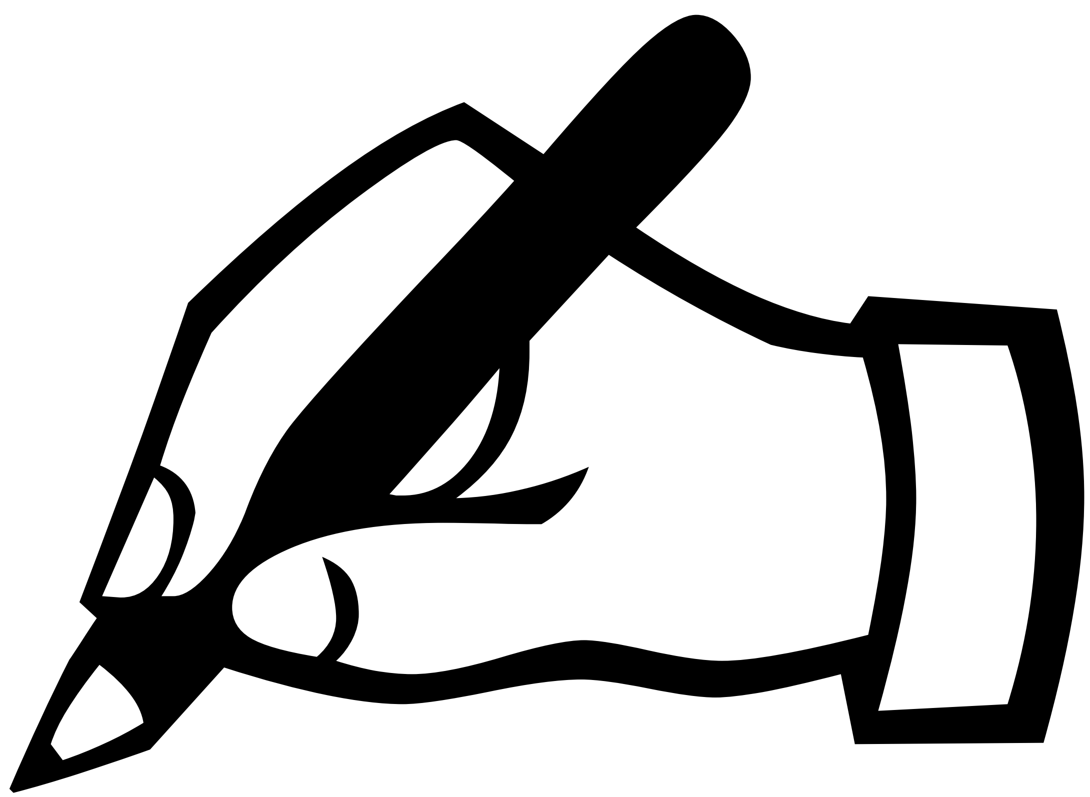
CAMPUS
DE EXCELENCIA
INTERNACIONAL

Text Mining 3
Text Similarity

Madrid Summer School on
Advanced Statistics and Data Mining

Florian Leitner
Data Catalytics, S.L.
leitner@datacatytics.com

Practical:
Floats and defaultdicts



Incentive and applications

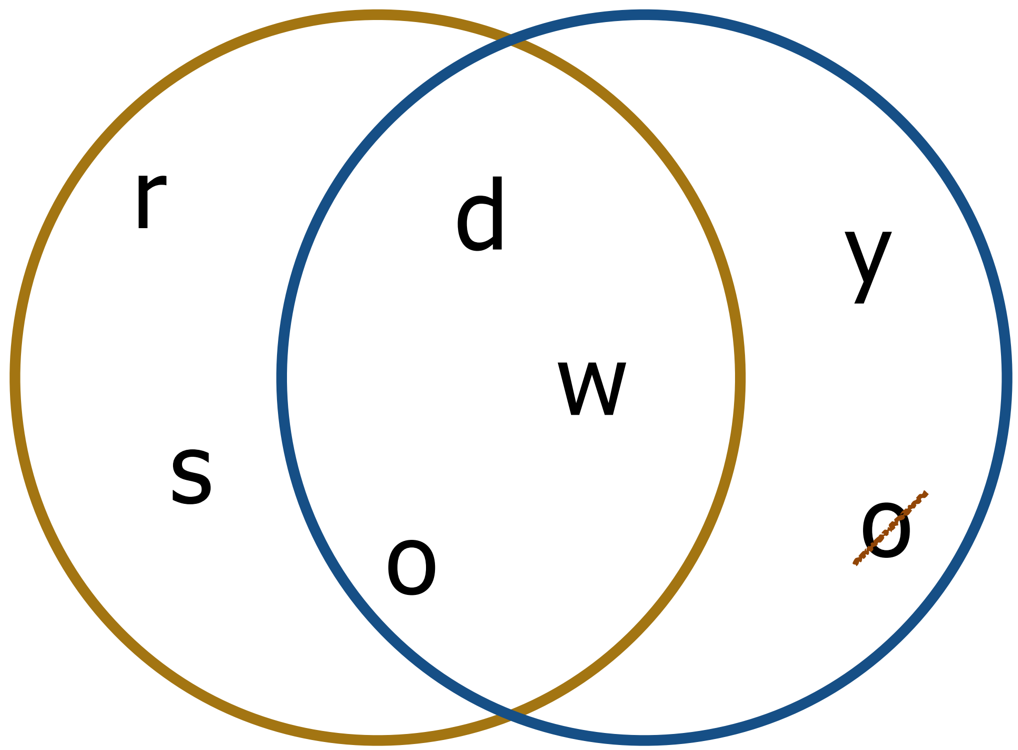
- Goals of string similarity matching:
  - Detecting syntactically or semantically similar words
- Grouping/clustering similar items
  - Content-based **recommender systems**; plagiarism detection
- Information retrieval
  - Ranking/searching for [query-specific] documents
- Entity grounding
  - Recognizing entities from a collection of strings (a "gazetteer" or dictionary)

String matching

- Finding similarly spelled or misspelled words
- Finding "similar" texts/documents
  - N.B.: no semantics (yet…)
- Detecting entries in a **gazetteer**
  - a list of domain-specific words
- Detecting **entities** in a **dictionary**
  - a list of words, each mapped to some **URI**
  - N.B.: "entities", not "concepts" (no semantics…)

Jaccard's set similarity (Jaccard's sim. coeff.)

"words" vs "wo~~o~~dy"



ABC
vs.
ABCABCABC

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{3}{\cancel{7}\ 6} = \cancel{0.43}\ 0.5$$

String similarity measures

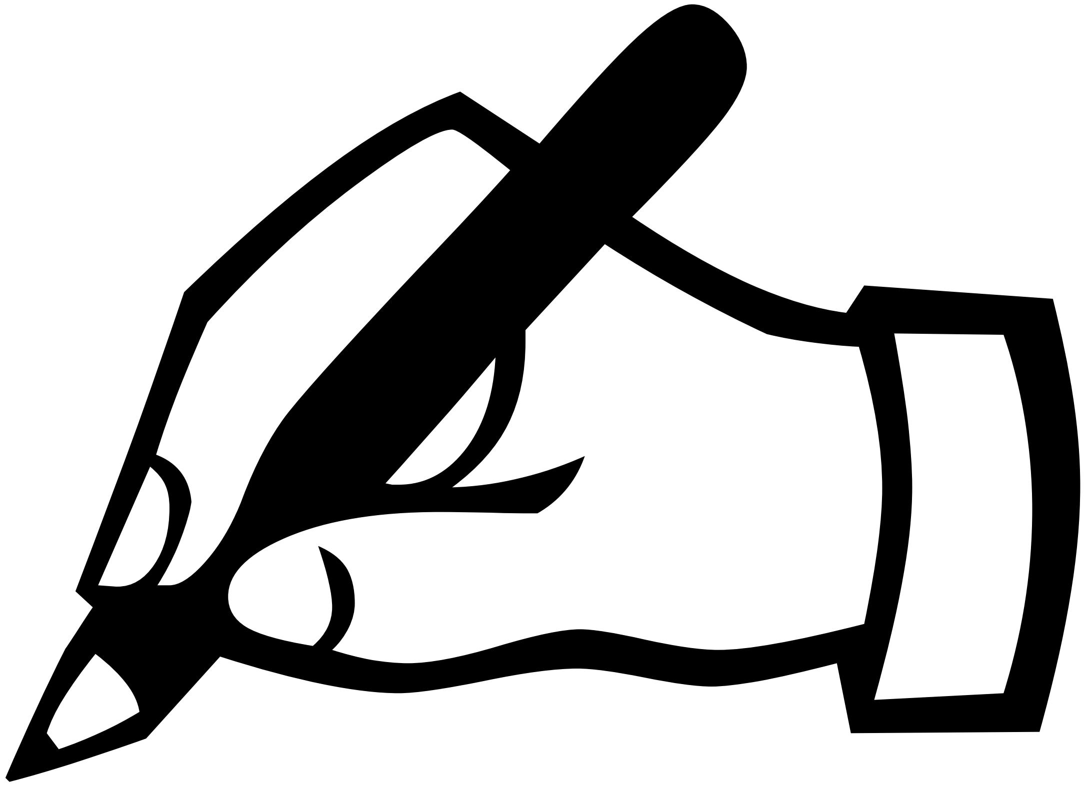
- Edit Distance Measures
  - Hamming Distance [1950]
  - Levenshtein-Damerau Distance [1964/5]
  - Needelman-Wunsch [1970] and Smith-Waterman [1981] Distance
    - also **align** the strings ("sequence alignment")
    - use **dynamic programming**
    - Modern approaches: BLAST [1990], BWT [1994]
- Other Similarity Metrics
  - Jaro-Winkler Similarity [1989/90]
    - coefficient of matching characters within a dynamic window minus transpositions
  - Soundex (→ homophones; spelling!)
  - ...
- Basic Operations: "indels"
  - Insertions
    - ac → abc
  - Deletions
    - abc → ac
- "Advanced" Operations
  - Require two "indels"
  - Substitutions
    - abc → aBc
  - Transpositions
    - ab → ba

typos!

String distance measures

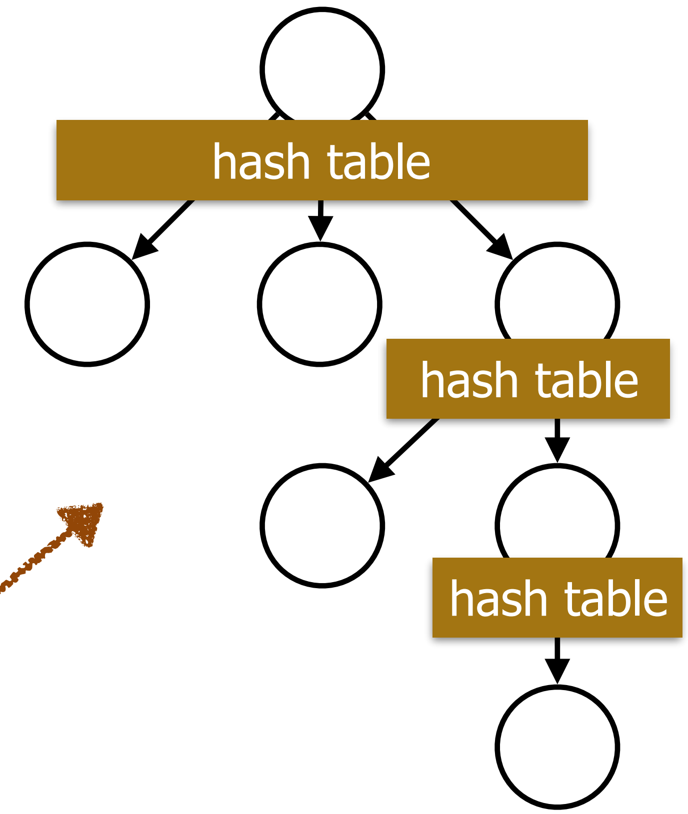
- Hamming Distance
  - only counts **substitutions**
  - requires **equal** string **lengths**
  - karolin ↔ kathrin = 3
  - karlo ↔ carol = 3
  - karlos ↔ carol = `undef`
- Levenshtein Distance
  - counts **all but transpositions**
  - karlo ↔ carol = 3
  - karlos ↔ carol = 4
- **Damerau**-Levenshtein D.
  - also allows **transpositions**
  - has **quadratic complexity**
  - karlo ↔ carol = 2
  - karlos ↔ carol = 3
- Jaro Similarity (a, b)
  - calculates $(m/|a| + m/|b| + (m-t)/m) / 3$
  - m → the # of matching chars in...
  - t → the # of transpositions in...
  - ...**window**: $\lfloor \max(|a|, |b|) / 2 \rfloor - 1$ chars
  - karlos ↔ carol = 0.74 [0,1] range
    - (4 ÷ 6 + 4 ÷ 5 + 3 ÷ 4) ÷ 3
- Jaro-**Winkler** Similarity
  - adds a bonus for matching prefixes

Practical: Jaccard word similarity



Gazetteers: Dictionary matching

- Finding all tokens that match the entries
  - hash table lookups: constant complexity - $O(1)$
- **Exact**, single token matches
  - regular hash table lookup (e.g., MURMUR3)
- Exact, multiple tokens
  - prefix tree of hash tables (each node being a token)



- **Approximate**, single tokens → LSH (next)
  - use some string metric - but do not compare all n-to-m cases…

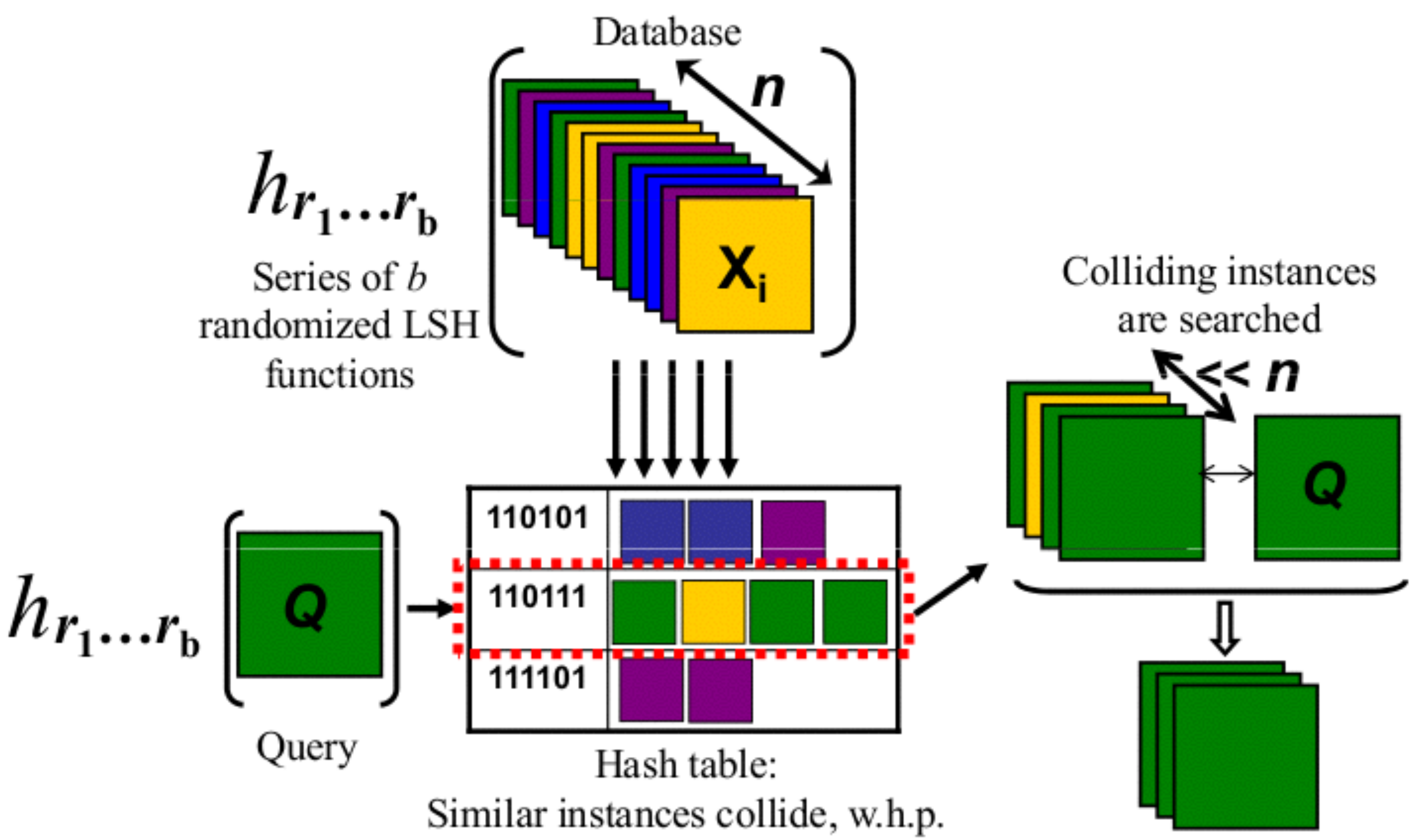
default/traditional approach: character n-gram similarity (e.g. databases)

- Approximate, multiple tokens
  - a prefix tree of whatever the single token lookup strategy…

Locality Sensitive Hashing (LSH) 1/2

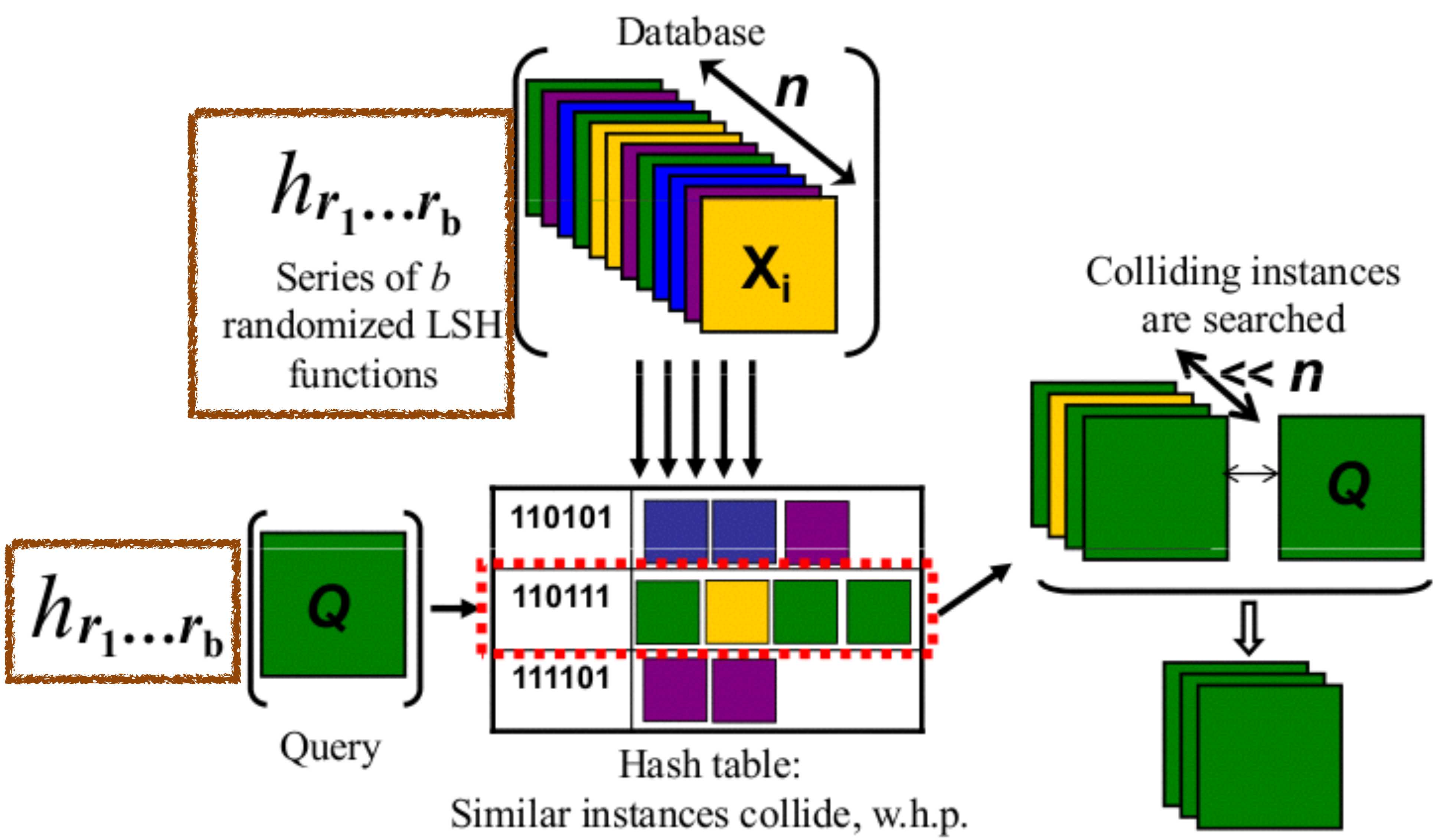
- A **hashing** approach to group **near neighbors**.
- Map similar items into the same [hash] buckets.
- LSH "**maximizes**" (instead of minimizing) hash **collisions**.
- It is another **dimensionality reduction** technique.
- For documents, texts or words, **minhashing** can be used.
  - Approach from Rajaraman & Ullman, Mining of Massive Datasets, 2010
    - http://infolab.stanford.edu/~ullman/mmds/ch3a.pdf

Locality Sensitive Hashing (LSH) 2/2



M Vogiatzis. micvog.com 2013.

Locality Sensitive Hashing (LSH) 2/2



M Vogiatzis. micvog.com 2013.

Minhash signatures (1/2)

n-gram/shingle ID

x	T_1	T_2	T_3	T_4	h_1	h_2
0	**T**	F	F	**T**	1	1
1	F	F	**T**	F	2	4
2	F	**T**	F	**T**	3	2
3	**T**	F	**T**	**T**	4	0
4	F	F	**T**	F	0	3

"permuted" row IDs

$h_1(x) = (x+1)\%n$

$h_2(x) = (3x+1)\%n$

$n=5$

Create character n-gram × token or word k-shingle × document matrix (likely **very** sparse!)

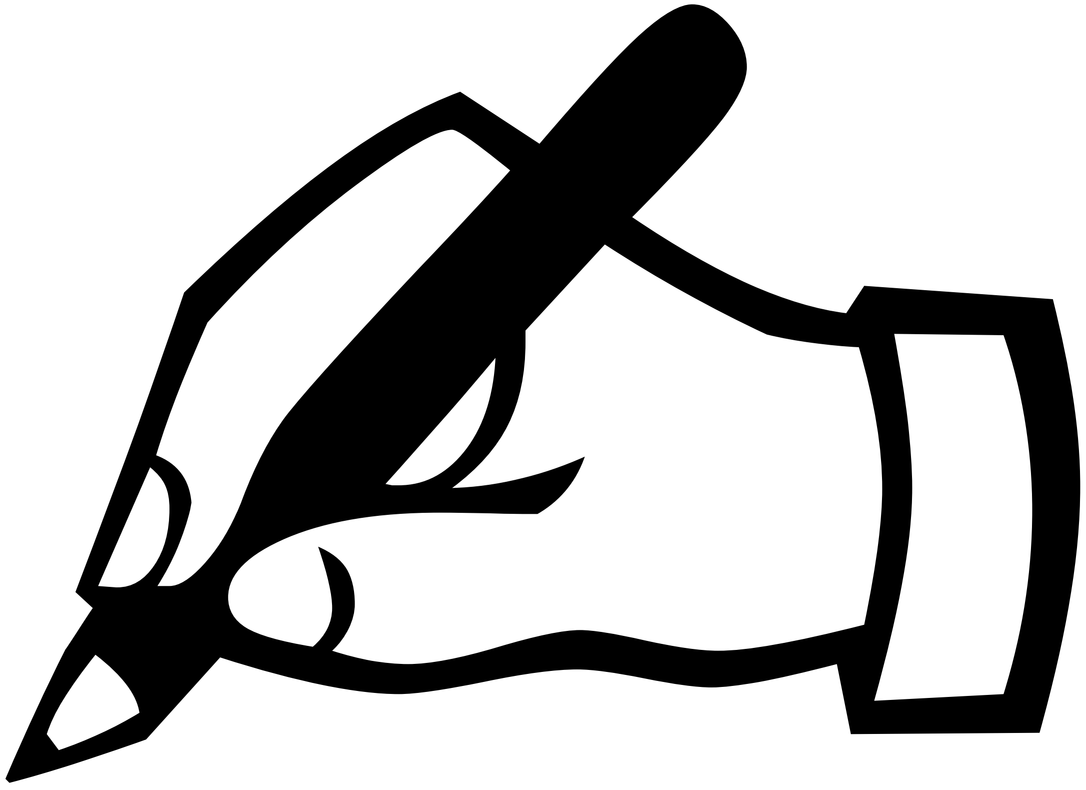
T: shingle/n-gram in Ti
F: shingle/n-gram not in Ti

Lemma: Two texts will have the same first "true" shingle/n-gram when looking from top to bottom with a probability equal to their Jaccard (Set) Similarity.

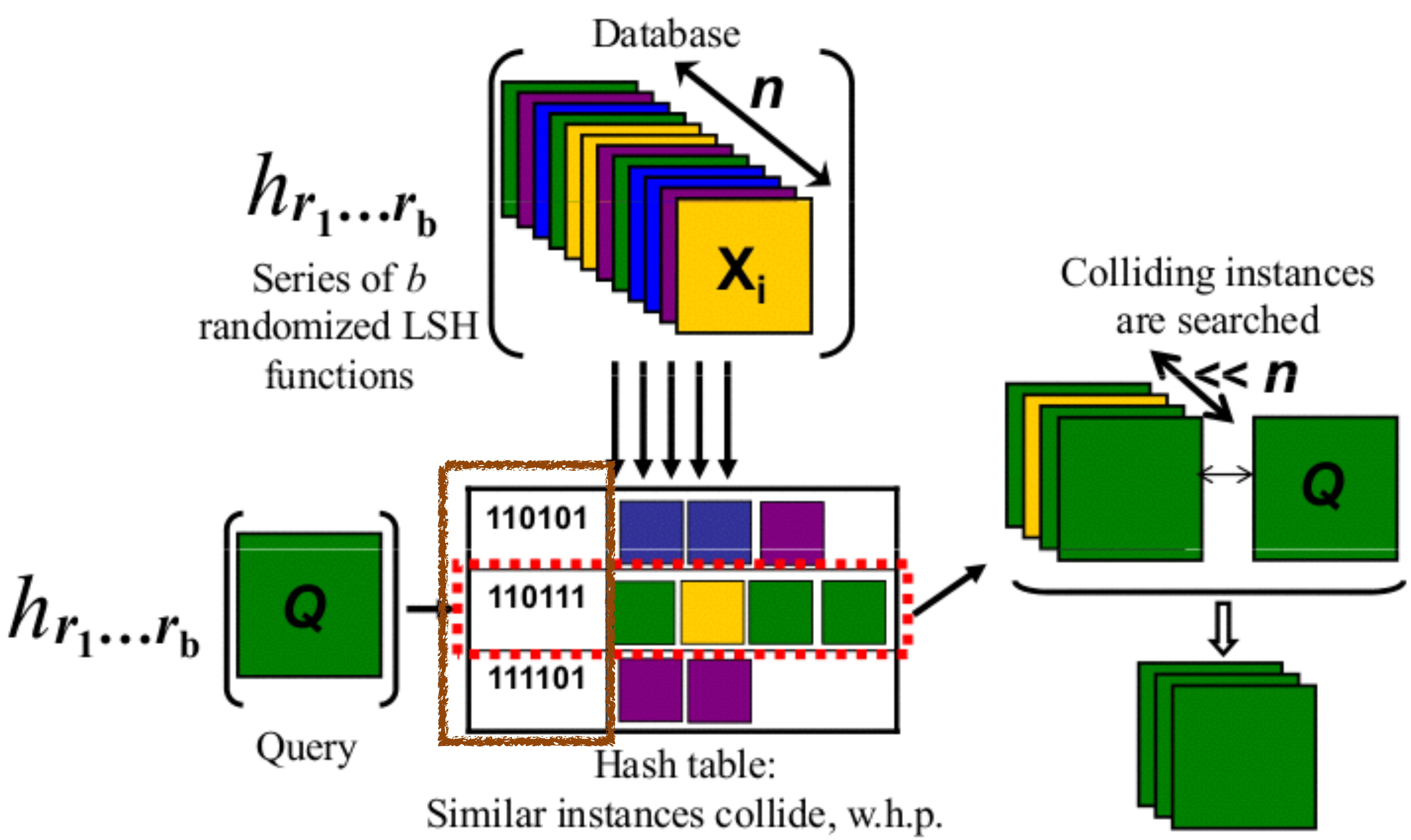
a family of hash functions h_i

Idea: Create ~~*sufficient* **permutations**~~ of the row (shingle/n-gram) ordering so that the Jaccard Similarity can be approximated by comparing the number of coinciding vs. differing rows.

Practical: Implementing minhashing



Locality Sensitive Hashing (LSH) 2/2



M Vogiatzis. micvog.com 2013.

Minhash signatures (2/2)

shingle or n-gram ID

S	T_1	T_2	T_3	T_4		h_1	h_2
0	T	F	F	T		1	1
1	F	F	T	F		2	4
2	F	T	F	T		3	2
3	T	F	T	T		4	0
4	F	F	T	F		0	3

$h_1(x) = (x+1)\%n$

$h_2(x) = (3x+1)\%n$

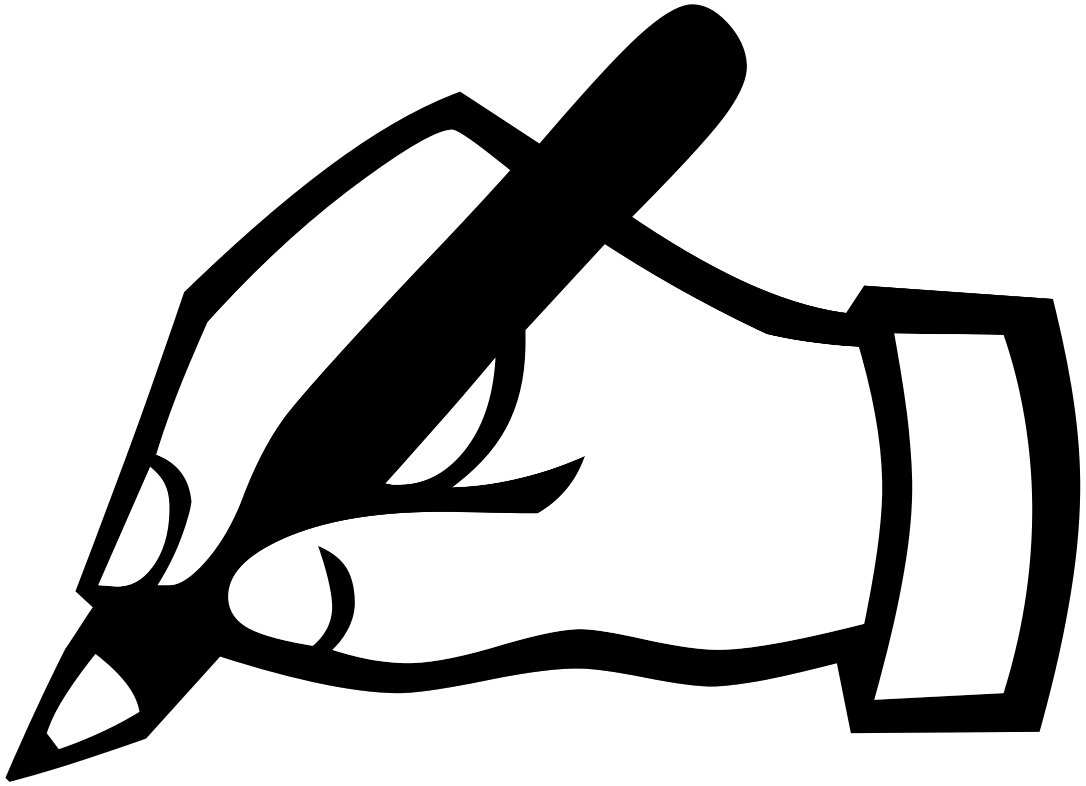
$n=5$

perfectly map-reduce-able and embarrassingly parallel!

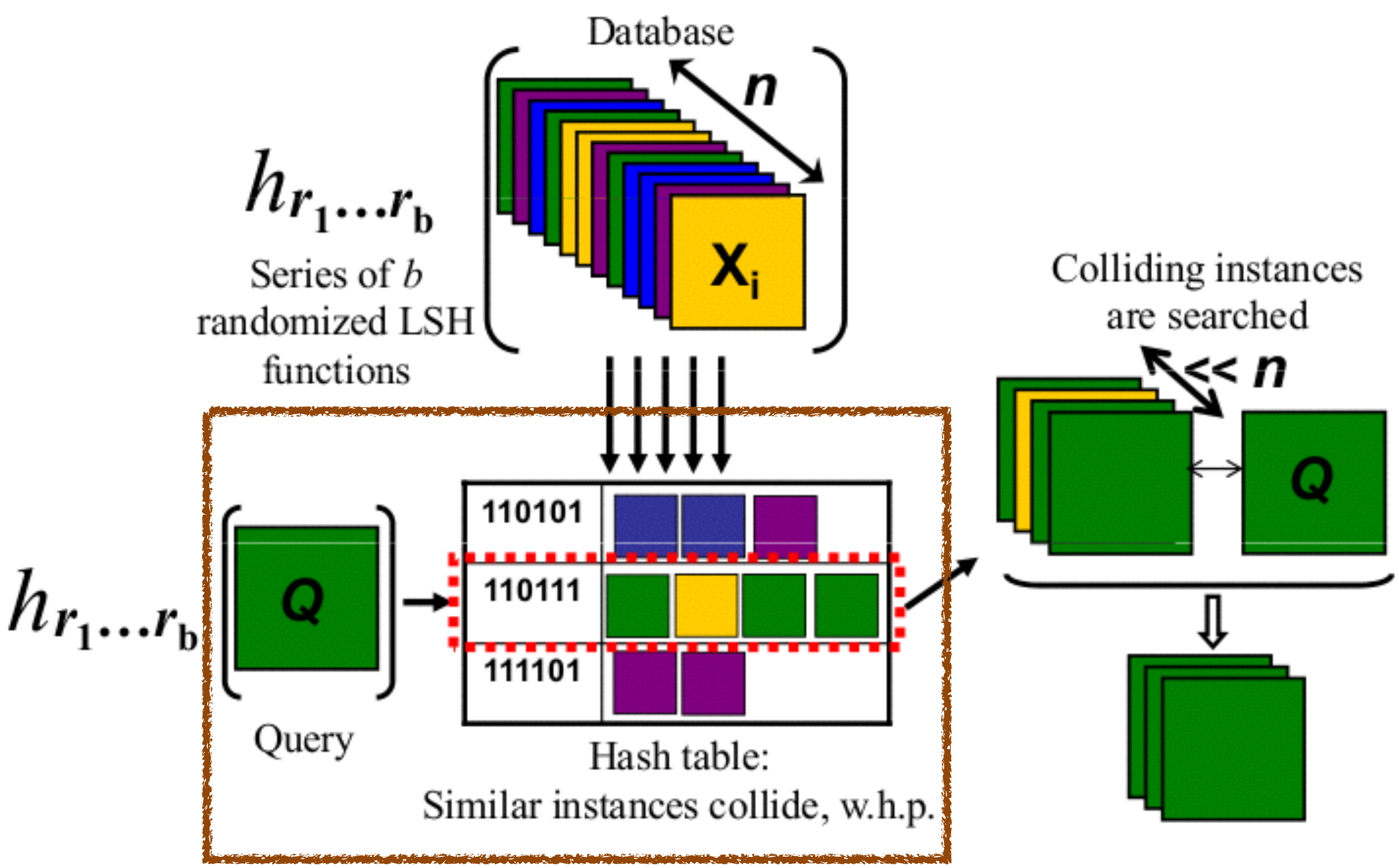
M	T_1	T_2	T_3	T_4
h_1	1	3	**0**	1
h_2	0	2	0	0

```
init matrix M = ∞
for each shingle c,
         hash h, and
         text t:
  if S[t,c] and
     M[t,h] > h(c):
    M[t,h] = h(c)
```

Practical: Implementing LSH with UF



Locality Sensitive Hashing (LSH) 2/2



M Vogiatzis. micvog.com 2013.

Banded locality sensitive minhashing

Bands		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	…
	h_1	**1**	0	2	**1**	7	1	4	5	
1	h_2	**1**	2	4	**1**	6	5	5	6	
	h_3	**0**	5	6	**0**	6	4	7	9	
	h_5	4	0	8	8	**7**	6	5	**7**	
2	h_1	7	7	0	8	**3**	8	7	**3**	
	h_4	8	9	0	7	**2**	4	8	**2**	
	h_2	8	5	**4**	0	9	8	**4**	7	
3	h_6	9	4	**3**	9	0	8	**3**	9	
	h_7	8	5	**8**	0	0	6	**8**	0	
…	…									

typical: r=10, b=30

Bands b $\propto$ $p_{agreement}$

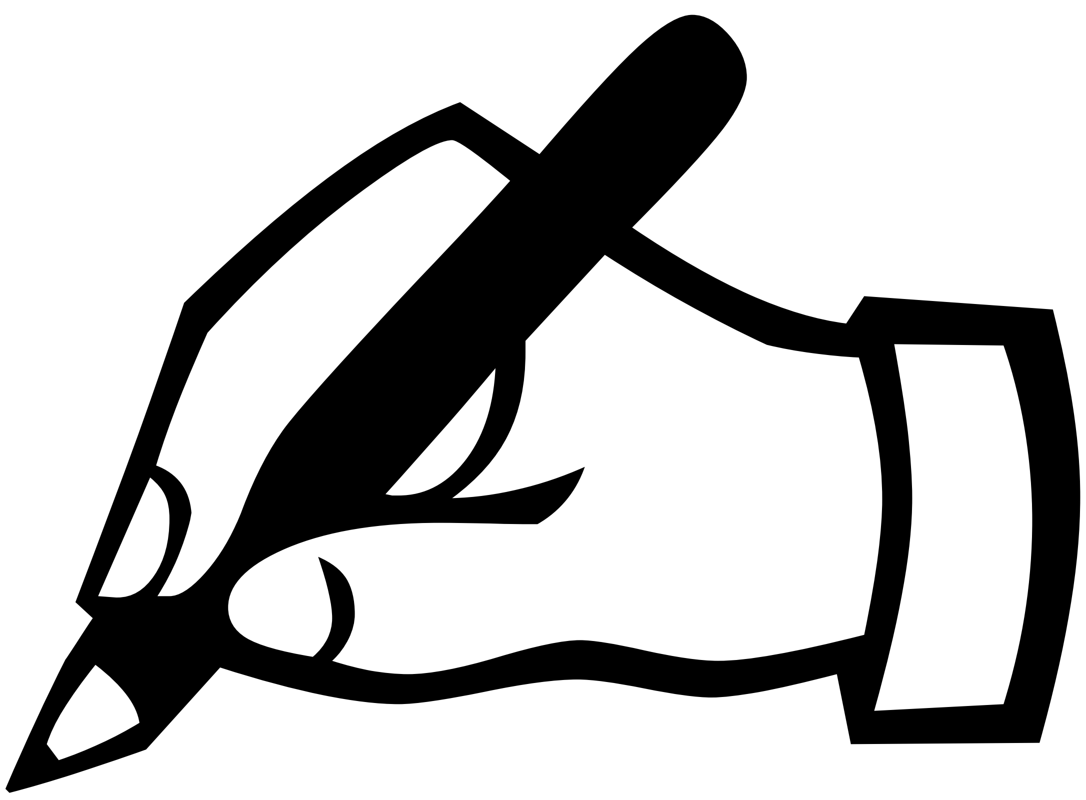
Hashes/Band r $\propto$ $1/p_{agreement}$

(in at least one band)

$p_{agreement} = \mathbf{1 - (1 - s^r)^b}$

s = Jaccard(A,B)

Practical:
Banded LSH



Document similarity

- Similarity measures
  - ✓ Cosine similarity (of document/text vectors)
  - ‣ **Correlation coefficients**
- Word vector normalization
  - ‣ **TF-IDF**
- Dimensionality Reduction/Clustering
  - ✓ Locality Sensitivity Hashing
  - ‣ **Latent semantic indexing**
  - ‣ **Latent Dirichlet allocation** (tomorrow)

Cosine similarity

- Define a similarity score between two document vectors (or the query vector in Information Retrieval)
- **Euclidian** vector **distance** is **length dependent**
- The **cosine** [angle] between two vectors **is not**

$$sim(\vec{x},\vec{y}) = cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{|\vec{x}||\vec{y}|} = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$$

can be dropped if using unit vectors
("length-normalized" a.k.a. "cosine normalization")
only dot-product is now left:
extremely efficient ways to compute

Alternative similarity coefficients

- **Spearman's rank correlation coefficient** ρ (r[ho])
  - Ranking is done by term frequency (**TF**; count)
  - Critique: sensitive to ranking differences that are likely to occur with high-frequency words (e.g., "the", "a", …) ➡ use the **log** of the term count, rounded to two significant digits
    - NB that this is not relevant when only short documents (e.g. titles) with low TF counts are compared
- **Pearson's chi-square test** χ^2
  - Directly on the TFs (counts) - intuition:
    Are the TFs "random samples" from the same base distribution?
  - Usually, χ^2 should be preferred over ρ (Kilgarriff & Rose, 1998)
    - NB that both measures have no inherent normalization of document size
      - preprocessing might be necessary!

Term Frequency times Inverse Document Frequency (TF-IDF)

- **Motivation and background**
- The problem
  - **Frequent terms** contribute most to a document vector's direction, but **not all** terms are **relevant** ("the", "a", …).
- The goal
  - **Separate** important terms from frequent, but **irrelevant terms** in the collection.
- The idea
  - Frequent **terms** appearing **in all documents** tend to be less important **versus** frequent terms in just a **few documents**. → Zipf's Law!
  - Also **dampens** the effect of **topic-specific noun phrases** or an **author's bias** for a specific set of adjectives

Term Frequency times Inverse Document Frequency (TF-IDF)

- **tf.idf**(w) := tf(w) × idf(w)
  - tf: (document-specific) term frequency
  - idf: inverse (global) document frequency

- $\mathbf{tf_{natural}}(w) := \mathrm{count}(w)$
  - $tf_{natural}$: n. of times term w occurs in a document
- $\mathbf{tf_{log}}(w) := \log(\mathrm{count}(w) + 1)$
  - tf_{log}: the TF is smoothed by taking its log

- $\mathbf{idf_{natural}}(w) := N / \sum^{N} \{w_i > 0\}$
  - $idf_{natural}$: n. documents divided by n. documents in which term w occurs
- $\mathbf{idf_{log}}(w) := \log(N / \sum^{N} \{w_i > 0\})$
  - idf_{log}: the IDF is smoothed by taking its log
  - where N is the **number of documents**, w_i the **count of word** w in document i, and $\{w_i>0\}$ is 1 if document i has word w or 0 otherwise

TF-IDF in information retrieval

- Document vectors = tf_{log}

→ i.e. the DVs do not use any IDF weighting (simply for efficiency: QV already has IDF that gets multiplied with DV values)

- Query vector = $tf_{log}\ idf_{log}$
  - Terms are counted on each individual document & the query
- Cosine vector length normalization for TF-IDF scores:
  - Document W normalization

$$\sqrt{\sum_{w \in W} tf_{log}(w)^2}$$

  - Query Q normalization

$$\sqrt{\sum_{q \in Q} (tf_{log}(q) \times idf_{log}(q))^2}$$

- IDF is calculated over the indexed collection of all documents

TF-IDF query score: An example

× ×

	Collection		Query Q				Document D			Similarity
Term	df	idf_{log}	tf	tf_{log}	tf.idf	norm	tf	tf_{log}	tf.1	cos(Q,D)
best	3,5E+05	1,46	1	0,30	0,44	0,21	**0**	0,00	0,00	0,00
text	2,4E+03	3,62	1	0,30	1,09	0,53	10	1,04	0,06	0,03
mining	2,8E+02	4,55	1	0,30	1,37	0,67	8	0,95	0,06	0,04
tutorial	5,5E+03	3,26	1	0,30	0,98	0,48	3	0,60	0,04	0,02
data	9,2E+05	1,04	**0**	0,00	0,00	0,00	10	1,04	0,06	0,00
…	…	…	…	…	0,00	0,00	…	16,00	…	0,00
Sums	1,0E+07		4		2,05		~355	16,11		**0,09**

$\sqrt{}$ of $\sum$ of ^{2} ÷ $\sqrt{}$ of $\sum$ of ^{2} ÷

3 out of hundreds of unique words match (Jaccard < 0.03)

Example idea from: Manning et al. Introduction to Information Retrieval. 2009 free PDF!

From syntactic to semantic similarity

Cosine Similarity, χ^2, Spearman's ρ, LSH, etc. all compare equal tokens.

But what if you are talking about "automobiles" and I am lazy, calling it a "car"?

We can solve this with Latent Semantic Indexing!

Latent Semantic Analysis (LSI 1/3)

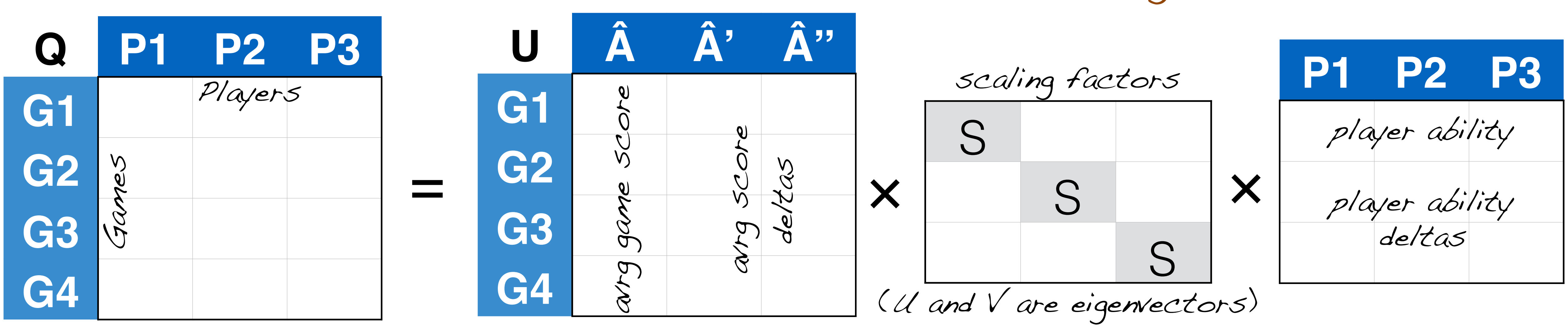
- a.k.a. Latent Semantic **Indexing** (in Text Mining): **feature extraction** for **semantic inference**
- Linear algebra background

  orthonormal factors of Q (QQ^T and Q^TQ)

  - Singular value decomposition of a matrix Q: $\mathbf{Q} = \mathbf{U\Sigma V^T}$

  the factors "predict" Q in terms of similarity (Frobenius norm) using as many factors as the lower dimension of Q

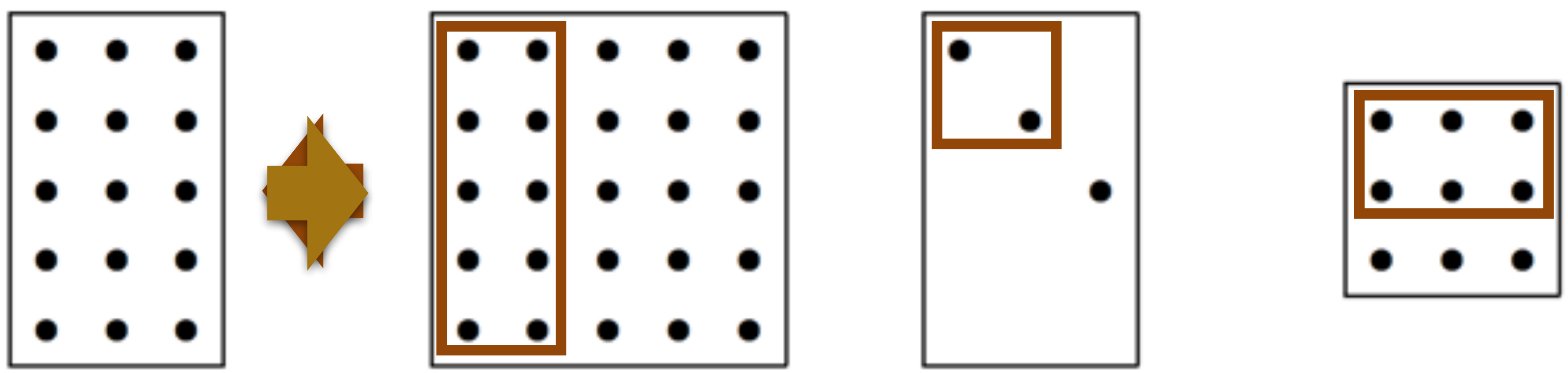
  singular values: scaling factor



- SVD in text mining
  - Inverted index = doc. eigenvectors × singular values × term eigenvectors

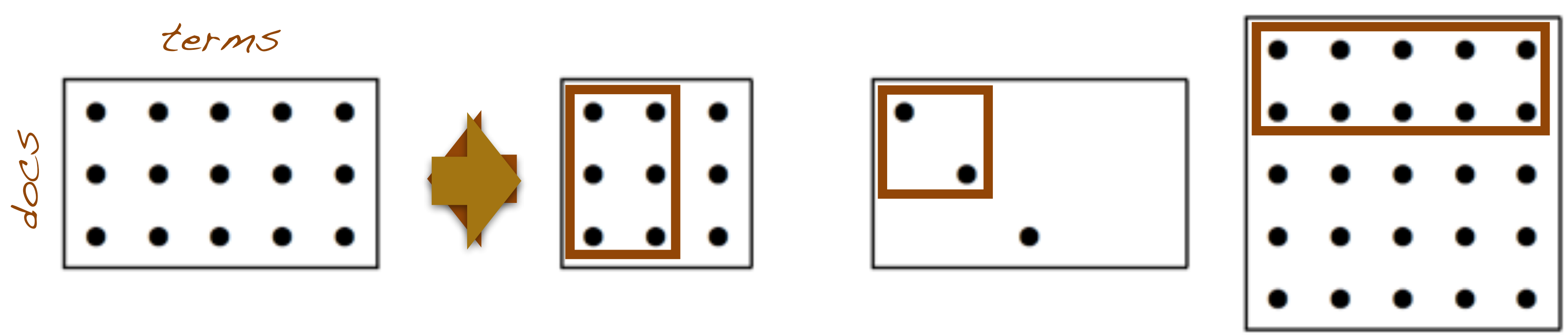
Latent Semantic Analysis (LSI 2/3)

C = ˆFeat. extraction by selecting only the largest n eigenvalues



- Inverted index = doc. eigenvectors × singular values × term eigenvectors

$$C \quad = \quad U \quad \Sigma \quad V^T$$



- Image taken from: Manning et al. An Introduction to IR. 2009

Latent Semantic Analysis (LSI 3/3)

[Spearman's] rho(human, user) = -0.38
rho(human, minors) = -0.29

c1: *Human* machine *interface* for ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user* perceived *response time* to error measurement

m1: The generation of random, binary, ordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

C

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Ĉ

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
user	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
system	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

top 2 dim

test # dim to use via synonyms or missing words

rho(human, user) = **0.94**
rho(human, minors) = **-0.83**

From: Landauer et al. An Introduction to LSA. 1998

Principal Component vs. Latent Semantic Analysis

best Frobenius norm: minimize "std. dev." of matrix
best affine subspace: minimize dimensions while maintaing the form

- **LSA** seeks for the **best linear subspace** in **Frobenius norm**, while **PCA** aims for the **best affine linear subspace**.
- **LSA** (**can**) **use** TF-IDF weighting as **preprocessing** step.
- **PCA requires the** (square) **covariance matrix** of the original matrix as its first step and therefore can only compute term-term or doc-doc similarities.
- **PCA matrices are more dense** (zeros occur only when true independence is detected).

From similarity to labels

- So far, we have seen how to establish if two documents are syntactically (kNN/LSH) and even semantically (LSI) similar.

- But how do we now assign a label (a “class”) to a document?
  - E.g., **relevant**/not relevant; **polarity** (positive, neutral, negative); a **topic** (politics, sport, people, science, healthcare, …)
  - We could use the distances (e.g., from LSI) to **cluster** the documents
  - Instead, let’s look at **supervised methods** next.

Text classification approaches

efficient/fast training

- **Multinomial naïve Bayes***
- **Nearest neighbor classification** (ASDM Course)
  - incl. **locality sensitivity hashing*** (already seen)
- **Latent semantic indexing*** (already seen)
- **Cluster analysis** (ASDM Course)
- **Maximum entropy classification***
- **Latent Dirichlet allocation***
- **Random forests**
- **Support vector machines** (ASDM Course)
- **Artificial neural networks** (ASDM Course)

high accuracy

* this course

Three text classifiers

- Multinomial naïve Bayes
- Maximum entropy (multinomial logistic regression)
- Latent Dirichlet allocation   *unsupervised!*

Maximum A Posterior (MAP) estimator

- Issue: Predict the class $c \in C$ for a given document $d \in D$
- Solution: MAP, a "perfect" Bayesian estimator:

$$C_{MAP}(d) = \underset{c \in C}{argmax} \underbrace{P(c|d)}_{\text{the posterior}} = \underset{c \in C}{argmax} \frac{P(d|c)P(c)}{\underbrace{\cancel{P(d)}}_{\text{redundant}}}$$

- Problem: d is really the set $\{w_1, \ldots, w_n\}$ of dependent words W
  - exponential parameterization: one for each possible combination of W and C

Multinomial naïve Bayes classification

- A simplification of the MAP Estimator
  - count(w) is a discrete, **multinomial** variable (unigrams, bigrams, etc.)
  - Reduce space by making a **strong independence assumption** ("naïve")

independence assumption: "each word is on its own"

$$C_{MAP}(d) = \underset{c \in C}{argmax}\ P(d|c)P(c) \approx \underset{c \in C}{argmax}\ P(c) \prod_{w \in W} P(w|c)$$

"bag of words/features"

- Easy **parameter estimation**

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{|V| + \sum_{w \in V} count(w, c)}$$

count(w_i, c): the total count of word i in all documents of class c [in our training set]

  - V is the entire **vocabulary** (collection of unique words/n-grams/…) in D
  - uses Laplacian/add-one smoothing

Multinomial naïve Bayes: Practical aspects

- Can gracefully handle **unseen words**
- Has **low space requirements**: $|V| + |C|$ floats → sum $\prod$ using logs!
- **Irrelevant** ("stop") **words** cancel each other out
- Opposed to SVM or Nearest Neighbor, it is very **fast**
  - (Locality Sensitivity Hashing was another very efficient approach we saw)
  - But fast approaches tend to result in lower accuracies (➡ good "**baselines**")
- Each class has its own **n-gram language model**
- **Logarithmic damping** ($\log(count)$) might improve classification

$$\hat{P}(w_i|c) = \frac{log(count(w_i, c) + 1)}{log(|V| + \sum_{w \in V} count(w, c))}$$

Practical:
Probabilities and underflows