# CSE 258 – Lecture 17

## Web Mining and Recommender Systems

## Temporal data mining

# This week

## Temporal models

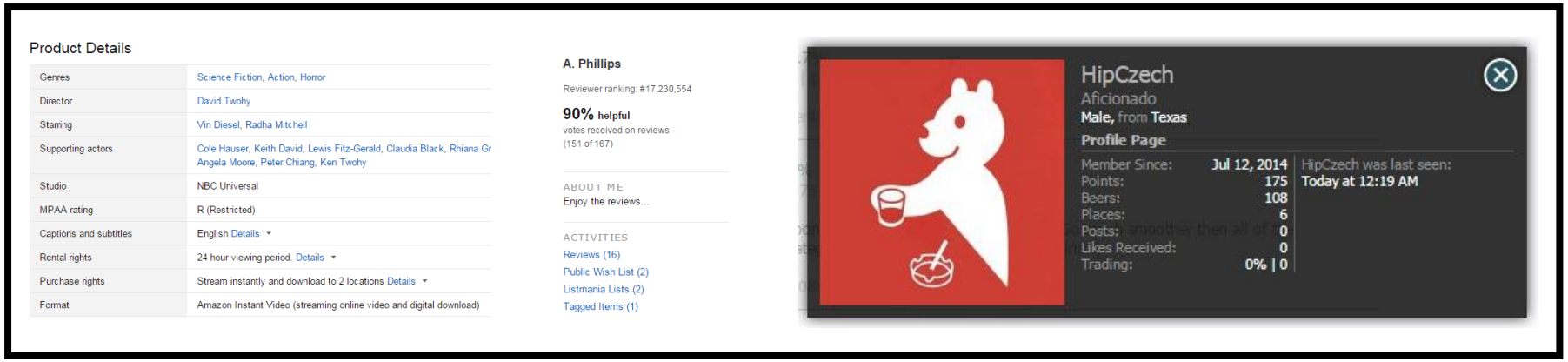
This week we'll look back on some of the topics already covered in this class, and see how they can be adapted to make use of **temporal** information

1. **Regression** – sliding windows and autoregression
2. **Classification** – dynamic time-warping
3. **Dimensionality reduction** - ?
4. **Recommender systems** – some results from Koren

Next lecture:

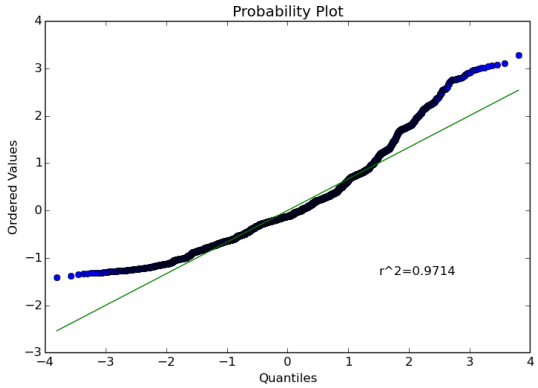
1. **Text mining** – "Topics over Time"
2. **Social networks** – densification over time

# 1. Regression




How can we use **features** such as product properties and user demographics to make predictions about **real-valued** outcomes (e.g. star ratings)?

How can we prevent our models from **overfitting** by favouring simpler models over more complex ones?



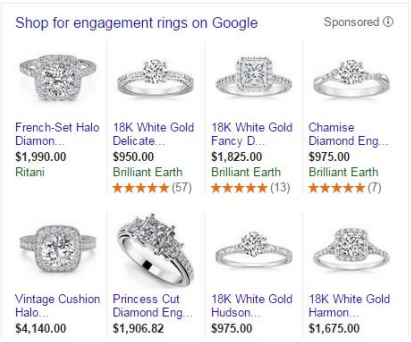

How can we assess our decision to optimize a particular error measure, like the MSE?

# 2. Classification

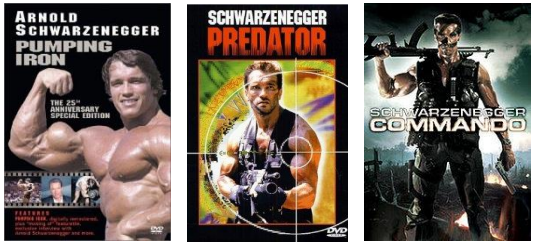
Next we adapted these ideas to **binary** or **multiclass** outputs

What animal is in this image?

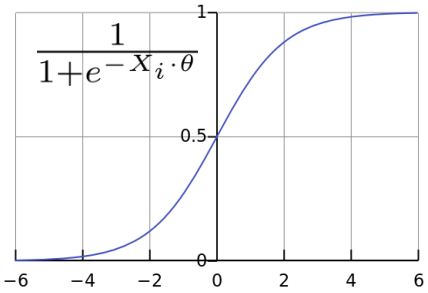
Will I **purchase** this product?



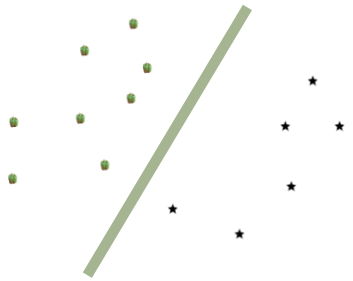

Will I **click on** this ad?




Combining features using naïve Bayes models

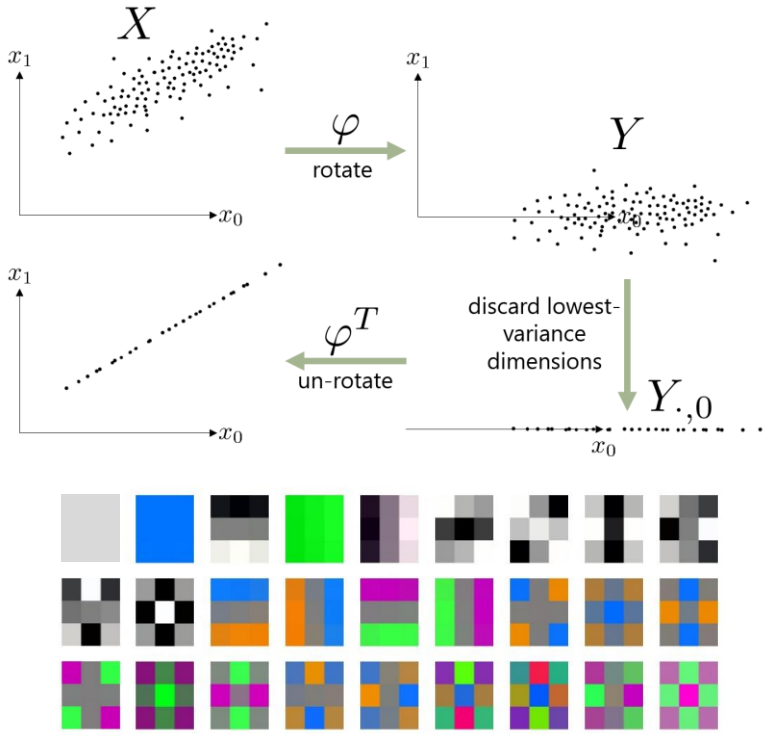



Logistic regression

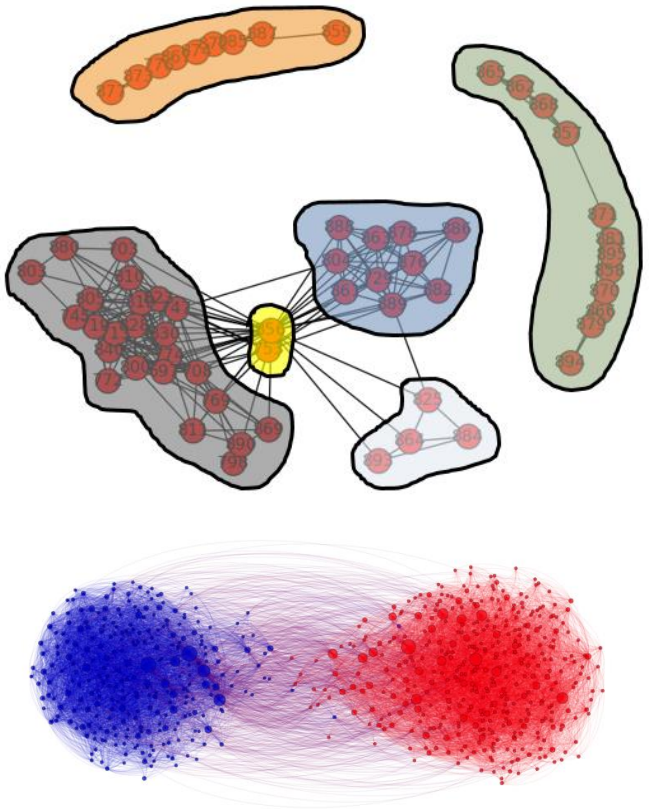


Support vector machines

# 3. Dimensionality reduction



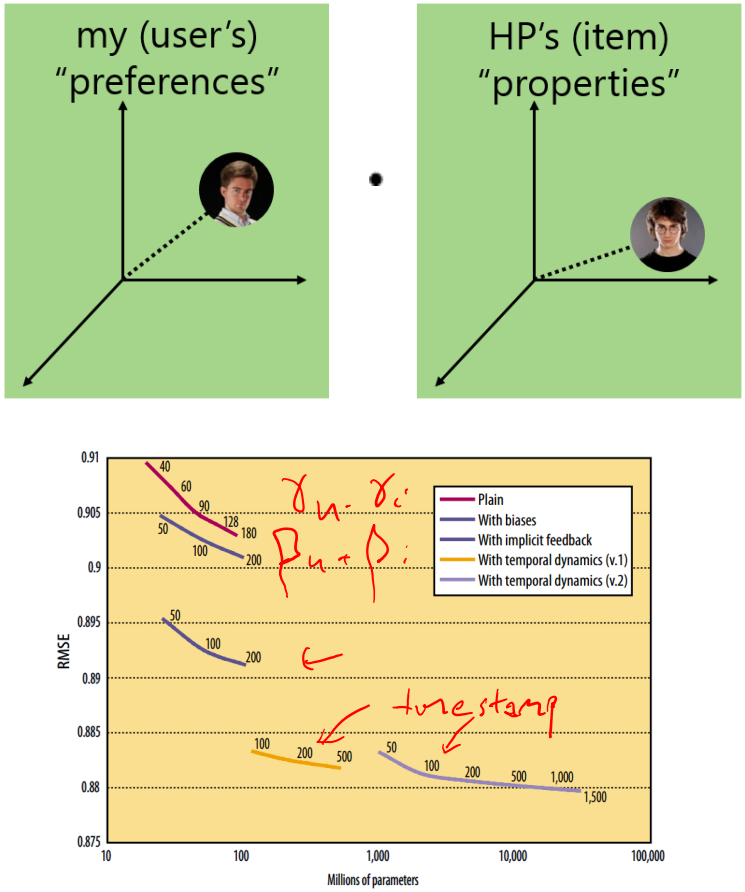

Principal component analysis



Community detection

# 4. Recommender Systems

Rating distributions and the missing-not-at-random assumption




Latent-factor models

# CSE 258 – Lecture 17

## Web Mining and Recommender Systems

## Regression for sequence data

# Week 1 – Regression

Given **labeled training data** of the form

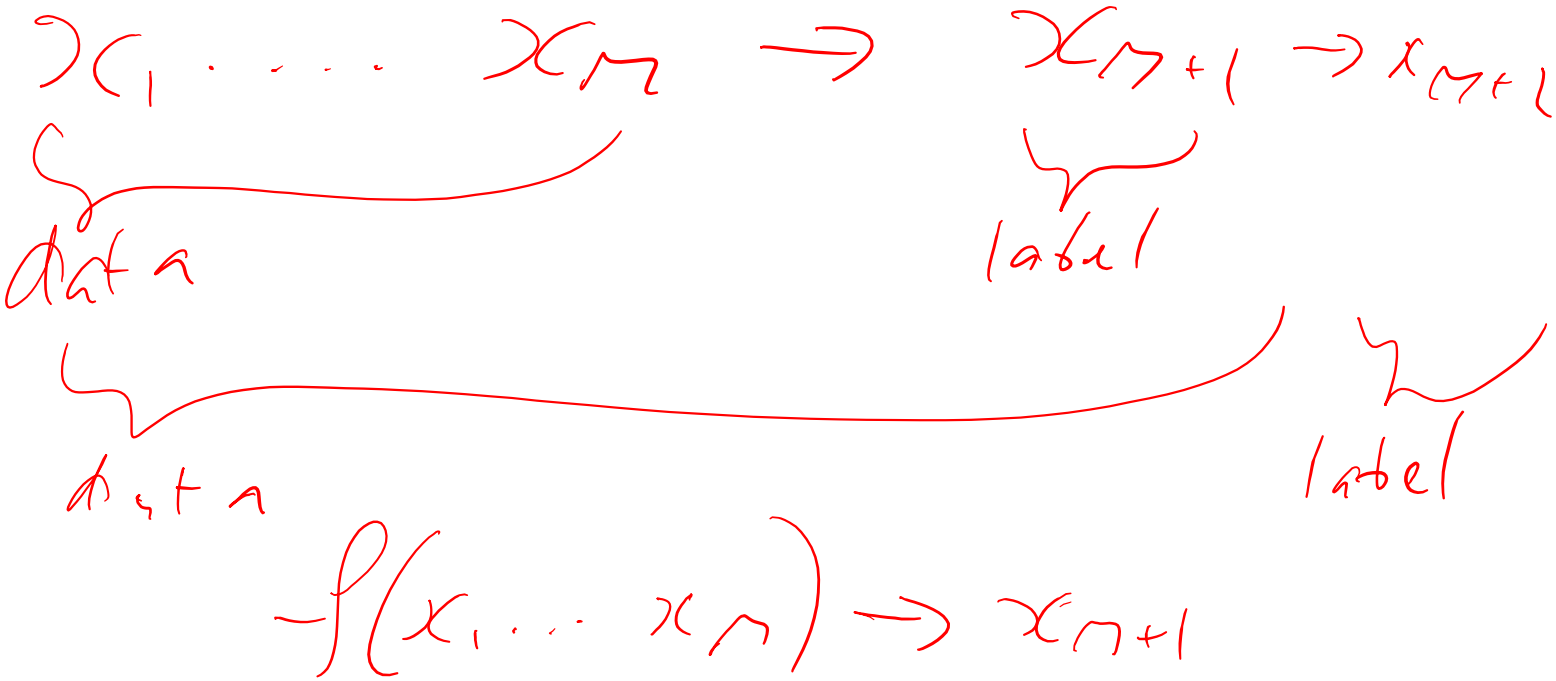
$$\{(\text{data}_1, \text{label}_1), \ldots, (\text{data}_n, \text{label}_n)\}$$

Infer the function

$$f(\text{data}) \xrightarrow{?} \text{labels}$$

# Time-series regression

Here, we'd like to predict sequences of **real-valued** events as accurately as possible.



# Time-series regression

**Method 1:** maintain a "moving average" using a window of some fixed length

$$f(x_1, \ldots, x_m) = \frac{x_m + x_{m-1} + x_{m-2} \cdots + x_{m-K+1}}{K}$$

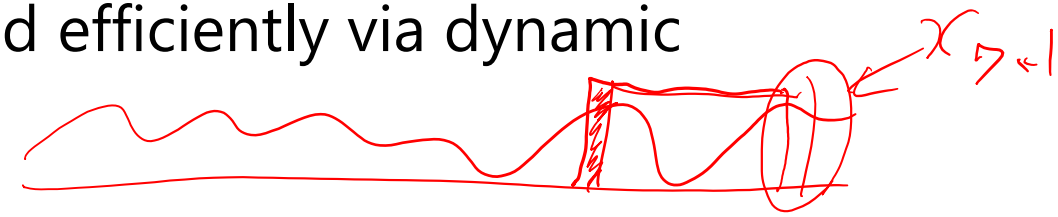
$$\frac{\sum_{k=0}^{K-1} x_{m-k}}{K}$$

$O(m \times K)$

# Time-series regression

**Method 1:** maintain a "moving average" using a window of some fixed length

- This can be computed efficiently via dynamic programming:



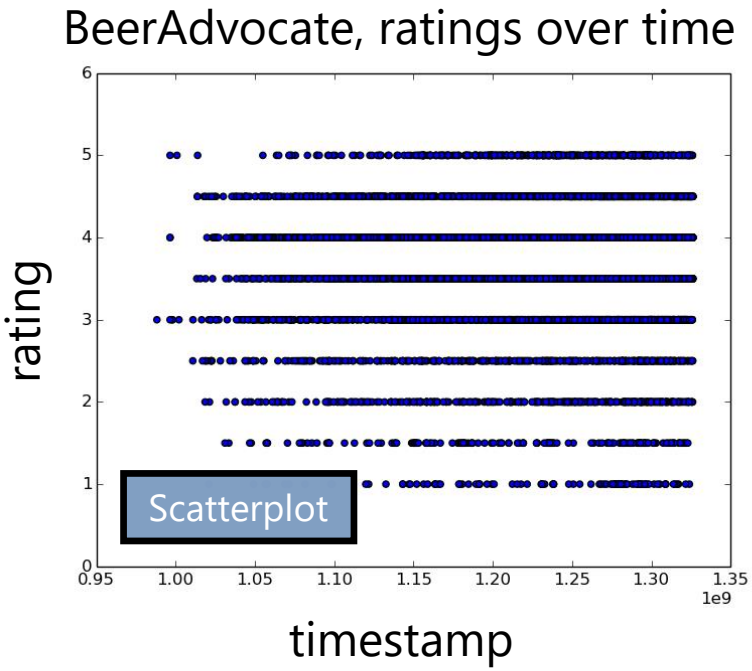
$$f(x_1, \ldots, x_{m+1}) =$$

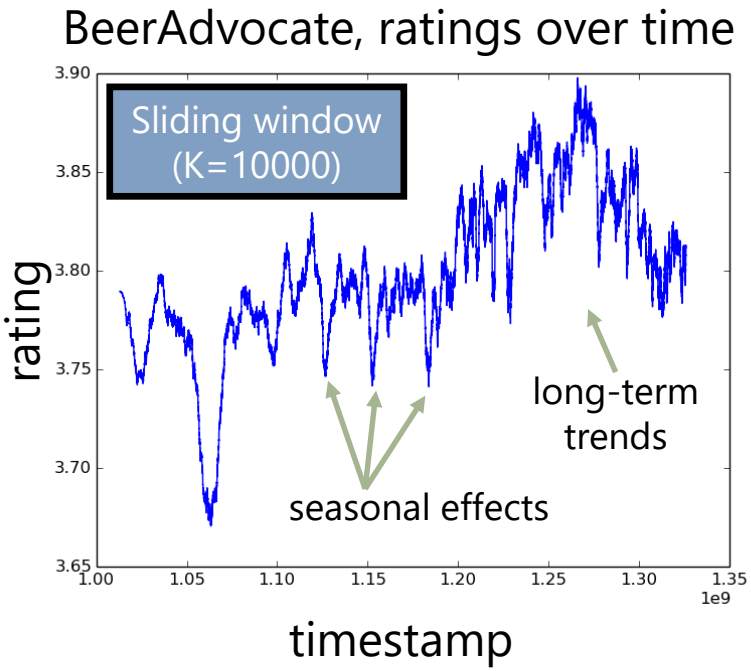
$$\frac{k f(x_1 \ldots x_m) - x_{m-k+1} + x_m}{k}$$

$O(m+k)$ vs $O(m \times k)$

# Time-series regression

## Also useful to plot data:



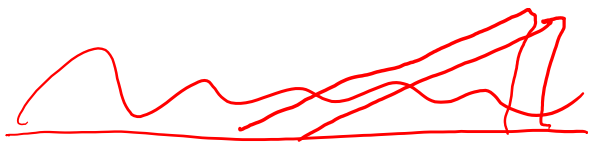




Code on:
http://jmcauley.ucsd.edu/code/week10.py

# Time-series regression

**Method 2:** weight the points in the moving average by age

$$f(x_1, \ldots, x_m) = \frac{K x_m + (K-1) x_{m-1} + (K-2) x_{m-2} \cdots 1 x_{m-K+1}}{1 + 2 + 3 \cdots K}$$



$$= \frac{\sum_{k=0}^{K-1} (K-k) x_{m-k}}{\binom{K}{2}}$$

# Time-series regression

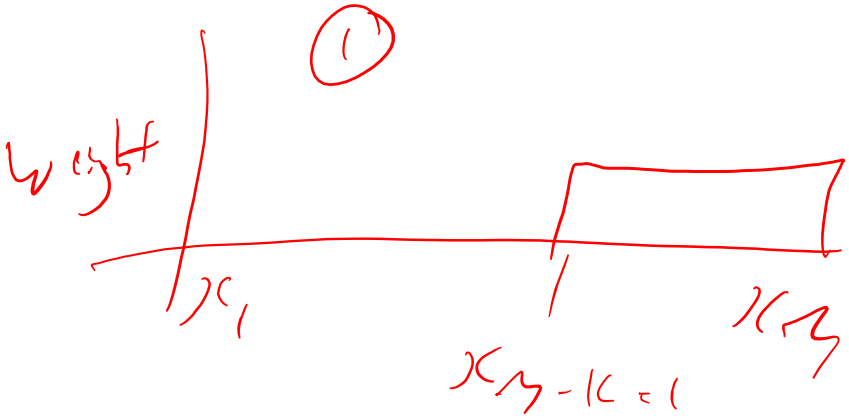
**Method 3:** weight the most recent points exponentially higher

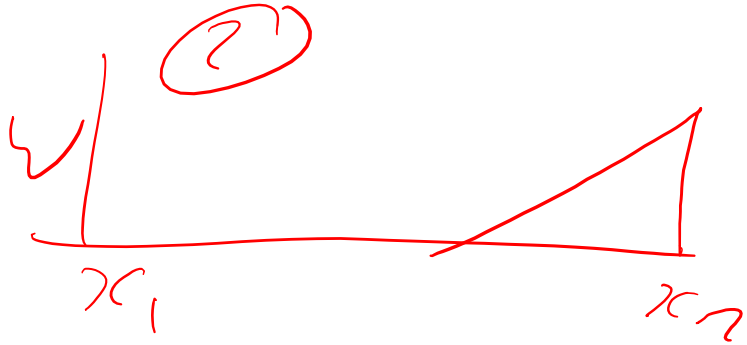
$$f(x_1) = x_1$$

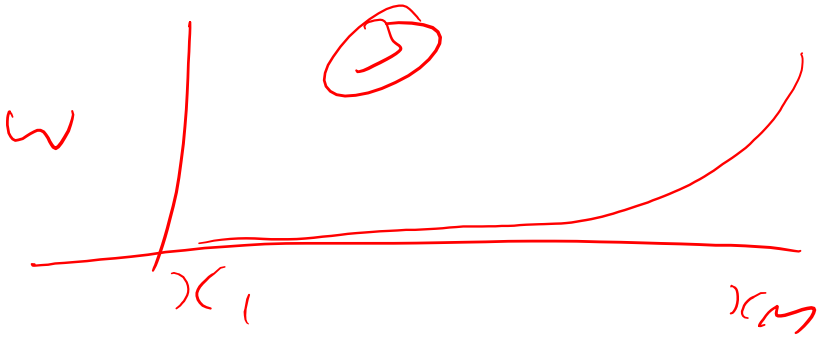
$$f(x_1, \ldots, x_m) = \alpha x_m + (1-\alpha) f(x_1, \ldots x_{m-1})$$

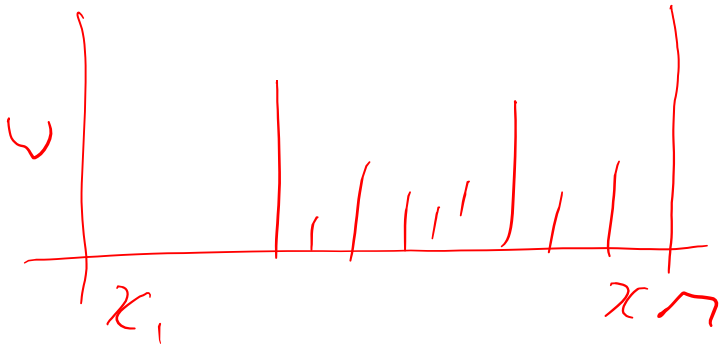
# Methods 1, 2, 3

Method 1: Sliding window
Method 2: Linear decay
Method 3: Exponential decay



# Time-series regression

**Method 4:** all of these models are assigning **weights** to previous values using some predefined scheme, why not just **learn** the weights?

$$f(x_1, \ldots, x_m) = \sum_{k=0}^{k-1} \theta_k x_{n-k}$$

$\theta_0 x_n + \theta_1 x_{n-1} + \ldots \theta_{k-1} x_{n-k+1}$

$\theta = \arg\min_{\theta} \sum_i \left( f(x_1 \ldots x_i) - x_{i+1} \right)^2$

# Time-series regression

**Method 4:** all of these models are assigning **weights** to previous values using some predefined scheme, why not just **learn** the weights?

- We can now fit this model using least-squares
- This procedure is known as **autoregression**
- Using this model, we can capture **periodic** effects, e.g. that the traffic of a website is most similar to its traffic 7 days ago

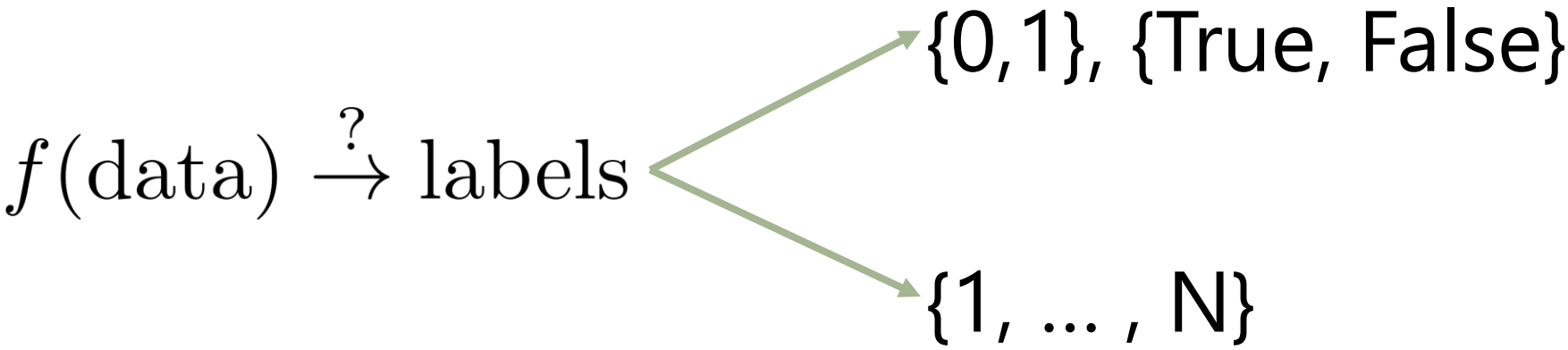
# CSE 258 – Lecture 17

## Web Mining and Recommender Systems

## Classification of sequence data

# Week 2

How can we predict **binary** or **categorical** variables?




Another simple algorithm: nearest neighbo(u)rs

# Time-series classification

As you recall…
The longest-common subsequence algorithm is a standard dynamic programming problem

| | - | A | G | C | A | T |
|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 1 | 1 | 1 | 1 |
| A | 0 | 1 | 1 | 1 | 2 | 2 |
| C | 0 | 1 | 1 | 2 | 2 | 2 |

1st sequence

2nd sequence

# Time-series classification

As you recall…
The longest-common subsequence algorithm is a standard dynamic programming problem

| | - | A | G | C | A | T |
|---|---|---|---|---|---|---|
| - | 0 | 0 | 0 | 0 | 0 | 0 |
| G | 0 | ↑ 0 | ↖ 1 | ← 1 | ← 1 | ← 1 |
| A | 0 | ↖ 1 | ↑ 1 | ↑ 1 | ↖ 2 | ← 2 |
| C | 0 | ↑ 1 | ↑ 1 | ↖ 2 | ↑ 2 | ↑ 2 |

1st sequence

2nd sequence

← = optimal move is to delete from 1st sequence

↑ = optimal move is to delete from 2nd sequence

↑ = either deletion is equally optimal

↖ = optimal move is a match

# Time-series classification

The same type of algorithm is used to find correspondences between time-series data (e.g. speech signals), whose length may vary in time/speed

```
DTW_cost = infty
for i in range(1,N):
  for j in range(1,M):
    d = dist(s[i], t[j]) # Distance between
sequences s and t and points i and j
    DTW[i,j] = d + min(DTW[i-1, j  ],
                       DTW[i,   j-1],
                       DTW[i-1, j-1]

return DTW[N,M]
```

skip from seq. 1

skip from seq. 2

match

output is a **distance** between the two sequences

# Time-series classification

- This is a simple procedure to infer the similarity between sequences, so we could classify them (for example) using nearest-neighbours (i.e., by comparing a sequence to others with known labels)

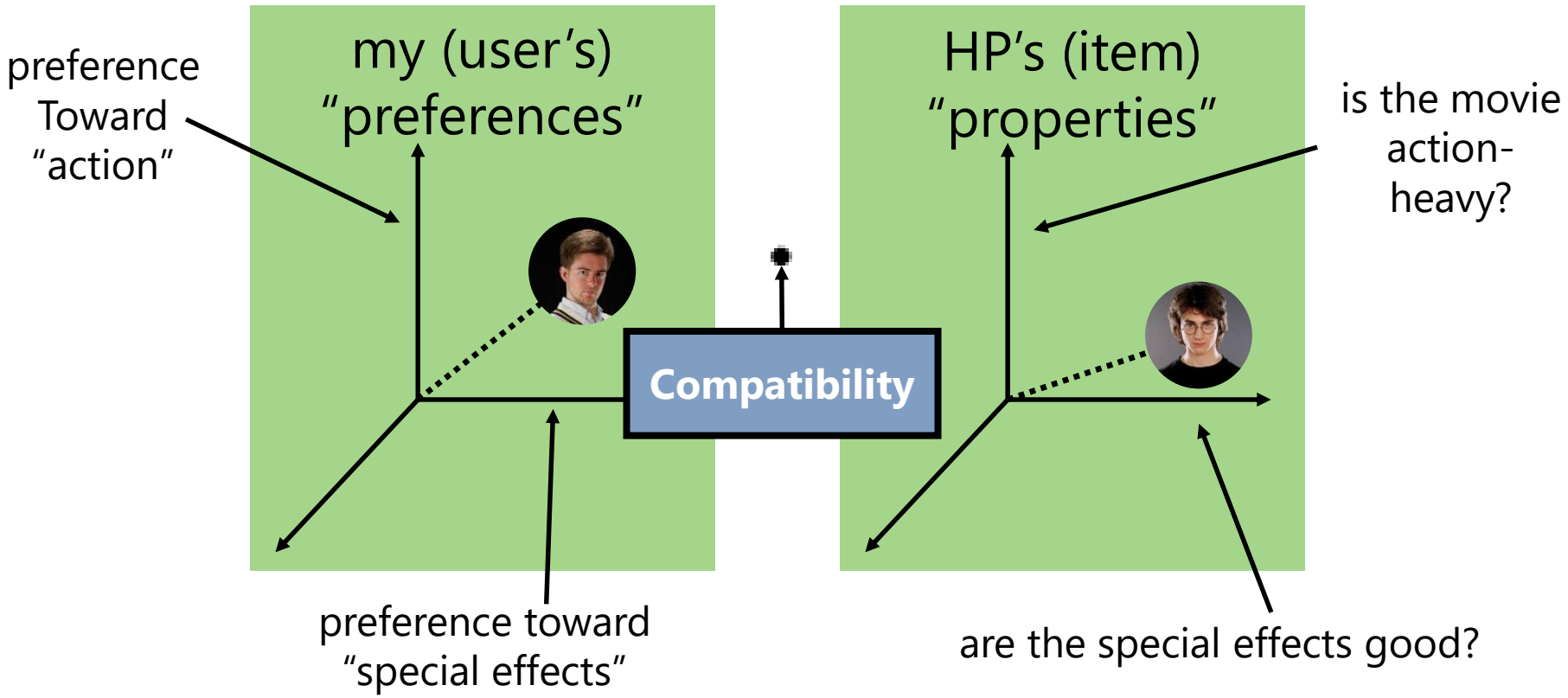
# CSE 258 – Lecture 17

## Web Mining and Recommender Systems

## Temporal recommender systems

# Week 4/5

**Recommender Systems** go beyond the methods we've seen so far by trying to model the **relationships** between people and the items they're evaluating



# Week 4/5

Predict a user's rating of an item according to:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

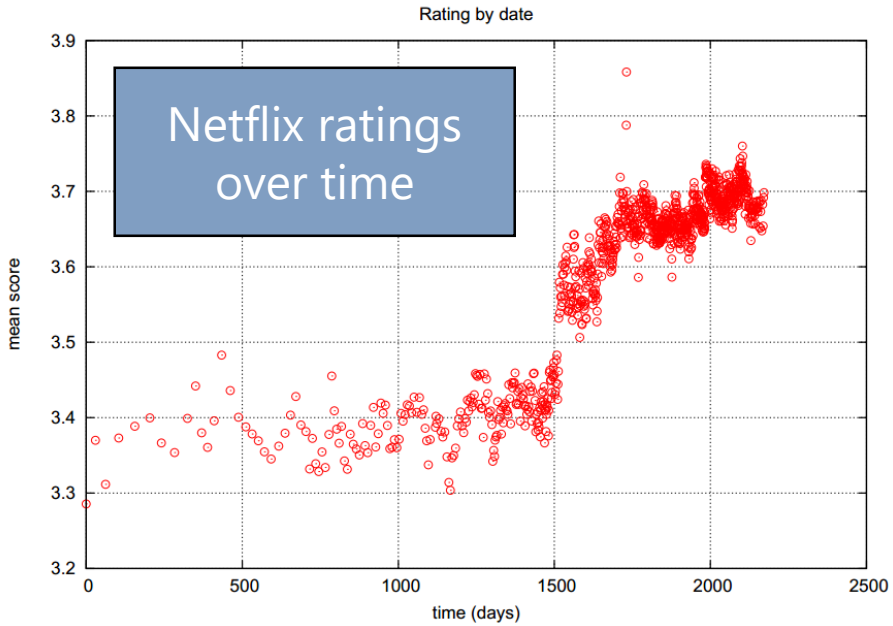
By solving the optimization problem:

$$\arg\min_{\alpha,\beta,\gamma} \underbrace{\sum_{u,i}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i})^2}_{\text{error}} + \underbrace{\lambda\left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2\right]}_{\text{regularizer}}$$

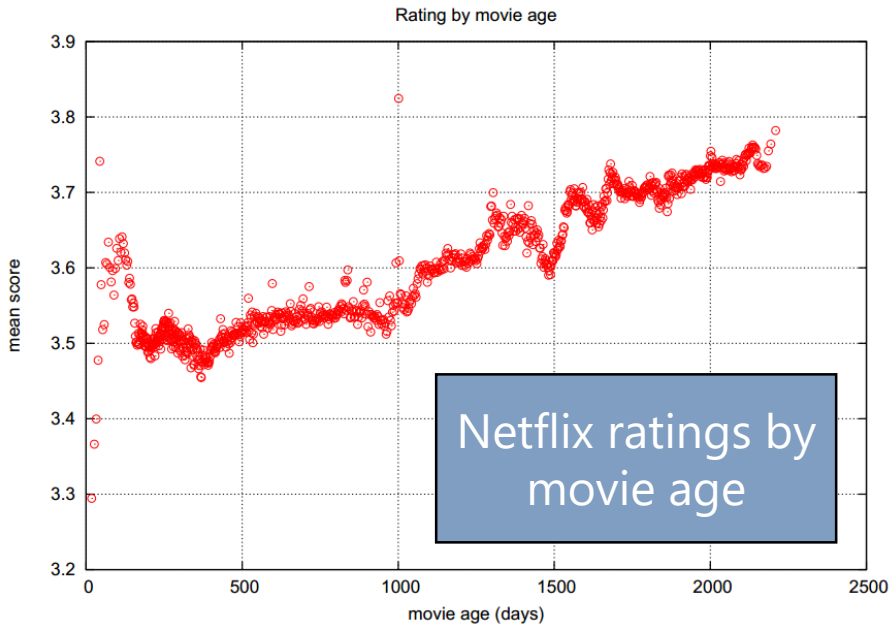
(e.g. using stochastic gradient descent)

# Temporal latent-factor models

To build a reliable system (and to win the Netflix prize!) we need to account for **temporal dynamics:**




(Netflix changed their interface)




(People tend to give higher ratings to older movies)

So how was this actually done?

Figure from Koren: "Collaborative Filtering with Temporal Dynamics" (KDD 2009)

# Temporal latent-factor models

To start with, let's just assume that it's only the **bias** terms that explain these types of temporal variation (which, for the examples on the previous slides, is potentially enough)

$$b_{u,i}(t) = \alpha + \beta_u(t) + \beta_i(t)$$

**Idea:** temporal dynamics for *items* can be explained by long-term, gradual changes, whereas for users we'll need a different model that allows for "bursty", short-lived behavior

# Temporal latent-factor models

temporal bias model:

$$b_{u,i}(t) = \alpha + \beta_u(t) + \beta_i(t)$$

For item terms, just separate the dataset into (equally sized) bins:*

$$\beta_i(t) = \beta_i + \beta_{i,\mathrm{Bin}(t)}$$

*in Koren's paper they suggested ~30 bins corresponding to about 10 weeks each for Netflix

or bins for periodic effects (e.g. the day of the week):

$$\beta_i(t) = \beta_i + \beta_{i,\mathrm{Bin}(t)} + \beta_{i,\mathrm{period}(t)}$$

What about user terms?

- We need something much finer-grained
- **But** – for most users we have far too little data to fit very short term dynamics

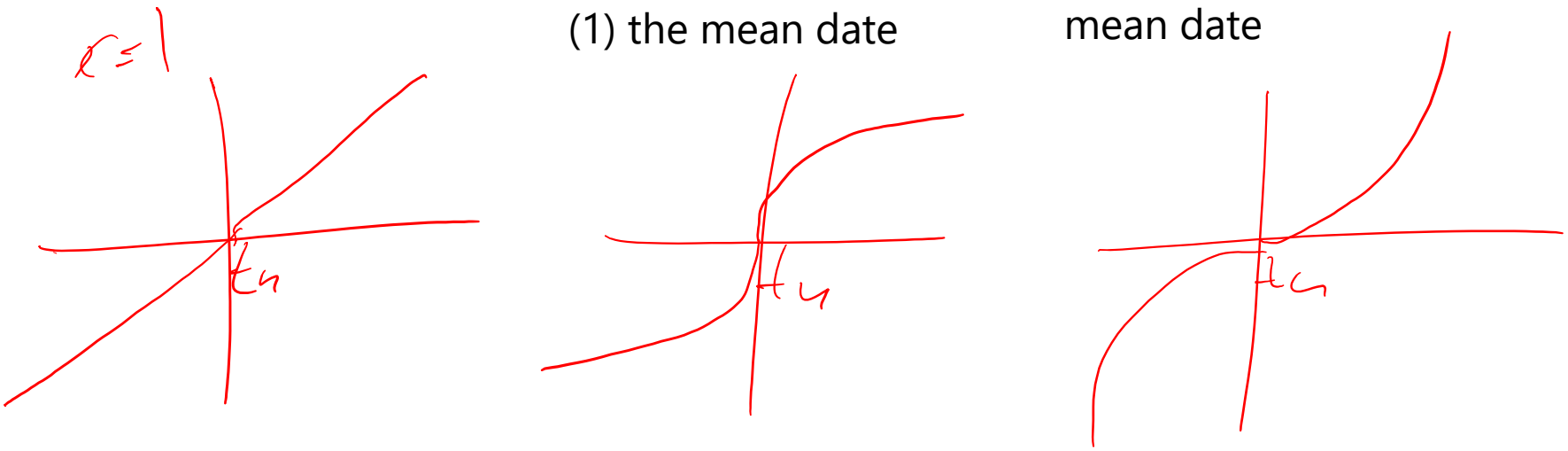
# Temporal latent-factor models

Start with a simple model of drifting dynamics for users:

**mean** rating date for user u

hyperparameter (ended up as x=0.4 for Koren)

$$\text{dev}_u(t) = \underbrace{\text{sign}(t - t_u)}_{\text{before (-1) or after (1) the mean date}} \cdot \underbrace{|t - t_u|}_{\text{days away from mean date}}^{x}$$



# Temporal latent-factor models

Start with a simple model of drifting dynamics for users:

**mean** rating date for user u

hyperparameter (ended up as x=0.4 for Koren)

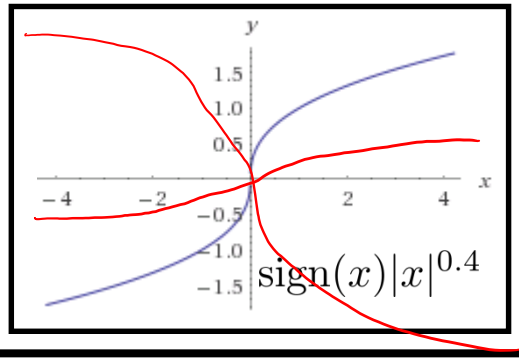
$$\mathrm{dev}_u(t) = \underbrace{\mathrm{sign}(t - t_u)}_{\text{before (-1) or after (1) the mean date}} \cdot \underbrace{|t - t_u|}_{\text{days away from mean date}}^{x}$$

time-dependent user bias can then be defined as:

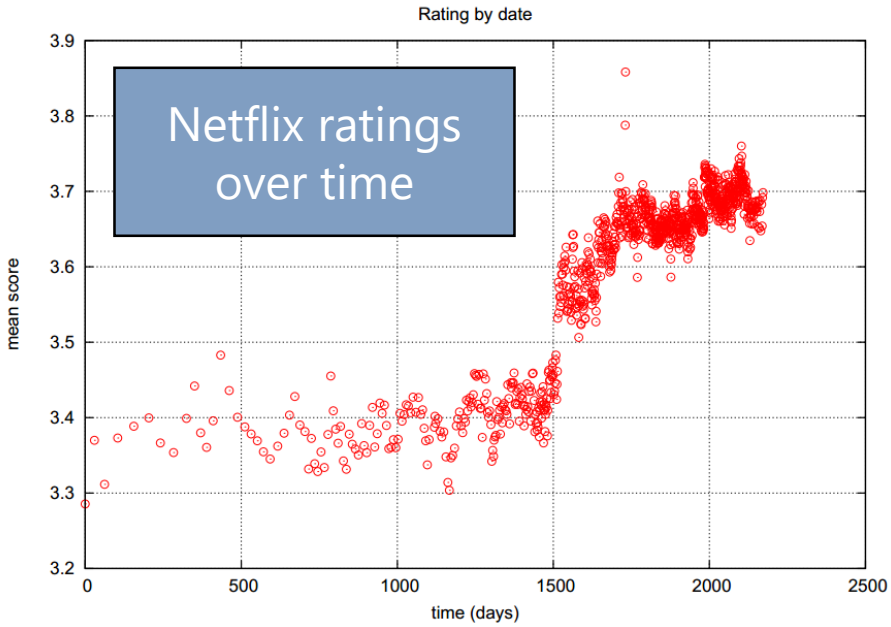
$$\beta_u^{(1)}(t) = \beta_u + \alpha_u \cdot \mathrm{dev}_u(t)$$

overall user bias

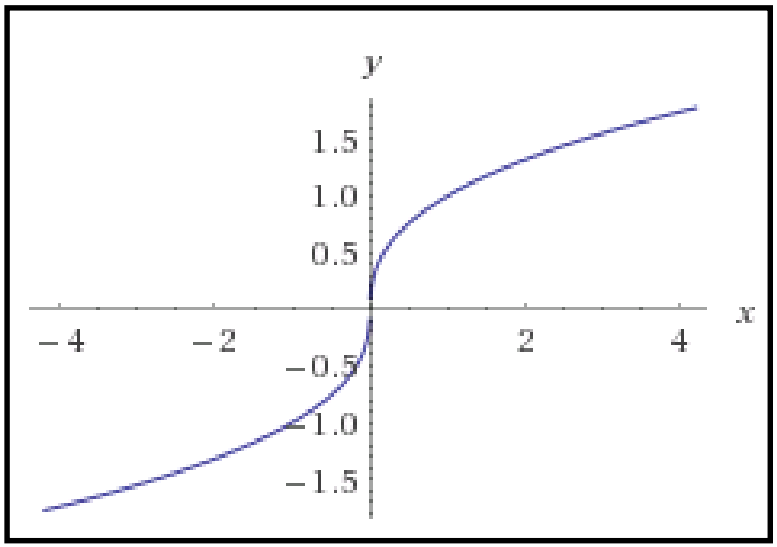
sign and scale for deviation term



# Temporal latent-factor models




Real data




Fitted model

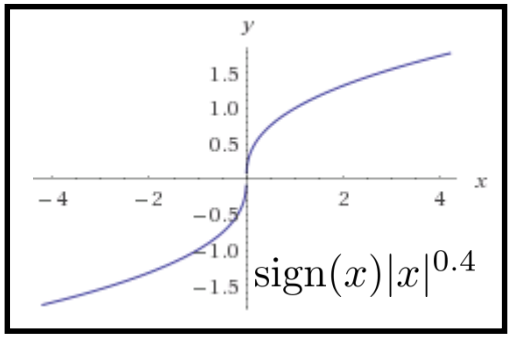
# Temporal latent-factor models

time-dependent user bias can then be defined as:

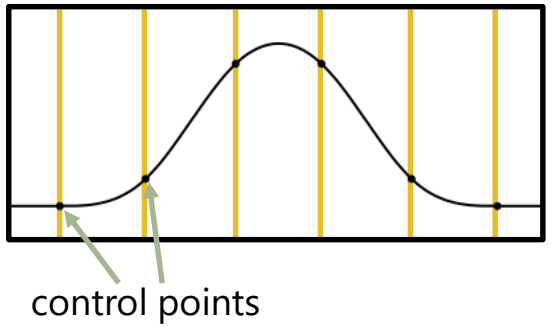
$$\beta_u^{(1)}(t) = \beta_u + \alpha_u \cdot \mathrm{dev}_u(t)$$

overall user bias

sign and scale for deviation term




- Requires only two parameters per user and captures some notion of temporal "drift" (even if the model found through cross-validation is (to me) completely unintuitive)
- To develop a slightly more expressive model, we can interpolate smoothly between biases using splines



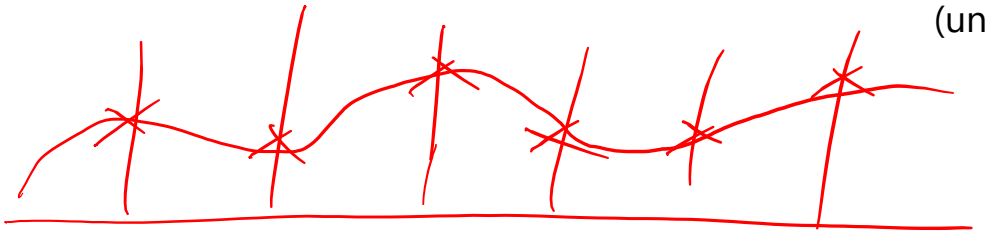
# Temporal latent-factor models

number of control points for this user
(k_u = n_u^0.25 in Koren)

user bias associated with this control point

$$\beta_u^{(2)}(t) = \beta_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}}$$

time associated with control point
(uniformly spaced)



# Temporal latent-factor models

number of control points for this user (k_u = n_u^0.25 in Koren)

user bias associated with this control point

$$\beta_u^{(2)}(t) = \beta_u + \frac{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|} b_{t_l}^u}{\sum_{l=1}^{k_u} e^{-\gamma|t-t_l^u|}}$$

time associated with control point (uniformly spaced)

- This is now a reasonably flexible model, but still only captures *gradual drift*, i.e., it can't handle sudden changes (e.g. a user simply having a bad day)

# Temporal latent-factor models

- Koren got around this just by adding a "per-day" user bias:

$$\beta_{u,t}$$

bias for a particular day (or session)

- Of course, this is only useful for particular days in which users have a lot of (abnormal) activity
- The final (time-evolving bias) model then combines all of these factors:

$$\beta_{u,i}(t) = \alpha + \beta_u + \alpha_u \cdot \mathrm{dev}_u(t) + \beta_{u,t} + \beta_i + \beta_{i,\mathrm{Bin}(t)}$$

global offset

gradual deviation (or splines)

item bias

gradual item bias drift

user bias

single-day dynamics

# Temporal latent-factor models

Finally, we can add a time-dependent scaling factor:

$$\beta_{u,i}(t) = \alpha + \beta_u + \alpha_u \cdot \mathrm{dev}_u(t) + \beta_{u,t} + (\beta_i + \beta_{i,\mathrm{Bin}(t)}) \cdot c_u(t)$$

**also** defined as $c_u + c_{u,t}$

Latent factors can also be defined to evolve in the same way:

$$\gamma_{u,k}(t) = \gamma_{u,k} + \alpha_{u,k} \cdot \mathrm{dev}_u(t) + \gamma_{u,k,t}$$

factor-dependent user drift

factor-dependent short-term effects

# Temporal latent-factor models

## Summary

- Effective modeling of temporal factors was absolutely critical to this solution outperforming alternatives on Netflix's data
  - In fact, even with only temporally evolving *bias* terms, their solution was already ahead of Netflix's previous ("Cinematch") model

On the other hand…

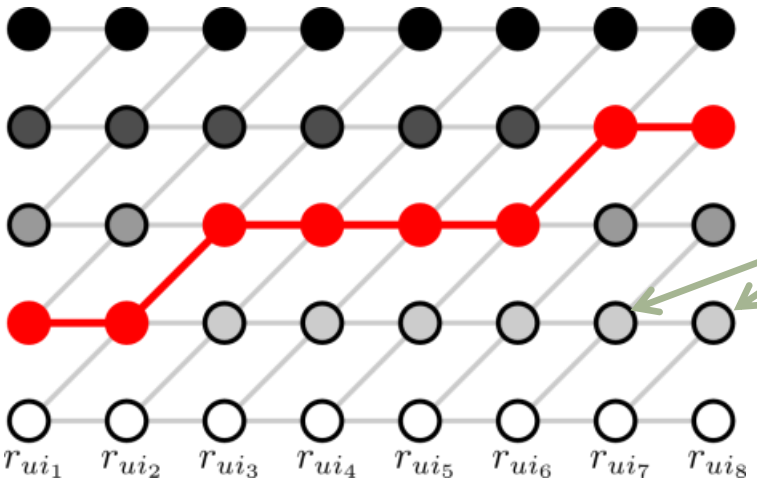
  - Many of the ideas here depend on dynamics that are quite specific to "Netflix-like" settings
- Some factors (e.g. short-term effects) depend on a high density of data per-user and per-item, which is not always available

# Temporal latent-factor models

## Summary

- Changing the setting, e.g. to model the stages of progression through the symptoms of a disease, or even to model the temporal progression of people's opinions on beers, means that alternate temporal models are required

**rows:** models of increasingly "experienced" users




**columns:** review timeline for one user

# Questions?

Further reading:
"Collaborative filtering with temporal dynamics"
Yehuda Koren, 2009
http://research.yahoo.com/files/kdd-fp074-koren.pdf