Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Recommendation

Peter Haider

Was ist Recommendation?

- Empfehlung von Produkten, Informationen, Personen, Musik, …
- Ziel: Rausfilterung von unwichtigen / unerwünschten / unsympathischen Optionen; Finden von wichtigen / gefallenden / beliebten Optionen
- Basierend auf Transaktionen aus der Vergangenheit: Käufe, Bewertungen, Klicks, …
- Nutzerspezifisch: Keine globale Sortierung der Optionen, sondern persönliche Empfehlung

Beispiel 1

- Empfehlung von Produkten:
- Kunden eines Onlineshops klicken sich durch die Angebotspallette, legen manche Artikel in den Warenkorb
- Ziel: Vorschlag von Produkten, die den Kunden auch interessieren könnten
- bei Amazon: „Kunden die x kaufen, kaufen auch y“
- Optimierungskriterium: Umsatz- bzw. Gewinnmaximierung

Beispiel 2

- Empfehlung von Filmen:
- Jeder Nutzer bewertet Filme, die er gesehen hat, mit 1-5 Sternen
- Ziel: Vorhersage, wie gut einem Nutzer die Filme gefallen würden, die er noch nicht gesehen/bewertet hat
- Erhoffter Nutzen: Man sieht sich nur noch Filme an, die einem auch gefallen

Netflix Prize

- Wettbewerb von 2006-2009
- 500000 Benutzer, 18000 Filme, 100 Mio. Bewertungen
- Aufgabe: Bewertungen eines Test-Sets vorhersagen, Genauigkeit des Netflix-eigenen Systems um mindestens 10% übertreffen
- Preis: 1 Mio. $

Allgemeine Problemstellung

- Menge von Nutzern $U = \{u_1,\ldots,u_k\}$
- Menge von Objekten $X = \{x_1,\ldots,x_m\}$
- Menge von Bewertungen $B = \{(u^1,x^1,y^1),\ldots, (u^n,x^n,y^n)\}$
- Bewertungslabels: $y^i \in Y$
- Bewertungsraum: Y
  - z.B.: Y = {-1 (dislike), +1 (like)}
  - oder: $Y = \{1, 2, 3, 4, 5\}$ (Sterne)
- Verlustfunktion $L : y_{\text{wahr}}\, x\, y_{\text{vorhergesagt}} \rightarrow R$
  - z.B.: L(like, like) = *0*, L(dislike, like) = *1* (schlechten Film gesehen), L(like, dislike) = *0,5* (guten Film verpasst), L(dislike, dislike) = *0*

Allgemeine Problemstellung

- Gesucht: Entscheidungsfunktion f, die das erwartete Risiko minimiert:

$$f^* = \arg\min_f \int\int\int L(y, f(u,x)) P(u,x,y) dy dx du$$

- Wie immer: Gemeinsame Verteilung über Beispiele und Labels $P(u,x,y)$ nicht bekannt
- Deswegen: Minimierung des regularisierten empirischen Risikos auf den Trainingsbeispielen:

$$f^* = \arg\min_f \sum_{i=1}^{n} L(y^i, f(u^i, x^i)) + \lambda\Omega(f)$$

Allgemeine Problemstellung

- Wichtige Unterscheidung:
- u^i: Benutzer des i-ten Trainingsbeispiels
- u_j: j-ter Benutzer
- x^i: Objekt des i-ten Trainingsbeispiels
- x_k: k-tes Objekt
- Jeder Benutzer und jedes Objekt kann in mehreren Trainingsbeispielen vorkommen

Matrixdarstellung

- Matrix der Trainingsbeispiele:

$$B = \begin{array}{c} \begin{matrix} u_1 & u_2 & u_3 & u_4 \end{matrix} \\ \begin{bmatrix} y^1 & ? & y^2 & ? \\ ? & ? & y^3 & ? \\ y^4 & y^5 & ? & y^6 \end{bmatrix} \end{array} \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix}$$

$$(u^1, x^1) = (u_1, x_1), \quad (u^5, x^5) = (u_2, x_3)$$

Inhaltsbasierte Empfehlung

- Idee: Filme, die ähnlich sind wie andere Filme, die einem gefallen, gefallen einem wahrscheinlich auch selber
- Voraussetzung für Umsetzbarkeit: Wissen über den Inhalt eines Films, z.B.
  - Tags
  - Genre
  - mitwirkende Schauspieler
  - Regisseur
  - …

Inhaltsbasierte Empfehlung

- Information über Filminhalt wird repräsentiert als Merkmalsvektor
- $\phi(x) = (x.\text{Drama}, x.\text{Action}, \ldots, x.\text{Jahr},$
  $[\text{„William Shatner“} \in x.\text{Cast}],$
  $[\text{„Sam Worthington“} \in x.\text{Cast}], \ldots)^T$
- $\phi(\text{Avatar}) = (0, 1, \ldots, 2009, 0, 1, \ldots)^T$

Inhaltsbasierte Empfehlung

- Menge von Nutzern $U = \{u_1,\ldots,u_k\}$
- Menge von Objekten $X = \{x_1,\ldots,x_m\}$
- Menge von Bewertungen $B = \{(u^1,x^1,y^1),\ldots, (u^n,x^n,y^n)\}$
- Bewertungslabels: $y^i \in Y$
- Bewertungsraum: Y
- Verlustfunktion $L : y_{\text{wahr}}\, x\, y_{\text{vorhergesagt}} \rightarrow R$
- Featureabbildung $\phi: X \rightarrow R^d$

Unabhängige Lernprobleme

- Einfachste Herangehensweise: Für jeden Benutzer separate Entscheidungsfunktion lernen
- Aufspaltung des Optimierungskriteriums:

$$f^* = \arg\min_f \sum_{i=1}^{n} L(y^i, f(u^i, x^i)) + \lambda\Omega(f)$$

$$\forall u_j : f^*_{u_j} = \arg\min_{f_{u_j}} \sum_{i:u^i=u_j}^{n} L(y^i, f_{u_j}(x^i)) + \lambda\Omega(f_{u_j})$$

Ein Optimierungsproblem pro Nutzer

nur abhängig von Bewertungen des einen Nutzers

Entscheidungsfunktion hat nur das Objekt als Argument

Unabhängige Lernprobleme

- Entscheidungsfunktion für einen Nutzer $f_u(x)$ kann ganz gewöhnlich modelliert werden
- Zum Beispiel: Lineare Funktion, parametrisiert mit Gewichtsvektor

$$f_{w_u}(x) = w_u^T \phi(x)$$

- Mit welchem Algorithmus der Gewichtsvektor gelernt wird, hängt von der Verlustfunktion L ab

Unabhängige Lernprobleme

- Wenn L der quadratische Verlust $L(y,y')=(y-y')^2$ ist, ergibt sich für jeden Benutzer ein Regressions-Optimierungsproblem:

$$\min_{w_u} \sum_{i:u^i=u}^{n} (y^i - w_u^T \phi(x^i))^2 + \lambda \Omega(w_u)$$

- Lösbar mit Standardverfahren
- Insbesondere wenn der Regularisierer die quadratische Norm des Gewichtsvektors ist:

$$\Omega(w_u) = \|w_u\|^2$$

- Dann: analytische Lösung (Ableitung gleich 0 setzen, nach w auflösen…)

Unabhängige Lernprobleme

- Wenn L der 0/1-Verlust $L(y,y')=[[y \neq y']]$ ist, ergibt sich ein normales Klassifikationsproblem
  - $y'=\text{sign}(f(x))$
- *0/1*-Verlust ist nicht konvex, daher optimiert man eine konvexe obere Schranke, wie z.B. den Hinge-Loss $L^h(y,f(x))=\max\{0, 1-y\, f(x)\}$
- Daraus ergibt sich für jeden Benutzer ein SVM-Optimierungsproblem

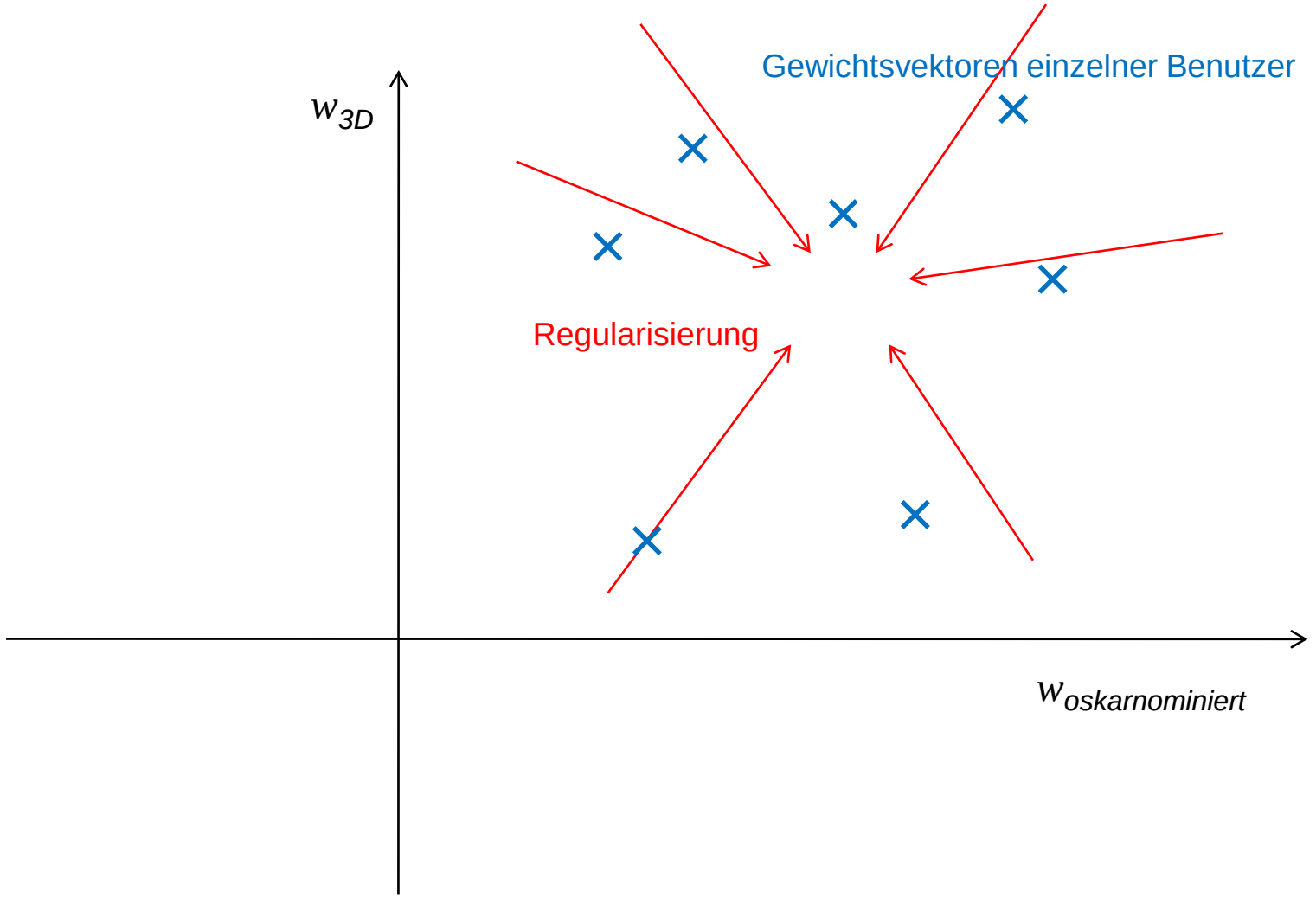
$$\min_{w_u} \sum_{i:u^i=u}^{n} \max\{0, 1 - y^i w_u^T \phi(x^i)\} + \lambda \|w_u\|^2$$

Gemeinsames Lernproblem

- Modellierung der inhaltsbasierten Empfehlung als nutzerspezifische, unabhängige Lernprobleme zwar einfach, aber offensichtliche Nachteile:
  - Gemeinsamkeiten zwischen Nutzern werden nicht ausgenutzt
  - Nutzerspezifische Entscheidungsfunktionen profitieren nicht von Bewertungen anderer Benutzer
  - Schlechte Vorhersagen für Benutzer mit sehr wenigen Bewertungen
- Deshalb: Lieber ein gemeinsames Lernproblem für alle Benutzer, so dass Informationen zwischen den Benutzern ausgetauscht werden können

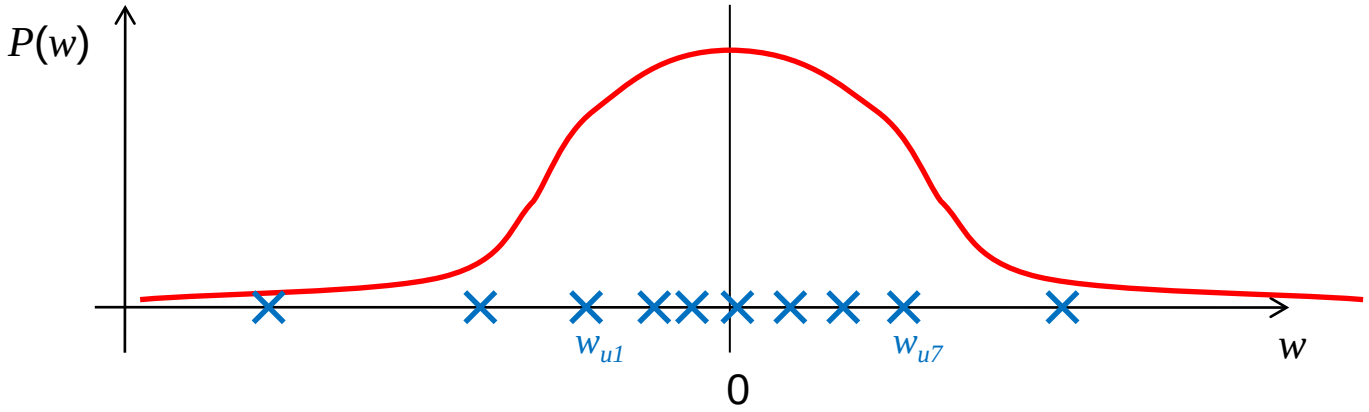
Unabhängige Lernprobleme:

Gemeinsames Lernproblem:

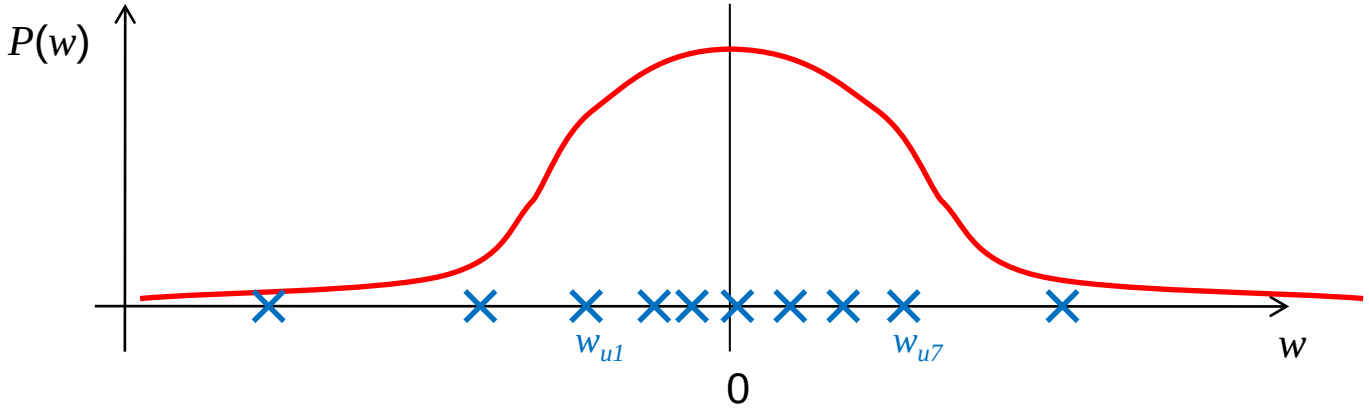


Gemeinsames Lernproblem

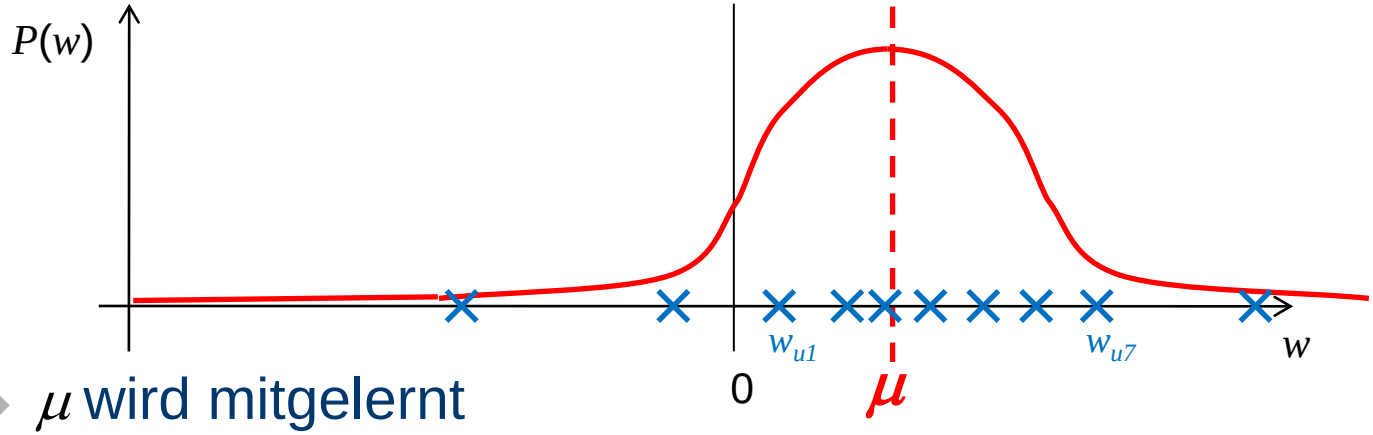
- Idee: Gewichtsvektoren verschiedener Benutzer ähneln sich
- Quadratischer Regularisierer entspricht Normalverteilung der Gewichtsvektoren um den Nullpunkt: $\lambda\|w_u\|^2 \equiv w_u \sim N(0, \frac{1}{\lambda}I)$



Gemeinsames Lernproblem

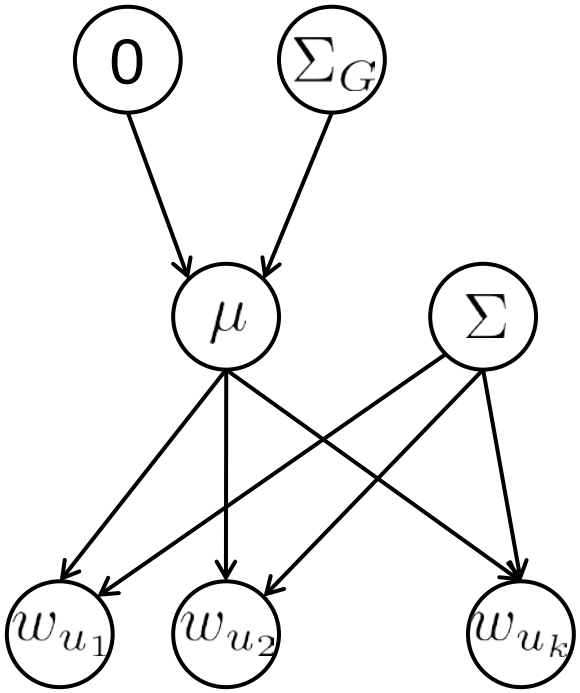


- Stattdessen: Normalverteilung um gemeinsamen Mittelpunkt μ



  - μ wird mitgelernt

Gemeinsames Lernproblem



Graphisches Modell des hierarchischen Priors

Gemeinsames Lernproblem

- Verteilung um gemeinsamen Mittelpunkt:

$$\forall_j : w_{u_j} \sim N(\mu, \tfrac{1}{\lambda'} I)$$

- Mittelpunkt ist selber wieder Normalverteilt:

$$\mu \sim N(0, \tfrac{1}{\lambda_G} I)$$

- Substitution: $w{:=}v{+}\mu$ (Grund: Verteilung um 0 einfacher zu handhaben)

$$\Rightarrow \forall_j : v_{u_j} \sim N(0, \tfrac{1}{\lambda'} I)$$

- Formulierung als quadratischer Regularisierer:

$$\sum_j \lambda' \|v_{u_j}\|^2 + \lambda_G \|\mu\|^2$$

Gemeinsames Lernproblem

- Das Optimierungsproblem ist dann:

$$\min_{\mu, v_{u_1}, \ldots, v_{u_k}} \sum_j \left[\sum_{i:u^i=u_j}^{n} L(y^i, (\underbrace{v_{u_j} + \mu}_{= w_{u_j}})^T \phi(x^i)) + \lambda' \|v_{u_j}\|^2 \right] + \lambda_G \|\mu\|^2$$

legt fest, wie stark sich die Gewichtsvektoren ähneln

globaler Regularisierungsparameter

- Aufgepasst: die v's sind voneinander unabhängig; nur die w's ähneln sich

Diskussion

- Vorteil: Jede benutzerspezifische Entscheidungsfunktion profitiert von allen Bewertungen aller Nutzer
- Nachteil: Es wird nicht berücksichtigt, dass Benutzer unterschiedliche Geschmäcker haben
  - Beispiel: die Empfehlungen bzw. Entscheidungsfunktionen für zwei Science-Fiction-Fans sollen sich ähneln, aber nicht die eines Action-Fans und eines Romantik-Fans
- Woran kann man erkennen, wie sehr sich die Geschmäcker zweier Benutzer ähneln?
  - an ihren Bewertungen!

Kollaborative Empfehlung

- Idee: Filme, die Personen gefallen, deren Geschmack dem eigenen Geschmack ähnlich ist, gefallen einem wahrscheinlich selbst.
- Und: Personen, deren Bewertung vieler Filme der eigenen Bewertung ähnlich ist, haben einen ähnlichen Geschmack wie man selbst.
- Keine Information über die Filminhalte notwendig

Kollaborative Empfehlung

- Menge von Nutzern $U = \{u_1,\ldots,u_k\}$
- Menge von Objekten $X = \{x_1,\ldots,x_m\}$
- Menge von Bewertungen $B = \{(u^1,x^1,y^1),\ldots, (u^n,x^n,y^n)\}$
- Bewertungslabels: $y^i \in Y$
- Bewertungsraum: Y
- Verlustfunktion $L : y_{\text{wahr}}\, x\, y_{\text{vorhergesagt}} \rightarrow R$

K-Nearest-Neighbor

- Definition einer Distanzfunktion zwischen Benutzern: $d(u, u')$
- Vorhersage für ein Tupel aus Benutzer und Objekt ergibt sich aus den Vorhersagen der k nächsten Nachbarn des Benutzers, die dieses Objekt auch bewertet haben
- Kombination der k Bewertungen je nach Bewertungsraum Y
  - $Y=\{-1, +1\}$ -> Mehrheitsentscheidung
  - $Y=R$ -> Mittelwert

Distanzmaß: Beispiel

- Euklidischer Abstand

$$d(u, u') = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y(u, x_i) - y(u', x_i))^2}$$

Durchschnitt über alle Objekte ∘

Quadratischer Abstand zwischen Bewertungen

Objekte, die nicht von beiden bewertet wurden, weglassen

Erweiterungen

- Viele Erweiterungen möglich:
  - Bewertungen normalisieren (z.B. Mittelwert des Benutzers subtrahieren, durch Standardabweichung des Nutzers dividieren)
  - Einflüsse der Nachbarn gewichten mit inverser Distanz
  - alle Benutzer berücksichtigen (K=k) (macht nur mit Gewichtung Sinn)
  - Einflüsse der Nachbarn gewichten mit Anzahl der gemeinsam bewerteten Objekte
- Allgemeine Formel:

$$f(u,x) = \frac{\sum_{u' \neq u \wedge \exists i: u^i = u' \wedge x^i = x} w(u,u') y^i}{\sum_{u' \neq u \wedge \exists i: u^i = u' \wedge x^i = x} w(u,u')}$$

Gewicht des Einflusses von *u'* auf die Vorhersage für *u*

Beispiel

$$B = \begin{bmatrix} 5 & 4 & 5 \\ 4 & ? & 3 \\ 1 & 5 & ? \\ ? & 4 & 4 \end{bmatrix}$$

Franz | Sissi | Josef

Matrix
Zombieland
Titanic
Schindlers Liste

- Wie würde der Sissi „Zombieland“ gefallen?
  - Berücksichtigung aller anderen Benutzer
  - Gewichtung mit inverser euklidischer Distanz: $w(u, u') = 1/d(u, u')$

$$f(Sissi, Zombieland) = ?$$

$$f(u, x) = \frac{\sum_{u' \neq u \wedge \exists i: u^i = u' \wedge x^i = x} w(u, u') y^i}{\sum_{u' \neq u \wedge \exists i: u^i = u' \wedge x^i = x} w(u, u')}$$

Beispiel

$$B = \begin{matrix} \text{Franz} & \text{Sissi} & \text{Josef} \\ \end{matrix}$$

$$\begin{bmatrix} 5 & 4 & 5 \\ 4 & ? & 3 \\ 1 & 5 & ? \\ ? & 4 & 4 \end{bmatrix}$$

Matrix
Zombieland
Titanic
Schindlers Liste

$$f(Sissi, Zombieland) = \frac{\sum_{u \neq Sissi} \frac{B_{u,Zombieland}}{d(Sissi,u)}}{\sum_{u \neq Sissi} \frac{1}{d(Sissi,u)}}$$

$$= \frac{\frac{B_{Franz,Zombieland}}{d(Sissi,Franz)} + \frac{B_{Josef,Zombieland}}{d(Sissi,Josef)}}{\frac{1}{d(Sissi,Franz)} + \frac{1}{d(Sissi,Josef)}}$$

Beispiel

$$B = \begin{matrix} \text{Franz} & \text{Sissi} & \text{Josef} \\ \end{matrix}$$

$$\begin{bmatrix} 5 & 4 & 5 \\ 4 & ? & 3 \\ 1 & 5 & ? \\ ? & 4 & 4 \end{bmatrix}$$

Matrix
Zombieland
Titanic
Schindlers Liste

$$d(u, u') = \sqrt{\frac{1}{m} \sum_{i=1}^{m} (y(u, x_i) - y(u', x_i))^2}$$

$$d(Sissi, Franz) = \sqrt{\frac{1}{l} \sum_l (B_{Sissi,l} - B_{Franz,l})^2} = \sqrt{\frac{1}{2}[(4-5)^2 + (5-1)^2]} \approx 2.9$$

$$d(Sissi, Josef) = \sqrt{(\frac{1}{2}[(4-5)^2 + (4-4)^2])} \approx 0.7$$

$$f(Sissi, Zombieland) = \frac{\frac{4}{2.9} + \frac{3}{0.7}}{\frac{1}{2.9} + \frac{1}{0.7}} \approx 3.2$$

Diskussion

- K-Nearest-Neighbor und Erweiterungen sind sogenannte *memory-based*-Ansätze
  - es werden keine Modellparameter gelernt
  - alle Trainingsbeispiele müssen gespeichert werden
- Vorteil: Trainingsaufwand = 0
- Nachteil: Das eigentliche Optimierungskriterium (erwarteter Verlust) wird nicht optimiert
  - dafür braucht man ein Modell, das gelernt werden kann (*model-based* Ansätze)

Latente Features

- Idee: Statt Geschmacksähnlichkeit ad-hoc zu definieren (Distanzmaß), Features lernen, die den Geschmack repräsentieren
- Angenommen, man hätte für jeden Benutzer u einen Featurevektor ψ_u, der seinen Geschmack beschreibt
- Dann könnte man für jedes Objekt x einen Gewichtsvektor w_x lernen, der charakterisiert, wie gut das Objekt x die verschiedenen Geschmäcker trifft

Latente Features

- Oder umgekehrt:
  - Hätte man zu jedem Objekt x einen Featurevektor ϕ_x, der alle relevanten Eigenschaften enthält,
  - könnte man für jeden Benutzer einen Gewichtsvektor w_u lernen, der beschreibt, welche Eigenschaften einem Benutzer wie gut gefallen
- Wie bei der inhaltsbasierten Empfehlung: da nimmt man an, dass die Features $\phi(x)$ ausreichend sind
- Aber: In der Realität reicht es oft nicht aus, die gegebenen Attribute zu kennen, um zu wissen, ob einem ein Objekt gefällt

Latente Features

- Also: Weder Featurevektoren der Benutzer ψ_u noch der Objekte ϕ_x gegeben
  - (wir gehen zunächst davon aus, dass zu den Objekten gar keine Attribute gegeben sind)
- Lösung: Beides gleichzeitig lernen!
  - Sowohl die ψ_u als auch die ϕ_x sind freie Parameter
  - (Dimensionalität legt man fest)
  - Entscheidungsfunktion:

$$f(u, x) = \psi_u^T \phi_x$$

  - Interpretation: Inneres Produkt aus Benutzergeschmack und Objekteigenschaften

Latente Features

- Erinnerung: Minimierung des empirischen Verlustes

$$f^* = \arg\min_f \sum_{i=1}^{n} L(y^i, f(u^i, x^i)) + \lambda\Omega(f)$$

- Optimierungsproblem mit latenten Features:

  L2-regularisiertes kollaboratives Filtern

$$(\psi^*, \phi^*) = \arg\min_{\psi,\phi} \sum_{i=1}^{n} L(y^i, \psi_{u^i}^T \phi_{x^i}) + \lambda \left[\sum_j \|\psi_{u_j}\|^2 + \sum_l \|\phi_{x_l}\|^2 \right]$$

Feature-Vektoren von allen Benutzern und Objekten werden regularisiert

Latente Features

- Statt festen Featurevektoren und freien Gewichtsvektoren: Sowohl Features der Benutzer als auch der Objekte werden gelernt
- Bedeutung der Features wird nicht festgelegt
- Lerner findet selbständig, welche Features nützlich sind, um das Optimierungsziel zu erreichen

Matrix-Faktorisierung

- Alternative Sichtweise des Lernens von latenten Features
- Erinnerung: Matrixdarstellung der Trainingsbeispiele

$$B = \begin{array}{c} \begin{matrix} u_1 & u_2 & u_3 & u_4 \end{matrix} \\ \begin{bmatrix} y^1 & ? & y^2 & ? \\ ? & ? & y^3 & ? \\ y^4 & y^5 & ? & y^6 \end{bmatrix} \end{array} \begin{matrix} x_1 \\ x_2 \\ x_3 \end{matrix}$$

Matrix-Faktorisierung

- Zusammenfassung der Feature-Vektoren in Matrizen:

$$\Psi = \begin{bmatrix} \Psi_{1,1} & \cdots & \Psi_{k,1} \\ \vdots & \ddots & \vdots \\ \Psi_{1,D} & \cdots & \Psi_{k,D} \end{bmatrix} \qquad \Phi = \begin{bmatrix} \Phi_{1,1} & \cdots & \Phi_{m,1} \\ \vdots & \ddots & \vdots \\ \Phi_{1,D} & \cdots & \Phi_{m,D} \end{bmatrix}$$

$$= \psi_{u_1} \qquad = \phi_{x_1}$$

- Vorhersage für ein Beispiel:

$$f(u_j, x_l) = \psi_{u_j}^T \phi_{x_l} = \Psi_{u_j,.}^T \Phi_{x_l,.} = \left[\Psi^T \Phi\right]_{j,l}$$

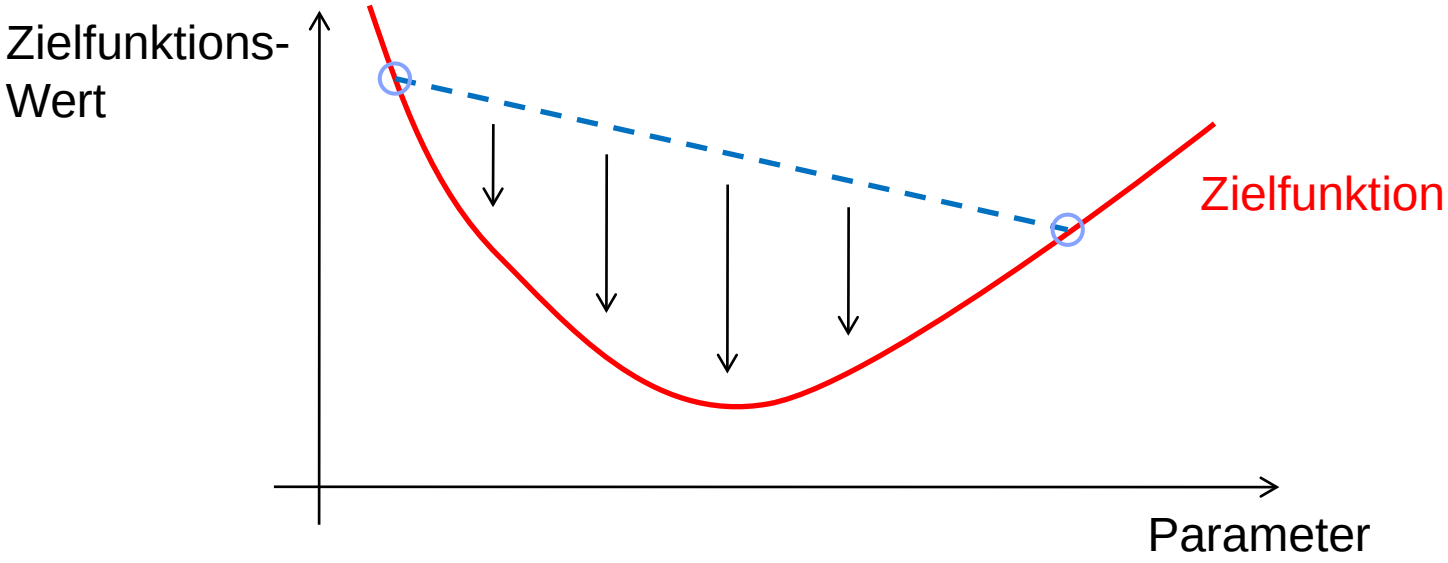
Matrix-Faktorisierung

- Produkt der Featurematrizen $\Psi^T \Phi$ ergibt Vorhersagen für alle Paare aus Benutzern und Objekten
- Ziel: $\Psi^T \Phi$ soll möglichst gut die Matrix der echten Bewertungen B approximieren (an den Stellen, an denen die Matrix B besetzt ist)
  - Ψ und Φ sind approximative Faktoren der Matrix B
- Indem man die Dimensionalität der Faktoren beschränkt, schränkt man den Rang der Produktmatrix $\Psi^T \Phi$ ein
  - Rausfilterung von Rauschen

Latente Features: Algorithmen

$$(\psi^*, \phi^*) = \arg\min_{\psi,\phi} \sum_{i=1}^{n} L(y^i, \psi_{u^i}^T \phi_{x^i}) + \lambda \left[\sum_j \|\psi_{u_j}\|^2 + \sum_l \|\phi_{x_l}\|^2 \right]$$

- Optimierungsproblem ist im Allgemeinen nicht konvex
  - Bedingung für Konvexität eines Optimierungskriteriums: Zwischen zwei Punkten im Raum Parameter x Funkionswert ist der Zielfunktionswert kleiner oder gleich der Verbindungslinie



Warum nicht konvex?

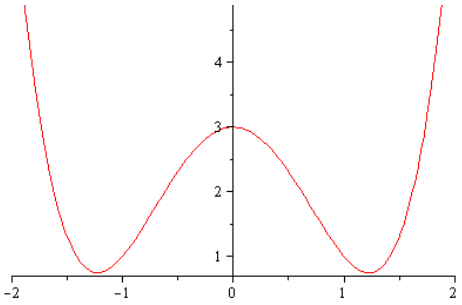
- Beim Optimierungskriterium mit latenten Features:
  - Multiplikation aller optimalen Feature-Vektoren mit -1 ändert die Werte der Entscheidungsfunktion nicht

$$f(u,x) = \psi_u^{*T}\phi_x^* = (-\psi_u^*)^T(-\phi_x^*)$$

  - Wert des Regularisierers bleibt auch konstant

$$\|\psi_{u_j}^*\|^2 = \|-\psi_{u_j}^*\|^2$$

  - Damit ändert sich auch der Wert der Zielfunktion nicht
  - Zwischen (ψ^*, ϕ^*) und $(-\psi^*, -\phi^*)$ liegt aber auch $(0,0)$, was i.A. keine optimale Lösung ist, also über der Verbindungslinie liegt
  - deshalb: nicht konvex



Latente Features: Algorithmen

- Optimierungsproblem hat mehrere lokale Optima
- Keine Garantie, dass man das globale Optimum findet
- Gute Ergebnisse liefert: Inkrementelles Aufbauen der Featurevektoren
  - `alle Featurevektoren mit 0 initialisieren`
  - `für alle Dimensionen d der Featurevektoren:`
    - `Dimension d zufällig initialisieren`
    - `bis zur Konvergenz:`
      - `Gradientenabstieg auf Dimension d der Benutzervektoren`
      - `Gradientenabstieg auf Dimension d der Objektvektoren`

Beispiel

Franz Sissi Josef

$$B = \begin{bmatrix} 5 & 4 & 5 \\ 4 & ? & 3 \\ 1 & 5 & ? \\ ? & 4 & 4 \end{bmatrix}$$

Matrix
Zombieland
Titanic
Schindlers Liste

1. Initialisierung:

$$\Psi = \begin{bmatrix} -0.7 & 0.2 & 0.5 \\ 0 & 0 & 0 \end{bmatrix} \qquad \Phi = \begin{bmatrix} 0.8 & -0.3 & -0.1 & 0.4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

zufällig

2. Optimierung der ersten Dimension (erste Zeile der Faktormatrizen)

Latente Features: Algorithmen

- 2. Optimierung der ersten Dimension (erste Zeile der Faktormatrizen)
  - Erste Zeile zufällig initialisieren
  - Ableitung des Optimierungskriteriums nach der ersten Zeile des Nutzerfaktors und Gradientenabstieg:

$$\frac{\partial}{\partial \Psi_{.,1}} \sum_{i=1}^{n} (y^i - [\Psi^T \Phi]_{u^i, x^i})^2 + \lambda \left[\|\Psi\|_F^2 + \|\Phi\|_F^2 \right]$$

  - Ableitung des Optimierungskriteriums nach der ersten Zeile des Objektfaktors und Gradientenabstieg
  - Wiederholen bis zur Konvergenz

Latente Features: Beispiel

$$B = \begin{bmatrix} 5 & 4 & 5 \\ 4 & ? & 3 \\ 1 & 5 & ? \\ ? & 4 & 4 \end{bmatrix}$$

- Nach der Optimierung der ersten Dimension:

$$\Psi = \begin{bmatrix} 1.9 & 2.1 & 2.1 \\ 0 & 0 & 0 \end{bmatrix} \quad \Phi = \begin{bmatrix} 2.3 & 1.5 & 1.3 & 1.8 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \Psi^T\Phi = \begin{bmatrix} 4.37 & 4.83 & 4.83 \\ 2.85 & 3.15 & 3.15 \\ 2.47 & 2.73 & 2.73 \\ 3.42 & 3.78 & 3.78 \end{bmatrix}$$

- Nach der Optimierung der zweiten Dimension:

$$\Psi = \begin{bmatrix} 1.9 & 2.1 & 2.1 \\ 1.1 & -1.4 & 0.4 \end{bmatrix} \quad \Phi = \begin{bmatrix} 2.3 & 1.5 & 1.3 & 1.8 \\ 0.5 & 0.8 & -1.5 & -0.2 \end{bmatrix} \quad \Psi^T\Phi = \begin{bmatrix} 4.92 & 4.13 & 5.03 \\ 3.73 & 2.03 & 3.47 \\ 0.82 & 4.83 & 2.13 \\ 3.20 & 4.06 & 3.70 \end{bmatrix}$$

- f(Sissi, Zombieland) = 2.03

Latente Features: Interpretation

- Jede Dimension des latenten Feature-Raums kann interpretiert werden anhand der Filme, die die höchsten bzw. niedrigsten Werte an dieser Stelle in ihrem Feature-Vektor haben
- Z.B. Dimension 2: $\Psi = \begin{bmatrix} 1.9 & 2.1 & 2.1 \\ 1.1 & -1.4 & 0.4 \end{bmatrix}$ $\Phi = \begin{bmatrix} 2.3 & 1.5 & 1.3 & 1.8 \\ 0.5 & 0.8 & -1.5 & -0.2 \end{bmatrix}$
  - höchste Werte: *Zombieland, Matrix*
  - niedrigste Werte: *Titanic, Schindlers Liste*
- Mögliche Interpretation: *Wie viel Action enthält der Film?*
- Entsprechend Dimension 2 der Benutzerfeatures: *Wie sehr steht der Benutzer auf Action-Filme?*
  - *Franz: 1.1, Sissi: -1.4, Josef: 0.4*

Netflix Prize: Ergebnis

- Gewinner: „BellKor's Pragmatic Chaos“
- Zusammenschluss mehrerer Teams
- Ensemble hunderter Methoden
- Verschiedenste Varianten von:
  - Latent-Feature-Modellen
  - Memory-Based-Methoden
- Veröffentlichung unzähliger Papers über Recommendation
- Neuer Standard-Datensatz zur Evaluierung von Recommendation-Methoden

Fragen?

Weiterführende Literatur

- B. Marlin. *Collaborative filtering: A machine learning perspective*. Master's thesis, University of Toronto, Canada, 2004.
- Y. Zhang, J. Koren: *Efficient Bayesian Hierarchical User Modeling for Recommendation Systems,* Proceedings of the 30th SIGIR conference, 2007
- Srebro, N. and Rennie, J.D.M. and Jaakkola, T.S.: *Maximum-margin matrix factorization*, Advances in neural information processing systems 17, 2005
- Paterek, A.: *Improving regularized singular value decomposition for collaborative filtering*, Proceedings of KDD Cup and Workshop, 2007