POLITÉCNICA
"Ingeniamos el futuro"
CAMPUS
DE EXCELENCIA
INTERNACIONAL

Text Mining 3
String Processing

Madrid Summer School 2014
Advanced Statistics and Data Mining

Florian Leitner
florian.leitner@upm.es

Incentive and Applications

- Today's goals:
  - converting strings/text into "processable units" (features for Machine Learning)
  - detecting patterns and comparing strings (string similarity and matching)

- Feature generation for machine learning tasks
  - Detecting a token's grammatical use-case (noun, verb, adjective, ...)
  - Normalizing/regularizing tokens ("[he] was" and "[I] am" are forms of the verb "be")
  - Recognizing entities from a collection of strings (a "gazetteer" or dictionary)
- Near[est] Neighbor-based text/string classification tasks
- String searching (find, UNIX' "grep")
- Pattern matching (locating e-mail addresses, telephone numbers, ...)

Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

Part-of-Speech Tagging

Stemming/Lemmatization

String Metrics & Matching

Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

Part-of-Speech Tagging

Stemming/Lemmatization

String Metrics & Matching

Text Extraction from Documents

- Mostly a commercial business
  - “an engineering problem, not a scientific challenge”
- PDF extraction tools
  - **xpdf** - “the” PDF extraction library
    - http://www.foolabs.com/xpdf/
      (available via `pdfto*` on POSIX systems)
  - **PDFMiner** - a Python **2.x (!)** library
    - https://euske.github.io/pdfminer/
  - **LA-PDF-Text** - layout aware PDF extraction from scientific articles (Java)
    - https://github.com/BMKEG/lapdftextProject
  - **Apache PDFBox** - the Java toolkit for working with PDF documents
    - http://pdfbox.apache.org/
- Optical Character Recognition (OCR) libraries
  - **Tesseract** - Google’s OCR engine (C++)
    - https://code.google.com/p/tesseract-ocr/
    - https://code.google.com/p/pytesser/
      (a Python **2.x (!)** wrapper for the Tesseract engine)
  - **OCRopus** - another Open Source OCR engine (Python)
    - https://code.google.com/p/ocropus/
- Generic text extraction (Office documents, HTML, Mobi, ePub, …)
  - **Apache Tika** “content analysis toolkit” (Language Detection!)
    - http://tika.apache.org/

Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

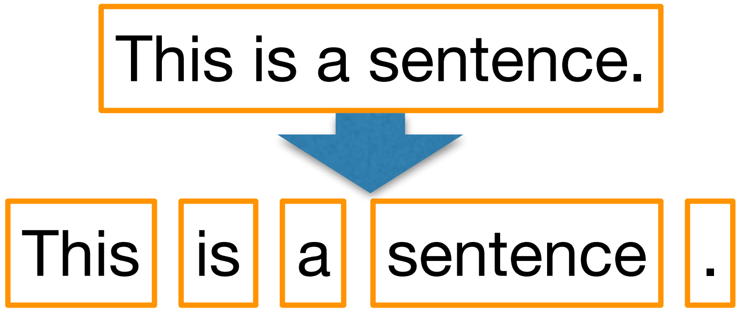
Part-of-Speech Tagging

Stemming/Lemmatization

String Metrics & Matching

Lexers and Tokenization

- **Lexer**: converts (“**tokenizes**”) strings into token sequences
  - a.k.a.: **Tokenizer**, **Scanner**
- Tokenization strategies
  - whitespace (“\s+”) [“PhD.”, “U.S.A.”]
  - word (letter vs. non-letter) boundary (“\b”) [“PhD”, “.”, “U”, “.”, “S”, “.”, “A”, “.”]
  - Unicode category-based ([“C”, “3”, “P”, “0”], [“Ph”, “D”, “.”, “U”, “.”, “S”, “.” …])
  - linguistic ([“that”, “’s”]; [“do”, “n’t”]; [“adjective-like”]; [“PhD”, “.”, “U.S.A”, “.”])
  - whitespace/newline-preserving ([“This”, “ ”, “is”, “ ”, …]) or not
- Task-oriented choice: document classification, entity detection, linguistic analysis, …



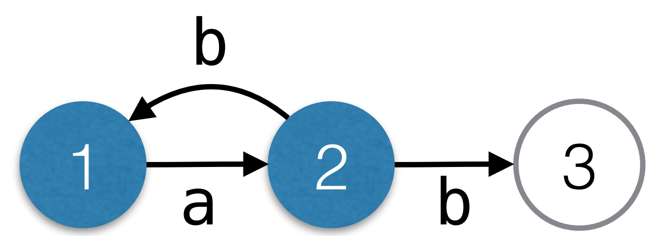
String Matching with Regular Expressions

- `/^[^@]{1,64}@\w(?:[\w.-]{0,254}\w)?\.\w{2,}$/`
  - (provided `\w` has Unicode support)
  - http://ex-parrot.com/~pdw/Mail-RFC822-Address.html

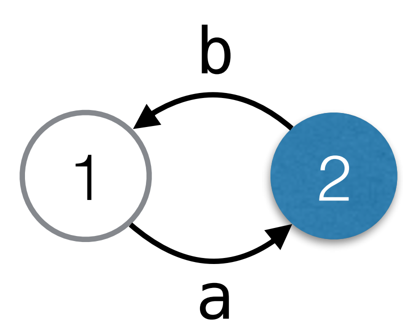
- Wild card: `.` (a dot → matches any character)
- Groupings: `[A-Z0-9]`; Negations: `[^a-z]`; Captures `(save_this)`
- Markers: `^` "start of string/line"; `$` "end of string/line"; [`\b` "word boundary"]
- Pattern repetitions: `*` "zero or more"; `*?` "zero or more, non-greedy"; `+` "one or more"; `+?` "one or more, non-greedy"; `?` "zero or one"
- Example: `\b([a-z]+)\b.+$` captures lower-case words except at the EOS

String Matching with Finite State Automata 1/2

- Matching **single words** or **patterns**:
  - **Regular Expressions** are “translated” to **finite-state automata**
  - Automata can be deterministic (DFA, $O(n)$) or non-deterministic (NFA, $O(nm)$)
    - where n is the length of the string being scanned and m is the length of the string being matched
  - Programming languages’ default implementation are (slower) NFAs because certain special expressions (e.g., “look-ahead” and “look-behind”) cannot be implemented with a DFA
  - DFA implementations “in the wild”
    - **RE2** by Google: https://code.google.com/p/re2/ [**C++**, and the default RE engine of **Golang**, with wrappers for a few other languages]
    - **dk.brics.automaton**: http://www.brics.dk/automaton/ [**Java**, but very memory “hungry”]



(ab)+



String Matching with Finite State Automata 2/2

Dictionaries:

Exact string matching of
word collections

Compressed Prefix Trees (**Tries**):
PATRICIA Trie

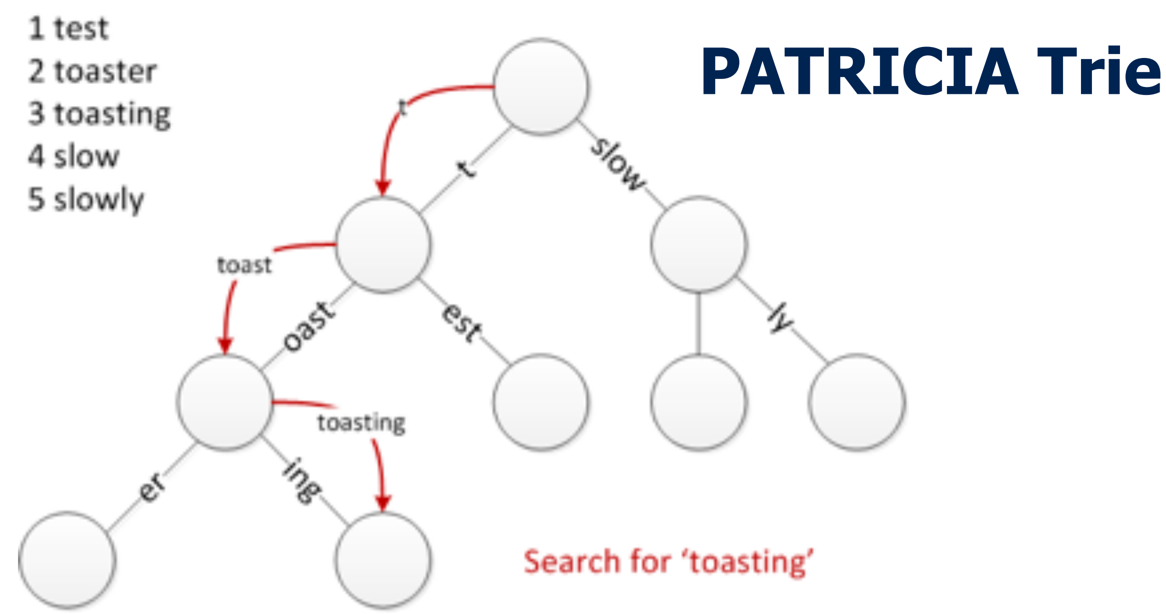
Prefixes → autocompletion functionality

Minimal (compressed) DFA
(a.k.a. **MADFA**, **DAWG** or **DAFSA**)

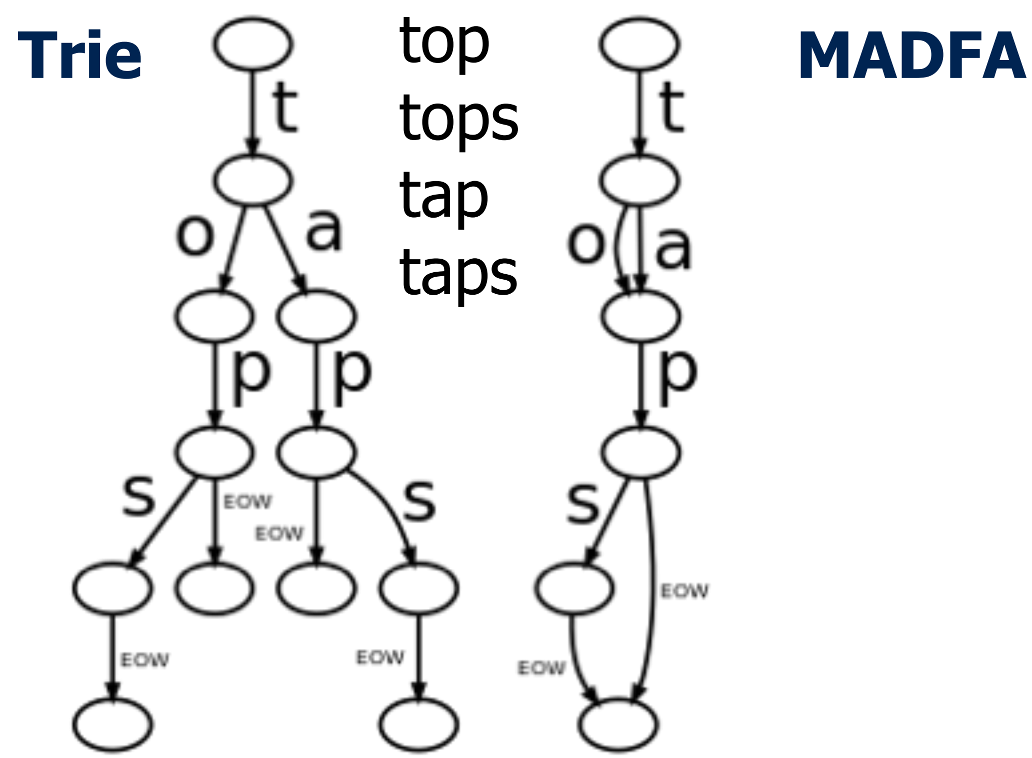
a compressed dictionary

Python: `pip install DAWG`
https://github.com/kmike/DAWG
["kmike" being a NLTK dev.]

Daciuk et al. 2000



Images Source: WikiMedia Commons, Saffles & Chkno



Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

Part-of-Speech Tagging

Stemming/Lemmatization

String Metrics & Matching

Sentence Segmentation

- Sentences are **the** fundamental linguistic unit
  - Sentence: boundary or "**constraint**" for **linguistic phenomena**
  - collocations ["United Kingdom", "vice president"], idioms ["drop me a line"], phrases [PP: "of great fame"] and clauses, statements, …
- Rule/pattern-based segmentation
  - Segment sentences if the marker is followed by an upper-case letter
    - `/[.!?]\s+[A-Z]/`

turns out letter case is a poor man's rule!

  - **Special cases**: abbreviations, digits, lower-case proper nouns (genes, "amnesty international", …), hyphens, quotation marks, …

maintaining a proper rule set gets messy fast

- **Supervised** sentence boundary detection
  - Use some Markov model or a conditional random field to identify possible sentence segmentation tokens
  - Requires labeled examples (segmented sentences)

Punkt Sentence Tokenizer (PST) 1/2

- **Unsupervised Multilingual** Sentence Boundary Detection
- $P(\bullet|\mathbf{w}_{-1}) > \mathbf{c}_{cpc}$

  Dr.

  - Determines if a marker • is used as an **abbreviation** marker by comparing the **conditional probability** that the word $\mathbf{w}_{-1}$ before • is followed by the marker against some (high) cutoff probability.
    - $P(\bullet|\mathbf{w}_{-1}) = P(\mathbf{w}_{-1}, \bullet) \div P(\mathbf{w}_{-1})$
    - K&S set $\mathbf{c} = 0.99$
- $P(\mathbf{w}_{+1}|\mathbf{w}_{-1}) > P(\mathbf{w}_{+1})$

  Mrs. Watson

  - Evaluates the likelihood that $\mathbf{w}_{-1}$ and $\mathbf{w}_{+1}$ surrounding the marker • are more commonly collocated than would be expected by chance: • is assumed an **abbreviation** marker ("not independent") if the LHS is greater than the RHS.
- $F_{length}(\mathbf{w}) \times F_{markers}(\mathbf{w}) \times F_{penalty}(\mathbf{w}) \geq \mathbf{c}_{abbr}$

  U.S.A.

  - Evaluates if any of $\mathbf{w}$'s morphology (length of $\mathbf{w}$ w/o marker characters, number of periods inside $\mathbf{w}$ (e.g., ["U.S.A"]), penalized when $\mathbf{w}$ is not followed by a •) makes it more likely that $\mathbf{w}$ is an abbreviation against some (low) cutoff.
- $F_{ortho}(\mathbf{w})$; $P_{sstarter}(\mathbf{w}_{+1}|\bullet)$; …

  . Therefore

  - Orthography: lower-, upper-case or capitalized word after a probable • or not
  - Sentence Starter: Probability that $\mathbf{w}$ is found after a •

Punkt Sentence Tokenizer (PST) 2/2

- **Unsupervised Multilingual** Sentence Boundary Detection
  - Kiss & Strunk, MIT Press 2006.
  - Available from NLTK: `nltk.tokenize.punkt` (http://www.nltk.org/api/nltk.tokenize.html)
- PST is language agnostic
  - Requires that the language uses the sentence segmentation marker as an abbreviation marker
  - Otherwise, the problem PST solves is not present
- PST factors in word length
  - Abbreviations are relatively shorter than regular words
- PST takes "internal" markers into account
  - E.g., "U.S.A"
- Main weakness: long lists of abbreviations
  - E.g., author lists in citations
  - Can be fixed with a pattern-based post-processing strategy
- NB: a marker must be present
  - E.g., chats or fora

Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

Part-of-Speech Tagging

Stemming/Lemmatization

String Metrics & Matching

The Parts of Speech

I	ate	the	pizza	with	green	peppers	.
PRP	VB	DT	NN	IN	JJ	NN	.

" PoS tags "

- The Parts of Speech:
  - noun: NN, verb: VB, adjective: JJ, adverb: RB, preposition: IN, personal pronoun: PRP, …
  - e.g. the full **Penn Treebank PoS tagset** contains 48 tags:
  - 34 grammatical tags (i.e., "real" parts-of-speech) for words
  - one for cardinal numbers ("CD"; i.e., a series of digits)
  - 13 for [mathematical] "SYM" and currency "$" symbols, various types of punctuation, as well as for opening/closing parenthesis and quotes

The Parts of Speech

I	ate	the	pizza	with	green	peppers	.
PRP	VB	DT	NN	IN	JJ	NN	.

- Automatic PoS Tagging ➡ **Supervised Machine Learning**
  - Maximum Entropy Markov Models
  - **Conditional Random Fields**
  - Ensemble Methods

will be explained in the Text Mining #5!
(last lesson)

Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

Part-of-Speech Tagging

Stemming/Lemmatization

String Metrics & Matching

Linguistic Morphology

➡ token normalization

- [Verbal] **Inflections**
  - conjugation (Indo-European languages)
  - tense ("availability" and use varies across languages)
  - modality (subjunctive, conditional, imperative)
  - voice (active/passive)
  - …
- **Contractions**

  not a contraction: possessive s

  - don't say you're in a man**'s** world…
- **Declensions**
  - on nouns, pronouns, adjectives, determiners
  - case (nominative, accusative, dative, ablative, genitive, …)
  - gender (female, male, neuter)
  - number forms (singular, plural, dual)
  - possessive pronouns (I→my, you→your, she→her, it→its, … car)
  - reflexive pronouns (for myself, yourself, …)
  - …

Stemming vs Lemmatization

➡ token normalization

a.k.a. token "regularization"

(although that is technically the wrong wording)

- Stemming
  - produced by "**stemmers**"
  - produces a word's "stem"
  - am → am
  - the going → the go
  - having → hav
  - fast and simple (pattern-based)
  - **Snowball**; **Lovins**; **Porter**
    - `nltk.stem.*`

- Lemmatization
  - produced by "**lemmatizers**"
  - produces a word's "lemma"
  - am → be
  - the going → the going
  - having → have
  - requires: a dictionary and PoS
  - **LemmaGen**; **morpha**
    - `nltk.stem.wordnet`

Text/Document Extraction

Tokenization: Lexer/Scanner

Sentence Segmentation

Part-of-Speech Tagging

Stemming/Lemmatization

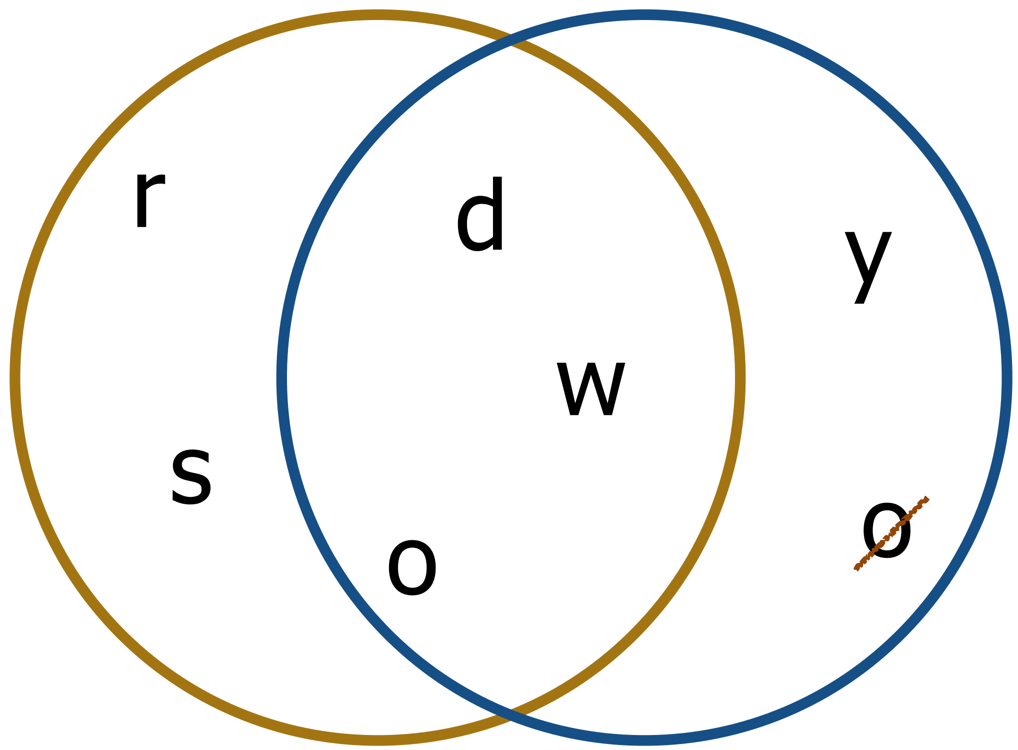
String Metrics & Matching

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 Similarity

"words" vs "woody"



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

String Similarity

Wikipedia is gold here!
(for once...)

- 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 Distance Metrics
  - Jaro-Winkler Distance [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

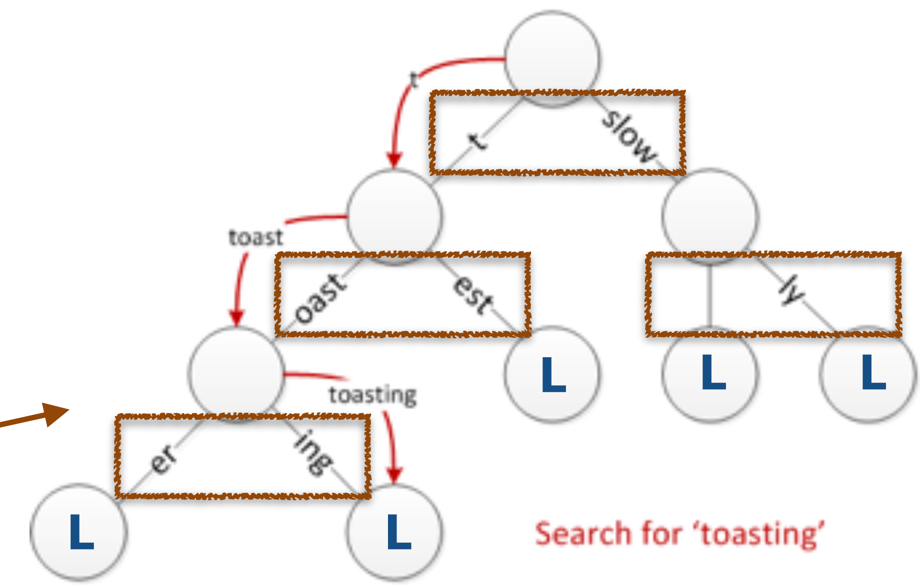
Distance Measures

- 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** *typos!*
  - has **quadratic complexity**
  - karlo ↔ carol = 2
  - karlos ↔ carol = 3
- Jaro Distance (a, b)
  - calculates $(m/|a| + m/|b| + (m-t)/m) / 3$
  - 1. m, the # of matching characters
  - 2. t, the # of transpositions
  - **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** Distance
  - adds a bonus for matching prefixes

`nltk.metrics.distance`

Gazetteer/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 pointing to child tables or leafs (=matches)

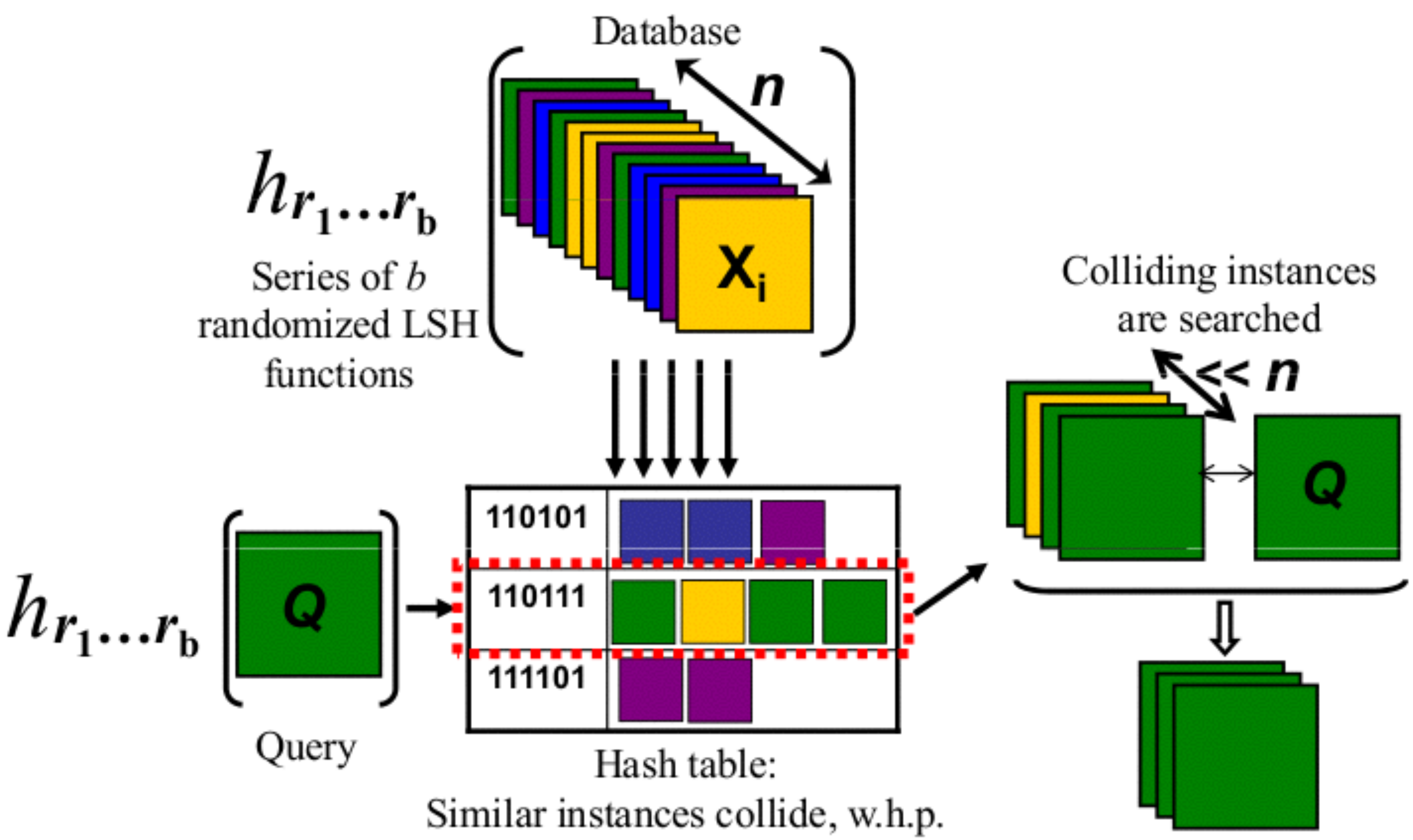


- **Approximate**, single tokens → LSH (next)
  - use some string metric - but do not compare all n-to-m cases…
- Approximate, multiple tokens
  - a prefix tree of whatever we will use for single tokens…

Locality Sensitive Hashing (LSH) 1/2

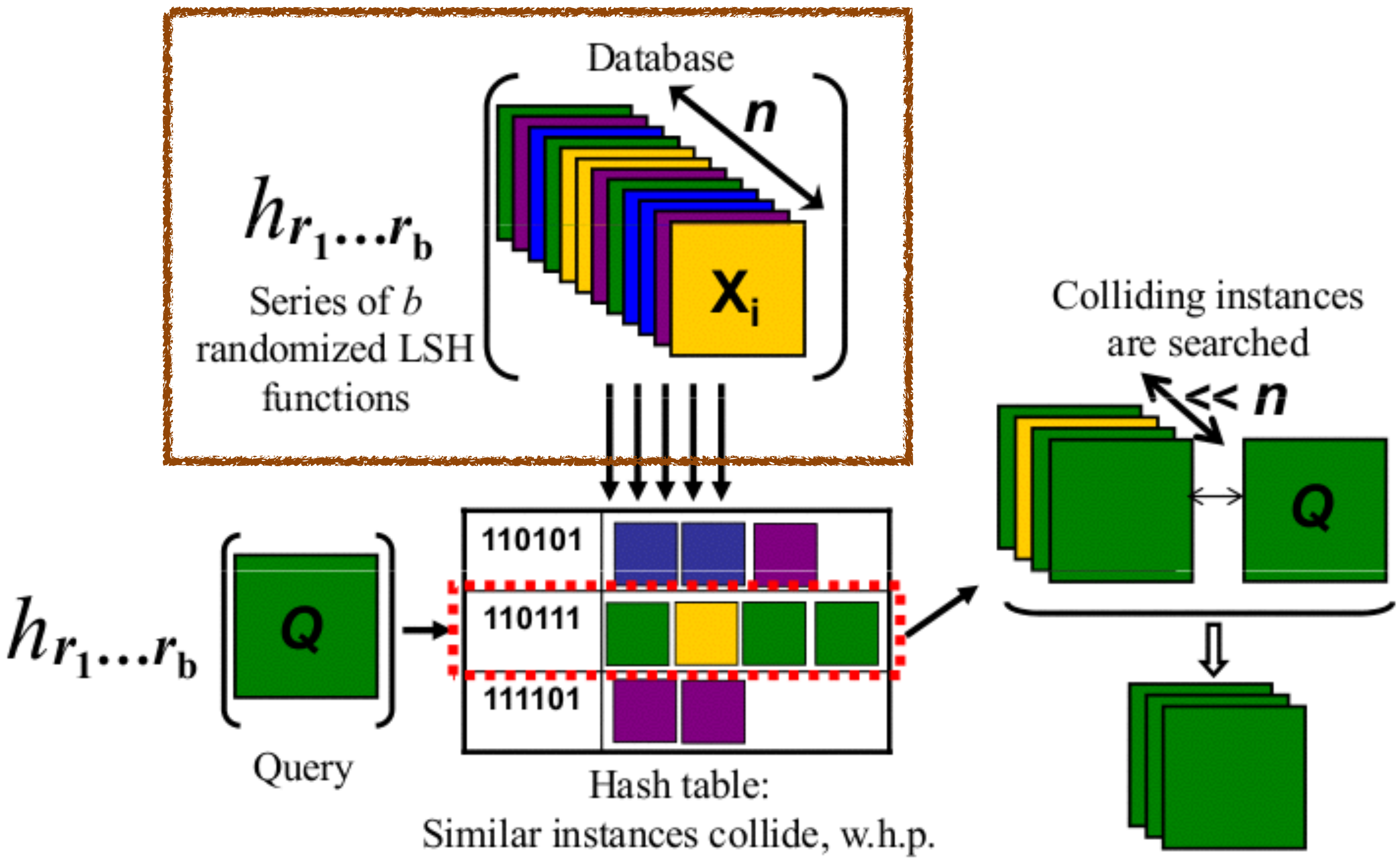
- A **hashing** approach to group **near neighbors**.
- Map similar items into the same [hash] buckets.
- LSH "**maximizes**" (instead of minimizing) hash **collisions**.
- It is another **dimensionality reduction** technique.
- For documents or words, **minhashing** can be used.
  - Approach from Rajaraman & 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)

shingle or n-gram ID	T	T	T	T	h	h
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

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

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

$n=5$

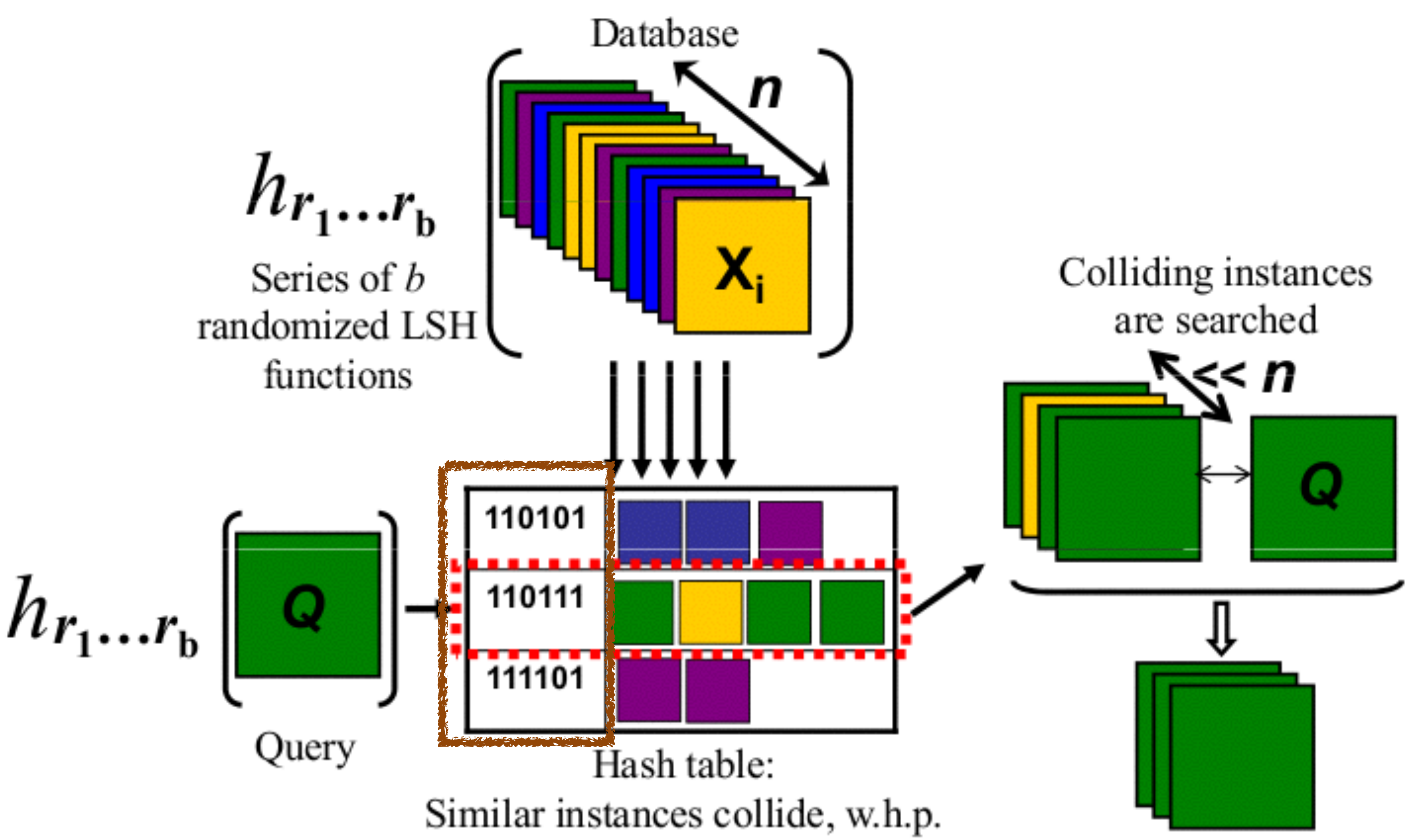
Create n-gram x token or
k-shingle x document matrix
(likely **very** sparse!)

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

a family of hash functions

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.

Locality Sensitive Hashing (LSH) 2/2



M Vogiatzis. micvog.com 2013.

Minhash Signatures (2/2)

shingle or n-gram ID

S	T	T	T	T	h	h
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

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

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

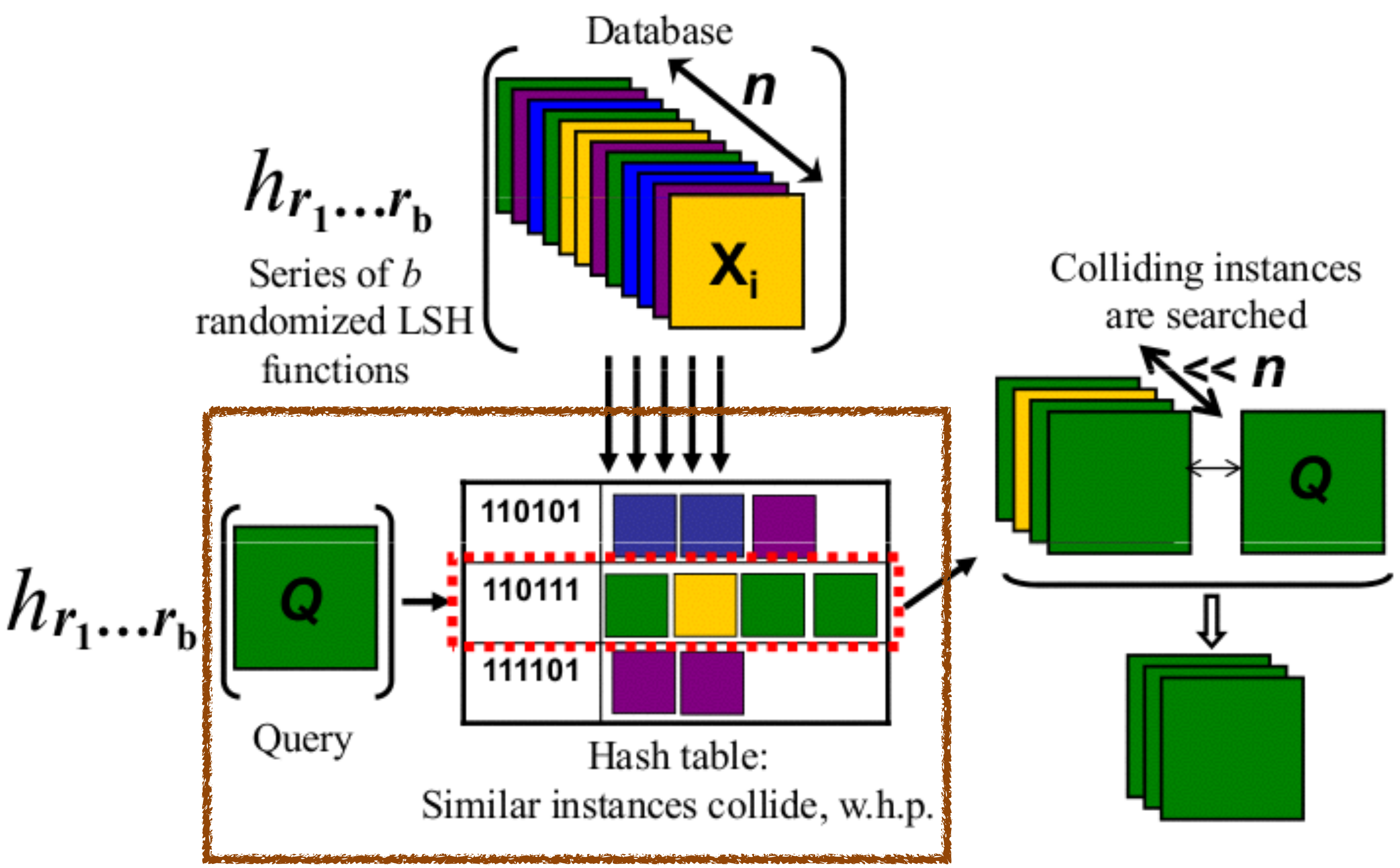
$n=5$

perfectly map-reduce-able and embarrassingly parallel!

M	T	T	T	T
h	1	3	**0**	1
h	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)
```

Locality Sensitive Hashing (LSH) 2/2



M Vogiatzis. micvog.com 2013.

Banded Locality Sensitive Minhashing

Bands		T	T	T	T	T	T	T	T	...
1	h	**1**	0	2	**1**	7	1	4	5	
	h	**1**	2	4	**1**	6	5	5	6	
	h	**0**	5	6	**0**	6	4	7	9	
2	h	4	0	8	8	**7**	6	5	**7**	
	h	7	7	0	8	**3**	8	7	**3**	
	h	8	9	0	7	**2**	4	8	**2**	
3	h	8	5	**4**	0	9	8	**4**	7	
	h	9	4	**3**	9	0	8	**3**	9	
	h	8	5	**8**	0	0	6	**8**	0	
...	...									

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: Faster Spelling Correction with LSH

- Using the spelling correction tutorial from yesterday and the provided banded, minhashing-based LSH implementation, develop a spelling corrector that is nearly as good as Peter Norvig's, but at least twice as fast (3x is possible!).
- Tip 1: if you are using NLTK 2.0.x (you probably are!), you might want to copy-paste the 3.0.x sources for the `nltk.metrics.distance.edit_distance` function
- The idea is to play around with the LSH parameters (threshold, size), the parameter K for K-shingling, and the post-processing of the matched set to pick the best solution from this "<< n subspace" (see LSH 2/2 slide).
- Tip 2/remark: without any post-processing of matched sets, you can achieve about 30% accuracy at a 100x speed up!

"There is a fine line between reading a message from the text and reading one into the text."

John Corvino, 2013