# Matrix Factorization

**Recommender Systems**

Dimitris Sacharidis

# Collaborative Filtering

- **Memory-based Methods** = remember the entire *history* of user-item interactions
  - User-User Neighborhood CF
  - Item-Item Neighborhood CF

- **Model-based Methods** = build a model that *describes* the history
  - Matrix Factorization
  - Other Model-based Methods

# model-based collaborative filtering

- **Matrix Factorization**
  - **motivation**
  - **Singular Value Decomposition (SVD)**
  - **Matrix Factorization**
  - **adding baseline estimates**
  - **adding other data sources**
- **Netflix prize**
- other Model-based methods

# motivation

# motivation

- ratings matrix is a **very detailed representation** of the preferences of users over items

- but we know that user preferences are **not overly complex**
  - can be described more compactly
  - e.g., I like Steven Spielberg and sci-fi movies, and actor Harrison Ford

- goal is to find a **representation** of user preferences that is:
  - **compact**
  - **explains the observed user behavior** to a large extent

# motivation

Benefits of a compact representation

- **efficiency**
  - faster recommendations (compared to neighborhood-based CF methods)
- **effectiveness**
  - naturally removes noisy user-item interactions
  - and focuses on the general trends

# build upon known ideas

- there is a standard method to compress a matrix, called **Singular Value Decomposition** (SVD)

- first, we overview SVD
- and then adapt it to compress the ratings matrix

# Singular Value Decomposition

# Singular Value Decomposition

- SVD is a method to **factorize** a matrix
- just as we can factorize a natural number into a *product of factors*
  - 24 = 2*3*4
- we can factorize a matrix into **a product of more elementary matrices**

# Singular Value Decomposition

- SVD factorizes an $m \times n$ matrix $M$ into three matrices:
- $U$ that is $m \times m$ orthogonal (its transpose is its inverse)
- $V$ that is $n \times n$ orthogonal
- $\Sigma$ that is $m \times n$ diagonal (non-zero values only at its diagonal)
    - its values are called *singular* and there are at most $\min\{m, n\}$

# Truncated SVD

- SVD has a nice property: it gives a **low-rank matrix approximation**:
- if we keep the $k$ largest singular values and set the rest to zero
- then, resulting matrix $\hat{M}$ is the **best approximation** (of rank $k$ under squared error) to the original $M$

$$\underset{m \times n}{M} = \underset{m \times m}{U} \cdot \underset{m \times n}{\Sigma} \cdot \underset{n \times n}{V^\intercal}$$

# SVD on ratings matrix

- apply truncated SVD on the *complete* ratings matrix $R$
- $k$ represents the num of **features** (latent/hidden variables) that describe users' preferences
  - e.g., comedy vs. drama, or woman vs man in main roles
- matrix $\hat{\Sigma}$ conveys the **significance** of features
- matrix $\hat{U}$ captures the **users' interests**
- matrix $\hat{V}$ captures the **items' descriptions**

$$\underset{m \times n}{\hat{R}} = \underset{m \times k}{\hat{U}} \cdot \underset{k \times k}{\hat{\Sigma}} \cdot \underset{k \times n}{\hat{V}^{\mathsf{T}}}$$

# approximate ratings

- the approximate/predicted rating of user $u$ to item $i$ is:

$\hat{R}$ ($m \times n$), $\hat{r}_{ui}$ = $\hat{U}$ ($m \times k$) · $\hat{\Sigma}$ ($k \times k$) · $\hat{V}^\intercal$ ($k \times n$)

$$\hat{r}_{ui} = \sum_f \hat{U}_{uf} \cdot \hat{\Sigma}_{ff} \cdot \hat{V}_{if}$$

- sum over all features
- for each feature, multiply the user's interest $\hat{U}_{uf}$ with the item's description $\hat{V}_{if}$ and scale by feature significance $\hat{\Sigma}_{ff}$

# issue with SVD on ratings matrix

- SVD assumes the **complete** matrix is known
    - no missing values
- not our case, where the **sparse ratings matrix** has unknown ratings

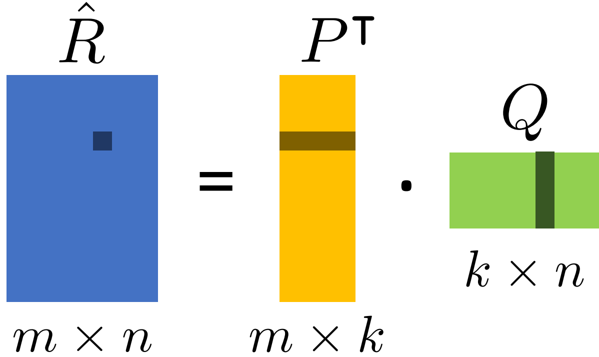
- what can we do?
- *old approach*: impute (fill in) the missing ratings and then do SVD
- *better approach*: focus on the end result (features of users and items) and **try to approximate only what we know**

# Matrix Factorization

# from SVD to matrix factorization

- **get rid of the significance diagonal matrix**
  - include its contribution to the user and item matrices
  - has some implications: not longer an SVD

- goal is to find a **user feature matrix** $P$ and an **item feature matrix** $Q$




- a column vector $p_u$ of $P$ represents the interest of user $u$
- a column vector $q_i$ of $Q$ represents the description of item $i$

- their inner product is the **predicted rating** $\hat{r}_{ui} = p_u^\intercal q_i$

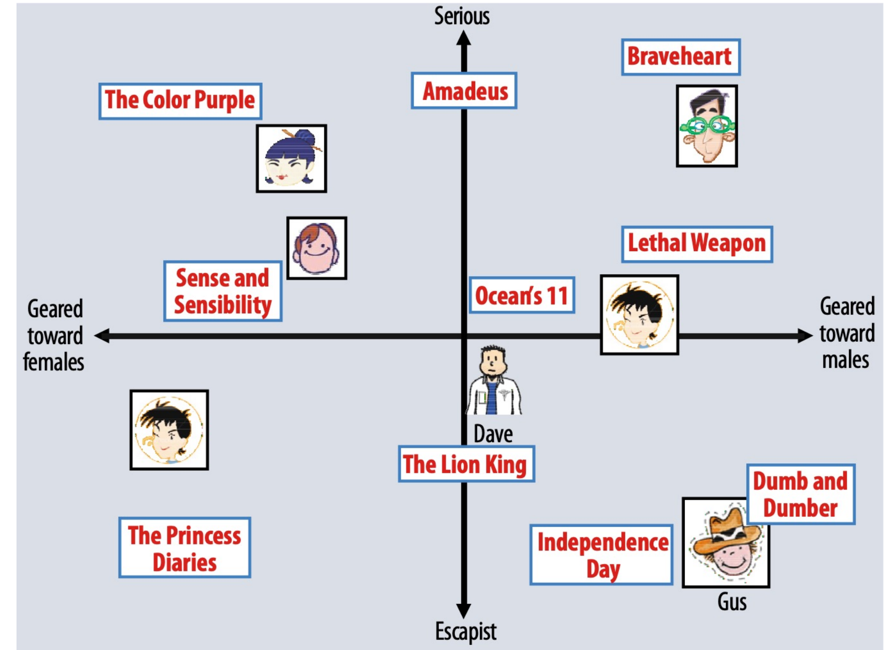
# latent feature space

- users and items are described by vectors of features
- live in a shared vector space

- assume two features: *male vs female*, and *serious vs escapist*

- inner product expresses distance in this feature space

- **large inner product between user and item =** better match = **high predicted rating**




[2009 IEEE Y. Koren et al] *Matrix Factorization Techniques for Recommender Systems*

# matrix factorization goal

- goal is to find a **user feature matrix** $P$ and an **item feature matrix** $Q$
- so the rating of user $u$ to item $i$ is predicted as:

$$\hat{r}_{ui} = p_u^\intercal q_i$$

- let's treat this as a **machine learning** task:
- matrices $P, Q$ are the *parameters* that we need to learn so that
    - G1: predictions on **known ratings** (train data) are **accurate**
    - G2: they **generalize** well to **unknown ratings** (test data)

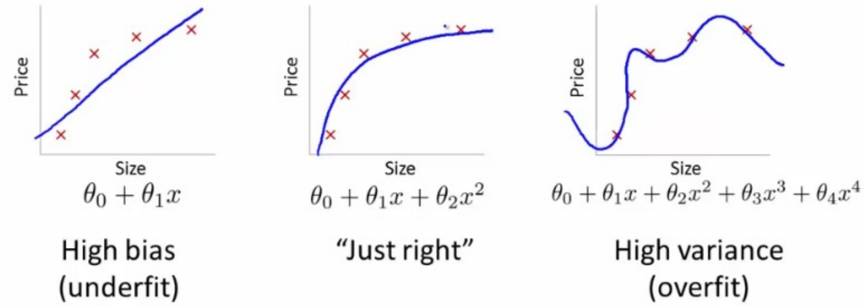
# G1: accurate predictions for known ratings

- consider a **known rating** $r_{ui} \in R$
- the **predicted rating** for the same user-item pair is $\hat{r}_{ui} = p_u^{\mathsf{T}} q_i$
- therefore the **prediction error** is $e_{ui} = r_{ui} - p_u^{\mathsf{T}} q_i$
- typically, we want to minimize the **squared error** $(r_{ui} - p_u^{\mathsf{T}} q_i)^2$
  - square penalizes larger deviations more
- and compute a total error over all known ratings, the **mean squared error** (MSE):

$$\frac{1}{|R|} \sum_{r_{ui} \in R} (r_{ui} - p_u^{\mathsf{T}} q_i)^2$$

# G2: generalize well to unknown ratings

- problem of **overfitting**:
  - learn matrices that minimize MSE very well, i.e., excellent accuracy for known ratings
  - but has terrible accuracy for unknown ratings, i.e., in real use




Andrew Ng, *Machine Learning* (https://www.coursera.org/learn/machine-learning/)

- standard solution is **regularization**:
  - require parameters (matrix values) to have small magnitude
  - measured by the square of each parameter (L2 regularization)
  - its strength is controlled by (a hyperparameter) $\lambda$

- thus also minimize: $\lambda(\|P\|^2 + \|Q\|^2)$ $= \lambda \left( \sum_u \sum_f p_{uf}^2 + \sum_i \sum_f q_{if}^2 \right)$

(a.k.a. Frobenius norm of a matrix)

# putting it all together

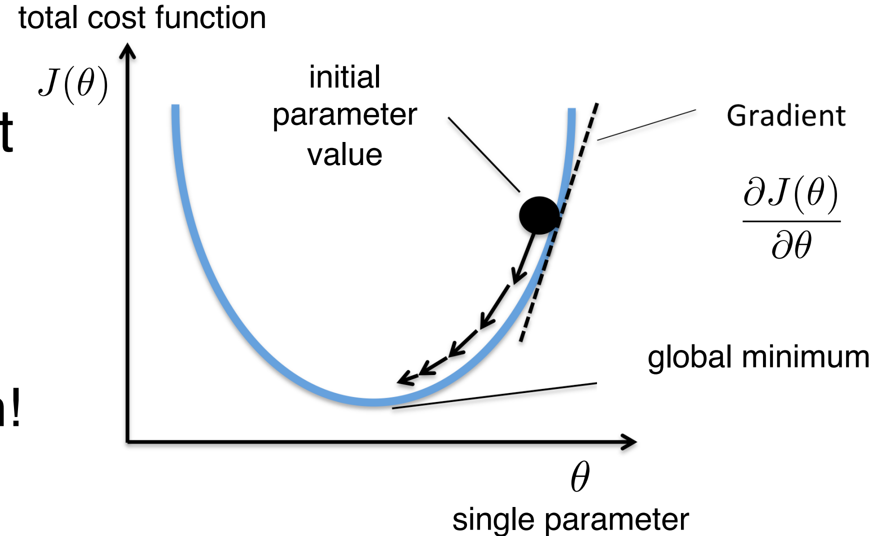
- goal is to find matrices $P, Q$ that minimize the **total cost function**:

$$J = \frac{1}{|R|} \sum_{r_{ui} \in R} (r_{ui} - p_u^\intercal q_i)^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

- there are several ways to solve the above (non-convex) optimization problem
- here we present **Stochastic Gradient Descent** (SGD)
- another option is **Alternating Least Squares** (ALS)

# background: gradient descent

- **gradient** = derivative = *slope* of function at a particular point
- specifies the **direction** of maximum increase
  - to minimize take a step towards the opposite direction!




**gradient descent**

- initialize random values for each parameter $\theta$
- repeat until convergence:
  - update each parameter by $\theta \leftarrow \theta - \eta \cdot \frac{\partial J(\theta)}{\partial \theta}$

$\eta$ is the **learning rate**: controls the step size

# background: stochastic gradient descent

- instead of computing the gradient of the **total cost function**, i.e., over all $N$ training examples (= ratings)
- **stochastic gradient descent** (SGD) computes the gradient **over one training example** (or a small number thereof)

$$J(\theta) = \frac{1}{N} \sum J^{(i)}(\theta)$$

**total cost function**

**cost function** of one example

**stochastic gradient descent**

- randomly shuffle training examples
- initialize random values for each parameter
- repeat until convergence:
  - for each training example
    - update each parameter by $\theta \leftarrow \theta - \eta \cdot \frac{\partial J^{(i)}(\theta)}{\partial \theta}$

# SGD for matrix factorization

- parameters are **user** and **item features** $\theta = \{P, Q\}$
- **total cost function** $J(\theta) = \frac{1}{|R|} \sum_{r_{ui} \in R} e_{ui}^2 + \lambda(\|P\|^2 + \|Q\|^2)$
  - where $e_{ui} = r_{ui} - p_u^\intercal q_i$
- we want to **rewrite** this as an **average of costs** per rating:

$$J(\theta) = \frac{1}{|R|} \sum_{r_{ui} \in R} J^{(u,i)}(\theta)$$

- therefore the **cost function** of one rating looks like:

$$J^{(u,i)} = e_{ui}^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

# SGD for matrix factorization

- compute gradients of **cost function**

$$J^{(u,i)} = e_{ui}^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

- with respect to **user features** $\frac{\partial J^{(u,i)}}{\partial p_u} = -2e_{ui} \cdot q_i + 2\lambda \cdot p_u$
- with respect to **item features** $\frac{\partial J^{(u,i)}}{\partial q_i} = -2e_{ui} \cdot p_u + 2\lambda \cdot q_i$
  - where $p_u$ is the **feature vector of user** $u$
  - and $q_i$ is the **feature vector of item** $i$

# SGD for matrix factorization

randomly shuffle ratings $R$
randomly initialize matrices $P$, $Q$
**repeat**
  **foreach** $r_{ui} \in R$ **do**
    $e_{ui} \leftarrow r_{ui} - p_u^\intercal q_i$
    $p_u \leftarrow p_u + \eta \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$
    $q_i \leftarrow q_i + \eta \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
**until** convergence

*remember:*

$$\theta \leftarrow \theta - \eta \cdot \frac{\partial J^{(i)}(\theta)}{\partial \theta}$$

*(the term 2 in the gradients is absorbed by the learning rate)*

# adding baseline estimates

# rating deviations from average

- recall in neighborhood-based collaborative filtering, we were predicting *rating deviations*
  - from the user's average rating in U-U
  - from the item's average rating in I-I

- essentially these average ratings were acting as a **baseline estimate** of any missing rating

- let's revisit this idea…

# baseline estimate

- let's now assume a better baseline estimate for the unknown rating of user $u$ to item $i$

$$b_{ui} = \mu + b_i + b_u$$

- $\mu$ is the **overall average rating** (among all ratings)
- $b_u$ is the **bias in ratings from user** $u$
- $b_i$ is the **bias in ratings for item** $i$

E.g.: what is the rating of Titanic by Joe?

- average rating over all movies is **3.7**
- Joe is a critical user, tends to rate **0.3** lower than the average user
- Titanic is good movie, tends to get rated **0.5** above the average movie
- so the baseline estimate of Joe's Titanic rating is **3.9** = 3.7 - 0.3 + 0.5

# improved rating prediction

- new rating prediction formula:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^\intercal q_i$$

- adapt algorithm to also learn user and item biases, besides user and item features
  - additional update formulas for SGD:

$$b_u \leftarrow b_u + \eta \cdot (e_{ui} - \lambda \cdot b_u)$$
$$b_i \leftarrow b_i + \eta \cdot (e_{ui} - \lambda \cdot b_i)$$

# adding other data sources

# implicit feedback

- information about items the user has **clicked**, liked, purchased, etc.
  - assume Boolean case, let $N(u)$ be the set of items user $u$ has clicked

- intuition: a **user’s preferences** are also conveyed by her/his clicks

- introduce additional features for items
  - *k*-dimensional vector $y_i$ for item $i$

- user’s preferences from implicit feedback are captured by: $\sum_{j \in N(u)} y_j$

# putting it all together

- so a user's preferences are represented by a **latent user feature vector**
- *and* the sum of **latent item feature vectors** for the items s/he has clicked

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^{\top} \left( p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$

*baseline estimate* ($\mu + b_u + b_i$)   *item model* ($q_i^{\top}$)   *user model* ($p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$)

- the term $|N(u)|^{-\frac{1}{2}}$ is just normalizing the sum of latent item feature vectors
- this model is called **SVD++** in the literature

[2008 KDD Y. Koren] *Factorization meets the Neighborhood: a Multifaceted CF Model*

# additional user features

- the matrix factorization model is quite flexible
- can easily integrate additional data sources (or signals)
- we have seen how to do this with implicit feedback…
- other option is **known user attributes**, such as demographics

- how to do it?
  - for each user attribute value, e.g., age group, introduce a feature vector
  - then add these vector to the user model part (shown blue in the previous slide)

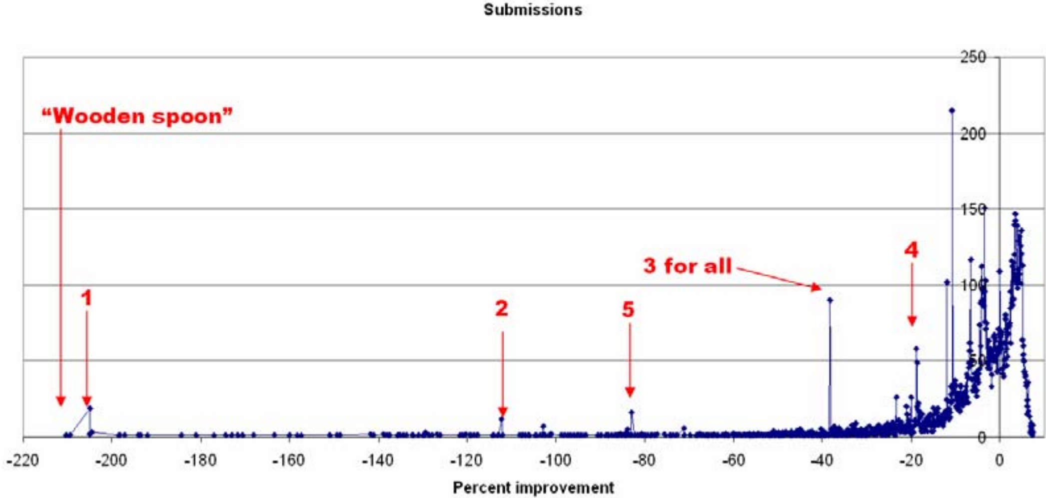
- **factorization machines** (reviewed later) generalize such ideas

# Netflix prize

# the rules

- https://www.netflixprize.com/rules.html
  - started on October 2, 2006; ended on September 21, 2009
- goal was to **improve** the movie recommendation algorithm of Netflix, called Cinematch,
  - based on Item-Item CF with Pearson corr. coefficient
- **by 10%** in terms of mean squared prediction error (actually **RMSE**)
- winner would receive the **Grand Prize** of $1M
  - each year the leading team would receive a **Progress Prize** of $50K
- dataset of 100M ratings from 480K users on 18K movies, split into
  - **training** set, which includes a known **probe** set
  - and **qualifying** set, which is further split into **quiz** and **test** sets

# early progress

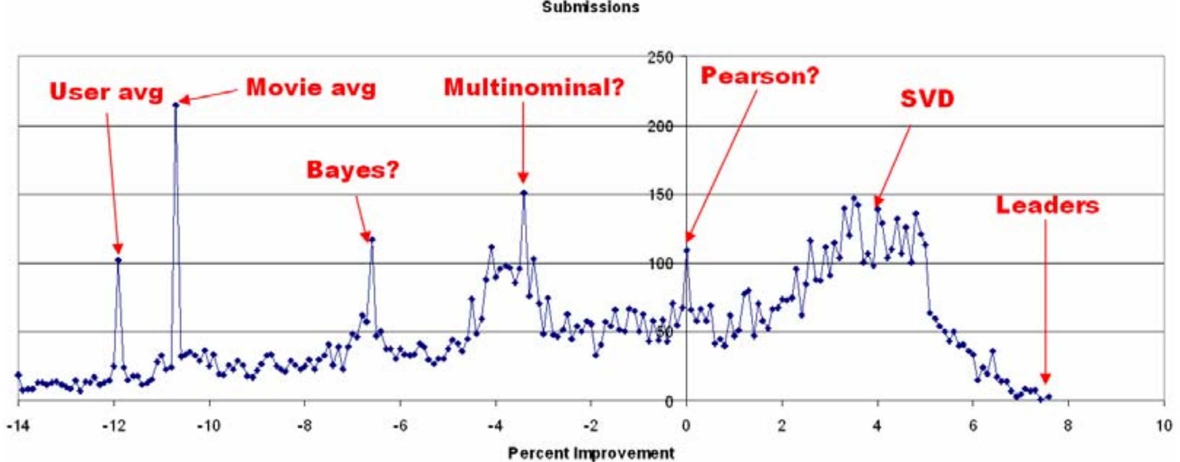



around summer of 2007, just before the 1st progress prize

- with red are Netflix's guesses for methods!

[2007 KDD J. Bennett et al.] *The Netflix Prize*

zoom-in at the leaders



# simon funk's SVD

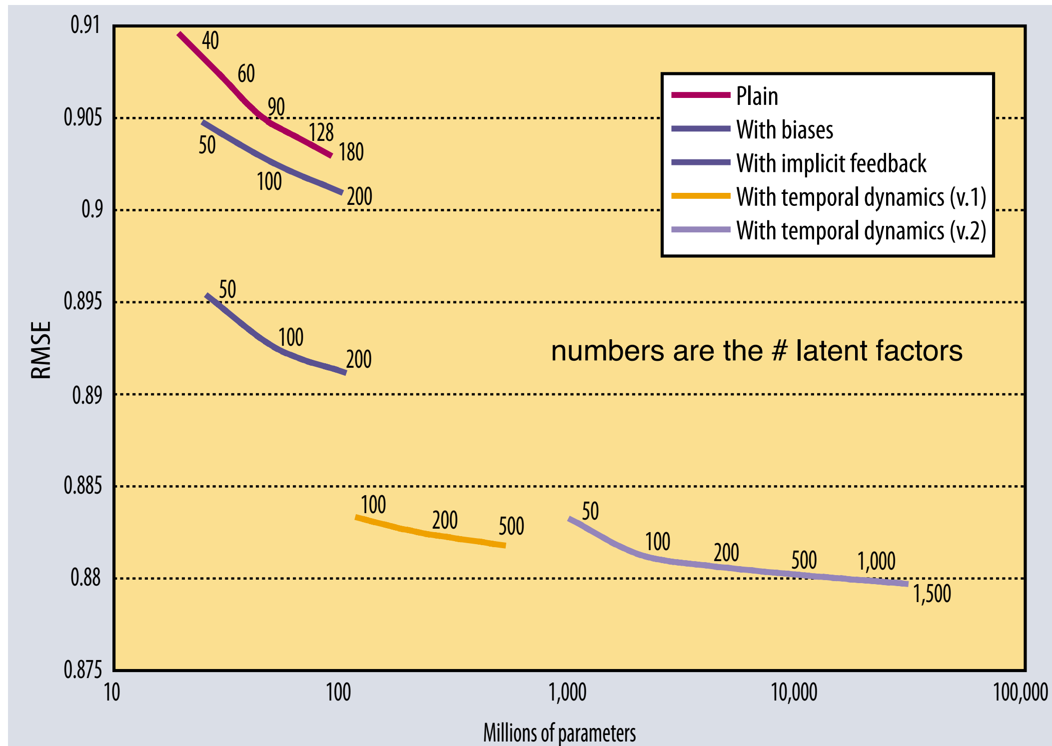
- lots of good submissions are doing something like SVD, how come?

- very early on (~ October 2006), Brandyn Webb (aka **Simon Funk**) started blogging about his progress together with Vincent DiCarlo to run SVD on the dataset
  - http://sifter.org/simon/journal/20061211.html

- he **publicly documented** in great detail essentially a practical version of the basic matrix factorization method we covered
  - which came to be known by funk's SVD

# towards the grand prize

- improving the basic MF, one step at a time
  - biases, implicit feedback, temporal dynamics




- Cinematch has RMSE 0.9514
- mean rating of a movie has RMSE 1.053
- for the Grand Prize you need RMSE 0.8563

[2009 IEEE Y. Koren et al] *Matrix Factorization Techniques for Recommender Systems*

# the grand prize

team BellKor (Koren, Bell, Volinsky) won the **2007 Progress Prize** with 8.43% improvement

later joined forces with Austrian team Big Chaos to win the **2008 Progress Prize** with 9.46% improvement

in the end, after Pragmatic Theory joined the two other teams, they won the **Grand Prize with 10.06% improvement**

the winning solution was a **blend** of multiple recommenders – not a single method is good enough

# the grand prize

- further reading :
  - https://www.wired.com/2009/09/bellkors-pragmatic-chaos-wins-1-million-netflix-prize/
  - https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf
  - https://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf
  - https://www.netflixprize.com/assets/GrandPrize2009_BPC_PragmaticTheory.pdf