IntroProp Mini-project:

Recommendation System

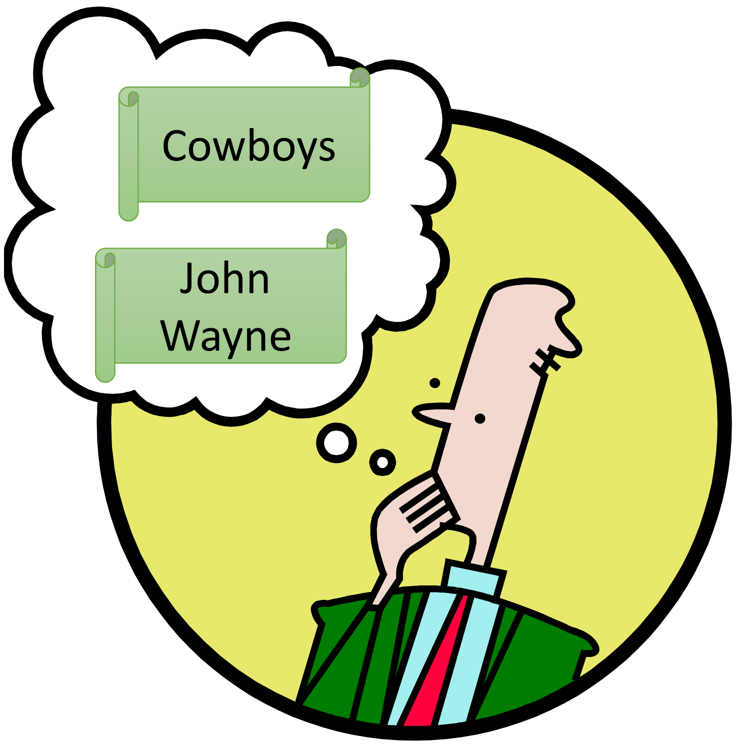
Barbara Jobstmann

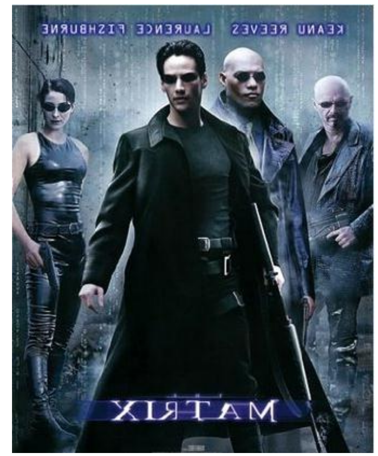
Oct 23, 2014

Outline

- Recall from last week:
  - Feature Vector/Matrices U and V
  - Update an element in U or V
- A complete UV-Decomposition
  - Optimization and Stopping criteria
  - Issues with local minima and Initialization
- Evaluation of (your) recommendations
- Netflix Challenge
- Tasks
  - optimizeU/optimizeV
  - recommend

Feature Vector of Item or User





Sci-Fi

Keanu Reeves

Item Feature Vector

	Matrix
Sci-Fi	5
Reeves	5
Cowboys	0
Wayne	0

Properties of item expressed in terms of a set of features

User Feature Vector

	Sci-Fi	Reeves	Cowboys	Wayne
Joe	0	0	4	5

Preference of user expressed in terms of a set of features

Feature Matrices U and V

- User Profile Matrix $U_{n\times d}$: **n users and d features**

	Sci-Fi	Reeves	Cowboys	Wayne	…
Joe	0	0	4	5	…
User 1	…	…	…	…	…
…	…	…	…	…	…

- Item Profile Matrix $V_{d\times m}$: **d features, m items**

	Matrix	Item 1	Item 2	Item 3	…
Sci-Fi	5	…	…	…	
Reeves	5	…	…	…	
Cowboys	0	…	…	…	
Wayne	0	…	…	…	

u(Joe, Matrix) = $\begin{pmatrix} 0 & 0 & 4 & 5 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 5 \\ 0 \\ 0 \end{pmatrix}$

- Idea: Rating ≈ User Profile · Item Profile
  - Entry (i,j) in Utility Matrix is rating of user i of item j

Dimensionality Reduction Systems

- Recommend items based on the conjecture that the utility matrix is actually a product of two long, thin matrices U and V.
  - Matrix $U_{n \times d}$ mapping users to features
  - Matrix $V_{d \times m}$ mapping features to items
- **Key idea**: $M_{n \times m} \approx U_{n \times d} \cdot V_{d \times m}$
- **Goal**: find matrices $U_{n \times d}$ and $V_{d \times m}$ (given $M_{n \times m}$ and d) such that their product $P_{n \times m} = U_{n \times d} \cdot V_{d \times m}$ is **similar** to M on all **non-zero** entries (actual rating).
- **One approach: UV-decomposition algorithm** (instance of a more general theory called SVD (singular-value decomposition)

UV-decomposition

$$M_{n\times m} \overset{\textbf{RMSE}}{\approx} U_{n\times d} \cdot V_{d\times m}$$

$$\overbrace{\begin{pmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{pmatrix}}^{\text{Number of Items (5)}} \approx \overbrace{\begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{pmatrix}}^{\text{Number of features (2)}} \cdot \overbrace{\begin{pmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{pmatrix}}^{\text{Number of Items (5)}}$$

Number of features (2)

Number of Users (5) Number of Users (5)

$$m_{ij} \ldots element\ in\ M$$

$$u_{ij} \ldots element\ in\ U, v_{ij} \ldots element\ in\ V$$

$$p_{ij} \ldots element\ in\ P = U \cdot V$$

UV-Decomposition: Iterative Approach

- Start with arbitrary matrices U and V
- Modify U and V locally to improve RMSE
- Local improvement means, e.g., one element
- **Question**: how does one element of U (or V) contribute to the error?

Adjusting Arbitrary Element in U and V

- To update element u_{rs} use

$$\overline{u}_{rs} = \frac{\sum_j v_{sj} \cdot \left(m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj}\right)}{\sum_j {v_{sj}}^2}$$

$\sum_j$ stands for the sum over all j s.t. m_{rj} is nonblank

- To update element v_{rs} use

$$\overline{v}_{rs} = \frac{\sum_i u_{ir} \cdot (m_{is} - \sum_{k \neq r} u_{ik} \cdot v_{ks})}{\sum_i {u_{ir}}^2}$$

$\sum_i$ stands for the sum over all i s.t. m_{is} is nonblank

Example : Adjusting Element in U

$$\begin{pmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & \boxed{} & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ x & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ x+1 & x+1 & x+1 & x+1 & x+1 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$$

$\overline{u}_{rs} = \frac{\sum_j v_{sj} \cdot (m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj})}{\sum_j {v_{sj}}^2}, j = 0, \ldots, 4$ s.t. m_{rj} is nonblank, $\mathrm{k} = 0{,}1$ s.t. $\mathrm{k} \neq s$

- r=2, s=0 (recall: it effects only row 2 = 3rd row):
  - j=0: $v_{00} \cdot (m_{20} - u_{21} \cdot v_{10})$ = 1 · (2 – 1 · 1) = 1
  - j=1: m_{21} is blank => 0
  - j=2: $v_{02} \cdot (m_{22} - u_{21} \cdot v_{12})$ = 1 · (3 – 1 · 1) = 2
  - j=3: 1 · (1 – 1 · 1) = 0
  - j=4: 1 · (4 – 1 · 1) = 3
- $\sum_j {v_{sj}}^2 = \sum_j {v_{0j}}^2 = 4$ (because m_{21} is blank)
- $\overline{u}_{rs}$ = (1+2+0+3)/4 = 1.5

Example : Adjusting Element in V

$$\begin{pmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & x \end{pmatrix} = \begin{pmatrix} 2 & 2 & 2 & 2 & x+1 \\ 2 & 2 & 2 & 2 & x+1 \\ 2 & 2 & 2 & 2 & x+1 \\ 2 & 2 & 2 & 2 & x+1 \\ 2 & 2 & 2 & 2 & x+1 \end{pmatrix}$$

$\overline{v}_{rs} = \frac{\sum_i u_{ir} \cdot (m_{is} - \sum_{k \neq r} u_{ik} \cdot v_{ks})}{\sum_i {u_{ir}}^2}$, $i = 0, \ldots, 4$ s.t. m_{is} is nonblank, $k = 0{,}1$ s.t. $k \neq$r

- r=1, s=4:
  - i=0: $u_{01} \cdot (m_{04} - u_{00} \cdot v_{04})$ = 1 · (3 – 1 · 1) = 2
  - i=1: 1 · (1 – 1 · 1) = 0; i=2: 1 · (4 – 1 · 1) = 3; i=3: 1 · (5 – 1 · 1) = 4
- $\sum_i {u_{i1}}^2$= 4 (because m_{44} is blank)
- $\overline{v}_{rs}$ = (2+0+3+4)/4 = 2.25

What happens if all entries in last column of M are empty?

A Complete UV-Decomposition

Optimization

Stopping criteria

Issues with local minima

Initialization

Optimization

- Aim: minimize RMSE
- Update entries in U and V to improve RMSE
- To reach a (local) minimum from given U and V
  - Pick order in which to update elements, e.g., row-by-row or randomly, we just need to make sure that every element if updated in a round

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

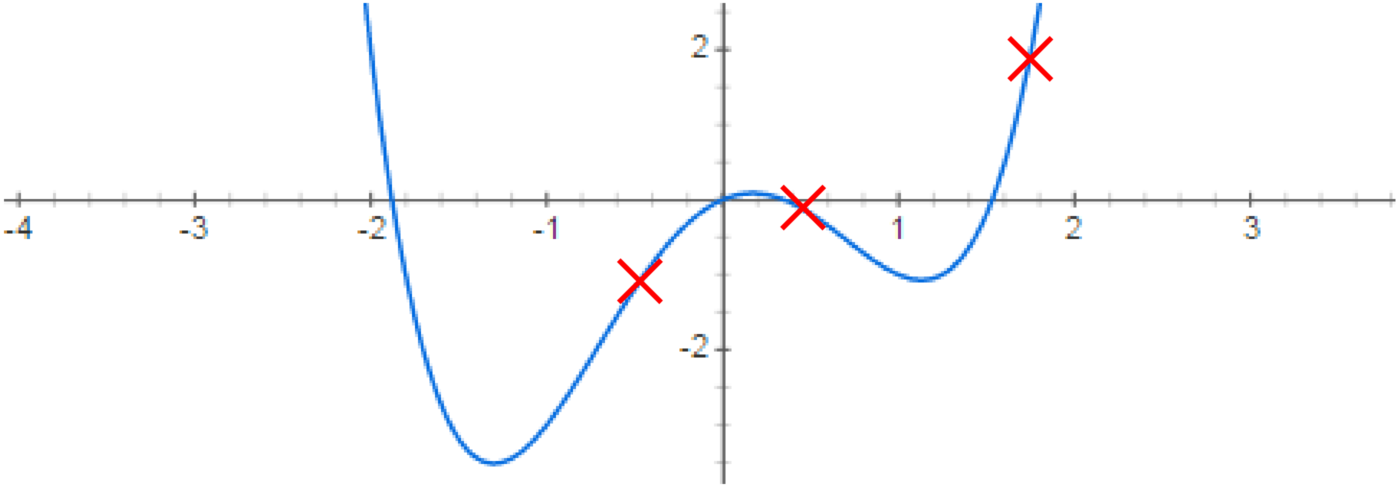
- Note: updating an element once does not mean we cannot find a better value $\Rightarrow$ we might need **many rounds**

Stopping Criteria: Converging to a Minimum

- Ideally: RMSE = 0 at some point
- In practice: unlike to reach RMSE = 0
- Need a way to detect when there is little benefit to revisiting elements in U and/or V
- **Idea: track the amount of improvement, i.e., change of RMSE, and stop is below a threshold**
- Options:
  - check improvement per element or
  - check improvement per round

Issue with Local Minima

- Recall: local improvement (one variable)
- **Local minima** – matrices U and V such that no allowable adjustment reduces the RMSE
- **Global minimum** – the matrices U and V that produce the least possible RMSE
- **Use different starting points to increase changes to reach global minimum**



Starting Points

- Simple starting point: all elements have the same value v
- **Good choice**: a value v that give elements in $P = U \cdot V$ the average of nonblank entries in M
  - Given $M_{n\times m}, U_{n\times d}, V_{d\times m}$, let a be the average of all nonblank entries of M, i.e., the sum of nonblank entries in M divided by the number of nonblank entries:
  - Note we aim for $p_{rs} = a$, recall that

$$p_{rs} = \sum_{k=0}^{d-1} u_{rk} \cdot v_{ks} = \sum_{k=0}^{d-1} v \cdot v = d \cdot v^2 = a$$

  - So, $v = \sqrt{a/d}$
- Multiple starting point? Perturb value v randomly (e.g., +/- some c)

Evaluation

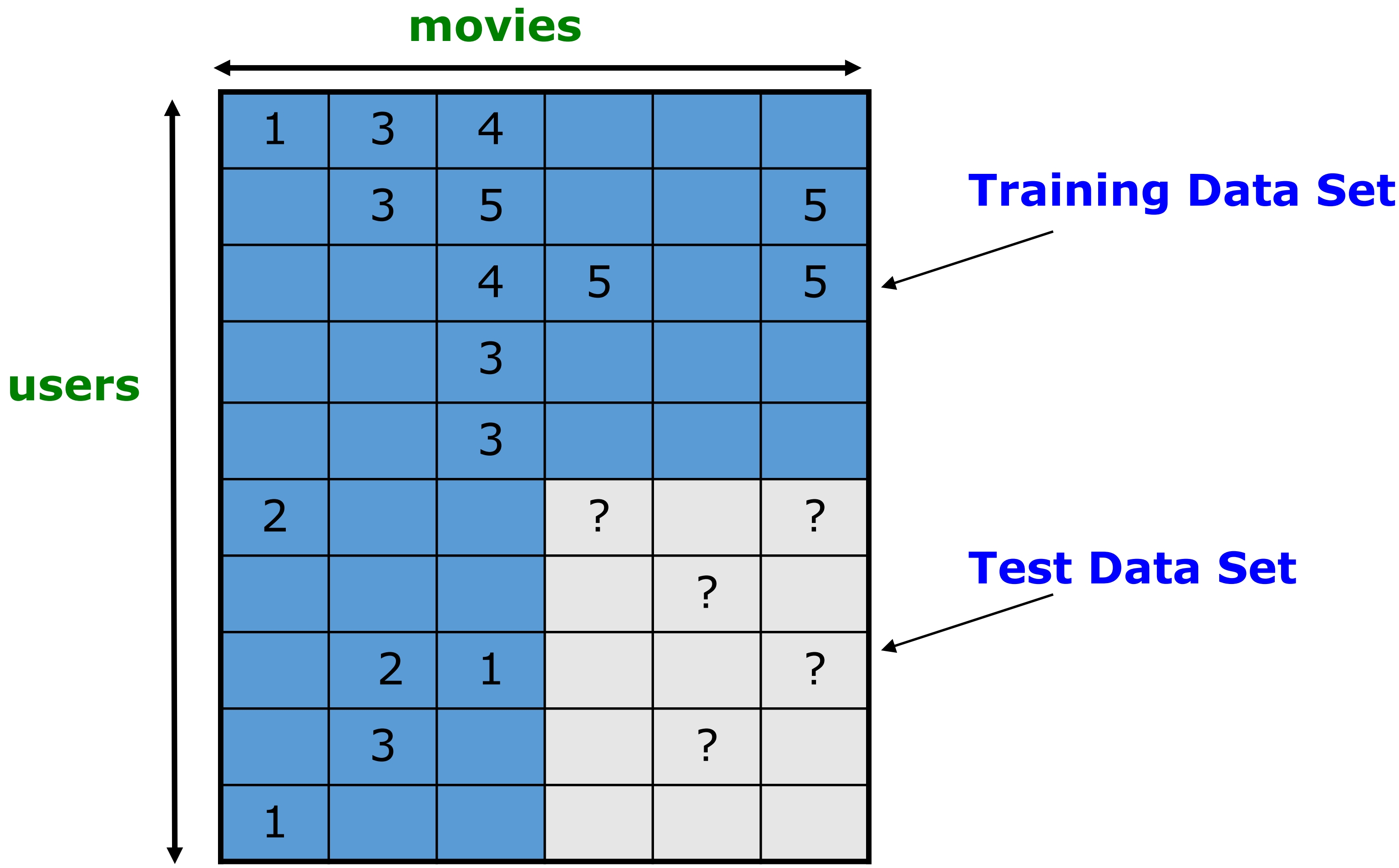
Evaluation

movies

users

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			2		2
				5	
	2	1			1
	3			3	
1					

Evaluation



Evaluating Predictions

- **Compare predictions with known ratings**
  - **Root-mean-square error** (RMSE)
    - $\sqrt{\sum_{xi}\left(r_{xi} - r_{xi}^{*}\right)^2}$ where $\boldsymbol{r_{xi}}$ is predicted, $\boldsymbol{r_{xi}^{*}}$ is the true rating of $\boldsymbol{x}$ on $\boldsymbol{i}$
  - **Precision at top 10**:
    - % of those in top 10
  - **Rank Correlation**

Evaluation in the Mini-Project

- Rank evaluation
- For reach user provide id of item with the highest ranking:
  - If item is one of the highest ranked items: 1 point
  - If item is one of the second highest ranked items: ½ point
- Overall: number of point/number of users
- E.g., assume we have 5 user
  - for 1st user we suggest top item = 1 point,
  - for 2nd user we suggest top item = 1 point,
  - 3rd and 4th user second top = ½ point and we mis-predicated the 5th user that give (1 + 1 + ½ + ½)/5=0.6

Test Data

- Recall: we assume that M is the product of two matrices U and V
- **Create your own test data:**
  1. Create two random matrices U and V with dimension $n \times d$ and $d \times m$, respectively, for some n, d, and m.
  2. Compute product $P = U \cdot V$
  3. Create a copy of P and called it M
  4. Select (maybe randomly) some entries of M and set them to 0
  5. Ask you algorithm to reconstruct P from M using dimension d.

Test Data: Example

- Step 1: create U and V (e.g., use createMatrix)

```
U = {{3.0, 3.0},
     {2.0, 1.0},
     {0.0, 2.0},
     {3.0, 3.0},
     {1.0, 3.0}};
```

```
V = {{1.0, 1.0, 1.0, 1.0, 1.0, 3.0},
     {1.0, 3.0, 0.0, 1.0, 2.0, 3.0}};
```

- Step 2: compute product P (e.g., use multiplyMatrix)

```
P = {{6.0, 12.0, 3.0, 6.0, 9.0, 18.0},
     {3.0,  5.0, 2.0, 3.0, 4.0,  9.0},
     {2.0,  6.0, 0.0, 2.0, 4.0,  6.0},
     {6.0, 12.0, 3.0, 6.0, 9.0, 18.0},
     {4.0, 10.0, 1.0, 4.0, 7.0, 12.0}};
```

- Step 3+4: Copy and replace some entries with 0s

```
M = {{6.0, 12.0, 3.0, 6.0, 0.0,  0.0},
     {3.0,  5.0, 2.0, 3.0, 0.0,  9.0},
     {0.0,  0.0, 0.0, 2.0, 4.0,  6.0},
     {6.0, 12.0, 3.0, 0.0, 0.0, 18.0},
     {4.0,  0.0, 1.0, 4.0, 0.0, 12.0}}
```

Test Data: Netflix Challenge

- Bonus
- For interested students, we will provide utility matrices based on the Netflix training data

Netflix Challenge

The Netflix Prize

- **Training data**
  - 100 million ratings, 480'000 users, 17'770 movies
  - 6 years of data: 2000-2005
- **Test data**
  - Last few ratings of each user (2.8 million)
  - **Evaluation criterion:** Root Mean Square Error (RMSE) $= \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$
  - **Netflix's system RMSE: 0.9514**
- **Competition (Started in 2006)**
  - 2,700+ teams
  - **$1 million** prize for 10% improvement on Netflix

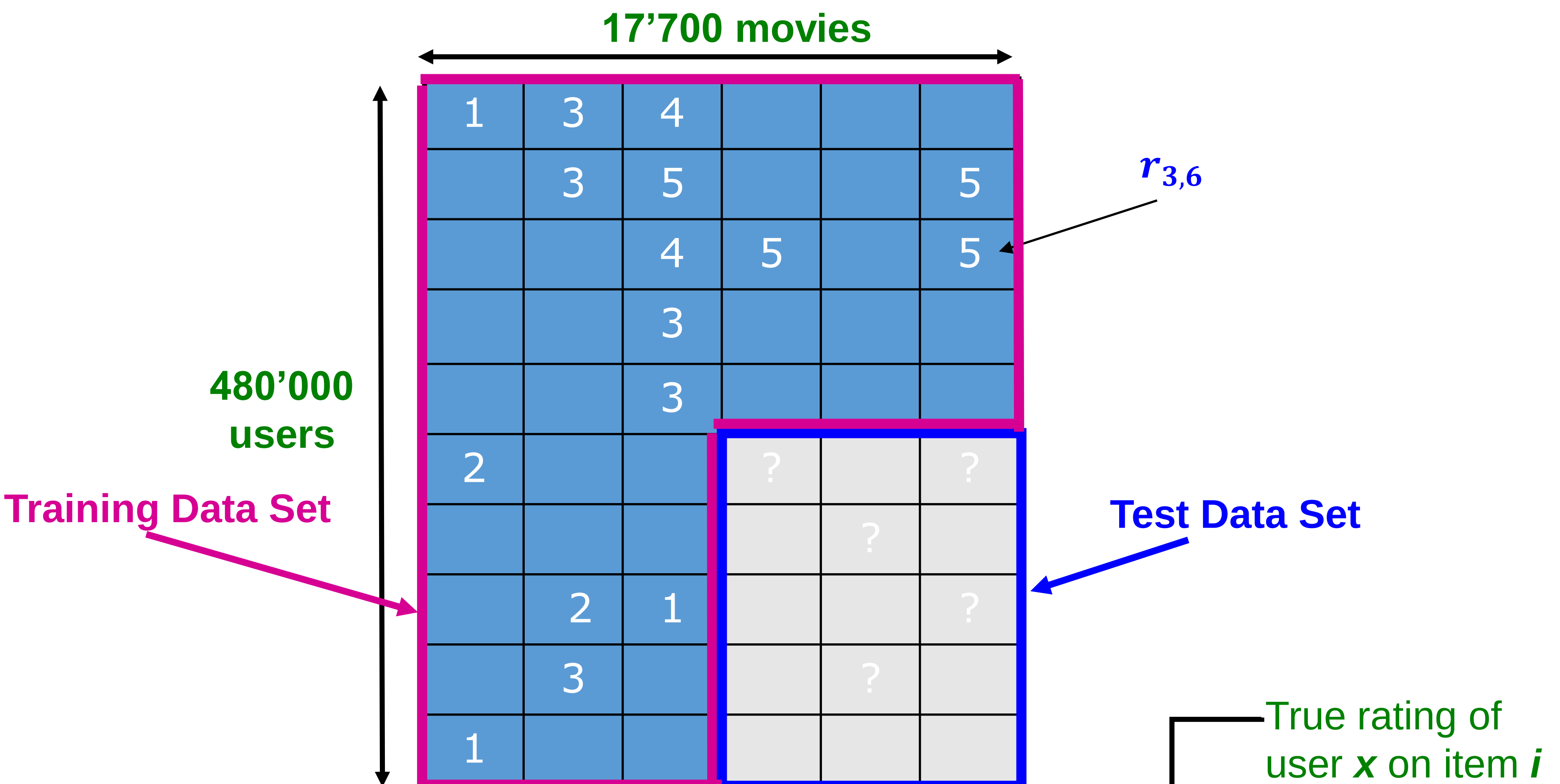
The Netflix Utility Matrix

17'700 movies

480'000 users

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			2		2
				5	
	2	1			1
	3			3	
1					

Utility Matrix: Evaluation

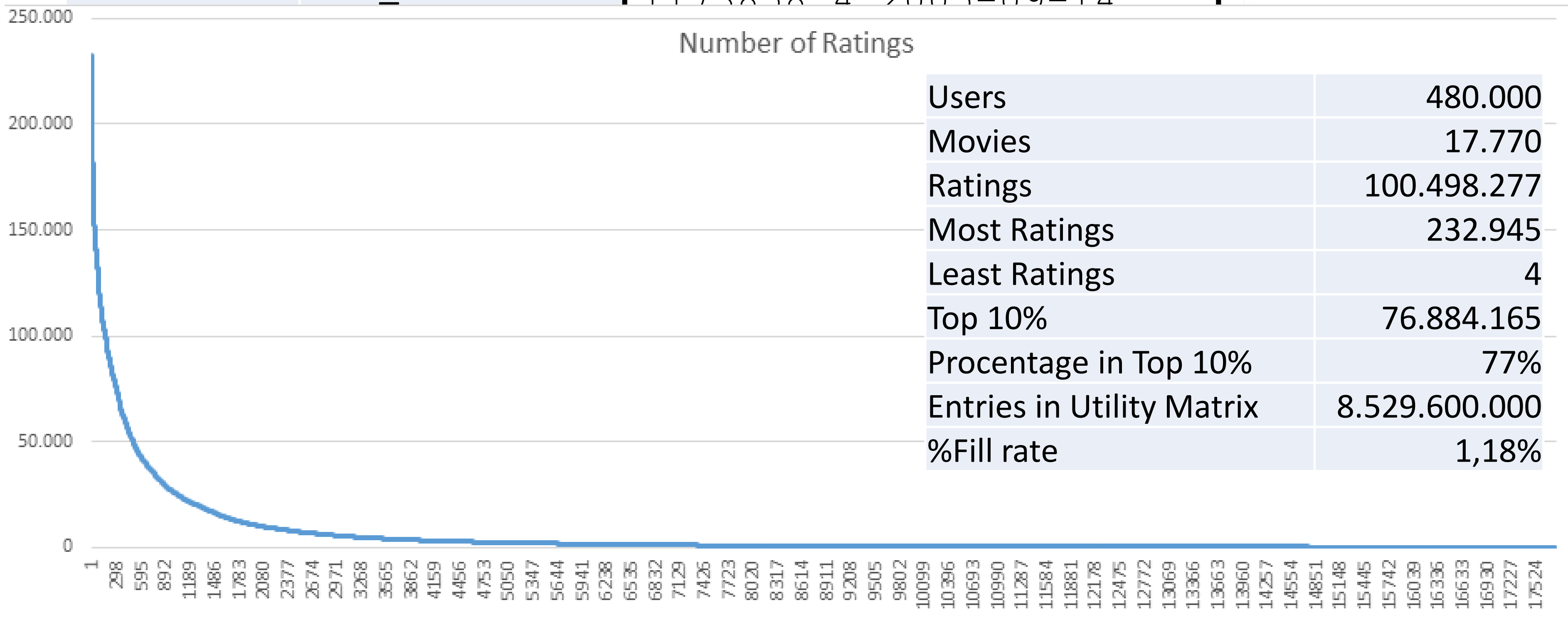


$$\text{RMSE} = \frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$$

Netflix Training Data

#Ratings	Movie File
232.945	mv_0005317.txt
216.597	mv_0015124.txt
200.833	mv_0014313.txt
196.398	mv_0015205.txt
193.942	mv_0001905.txt

```
7961:
1744308,1,2004-10-07
2619050,1,2005-08-09
2144174,4,2005-06-20
2125733,2,2003-05-05
269983,2,2004-05-12
2026560,4,2004-03-23
1173858,4,2005-09-14
```



Users	480.000
Movies	17.770
Ratings	100.498.277
Most Ratings	232.945
Least Ratings	4
Top 10%	76.884.165
Procentage in Top 10%	77%
Entries in Utility Matrix	8.529.600.000
%Fill rate	1,18%

Small Utility Matrix from Netflix data

```
{{ 0.0, 4.0, 5.0, 4.0, 3.0},
 { 3.0, 3.0, 2.0, 0.0, 4.0},
 { 5.0, 4.0, 5.0, 3.0, 4.0},
 { 2.0, 0.0, 0.0, 0.0, 0.0},
 { 2.0, 0.0, 0.0, 4.0, 5.0},
 { 1.0, 0.0, 4.0, 0.0, 0.0},
 { 3.0, 2.0, 3.0, 3.0, 5.0},
 { 5.0, 3.0, 4.0, 4.0, 4.0},
 { 0.0, 5.0, 5.0, 4.0, 4.0},
 { 4.0, 3.0, 3.0, 0.0, 4.0}}
```

ID	Movie File	Title
0	mv_0005317.txt	Miss Congeniality/Miss Détective
1	mv_0015124.txt	Independence Day
2	mv_0014313.txt	The Patriot
3	mv_0015205.txt	The Day After Tomorrow
4	mv_0001905.txt	Pirates of the Caribbean: The Curse of the Black Pearl

#Movies: 5, #Users: 10

Small Utility Matrix from Netflix data

```
{{ 0.0, 4.0, 5.0, 4.0, 3.0, 1.0, 5.0, 3.0, 3.0, 3.0},
 { 3.0, 3.0, 2.0, 0.0, 4.0, 5.0, 2.0, 0.0, 4.0, 2.0},
 { 5.0, 4.0, 5.0, 3.0, 4.0, 4.0, 3.0, 4.0, 4.0, 0.0},
 { 2.0, 0.0, 0.0, 0.0, 0.0, 3.0, 5.0, 0.0, 0.0, 2.0},
 { 2.0, 0.0, 0.0, 4.0, 5.0, 4.0, 3.0, 5.0, 3.0, 0.0},
 { 1.0, 0.0, 4.0, 0.0, 0.0, 5.0, 5.0, 5.0, 0.0, 0.0},
 { 3.0, 2.0, 3.0, 3.0, 5.0, 3.0, 4.0, 4.0, 2.0, 2.0},
 { 5.0, 3.0, 4.0, 4.0, 4.0, 5.0, 5.0, 4.0, 3.0, 3.0},
 { 0.0, 5.0, 5.0, 4.0, 4.0, 3.0, 0.0, 0.0, 4.0, 5.0},
 { 4.0, 3.0, 3.0, 0.0, 4.0, 2.0, 5.0, 5.0, 3.0, 3.0},
 { 5.0, 5.0, 4.0, 4.0, 5.0, 5.0, 5.0, 5.0, 4.0, 5.0},
 { 4.0, 3.0, 0.0, 0.0, 5.0, 3.0, 0.0, 0.0, 0.0, 0.0},
 { 0.0, 0.0, 0.0, 0.0, 4.0, 0.0, 5.0, 5.0, 0.0, 0.0},
 { 3.0, 0.0, 4.0, 0.0, 4.0, 4.0, 4.0, 4.0, 3.0, 2.0},
 { 4.0, 2.0, 2.0, 0.0, 5.0, 2.0, 5.0, 3.0, 0.0, 3.0},
 { 1.0, 3.0, 0.0, 0.0, 0.0, 2.0, 2.0, 0.0, 4.0, 3.0},
 { 4.0, 3.0, 0.0, 2.0, 3.0, 3.0, 4.0, 4.0, 2.0, 4.0},
 { 4.0, 4.0, 3.0, 3.0, 5.0, 5.0, 2.0, 0.0, 0.0, 4.0},
 { 3.0, 0.0, 3.0, 0.0, 1.0, 5.0, 0.0, 1.0, 0.0, 5.0},
 { 2.0, 2.0, 2.0, 3.0, 4.0, 5.0, 4.0, 0.0, 0.0, 3.0}}
```

#Movies: 10, #Users: 20

ID		
0	mv_0005317.txt	Miss Congeniality
1	mv_0015124.txt	Independence Day
2	mv_0014313.txt	The Patriot
3	mv_0015205.txt	The Day After Tomorrow
4	mv_0001905.txt	Pirates of the Caribbean
5	mv_0006287.txt	Pretty Woman
6	mv_0011283.txt	Forrest Gump
7	mv_0016377.txt	The Green Mile
8	mv_0016242.txt	Con Air
9	mv_0012470.txt	Twister

Performance of Various Methods

Global average: 1.1296

User average: 1.0651

Movie average: 1.0533

Netflix: 0.9514

Basic Collaborative filtering: 0.94

CF+Biases+learned weights: 0.91

Grand Prize: 0.8563

```
{{ 0.0, 4.0, 5.0, 4.0, 3.0},
 { 3.0, 3.0, 2.0, 0.0, 4.0},
 { 5.0, 4.0, 5.0, 3.0, 4.0},
 { 2.0, 0.0, 0.0, 0.0, 0.0},
 { 2.0, 0.0, 0.0, 4.0, 5.0},
 { 1.0, 0.0, 4.0, 0.0, 0.0},
 { 3.0, 2.0, 3.0, 3.0, 5.0},
 { 5.0, 3.0, 4.0, 4.0, 4.0},
 { 0.0, 5.0, 5.0, 4.0, 4.0},
 { 4.0, 3.0, 3.0, 0.0, 4.0}}
```

```
            3.0, 2.0, 4.0},
~1.12       5.0, 4.0, 3.0},
            1.0, 0.0, 4.0}}
```

```
            3.0, 3.0, 4.0},
~0.94       5.0, 4.0, 3.0},
            1.0, 0.0, 4.0}}
```

```
            3.0, 4.0, 4.0},
~0.86       5.0, 4.0, 3.0},
            1.0, 0.0, 4.0}}
```

NETFLIX

Netflix Prize

COMPLETED

Home | Rules | Leaderboard | Update | Download

Leaderboard

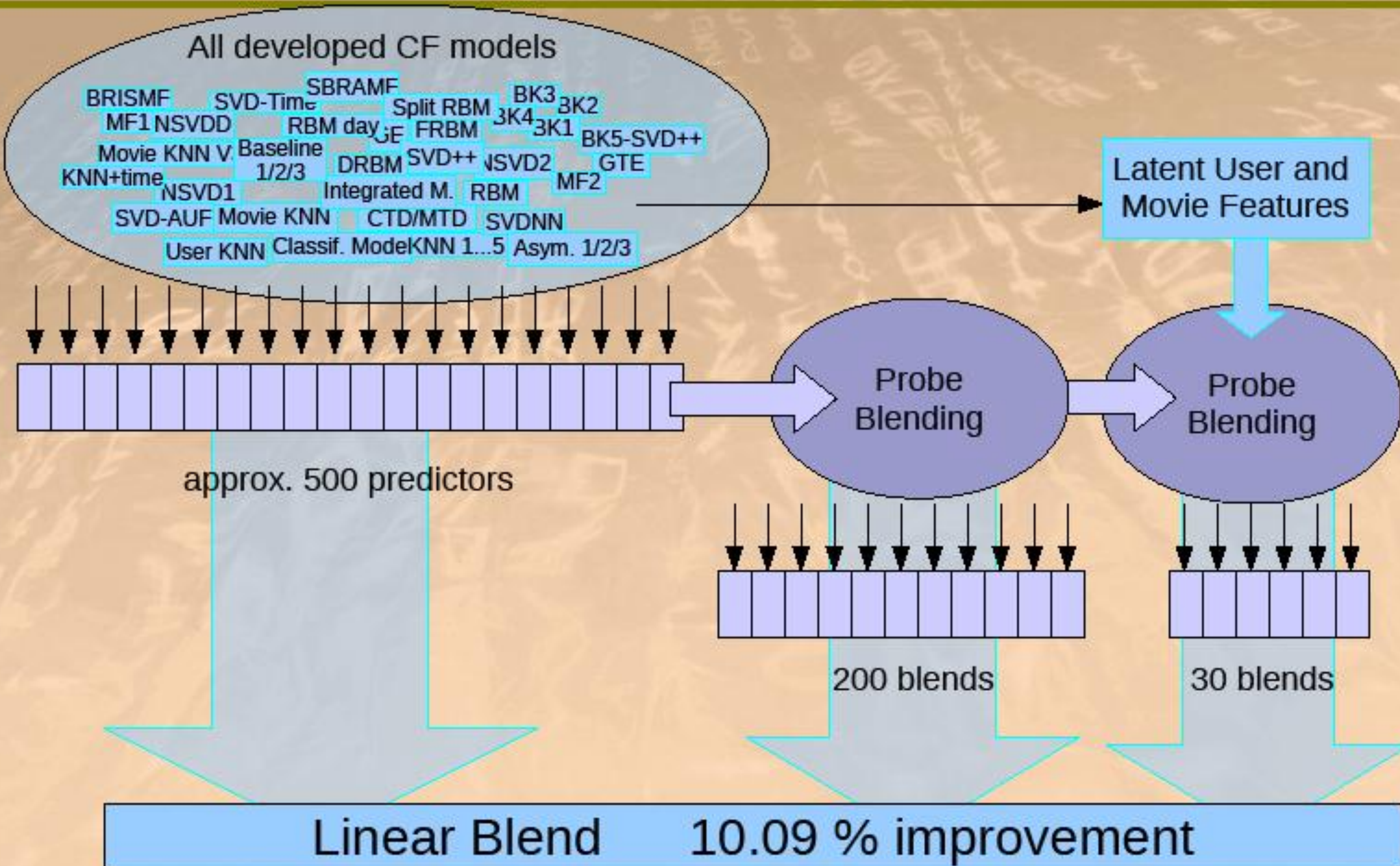
Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	[illegible]	[illegible]	[illegible]
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22
14	Gravity	0.8643	9.26	2009-04-22 18:31:32
15	Ces	0.8651	9.18	2009-06-21 19:24:53
16	Invisible Ideas	0.8653	9.15	2009-07-15 15:53:04
17	Just a guy in a garage	0.8662	9.06	2009-05-24 10:02:54
18	J Dennis Su	0.8666	9.02	2009-03-07 17:16:17
19	Craig Carmichael	0.8666	9.02	2009-07-25 16:00:54
20	acmehill	0.8668	9.00	2009-03-21 16:20:50
Progress Prize 2007 - RMSE = 0.8723 - Winning Team: KorBell				

Million $ Awarded Sept 21st 2009

The big picture
Solution of BellKor's Pragmatic Chaos



Michael Jahrer / Andreas Töscher – Team BigChaos – September 21, 2009

Further Reading:

- Y. Koren, Collaborative filtering with temporal dynamics, KDD '09
- http://www2.research.att.com/~volinsky/netflix/bpc.html
- http://www.the-ensemble.com/

Tasks

optimizeU/optimizeV

recommend

Task: Optimize U and V

- Compute an improved version of U (or V, respectively)
  - Input: three matrices M, U, and V
  - Return value: a new matrix U (or V, respectively) that minimizes the RMSE (up to a reasonable bound) between M and P=U·V.
- Signatures:

```
public static double[][] optimizeU( double[][] M,
                                     double[][] U,
                                     double[][] V)

public static double[][] optimizeV( double[][] M,
                                     double[][] U,
                                     double[][] V)
```

- **Your choice**: order of updates, stopping criteria

Task: Recommend

- Provide a recommendation for each user:
  - Input: matrix M and an integer d (number of features)
  - Return value: an integer array indicating at position i the top recommendation for user i.
- An item should only be recommended if
  1. it was previously not rated by user i (i.e., the corresponding entry in M is 0) and
  2. It is recommended by the UV-decomposition algorithm with dimension d for the matrix M (i.e., this item has the highest score among the non-ranked items).
- If there is no item without ranking for user i, return -1
- Signature:

```
public static int[] recommend( double[][] M, int d)
```

- **Your choice**: initial states, order of updates, stopping criteria

Summary

- Feature Vector/Matrices U and V
  - Update an element in U or V
- A complete UV-Decomposition
  - Optimization and Stopping criteria
  - Issues with local minima and Initialization
- Evaluation of (your) recommendations
- Netflix Challenge
- Tasks

Questions?