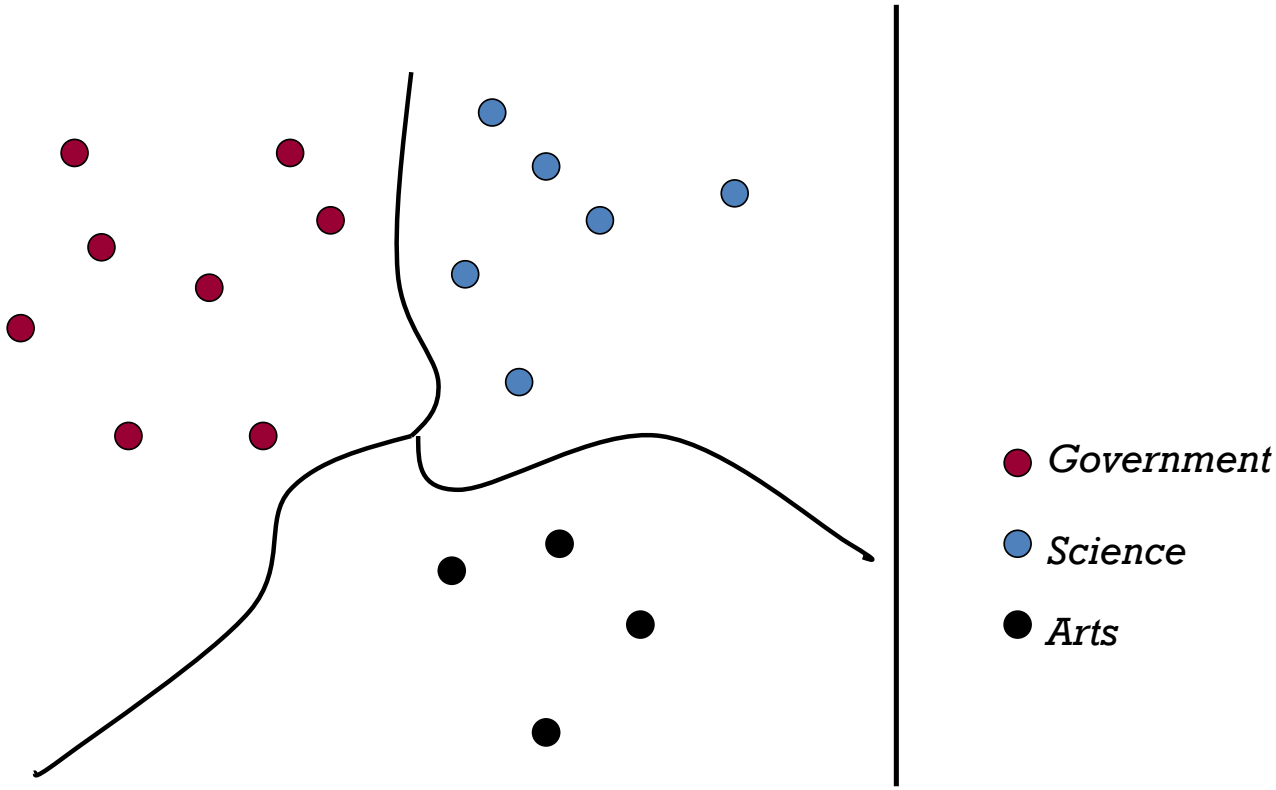
Recall Vector Space Representation

- Each doc *j* is a vector, one component for each term (= word).
- Normalize to unit length.
- Have a vector space
  - terms are axes
  - *n* docs live in this space
  - even with stemming, may have 10000+ dimensions, or even 1,000,000+

Classification Using Vector Spaces

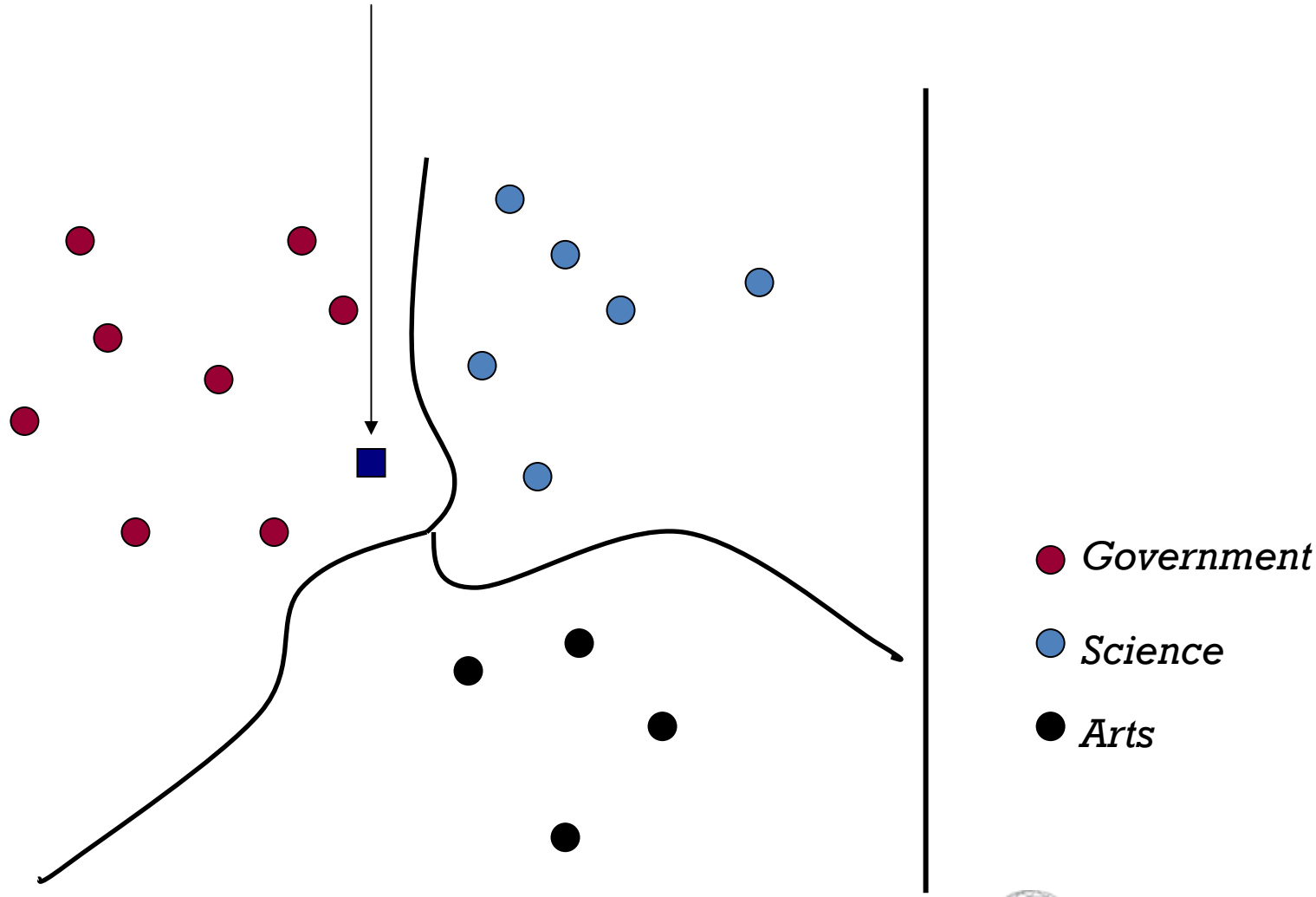
- Each training doc a point (vector) labeled by its topic (= class)
- Hypothesis: docs of the same topic form a contiguous region of space
- Define surfaces to delineate topics in space

Topics in a vector space



- Given a test doc
  - Figure out which region it lies in
  - Assign corresponding class

Test doc = Government



Separating Multiple Topics

- Build a separator between each topic and its complementary set (docs from all other topics).
- Given test doc, evaluate it for membership in each topic.
- Declare membership in topics
  - One-of classification:
    - for class with maximum score/confidence/probability
  - Multiclass classification:
    - For classes above threshold

k Nearest Neighbor Classification

- To classify document d into class c
- Define k-neighborhood N as k nearest neighbors of d
- Count number of documents l in N that belong to c
- Estimate P(c|d) as l/k

kNN: Discussion

- Classification time linear in training set
- Training set generation
  - incompletely judged set can be problematic for multiclass problems
- No feature selection necessary
- Scales well with large number of categories
  - Don't need to train n classifiers for n classes
- Categories can influence each other
  - Small changes to one category can have ripple effect
- Scores can be hard to convert to probabilities
- No training necessary
  - Actually: not true. Why?

Bayesian Methods

- Learning and classification methods based on probability theory.
- Bayes theorem plays a critical role in probabilistic learning and classification.
- Build a *generative model* that approximates how data is produced
- Uses *prior* probability of each category given no information about an item.
- Categorization produces a *posterior* probability distribution over the possible categories given a description of an item.

Feature Selection: Why?

- Text collections have a large number of features
  - 10,000 – 1,000,000 unique words – and more
- Make using a particular classifier feasible
  - Some classifiers can't deal with 100,000s of feat's
- Reduce training time
  - Training time for some methods is quadratic or worse in the number of features (e.g., logistic regression)
- Improve generalization
  - Eliminate noise features
  - Avoid overfitting

Recap: Feature Reduction

- Standard ways of reducing feature space for text
  - Stemming
    - Laugh, laughs, laughing, laughed -> laugh
  - Stop word removal
    - E.g., eliminate all prepositions
  - Conversion to lower case
  - Tokenization
    - Break on all special characters: fire-fighter -> fire, fighter

Feature Selection

- Different selection criteria
  - DF – document frequency
  - IG – information gain
  - MI – mutual information
  - CHI – chi square
- Common strategy
  - Compute statistic for each term
  - Keep n terms with highest value of this statistic