CSE 158 – Lecture 8

Web Mining and Recommender Systems

Extensions of latent-factor models,

(and more on the Netflix prize)

Summary so far

Recap

1. Measuring similarity between users/items for **binary** prediction
Jaccard similarity
2. Measuring similarity between users/items for **real-valued** prediction
cosine/Pearson similarity
3. Dimensionality reduction for **real-valued** prediction
latent-factor models

Last lecture…

In 2006, Netflix created a dataset of **100,000,000** movie ratings
Data looked like:

$$(\text{userID}, \text{itemID}, \text{time}, \text{rating})$$

The goal was to reduce the (R)MSE at predicting ratings:

$$\text{RMSE}(f) = \sqrt{\frac{1}{N} \sum_{u,i,t \in \text{test set}} (f(u,i,t) - r_{u,i,t})^2}$$

model's prediction ($f(u,i,t)$)

ground-truth ($r_{u,i,t}$)

Whoever first manages to reduce the RMSE by **10%** versus Netflix's solution wins **$1,000,000**

Last lecture...

Let's start with the simplest possible model:

$$f(u, i) = \alpha$$

user item

$$\alpha = \overline{R}$$

Last lecture…

What about the **2nd** simplest model?

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

user item

how much does this user tend to rate things above the mean?

does this item tend to receive higher ratings than others

e.g.

$\alpha = 4.2$

$\beta_{\text{pitch black}} = -0.1$

$\beta_{\text{julian}} = -0.2$

Rating prediction

The optimization problem becomes:

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

Rating prediction

The optimization problem becomes:

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

Rating prediction

Iterative procedure – repeat the following updates until convergence:

$$\alpha = \frac{\sum_{u,i \in \text{train}} (R_{u,i} - (\beta_u + \beta_i))}{N_{\text{train}}}$$

$$\beta_u = \frac{\sum_{i \in I_u} R_{u,i} - (\alpha + \beta_i)}{\lambda + |I_u|}$$

$$\beta_i = \frac{\sum_{u \in U_i} R_{u,i} - (\alpha + \beta_u)}{\lambda + |U_i|}$$

t

t+1

t+2

(exercise: write down derivatives and convince yourself of these update equations!)

Rating prediction

Looks good (and actually works surprisingly well), but doesn't solve the basic issue that we started with

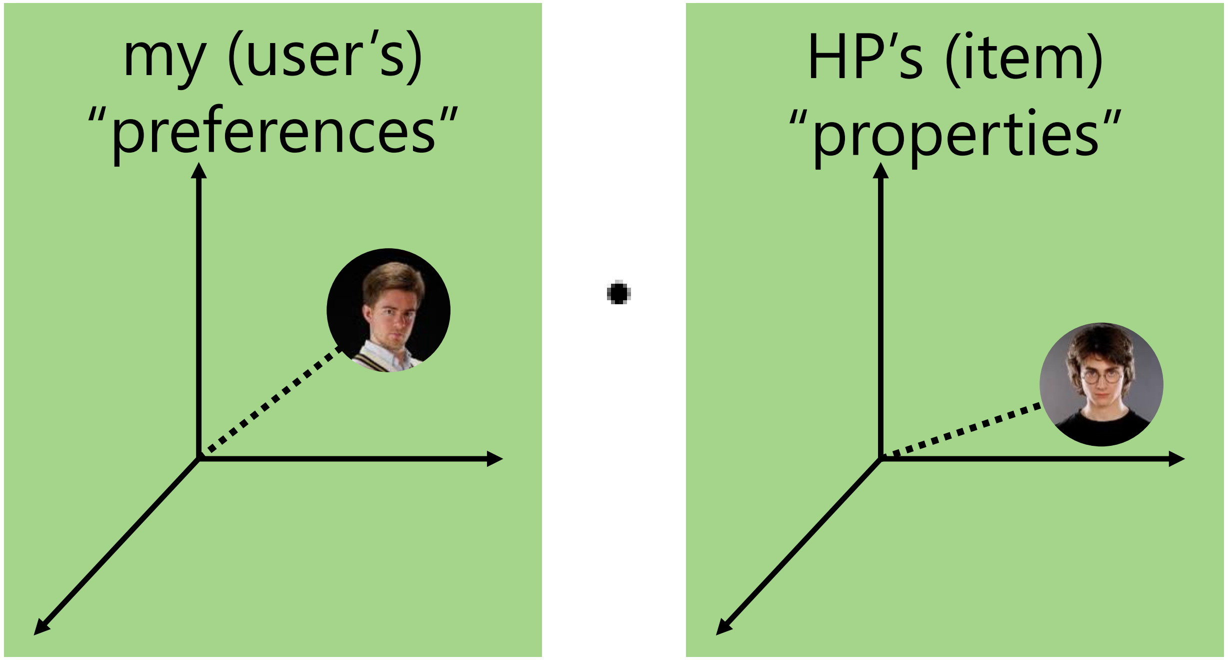
$$f(\text{user features}, \text{movie features}) =$$

$$= \underbrace{\langle \phi(\text{user features}), \theta_{\text{user}} \rangle}_{\text{user predictor}} + \underbrace{\langle \phi(\text{movie features}), \theta_{\text{movie}} \rangle}_{\text{movie predictor}}$$

That is, we're **still** fitting a function that treats users and items independently

Recommending things to people

How about an approach based on **dimensionality reduction?**



i.e., let's come up with low-dimensional representations of the users and the items so as to best explain the data

Dimensionality reduction

We already have some tools that ought to help us, e.g. from week 3:

$$R = \begin{pmatrix} 5 & 3 & \cdots & 1 \\ 4 & 2 & & 1 \\ 3 & 1 & & 3 \\ 2 & 2 & & 4 \\ 1 & 5 & & 2 \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & 1 \end{pmatrix}$$

What is the best low-rank approximation of R in terms of the mean-squared error?

Dimensionality reduction

We already have some tools that ought to help us, e.g. from week 3:

$$R = \begin{pmatrix} 5 & 3 & \cdots & 1 \\ 4 & 2 & & 1 \\ 3 & 1 & & 3 \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & 1 \end{pmatrix}$$

Singular Value Decomposition

$$R = U \Sigma V^T$$

(square roots of) eigenvalues of RR^T

eigenvectors of RR^T

eigenvectors of $R^T R$

The "best" rank-K approximation (in terms of the MSE) consists of taking the eigenvectors with the highest eigenvalues

Dimensionality reduction

But! Our matrix of ratings is only partially observed; and it's **really big!**

$$R = \begin{pmatrix} 5 & 3 & \cdots & \cdot \\ 4 & 2 & & 1 \\ 3 & \cdot & & 3 \\ \cdot & 2 & & 4 \\ 1 & 5 & & \cdot \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & \cdot \end{pmatrix}$$

Missing ratings

SVD is **not defined** for partially observed matrices, and it is **not practical** for matrices with 1Mx1M+ dimensions

Latent-factor models

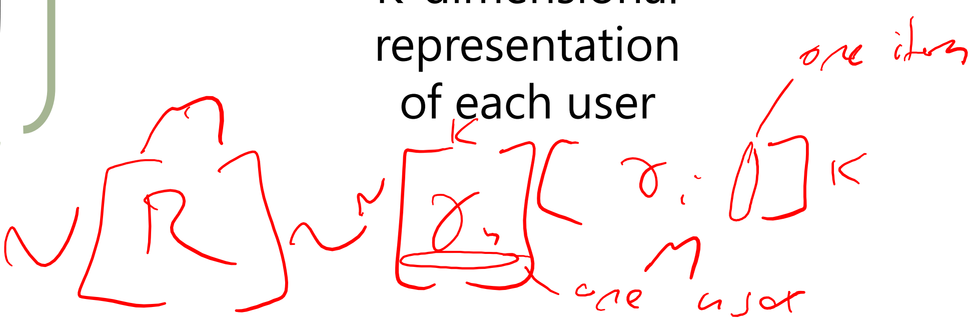
Instead, let's solve approximately using gradient descent

$$R = \underbrace{\begin{pmatrix} 5 & 3 & \cdots & \cdot \\ 4 & 2 & & 1 \\ 3 & \cdot & & 3 \\ \cdot & 2 & & 4 \\ 1 & 5 & & \cdot \\ \vdots & & \ddots & \vdots \\ 1 & 2 & \cdots & \cdot \end{pmatrix}}_{\text{items}} \Bigg\} \text{users}$$

K-dimensional representation of each item

$$R \simeq UV^T$$

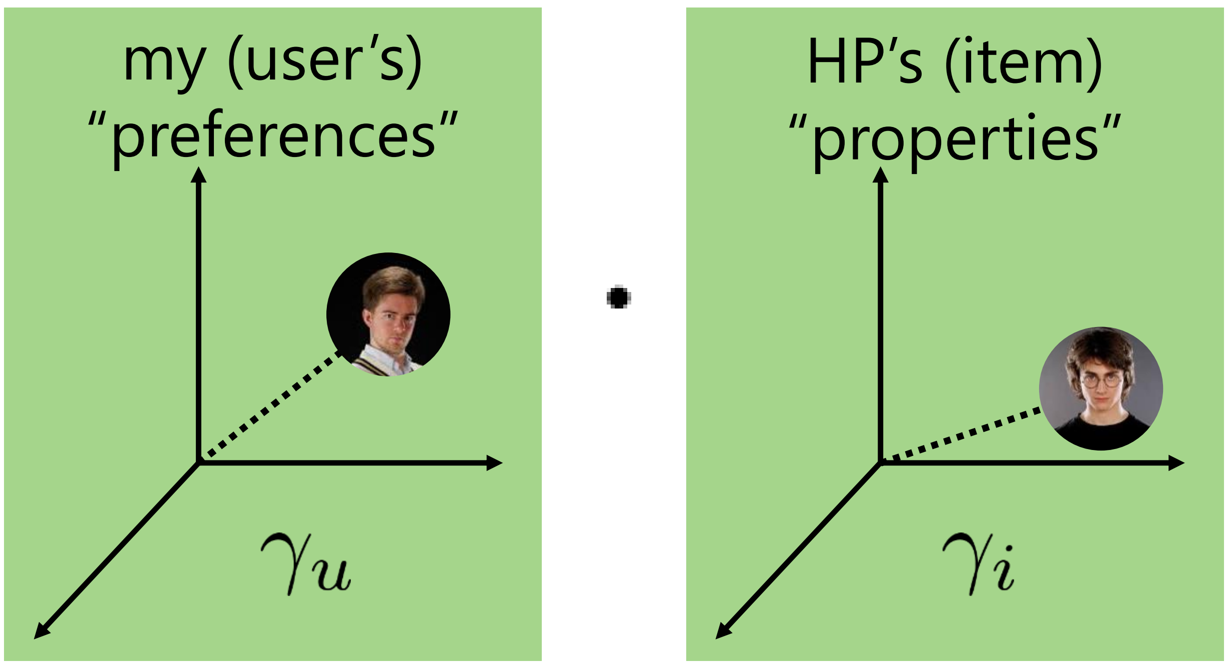
K-dimensional representation of each user



Latent-factor models

Let's write this as:

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



Latent-factor models

Let's write this as:

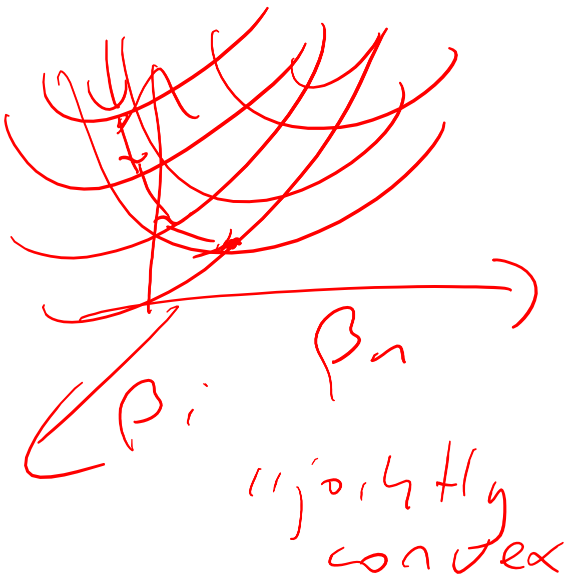
$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i$$

Our optimization problem is then

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

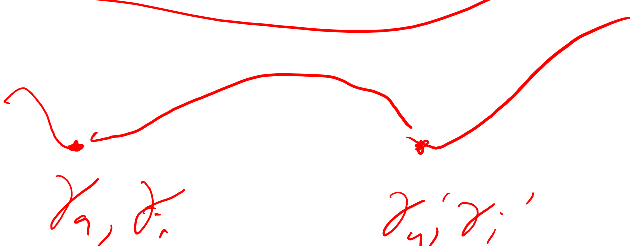
Latent-factor models

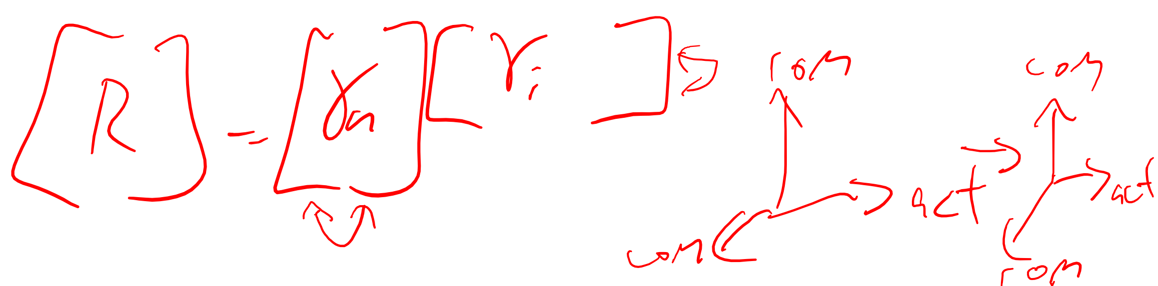
Problem: this is certainly not convex



① it's continuous

② If you give me a local minimum γ_u, γ_i, I can find another equivalent minimum





Latent-factor models

Oh well. We'll just solve it approximately

Observation: if we know either the user or the item parameters, the problem becomes easy

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

e.g. fix gamma_i – pretend we're fitting parameters for features

Latent-factor models

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

$$\frac{\partial obj}{\partial \gamma_{uk}} = \sum_{i \in I_u} 2\gamma_{ik}(\alpha+\beta_u+\beta_i+\gamma_u\cdot\gamma_i-R_{u,i}) + 2\lambda\gamma_{uk}$$

This gives rise to a simple (though approximate) solution

objective:

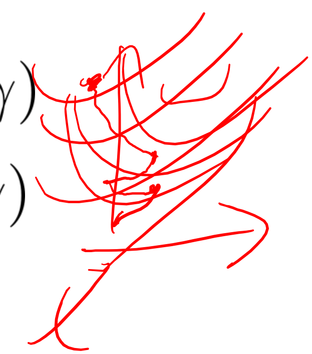
$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda \left[\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \|\gamma_i\|_2^2 + \sum_u \|\gamma_u\|_2^2 \right]$$

$$= \arg\min_{\alpha,\beta,\gamma} objective(\alpha, \beta, \gamma)$$

1) fix γ_i. Solve $\arg\min_{\alpha,\beta,\gamma_u} objective(\alpha, \beta, \gamma)$

2) fix γ_u. Solve $\arg\min_{\alpha,\beta,\gamma_i} objective(\alpha, \beta, \gamma)$

3,4,5…) repeat until convergence



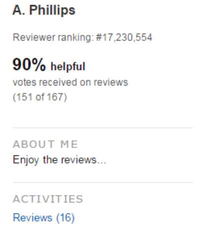
Each of these subproblems is "easy" – just regularized least-squares, like we've been doing since week 1. This procedure is called **alternating least squares.**

Latent-factor models

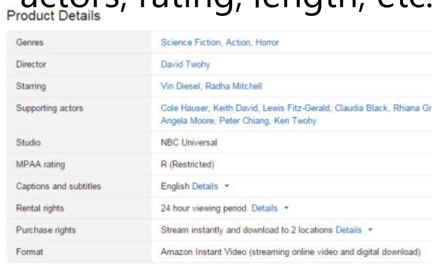
Observation: we went from a method which uses **only** features:

$$f(\text{user features}, \text{movie features}) \rightarrow \text{star rating}$$

User features: age, gender, location, etc.



Movie features: genre, actors, rating, length, etc.



Product Details

Genres	Science Fiction, Action, Horror
Director	David Twohy
Starring	Vin Diesel, Radha Mitchell
Supporting actors	Cole Hauser, Keith David, Lewis Fitz-Gerald, Claudia Black, Rhiana Gr Angela Moore, Peter Chiang, Ken Twohy
Studio	NBC Universal
MPAA rating	R (Restricted)
Captions and subtitles	English Details
Rental rights	24 hour viewing period. Details
Purchase rights	Stream instantly and download to 2 locations Details
Format	Amazon Instant Video (streaming online video and digital download)

to one which **completely ignores** them:

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

Overview & recap

So far we've followed the programme below:

1. Measuring similarity between users/items for **binary** prediction (e.g. Jaccard similarity)
2. Measuring similarity between users/items for **real-valued** prediction (e.g. cosine/Pearson similarity)
3. Dimensionality reduction for **real-valued** prediction (latent-factor models)
4. **Finally** – dimensionality reduction for **binary** prediction

One-class recommendation

How can we use **dimensionality reduction** to predict **binary** outcomes?

- In weeks 1&2 we saw **regression** and **logistic regression.** These two approaches use the same type of linear function to predict real-valued and binary outputs
- We can apply an analogous approach to binary recommendation tasks

One-class recommendation

This is referred to as "**one-class**" recommendation

- In weeks 1&2 we saw **regression** and **logistic regression.** These two approaches use the same type of linear function to predict real-valued and binary outputs
- We can apply an analogous approach to binary recommendation tasks

One-class recommendation

Suppose we have binary (0/1) observations (e.g. purchases) or positive/negative feedback (thumbs-up/down)

$$R = \begin{pmatrix} 1 & 0 & \cdots & 1 \\ 0 & 0 & & 1 \\ \vdots & & \ddots & \vdots \\ 1 & 0 & \cdots & 1 \end{pmatrix} \text{ or } \begin{pmatrix} -1 & ? & \cdots & 1 \\ ? & ? & & -1 \\ \vdots & & \ddots & \vdots \\ 1 & ? & \cdots & -1 \end{pmatrix}$$

purchased — didn't purchase — liked — didn't evaluate — didn't like

One-class recommendation

So far, we've been fitting functions of the form

$$R \simeq UV^T$$

- Let's change this so that we maximize the **difference** in predictions between positive and negative items
- E.g. for a user who likes an item *i* and dislikes an item *j* we want to maximize:

p(user prefers i over j)

$$\max \ln \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

user — clicked item — non-clicked item

One-class recommendation

We can think of this as maximizing the probability of correctly predicting pairwise preferences, i.e.,

$$p(i \text{ is preferred over } j) = \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

- As with logistic regression, we can now maximize the likelihood associated with such a model by gradient ascent
- In practice it isn't feasible to consider all pairs of positive/negative items, so we proceed by stochastic gradient ascent – i.e., randomly sample a (positive, negative) pair and update the model according to the gradient w.r.t. that pair

One-class recommendation

$$\max \ln \sigma(\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j)$$

$$obj = \sum_{u,i,j} -\log\left(1 + e^{\gamma_u \cdot \gamma_i - \gamma_u \cdot \gamma_j}\right)$$

$$\frac{\partial obj}{\partial \gamma_{uk}} \ldots$$

Summary

Recap

1. Measuring similarity between users/items for **binary** prediction
Jaccard similarity
2. Measuring similarity between users/items for **real-valued** prediction
cosine/Pearson similarity
3. Dimensionality reduction for **real-valued** prediction
latent-factor models
4. Dimensionality reduction for **binary** prediction
one-class recommender systems

Questions?

Further reading:

One-class recommendation:
http://goo.gl/08Rh59
Amazon's solution to collaborative filtering at scale:
http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf
An (expensive) textbook about recommender systems:
http://www.springer.com/computer/ai/book/978-0-387-85819-7
Cold-start recommendation (e.g.):
http://wanlab.poly.edu/recsys12/recsys/p115.pdf

CSE 158 – Lecture 8

Web Mining and Recommender Systems

Extensions of latent-factor models,

(and more on the Netflix prize!)

Extensions of latent-factor models

So far we have a model that looks like:

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

How might we extend this to:

- Incorporate features about users and items
- Handle implicit feedback
- Change over time

See **Yehuda Koren** (+Bell & Volinsky)'s magazine article:
"Matrix Factorization Techniques for Recommender Systems"
IEEE Computer, 2009

Extensions of latent-factor models

1) Features about users and/or items

(simplest case) Suppose we have **binary attributes** to describe users or items

$$A(u) = [1,0,1,1,0,0,0,0,0,1,0,1]$$

attribute vector for user *u*

e.g. is female — is male — is between 18-24yo

Extensions of latent-factor models

1) Features about users and/or items

(simplest case) Suppose we have **binary attributes** to describe users or items

- Associate a **parameter vector** with each attribute
- Each vector encodes how much a particular feature "offsets" the given latent dimensions

$$A(u) = [1,0,1,1,0,0,0,0,0,1,0,1]$$

attribute vector for user *u*

e.g. y_0 = [-0.2,0.3,0.1,-0.4,0.8]
~ "how does being male impact gamma_u"

Extensions of latent-factor models

1) Features about users and/or items

(simplest case) Suppose we have **binary attributes** to describe users or items

- Associate a **parameter vector** with each attribute
- Each vector encodes how much a particular feature "offsets" the given latent dimensions
- Model looks like:

$$f(u,i) = \alpha + \beta_u + \beta_i + \left(\gamma_u + \sum_{a \in A(u)} \rho_a\right) \cdot \gamma_i$$

- Fit as usual:

$$\arg\min_{\alpha,\beta,\gamma,\rho} \sum_{u,i \in \text{train}} \underbrace{(f(u,i) - r_{u,i})^2}_{\text{error}} + \underbrace{\lambda\Omega(\beta,\gamma)}_{\text{regularizer}}$$

Extensions of latent-factor models

2) Implicit feedback

Perhaps many users will never actually rate things, but may still interact with the system, e.g. through the movies they view, or the products they purchase (but never rate)

- Adopt a similar approach – introduce a binary vector describing a user's actions

$$N(u) = [1,0,0,0,1,0,....,0,1]$$

implicit feedback vector for user *u*

e.g. y_0 = [-0.1,0.2,0.3,-0.1,0.5]
Clicked on "Love Actually" but didn't watch

Extensions of latent-factor models

2) Implicit feedback

Perhaps many users will never actually rate things, but may still interact with the system, e.g. through the movies they view, or the products they purchase (but never rate)

- Adopt a similar approach – introduce a binary vector describing a user's actions
  - Model looks like:

$$f(u,i) = \alpha + \beta_u + \beta_i + (\gamma_u + \frac{1}{\|N(u)\|} \sum_{a \in N(u)} \rho_a) \cdot \gamma_i$$

normalize by the number of actions the user performed

Extensions of latent-factor models

3) Change over time

There are a number of reasons why rating data might be subject to temporal effects…

Extensions of latent-factor models

3) Change over time

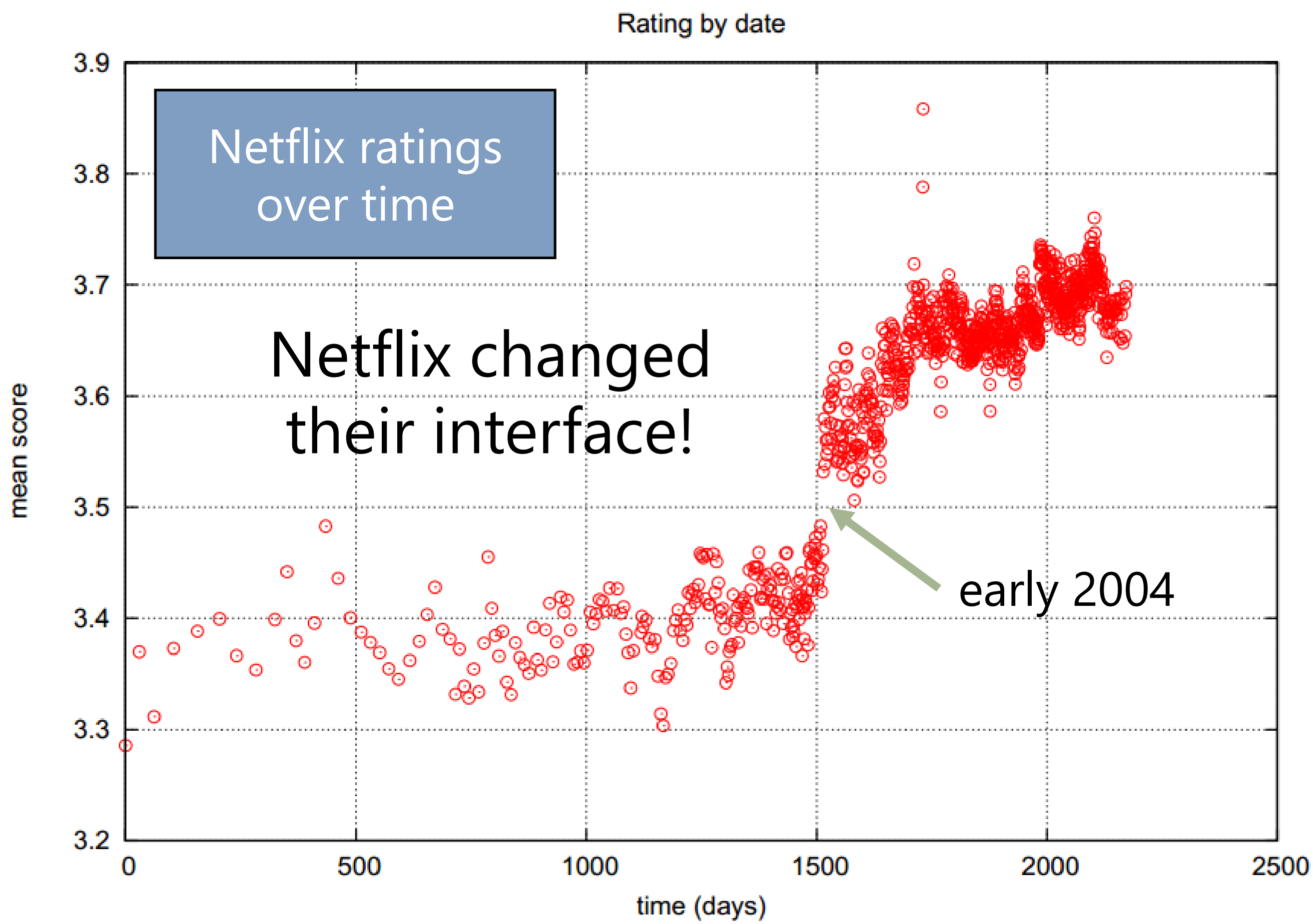


Figure from Koren: “Collaborative Filtering with Temporal Dynamics” (KDD 2009)

Extensions of latent-factor models

3) Change over time

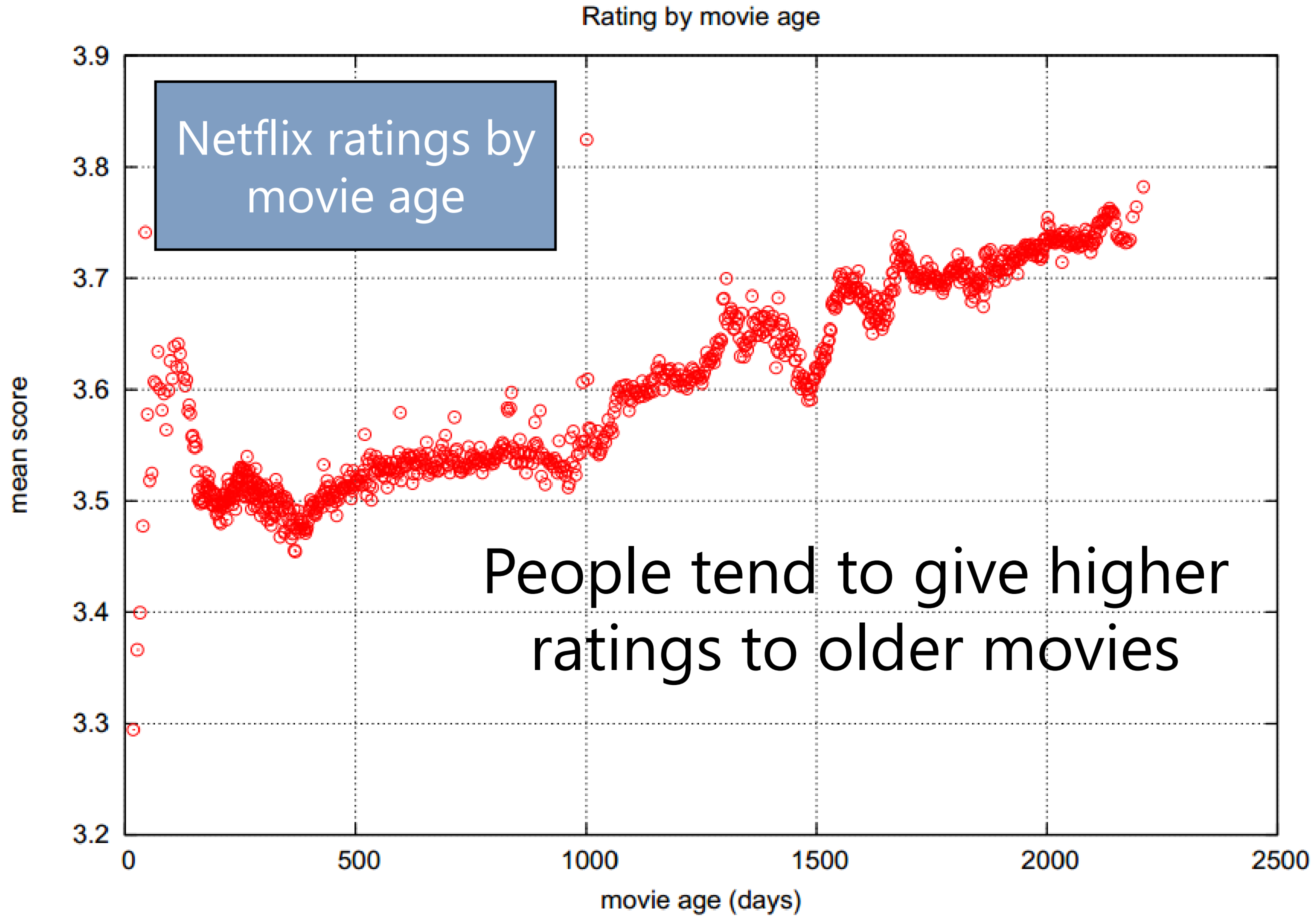
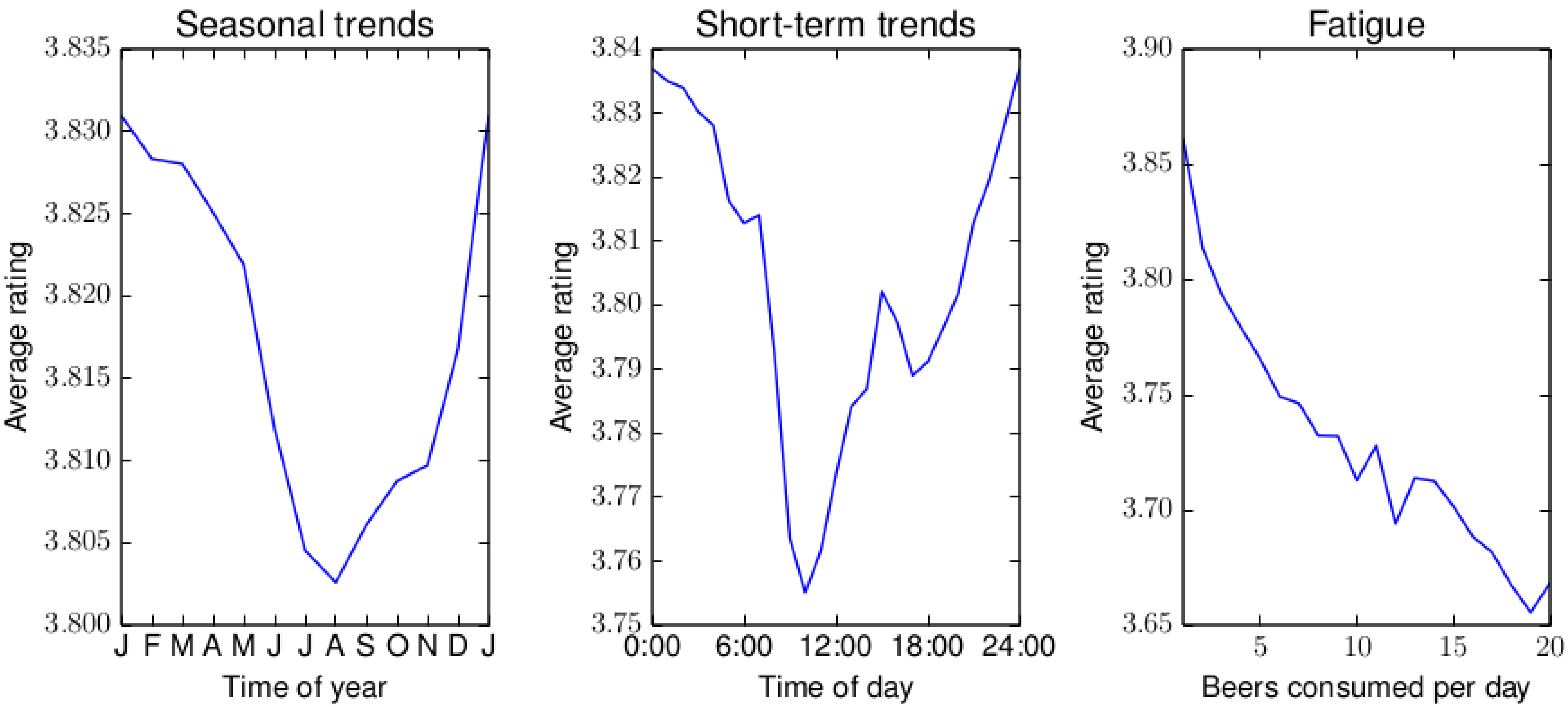


Figure from Koren: "Collaborative Filtering with Temporal Dynamics" (KDD 2009)

Extensions of latent-factor models

3) Change over time



A few temporal effects from beer reviews

Extensions of latent-factor models

3) Change over time

There are a number of reasons why rating data might be subject to temporal effects…

- Changes in the interface
- People give higher ratings to older movies (or, people who watch older movies are a biased sample)
- The community's preferences gradually change over time
- My girlfriend starts using my Netflix account one day
- I binge watch all 144 episodes of buffy one week and then revert to my normal behavior
- I become a "connoisseur" of a certain type of movie
- Anchoring, public perception, seasonal effects, etc.

e.g. "Collaborative filtering with temporal dynamics" Koren, 2009

e.g. "Sequential & temporal dynamics of online opinion" Godes & Silva, 2012

e.g. "Temporal recommendation on graphs via long- and short-term preference fusion" Xiang et al., 2010

e.g. "Modeling the evolution of user expertise through online reviews" McAuley & Leskovec, 2013

Extensions of latent-factor models

3) Change over time

Each definition of temporal evolution demands a slightly different model assumption (we'll see some in more detail later tonight!) but the basic idea is the following:

1) Start with our original model:

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

2) And define some of the parameters as a function of time:

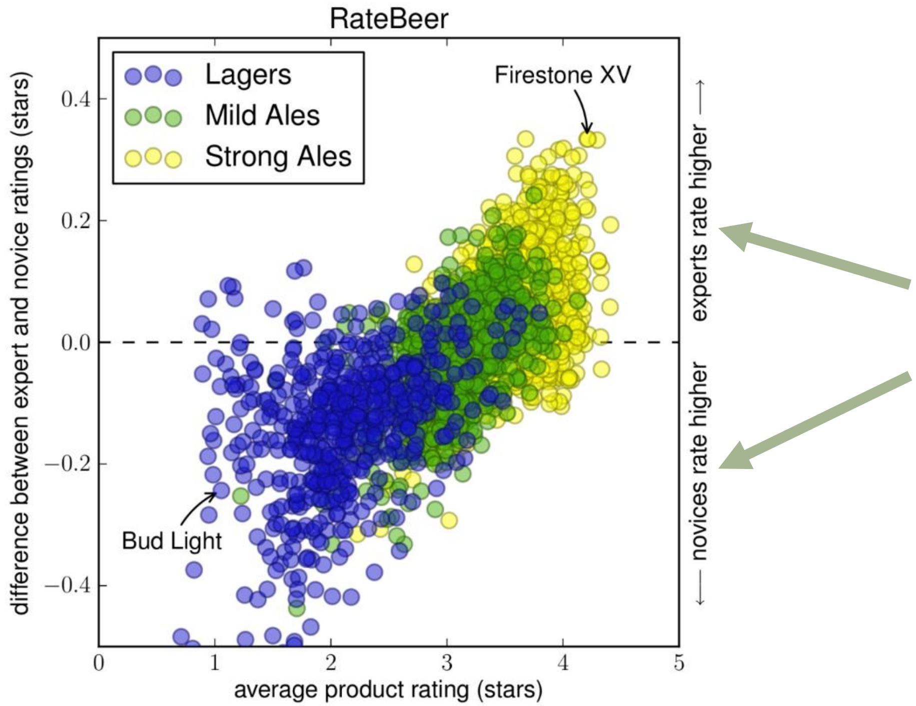
$$f(u,i,t) = \alpha + \beta_u(t) + \beta_i(t) + \gamma_u(t) \cdot \gamma_i$$

3) Add a regularizer to constrain the time-varying terms:

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i,t \in \text{train}} (f(u,i,t) - r_{u,i,t})^2 + \lambda_1 \Omega(\beta,\gamma) + \underbrace{\lambda_2 \|\gamma(t) - \gamma(t+\delta)\|}_{\text{parameters should change smoothly}}$$

3) Change over time

Case study: how do people acquire tastes for beers (and potentially for other things) over time?



Differences between "beginner" and "expert" preferences for different beer styles

Extensions of latent-factor models

4) Missing-not-at-random

- Our decision about whether to purchase a movie (or item etc.) is a function of how we **expect** to rate it
- Even for items we've purchased, our decision to **enter a rating** or write a review **is a function of our rating**
  - e.g. some rating distribution from a few datasets:

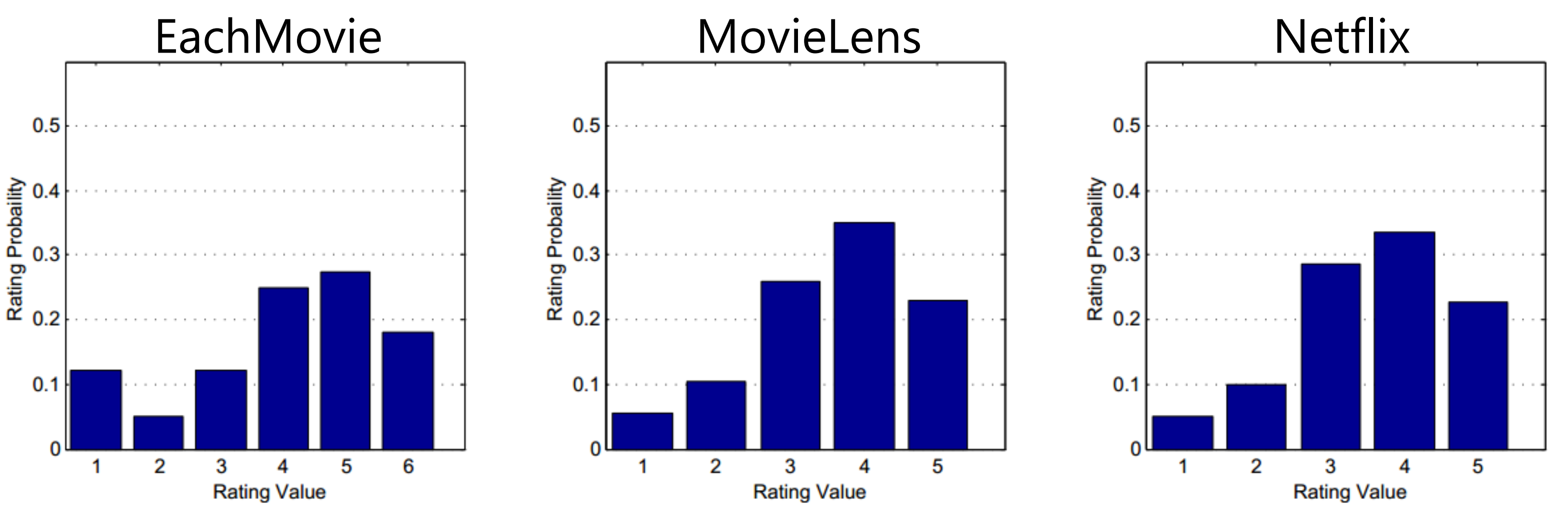
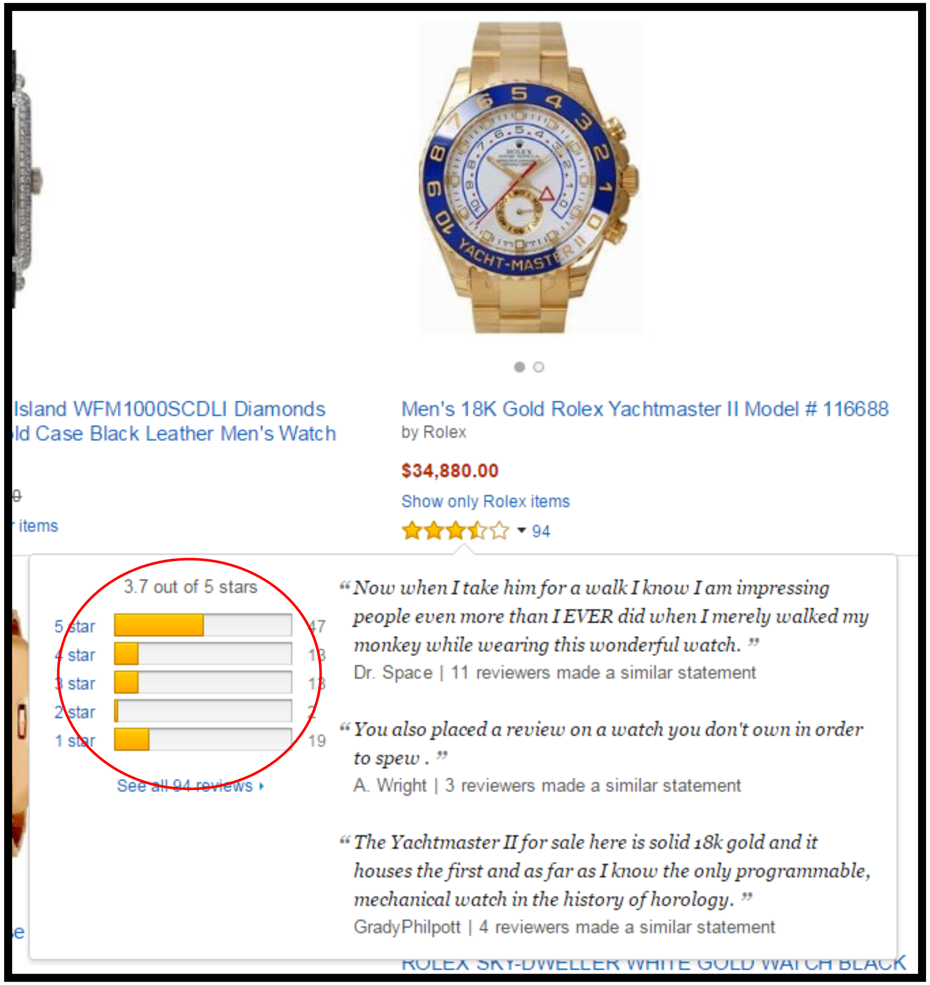


Figure from Marlin et al. "Collaborative Filtering and the Missing at Random Assumption" (UAI 2007)

Extensions of latent-factor models

4) Missing-not-at-random

e.g. Men’s watches:



Extensions of latent-factor models

4) Missing-not-at-random

- Our decision about whether to purchase a movie (or item etc.) is a function of how we **expect** to rate it
- Even for items we've purchased, our decision to **enter a rating** or write a review **is a function of our rating**
- So we can predict ratings more accurately by building models that account for these differences

1. Not-purchased items have a different prior on ratings than purchased ones
2. Purchased-but-not-rated items have a different prior on ratings than rated ones

Figure from Marlin et al. "Collaborative Filtering and the Missing at Random Assumption" (UAI 2007)

Moral(s) of the story

How much do these extension help?

Moral: increasing complexity helps a bit, but changing the model can help **a lot**

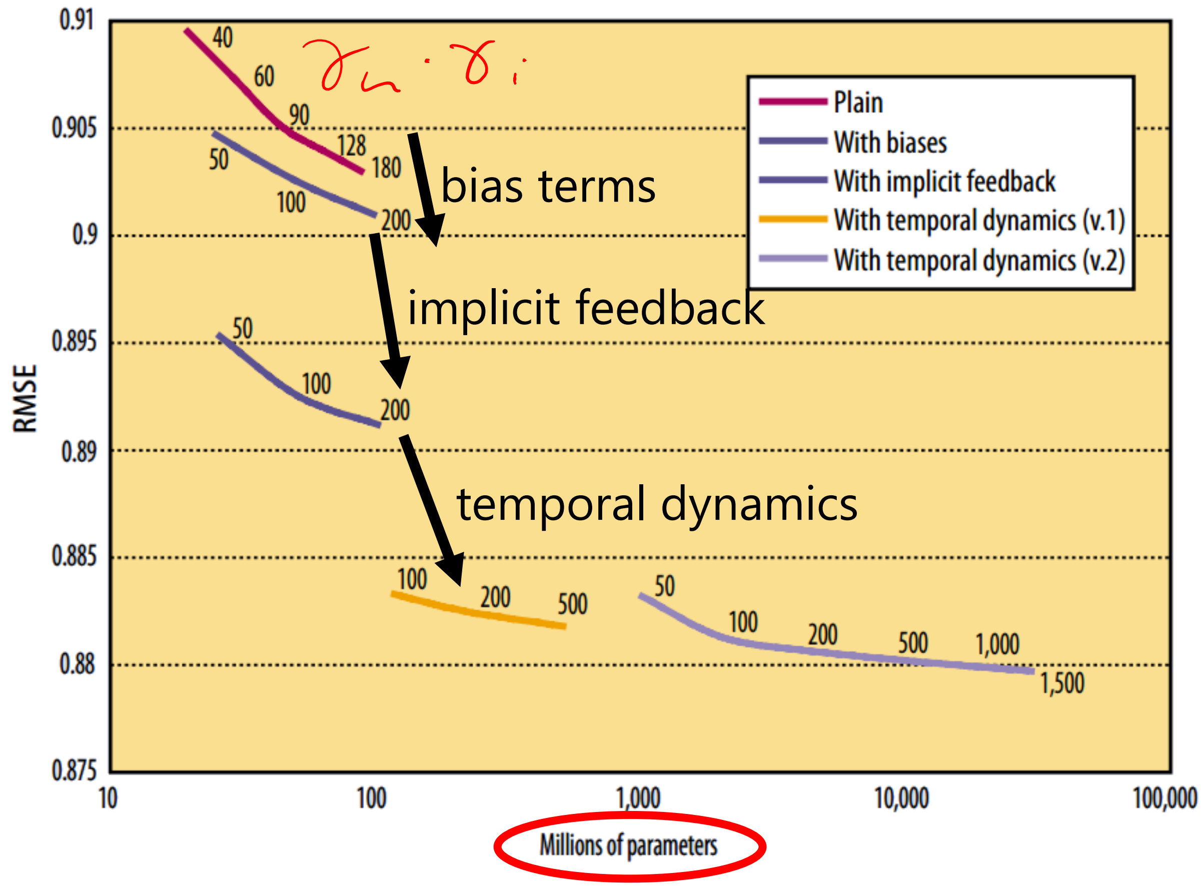


Figure from Koren: "Collaborative Filtering with Temporal Dynamics" (KDD 2009)

Moral(s) of the story

So what actually happened with Netflix?

- The AT&T team “BellKor”, consisting of Yehuda Koren, Robert Bell, and Chris Volinsky were early leaders. Their main insight was how to effectively incorporate temporal dynamics into recommendation on Netflix.
- Before long, it was clear that no one team would build the winning solution, and Frankenstein efforts started to merge. Two frontrunners emerged, “BellKor’s Pragmatic Chaos”, and “The Ensemble”.
- The BellKor team was the first to achieve a 10% improvement in RMSE, putting the competition in “last call” mode. The winner would be decided after 30 days.
- After 30 days, performance was evaluated on the hidden part of the test set.
- Both of the frontrunning teams had **the same** RMSE (up to some precision) but BellKor’s team submitted their solution 20 minutes earlier and won $1,000,000

For a less rough summary, see the Wikipedia page about the Netflix prize, and the nytimes article about the competition: http://goo.gl/WNpy7o

Moral(s) of the story

Afterword

- Netflix had a class-action lawsuit filed against them after somebody de-anonymized the competition data
- $1,000,000 seems to be **incredibly cheap** for a company the size of Netflix in terms of the amount of research that was devoted to the task, and the potential benefit to Netflix of having their recommendation algorithm improved by 10%
- Other similar competitions have emerged, such as the Heritage Health Prize ($3,000,000 to predict the length of future hospital visits)

- But… the winning solution never made it into production at Netflix – it's a monolithic algorithm that is very expensive to update as new data comes in*

*source: a friend of mine told me and I have no actual evidence of this claim

Moral(s) of the story

Finally…

Q: Is the RMSE really the right approach? Will improving rating prediction by 10% actually improve the user experience by a significant amount?
A: Not clear. Even a solution that only changes the RMSE slightly could drastically change which items are top-ranked and ultimately suggested to the user.
Q: But… are the following recommendations actually any good?
A1: Yes, these are my favorite movies!
or **A2:** No! There's no **diversity**, so how will I discover **new** content?

5.0 stars | 5.0 stars | 5.0 stars | 5.0 stars | 4.9 stars | 4.9 stars | 4.8 stars | 4.8 stars

predicted rating

Summary

Various extensions of latent factor models:

- Incorporating features

e.g. for cold-start recommendation

- Implicit feedback

e.g. when ratings aren't available, but other actions are

- Incorporating temporal information into latent factor models

seasonal effects, short-term "bursts", long-term trends, etc.

- Missing-not-at-random

incorporating priors about items that were not bought or rated

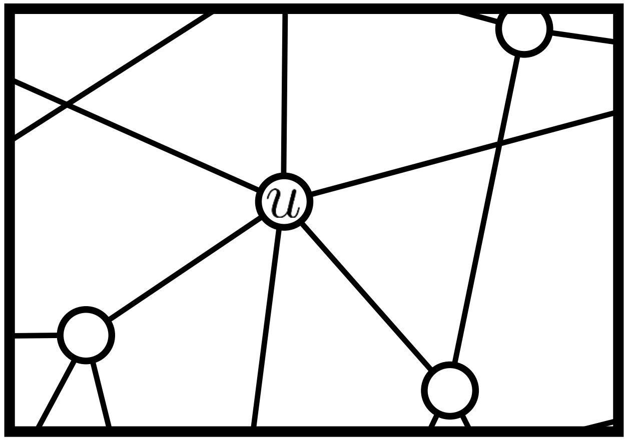
- The Netflix prize

Things I didn't get to…

Socially regularized recommender systems

see e.g. "Recommender Systems with Social Regularization"

http://research.microsoft.com/en-us/um/people/denzho/papers/rsr.pdf



network

$$\arg\min_{\alpha,\beta,\gamma} \sum_{u,i\in\text{train}} (f(u,i) - r_{u,i})^2 + \lambda_1 \Omega(\beta,\gamma) + \lambda_2 \sum_{u,v\in\mathcal{E}} \|\gamma_u - \gamma_v\|$$

social regularizer

Questions?

Further reading:

Yehuda Koren's, Robert Bell, and Chris Volinsky's IEEE computer article:
http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf
Paper about the "Missing-at-Random" assumption, and how to address it:
http://www.cs.toronto.edu/~marlin/research/papers/cfmar-uai2007.pdf
Collaborative filtering with temporal dynamics:
http://research.yahoo.com/files/kdd-fp074-koren.pdf
Recommender systems and sales diversity:
http://papers.ssrn.com/sol3/papers.cfm?abstract_id=955984