

Recommender Systems 1

Internet Analytics (COM-308)

Prof. Matthias Grossglauser
School of Computer and Communication
Sciences

EPFL

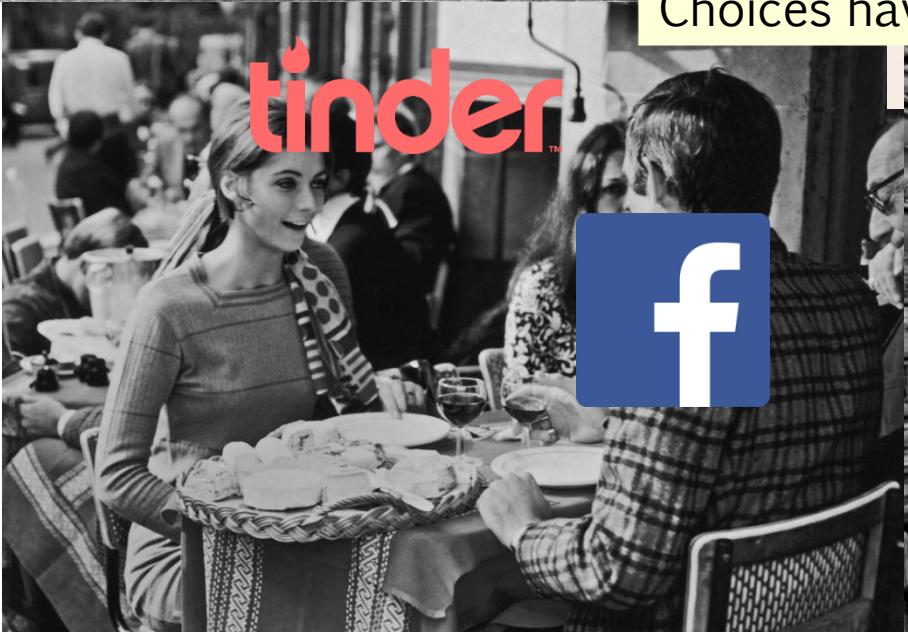
Overview

- Motivation: why are recommenders so prevalent today?
- Collaborative filtering vs content-based recommenders
- Example: Netflix Prize
- Neighborhood methods
- Latent factor methods
 - Overfitting, regularization, stochastic gradient descent

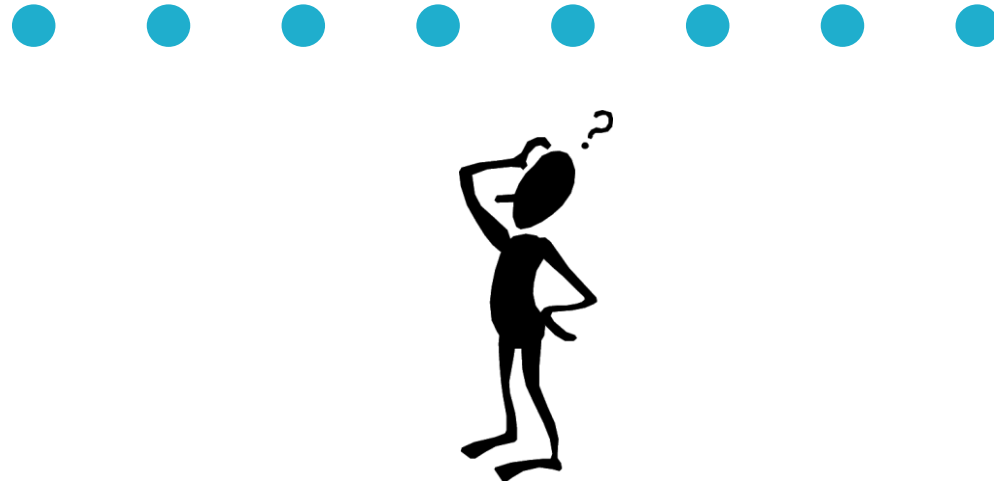
Choices: the good old days → the brave new world



Choices have exploded!

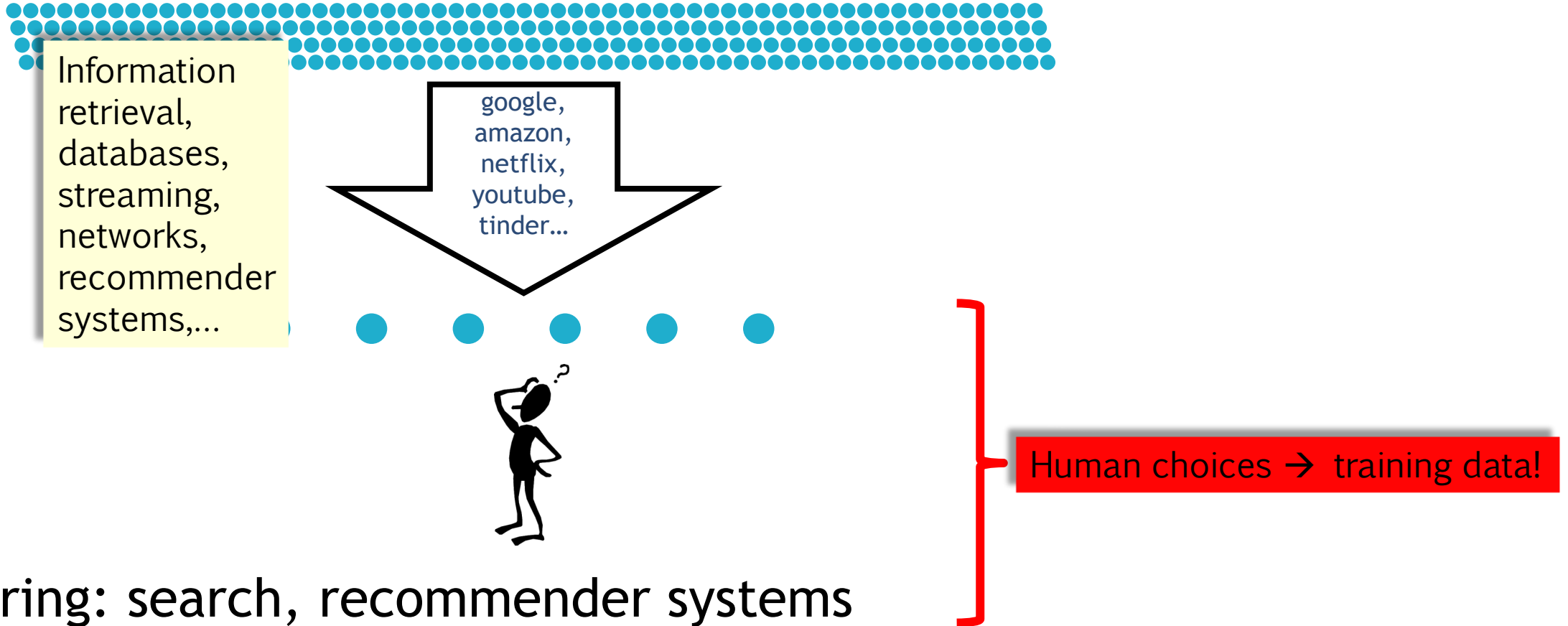


Choosing in the old days



- Small number of alternatives
 - Physical limits: shelf space, weight of the encyclopedia, surface area of the dance floor, cost of the record collection,...

Filtering + choosing



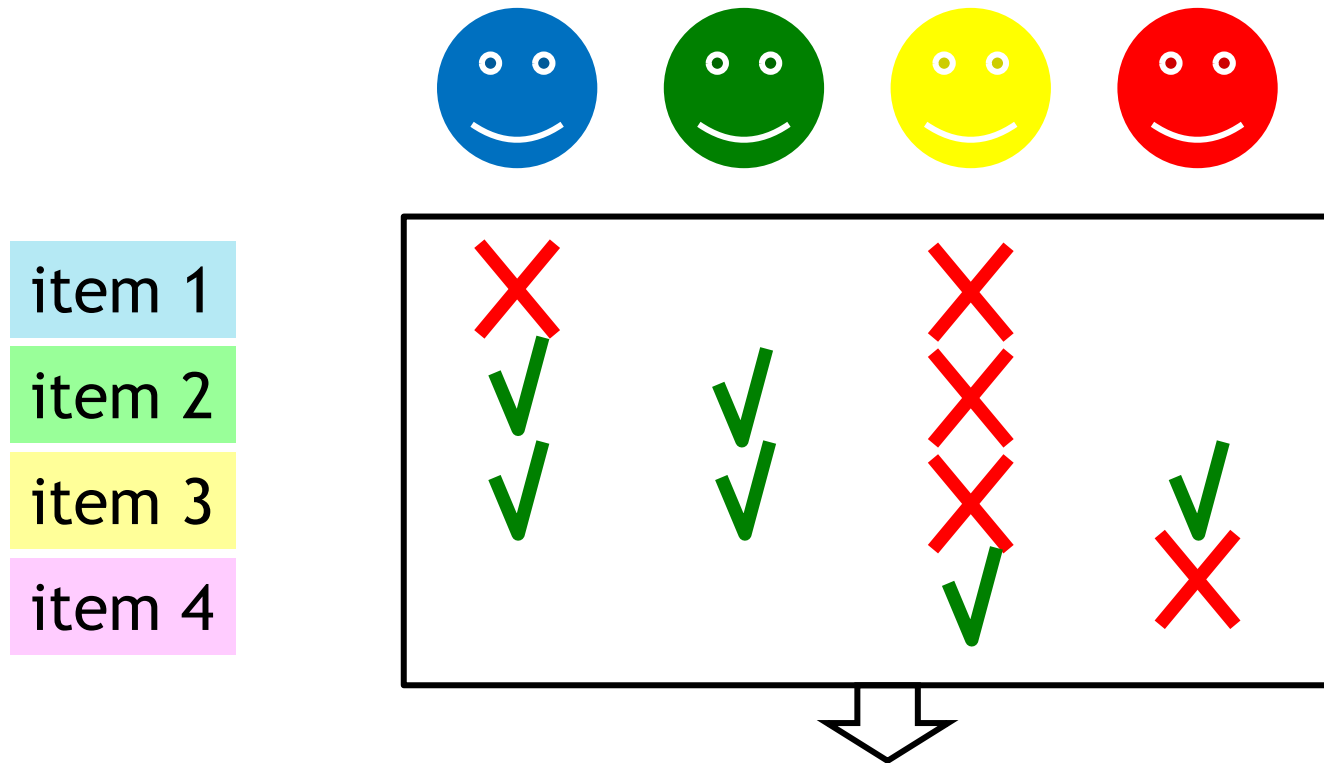
- Filtering: search, recommender systems
- Choosing: the “last cognitive mile”

Situation today

- Traditional retailers:
 - Shelf space & warehouse: expensive → carry only items with sufficient sales volume
- Online:
 - Potentially unlimited catalogue for digital goods (or physical goods - amazon, iTunes, ...)
 - Needs better filters: search & recommendations
- Recommenders are integral part of most online services
 - Amazon, Youtube, Spotify, LinkedIn, Twitter, ...
 - Even search: google → “filter bubble”
- Limited user interface (mobile!):
 - Recommendations even more important than search

Collaborative filtering

- Content-agnostic \rightarrow learning from other users



Model for (user, item) \rightarrow rating

Overview: recommender systems

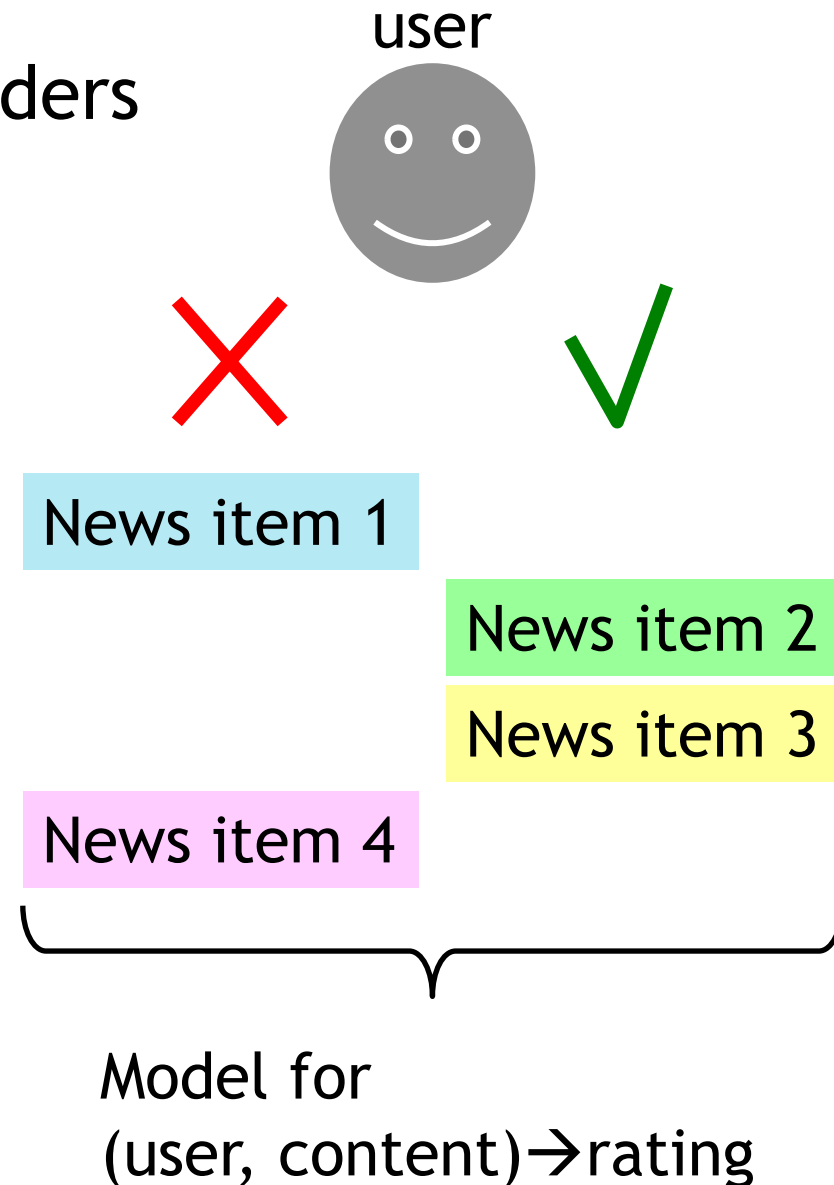
- Content-based recommenders

item 1:
“Plane hijacked...”

item 2:
“soccer game...”

item 3:
“swiss skiers win...”

item 4:
“50.3% vote yes...”



The Netflix competition

- Netflix: mail-based DVD rental company (today streaming)
- New form of research outsourcing: Netflix Prize
 - Goal: “increase performance of in-house system by 10%”
 - Prize: 1m USD + yearly progress prize
 - Anyone can participate
 - Careful setup to avoid reverse-engineering of dataset, overfitting, etc.
- Early example of data-driven competitions
 - Kaggle etc.



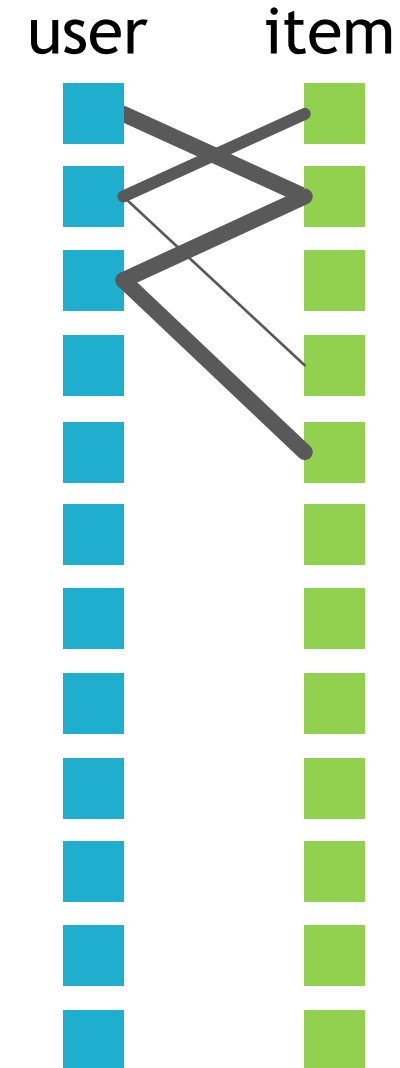
Model

- Set of users U (size n), set of items I (size m)
- Utility or rating function $r: U \times I \rightarrow R$
 - R : e.g. 0-5 stars; probability of liking an item; yes/no;...
- Collecting r_{ui} values:
 - Amazon: buying a product
 - Youtube: watching/liking a video
 - News reader: opening a news item from list
 - In general: depends on context and design
- Explicit vs implicit:
 - Explicit: Ask people to rate (stars, etc.)
 - effort, sparse, reliable
 - Implicit: derive from actions (delete, save, forward, etc.)
 - for free, dense, noisy

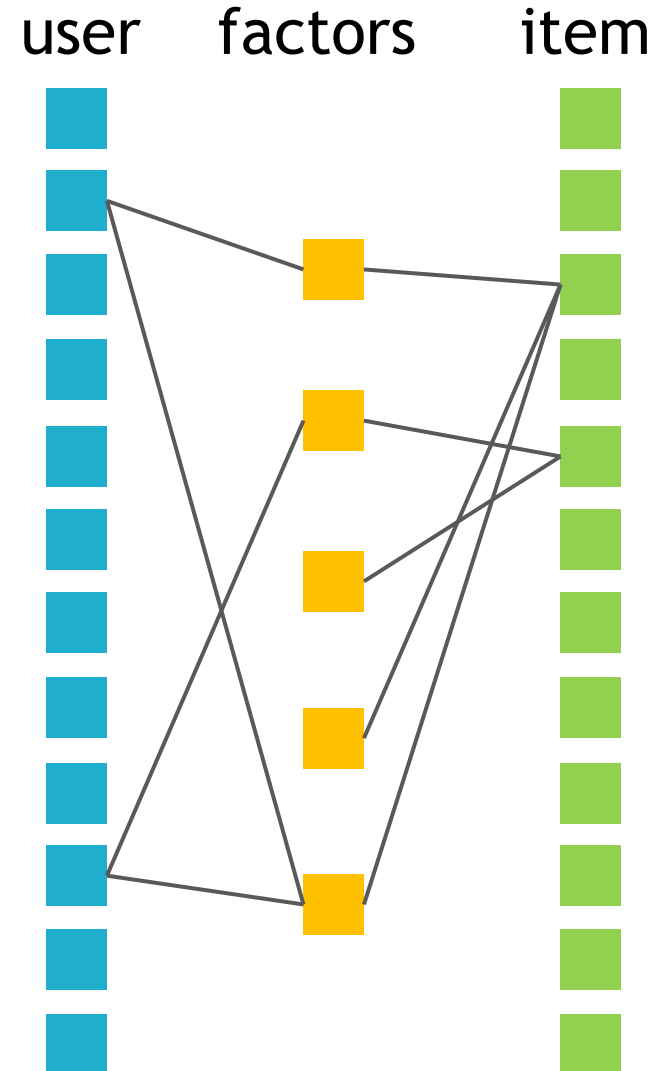
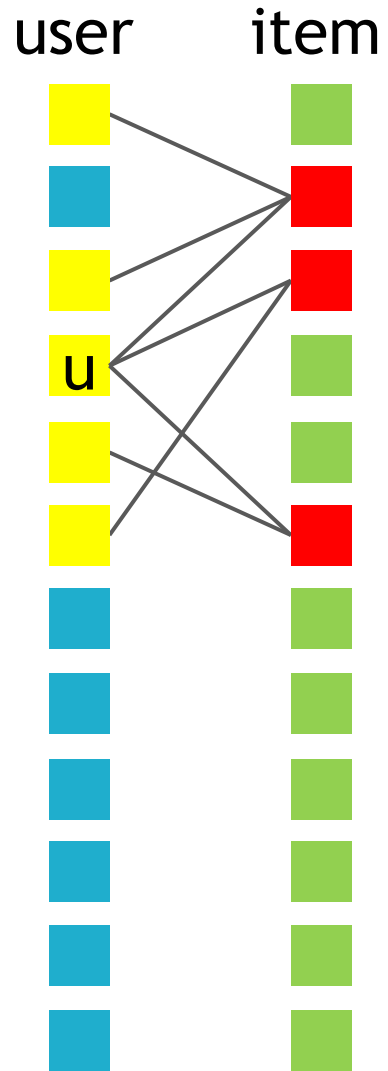
Model

- Ratings matrix R :
 - Captures preference on some scale
 - Matrix representation:
 - Missing elements = unknown
 - Bipartite weighted graph representation

	1	2	3	4	5	6	7	8
1		5		2	4			
2	4		3	1			3	
3		5	4		5		4	
4						1	1	2
5	3					3		
6			2		4			



Neighborhood vs latent factor methods



Performance criterion for Netflix Prize

- RMSE: root mean squared error:

- $$RMSE = \sqrt{\sum_{(u,i)} \frac{(r_{ui} - \hat{r}_{ui})^2}{C}},$$

with $C = \#$ of rated pairs

- Why RMS?

- Penalizes larger errors

- Why not RMS?

- Often only interested in precision on top ratings \rightarrow cost function that penalizes only error in high r_{ui}

- Often not interested in absolute value, only order \rightarrow cost function that depends only on order of r_{ui} vs order of \hat{r}_{ui}

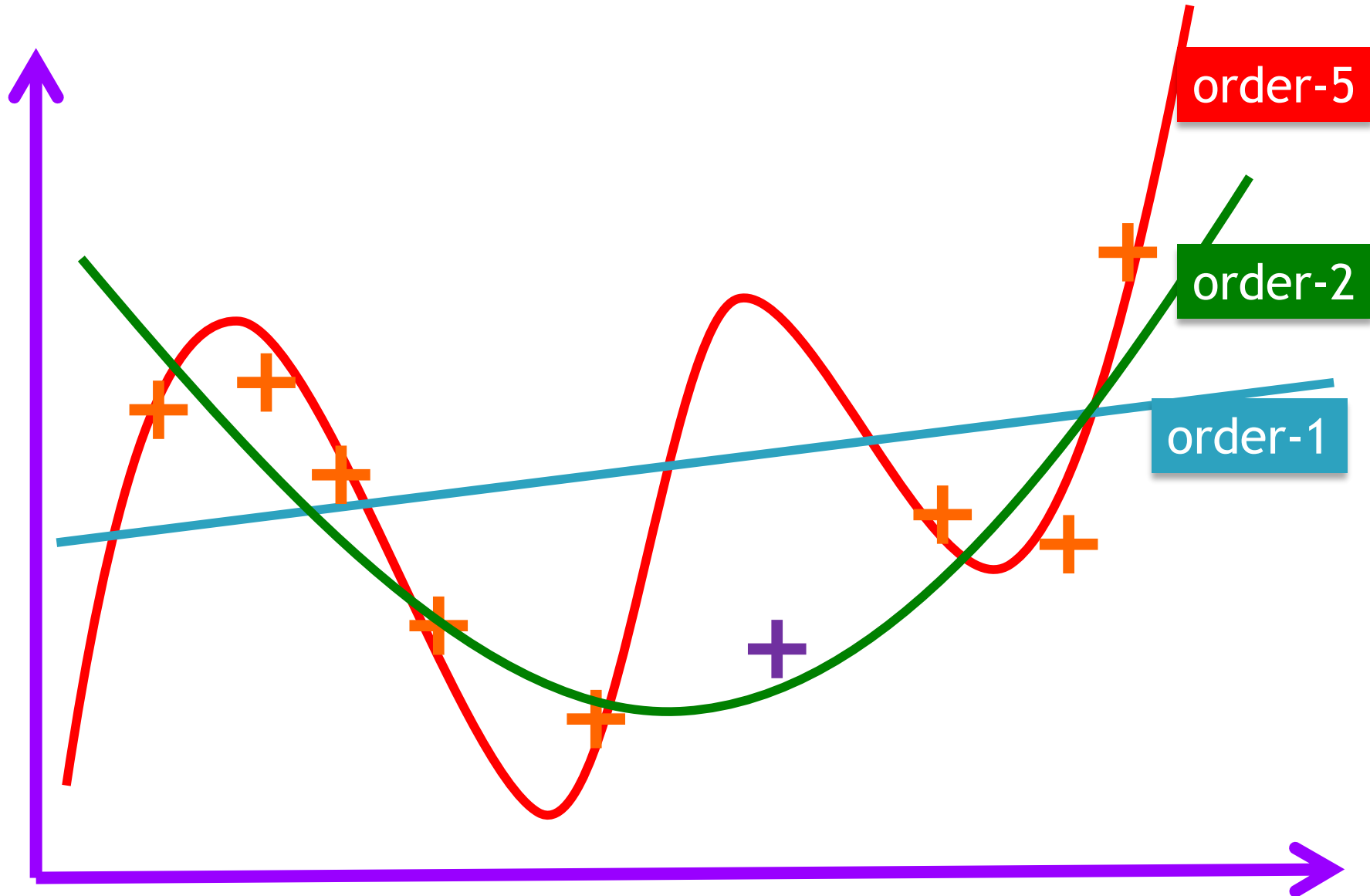
Baseline predictor

- Assumption:
 - Global average rating \bar{r}
 - Each user u has a bias (or average opinion) b_u
 - Each item i has a bias (or average quality) b_i
- First approximation:
 - $b_{ui} = \bar{r} + b_u + b_i$
 - No “interaction” between users and items
- Learning: data \rightarrow model parameters?
 - $n + m$ parameters
 - Data: up to nm
 - In general overdetermined \rightarrow find best solution

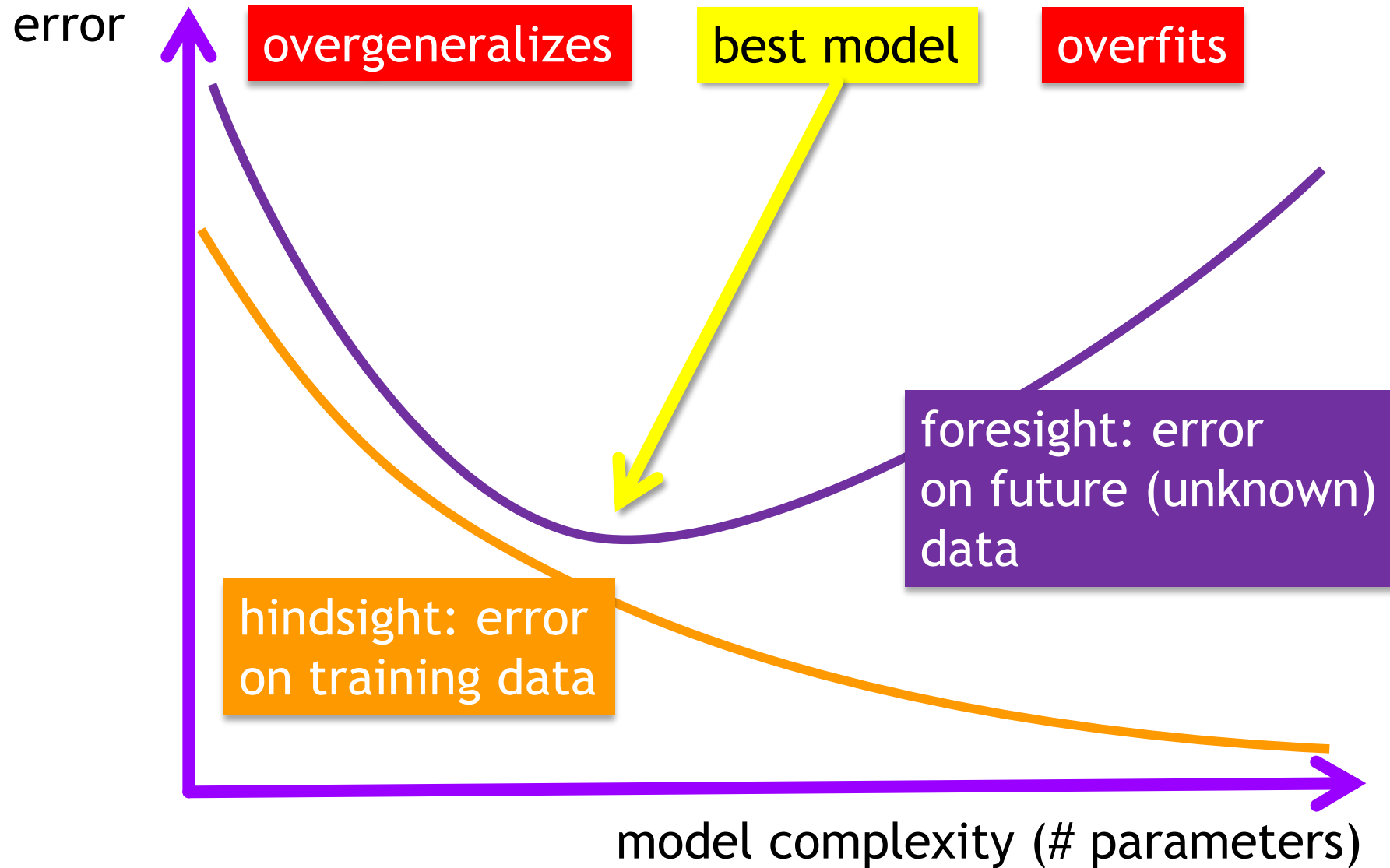
Learning baseline predictor

- Given: training set of ratings $R = \{(u, i, r_{ui})\}$
- Could just use averages per user/item
 - Not optimal
- Min RMS on training set:
 - $\min_{\{b_u, b_i\}} \sum_{(u,i) \in R} (r_{ui} - b_{ui})^2$
- General form of quadratic min problem:
$$\|Ab - c\|_2^2 = (Ab - c)^T (Ab - c) =$$
$$= b^T A^T A b - 2b^T A^T c + c^T c$$
- Derivative w.r.t. b
 - $2(A^T A)b - 2A^T c = 0 \rightarrow$ find b
- This may lead to overfitting \rightarrow regularization

Learning: overfitting



Hindsight vs Foresight

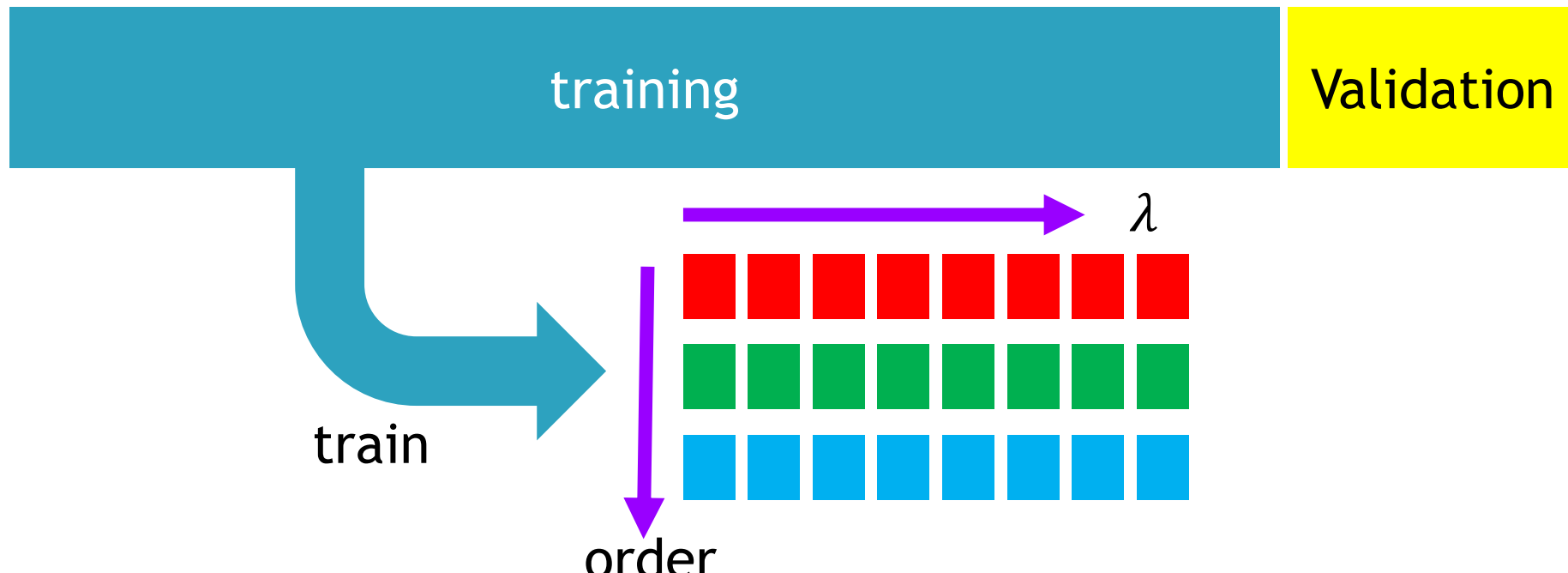


Regularization: penalizing complexity

- Loss function to minimize:
 - $\min_{\theta} f_{\theta}(X)$: X is the data (here $\{r_{ui}\}$), θ the model parameters ($\{b_u, b_i\}$)
- Penalize complexity:
 - Overfitting: complex model does well on training set, but poorly on future data (or test set)
 - We want to use complex models (e.g., higher order polynomial), but avoid overfitting
 - Solution: build in preference for “small” parameters
- Regularizer:
 - $\min_{\theta} f_{\theta}(X) + \lambda g(\theta)$
 - Choice of $g(\theta) \geq 0$ depends on context
 - Example: $g(\theta) = \|\theta\|_2^2$

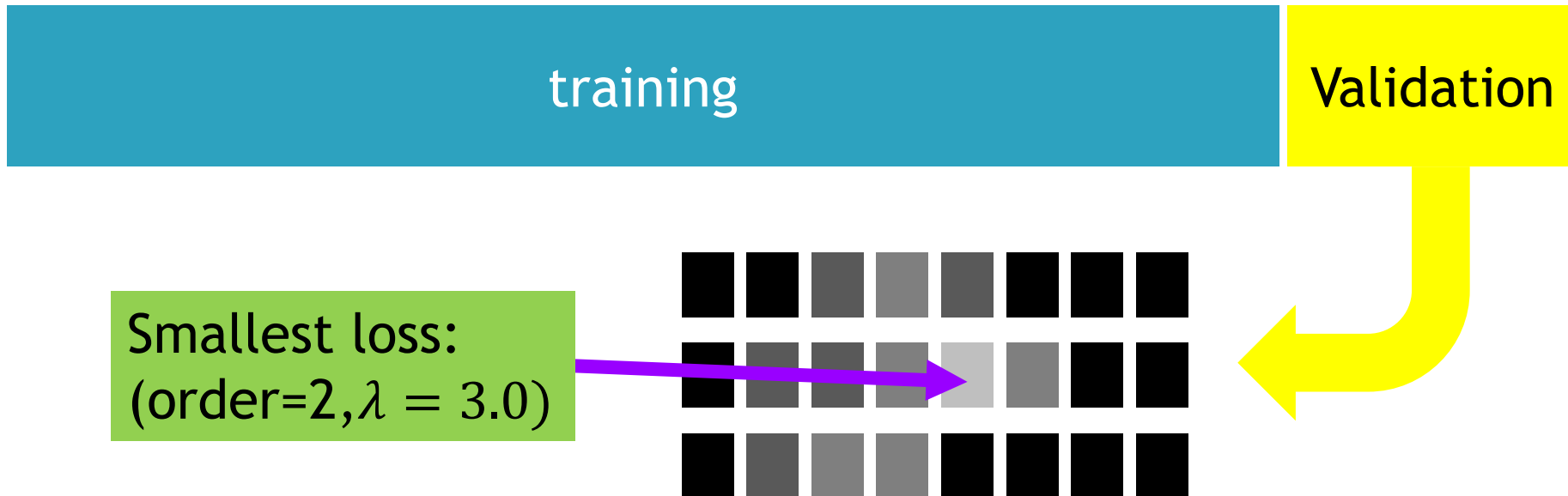
Validation: simulating foresight

- Model selection:
 - Degree of polynomial
 - Regularization (hyper)parameter λ
- We do not have future data \rightarrow set aside some training data and pretend it is future data
 - Validation set



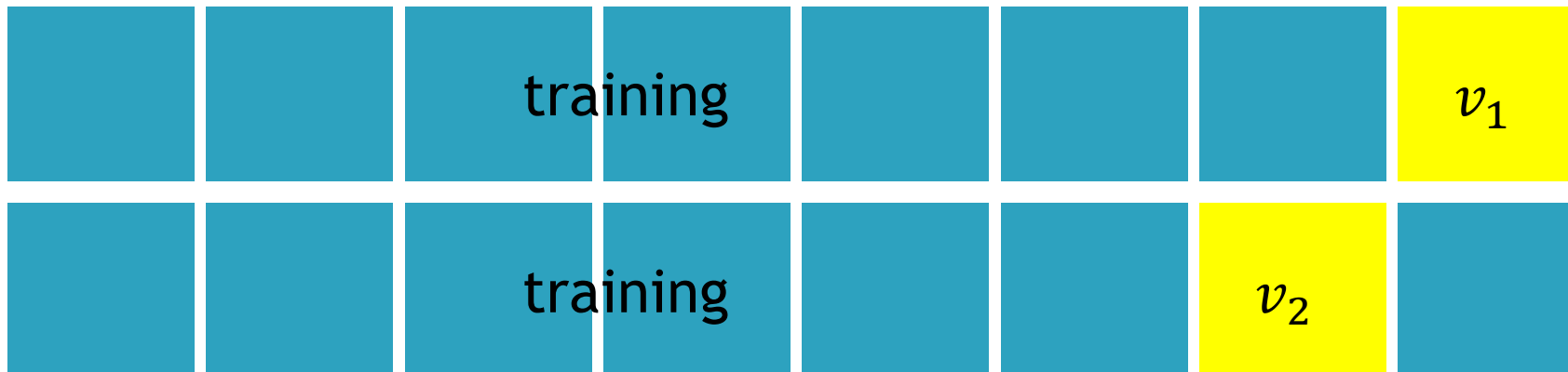
Validation: simulating foresight

- Model selection:
 - Degree of polynomial
 - Regularization (hyper)parameter λ
- We do not have future data \rightarrow set aside some training data and pretend it is future data
 - Validation set



Cross-validation: averaged validation

- If data is not abundant: validation is costly
 - Tradeoff between training and validation data
- k -fold CV:
 - Chop data into equal sized blocks (e.g., $k = 10$)
- For each block $x = 1, \dots, k$:
 - Train model on all other blocks (training set)
 - Evaluate model on v_x (validation set)
- Performance = average error over all iterations



Regularized bias estimates

- For our specific problem so far (finding user and item bias terms from ratings): quadratic form to minimize to obtain the bias terms $\{b_u\}$ and $\{b_i\}$:

$$\begin{aligned} (b_u^*, b_i^*) &= \\ &= \operatorname{argmin}_{\{b_u, b_i\}} \sum_{(u,i) \in R} (r_{ui} - \bar{r} - b_u - b_i)^2 + \lambda_1 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) \end{aligned}$$

Residual error after baseline predictor

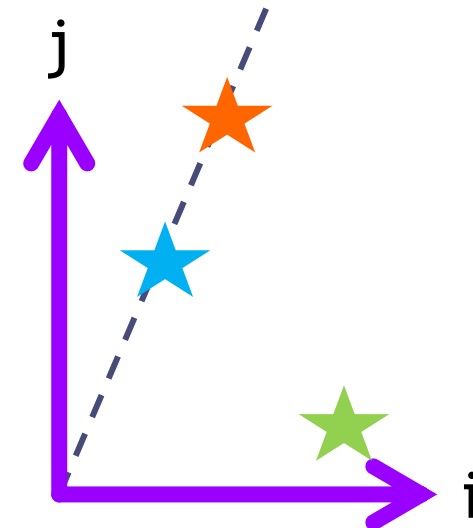
- Residual error:
 - Captures dependence between user and item
 - $\tilde{r}_{ui} = r_{ui} - b_{ui} = r_{ui} - (\bar{r} + b_u + b_i)$
 - $\tilde{R} = [\tilde{r}_{ui}]$
- How to capture residual error, i.e., how to model user-item interaction?

Neighborhood models

- Goal: estimate rating r_{ui} for user u and item i
- Approach: pairwise user-user or item-item correlation
- User-user:
 - For user u , find other users $\{v\}$ that are similar to u
 - Combine $\{r_{vi}\}$ into an estimate for r_{ui}
- Item-item:
 - For item i , find other items $\{j\}$ that are similar to i
 - Combine $\{r_{uj}\}$ into an estimate for r_{ui}

Similarity metric (user-user variant)

- Similarity between users u and v ?
- Degree of agreement in ratings for joint items
- Cosine similarity:
 - Def: x_u, x_v are vectors of ratings \tilde{r}_u, \tilde{r}_v over items rated by both
 - $$\text{sim}(u, v) = \frac{x_u^T x_v}{\|x_u\|_2 \|x_v\|_2} = \cos(\angle x_u, x_v)$$
- For user u , evaluate $\text{sim}(u, v)$ over all other users v ;
retain L highest (top- L) \rightarrow set L_u



Neighborhood model

- Combine “opinions” of all the similar users L_u :

- $$\hat{r}_{ui} = \bar{r} + b_u + b_i + \frac{\sum_{v \in L_u} \text{sim}(u,v) \tilde{r}_{vi}}{\sum_{v \in L_u} \text{sim}(u,v)}$$

- Pros:

- Intuitively appealing and natural

- Cons:

- Hard to tune: choice of similarity metric $\text{sim}(\cdot, \cdot)$, L (and other parameters depending on variant)
- Sparsity: for many (u, i) pairs, no rating possible (if nobody in set L_u has rated i)

Neighborhood model: user-user vs item-item

- Two symmetric approaches:
 - User-user: for user u , find similar users v , and their ratings of i
 - Item-item: for item i , find similar items j , and their ratings by u
- In practice they are different:
 - Argument for item-item: in most applications, items tend to “cluster” better (e.g., movies belong to a single genre); users are more “mixed” (e.g., one user may like many genres)
 - Argument for user-user: often the application is not “estimate r_{ui} ”, but “get highest rated items for u ”

Recommending best item i to a user u

- With user-user:
 - First find users v having rated same movie(s)
 - Estimate all unrated items for u from these users
 - Equal to two-hop neighborhood in bipartite rating graph

	1	2	3	4	5	6	7	8
1		5		2	2			
2	4		3	1	1		4	
u	???	5	4	???	1	???	4	???
4						1	1	2
5	3		?		?	3		
6		?	2		4		?	

Recommending best item i to a user u

- With item-item:
 - For each item i not rated by u , find similar items L_i
 - Obtain \hat{r}_{ui} from user u 's ratings for $j \in L_i$
 - Much more costly: many unrated items $j \in L_i$

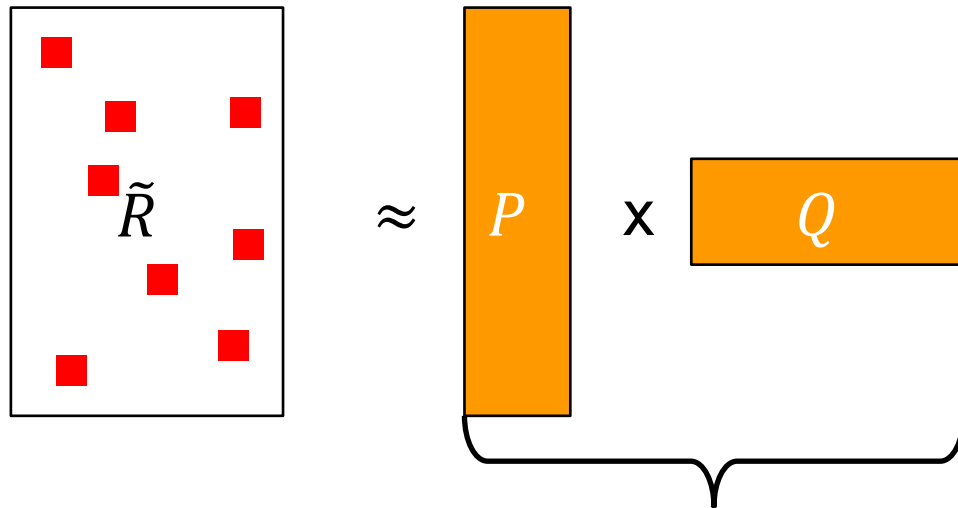
	1	2	3	4	5	6	7	8
1		5		2	2			
2	4		3	1	1		4	
u	???	5	4	???	1	???	4	???
4						1	1	2
5	3		5		?	3	4	
6			2		4			

Latent factor models

- Dimensionality-reduction technique
 - Hypothesis: simpler (lower-dim) space capturing user-item dependencies
- Assume K concepts/latent factors/taste dimensions
 - Movies: Comedy vs drama; historic vs sci-fi; intellectual vs entertainment; romantic vs action; specific cast, directors; etc.
- Each user u has a K -dim factor vector p_u :
 - $p_u[k]$: degree to which user u enjoys/hates factor k
- Each item i has a K -dim factor vector q_i :
 - $q_i[k]$: degree to which item i possesses factor k

Latent factor models

- $\hat{r}_{ui} = \bar{r} + b_u + b_i + p_u^T q_i$
- Note:
 - Contrary to SVD, no requirement that P, Q be orthogonal (columns of P and rows of Q not unitary nor orthogonal)
- Training the model: $\min_{P, Q} \sum_{(u,i) \in R} (\tilde{r}_{ui} - p_u^T q_i)^2$



1) What regularizer?

2) How to find P, Q ?

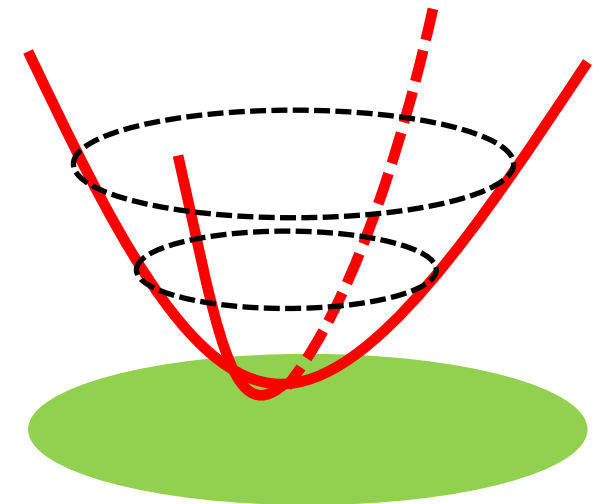
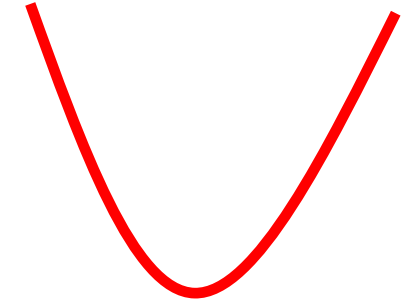
parameters $\theta = (P, Q)$: $d = K(n + m)$ degrees of freedom

Regularized latent factor model

- Cost function with regularizer: $(b_u^*, b_i^*, P^*, Q^*) =$
 - $= \operatorname{argmin}_{\{b_u, b_i, P, Q\}} \sum_{(u,i) \in R} (r_{ui} - (p_u^T q_i + \bar{r} + b_u + b_i))^2$
 $+ \lambda_1 (\|b_u\|^2 + \|b_i\|^2)$
 $+ \lambda_2 (\|P\|_F^2 + \|Q\|_F^2)$
 - Three hyperparameters $(K, \lambda_1, \lambda_2)$ to optimize
 - Not actually a convex problem!
 - But convex (quadratic) over P, Q individually \rightarrow biconvex
 - (P^*, Q^*) is only a local, not necessarily global, minimum

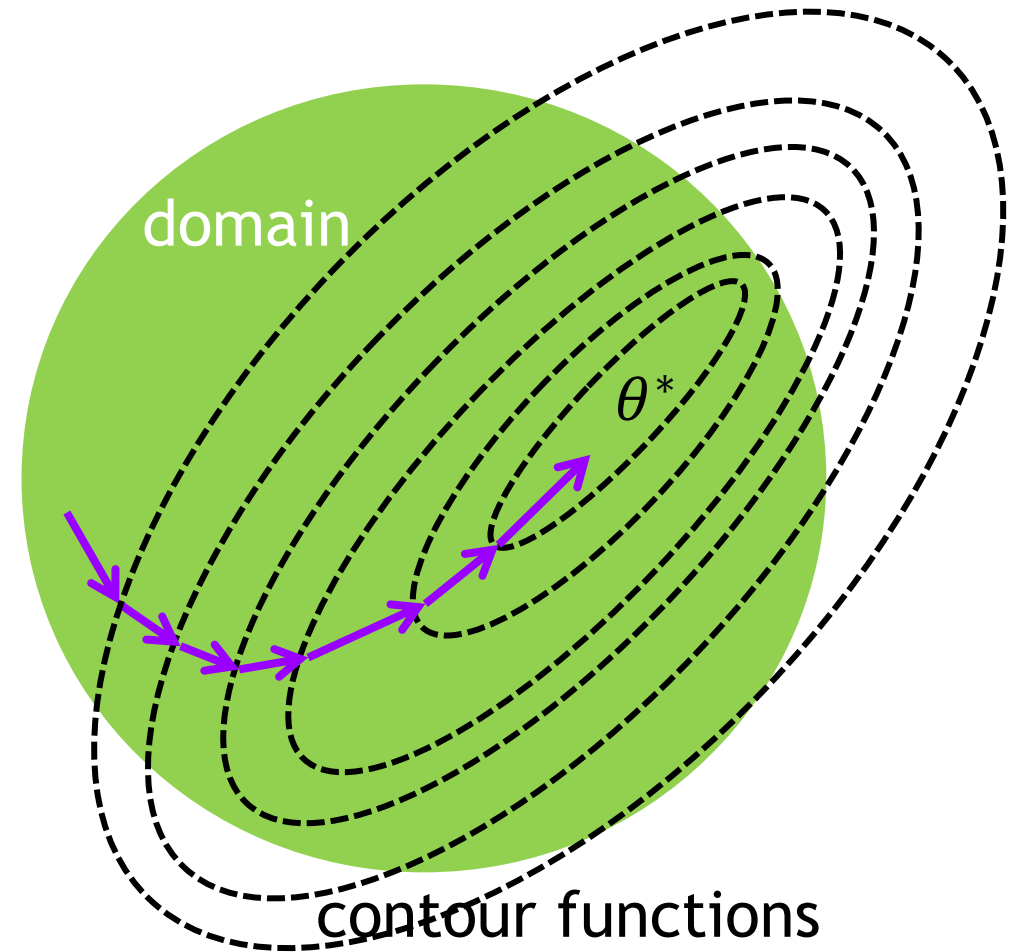
Convex optimization

- Convex function $f(\theta)$, $\theta = (\theta_1, \dots, \theta_d)$
 - For every two points θ, ϕ , the line between $f(\theta)$ and $f(\phi)$ is «above» the function
- Convex domain:
 - For every two points θ, ϕ in the domain, the line connecting them also in domain
- Test for convexity of differentiable function:
 - Hessian $(\nabla^2 f)_{ij} = \frac{\partial^2 f}{\partial \theta_i \partial \theta_j}$
must be PSD (positive semidefinite)
- Convex optimization:
 - Local min = global min \rightarrow we can use methods for local min search



Minimization: stochastic gradient descent

- Gradient $(\nabla f)_i = \frac{\partial f}{\partial \theta_i}$
- Gradient descent:
 - $\theta^{(k+1)} = \theta^{(k)} - \alpha \nabla f(\theta^{(k)})$
 - α : learning rate
 - Intuition: move in direction of local decrease
- In ML, f is often a sum over the data:
 - $f = \sum f_n$
 - Here: $f = \sum_{(u,i) \in R} (\tilde{r}_{ui} - p_u^T q_i)^2$
 - Log-likelihood: $f = \log P(Y|\theta) = \sum \log P(Y_n|\theta)$
 - Gradient costly to compute! ($O(n)$ per step)

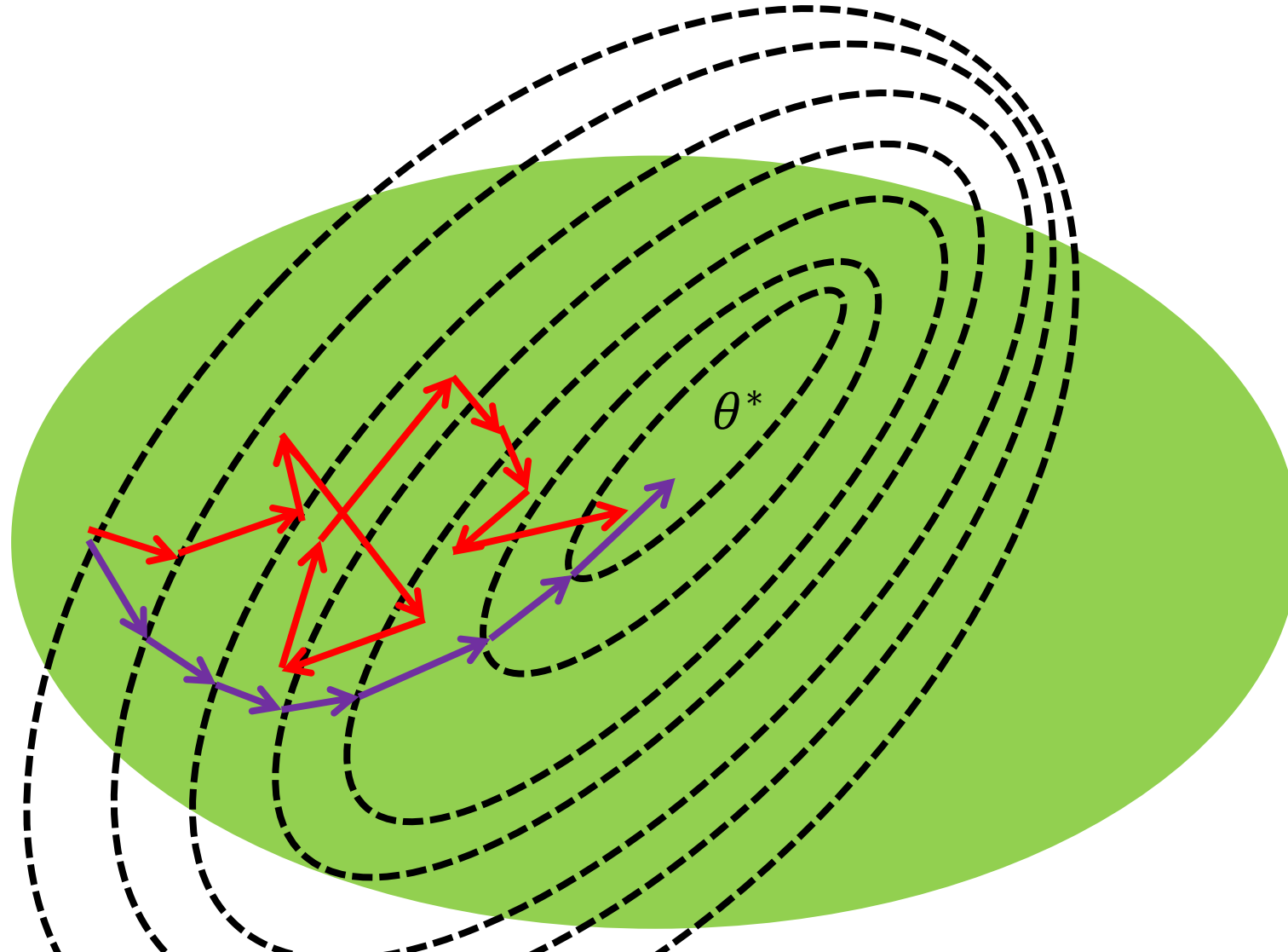


Stochastic gradient descent

- Gradient descent: expensive
 - Sum over all data points $(u, i) \in R$
- Stochastic gradient descent:
 - Idea: noisy but cheap gradient approximation
 - Pick a random data point (u, i) (or some other order)
 - Compute gradients w.r.t. this data point (or a small batch of data points)
 - Iterate until convergence
- Intuition:
 - Random walk that is biased towards minimum
 - Pro: gradient much cheaper to compute
 - Con: random walk may “veer” in the wrong direction
 - Worth it if “detours” do not outweigh reduction in computational cost

Stochastic gradient descent

- Gradient descent vs stochastic gradient descent



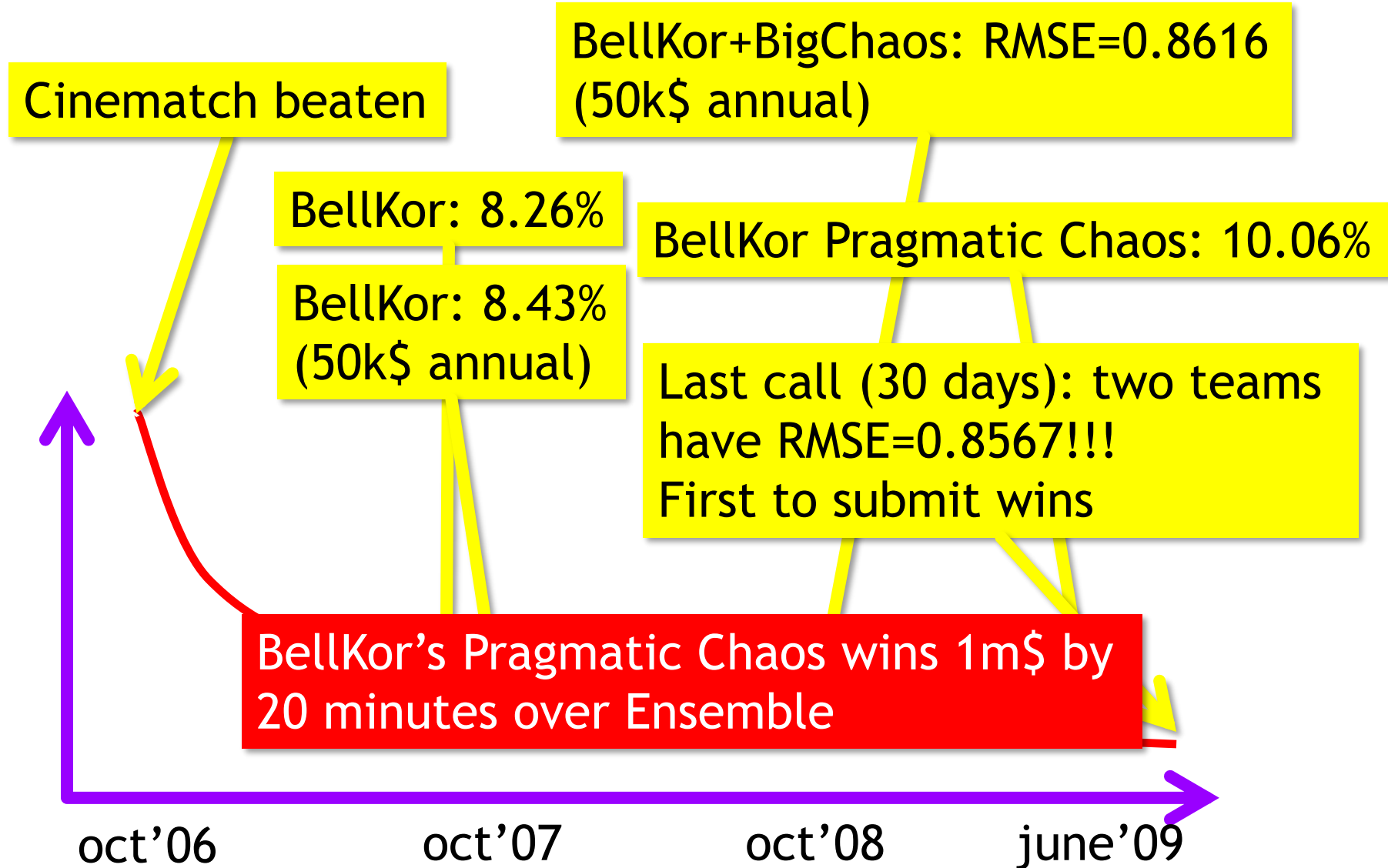
Specifically for matrix factorization: ALS vs SGD

- Two solutions to deal with non-convexity:
 - Optimize using SGD anyway → no guarantee of finding global minimum, but in practice it usually finds a good solution
 - (some convexity guarantees exist under slightly stronger assumptions)
 - Alternating Least Squares (ALS) for biconvex problems:
 - Fix P , minimize in Q : quadratic, explicit solution
 - Fix Q , minimize in P : quadratic, explicit solution
 - Repeat until convergence
- From studies with real data: SGD usually wins in terms of computational cost for a good solution, except for very sparse datasets

Netflix Prize: outcomes and stats

- Data set:
 - ~500k users, ~18k movies
 - 100m ratings over 5 years
- Recommender system for movies: Cinematch
 - RMSE = 0.9514
 - One week until Cinematch got outperformed!
- Stats:
 - 5000 teams (200 USD/team)
 - 44000 submissions
- Netflix required for all results to be published

Netflix Prize



Summary & lessons

- Advantages of collaborative filtering (CF):
 - Content-independent: works for any type of item
 - Big data: exploits large user population
- Collaborative Filtering drawbacks:
 - Cold start (new user and new item)
 - Sparsity: most user-item pairs never observed → matrix factorization produces an estimate for every (u, i) !
- Model extensions, additional inputs:
 - Context: location, time, mood, etc.
 - Temporal factors: e.g., age of a movie since release - critical in netflix challenge
- Next lecture: using content for recommendation

Time is money! 😊
(1'000'000\$/20 minutes)

References

- [M. Chiang: Networked Life (chapter 4), 2012]
- [A. Rajaraman, J. D. Ullman: Mining of Massive Datasets (chapter 9), 2012]
- [S. Shalev-Shwartz, S. Ben-David: Understanding Machine Learning: From Theory to Algorithms, 2014]
- [Y. Koren, R. Bell, Ch. Volinsky: Matrix Factorization Techniques for Recommender Systems, IEEE Computer, Aug 2009]
- [E. Pariser: The Filter Bubble: What the Internet is hiding from you, Penguin 2011]