Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Recommendation

Tobias Scheffer

Recommendation Engines

- Recommendation of products, music, contacts, ..
- Based on user features, item features, and past transactions: sales, reviews, clicks, …
- User-specific recommendations, no global ranking of items.
- Feedback loop: choice of recommendations influences available transaction and click data.

Netflix Prize

- Data analysis challenge, 2006-2009
- Netflix made rating data available: 500,000 users, 18,000 movies, 100 million ratings
- Challenge: predict ratings that were held back for evaluation; improve by 10% over Netflix's recommendation
- Award: $ 1 million.

Problem Setting

- Users $U = \{1, \dots, m\}$
- Items $X = \{1, \dots, m'\}$
- Ratings $Y = \{(u_1, x_1, y_1) \dots, (u_n, x_n, y_n)\}$
- Rating space $y_i \in Y$
  - E.g., $Y = \{-1, +1\}$, $Y = \{\star, \dots, \star\star\star\star\star\}$
- Loss function $\ell(y_i, y_j)$
  - E.g., missing a good movie is bad but watching a terrible movie is worse.

- Find rating model: $f_\theta: (u, x) \mapsto y$.

Problem Setting: Matrix Notation

- Users $U = \{1, \dots, m\}$
- Items $X = \{1, \dots, m'\}$
- Ratings $Y = \begin{bmatrix} y_{11} & y_{12} & \\ y_{21} & & y_{23} \\ & & y_{33} \end{bmatrix}$

$$\begin{array}{cccc} \text{item } 1 & 2 & 3 & \\ y_{11} & y_{12} & & \text{user } 1 \\ y_{21} & & y_{23} & \text{user } 2 \\ & & y_{33} & \text{user } 3 \end{array}$$

Incomplete matrix

- Rating space $y_i \in \Upsilon$
  - E.g., $\Upsilon = \{-1, +1\}$, $\Upsilon = \{\star, \dots, \star\star\star\star\star\}$
- Loss function $\ell(y_i, y_j)$

Problem Setting

- Model $f_\theta(u,x)$
- Find model parameters that minimize risk
  $\theta^* = \text{argmin}_\theta \int\int\int \ell\big(y, f_\theta(u,x)\big)p(u,x,y)dxdudr$
- As usual: $p(u,x,y)$ is unknown → minimize regularized empirical risk

$$\theta^* = \text{argmin}_\theta \sum_{i=1}^{n} \ell(y_i, f_\theta(u_i,x_i)) + \lambda\Omega(\theta)$$

Content-Based Recommendation

- Idea: User may like movies that are similar to other movies which they like.
- Requirement: attributes of items, e.g.,
  - Tags,
  - Genre,
  - Actors,
  - Director,
  - …

Content-Based Recommendation

- Feature space for items
- E.g., $\Phi = (\text{comedy}, \text{action}, \text{year}, \text{dir tarantino}, \text{dir cameron})^\mathsf{T}$
- $\phi(\text{avatar}) = (0, 1, 2009, 0, 1)^\mathsf{T}$

Content-Based Recommendation

- Users $U = \{1, \dots, m\}$
- Items $X = \{1, \dots, m'\}$
- Ratings $Y = \{(u_1, x_1, y_1) \dots, (u_n, x_n, y_n)\}$
- Rating space $y_i \in Y$
  - E.g.,$\Upsilon = \{-1, +1\}$,$\Upsilon = \{\star, \dots, \star\star\star\star\star\}$
- Loss function $\ell(y_i, y_j)$
  - E.g., missing a good movie is bad but watching a terrible movie is worse.
- Feature function for items: $\phi: x \mapsto \mathbb{R}^d$
- Find rating model: $f_\theta: (u, x) \mapsto y$.

Intelligent Data Analysis II

Independent Learning Problems for Users

- Minimize regularized empirical risk

$$\theta^* = \text{argmin}_\theta \sum_{i=1}^{n} \ell(y_i, f_\theta(u_i, x_i)) + \lambda\Omega(\theta)$$

- One model per user:

$$f_{\theta_u}(x) \mapsto \Upsilon$$

- One learning problem per user:

$$\theta_u^* = \text{argmin}_{\theta_u} \sum_{i:u_i=u} \ell\left(y_i, f_{\theta_u}(x_i)\right) + \lambda\Omega(\theta_u)$$

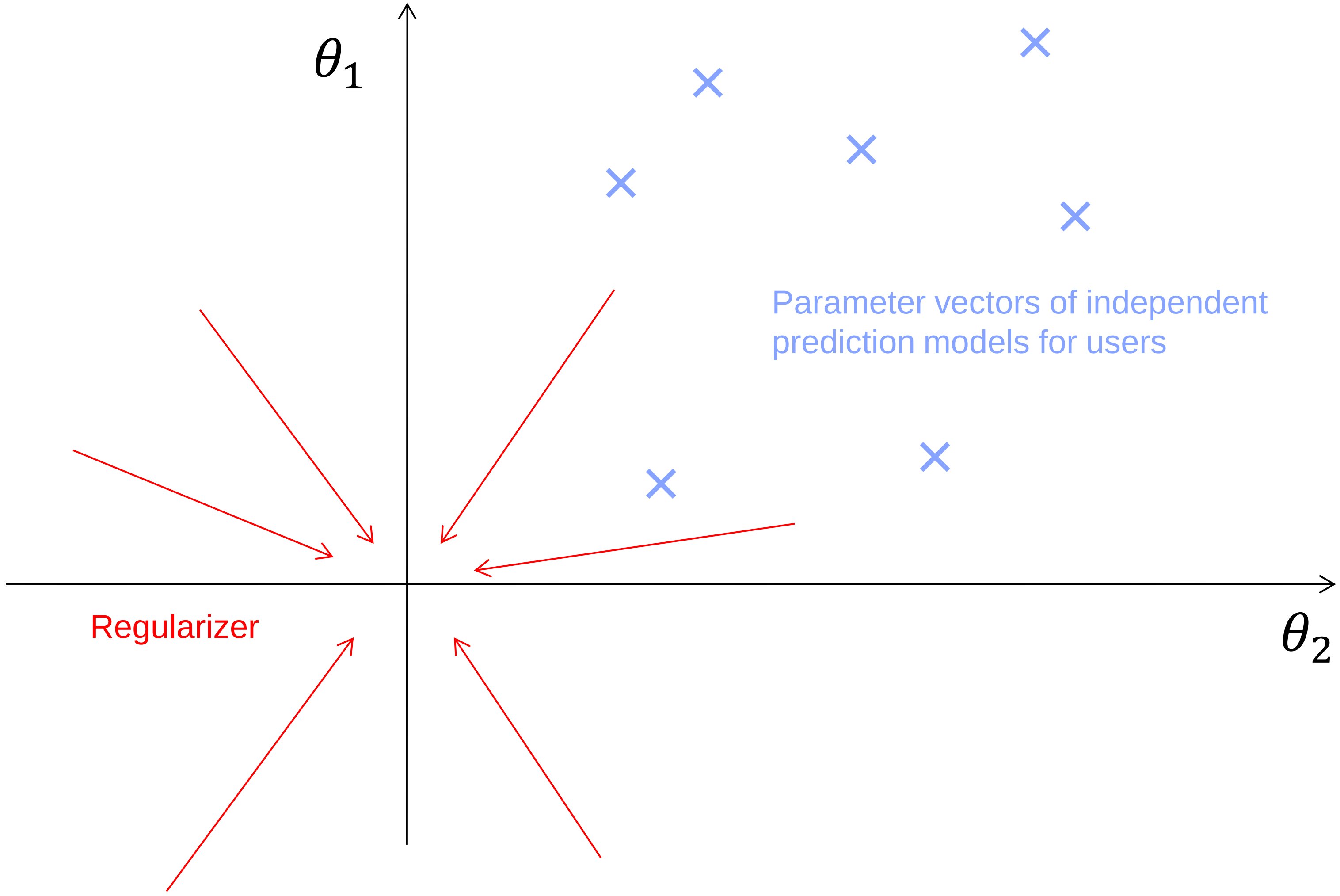
Independent Learning Problems for Users

- One learning problem per user:
  $\forall u: \theta_u^* = \operatorname{argmin}_{\theta_u} \sum_{i:u_i=u} \ell\left(y_i, f_{\theta_u}(x_i)\right) + \lambda\Omega(\theta_u)$
- Use any model class and learning mechanism; e.g.,
  - $f_{\theta_u}(x_i) = \phi(x_i)^{\mathrm{T}}\theta_u$
  - Logistic loss + ℓ_2 regularization: logistic regression
  - Hinge loss + ℓ_2 regularization: SVM
  - Squared loss + ℓ_2 regularization: ridge regression

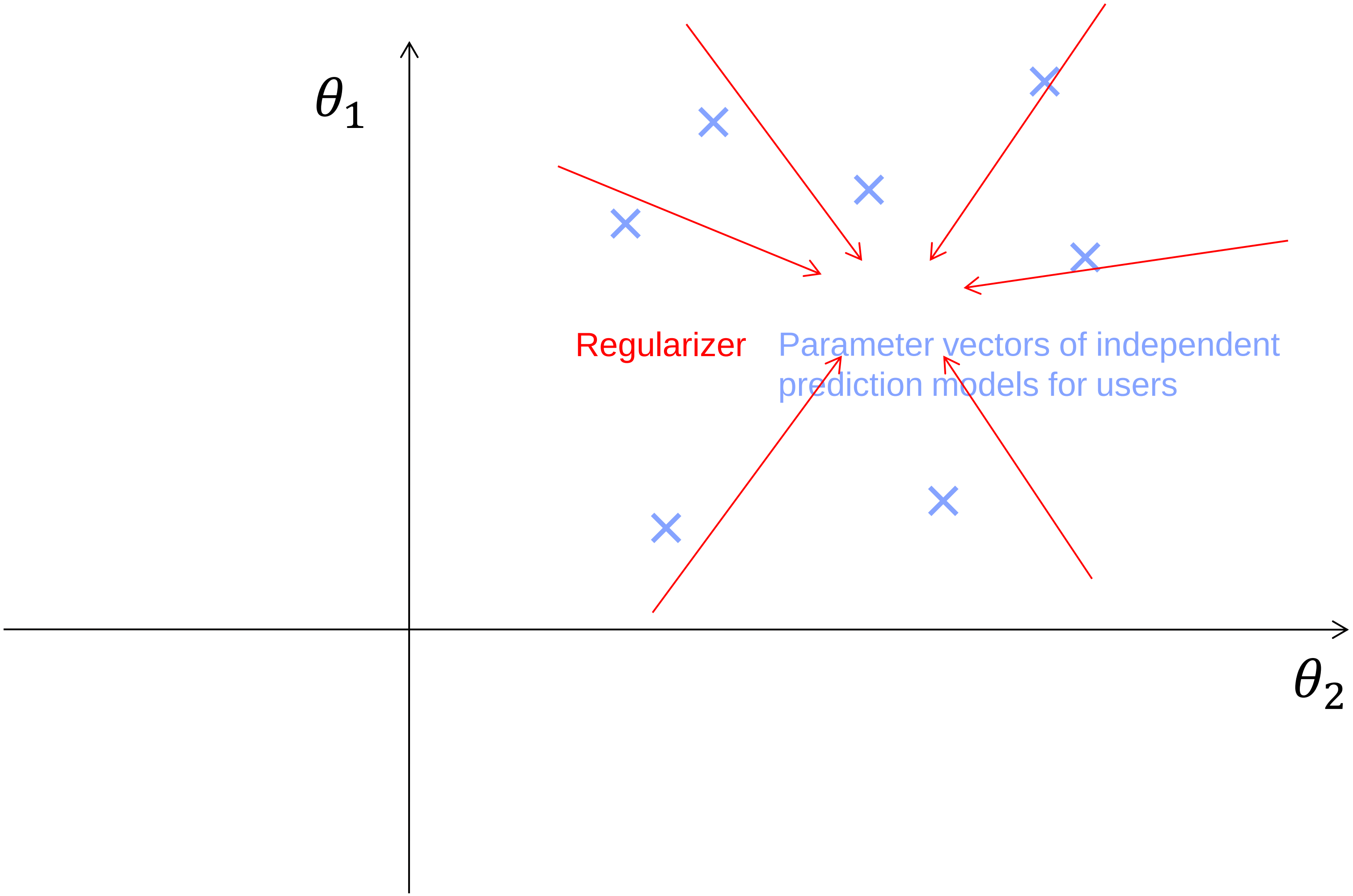
Independent Learning Problems for Users

- Obvious disadvantages of independent problems:
  - Commonalities of users are not exploited,
  - User does not benefit from ratings given by other users,
  - Poor recommendations for users who gave few ratings.
- Rather use joint prediction model:
  - Recommendations for each user should benefit from other osers' ratings.

Independent Learning Problems



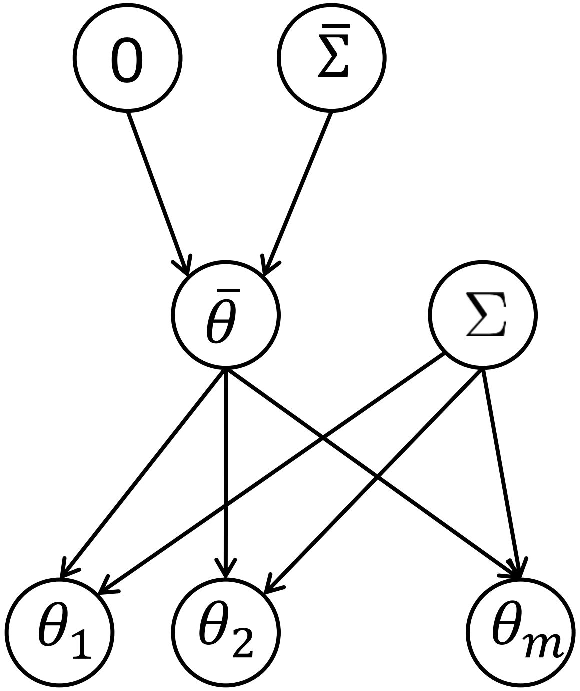
Joint Learning Problem



Joint Learning Problem

- Standard ℓ_2 regularization follows from the assumption that model parameters are governed by normal distribution with mean vector zero.
- Instead assume that there is a non-zero population mean vector.

Joint Learning Problem



Graphical model of hierarchical prior

Joint Learning Problem

- Population mean vector

$$\bar{\theta} \sim N\left[0, \frac{1}{\bar{\lambda}} I\right]$$

- User-specific mean vector:

$$\theta_u \sim N\left[\bar{\theta}, \frac{1}{\lambda} I\right]$$

- Substitution: $\theta_u = \bar{\theta} + \theta'_u$; now $\bar{\theta}$ and θ'_u have mean vector zero.
- -Log-prior = regularizer

$$\Omega(\bar{\theta} + \theta'_u) = \bar{\lambda}\left|\left|\bar{\theta}\right|\right|^2 + \lambda\left|\left|\theta'_u\right|\right|^2$$

Joint Learning Problem

- Joint optimization problem:

$$\min_{\theta'_1,\ldots,\theta'_m,\bar{\theta}} \sum_u \sum_{i:u_i=u} \ell\left(y_i, f_{\theta'_u+\bar{\theta}}(x_i)\right) + \lambda\Omega(\theta'_u) + \bar{\lambda}\Omega(\bar{\theta})$$

↑ $\theta_u = \bar{\theta} + \theta'_u$

↑ Coupling strength

↑ Global regularization

- Parameters θ'_u are independent, $\bar{\theta}$ is shared.
- Hence, θ_u are coupled.

Discussion

- Each user benefits from other users‘ ratings.
- Does not take into account that users have different tastes.
- Two sci-fi fans may have similar preferences, but a horror-movie fan and a romantic-comedy fan do not.
- Idea: look at ratings to determine how similar users are.

Collaborative Filtering

- Idea: People like items that are liked by people who have similar preferences.
- People who give similar ratings to items probably have similar preferences.
- This is independent of item features.

Collaborative Filtering

- Users $U = \{1, \ldots, m\}$
- Items $X = \{1, \ldots, m'\}$
- Ratings $Y = \{(u_1, x_1, y_1) \ldots, (u_n, x_n, y_n)\}$
- Rating space $y_i \in \Upsilon$
  - E.g.,$\Upsilon = \{-1, +1\}$,$\Upsilon = \{\star, \ldots, \star\star\star\star\star\}$
- Loss function $\ell(y_i, y_j)$

- Find rating model: $f_\theta: (u, x) \mapsto y$.

Collaborative Filtering by Nearest Neighbor

- Define distance function on users:
$$d(u, u')$$
- Predicted rating:
$$f_\theta(u, x) = \sum_{\substack{k \text{ nearest} \\ \text{neighbors } u_i \text{ of } u}} \frac{1}{k} y_{u_i, x}$$

- Predicted rating is the average rating of the k nearest neighbors in terms of $d(u, u')$.
- No learning involved.
- Performance hinges on $d(u, u')$.

Collaborative Filtering by Nearest Neighbor

- Define distance function on users:

$$d(u,u') = \sqrt{\frac{1}{m'}\sum_{x=1}^{m'}\left(y_{u',x}, -y_{u,x}\right)^2}$$

- Euclidean distance between ratings for all items.
- Skip items that have not been rated by both users.

Extensions

- Normalize ratings (subtract mean rating of user, divide by user's standard deviation)
- Weight influence of neighbors by inverse of distance.
- Weight influence of neighbors with number of jointly rated items.

$$f_\theta(u,x) = \frac{\sum_{\substack{k \text{ nearest} \\ \text{neighbors } u_i \text{ of } u}} \frac{1}{d(u,u_i)} y_{u_i,x}}{\sum_{\substack{k \text{ nearest} \\ \text{neighbors } u_i \text{ of } u}} \frac{1}{d(u,u_i)}}$$

Collaborative Filtering: Example

- $$Y = \begin{bmatrix} 4 & & 5 & 4 \\ 5 & 5 & 1 & \\ 5 & 3 & & 4 \end{bmatrix}$$

	Matrix	Zombiland	Titanic	Death Proof
Alice	4		5	4
Bob	5	5	1	
Carol	5	3		4

- How much would Alice enjoy Zombiland?

Collaborative Filtering: Example

M N T D

- $Y = \begin{bmatrix} 4 & & 5 & 4 \\ 5 & 5 & 1 & \\ 5 & 3 & & 4 \end{bmatrix}$ Alice Bob Carol

- $d(u, u') = \sqrt{\frac{1}{m'} \sum_{x=1}^{m'} \left(y_{u',x'} - y_{u,x}\right)^2}$
- $d(A, B) =$
- $d(A, C) =$
- $d(B, C) =$

Collaborative Filtering: Example

	M	N	T	D	
	4		5	4	Alice
	5	5	1		Bob
	5	3		4	Carol

- $Y = \begin{bmatrix} 4 & & 5 & 4 \\ 5 & 5 & 1 & \\ 5 & 3 & & 4 \end{bmatrix}$

- $d(u, u') = \sqrt{\frac{1}{m'} \sum_{x=1}^{m'} \left(y_{u',x}, -y_{u,x}\right)^2}$
- $d(A, B) = 2.9$
- $d(A, C) = 1$
- $d(B, C) = 1.4$

Collaborative Filtering: Example

	M	N	T	D	
	4		5	4	Alice
	5	5	1		Bob
	5	3		4	Carol

- $Y = \begin{bmatrix} 4 & & 5 & 4 \\ 5 & 5 & 1 & \\ 5 & 3 & & 4 \end{bmatrix}$

- $f_\theta(A, Z) = \dfrac{\sum_{\substack{\text{2 nearest} \\ \text{neighbors } u_i \text{ of } A}} \frac{1}{d(A, u_i)} y_{u_i, Z}}{\sum_{\substack{\text{2 nearest} \\ \text{neighbors } u_i \text{ of } A}} \frac{1}{d(A, u_i)}} =$

Collaborative Filtering: Example

M Z T D

- $Y = \begin{bmatrix} 4 & & 5 & 4 \\ 5 & 5 & 1 & \\ 5 & 3 & & 4 \end{bmatrix}$ Alice Bob Carol

- $f_\theta(A, Z) = \frac{\sum_{\substack{2 \text{ nearest} \\ \text{neighbors } u_i \text{ of } A}} \frac{1}{d(A, u_i)} y_{u_i, Z}}{\sum_{\substack{2 \text{ nearest} \\ \text{neighbors } u_i \text{ of } A}} \frac{1}{d(A, u_i)}} = \frac{\frac{1}{2.9} 5 + \frac{1}{1} 3}{\frac{1}{2.9} + \frac{1}{1}}$

Collaborative Filtering: Discussion

- K nearest neigbor and similar methods are called *memory-based* approaches.
  - There are no model parameters, no optimization criterion is being optimized.
  - Each prediction reuqires an iteration over all training instances → impractical!

- Better to train a model by minimizing an appropriate loss function over a space of model parameter, then use model to make predictions quickly.

Latent Features

- Idea: Instead of ad-hoc definition of distance between users, learn features that actually represent preferences.
- If, for every user u, we had a feature vector ψ_u that describes their preferences,
- Then we could learn parameters θ_x for item x such that $\theta_x^{\mathrm{T}}\psi_u$ quantifies how much u enjoys x.

Latent Features

- Or, turned around,
  - If, for every item x we had a feature vector ϕ_x that characterizes its properties,
  - We could learn parameters θ_u such that $\theta_u^{\mathrm{T}} \phi_x$ quantifies how much u enjoys x.

- In practice some user attributes ψ_u and item attributes ϕ_x are usually available, but they are insufficient to understand u's preferences and x's relevant properties.

Latent Features

- Idea: construct user attributes ψ_u and item attributes ϕ_x such that ratings in training data can be predicted accurately.
- Decision function:

$$f_{\Psi,\Phi}(u,x) = \psi_u^{\mathrm{T}} \phi_x$$

  - Prediction is product of user preferences and item properties.
- Model parameters:
  - Matrix Ψ of user features ψ_u for all users,
  - Matrix Φ of item features ϕ_x for all items.

Latent Features

- Optimization criterion:

$$(\Psi^*, \Phi^*) = \operatorname{argmin}_{\Psi,\Phi} \sum_{x,u} \ell(y_{u,x}, f_{\Psi,\Phi}(u,x)) + \lambda \left[\sum_u ||\psi_u||^2 + \sum_x ||\phi_x||^2 \right]$$

Feature vectors of all users and all Items are regularized

Latent Features

- Both item and user features are the solution of an optimization problem.
- Number of features k has to be set.
- Meaning of the features is not pre-determined.
- Sometimes they turn out to be interpretable.

Matrix Factorization

- Decision function:

$$f_{\Psi,\Phi}(u,x) = \psi_u^{\mathrm{T}} \phi_x$$

- In matrix notation:

$$\hat{Y}_{\Psi,\Phi} = \Psi \Phi^{\mathrm{T}}$$

- Matrix elements:

$$\begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{1m'} \\ & \ddots & \\ \hat{y}_{m1} & & \hat{y}_{mm'} \end{bmatrix}$$

$$= \begin{bmatrix} \psi_{11} & \cdots & \psi_{1k} \\ & \ddots & \\ \psi_{m1} & & \psi_{mk} \end{bmatrix} \begin{bmatrix} \phi_{11} & \cdots & \phi_{m'1} \\ & \ddots & \\ \phi_{1k} & & \phi_{m'k} \end{bmatrix}$$

Matrix Factorization

- Decision function in matrix notation:

M N T D

$$\begin{bmatrix} \hat{y}_{11} & \cdots & \hat{y}_{1m'} \\ & \ddots & \\ \hat{y}_{m1} & & \hat{y}_{mm'} \end{bmatrix} \begin{matrix} \text{Alice} \\ \text{Bob} \\ \text{Carol} \end{matrix}$$

$$= \begin{bmatrix} \psi_{11} & \cdots & \psi_{1k} \\ & \ddots & \\ \psi_{m1} & & \psi_{mk} \end{bmatrix} \begin{bmatrix} \phi_{11} & \cdots & \phi_{m'1} \\ & \ddots & \\ \phi_{1k} & & \phi_{m'k} \end{bmatrix}$$

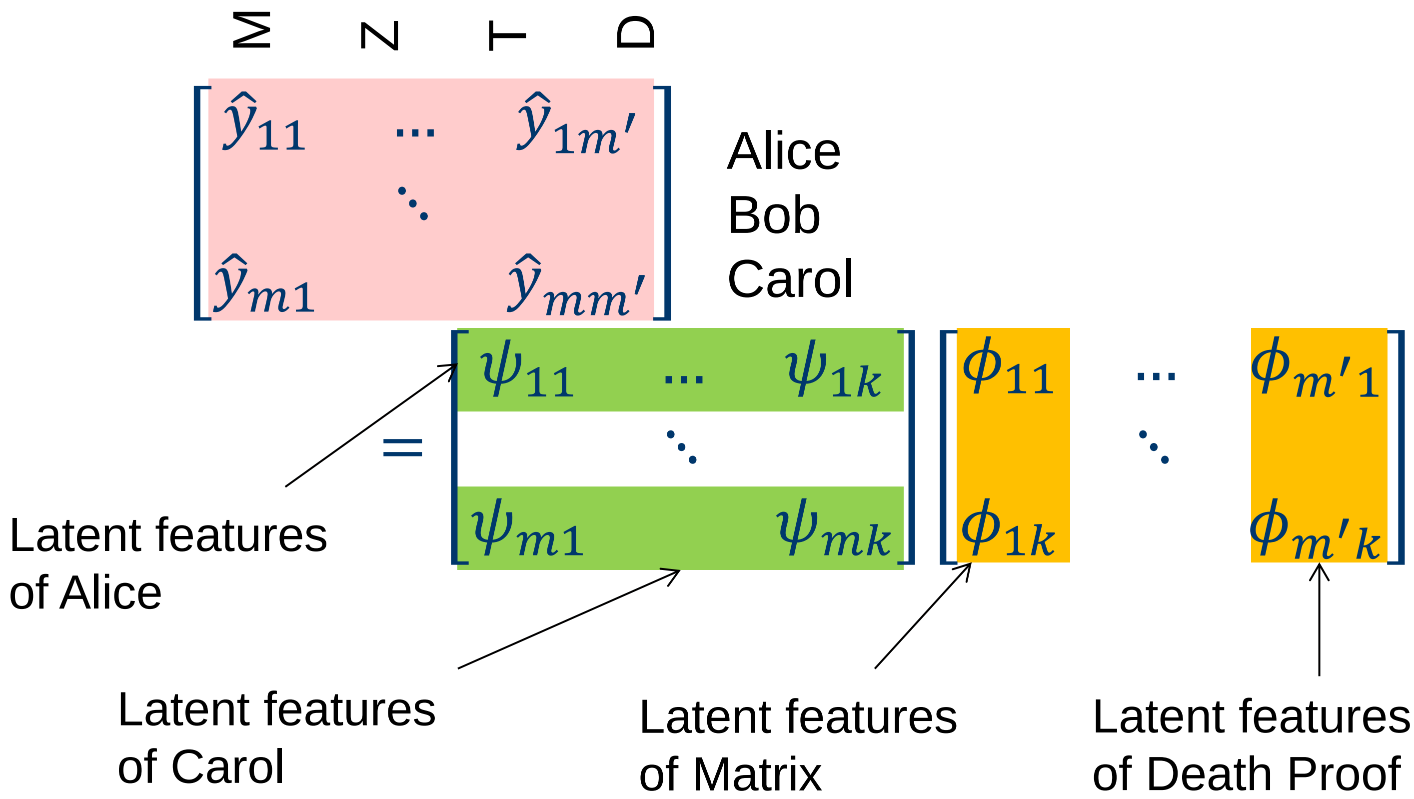
Predicted rating of Matrix for Alice

Latent features of Alice

Latent features of Matrix

Matrix Factorization

- Decision function in matrix notation:



Matrix Factorization

- Optimization criterion:

$$(\Psi^*, \Phi^*) = \mathrm{argmin}_{\Psi,\Phi} \sum_{x,u} \ell(y_{u,x}, f_{\Psi,\Phi}(u,x)) + \lambda\left(||\Psi||^2 + ||\Phi||^2\right)$$

- Criterion is not convex:
  - For instance, multiplying all feature vectors with -1 gives an equally good solution:

$$f_{\Psi,\Phi}(u,x) = \psi_u^{\mathrm{T}}\phi_x = (-\psi_u^{\mathrm{T}})(-\phi_x)$$

- Limiting the number of latent features to k restricts the rank of matrix $\hat{Y}$.

Matrix Factorization

- Optimization criterion:

$$(\Psi^*, \Phi^*) = \operatorname{argmin}_{\Psi,\Phi} \sum_{x,u} \ell(y_{x,u}, f_{\Psi,\Phi}(u,x)) + \lambda\left(||\Psi||^2 + ||\Phi||^2\right)$$

- Optimization by
  - Stochastic gradient descent or
  - Alternating least squares.

Matrix Factorization by Stochastic Gradient Descent

- Iterate through ratings $y_{u,x}$ in training sample
  - Let $\psi'_u \leftarrow \psi_u - \alpha \frac{\partial f_{\Psi,\Phi}(u,x)}{\partial \psi_u}$
  - Let $\phi'_x \leftarrow \phi_x - \alpha \frac{\partial f_{\Psi,\Phi}(u,x)}{\partial \phi_x}$
- Until convergence.

- Requires differentiable loss function; e.g., squared loss, …

Matrix Factorization by Alternating Least Squares

- For squared loss and parallel architectures.
- Alternate between 2 optimization processes:
  - Keep Φ fixed, optimize ψ_u in parallel for all u.
  - Keep Ψ fixed, optimize ϕ_x in parallel for all x.

Matrix Factorization by Alternating Least Squares

- For squared loss and parallel architectures.
- Alternate between 2 optimization processes:
  - Keep Φ fixed, optimize ψ_u in parallel for all u.
  - Keep Ψ fixed, optimize ϕ_x in parallel for all x.
- Optimization criterion for Ψ:

$$\psi_u^* = \operatorname{argmin}_{\psi_u} (y_u - \hat{y}_u)^2 - \lambda \left|\left|\psi_u\right|\right|^2$$
$$= \operatorname{argmin}_{\psi_u} \left(y_u - \psi_u^{\mathrm{T}} \Phi^{\mathrm{T}}\right)^2 - \lambda \left|\left|\psi_u\right|\right|^2$$

$$[\hat{y}_{u1} \quad \cdots \quad \hat{y}_{um'}] = [\psi_{u1} \quad \cdots \quad \psi_{uk}] \begin{bmatrix} \phi_{11} & \cdots & \phi_{m'1} \\ & \ddots & \\ \phi_{1k} & & \phi_{m'k} \end{bmatrix}$$

Matrix Factorization by Alternating Least Squares

- For squared loss and parallel architectures.
- Alternate between 2 optimization processes:
  - Keep Φ fixed, optimize ψ_u in parallel for all u.
  - Keep Ψ fixed, optimize ϕ_x in parallel for all x.
- Optimization criterion for Ψ:

$$\psi_u^* = \text{argmin}_{\psi_u}\left(y_u - \psi_u^{\mathrm{T}}\Phi^{\mathrm{T}}\right)^2 - \lambda\left|\left|\psi_u\right|\right|^2$$

$$\psi_u^* = \left(\Phi\Phi^{\mathrm{T}} + \lambda I\right)^{-1}\Phi y_u$$

Matrix Factorization by Alternating Least Squares

- For squared loss and parallel architectures.
- Alternate between 2 optimization processes:
  - Keep Φ fixed, optimize ψ_u in parallel for all u.
  - Keep Ψ fixed, optimize ϕ_x in parallel for all x.
- Optimization criterion for Ψ:

$$\psi_u^* = \text{argmin}_{\psi_u}\left(y_u - \psi_u^{\mathrm{T}}\Phi^{\mathrm{T}}\right)^2 - \lambda\left|\left|\psi_u\right|\right|^2$$

$$\psi_u^* = \left(\Phi\Phi^{\mathrm{T}} + \lambda I\right)^{-1}\Phi y_u$$

- Optimization criterion for Φ:

$$\phi_x^* = \text{argmin}_{\phi_x}\left(y_x - \phi_x^{\mathrm{T}}\Psi^{\mathrm{T}}\right)^2 - \lambda\left|\left|\phi_x\right|\right|^2$$

$$\phi_x^* = \left(\Psi\Psi^{\mathrm{T}} + \lambda I\right)^{-1}\Psi y_x$$

Matrix Factorization by Alternating Least Squares

- For squared loss and parallel architectures.
- Initialize $\Psi,\ \Phi$ randomly.
- Repeat until convergence:
  - Keep Ψ fixed, for all u in parallel:
    - $\psi_u = \left(\Phi\Phi^{\mathrm{T}} + \lambda I\right)^{-1}\Phi y_u$
  - Keep Φ fixed, for all x in parallel:
    - $\phi_u = \left(\Psi\Psi^{\mathrm{T}} + \lambda I\right)^{-1}\Psi y_x$

Extensions: Biases

- Some users just give optimistic or pessimistic ratings; some items are hyped. Decision function:
$$f_{\Psi,\Phi,B_u,B_x}(u,x) = b_u + b_x + \psi_u^{\mathrm{T}} \phi_x$$
- Optimization criterion:
$$(\Psi^*, \Phi^*, B_u, B_x)$$
$$= \mathrm{argmin}_{\Psi,\Phi} \sum_{x,u} \ell(y_{x,u}, f_{\Psi,\Phi,B_u,B_x}(u,x))$$
$$+ \lambda \left(\left|\left|\Psi\right|\right|^2 + \left|\left|\Phi\right|\right|^2 + \left|\left|B_u\right|\right|^2 + \left|\left|B_x\right|\right|^2 \right)$$

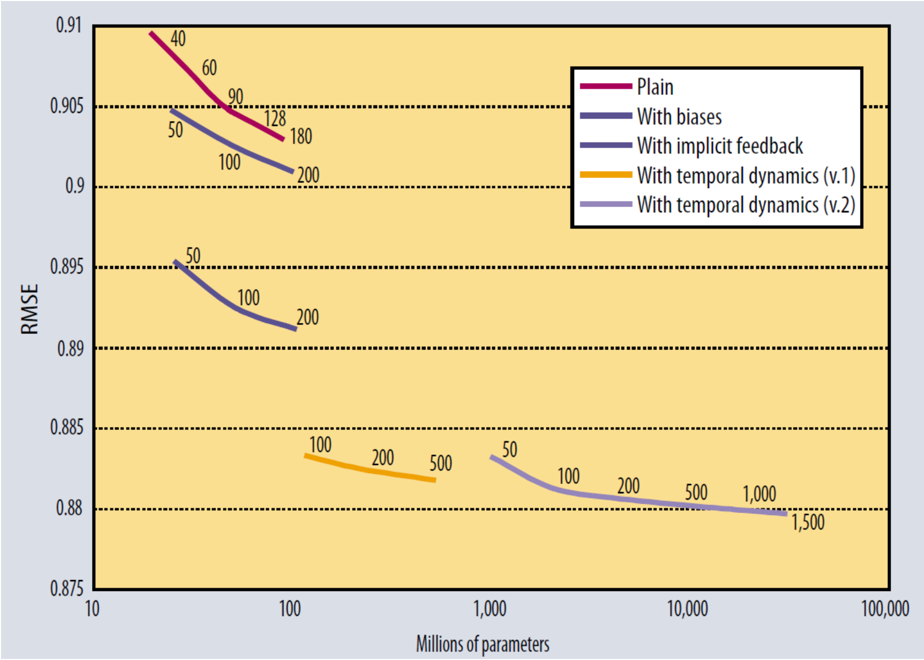
Extensions: Explicit Features

- Often, explicit user and item features are available.
- Concatenate vectors ψ_u and ϕ_x; explicit features are fixed, latent features are free paremeters.

Extensions: Temporal Dynamics

- How much a user likes an item depends on the point in time when the rating takes place.

$$f_{\Psi,\Phi,B_u,B_x,t}(u,x) = b_u(t) + b_x(t) + \psi_u(t)^{\mathrm{T}}\phi_x$$



Summary

- Purely content-based recommendation: users don‘t benefit from other users‘ ratings.
- Collaborative filtering by nearest neighbors: fixed definition of similarity of users. No model parameters, no learning. Has to iterate over data to make recommendation.
- Latent factor models, matrix factorization: user preferences and item properties are free parameters, optimized to minimized discrepancy between inferred and actual ratings.