Vektordatenbanken

Grundlagen und praktische Anwendungen

Olaf Herden
DHBW (Duale Hochschule Baden-Württemberg)
Stuttgart Campus Horb
o.herden@hb.dhbw-stuttgart.de

DHBW Duale Hochschule Baden-Württemberg

JFS 2026
09.07.2026 - Stuttgart

JAVAFORUM stuttgart 2026

Speaker

- Olaf Herden
- Duale Hochschule Baden-Württemberg
- Seit 2002 Professor für Informatik
- Interessens-/Lehr-/Forschungsgebiete:
  - Datenbanken
  - Data Analytics
  - Data Mining/Machine Learning
  - Algorithmen und Datenstrukturen
  - Programmierung
  - Learning to code

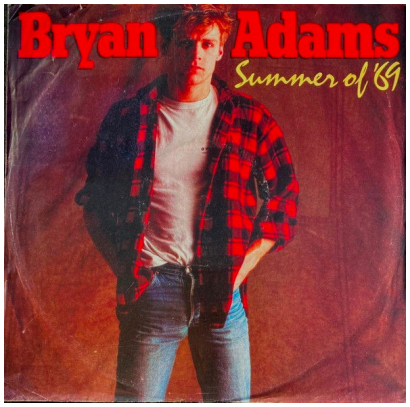
- LinkedIn: https://www.linkedin.com/in/olaf-herden-896a14157/
- Email: o.herden@hb.dhbw-stuttgart.de

Gliederung

- Einführung
- Ähnlichkeit
- Suchen und Indexieren
- Realisierungen
- Zusammenfassung

Motivation (I)

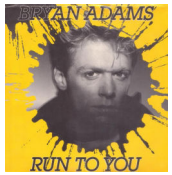
GreMuSt - Great Music Streaming



Summer Of 69
Bryan Adams

Recommended for you:

Born to run
Bruce Springsteen



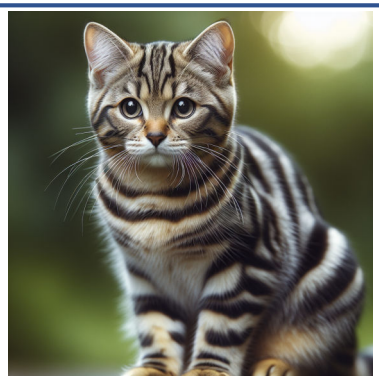
Run to you
Bryan Adams

The boys of summer
Don Henley

You give love a bad name
Bon Jovi

Motivation (II)

TGIS - **T**he **G**reatest **I**mage **S**earch



Search

Result:

[All images on this slide are generated by ChatGPT.]

Motivation (III)

My company's RAG system for databases

I need a SQL statement to get the top 5 customers by total purchase amount this year.

"Based on company database structure (`customers`, `orders`, `order_items` tables), here is the SQL statement you can use:"

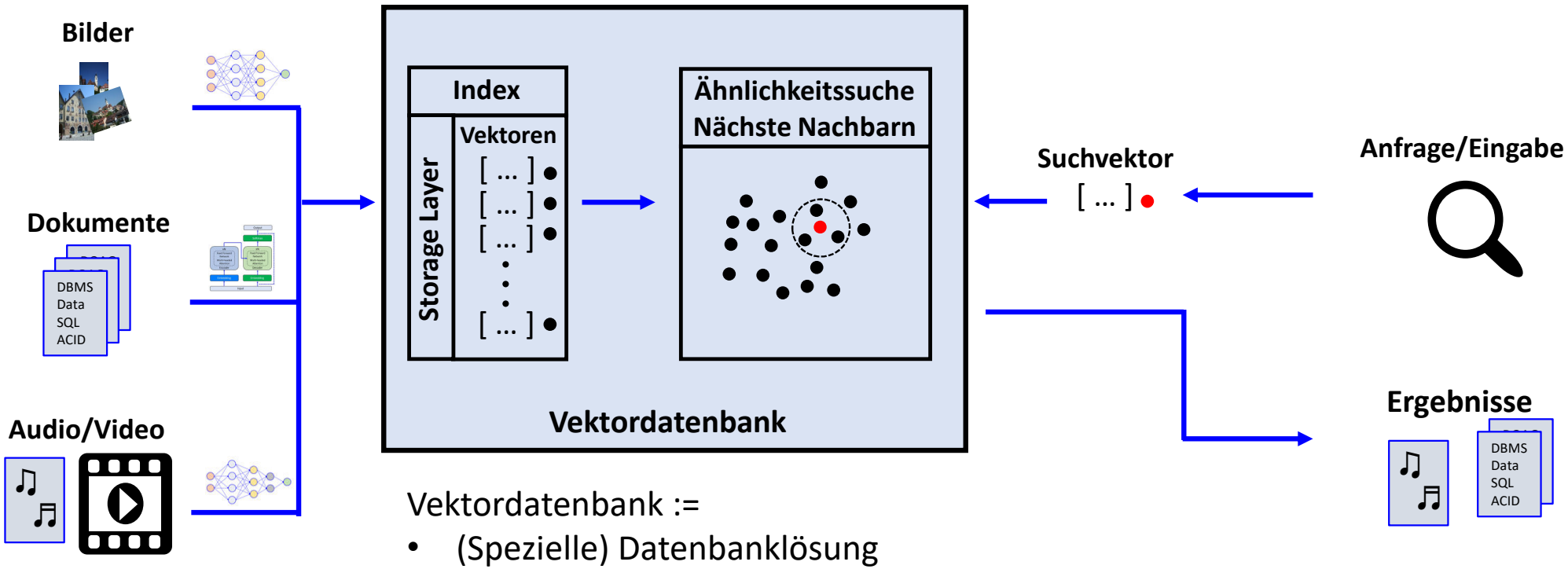
sql Kopieren Bearbeiten

```sql
SELECT c.customer_id, c.name, SUM(oi.quantity * oi.unit_price) AS total_spent
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_items oi ON o.order_id = oi.order_id
WHERE o.order_date >= DATE_TRUNC('year', CURRENT_DATE)
GROUP BY c.customer_id, c.name
ORDER BY total_spent DESC
LIMIT 5;
```

"This query calculates the total purchase amount by summing product of quantity and unit price per order item, filtering only for orders placed this year."

[Example on this slide is generated by ChatGPT.]

Architektur



Vektordatenbank :=

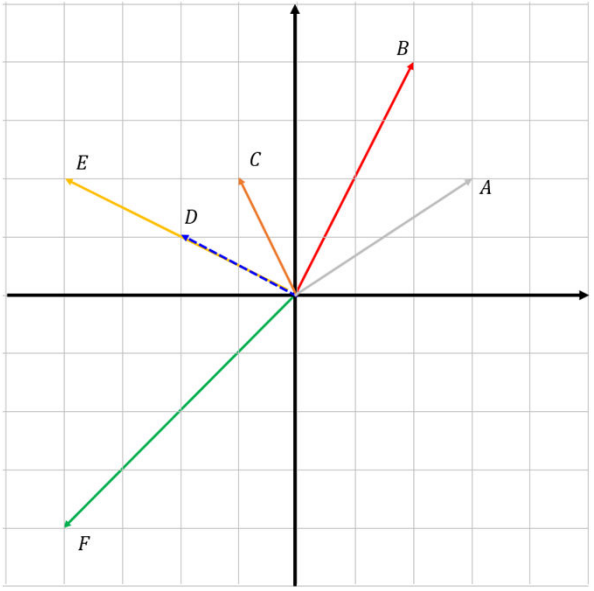
- (Spezielle) Datenbanklösung
- Speichert Vektoren
- Indexiert Vektoren für schnelles Lesen und Ähnlichkeitssuche

Übersicht

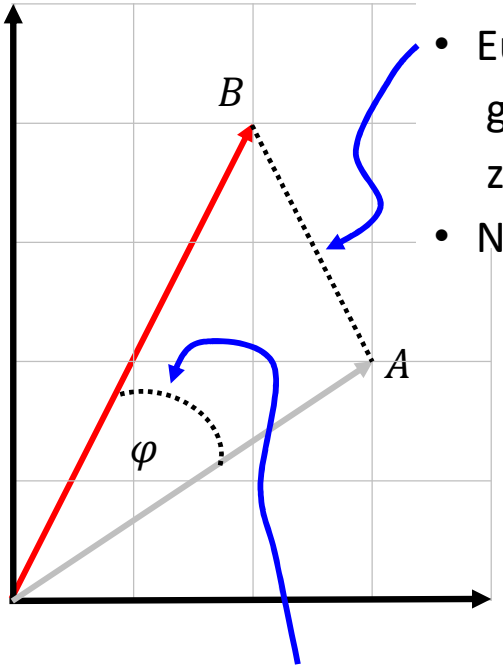
- Einführung
- Ähnlichkeit
- Suchen und Indexieren
- Realisierungen
- Zusammenfassung

Ähnlichkeitsproblem und -maße

- Problem:
  - Gegeben zwei (oder mehr) Vektoren
  - Wie ähnlich (oder unterschiedlich) sind diese?
- Beispiel:
  - Ist D ähnlicher zu C oder zu E?



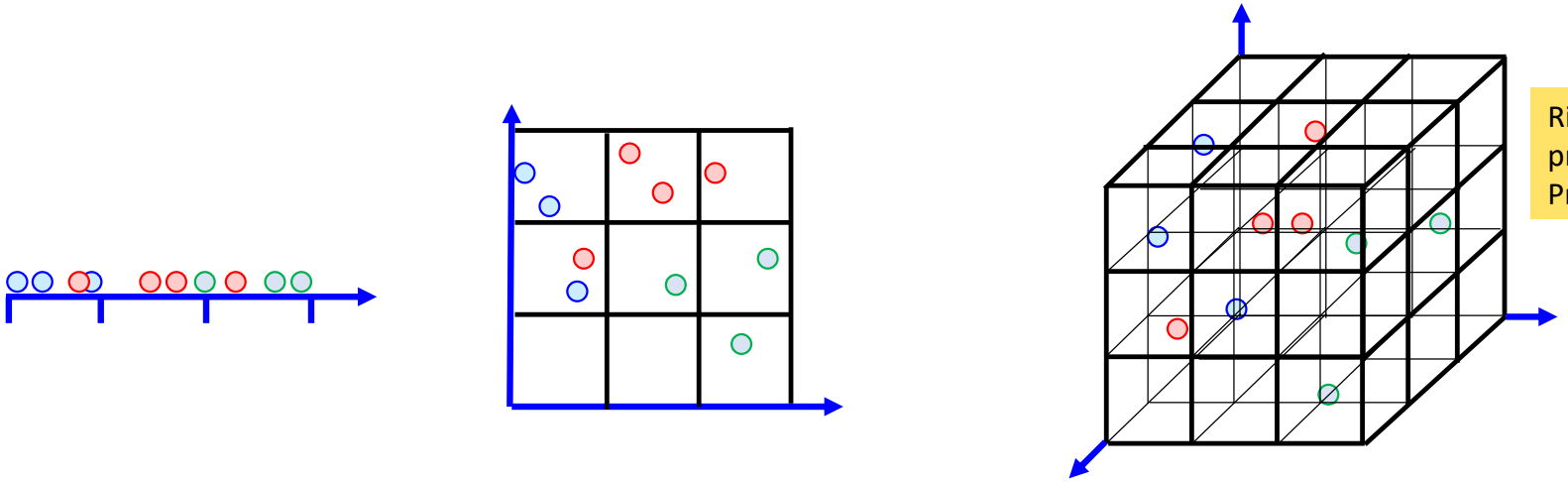
- Zwei wichtige Maße:



  - Euklidische Distanz: absoluter geometrischer Abstand zwischen Vektoren
  - Notation: $euDist(A, B)$
  - Kosinusähnlichkeit: Kosinus des Winkels zwischen Vektoren
  - Notation: $cosSim(A, B)$

Euklidische Distanz vs. Kosinusähnlichkeit

- Euklidische Distanz: Curse of Dimensionality:

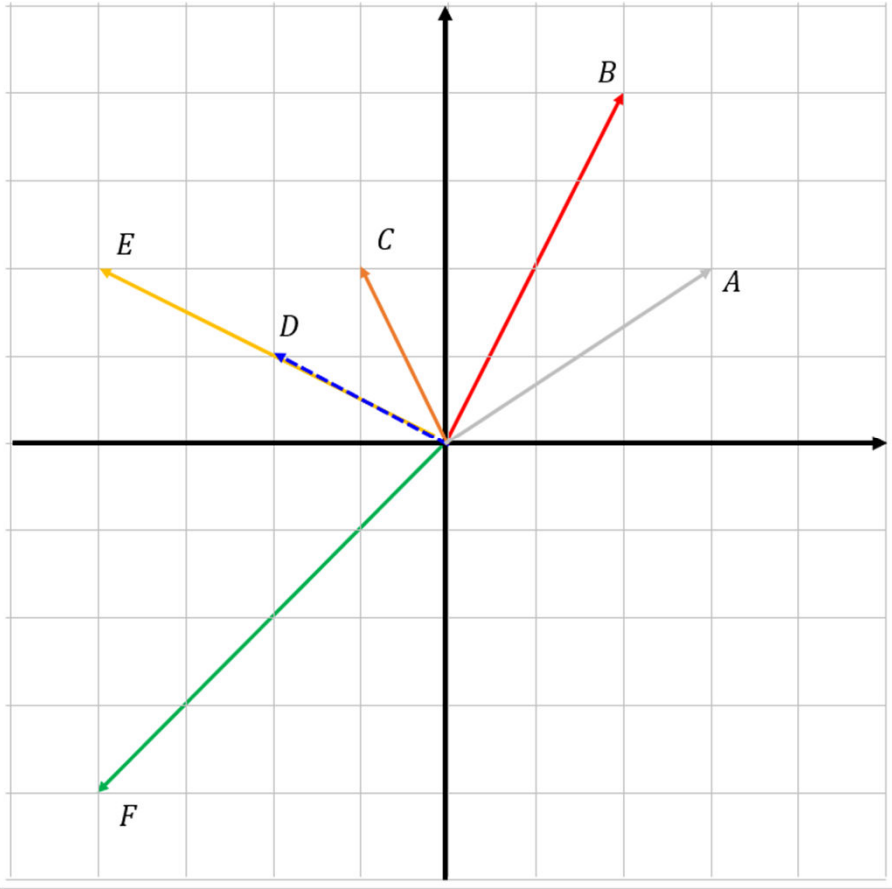


Richard Bellman. Adaptive control processes: a guided tour. Princeton University Press, 1961.

⇒ Häufig/meistens: Kosinusähnlichkeit bei Bestimmung Ähnlichkeit von Embeddings repräsentierenden Vektoren

Kosinusähnlichkeit

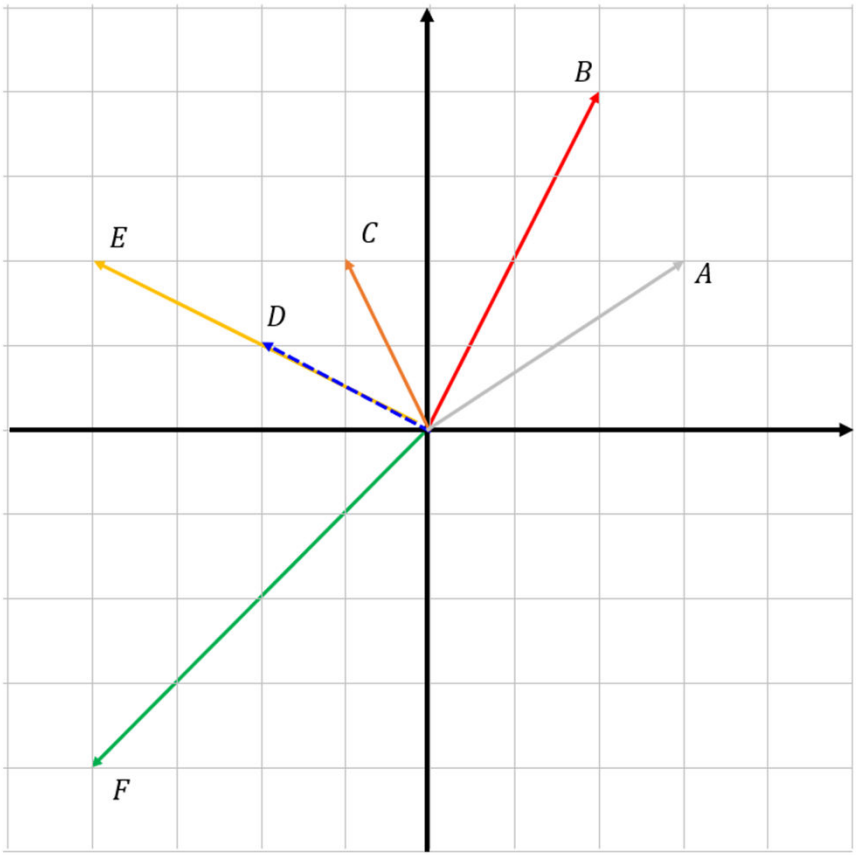
- Beispiel:



Kosinusähnlichkeit						
	A	B	C	D	E	F
A		0,868	0,124	-0,496	-0,496	-0.981
B			0,6	0	0	-0,949
C				0,8	0,8	-0,316
D					1	0,316
E						0,316
F						

- Kosinusähnlichkeit ist
  - nahe 1, wenn Winkel zwischen Vektoren nahe $0°$
  - nahe 0, wenn Winkel zwischen Vektoren nahe $90°$
  - nahe -1, wenn Winkel zwischen Vektoren nahe $180°$
- Kosinusähnlichkeit
  - berücksichtigt nur Richtung der Vektoren
  - vernachlässigt Länge der Vektoren

Euklidische Distanz vs. Kosinusähnlichkeit



- Ist D ähnlicher zu C oder zu E?
  - $euDist(D, C) = 1.41, euDist(D, E) = 1.73$
  - D ist ähnlicher zu C (Euklidische Distanz)
  - $cosSim(D, C) = 0.8, cosSim(D, E) = 1$
  - D ist ähnlicher zu E (Kosinusähnlichkeit)

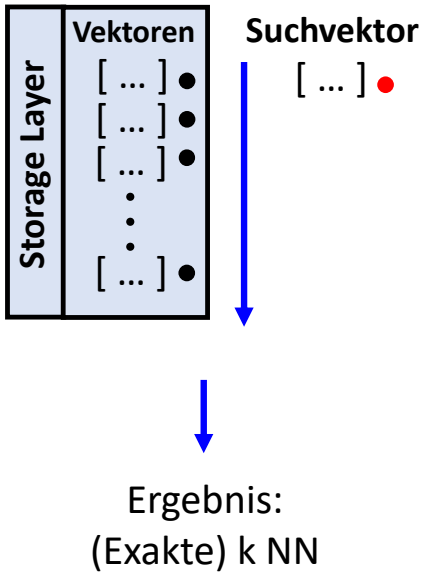
- Anm.: Kosinusdistanz
  - $cosDist(A, B) \coloneqq 1 - cosSim(A, B)$
  - Grund:
    - Ebenfalls Distanzmaß
    - Positiver Wertebereich ($[0,2]$ statt $[-1,1]$)

Übersicht

- Einführung
- Ähnlichkeit
- Suchen und Indexieren
- Realisierungen
- Zusammenfassung

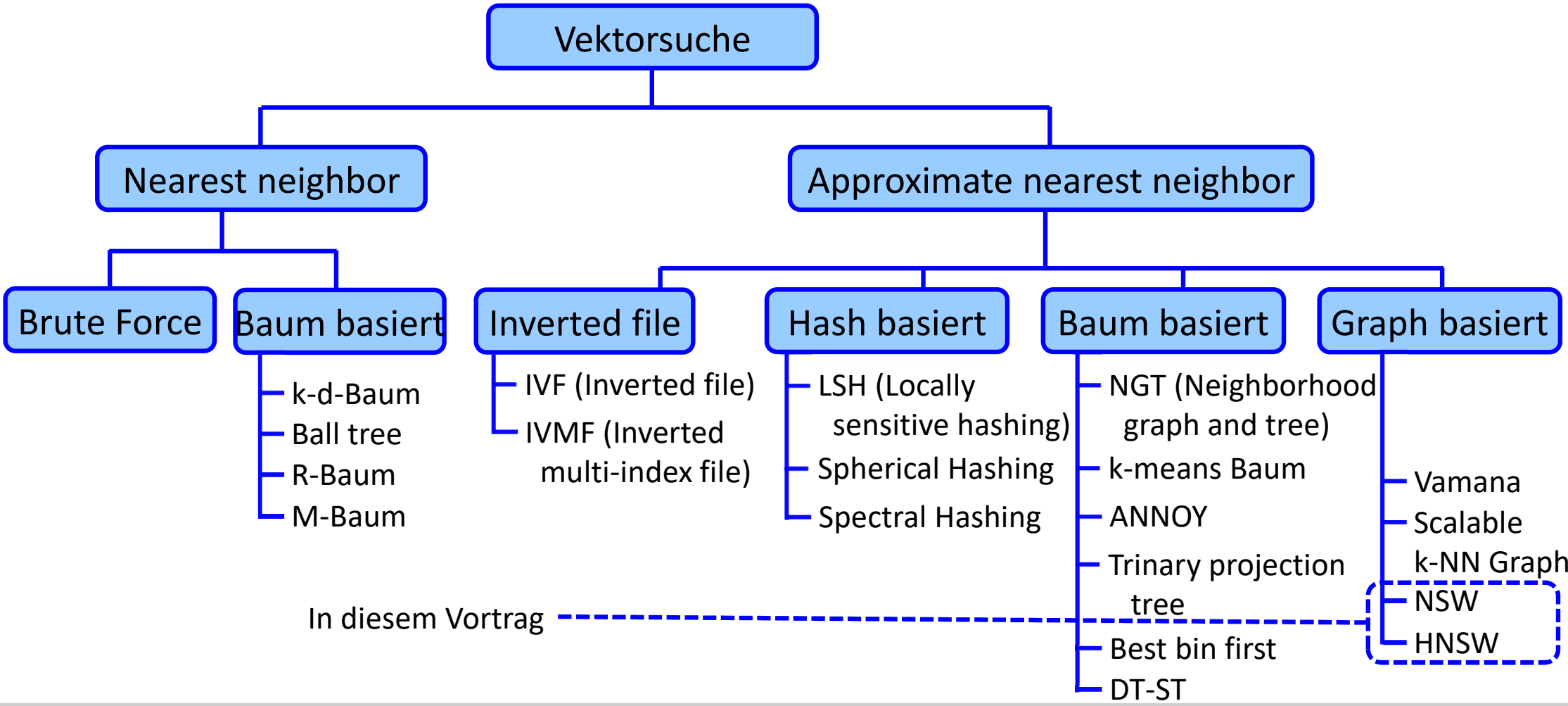
Brute Force k-NN-Suche

- Eingabe: Data Set V Vektoren, Suchvektor Q, k gewünschte nächste Nachbarn, Distanz-/Ähnlichkeitsfunktion
- Ausgabe: Liste L mit k Vektoren, am ähnlichsten zu Q, sortiert nach Ähnlichkeit zu Q



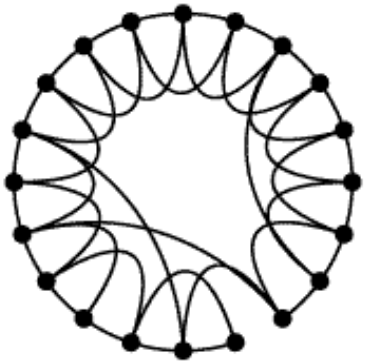
- Fazit:
  - Resultat korrekt
  - Aber zu berechnungsintensiv
  - Frage: Gibt es alternative Möglichkeiten, um schneller zu suchen, ggf. unter Verlust an Genauigkeit?
- Antwort: Ja, es existiert eine Vielzahl an Indextechniken!

Übersicht


Vektorsuche
Nearest neighbor
Approximate nearest neighbor
Brute Force
Baum basiert
Inverted file
Hash basiert
Baum basiert
Graph basiert
k-d-Baum
Ball tree
R-Baum
M-Baum
IVF (Inverted file)
IVMF (Inverted multi-index file)
LSH (Locally sensitive hashing)
Spherical Hashing
Spectral Hashing
NGT (Neighborhood graph and tree)
k-means Baum
ANNOY
Trinary projection tree
Best bin first
DT-ST
Vamana
Scalable k-NN Graph
NSW
HNSW
In diesem Vortrag

Navigable Small World (NSW)

- Small World Graph:
  - Hoher Clusterkoeffizient
  - Geringe Distanzen
  - Charakteristische Pfad wächst in $O(\log(n))$

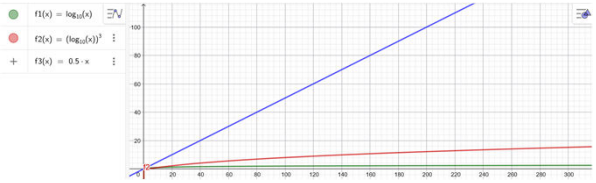


Watts, D., Strogatz, S. Collective dynamics of 'small-world' networks. Nature 393, 440–442 (1998). https://doi.org/10.1038/30918

- Navigable Graph:
  - Länge des durch Best-First-Search-Algorithmus gefundenen Suchpfads wächst in $O(\log(n))$

Kleinberg, J. Navigation in a small world. *Nature* 406, 845 (2000). https://doi.org/10.1038/35022643

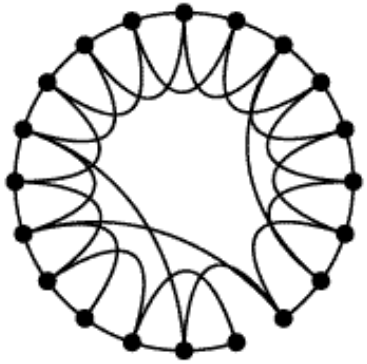
- NSW:
  - Graph mit beiden Eigenschaften
  - Logarithmische Suchkomplexität
  - Aber: Tendenz Skalierung Ausgangsgrad $\log(n) \Rightarrow O(\log(n)^k)$ with $k > 1$



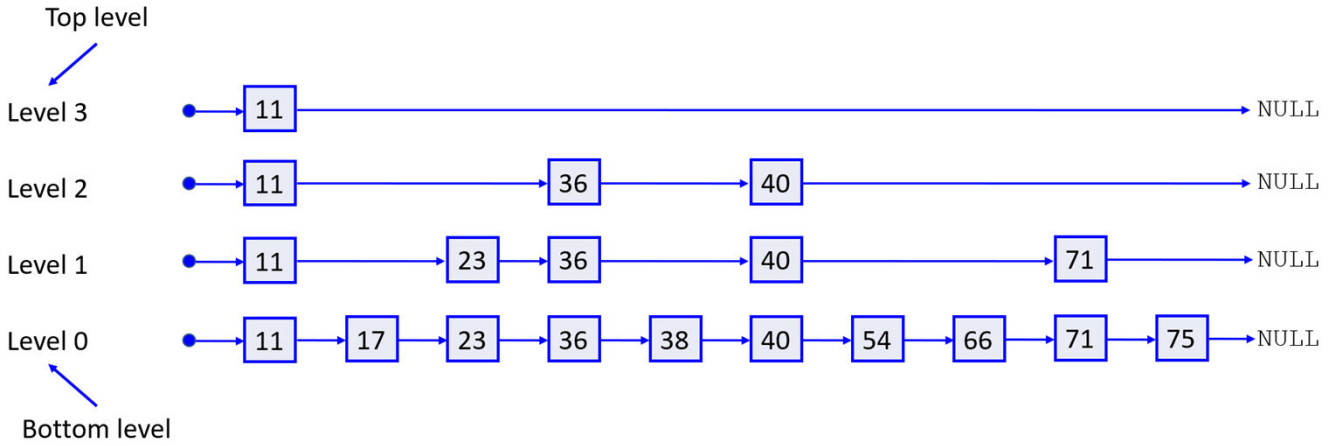
Hierarchical Navigable Small World (HNSW)

William W. Pugh: Skip Lists: A Probabilistic Alternative to Balanced Trees. Commun. ACM 33(6): 668-676 (1990).

- Konzept 1: NSW Graph

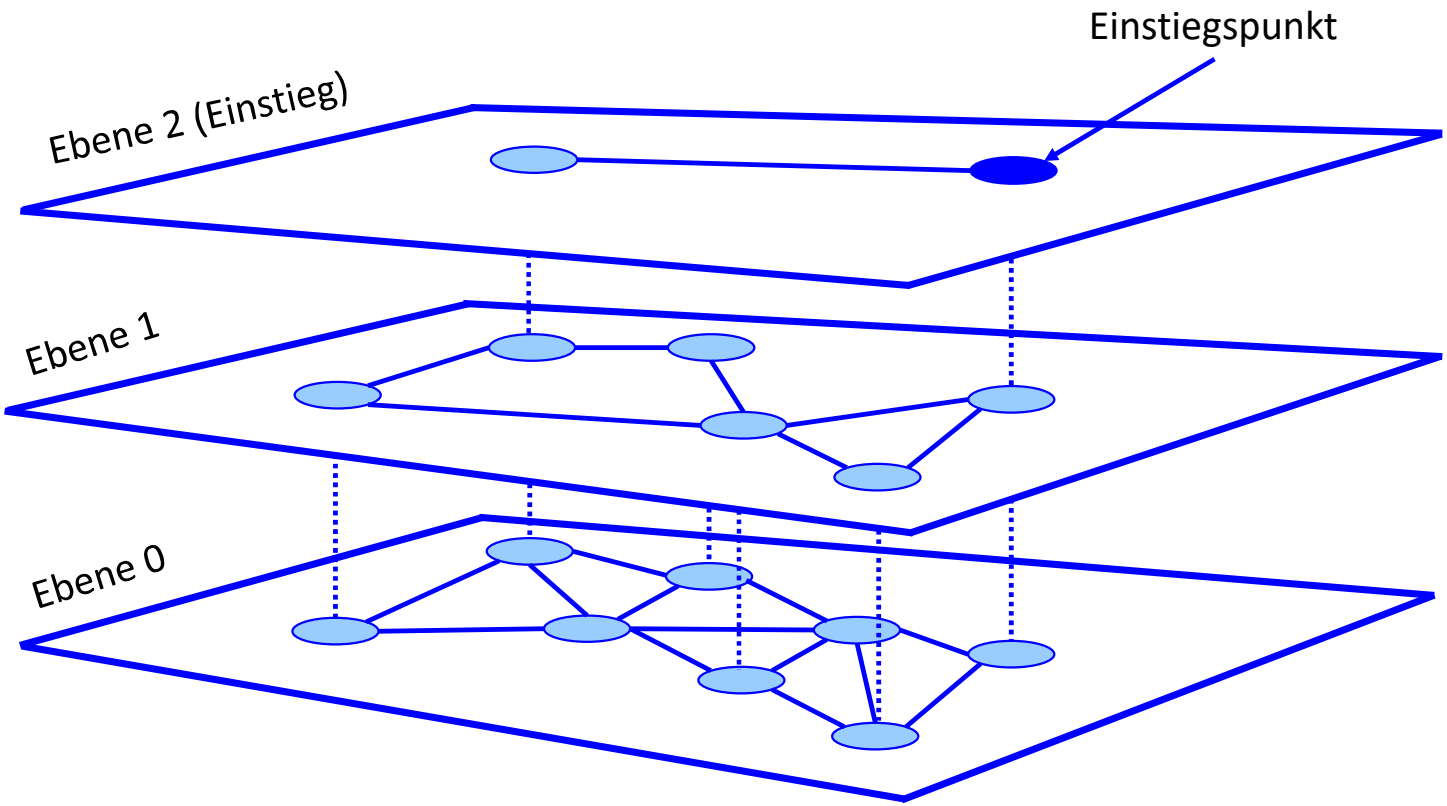


- Konzept 2: Skip List



Kombination beider Ideen
Ziel: Erhalten logarithmischer Komplexität

HNSW (II): Struktur

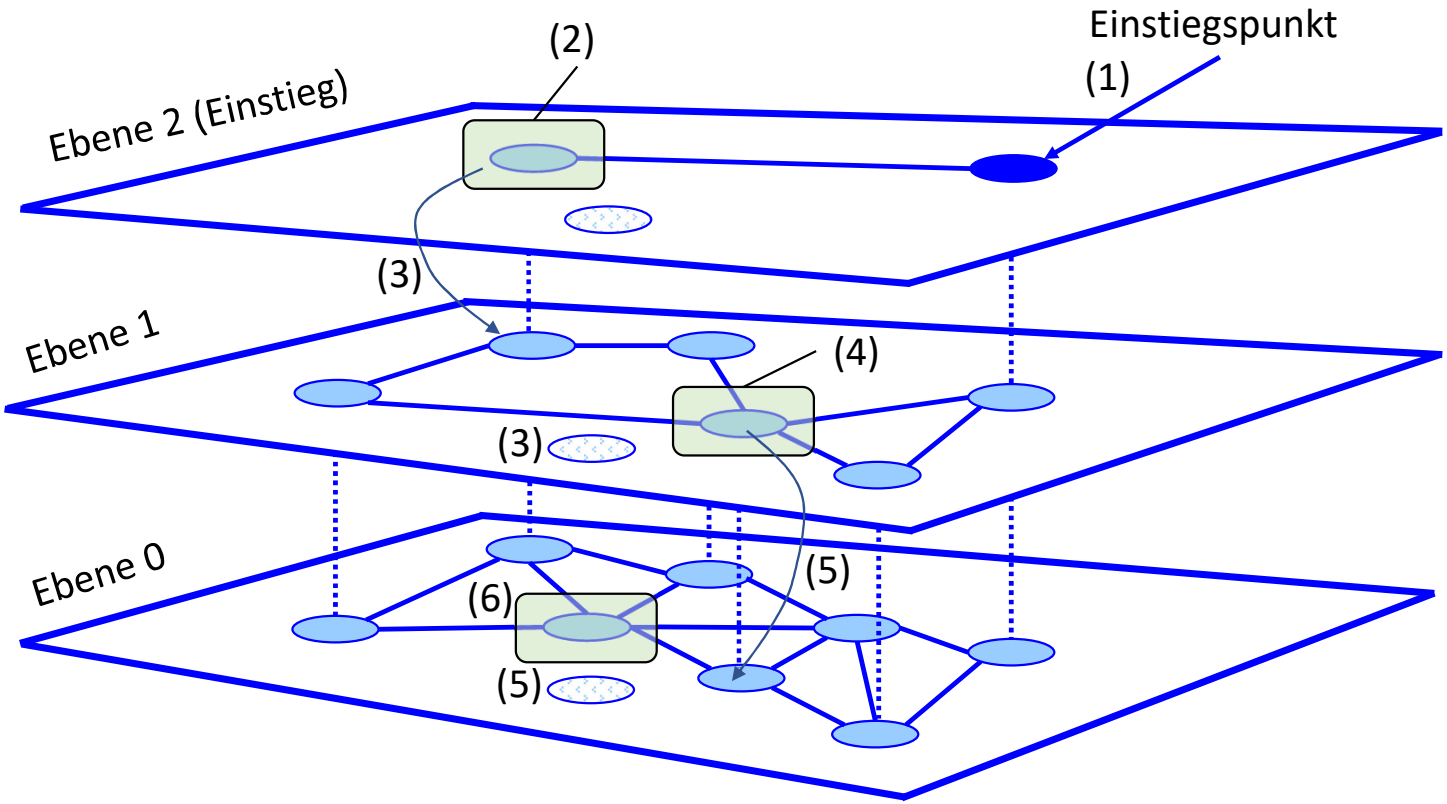


- Struktur:
  - M Ebenen (layers)
  - Einstiegsebene (M) bis Basis-ebene (0)
  - Duplikation Teilmenge der Knoten von Ebene N in Ebene $N+1$ (für $0 \leq N \leq M-1$)
  - Von M bis 0:
    - Wachsende Anzahl Knoten
    - Wachsende Anzahl Links
    - Abnehmende Länge der Links
  - In Einstiegsebene: ein Knoten Einstiegpunkt

Yury A. Malkov, Dmitry A. Yashunin: Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. IEEE Trans. Pattern Anal. Mach. Intell. 42(4): 824-836 (2020)

HNSW (III): Suchen

- Gegeben: Suchvektor Q ( )



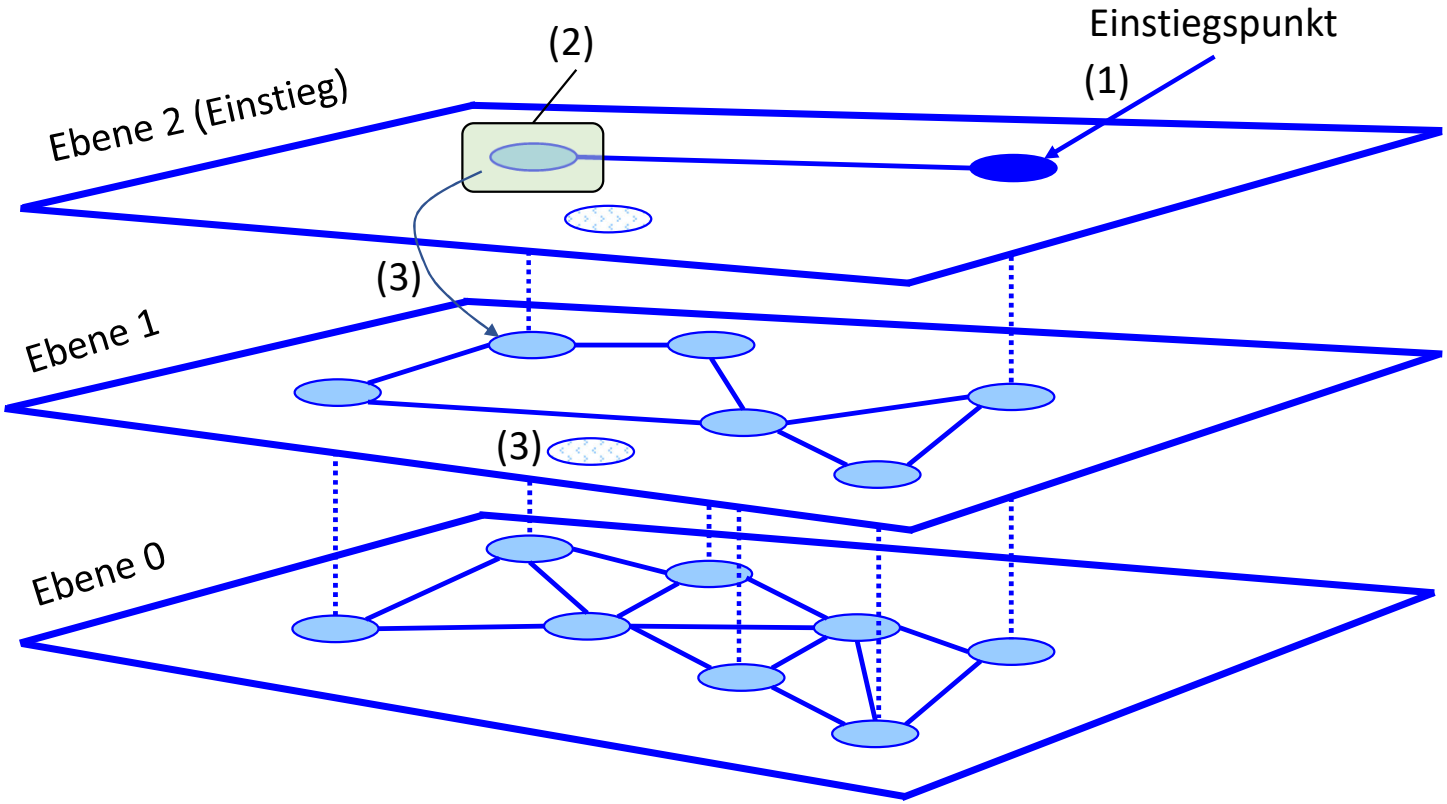
(1) Gehe zu Einstiegsebene
(2) Suche NN von Q (=: X)
(3) Gehe zu Ebene $M - 1$
(4) Suche NN von Q beginnend mit X

- Wiederhole bis Basisebene erreicht
- Im Beispiel:

(5) Gehe zu Ebene 0
(6) Suche NN von Q

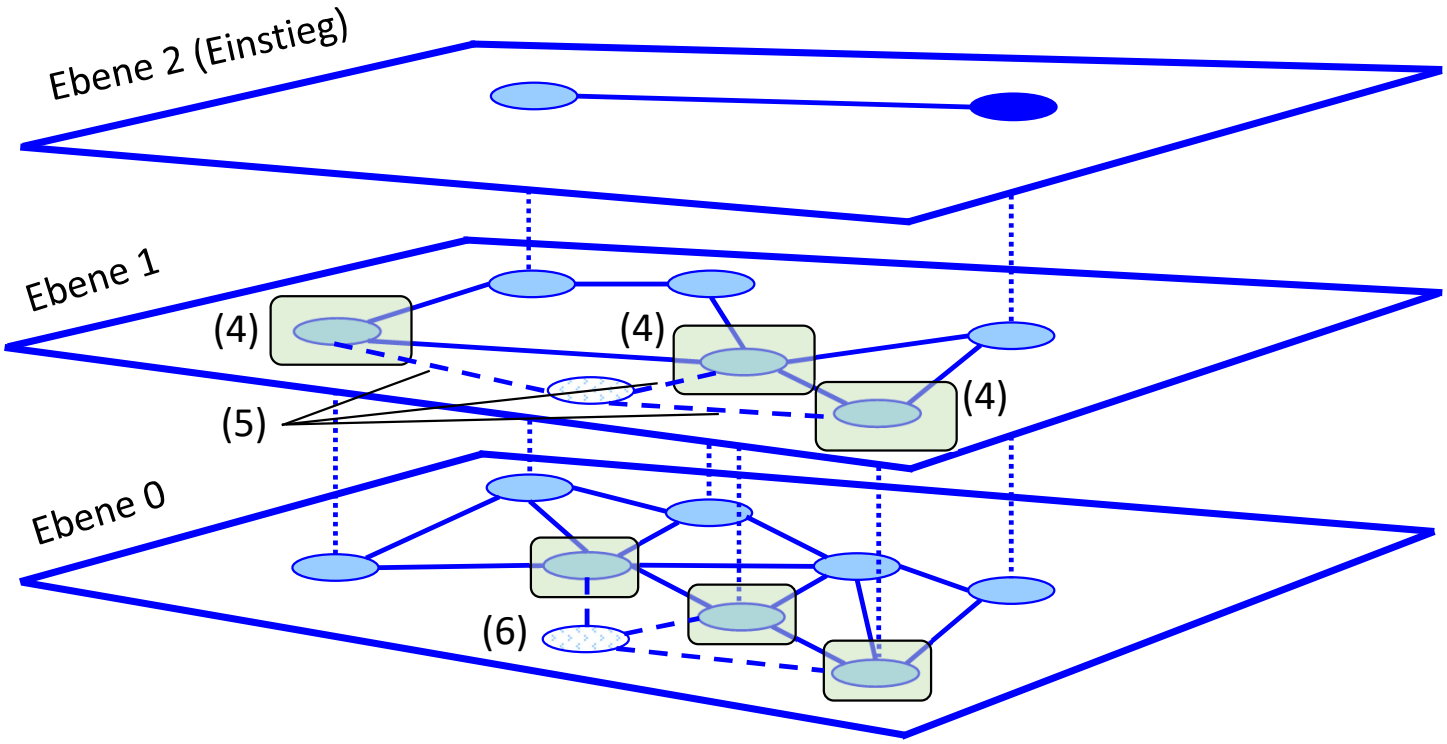
HNSW (IV): Einfügen (I)

- Gegeben:
  - Einzufügender Vektor ()
  - Parameter k (zu berücksichtigende NN in Einfügeebene)



- Zufallsfunktion f_{rand} liefert Einfügeebene l
- Sei $l = 1$
- Phase 1:
  (1) - (3): Starte in Einstiegsebene und traversiere zu Ebene l (analog zum Suchen)

HNSW (V): Einfügen (II)



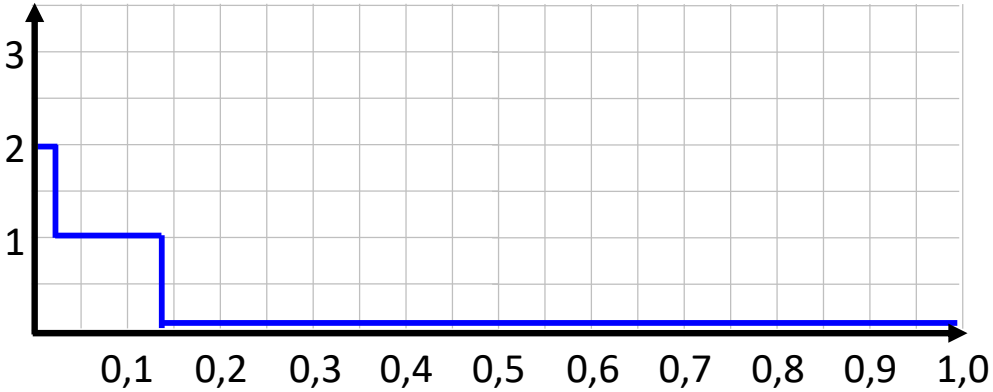
- Phase 2:
  - (4) Berechne k NN von Q in l (=: X)
  - (5) Füge Links ein von Q zu jedem $x \in X$ (- - - -)
  - (6) Wiederhole (4) und (5) bis Basisebene erreicht

HNSW (VI): Einfügen (III)

- Eigenschaft von f_{rand}:
  - Exponentiell fallende Wahrscheinlichkeitsfunktion
  - Nehme Faktor $m_L > 0$
  - $l = [-ln(uniform(0,1)) \cdot m_L]$

- Anmerkung:
  - Darstellung Algorithmus vereinfacht
  - Problem: Anzahl Kanten wächst sehr schnell
  - Lösung: Zwei Parameter zur Beschränkung Knotengrad:
    - M_{MAX} : Maximum Knotengrad in Ebenen M bis 1
    - M_{MAX0}: Maximum Knotengrad in Ebene 0

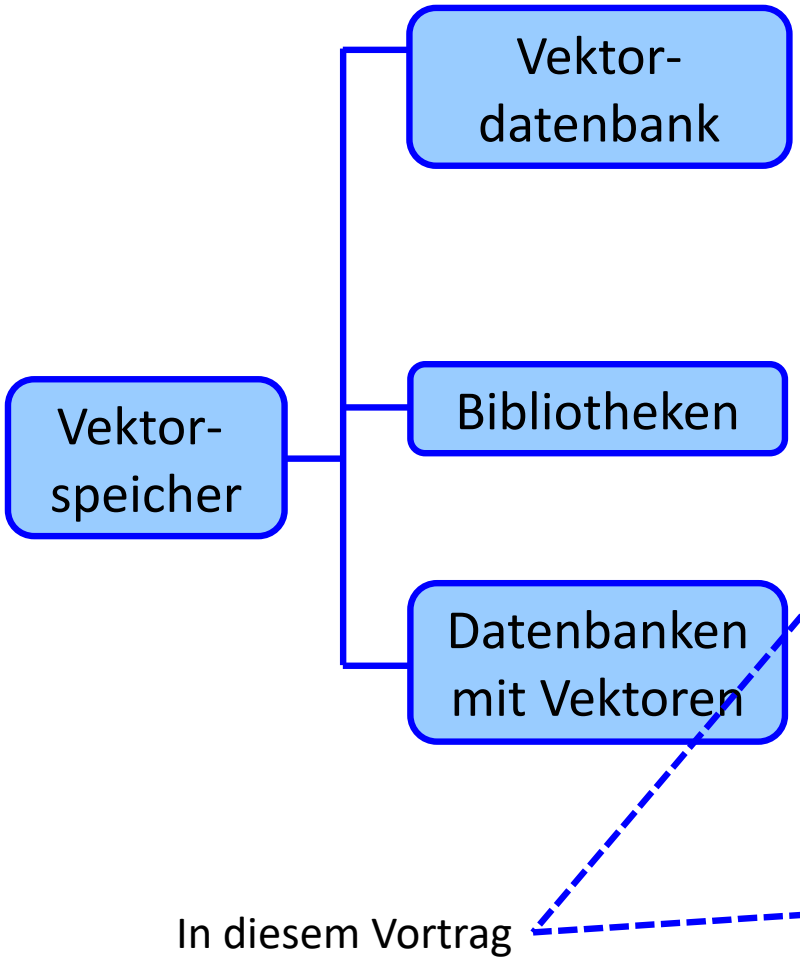
Bsp.: $m_L = 0.5$



Übersicht

- Einführung
- Ähnlichkeit
- Suchen und Indexieren
- Realisierungen
- Zusammenfassung

Ansätze



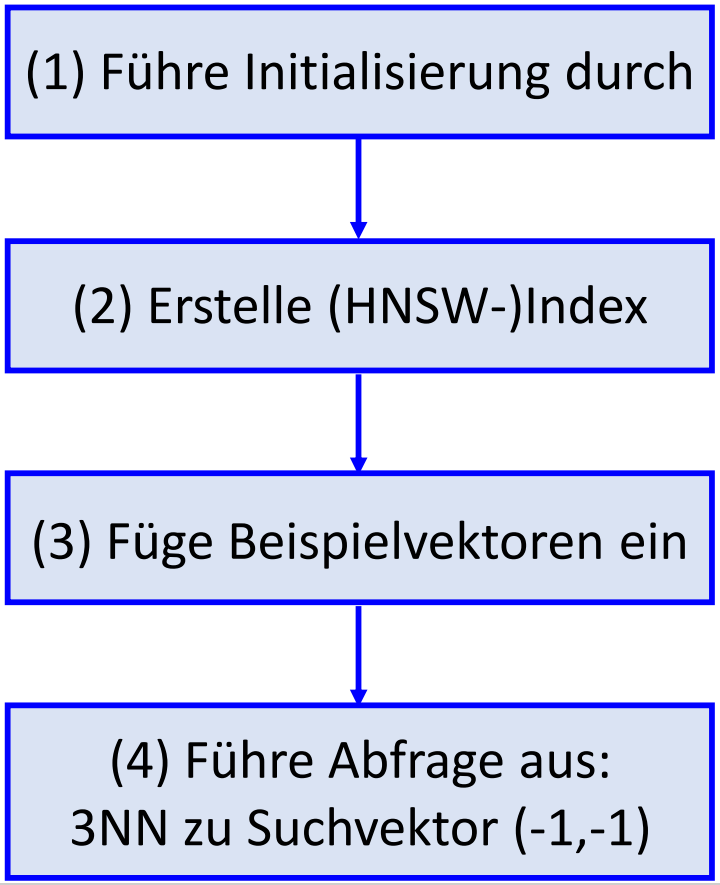
- Speziallösungen
- Gute Performance in Vektorindexierung and -suche
- Häufig unausgereift bezgl. DB-Eigenschaften (z.B. Recovery, Security, ACID)
- Originaldaten müssen separat gehalten und gelesen werden
- Beispiele: Pinecone, Weaviate, Milvus

- Direkte Einbindung in Applikationscode
- Typischerweise im Hauptspeicher ⇒ Limitierte Skalierbarkeit
- Beispiele: ANNOY, FAISS, ScaNN

- Erweiterung Datenbanken um Vektoren
- Alle Daten an einem Ort: strukturierte Daten, Originaldaten und Vektoren können gemeinsam gelesen werden
- Ausgereift bezgl. DB-Eigenschaften
- Beispiele:
  - Relational: Oracle, PostgresQL
  - NoSQL: MongoDB (Document Store), Neo4j (Graph)

Prototyp

- Ziel: Beispielcode für Pinecone und Oracle
- Ablauf:



- Anm.:
  - Metrik jeweils Kosinusähnlichkeit
  - Bei Oracle: Initialisierung ≅ Tabelle anlegen

Pinecone (I)

(1) Führe Initialisierung durch

(2) Erstelle Index

• • •

```
( 1) from pinecone import Pinecone
( 2) from pinecone import ServerlessSpec
( 3) #1: Verbindung zu Pinecone herstellen
( 4) api_key = "YOUR_API_KEY"
( 5) pc = Pinecone (api_key=api_key)
```

```
( 6) # 2: Index anlegen
( 7) index_name = "vector-demo"
( 8) if index_name not in pc.list_indexes().names():
( 9)     pc.create_index(
(10)       name=index_name, dimension=2,
(11)       metric="cosine",
(12)       spec=ServerlessSpec(
(13)         cloud="aws", region="us-east-1")
(14) )
```

Pinecone (II)

(3) Füge Beispielvektoren ein

```
(15) #3: Vektoren einfügen
(16) index = pc.Index("vector-demo")
(17) vectors = [("V_00", [-6.,4.]), ... ,
(18)                ("V_21", [4.,0.])]
(19) index.upsert(vectors)
```

(4) Führe Abfrage aus:
3NN zu Suchvektor (-1,-1)

```
(20) #4: Suche 2NN zu Suchvektor (-1,-1)
(21) query_vector = [-1,-1]
(22) result = index.query(
(23)     vector=query_vector, top_k=3,
(24)     include_values=True
(25) )
```

(5) Gebe Ergebnis aus

```
(26) #5: Ausgeben Ergebnis
(27) print("Suchergebnis:")
(28) for match in result["matches"]:
(29)   print(f"ID: {match['id']},
(30)         Score: {match['score']},
(31)         Vektor: {match['values']}")
```

Oracle (I)

(1) Führe Initialisierung durch

Initialisierung in RDB
≅ Tabelle anlegen

(2) Erstelle HNSW-Index

• • •

```
( 1) -- 1: Tabelle anlegen
( 2) CREATE TABLE myTable (
( 3)     id NUMBER PRIMARY KEY,
( 4)     name VARCHAR2(50),
( 5)     vec VECTOR(2)
( 6) );
```

```
( 7) -- 2: HNSW-Index erstellen
( 8) CREATE VECTOR INDEX myTable_vec_idx
( 9)   ON myTable(vec)
(10)   ORGANIZATION INMEMORY NEIGHBOR GRAPH
(11)   DISTANCE COSINE;
```

Oracle (II)

(3) Füge Beispielvektoren ein

```
(12) -- 3: Vektoren einfügen
(13) INSERT INTO myTable VALUES(1,'V_00','[-6,4]');
(14) ...
(15) INSERT INTO myTable VALUES
(16)                         (22,'V_21','[4,0]');
```

(4) Führe Abfrage aus:
3NN zu Suchvektor (-1,-1)

```
(17) -- 4: Suche 2NN zu Suchvektor (-1,-1)
(18) SELECT id, name, vec
(19) FROM   myTable
(20) ORDER BY COSINE_DISTANCE(vec,'[-1,-1]')
(21) FETCH FIRST 3 ROWS ONLY;
```

Resultat:

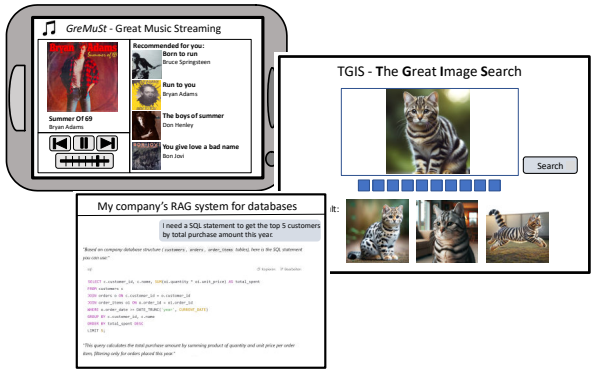
```
+-------+-----------+
|   name|        vec|
+-------+-----------+
|   V_17|[-5.0,-5.0]|
|   V_07|[-4.0,-4.0]|
|   V_02|[-5.0,-6.0]|
+-------+-----------+
```

Übersicht

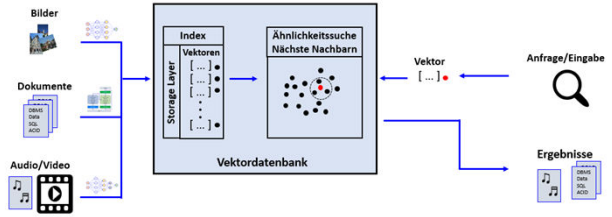
- Einführung
- Ähnlichkeit
- Suchen und Indexieren
- Realisierungen
- Zusammenfassung

Zusammenfassung

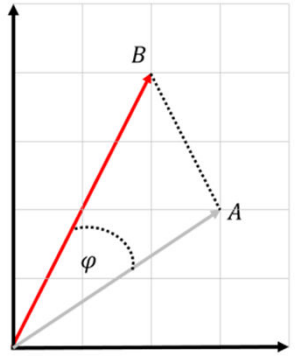
Use Cases



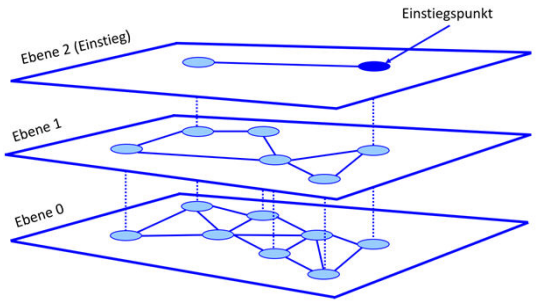
Architektur



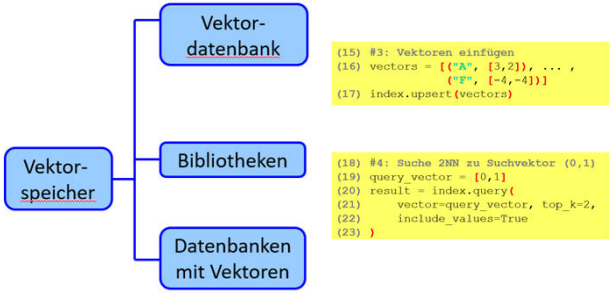
Ähnlichkeit



Suchen und Indexieren



Lösungen



Vielen Dank für Eure Aufmerksamkeit!

Fragen?

o.herden@hb.dhbw-stuttgart.de