# Item-based Collaborative Filtering

## Intermediate Presentation

**Martin Krüger, Sebastian Kölle**

09.06.2011

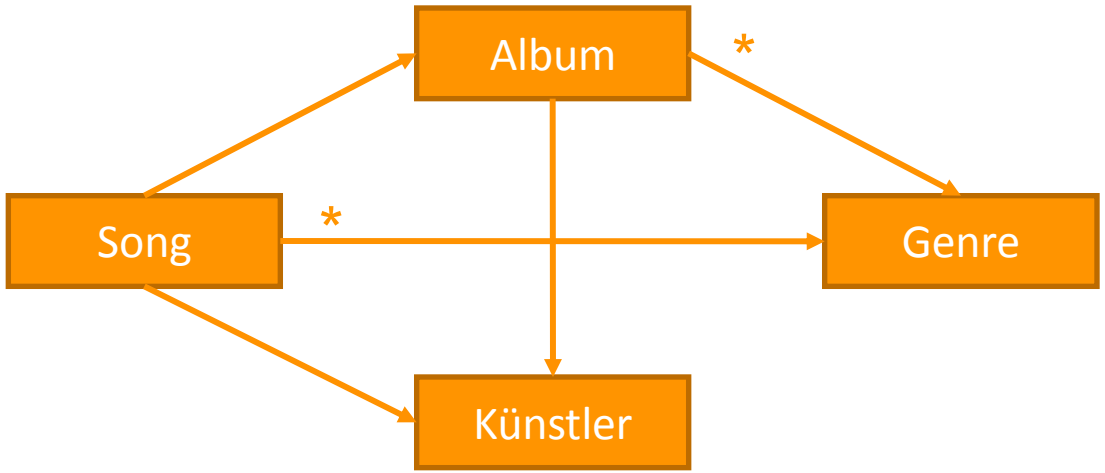
Seminar Collaborative Filtering

# KDD Cup 2011 – Track 1

**Daten**

| | |
|---|---|
| 1.000.990 | Nutzer |
| 624.961 | Items |
| 262.810.175 | Bewertungen (Training + Validierung + Test) |

**Items**



Ansatz

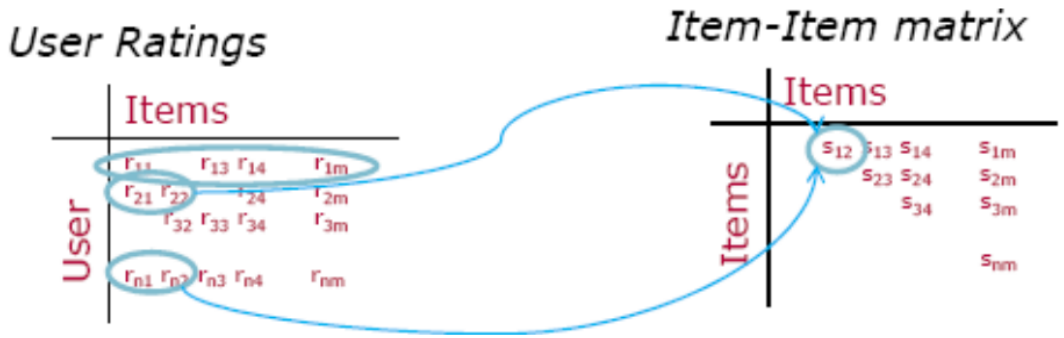
Aktueller Stand

Geplant

# Wdh.: Item-based Collaborative Filtering

**Vorberechnung**

Erstelle eine Item-Item Matrix, berechne dabei die Ähnlichkeit jedes Item-Paares unter Verwendung eines Ähnlichkeitsmaßes (*Cosinus-based*, *Correlation-based* oder *Adjusted Cosine similarity*).




**Vorhersage**

Gegeben: User **u**, Item **i**. Gesucht: Rating $\mathbf{r_{ui}}$

1. Finde die **K** zu **i** ähnlichsten Nachbarn **N(i;u)**, die von **u** bewertet wurden.
2. Berechne den gewichteten Mittelwert auf Basis der Ähnlichkeiten oder berechne das Rating mit einem Regressionsmodell.

[1, S. 6 ff]

# Herausforderungen für den KDD Cup

*„It has a very large set of items (over 600K) - much larger than any similar dataset, where usually only the number of users is large.“*

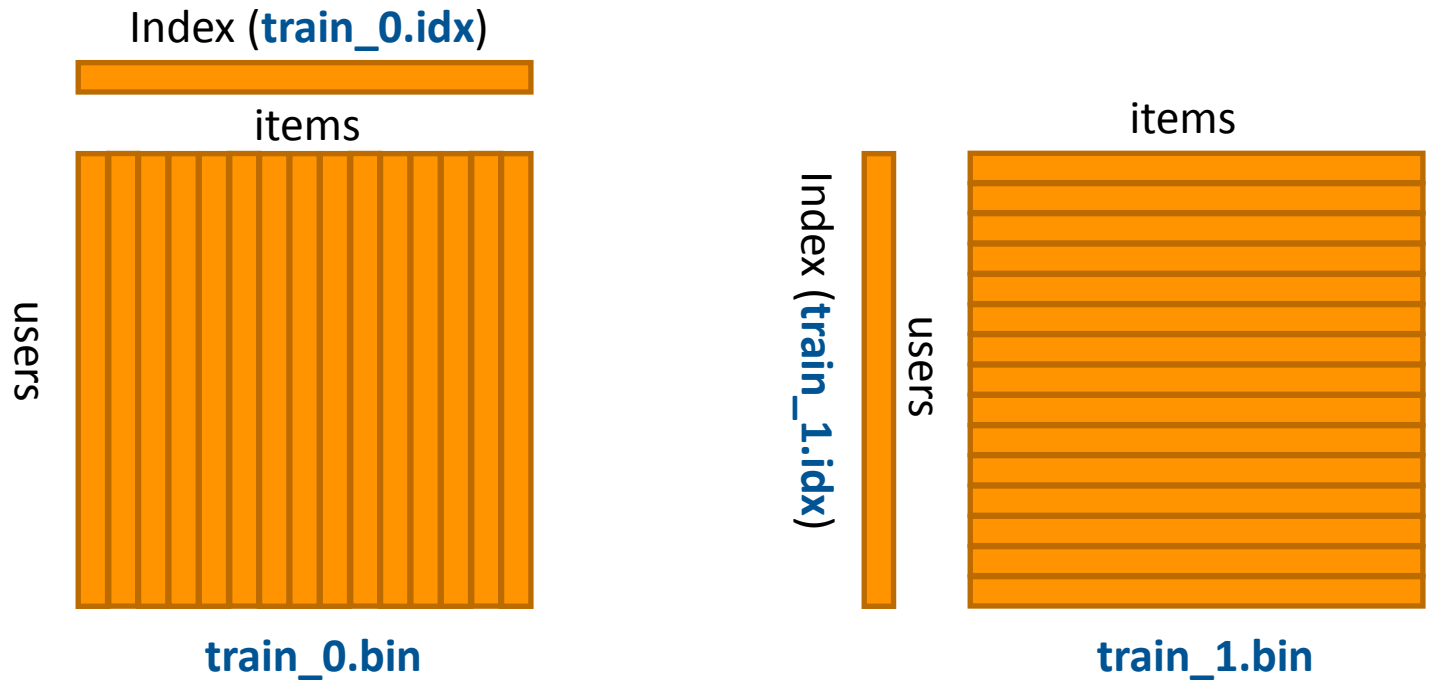
*Yahoo*



5

# Herausforderung: Speicherbedarf

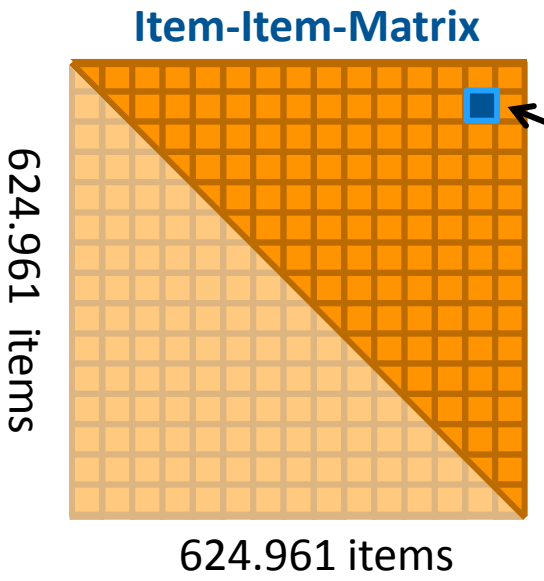
Vollständige User-Item-Matrix wäre ca. **582 GB** groß




**Lösung:** speicher-effizientes Matrix-Format (**1,9 GB**)

# Herausforderung: Laufzeit (I)

**1. Vorberechnung der Item-Item-Matrix**

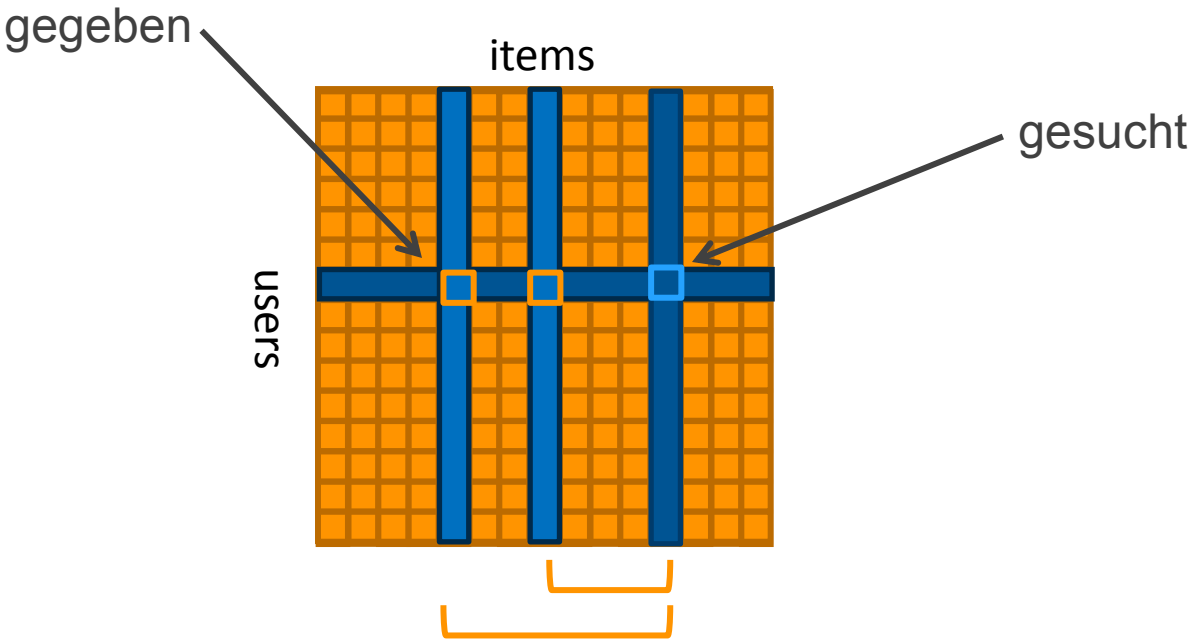



**4 Byte** Float
**0,45 ms** Berechnung

**Insgesamt**
Speicherbedarf **727,5 GB**
Laufzeit **1017,1 Tage**

# Herausforderung: Laufzeit (II)

**2. Ad-Hoc Berechnung**




**Sample mit 1000 Usern** (von 1.000.990):

- **4 - 10 min** C, Single-Threaded
- **15 - 35 min** Java, 4 Threads

**Gesamter Datensatz** (Hochrechnung):

- **> 67 Stunden** Validation-Set
- **> 100h** Test-Set

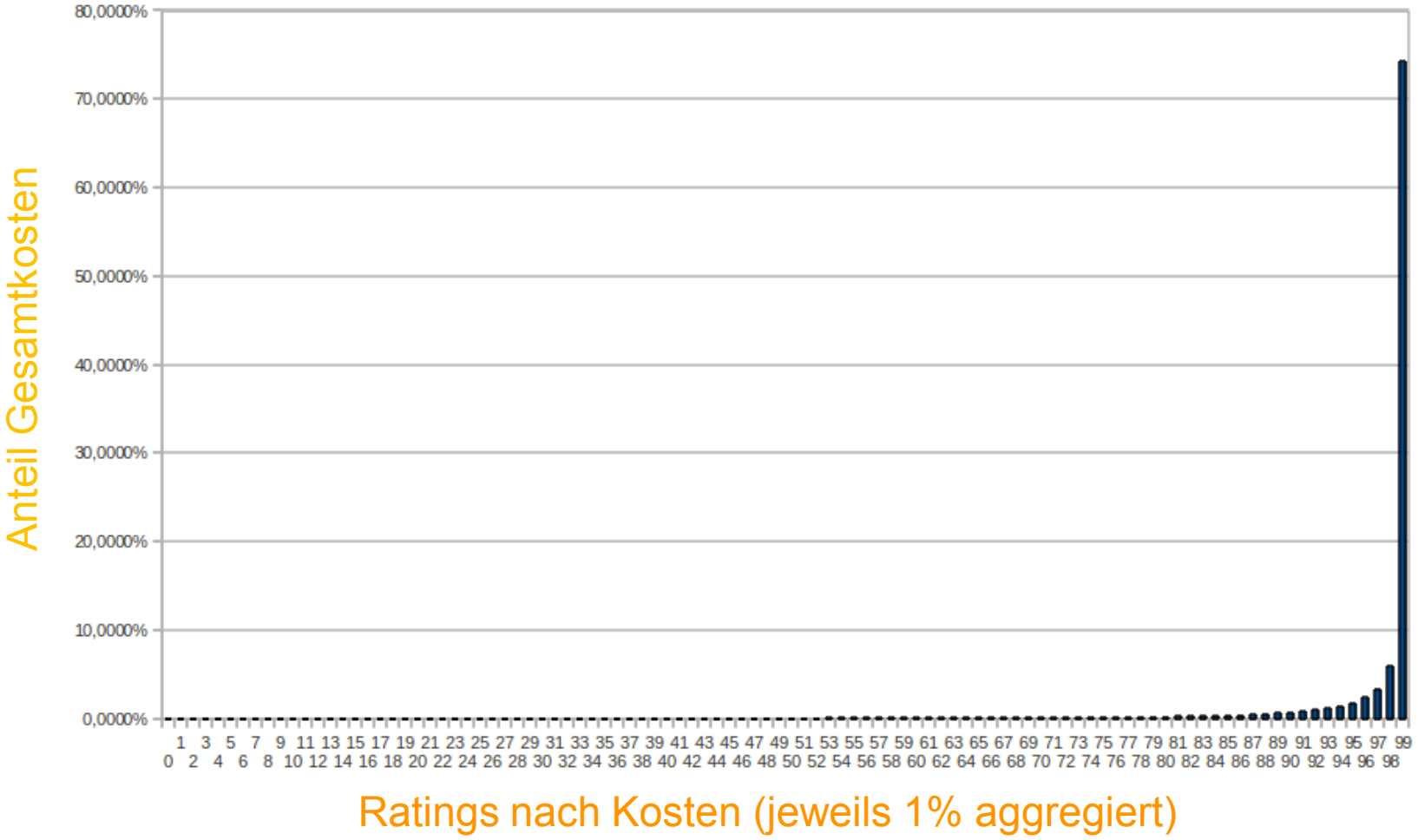
# Beobachtungen

1. Bottleneck: Ähnlichkeitsberechnung (**> 90%**)

2. **Pareto-Prinzip**: Ähnlichkeit weniger Item-Paare wird sehr häufig berechnet

3. Aufwand für die Ähnlichkeitsberechnung: **O(n + m)**

cost(pair) = count(pair) * (ratingCount(pair.item1) + ratingCount(pair.item2))



# Herausforderung: Laufzeit (III)

**3. Hybrid-Ansatz**

**1) Vorberechnung:**

Ähnlichkeit für die 89.273.180 (10%) „teuersten“ Item-Paare (683,5 MB)

**2) Prediction:**

Wenn Ähnlichkeit nicht vorberechnet, berechne sie ad-hoc

**Ergebnisse**

| | | |
|---|---|---|
| Ermitteln der teuersten Paare | **8 Stunden** | |
| Vorberechnung | **4 Stunden** | |
| Prediction auf Validation | **2 Stunden** | (RMSE: **24,37**) |
| Prediction auf Test | **3 Stunden** | (RMSE: **26,48**) |

Ansatz

Aktueller Stand

Geplant

# Aktueller Stand

| | RMSE (Validation) | RMSE (Test) |
|---|---|---|
| **Immer 68** | 32,9442 | 32,9609 |
| **IA * 1/4 + UA * 3/4** | 26.7055 | - |
| **UA * 1/3 + IA * 1/3 + (IA – (GA – UA)) * 1/3 in Grenzen [0;90]** | 26.3943 | 28.2045 |
| **Item-based (Adjusted Cosine)** | 24,3688 | 27,4018 |
| **Item-based, Training und Validation als Wissensbasis** | - | 26,4754 |

IA = ItemAverage; UA = UserAverage; GlobalAverage

# Aktueller Stand

**Item-Based**

- Ad-Hoc Ansatz in Java
- Hybrider Ansatz in C vollständig implementiert

- Durchlauf in akzeptabler Zeit
  → Austesten verschiedener Parameter möglich (Nachbarschaftsgröße, Mindestanzahl vergleichbarer Ratings, Mindestähnlichkeit)

**Hierarchy-Based**

- Hierarchie kann eingelesen und abgefragt werden.

| | |
|---|---|
| ID | → Typ |
| Track ID | → Album |
| Track-/Album ID | → Artist |
| Track-/Album ID | → Genres |
| Album ID | → Tracks |
| Artist ID | → Alben |
| Artist ID | → Tracks |

Ansatz

Aktueller Stand

Geplant

# Geplant

**Hierarchy-Based**

- Ermittlung verschiedener simpler Vorhersagen anhand der Hierarchie
- Gewichteter Mittelwert als Hierarchy-Based Prediction

| | Album | Artist | Genre | Track |
|---|---|---|---|---|
| **Items** | 88909 | 27888 | 992 | 507172 |
| **Items in validation** | 44546 | 18623 | 749 | 194776 |
| **Items in test** | 50690 | 20398 | 790 | 238031 |
| **Ratings in validation** | 425278 | 2087772 | 426761 | 1064149 |
| **Ratings in test** | 661317 | 3099881 | 521122 | 1723620 |

# Geplant - Hierarchie

**Track**
- Bewertung des Albums
- Bewertung des Künstlers
- Bewertung der Genres (Durchschnitt)
- Bewertung aller Tracks des gleichen Albums (Durchschnitt)
- Bewertung aller Tracks des gleichen Künstlers (Durchschnitt)

**Album**
- Bewertung der enthaltenen Items (Durchschnitt)
- Bewertung des Künstlers
- Bewertung der Genres (Durchschnitt)
- Bewertung von Alben des gleichen Künstlers (Durchschnitt)
- Bewertungen von Alben der gleichen Genres (Durchschnitt)

**Künstler**
- Bewertung aller Alben und Tracks des Künstlers (Durchschnitt)

**Genre**
- Bewertung von Alben und Tracks des Genres (Durchschnitt)

# Geplant

**Item-Based**

Austesten verschiedener Parameter für Nachbarschaftsgröße, Mindestanzahl vergleichbarer Ratings, Mindestähnlichkeit

**Shrinkage**

Verschiedene Gewichtungen für die verschiedenen Vorhersagen (Item-Based, Hierarchie-Based und Fallback) austesten

**Zusammenarbeit**

Wie unterschiedlich sind die Vorhersagen? (RMSE)

Wie gut sind die Vorhersagen des einen Verfahrens für die 'Ausreißer' des anderen?

Einfaches und falls Charakteristika erkennbar Mischen

# Fragen?

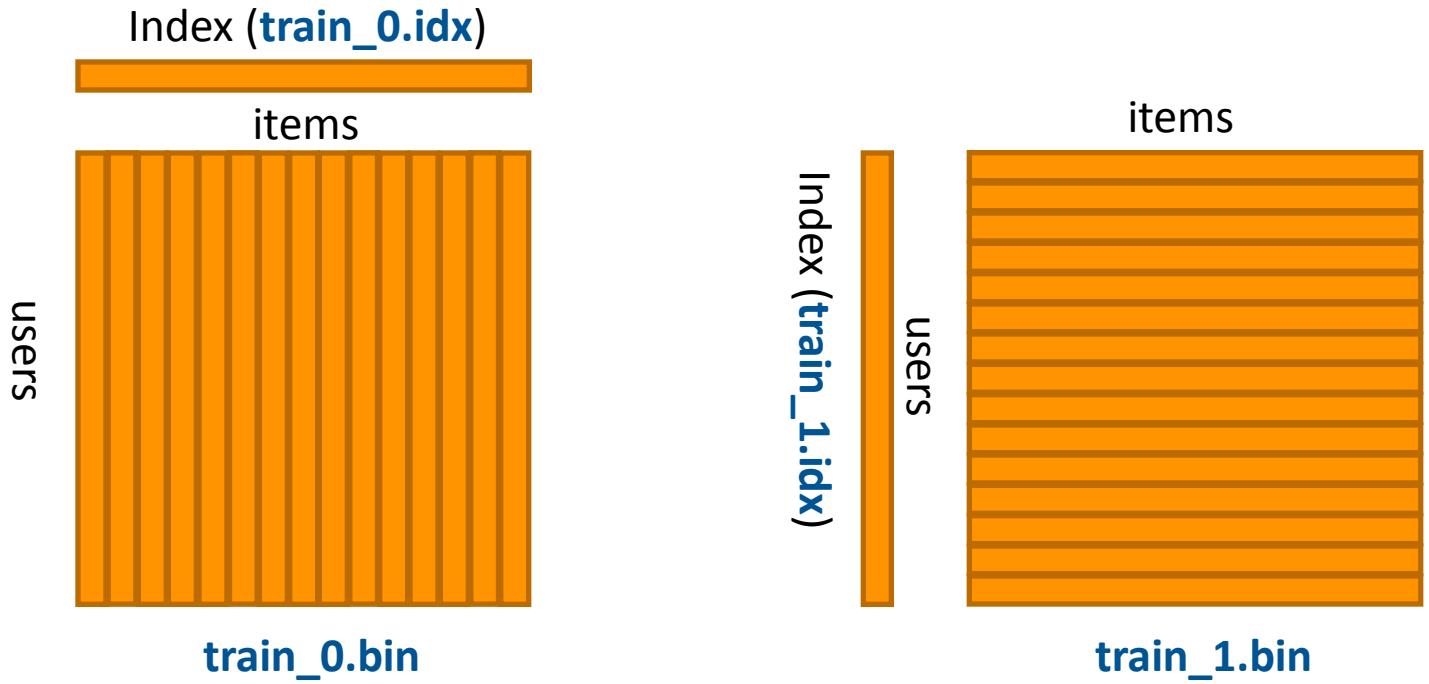
# Effiziente Speicherung der User-Item Matrix

- **Problem**: Größe der User-Item-Matrix
  - 1.000.990 Users x 624.961 Items x 1 Byte ≈ 582,62 GB
- Benötigte **Funktionalität**:
  - Alle Ratings eines Users ermitteln
  - Alle Ratings eines Items ermitteln
  - Gezielt ein einzelnes Rating ermitteln

  } O(1)

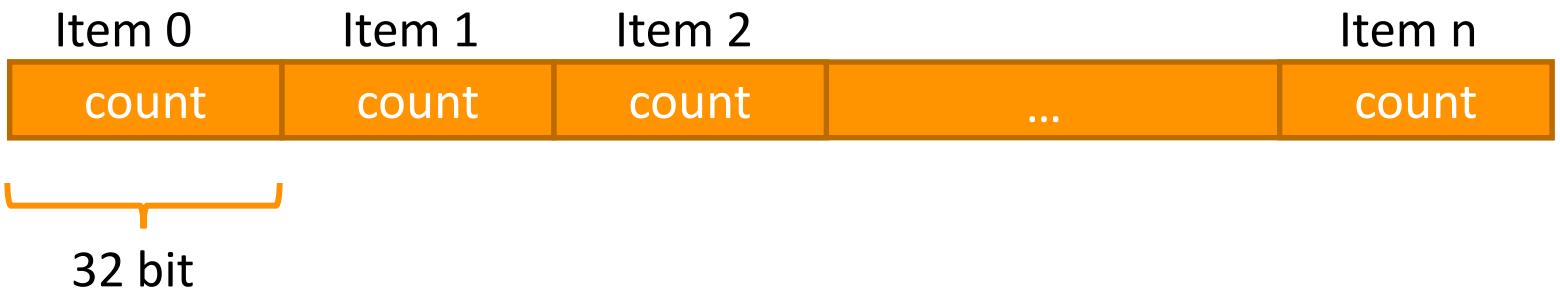
  } O(log (N/U))
- Aber: größtenteils gefüllt mit **Nullwerten**
  - Nur 0,04% der Matrixelemente haben einen Wert
- **Idee**:
  - Speichere nur die tatsächlich ‚gefüllten' Elemente
  - Verwende Indizes für effizienten Zugriff

# Datenformat

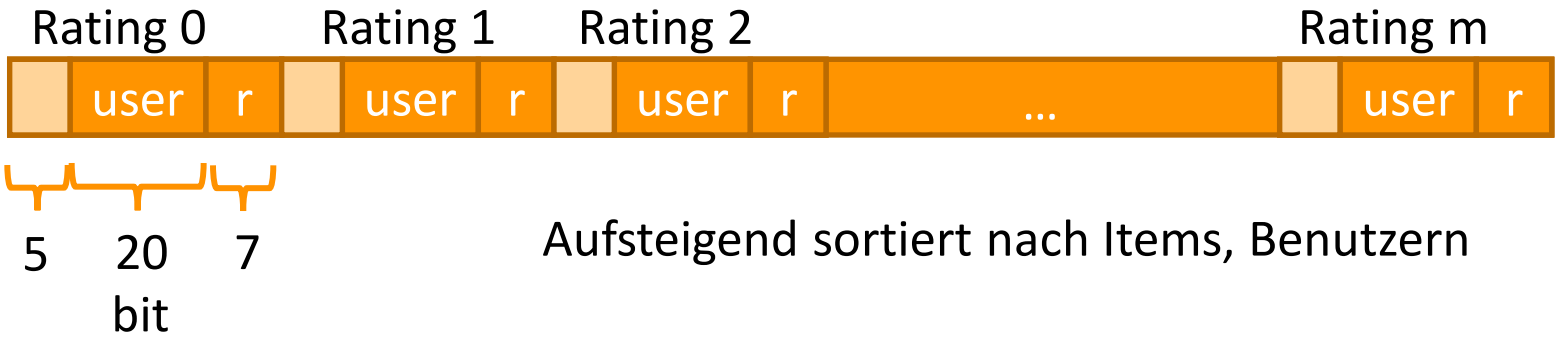


# Matrix train_0 auf der Festplatte

**train_0.idx**
(2,4 MB)




**train_0.bin**
(965 MB)




Aufsteigend sortiert nach Items, Benutzern

# Matrix train_0 im Hauptspeicher