



# Recommendersysteme

- Matrixfaktorisierungs-Recommender

# Recommendersysteme



- Information Overload:
  - $10^7$ - $10^8$  Bücher auf Amazon
  - Welche sind interessant für welchen Kunden?
- Anders als bei „Information Retrieval“ gibt der User keine Suchabfrage ein
- Ziel des Händlers: Umsatz steigern!
- Ziel formal: Wähle für jeden User  $k=5$  Items aus  $\sim 10^6$  Items, die ihm/ihr gefallen könnten
- Mögliche Zielformulierung: Gegeben eine Rating Matrix  $R$  mit Elementen  $r_{ui}$   
Meint User  $u$  bewertet Item  $i$  mit Rating  $r_{ui}$  in  $1, \dots, 5$   
Schätze ich die Ratings
$$\hat{r}_{ui} \sim r_{ui}$$
- In dieser Form ist es ein supervised ML Problem, aber mit besonderen Herausforderungen



# Herausforderungen

- Sparsity: User-Item-Matrix ist **sparse**: Nur sehr wenige Einträge sind bekannt (einige 10ppm?)

$$R = \begin{pmatrix} ? & ? & \dots & 4 \\ 1 & ? & \dots & 0 \\ ? & 0 & \dots & 0 \end{pmatrix}$$

- Scalability:
  - $\sim 10^6$  Items (Bücher, Musikstücke, News,...)
  - $\sim 10^6$  User
  - Brauche für jeden User k empfohlene Elemente
- Value of Time: Trainings-Ratings aus der Vergangenheit!
  - Bildet dies die Wünsche des Users jetzt gut ab?
- Evaluation: Wie finde ich raus, ob ein Recommender gut ist?
  - Accuracy vs. Diversity ?
- Cold Start: Was tun wenn ich noch nichts über den Benutzer weiss?

# Algorithmenklassen



- Wie sollen die Elemente ausgewählt werden? → Algorithmen
- Popularity Recommender: top-k
  - Keine Personalisierung!
- User-Ähnlichkeitsbasiert:
  - „Ähnliche Benutzer mögen ähnliche Items“?

## Die Schweizer Kinocharts

Woche vom 27.04.2017 bis 03.05.2017

1	▼ (17)	Guardians of the Galaxy Vol. 2 (2017)	★★★★★
2	▲ (1)	Fast & Furious 8 (2017)	★★★★★
3	▲ (2)	The Boss Baby (2017)	★★★★★
4	▲ (3)	Die göttliche Ordnung (2017)	★★★★★
5	▲ (4)	Going in Style (2017)	★★★★★

$$\hat{r}_{ui} = \bar{r}_u + \lambda \sum_{v \in U_u} s_{uv} (r_{vi} - \bar{r}_v)$$

- Aufgabe: Was ist wohl  $U_u$  ?
- Aufgabe: Was ist wohl  $\bar{r}_v$  ?
- Aufgabe: Was tun, wenn nur implizite Ratings (gesehen oder nicht) verfügbar sind?

# Algorithmenklassen 2



- Item-Ähnlichkeitsbasiert: „Benutzer mag ähnliche Items zu denen die er bisher mochte“?

$$\hat{r}_{ui} = \frac{\sum_{j \in I_u} s_{ij} r_{uj}}{\sum_{k \in I_u} |s_{ik}|}$$

- Dimensionsreduktion: Approximiere  $R \sim \hat{R}$  mit nur wenigen Parametern
  - Matrixfaktorisierung
  - Singular-Value-Decomposition
- Diffusionsbasiert: Benutze Graph-Eigenschaften der Rating-matrix
  - Z.B. Google-PageRank
- Social filtering: Benutzer empfehlen sich gegenseitig Items
- Hybride Ansätze: Kombiniere Recommendersysteme
- ...



# Metriken zur Evaluation von RS

- Offline-Metriken: Benutze ein Trainingsset aus der Vergangenheit
- Fehlermasse wie im Machine Learning:

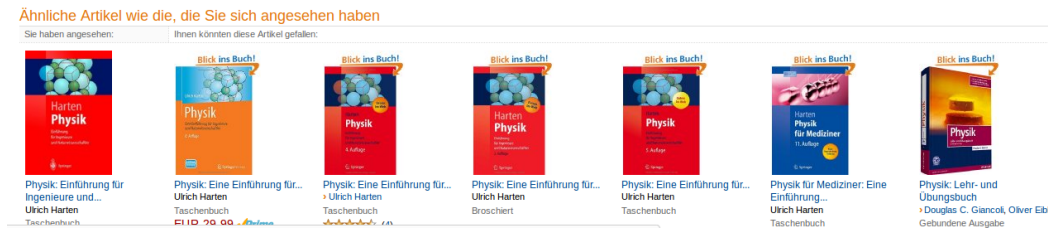
$$\text{MAE} = \frac{1}{|E^P|} \sum_{(i,\alpha) \in E^P} |r_{i\alpha} - \tilde{r}_{i\alpha}|,$$
$$\text{RMSE} = \left( \frac{1}{|E^P|} \sum_{(i,\alpha) \in E^P} (r_{i\alpha} - \tilde{r}_{i\alpha})^2 \right)^{1/2}.$$

- **Einwand:** Nur die Top-k Items sind wirklich relevant!
- Precision: Bruchteil der relevanten Items in den top-k
- Recall: Bruchteil der relevanten Items bzgl aller für den User relevanten Objekte
- Vergleiche immer mit Zufallsempfehlung

# Weitere Evaluationsmetriken



- Weitere Aspekte sind relevant und sollten quantifiziert werden
- Diversity: Vielseitigkeit der Empfehlungen



- Coverage: Wieviele Items kann der Recommender überhaupt empfehlen?
- ...

# Matrixfaktorisierung



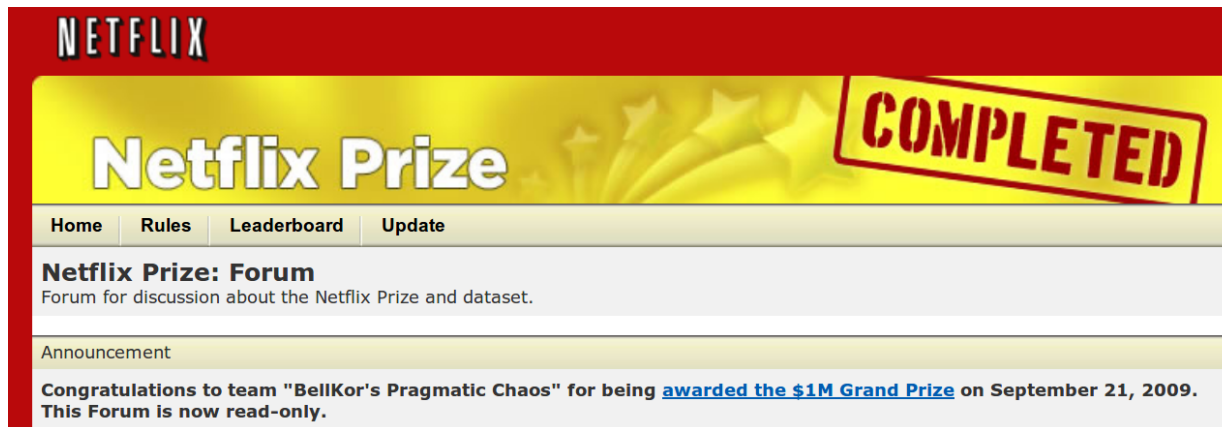
The screenshot shows the Netflix Prize website. At the top is a red banner with the "NETFLIX" logo in white. Below this is a yellow banner with the text "Netflix Prize" in large white letters. To the right of the text is a red stamp that says "COMPLETED". Below the yellow banner is a navigation bar with links: "Home", "Rules", "Leaderboard", and "Update". Below the navigation bar is a section titled "Netflix Prize: Forum" with the subtitle "Forum for discussion about the Netflix Prize and dataset." Below this is an "Announcement" section with the text: "Congratulations to team 'BellKor's Pragmatic Chaos' for being [awarded the \\$1M Grand Prize](#) on September 21, 2009. This Forum is now read-only."



# Matrixfaktorisierung



- Gewinner der Netflix Competition:  
Matrix-Faktorisierung als wichtige Komponente



- „Baseline predictor“:

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,\text{Bin}(t_{ui})}) \cdot c_u(t_{ui})$$

- Matrix Faktoris

$$\hat{r}_{ui} = b_{ui} + q_i^T \left( p_u(t_{ui}) + |\mathbf{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}(u)} y_j \right)$$

# Matrixfaktorisierung durch SVD



- SVD: Singulärwertzerlegung
- **Aufgabe:** Bearbeiten Sie das Notebook `Matrix_Faktorisierung_SVD.ipynb`
- Typischerweise haben wir sehr viel mehr Benutzer und Items als in diesem Ansatz.
- Wie geht das mit Big Data? Z.B. Hadoop MapReduce, Spark?
  - Hadoop/Spark: Charakterisiert durch viele unabhängige Nodes und ein eher langsames Netzwerk



# Matrixfaktorisierung Theorie 1

- Modelliere Ratings ( $p_u$  und  $q_i$  sind Vektoren mit  $k$  Komponenten):

$$\hat{r}_{ui} = p_u^T q_i = \sum_{k=1}^K p_{uk} q_{ki}$$

- Idee: Beschreibe jedes Item und jeden User durch  $k$  Zahlen (z.B.  $k=5$ ; nicht  $k=1000$ !)

**Latente Faktoren für User und Items:**

$$\vec{p}_u = \begin{pmatrix} p_{u1} \\ p_{u2} \\ \vdots \\ p_{uk} \end{pmatrix} \quad \vec{q}_i = \begin{pmatrix} q_{i1} \\ q_{i2} \\ \vdots \\ q_{ik} \end{pmatrix}$$

# Matrixfaktorisierung Theorie 2



- Rating ist das Skalarprodukt

$$\hat{r}_{ui} = \vec{p}_u \cdot \vec{q}_i = p_{u1}q_{i1} + p_{u2}q_{i2} + \dots + p_{uk}q_{ik}$$

- Die Matrix  $\mathcal{R}$  wird so durch eine Matrix  $\hat{\mathcal{R}}$  mit Rang k bestmöglich angenähert.

# MatrixMatrixfaktorisierung Theorie 3



- Modelliere Ratings:

$$\hat{r}_{ui} = p_u^T q_i = \sum_{k=1}^K p_{uk} q_{ki}$$

- Kostenfunktion mit Regularisierungstermen:

$$e^2 = \sum_{u,i} (r_{ui} - \hat{r}_{ui})^2 + \lambda \sum_{u,k} p_{uk}^2 + \lambda \sum_{i,k} q_{ik}^2$$

- Minimiere  $e^2(p, q)$

$$\frac{\partial}{\partial p} e^2(p, q) = 0$$



# Matrixfaktorisierung Theorie 4

- Halte  $q$  konstant, variere  $p$ : wo ist das Minimum?

- Ableitung: 
$$0 = \frac{\partial}{\partial p_{uk}} e^2(p, q) = \dots$$
$$= -2q_{\bar{k}i} \left( r_{ui} - \sum_m p_{u\bar{m}} q_{\bar{m}i} \right) + 2\lambda p_{uk}$$

- Dies lässt sich nach  $p$  auflösen!

$$p_u = \left( \sum_{i': r_{ui'} \neq \emptyset} q_{i'} q_{i'}^T + \lambda 1_{k \times k} \right)^{-1} \sum_{i: r_{ui} \neq \emptyset} q_i r_{ui}$$

- Struktur: neues  $p = (k\text{-mal-}k \text{ Matrix})^{-1} * (q_1 r_{u1} + q_2 r_{u2} + \dots)$
- Genau gleich, wenn  $p$  konstant und nach  $q$  abgeleitet wird

# Kochrezept!



- Beispiel für  $k=5$ :  $p$  und  $q$  sind Vektoren mit 5 Komponenten
- Initialisiere beide mit zufälligen Werten
- Schritt 3: Gradientenabstieg: „Für alle Benutzer, rechne:“
- Stochastischer Gradientenabstieg: „Für einen zufälligen Benutzer, rechne:“
- Finde nun einen besseren Wert für den Latenten Faktor Vektor

$$p_u = \left( \sum_{i': r_{ui'} \neq \emptyset} q_{i'} q_{i'}^T + \lambda 1_{k \times k} \right)^{-1} \sum_{\bar{i}: r_{u\bar{i}} \neq \emptyset} q_{\bar{i}} r_{u\bar{i}}$$

- Wiederhole ab Schritt 3 für ein Item:

$$q_i = \left( \sum_{u': r_{u'i} \neq \emptyset} p_{u'} p_{u'}^T + \lambda 1_{k \times k} \right)^{-1} \sum_{\bar{u}: r_{\bar{u}i} \neq \emptyset} p_{\bar{u}} r_{\bar{u}i}$$

# Aufgabe: Skalierbarkeit



- Die Update-Regel

$$p_u = \left( \sum_{i': r_{ui'} \neq \emptyset} q_{i'} q_{i'}^T + \lambda 1_{k \times k} \right)^{-1} \sum_{\bar{i}: r_{u\bar{i}} \neq \emptyset} q_{\bar{i}} r_{u\bar{i}}$$

- Funktioniert wunderbar auf Spark! Annahmen:
  - $r$  ist sparse
  - $k$  ist sehr klein (z.B.  $k=5$ )
- Welche Daten müssen über das Netzwerk geschickt werden?
- Wie sollen die Daten auf den Nodes Verteilt werden, damit die Berechnung effizient wird?