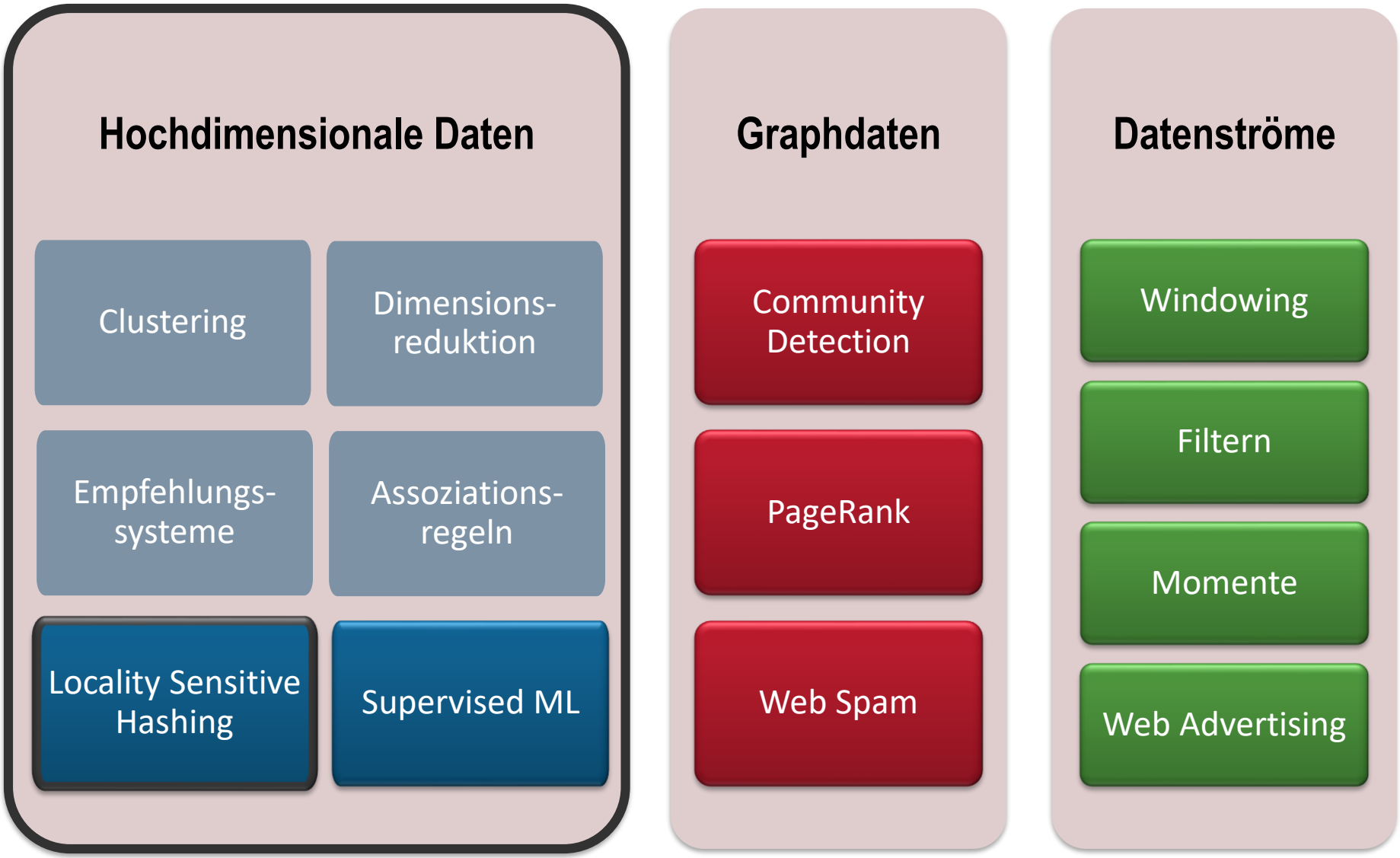
Data Mining

Locality Sensitive Hashing

Dr. Hanna Köpcke
Wintersemester 2020

Abteilung Datenbanken, Universität Leipzig
http://dbs.uni-leipzig.de

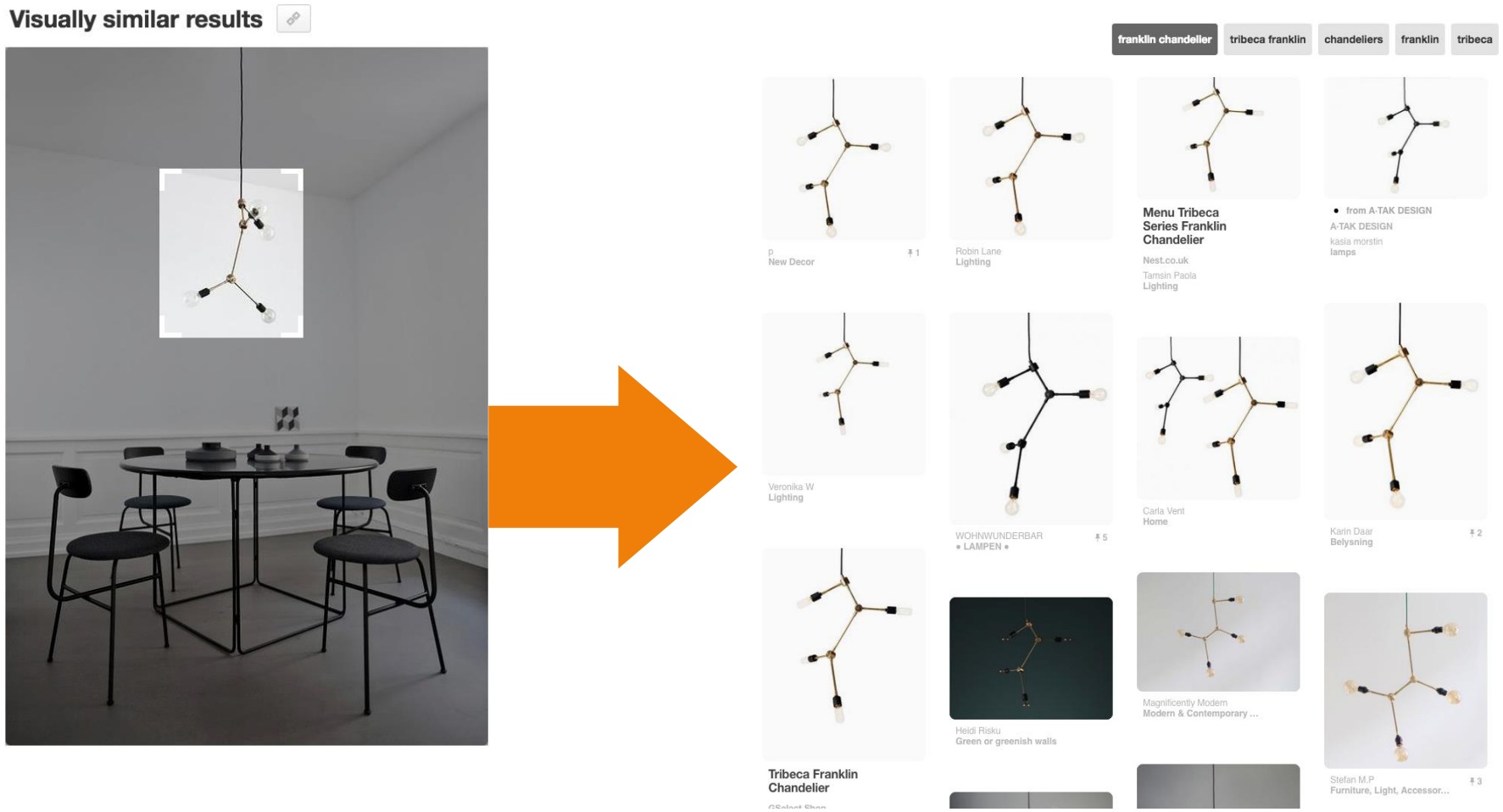
Übersicht


Hochdimensionale Daten
Clustering
Dimensions-reduktion
Empfehlungs-systeme
Assoziations-regeln
Locality Sensitive Hashing
Supervised ML
Graphdaten
Community Detection
PageRank
Web Spam
Datenströme
Windowing
Filtern
Momente
Web Advertising

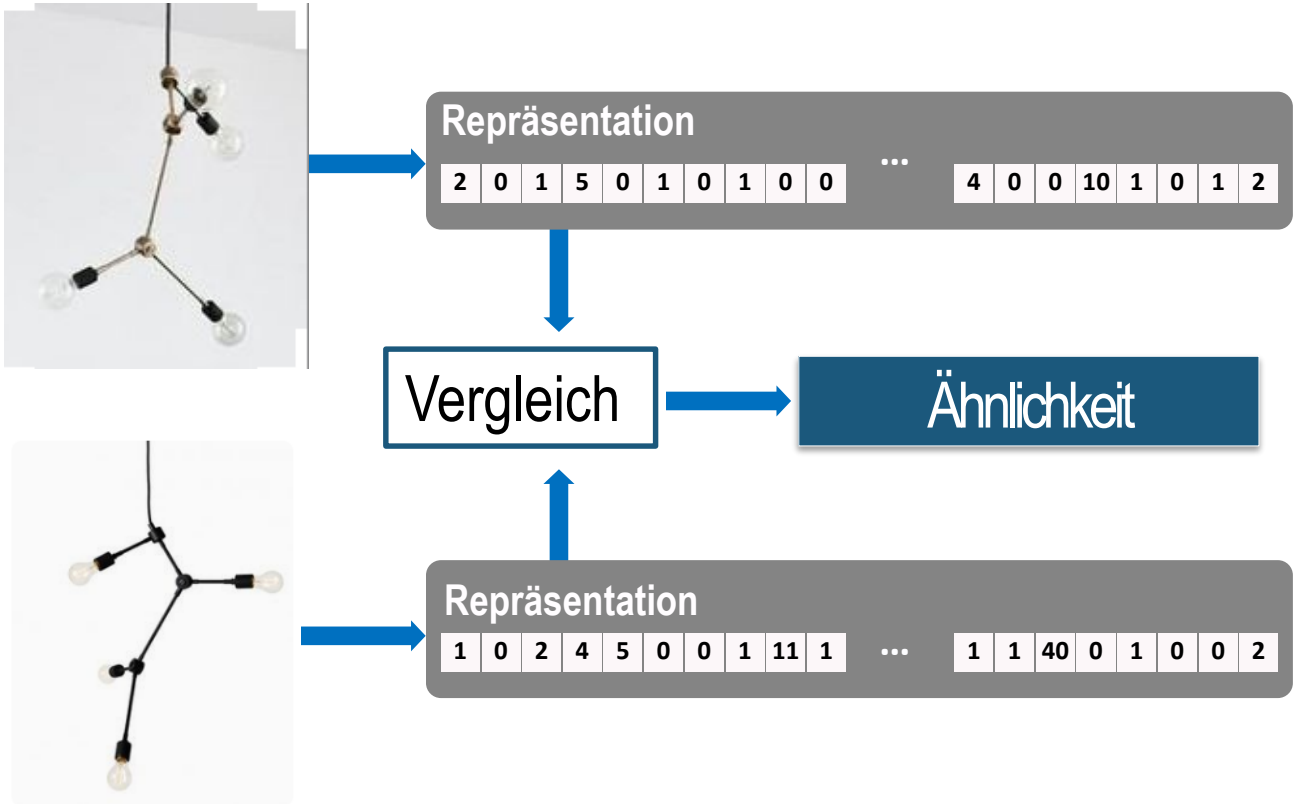
Inhaltsverzeichnis

Pinterest Visual Search

Finde ähnliche Bilder für einen gegebenen Bildbereich



Funktionsweise



- Sammlung aus Milliarden von Bildern
- Berechne Repräsentation für jedes Bild (z.B. 4000 Dimensionen)
- **Finde die ähnlichsten Bilder für ein gegebenes Bild**

Anwendungen

- Viele Probleme können als folgendes Suchproblem formuliert werden:

  Finde die nächsten Nachbarn in einem hochdimensionalen Raum

- *Beispiele:*
  - Dokumente mit ähnlichen Wörtern
    - Klassifikation
    - Duplikateleminierung, z.B. in der Webseiten-/Nachrichtensuche
    - Plagiaterkennung
  - Kunden mit ähnlichem Kaufverhalten
  - *Empfehlungen*: Musik, Filme, ...
  - Bilder mit ähnlichen Merkmalen, z.B. Bildkomplettierung

Bildkomplettierung

[Hays and Efros, SIGGRAPH 2007, http://graphics.cs.cmu.edu/projects/scene-completion/scene-completion.pdf]

Allgemeine Beschreibung des Problems

- Gegeben einer Menge von N **hochdimensionalen Datenpunkten**

$$x_1, x_2, \ldots, x_N$$

- Distanzfunktion $d(x_i, x_j)$
- **Ziel**: Finde alle Paare (x_i, x_j) innerhalb einer Ähnlichkeitsgrenze s:

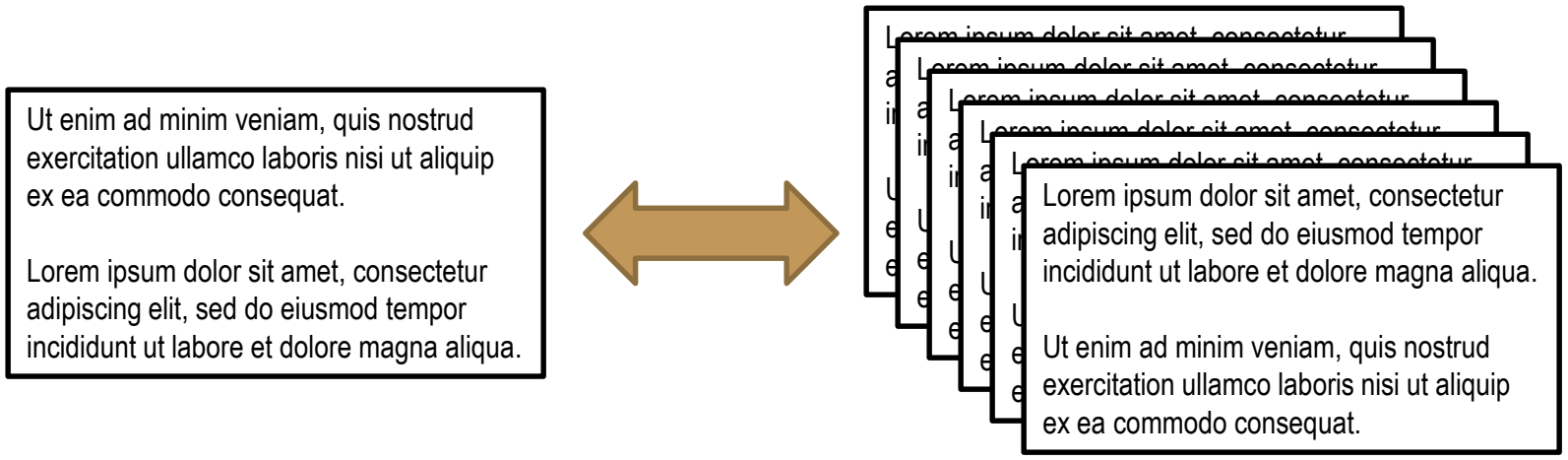
$$d(x_i, x_j) \leq s$$

- Naive Lösung: Vergleich aller Paare benötigt $O(N^2)$ Berechnungszeit
  - Bei $N = 10^6$ Punkten bedeutet dies $\frac{N(N-1)}{2} \approx 5 \cdot 10^{11}$ Vergleiche
  - Mit 10^6 Vergleichen pro Sekunde, benötigt man dazu etwa 5 Tage
  - Über 1 Jahr bei $N = 10^7$ Punkten

- Mit Locality Sensitive Hashing: $O(N)$ Berechnungszeit *möglich*

Beispiel: Ähnliche Textdokumente

Finde ähnliche Paare in einer *riesigen* Menge von Textdokumenten

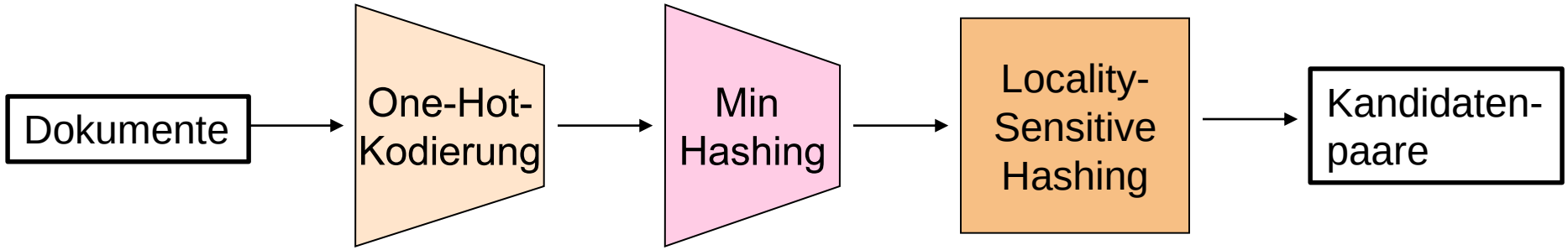


Probleme:

- Repräsentation als (hochdimensionaler) Datenpunkt
- Gleiche Teile der Dokumente werden in unterschiedlichen Reihenfolgen angezeigt
- Dokumente sind so groß/zahlreich, dass sie nicht in den Hauptspeicher passen
- Zu viele Dokumente zum Vergleichen aller Paare

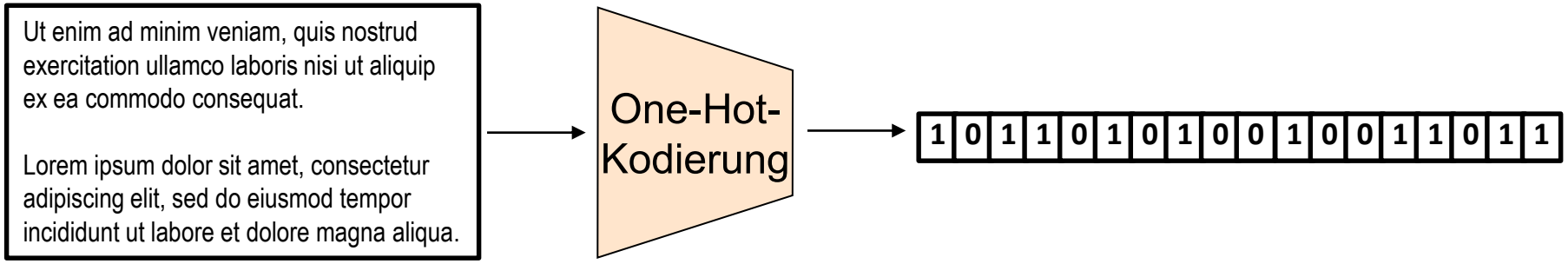
3 Schritte zum Finden ähnlicher Dokumente

1. **One-Hot-Kodierung**: Konvertierung der Dokumente in numerische Repräsentationen
2. **Min-Hashing**: Konvertierung langer (hochdimensionaler) Repräsentationen in kurze *ähnlichkeitserhaltende* Signaturen
3. **Locality-Sensitive Hashing:** Bestimmen von Paaren an Signaturen, die mit hoher Wahrscheinlichkeit sehr ähnlich sind



Inhaltsverzeichnis

Schritt 1: One-Hot-Kodierung



- Darstellung eines Textdokuments als **0-1-Array**
- Jedes Element des Arrays steht für ein **Merkmal**
  - 0 bedeutet, dass Merkmal nicht vorhanden ist
  - 1 bedeutet, dass Merkmal vorhanden ist
- Mehrere Möglichkeiten, Merkmale aus Textdokumenten zu extrahieren
- **Beispiel:** Merkmal = Wort (Bag-Of-Words)

John ist häufig übermüdet.

Lieber häufig übermüdet als ständig überwacht.

John	ist	ständig	übermüdet	lieber	häufig	als	überwacht
1	1	0	1	0	1	0	0
0	0	1	1	1	1	1	1

N-Gramme und K-Shingles

- Menge aller Wörter (Bag-Of-Words) berücksichtigt nicht die Anordnung der Wörter (zu Sätzen)

- **N-Gramm:** Folge von Wörtern
  - 2-Gramme: „lieber häufig“, „häufig übermüdet“, „übermüdet als“, „als ständig“, „ständig überwacht“
  - 3-Gramme: „ lieber häufig übermüdet“, „häufig übermüdet als“, „übermüdet als ständig“, „als ständig überwacht“

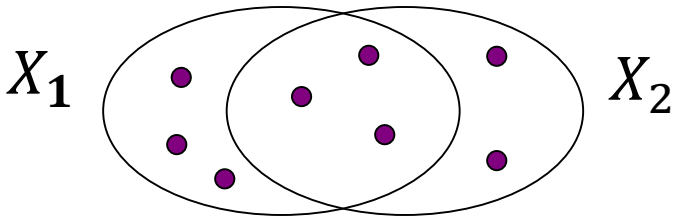
- **K-Shingle:** Teil einer Zeichenkette der Länge k
  - 2-Shingles: “li”, “ie”, “eb”, “be”, “er”, “r “, “ h”, “hä”, “äu”, “uf”, “fi”, “ig”, “g “, “ ü”, “üb”, …
  - 3-Shingles: “lie”, “ieb”, “ebe”, “ber”, “er ”, “r h“, “ hä”, “häu”, “äuf”, “ufi”, “fig”, “ig ”, …

Distanzmaß

- Distanz zwischen zwei Mengen X_1 und X_2: **Jaccarddistanz**

$$\mathrm{d}(X_1, X_2) = 1 - \frac{|X_1 \cap X_2|}{|X_1 \cup X_2|}$$

- Jaccarddistanz beschreibt den Anteil der Elemente, welche nicht in beiden Mengen vorkommen



$$d(X_1, X_2) = 5/8$$

Jaccarddistanz

- Berechnung über One-Hot-Kodierung
- Seien C_1 und C_2 die Kodierungen zweier Mengen
- Dann kann die Jaccarddistanz berechnet werden, indem die Elemente beider Kodierungen an jeder Position vergleicht:

$$\mathrm{d}(C_1, C_2) = \frac{\text{Anzahl der Positionen mit genauer einer Eins}}{\text{Anzahl der Positionen mit mind. einer Eins}}$$

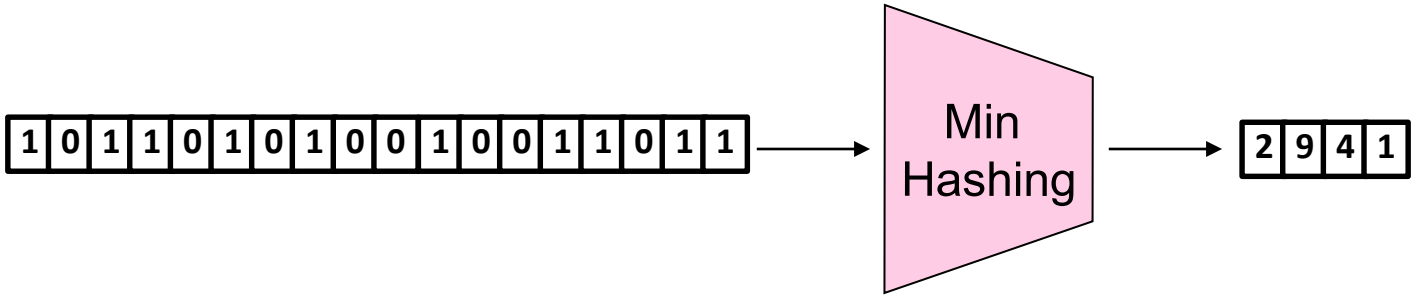
C_1	1	1	0	0	1	1
C_2	1	1	0	0	1	0
C_3	0	1	1	1	0	0

$$d(C_1, C_2) = \frac{1}{4} \quad d(C_1, C_3) = \frac{5}{6} \quad d(C_2, C_3) = \frac{4}{5}$$

Inhaltsverzeichnis

- Einleitung
- One-Hot-Kodierung
- Min-Hash Signaturen
- Locality Sensitive Hashing

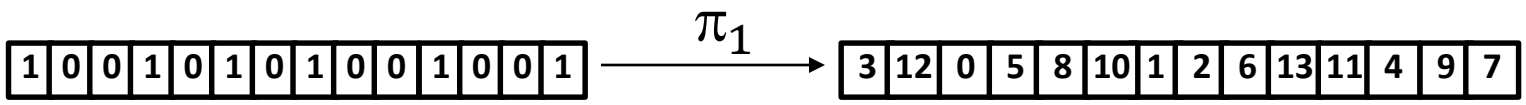
Schritt 2: Min-Hashing



- **Min-Hashing:** Konvertierung der Kodierungen in kurze *ähnlichkeitserhaltende* Signaturen
  - Signaturen sind kürzere Repräsentationen
  - Vergleich über Signaturen anstatt über Kodierungen
- *Idee: „Hash" die Kodierungen in Buckets, so dass die meisten ähnlichen Paare in dem gleichen Bucket landen*
- *Formal:* Finde eine Hash-Funktion h(·), so dass:
  - Falls $d(C_1,C_2)$ klein, dann $h(C_1) = h(C_2)$ mit hoher Wahrscheinlichkeit.
  - Falls $d(C_1,C_2)$ groß, dann $h(C_1) = h(C_2)$ mit niedriger Wahrscheinlichkeit.
- Hash-Funktion hängt vom Distanzmaß ab: **Min-Hashing** für Jaccard

Min-Hashing

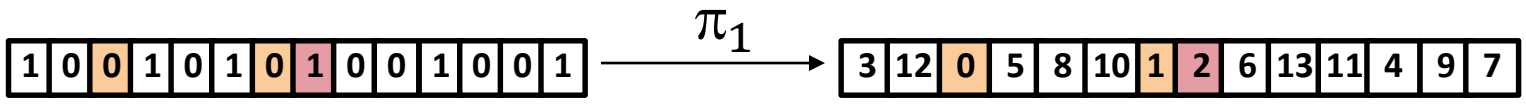
- Anwenden einer **Permutation** π_1 auf die Elemente der Kodierung



- **Min-Hash-Funktion** h_1 auf Kodierung C:

$$h_1(C) := \min_{i:C_i=1} \pi_1(i),$$

d.h. der Index der ersten Position der permutierten Kodierung mit einer 1



$$h_1(C) = 2$$

- **Signatur** einer Kodierung besteht aus den Min-Hash-Werten

$h_1, h_2, \ldots, h_n$ mehrerer unabhängiger Permutationen $\pi_1, \pi_2, \ldots, \pi_n$

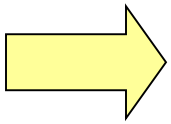
Min-Hashing: Beispiel

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2	0	4	0
0	1	0	1
3	0	5	0

Min-Hash: Zentrale Eigenschaft

Für zwei Mengen mit One-Hot-Kodierung C und D bezeichne $d(C, D)$ deren Jaccarddistanz. Für jede Permutation π gilt:

$$Pr[h_\pi(C) \neq h_\pi(D)] = d(C, D)$$

Begründung:

- Wir betrachten die kleinste Zahl aus π, die nicht auf eine Zeile (0,0) verweist
- Die Wahrscheinlichkeit $Pr[h_\pi(C) \neq h_\pi(D)]$ bezieht sich auf das Ereignis, dass diese Zahl auf eine Zeile mit nur einer Eins verweist: (0,1) oder (1,0)
- Da die Permutation zufällig, entspricht dies:

$$Pr[h_\pi(C) \neq h_\pi(D)] =$$

$$\frac{\text{Anzahl der Positionen mit genauer einer Eins}}{\text{Anzahl der Positionen mit mind. einer Eins}} = d(C, D)$$

π	C	D
2	0	0
3	1	1
0	0	0
1	0	1
6	1	1
5	1	0
4	0	1

Ähnlichkeit der Signaturen

- Für jede Permutation π gilt $Pr[h_\pi(C) \neq h_\pi(D)] = d(C, D)$
- Bei mehreren unabhängigen Permutationen $\pi_1, \pi_2, \ldots, \pi_n$, kann der Wert für $d(C, D)$ über die relative Häufigkeit von

$$h_{\pi_i}(C) \neq h_{\pi_i}(D)$$

geschätzt werden

- Je länger die Signatur, desto genauer die Schätzung

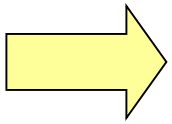
Min-Hashing: Beispiel

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2	0	4	0
0	1	0	1
3	0	5	0

Paar	Spalte	Signatur
1-2	1	1
1-3	0.5	2/3
1-4	6/7	1
2-3	1	1
2-4	0.25	0
3-4	1	1

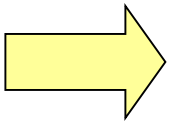
Min-Hashing: Berechnung

Permutation π

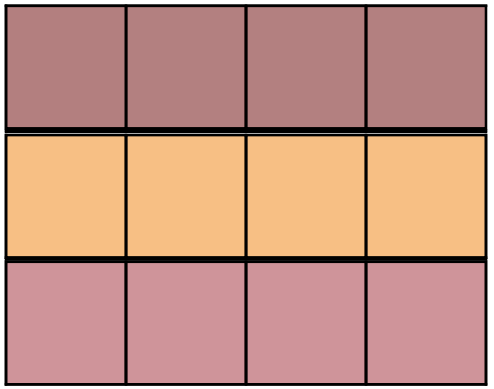
2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen



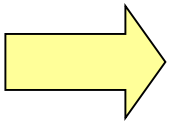
Min-Hashing: Berechnung

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2			
4			
3			

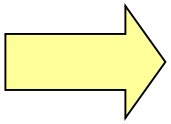
Min-Hashing: Berechnung

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2			**3**
2			**2**
3			**4**

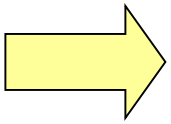
Min-Hashing: Berechnung

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2	**0**		**0**
2	**1**		**1**
3	**0**		**0**

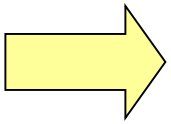
Min-Hashing: Berechnung

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2	0		0
2	1		1
3	0		0

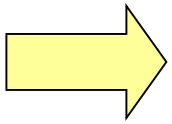
Min-Hashing: Berechnung

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2	0	**5**	0
0	1	**0**	1
3	0	**6**	0

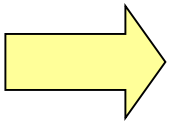
Min-Hashing: Berechnung

Permutation π

2	4	3
3	2	4
0	1	0
6	3	2
1	6	1
5	0	6
4	5	5

Matrix (Shingles x Dokumenten)

1	0	0	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0



Signaturen

2	0	**4**	0
0	1	0	1
3	0	**5**	0

Min-Hash: Implementierung

- Tatsächliche Permutationen wären zu umfangreich (Speicherplatz)
- **Simulation einer Permutation** über Zufallshashfunktionen
  - Zufällige Abbildung der Zahlen 1, …, N auf Zahlen 1, …, N
  - z.B. $h(x) = ((a \cdot x + b) \bmod p) \bmod N$, wobei a, b Zufallszahlen und p eine Primzahl mit p > N
- Zufallshashfunktionen $h_1, h_2, \ldots, h_n$
- Initiale **Signaturmatrix**: $SIG(i, c) = \infty$ für alle i, c
- **Algorithmus**: Für jede Zeile r der Dokumentenmatrix
  - Berechne $h_1(r), h_2(r), \ldots, h_n(r)$
  - Für jede Spalte c mit 1: $SIG(i, c) \leftarrow \min(SIG(i, c), h_i(r))$ für alle $i = 1, \ldots, n$

Implementierung: Beispiel

Zeile	C_1	C_2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

h(x) = x mod 5

g(x) = (2x+1) mod 5

	$SIG(i, C_1)$	$SIG(i, C_2)$
	∞	∞
	∞	∞
$h(1) = 1$	**1**	∞
$g(1) = 3$	**3**	∞
$h(2) = 2$	1	**2**
$g(2) = 0$	3	**0**
$h(3) = 3$	1	2
$g(3) = 2$	**2**	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	**0**
$g(5) = 1$	2	0

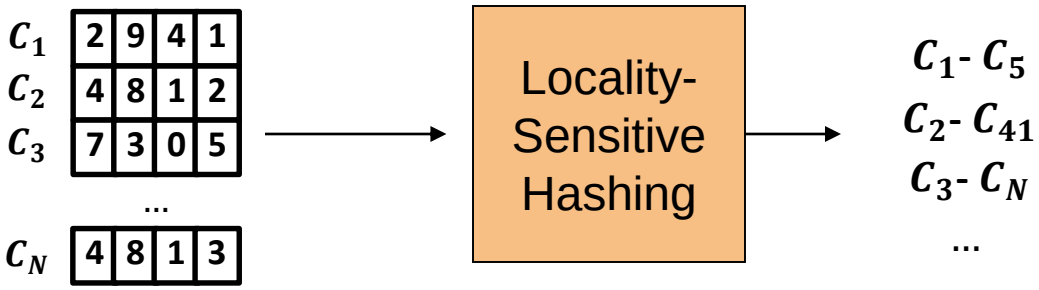
Signaturmatrix

Inhaltsverzeichnis

- Einleitung
- One-Hot-Kodierung
- Min-Hash Signaturen
- Locality Sensitive Hashing

Schritt 3: Locality Sensitive Hashing

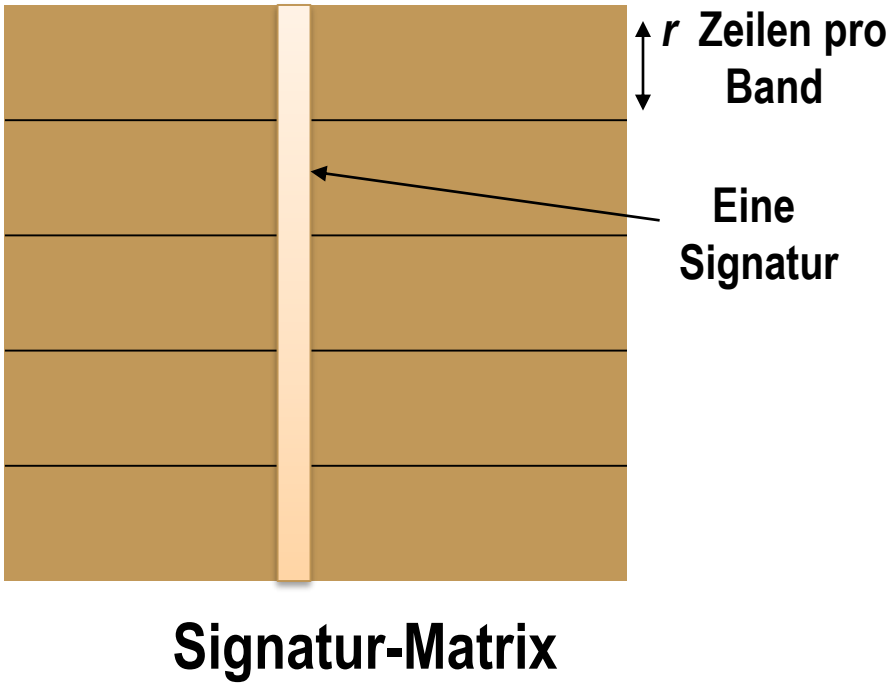
Locality Sensitive Hashing (LSH): Beschränkung auf Paare von Signaturen, die höchst wahrscheinlich ähnlich sind



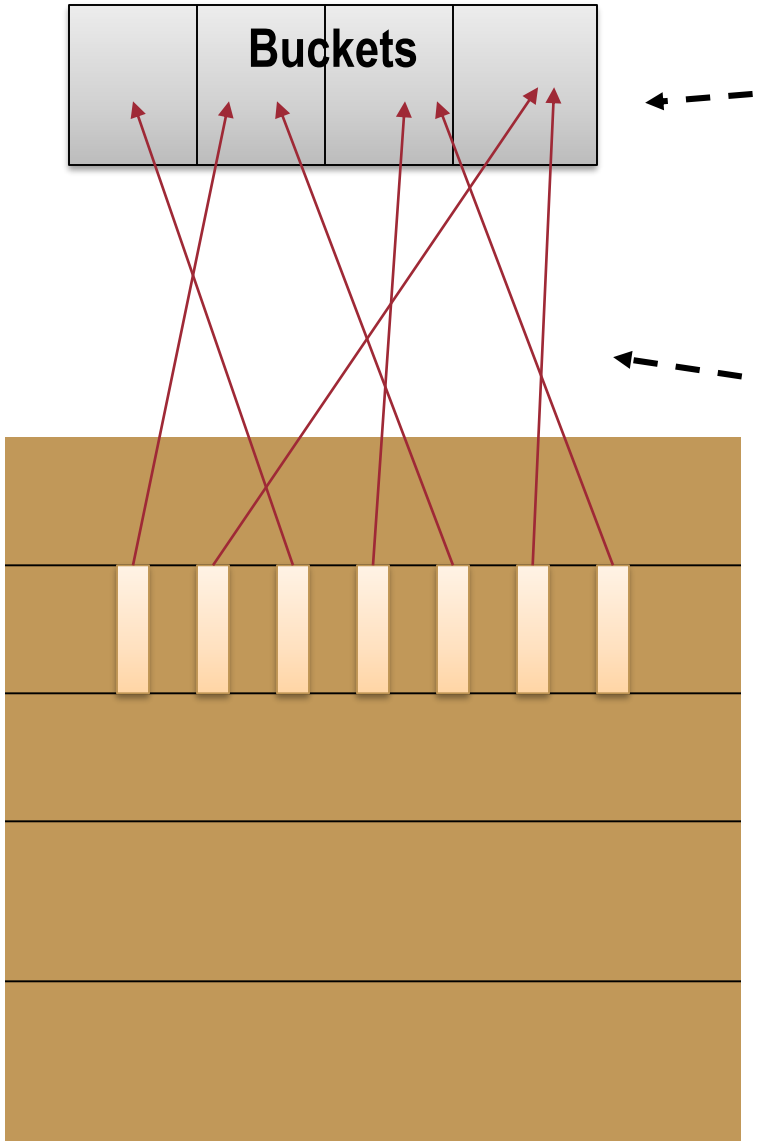
- Nur die Paare von Signaturen, die über LSH ausgewählt wurden, werden auf Ähnlichkeit untersucht (Schätzung der Jaccarddistanz)
- **Vorteil**: Im Idealfall wird nur ein kleiner Bruchteil von Paaren untersucht
- **Nachteil**: Es gibt **False Negatives**
  - Ähnliche Paare, die nicht entdeckt wurden
  - Die Rate der False Negatives gilt es, so klein wie möglich zu halten

LSH

- **Ziel**: Dokumentenpaare mit Jaccarddistanz kleiner als ein *Schwellenwert s* (z.B. s = 0.2)
- **Idee**: Verwende Hash-Funktion, die ähnliche Dokumente in gleiche Buckets sortiert und unähnliche Dokumente in unterschiedliche Buckets
- **Effektive Methode** (zum Beispiel):
  - Teilung der Signaturmatrix in b Bänder mit jeweils r Reihen
  - Eine Hash-Funktion für jedes Band, welche einen Vektor aus r Zahlen auf eine große Anzahl an Buckets verteilt
  - Kandidaten sind zwei Signaturen, die mind. einmal in den gleichen Bucket sortiert wurden
  - Einstellung von b und r zur Optimierung



Hashing der Bänder



Spalten 2 und 6 sind in diesem Band *wahrscheinlich* gleich

Spalten 6 und 7 sind hier *garantiert* unterschiedlich

Annahme: Es gibt genügend Buckets, so dass zwei Spalten mit hoher Wahrscheinlichkeit nur dann in den gleichen Bucket sortiert werden, wenn sie in einem Band identisch sind

Hash-Funktion: Beispiel

- Man benötigt eine Hash-Funktion pro Band
- Wie findet man genügend Hash-Funktionen?

1. MurmurHash mit verschiedenen Seeds
   - Original in C++: https://github.com/aappleby/smhasher/wiki
   - Java Implementierung: z.B. in Guava (https://github.com/google/guava)
2. Beliebige Hash-Funktion und XOR mit Zufallszahl
   - Liste mit Hash-Funktionen: https://en.wikipedia.org/wiki/List_of_hash_functions
   - Auch unsichere Hash-Funktionen wie MD5 können verwendet werden
   - Java Implementierung: z.B. in Guava (https://github.com/google/guava/wiki/HashingExplained)
   - XOR in Java:

```
int a = 60; /* 60 = 0011 1100 */
int b = 13; /* 13 = 0000 1101 */
int c = a ^ b; /* 49 = 0011 0001 */
```

Beispiel

- Signaturmatrix mit 100 Zeilen
- Setze b = 20 und r = 5

Fall 1: 2 Dokumente mit d(C, D) = 0.2

- Wahrscheinlichkeit, dass C und D in einem bestimmten Band identisch: $(1 - 0.2)^5 = 0.328$
- Wahrscheinlichkeit, dass C und D in keinem Band identisch: $(1 - 0.328)^{20} = 0.00035$
- d.h. einer von 3000 Paaren mit 80%-Ähnlichkeit werden nicht entdeckt (False Negative)

Fall 2: 2 Dokumente mit d(C, D) = 0.7

- Wahrscheinlichkeit, dass C und D in einem bestimmten Band identisch: $(0.3)^5 = 0.00243$
- Wahrscheinlichkeit, dass C und D in mind. einem Band identisch: $1 - (1 - 0.00243)^{20} = 0.0474$
- d.h. 4.74% der Paaren mit 30%-Ähnlichkeit werden zu Kandidaten (False Positives)

LSH: Analyse

- Angenommen $t = 1 - d(C, D)$, b Bänder mit jeweils r Zeilen
- Wahrscheinlichkeit, dass C und D in mind. einem Band identisch:

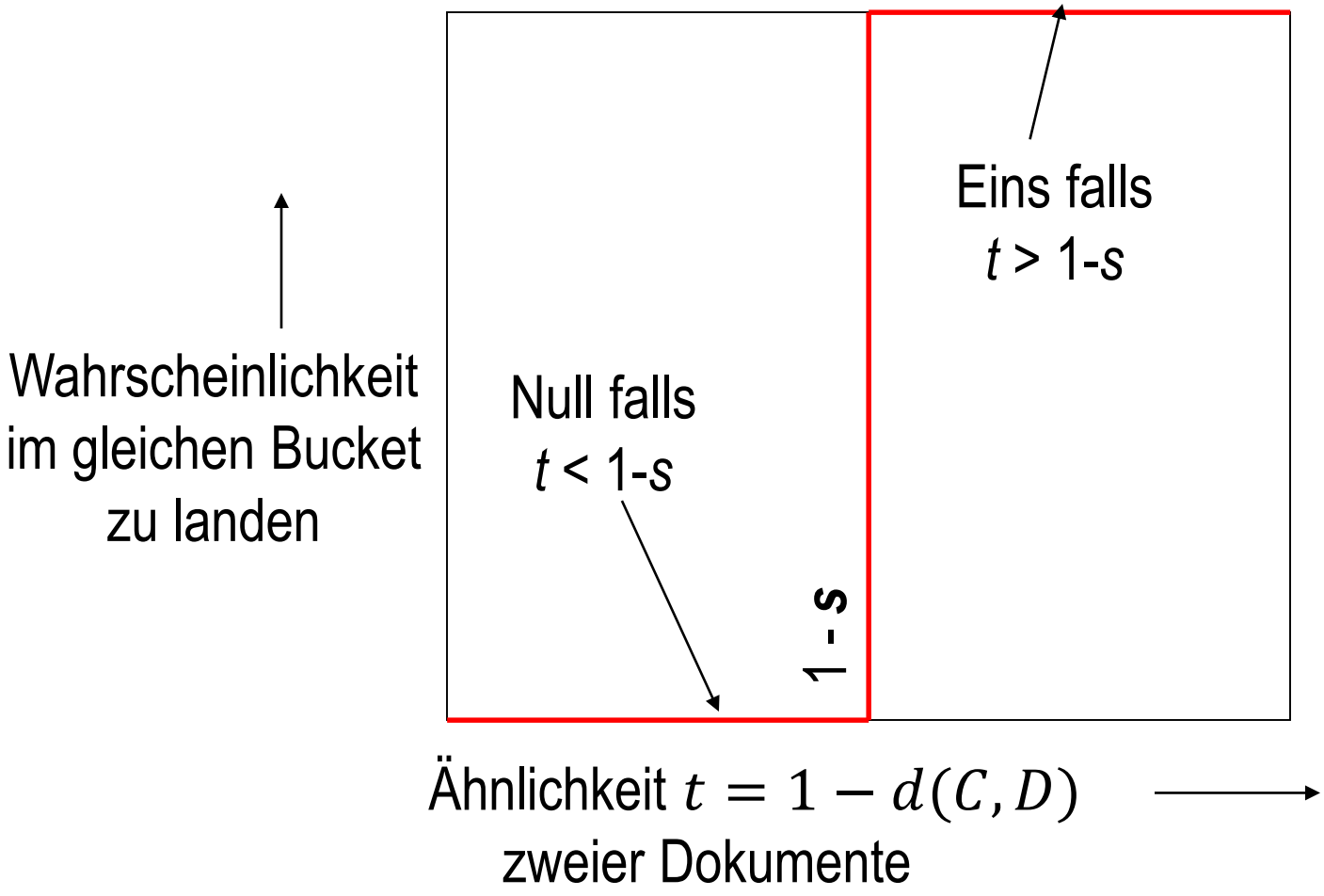
$$1 - (1 - t^r)^b$$

- **Wähle (2 aus 3):**
  - Die Anzahl der Min-Hashes (Zeilen der Signatur-Matrix)
  - Die Anzahl der Bänder b
  - Die Anzahl der Reihen pro Band r

  um die Raten der False Positives und False Negatives anzugleichen
- **Beispiel:** Bei b = 10 und r = 10 anstatt b = 20 und r = 5 würde es weniger False Positives aber mehr False Negatives geben

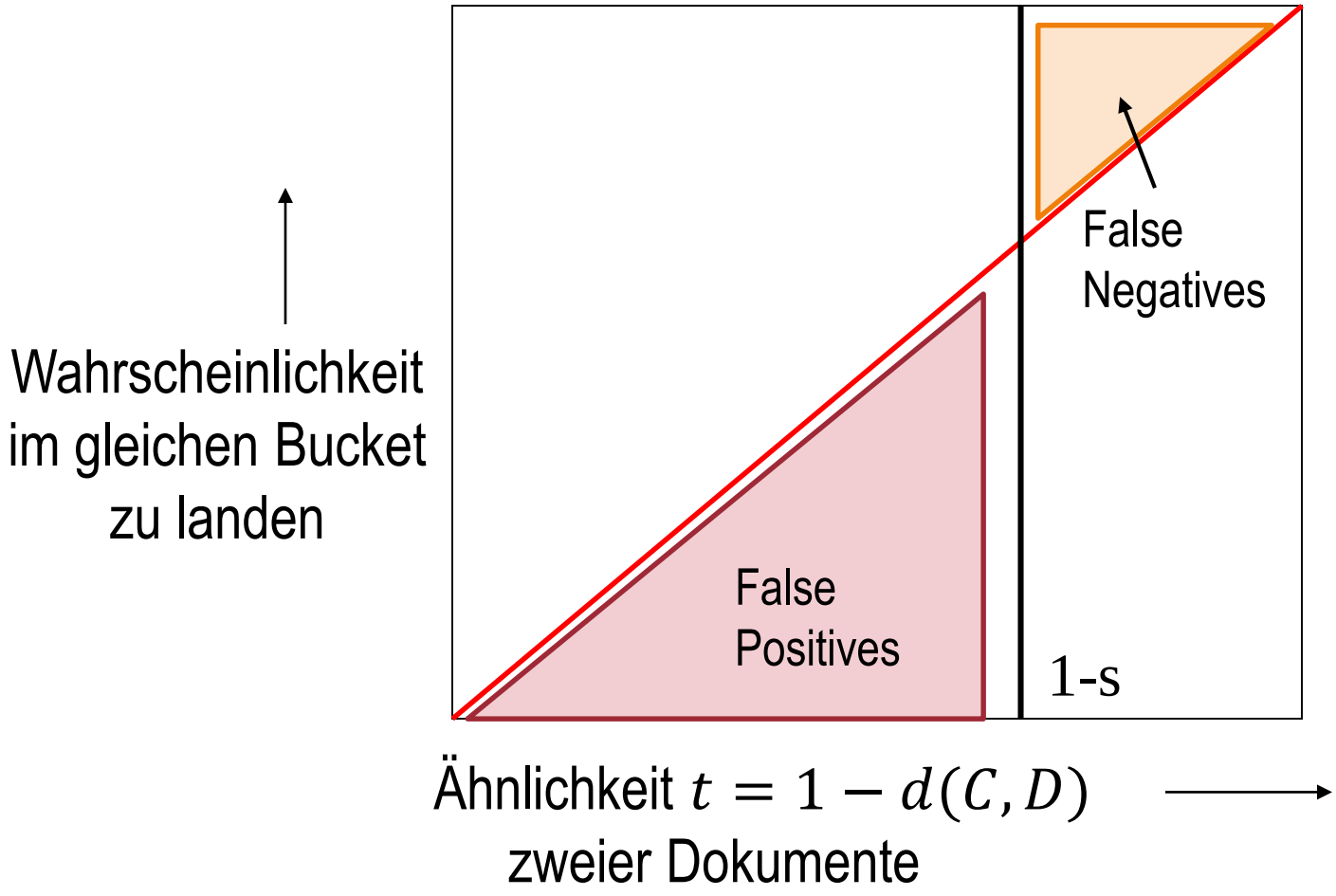
LSH: Analyse

Idealfall



LSH: Analyse

1 Zeile und 1 Band: $1 - (1 - t^r)^b = t$



LSH: Analyse

r Zeilen und b Bänder