Datenbanksysteme

Wintersemester 2016/17

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Ähnlichkeitssuche in hohen Dimensionen

Mehrdimensionale Indexstrukturen

Bislang im B-Baum gesehen: Eindimensionale Schlüssel.

In vielen Anwendungen ist die Anzahl der Dimensionen höher, was tun?

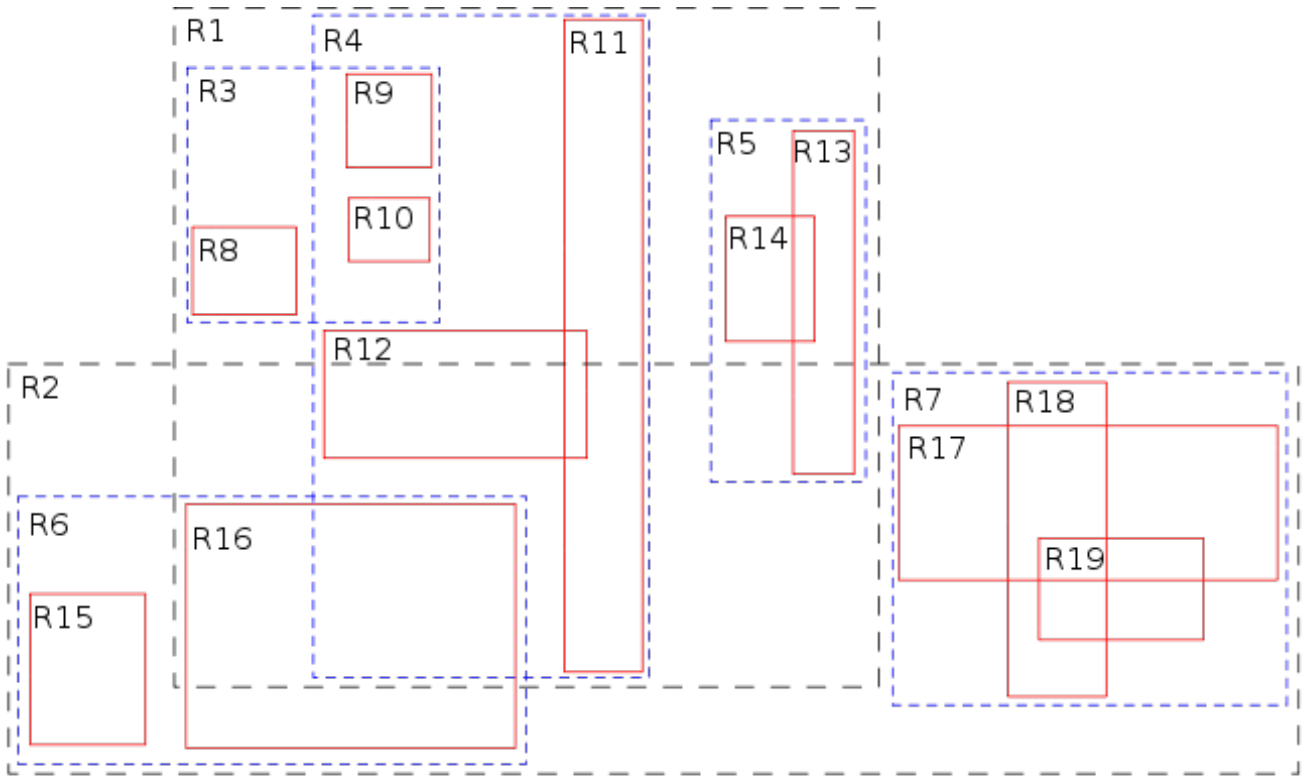
R Baum

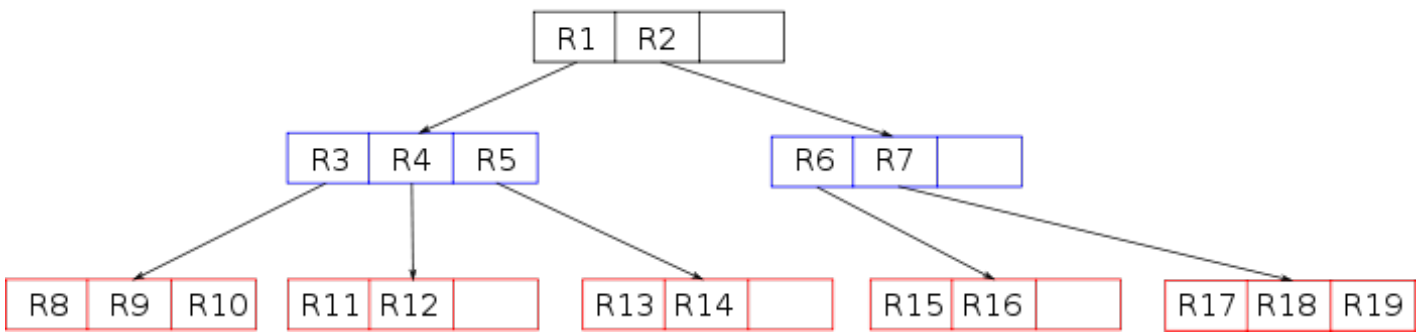
- R steht für Rechteck/Rectangle
- Knoten definieren minimale Rechtecke, die die enthaltenen Rechtecke umschließen
- Balanciert. Aufbau (Algorithmus) ähnlich zum B+ Baum
- Überlappung der MBRs (=Minimum Bounding Rectangles)
- Überlappung führt zu Ineffizienz während der Anfrage.

Wir werden uns den R Baum noch im Detail anschauen.

R Baum: Beispiel

Quelle: wikipedia



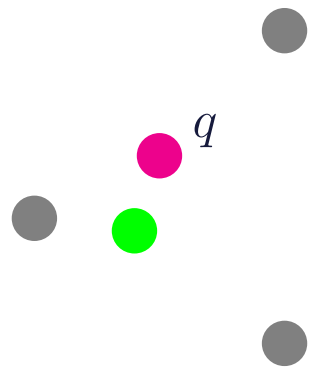


K-Nächste-Nachbarn-Suche (K-Nearest-Neighbor Search)

Objekte werden charakterisiert anhand Menge relevanter Attribute (Features), z.B. **Punkte in einem d-dimensionalen Raum**.
Z.B. Städte und ihre X-Y-Positionen im 2-dimensionalen Raum.

K-Nearest Neighbor (KNN) Anfragen

- Für ein Anfrage-Objekt q aus einem d-dimensionalen Raum: **Finde den Punkt, der am nächsten an q liegt, für eine gegebene Distanzfunktion**
- **K-Nearest Neighbor Search: Suche die K nächsten Nachbar-Punkte**



Der Fluch der Dimensionalität

(Englisch: **Curse of Dimensionality**)

- **Mit wachsender Dimensionalität wird es zunehmend schwieriger den Raum geeignet zu indexieren.**
- Beispiel: Betrachten Rechteck (2d) mit Kantenlänge 1.
  - Kleines Rechteck mit Kantenlänge $0{,}1$ beschreibt 1% des großen Rechtecks
  - Um 1% eines Würfels (3d) mit gleicher Kantenlänge 1 zu beschreiben braucht der Kleine eine Kantenlänge von $0{,}21$!
  - In 100 Dimensionen: Kantenlänge von $0{,}954$ erforderlich!
- **Abstand zwischen nächstem Objekt und maximal entfernten Objekt nähert sich an.**

$$lim_{d\to\infty} \frac{d_{max} - d_{min}}{d_{min}} \to 0$$

Ab einer bestimmten Anzahl von Dimensionen ist der R Baum ineffektiv.

Wirklich hohe Dimensionen

Ähnlichkeitssuche (Similarity Search) in Foto-Kollektion

- **Gegeben: Ein Foto**
- **Gesucht: Die k ähnlichsten Fotos der Kollektion.**

Beispiel:

Anfrageobjekt

Treffer

- Eigenschaften (Features) werden aus den Fotos extrahiert.
- Z.B: Farbverteilung, Kanten, ... (MPEG-7 Features)
- **Anzahl der Features wird sehr schnell sehr groß:** $>> 10$

Locality Sensitive Hashing (LSH)

Locality Sensitive Hashing (LSH) ist eine approximative Methode, die es erlaubt in sehr hohen Dimensionen KNN Anfragen zu bearbeiten.

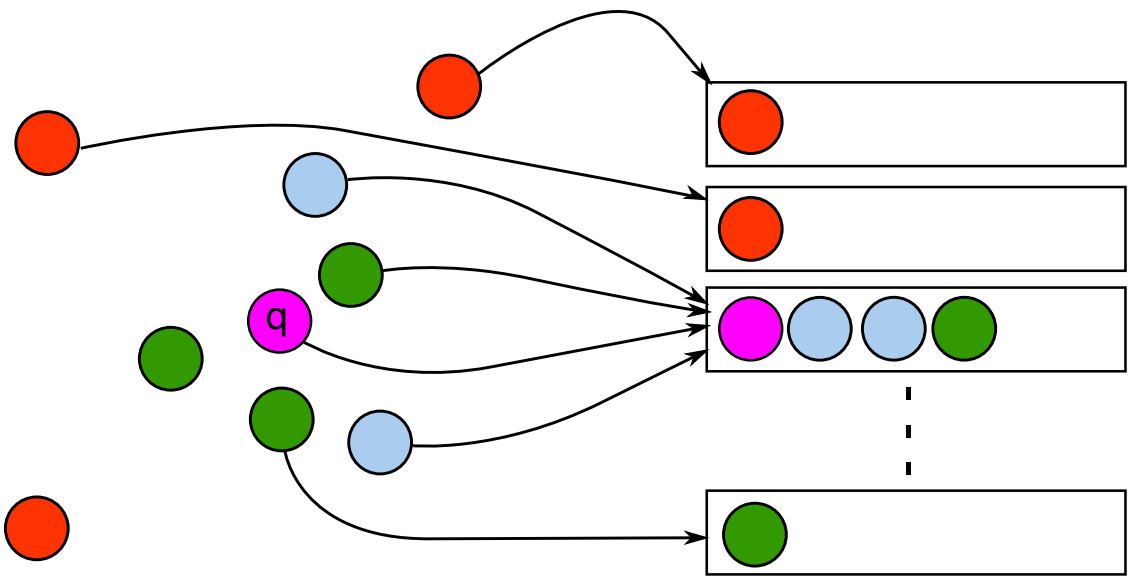
Erinnern Sie sich an Hash-Verfahren:

- Hash-Funktion h platziert Objekte (Tupel) in Hash-Buckets.
- Z.B. basierend auf Schlüssel des Objekts (wie MatrNr für Relation Studenten).
- Schneller Zugriff: Suche nach Objekt x? Schaue in Bucket $h(x)$ nach.
- Wichtig dabei, Kollisionen sind zu reduzieren/vermeiden, wegen (teurer) Handhabung von "überlaufenden" Buckets

Locality Sensitive Hashing (LSH): Idee

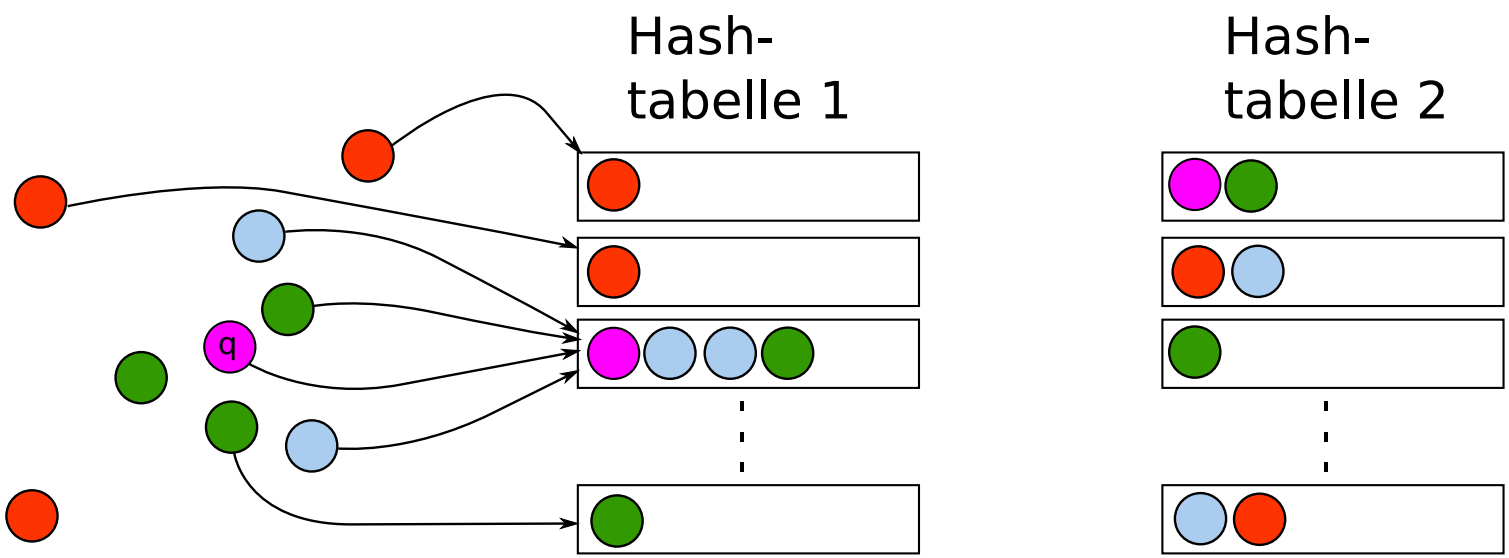
Idee:

- **Kollisionen:** ähnliche Objekte sollen(!) in den gleichen Hash-Bucket fallen.
- Für Anfrage Objekt q wird in dem Bucket in den q fällt nachgeschaut.



Locality Sensitive Hashing (LSH): Idee (2)

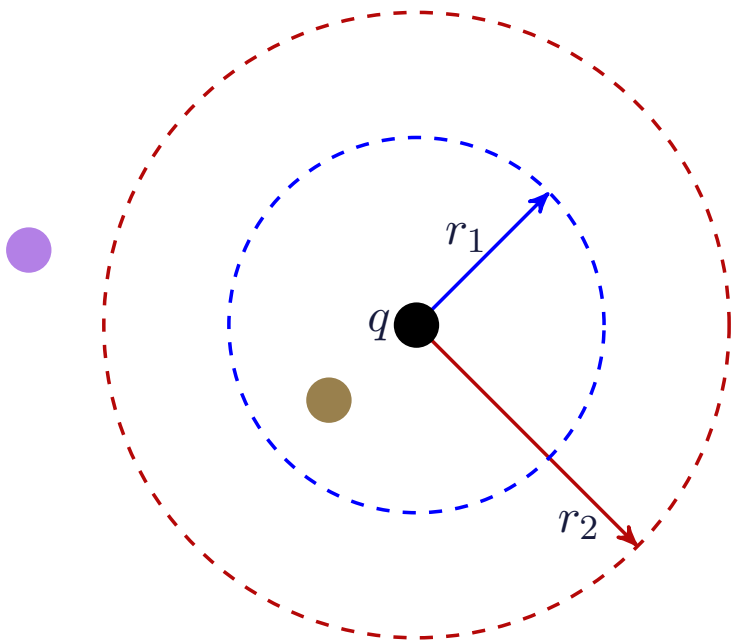
- Um die Wahrscheinlichkeit von Kollisionen zu erhöhen werden mehrere Hashtabellen aufgebaut.



Was bedeutet es Locality Sensitive zu sein?

Eine Familie von Hashfunktionen $H = \{h : S \rightarrow U\}$ heißt (r_1,r_2,p_1,p_2)-sensitive falls die folgenden beiden Bedingungen für beliebige Punkte $\mathbf{q},\mathbf{v} \in S$ gelten:

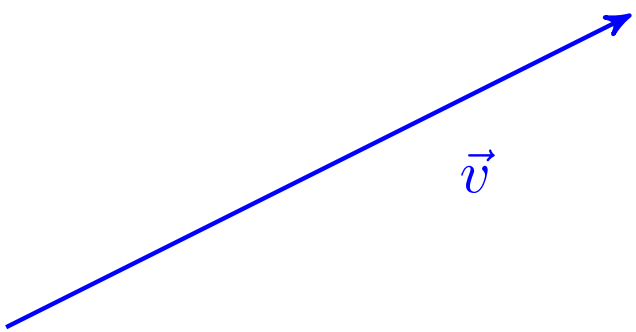
- if $dist(\mathbf{q},\mathbf{v}) \leq r_1$ dann $Pr_H(h(\mathbf{q}) = h(\mathbf{v})) \geq p_1$
- if $dist(\mathbf{q},\mathbf{v}) > r_2$ dann $Pr_H(h(\mathbf{q}) = h(\mathbf{v})) \leq p_2$



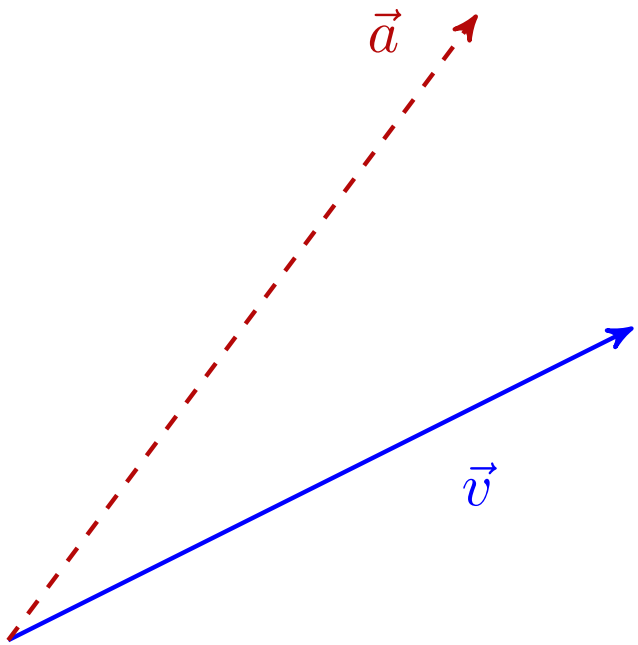
Wichtig: Falls $r_1 < r_2$ und $p_1 > p_2$: ähnliche Objekte bekommen häufiger den gleichen Hash-Wert, im Vergleich zu weniger ähnlichen.

d Dimensionaler Vektorraum: Idee für Hashfunktion

- Gegeben Vektor $\vec{v}$ im d-dimensionalen Raum.
- Wie kann man $\vec{v}$ mit weniger als d Werten beschreiben?

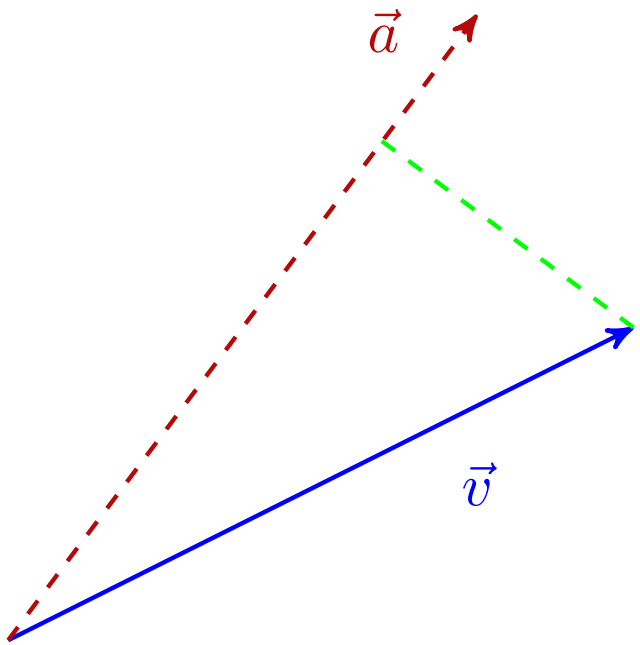


d Dimensionaler Vektorraum: Idee für Hashfunktion



- Gegeben Vektor $\vec{v}$ im d-dimensionalen Raum.
- Wie kann man $\vec{v}$ mit weniger als d Werten beschreiben?
- Nehme weiteren Vector $\vec{a}$ hinzu.

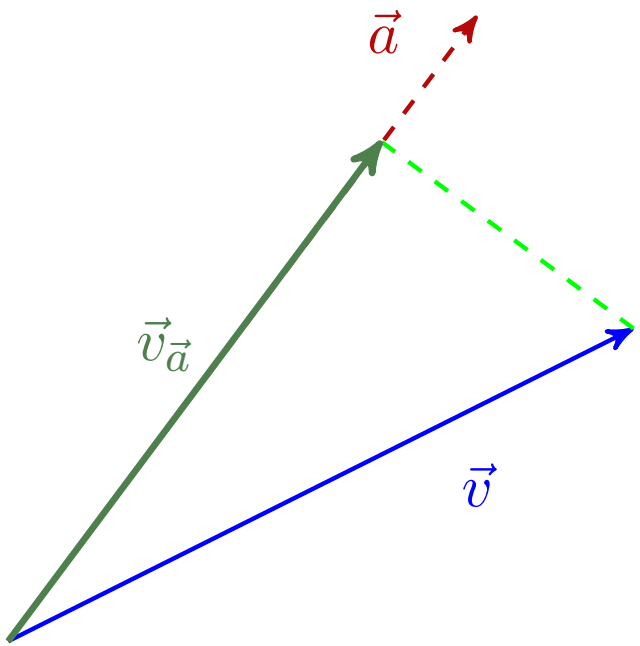
d Dimensionaler Vektorraum: Idee für Hashfunktion



- Gegeben Vektor $\vec{v}$ im d-dimensionalen Raum.
- Wie kann man $\vec{v}$ mit weniger als d Werten beschreiben?
- Nehme weiteren Vector $\vec{a}$ hinzu.
- Betrachte Skalarprodukt

$$\vec{a} \cdot \vec{v}$$

d Dimensionaler Vektorraum: Idee für Hashfunktion

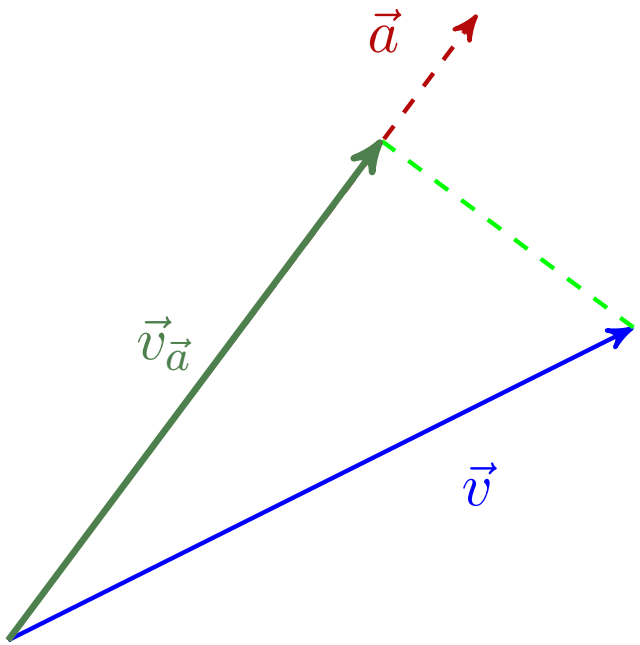


- Gegeben Vektor $\vec{v}$ im d-dimensionalen Raum.
- Wie kann man $\vec{v}$ mit weniger als d Werten beschreiben?
- Nehme weiteren Vector $\vec{a}$ hinzu.
- Betrachte Skalarprodukt

$$\vec{a} \cdot \vec{v}$$

- Idee: ähnliche Vektoren $\vec{v}_i$ haben ähnliche Werte $\vec{a} \cdot \vec{v}_i$

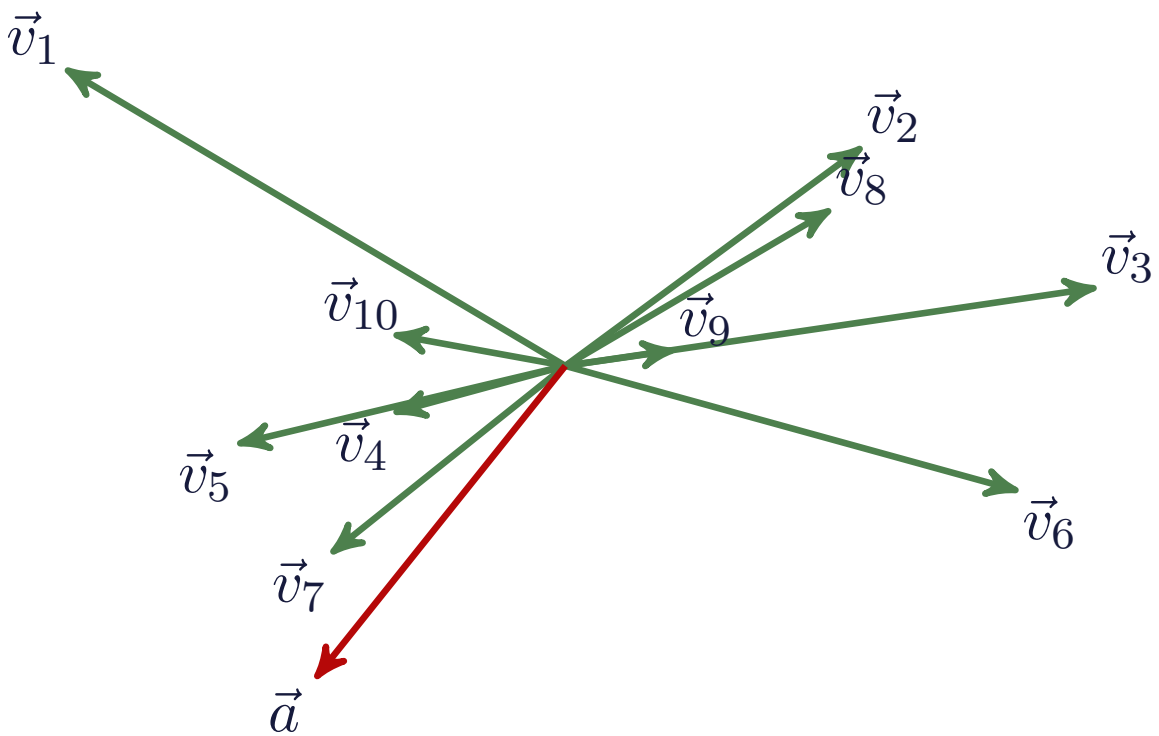
d Dimensionaler Vektorraum: Idee für Hashfunktion



- Gegeben Vektor $\vec{v}$ im d-dimensionalen Raum.
- Wie kann man $\vec{v}$ mit weniger als d Werten beschreiben?
- Nehme weiteren Vector $\vec{a}$ hinzu.
- Betrachte Skalarprodukt

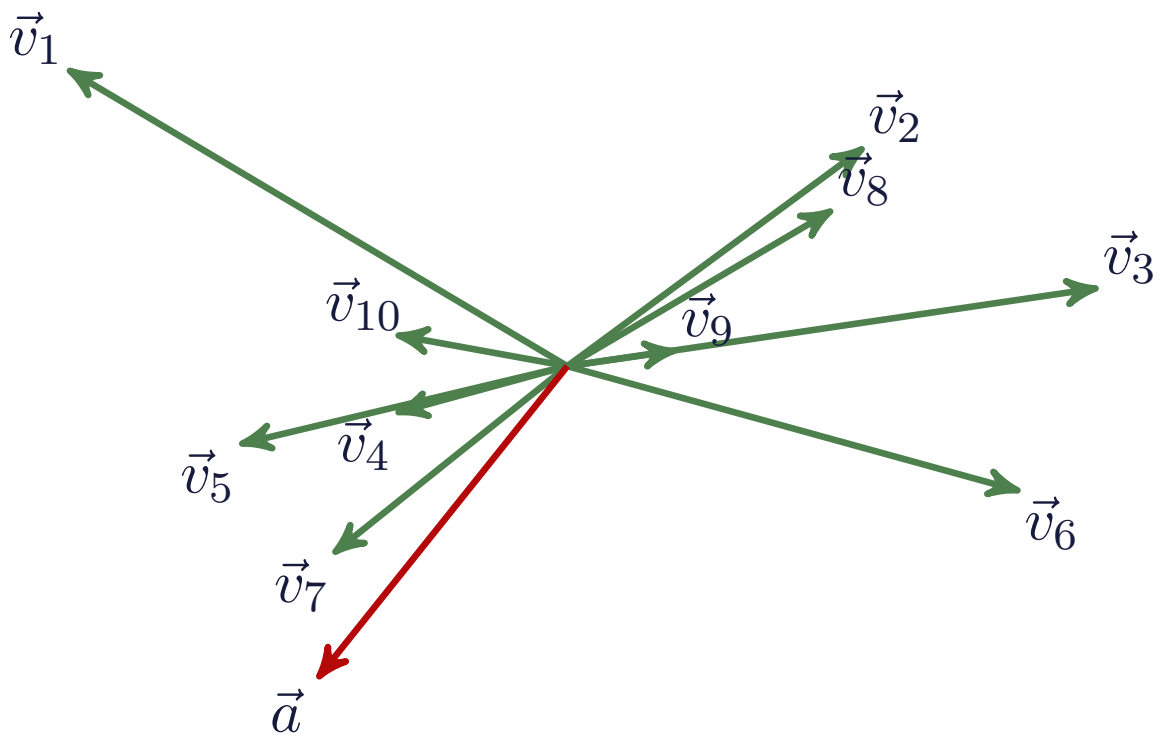
$$\vec{a} \cdot \vec{v}$$

- Idee: ähnliche Vektoren $\vec{v}_i$ haben ähnliche Werte $\vec{a} \cdot \vec{v}_i$
  Also, erste Idee für Hashfunktion: $h(\vec{v}) := \lfloor \vec{a} \cdot \vec{v}_i \rfloor$



$$\vec{v_1} = (-2.1, -1.1)$$
$$\vec{v_2} = (2.4, -1.0)$$
$$\vec{v_3} = (-1.6, -0.5)$$
$$\vec{v_4} = (-3.9, -0.9)$$
$$\vec{v_5} = (-1.0, -0.2)$$
$$\vec{v_6} = (-2.6, 0.0)$$
$$\vec{v_7} = (-0.7, 1.2)$$
$$\vec{v_8} = (1.2, -0.7)$$
$$\vec{v_9} = (-2.6, 0.5)$$
$$\vec{v_{10}} = (-0.8, 1.1)$$

$$\vec{a} = (-1.6, -2)$$



Mit Hashfunktion

$$h(v) = \lfloor \vec{a} \cdot \vec{v} \rfloor$$

erhalten wir

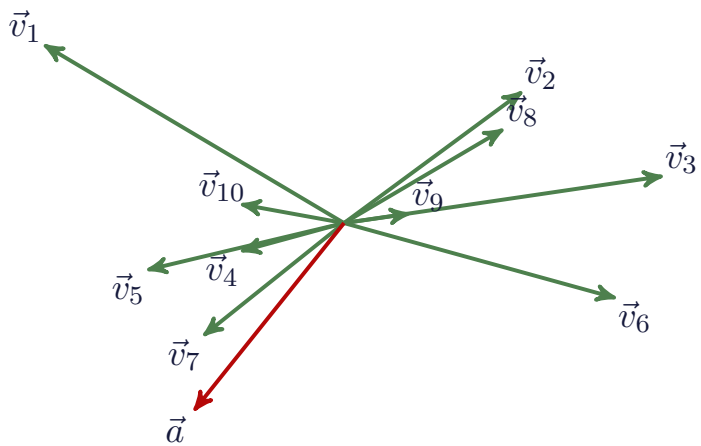
$$h(\vec{v}_1) = 1$$
$$h(\vec{v}_2) = -6$$
$$h(\vec{v}_3) = -7$$
$$h(\vec{v}_4) = 2$$
$$h(\vec{v}_5) = 4$$
$$h(\vec{v}_6) = -4$$
$$h(\vec{v}_7) = 4$$
$$h(\vec{v}_8) = -5$$
$$h(\vec{v}_9) = -2$$
$$h(\vec{v}_{10}) = 1$$

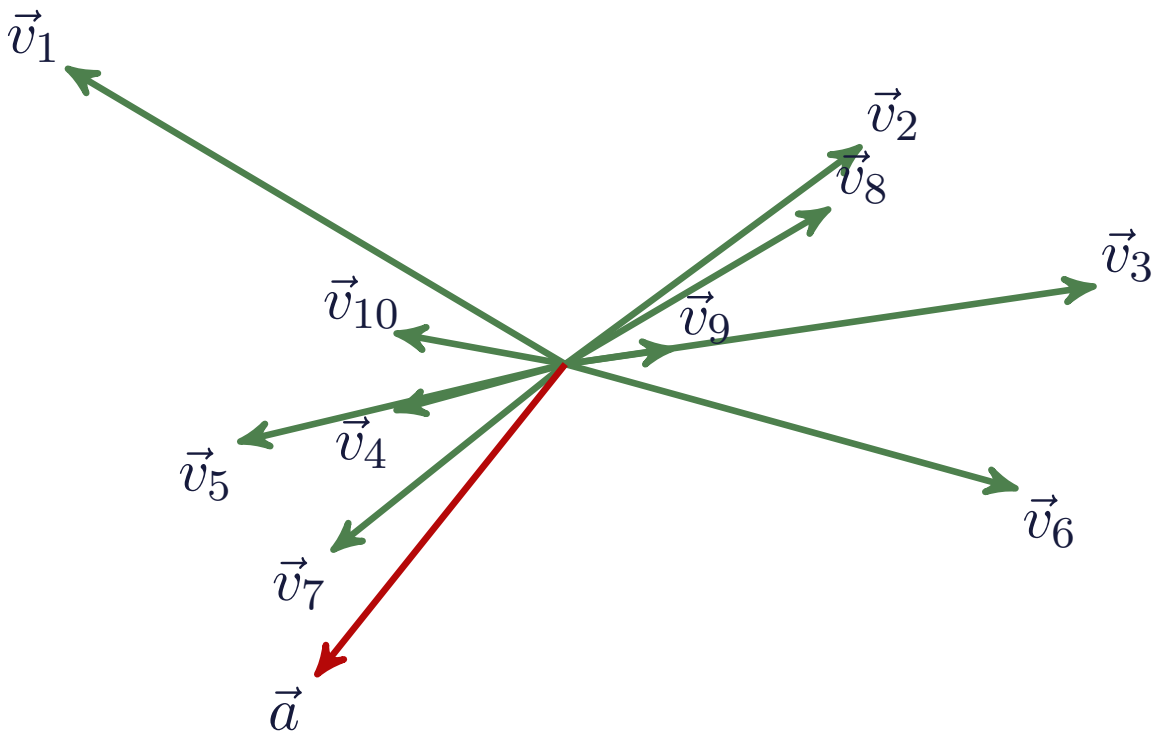
Wir interpretieren die Hashwerte als Label der Buckets, in die wir die Objekte (Vektoren) stecken....

Bucket -7	Bucket -6	Bucket -5
$\vec{v}_3$	$\vec{v}_2$	$\vec{v}_8$
Bucket -4	**Bucket -3**	**Bucket -2**
$\vec{v}_6$		$\vec{v}_9$
Bucket -1	**Bucket 0**	**Bucket 1**
		$\vec{v}_1$ $\vec{v}_{10}$
Bucket 2	**Bucket 3**	**Bucket 4**
$\vec{v}_4$		$\vec{v}_5$ $\vec{v}_7$
Bucket 5	**Bucket 6**	**Bucket 7**

$$h(\vec{v}_1) = 1$$
$$h(\vec{v}_2) = -6$$
$$h(\vec{v}_3) = -7$$
$$h(\vec{v}_4) = 2$$
$$h(\vec{v}_5) = 4$$
$$h(\vec{v}_6) = -4$$
$$h(\vec{v}_7) = 4$$
$$h(\vec{v}_8) = -5$$
$$h(\vec{v}_9) = -2$$
$$h(\vec{v}_{10}) = 1$$

Kaum Kollisionen, viele leere Buckets. Was können wir tun?





Betrachte nun Hashfunktion

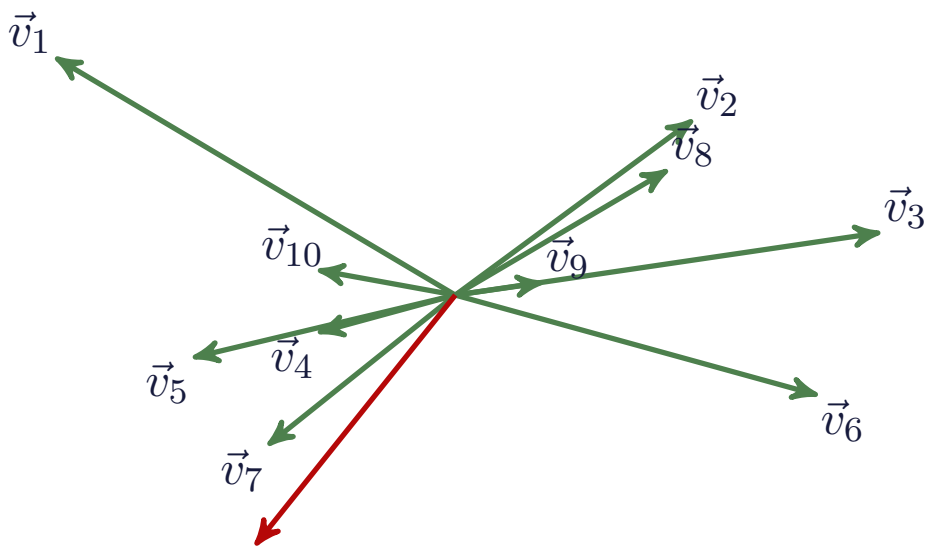
$$h(v) = \lfloor \frac{\vec{a} \cdot \vec{v}}{2} \rfloor$$

$h(\vec{v}_1) = 0$
$h(\vec{v}_2) = -3$
$h(\vec{v}_3) = -4$
$h(\vec{v}_4) = 1$
$h(\vec{v}_5) = 2$
$h(\vec{v}_6) = -2$
$h(\vec{v}_7) = 2$
$h(\vec{v}_8) = -3$
$h(\vec{v}_9) = -1$
$h(\vec{v}_{10}) = 0$

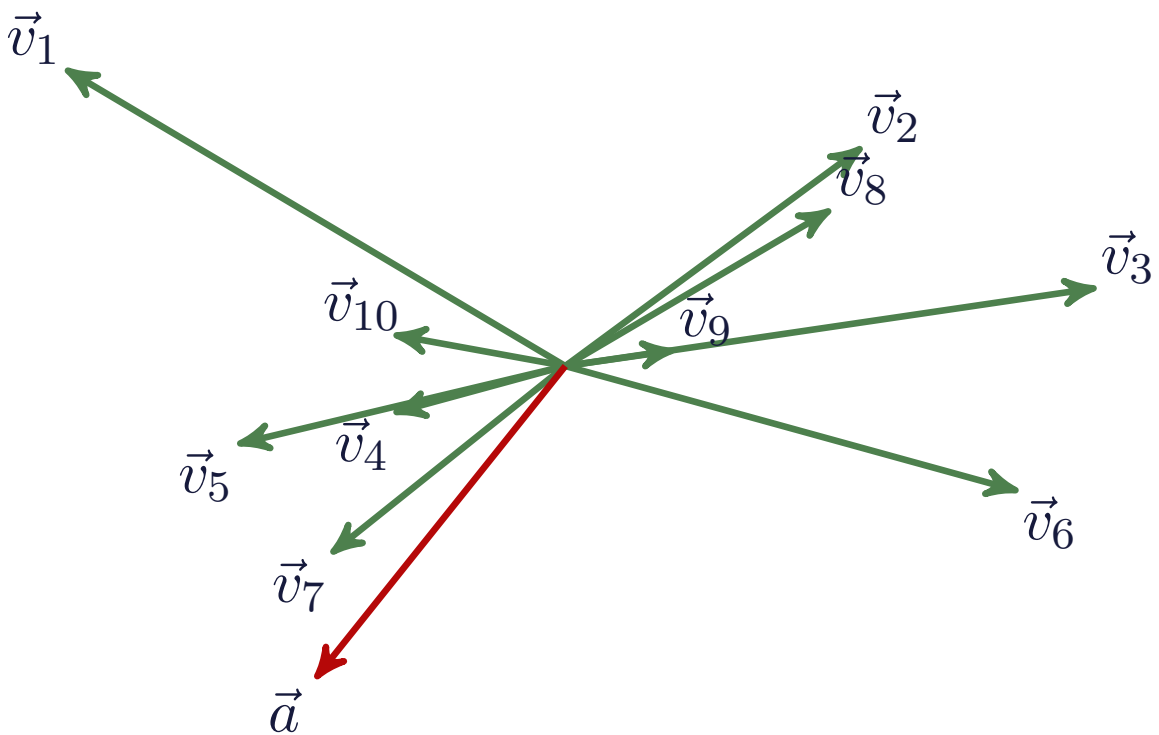
Bucket -4	Bucket -3	Bucket -2
$\vec{v}_3$	$\vec{v}_2$ $\vec{v}_8$	$\vec{v}_6$
Bucket -1	**Bucket 0**	**Bucket 1**
$\vec{v}_9$	$\vec{v}_1$ $\vec{v}_{10}$	$\vec{v}_4$
Bucket 2	**Bucket 3**	**Bucket 4**
$\vec{v}_5$ $\vec{v}_7$		

Hashwerte:

$h(\vec{v}_1) = 0$
$h(\vec{v}_2) = -3$
$h(\vec{v}_3) = -4$
$h(\vec{v}_4) = 1$
$h(\vec{v}_5) = 2$
$h(\vec{v}_6) = -2$
$h(\vec{v}_7) = 2$
$h(\vec{v}_8) = -3$
$h(\vec{v}_9) = -1$
$h(\vec{v}_{10}) = 0$



Wir sehen den Effekt: Weniger Buckets, mehr Kollisionen.



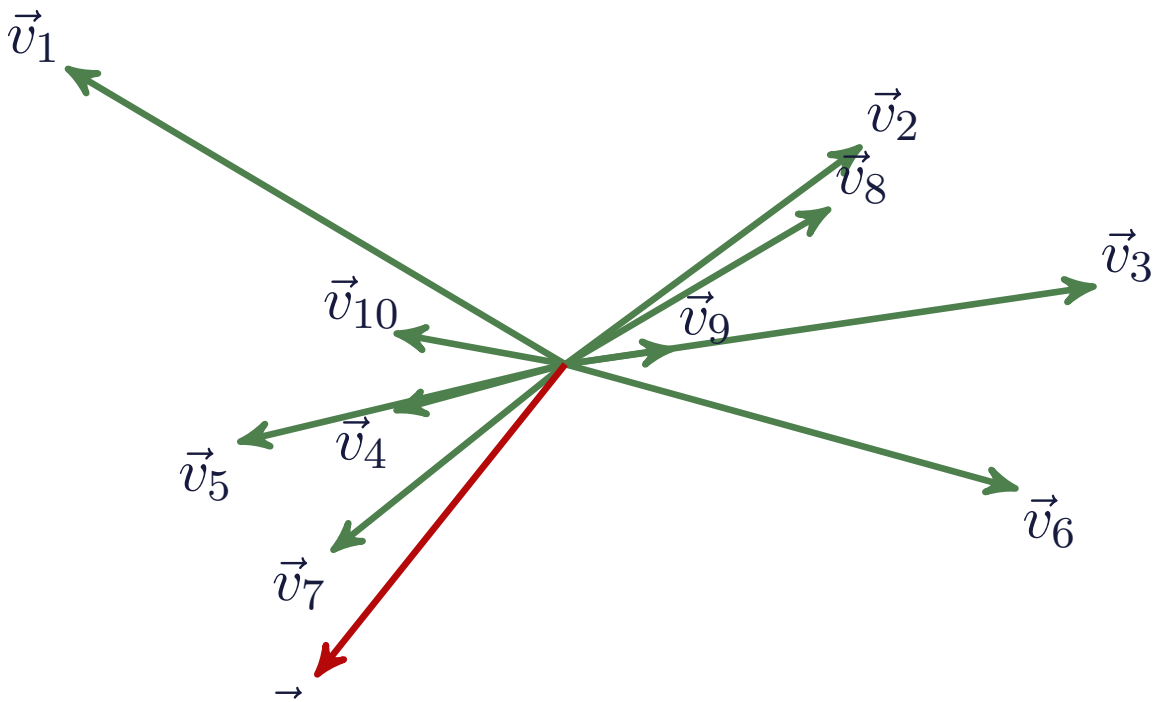
Mit Hashfunktion

$$h(v) = \lfloor \frac{\vec{a} \cdot \vec{v}}{4} \rfloor$$

$$h(\vec{v}_1) = 0$$
$$h(\vec{v}_2) = -2$$
$$h(\vec{v}_3) = -2$$
$$h(\vec{v}_4) = 0$$
$$h(\vec{v}_5) = 1$$
$$h(\vec{v}_6) = -1$$
$$h(\vec{v}_7) = 1$$
$$h(\vec{v}_8) = -2$$
$$h(\vec{v}_9) = -1$$
$$h(\vec{v}_{10}) = 0$$

Bucket -2	Bucket -1	Bucket 0
$\vec{v}_2$	$\vec{v}_6$	$\vec{v}_1$
$\vec{v}_3$	$\vec{v}_9$	$\vec{v}_4$
$\vec{v}_8$		$\vec{v}_{10}$

Bucket 1	Bucket 2
$\vec{v}_5$	
$\vec{v}_7$	



Hashwerte:

$h(\vec{v}_1) = 0$
$h(\vec{v}_2) = -2$
$h(\vec{v}_3) = -2$
$h(\vec{v}_4) = 0$
$h(\vec{v}_5) = 1$
$h(\vec{v}_6) = -1$
$h(\vec{v}_7) = 1$
$h(\vec{v}_8) = -2$
$h(\vec{v}_9) = -1$
$h(\vec{v}_{10}) = 0$

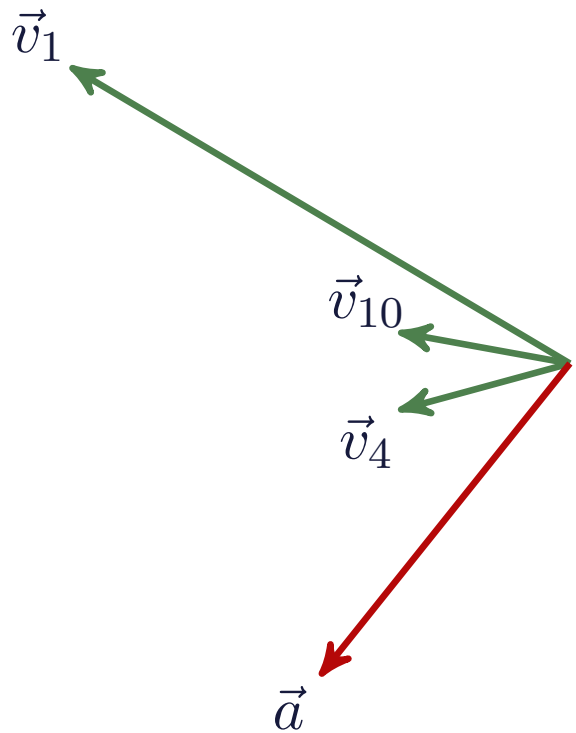
Anfrage: Finde Nachbarn von $\vec{v}_8$.

Vorgehensweise:

Schaue in Bucket mit Label -2 $(= h(\vec{v}_8))$.
Finde zwei Objekte.
Evaluiere diese.

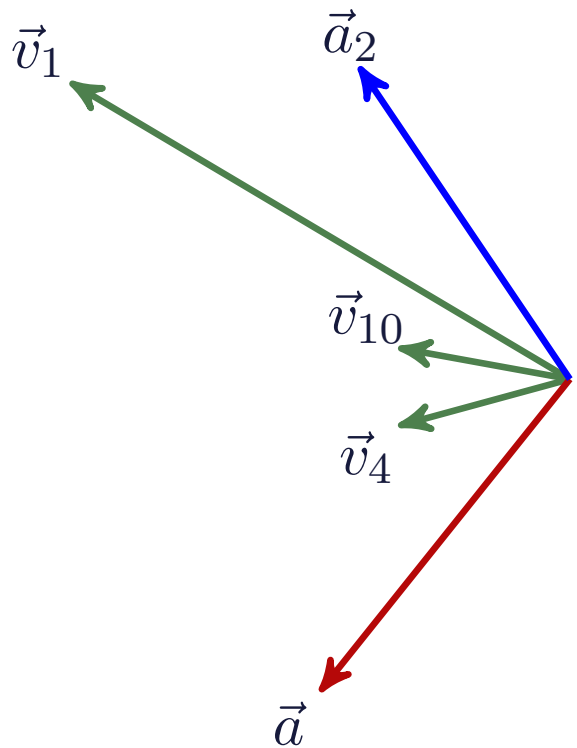
Beobachtung: $\vec{v}_{10}$, $\vec{v}_4$ und $\vec{v}_1$ fallen in einen Bucket. Ok, $\vec{v}_{10}$ und $\vec{v}_4$ sind in der Tat ähnlich, aber $\vec{v}_1$ passt nicht gut.

- **Wo ist das Problem?**
  Vektor $\vec{a}$ ist nicht in der Lage gut zwischen diesen drei Vektoren zu unterscheiden.



Bucket 0
$\vec{v}_1$
$\vec{v}_4$
$\vec{v}_{10}$

Beobachtung: $\vec{v}_{10}$, $\vec{v}_4$ und $\vec{v}_1$ fallen in einen Bucket. Ok, $\vec{v}_{10}$ und $\vec{v}_4$ sind in der Tat ähnlich, aber $\vec{v}_1$ passt nicht gut.



- **Wo ist das Problem?**
  Vektor $\vec{a}$ ist nicht in der Lage gut zwischen diesen drei Vektoren zu unterscheiden.
- **Lösung:** Wir nehmen einen zweiten Vektor $\vec{a}_2$ hinzu.

$$h_{a_2}(\vec{v}) := \lfloor \frac{\vec{a}_2 \cdot \vec{v}}{4} \rfloor$$

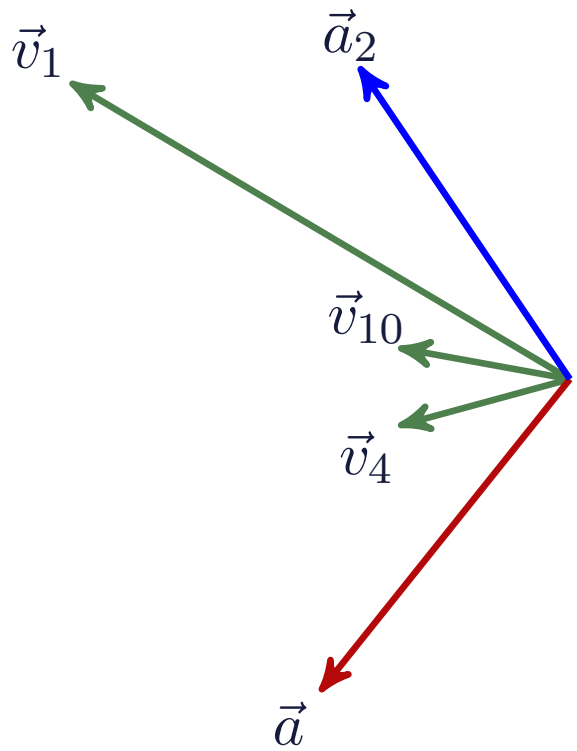
Mit $h_{a_2}(\vec{v}_{10}) = 0$, $h_{a_2}(\vec{v}_4) = 0$, $h_{a_2}(\vec{v}_1) = 2$

Beobachtung: $\vec{v}_{10}$, $\vec{v}_4$ und $\vec{v}_1$ fallen in einen Bucket. Ok, $\vec{v}_{10}$ und $\vec{v}_4$ sind in der Tat ähnlich, aber $\vec{v}_1$ passt nicht gut.

- **Wo ist das Problem?**
  Vektor $\vec{a}$ ist nicht in der Lage gut zwischen diesen drei Vektoren zu unterscheiden.
- **Lösung:** Wir nehmen einen zweiten Vektor $\vec{a}_2$ hinzu.

$$h_{a_2}(\vec{v}) := \lfloor \frac{\vec{a}_2 \cdot \vec{v}}{4} \rfloor$$

  Mit $h_{a_2}(\vec{v}_{10}) = 0$, $h_{a_2}(\vec{v}_4) = 0$, $h_{a_2}(\vec{v}_1) = 2$
- Wie sehen dann die Labels der Buckets aus? Konkatenation der einzelnen Hashwerte.
- Wir haben also die "Genauigkeit" der Buckets erhöht.
- Was ist der potentielle Nachteil?



Bucket (0,0)
$\vec{v}_4$
$\vec{v}_{10}$

...

Eine bekannte LSH Variante

Für jeden d-dimensionalen Punkt $\vec{v}$ betrachten wir k unabhängige Hashfunktionen der Form:

$$h_{\vec{a},B}(\vec{v}) = \left\lfloor \frac{\vec{a} \cdot \vec{v} + B}{W} \right\rfloor$$

$\vec{a}$: d-dimensionaler Vektor, zufällig anhand Wahrscheinlichkeitsverteilung ausgewählt.
$W \in \mathbb{R}$, und $B \in [0,W]$. $\vec{v}$ wird auf $\vec{a}$ "projiziert" (Skalarprodukt).

"Beschriftung" der LSH Buckets

Mit k Hashfunktionen ist die Beschriftung des Buckets ein Integer-Vektor der Länge k:

$$g(\vec{v}) = (h_{\vec{a_1},B_1}(\vec{v}), ..., h_{\vec{a_k},B_k}(\vec{v}))$$

Welchen Einfluss hat k auf die Suche?

Paper: M. Datar et al.: Locality-sensitive hashing scheme based on p-stable distributions. Symposium on Computational Geometry, 2004.

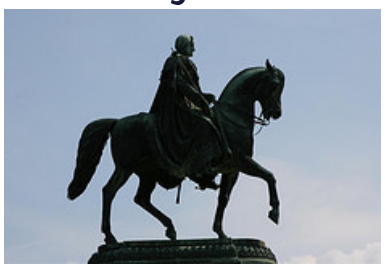
Objekte den Hash-Buckets zuweisen

LSH Bucket "Labels"

Mit k Hashfunktionen ist ein Label ein Integer-Vektor der Länge k:

$$g(\vec{v}) = (h_{\vec{a_1},B_1}(\vec{v}), ..., h_{\vec{a_k},B_k}(\vec{v}))$$

Objekt:



Anwendung von 4 Hashfunktionen:

$$\begin{aligned} h_1(...) &= 0 \\ h_2(...) &= 1 \\ h_3(...) &= 1 \\ h_4(...) &= 1 \end{aligned}$$

Ergibt das Label: g(...)=(0,1,1,1)

Wie kann man das Problem adressieren, dass ähnliche Objekte in einem anderen Bucket landen und somit nicht gefunden werden?

Suche nach ähnlichen Objekten: Beispiel

		Bucket-Label
Anfrage:		0,1,0,1
Potentieller Treffer:		0,1,1,1

Nicht gefunden!

Idee: Mehrere Hashtabellen!

- Um Trefferwahrscheinlichkeit zu erhöhen werden Objekte nicht nur in einen Bucket gesteckt anhand von g indexiert, sondern anhand von verschiedenen g in mehrere Buckets in verschiedenen Hashtabellen.

Anfrageverarbeitung: Suche der K nächsten Nachbarn

Gegeben ein Anfrage-Punkt q

- Berechne Bucket-Labels $g_i(q)$ für jede Hashtabelle i.
- Hole Objekte aus diesen Buckets
- Berechne echte Distanz und ordne Objekte entsprechend

Trotzdem: LSH ist eine approximative Technik

- Keine harte Garantie, dass alle Treffer (und nur diese) gefunden werden ...
- ... das ist oftmals aber akzeptabel.
- Es gibt theoretische Ansätze die Ergebnisgüte voraus zu sagen

Mehrere Hashtabellen

Benutze mehrere Hashtabellen, um die Wahrscheinlichkeit der Kollisionen zu vergrößern

	Hashtabelle 1	Hashtabelle 2
	0,1,0,1	**1,0,0,1**
	0,1,1,1	**1,0,0,1**
Ergebnis:	Kein Treffer	Treffer!

Beobachtungen

Tuning

- Tradeoff zwischen Größe der Hash-Buckets und der Effektivität
- Mehrere Hashtabellen: Höhere Trefferwahrscheinlichkeit aber größerer Platzverbrauch
- Achtung: auch hier gilt wieder: Ab einem bestimmen Punkt kann ein Full-Scan günstiger sein (und dieser ist sogar noch exakt!)

Erweiterungen

- Schaue in mehrere Hash-Buckets *per* Hashtabelle (aka. multi-probe LSH)
- Auch: verteilte Implementierungen von LSH (z.B. in Peer-to-Peer-Systemen) oder in MapReduce