IntroProp Mini-project:
Recommendation System

Barbara Jobstmann

Oct 16, 2014

Outline (Part 1) – Today

- Motivation
- Formal Model
  - Utility matrix
  - Representation of utility matrix in JAVA
- Tasks:
  - matrixToString
  - isMatrix
  - createMatrix

Outline (Part 2) – (Partly) Today

- Key Problems and Main Approaches
- UV-Decomposition
  - Error computation (Root-Mean-Square-Error)
  - Updating a single element
- Tasks:
  - multiplyMatrix
  - RMSE
  - updateUElem/updateVElem

Outline (Part 3) – Next Week

- A complete UV-Decomposition
  - Issues with local minima
  - Initialization
  - Optimization
  - Stopping criteria
- Evaluation of (your) recommendations
  - Netflix data set
- Tasks
  - optimizeU/optimizeV
  - recommend

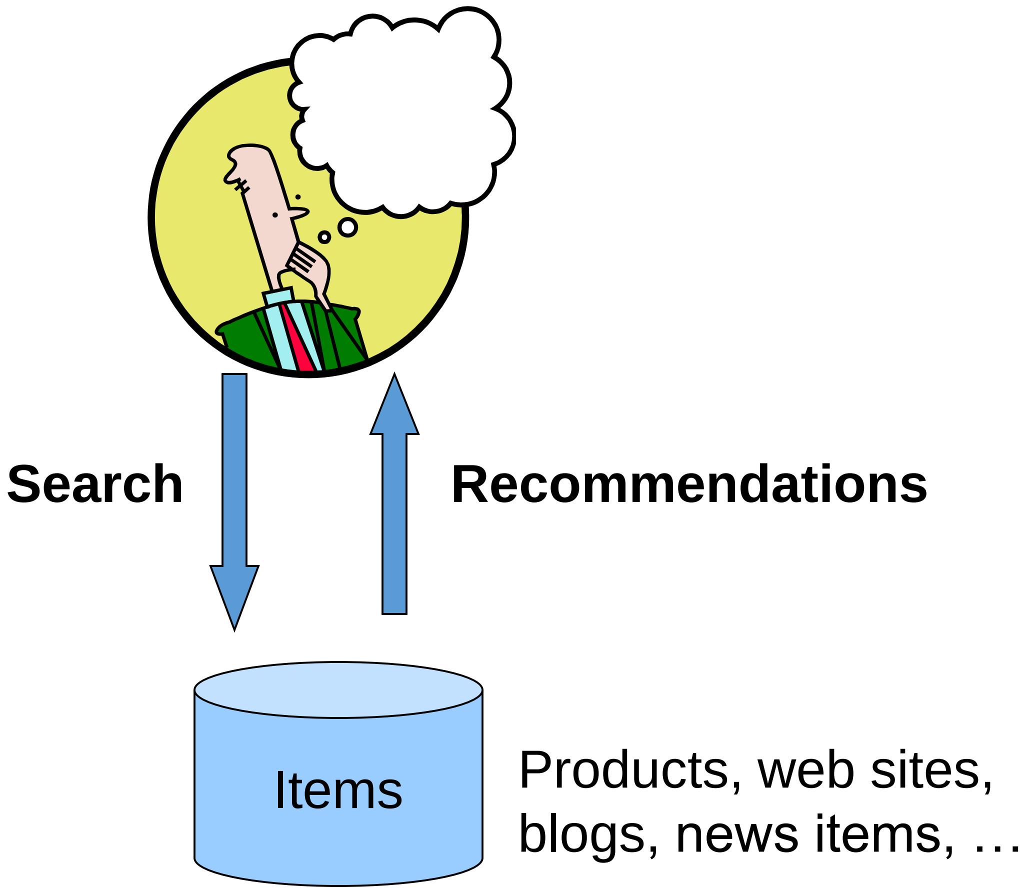
Motivation

Recommendation Systems

- Customer 1
  - Buys X-men
  - Buys Captain America

- Customer 2
  - Searches for X-men
  - Recommendation system suggests Captain America from data collected about customer 1

Recommendations

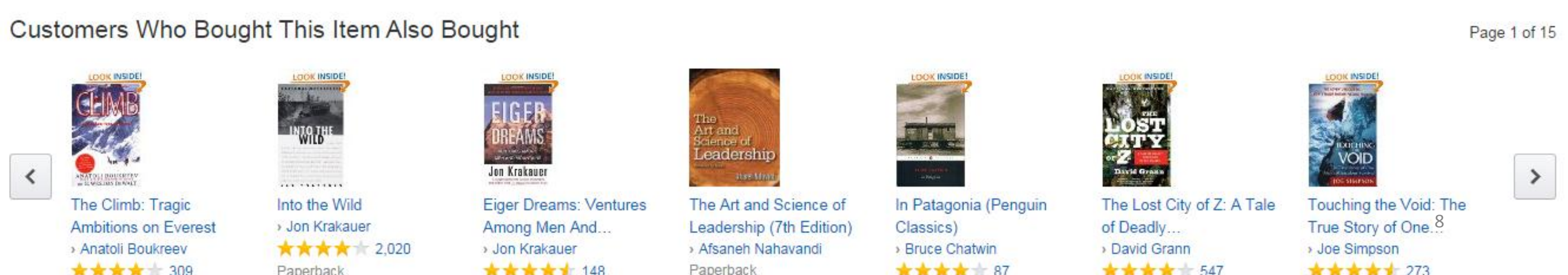


Examples:

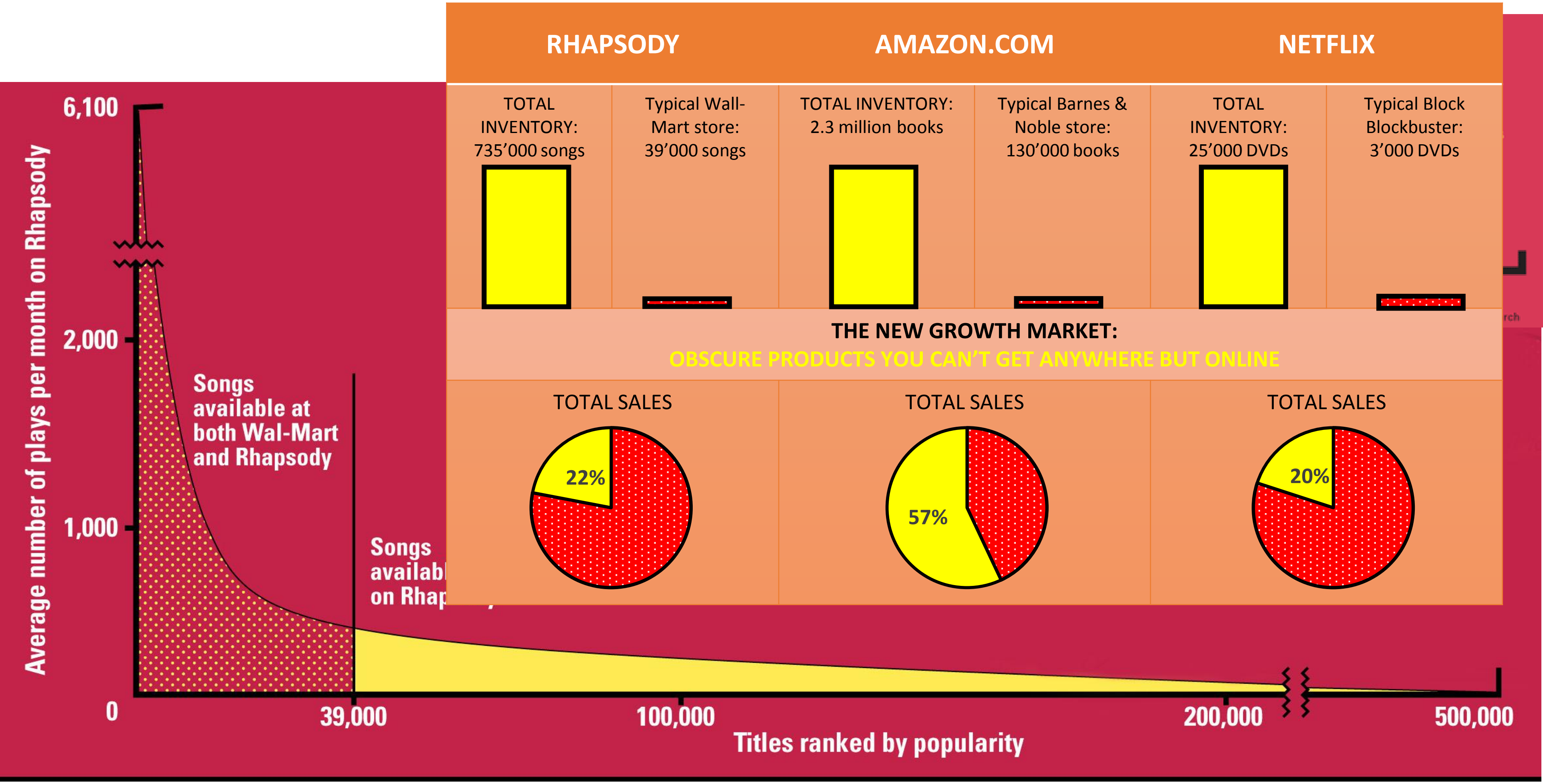
amazon.com
ANDROID APP ON Google play
NETFLIX
XBOX LIVE
last.fm the social music revolution
Google News
YouTube

Nowadays: Customers Spoilt for Choice

- Shelf space is limited for traditional shop owners
  - Also: TV networks, movie theaters,…
  - Resulting into limit choice for customers (shop owner was pre-selecting products)
- Web enables near-zero-cost distribution of information about products
  - Explosion of available products
- More choice implies the need for better filters
  - Recommendation engines
- How Into Thin Air made Touching the Void a bestseller:



The New Marketplace: The Long Tail



Source: Erik Brynjolfsson and Jeffrey Hu, MIT, and Michael Smith, Carnegie Mellon; Barnes & Noble; Netflix; RealNetworks

Types of Recommendations

- Editorial and managed by one or several individuals
  - List of favorites
  - Lists of “essential” items

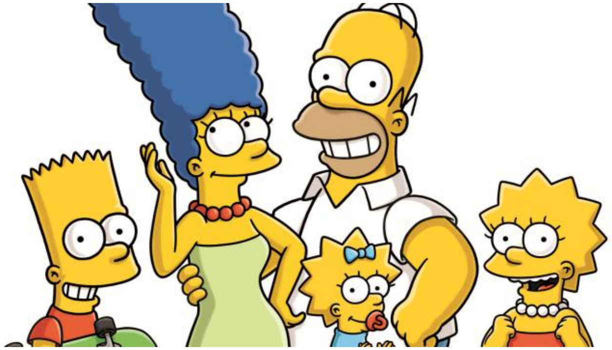
- Simple aggregates
  - Top 10, Most Popular, Recent Uploads

- Tailored to individual users
  - Amazon, Netflix, …

Focus of this mini-project

Formal Model

Formal Model



- X = set of Customers

- S = set of Items
- Utility function $u: X \times S \rightarrow R$
  - R = set of ratings
  - R is a totally ordered set,
    e.g., 0-5 stars, real number in $[0,1]$

Utility Matrix M

	Avatar	LOTR	Matrix	Pirates
Alice	10		2	
Bob		5	?	3
Carlo	2		1	
David				4

Question:
Does Bob like Matrix?

Representation of Utility Matrix

- Matrix: two-dimensional double array
- For example:

	Avatar	LOTR	Matrix	Pirates
Alice	10		2	
Bob		5.5		3
Carlo	2		1	
David				4.5

```
double[][] M = {{10,   0, 2, 0},
                { 0, 5.5, 0, 3},
                { 2,   0, 1, 0},
                { 0,   0, 0, 4.5}};
```

Blank entries translate into 0s

Representation of Utility Matrix

- Matrix: two-dimensional double array
- For example:

	0	1	2	3
0	10		2	
1		5.5		3
2	4		1	
3				4.5

n users = n rows
(numbered from 0 to n-1)
m items = m columns
(numbered from 0 to m-1)

```
double[][] M = {{10,   0, 2, 0},
                { 0, 5.5, 0, 3},
                { 4,   0, 1, 0},
                { 0,   0, 0, 4.5}};
//Access: M[row][column],
//e.g., M[0][2]; row 0, column 2
System.out.println(M[0][2]); //2.0
//Size: M.length
//number of rows in M
int n = M.length;
//number of columns in row 2 of M
int m = M[2].length;
//new array of size n x m
double[][] C = new double[n][m];
double[][] N = M.clone();
```

Tasks (Part 1)

Getting familiar with matrices that are represented as two-dimensional arrays

Task: matrixToString

Task: isMatrix

Task: createMatrix

Task: Matrix to String (1)

- Create a String representing a matrix (two-dimensional double array) in Java syntax
  - Input: two-dimensional double array
  - Result value: String
- Signature:

```
public static String matrixToString(double[][] A)
```

- Propose: help with debugging (e.g., print intermediate results, create test data,…)

Task: Matrix to String (2)

- Example:

```java
double[][] A = {{ 3, 5.4, 7.777, 1.0/3}, {0, 1, 2, 3},
                {1.1, 2.222, 3}};
System.out.println(matrixToString(A));
```

- Outcome:

```
Console ⌧   Call Hierarchy
<terminated> RecommendationRef [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\javaw.exe (13.10.2014 23:31:42)
{ { 3.0, 5.4, 7.777, 0.3333333333333333},
 { 0.0, 1.0, 2.0, 3.0},
 { 1.1, 2.222, 3.0}}
```

- Requirement: valid Java syntax and "same matrix" (i.e., entries are similar B[i][j] = A[i][j] ± 10^{-6})

```java
double[][] B = { { 3.0, 5.4, 7.777, 0.3333333333333333},
 { 0.0, 1.0, 2.0, 3.0},
 { 1.1, 2.222, 3.0}};
```

Task: Check if Array is Matrix (1)

- Check if given two-dimensional double array is a valid matrix, i.e.,
  1. it is not empty (size zero or having the value null),
  2. none of the rows are empty and
  3. all of them have the same length.
  - Input : two-dimensional double array
  - Return value: Boolean (true if array is matrix, otherwise false)
- Signature:

```
public static boolean isMatrix( double[][] A )
```

- Purpose: sanity check of input data (to avoid null pointer errors), e.g., if matrix A is multiplied with B.

Task: Check if Array is Matrix (2)

- Example of arrays:

```java
//false: array is null
double [][] A1 = null;

//false: empty array
double [][] A2 = {{}};

//false: rows have different lengths
double [][] A3 = {{1.0,2},{1,2,3.3}};

//true: this is a valid matrix
double [][] A4 = {{1.0,2,0},{1,2,3.3}};
```

Task: Create Random Matrix (1)

- Create a two-dimensional double array that represent a matrix with n rows and m columns and random entries in the interval [k;l].
  - Inputs: Integers n, m, k ,l
  - Return value: two-dimensional double array A
    - with A.length=n and A[i].length=m for all i=0,..,n-1 and
    - $k \leq A[i][j] \leq l$ for all i=0,..,n-1 and j=0,..,m-1
    - If m = 0, n = 0, or k > l, the method should return null.
    - If the method is called twice with the same arguments the results should be different.
- Signature:

```
public static double[][] createMatrix( int n, int m, int k, int l)
```

Task: Create Random Matrix (2)

- Propose: create test date for the following tasks
- Useful Java methods:

Example	Functionality
Random random = new Random();	Create a pseudorandom number generator
random.nextInt(100);	Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and 100 (exclusive)
random.nextDouble();	Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0.

- Example:

```java
double[][] A = createMatrix(3,5,0,10);
System.out.println(matrixToString(A));
```

```
{ { 6.956183763934913, 0.9714459479474902, 6.727736624831974, 8.336967798453953, 0.986801776579298},
 { 3.5202134832646625, 2.4852650677633448, 1.4076732341578213, 2.2782789831606842, 1.7789731572069378},
 { 5.7662008436273835, 5.89675976730843, 1.2829479453222148, 1.486100883078585, 6.4272971379721575}}
```

Summary (Part 1)

- Motivation
- Formal Model
    - Utility matrix
    - Representation of utility matrix in JAVA
- Tasks:
    - matrixToString
    - isMatrix
    - createMatrix

Outline (Part 2)

- Key Problems and Main Approaches
- UV-Decomposition
  - Error computation (Root-Mean-Square-Error)
  - Updating a single element
- Tasks:
  - multiplyMatrix
  - RMSE
  - updateUElem/updateVElem

Key Problems
Main Approaches

Recall

	Avatar	LOTR	Matrix	Pirates
Alice	10		2	
Bob		5	?	3
Carlo	2		1	
David				4

Question:
Does Bob like Matrix?

Key Problems

1. **Gathering** "known" ratings for matrix
   - How to collect the data in the utility matrix
2. **Extrapolate** unknown ratings from known ones
   - Mainly interested in high unknown ratings. We are not interested in knowing what you don't like but what you like
3. **Evaluating** extrapolation methods
   - How to measure success/performance of recommendation methods

	Avatar	LOTR	Matrix	[illegible]
Alice	10		2	
Bob		5	?	3
Carlo	[illegible]		1	[illegible]

Does Bob like Matrix?

How does Bob rate LOTR?

How good is the recommendation?

Key Problems

1. **Gathering** "known" ratings for matrix
   - How to collect the d… …ix
2. **Extrapolate** … known ones
   - Mainly interested in … s. We are not interested in knowing what you don't like but what you like
3. **Evaluating** extrapolation methods
   - How to measure success/performance of recommendation methods

Focus of this mini-project

Do you want to learn more about this or similar topics?

- Visit http://www.mmds.org: an online book and course on "Mining of Massive Datasets" by J. Leskovec, A. Rajaraman, J. Ullman
- Check out courses offered by the Data Analysis Theory and Application lab http://data.epfl.ch

Extrapolation

- Difficulties: Utility matrix U is sparse
  - Most people have not rated most items
    Netflix: 99% of entries in Utility matrix are empty
  - Cold start:
    New items have no ratings,
    New users have no history
- Basic approaches:
  1. Collaborative-Filtering Systems
  2. Content-Based Systems
  3. Dimensionality Reduction

Focus of this mini-project

- Real-life systems combine approaches

Collaborative-Filtering Systems

- Recommend items based on similarities between users and items.
  - The items recommended to a user are those preferred by similar users. Similarity is derived from similar ratings.

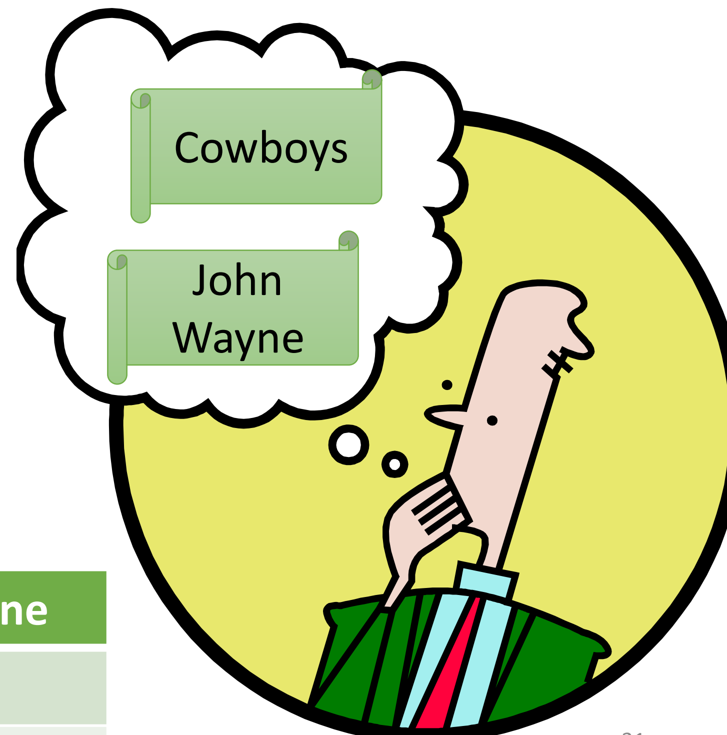
	Avatar	LOTR	Matrix	Pirates
	1	8		3
	2			3

Content-Based Systems

- Focus on properties of items.
  - E.g., if a user has seen many cowboy movies, then recommend another cowboy movie.
  - Create **item/user profiles** (list of properties/features)

Sci-Fi

Keanu Reeves



Profile/feature vector:

	Sci-Fi	Reeves	Cowboys	Wayne
Joe	0	0	4	5
Matrix	5	5	0	0

Profile/Feature Vectors

- 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	…	…	…	

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

- Idea: Rating ≈ User Profile · Item Profile

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**

Example: UV-decomposition

What does similar mean?

$$\begin{pmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{pmatrix} \approx \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{pmatrix} \cdot \begin{pmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{pmatrix}$$

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

$$u_{ij} \ldots element\ in\ U$$

$$v_{ij} \ldots element\ in\ V$$

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

Similarity: Root-Mean-Square-Error

- We could use any measure of how close P is to M to define “similar". We take the typical choice.
- **Root-Mean-Square-Error (RMSE)**, where we
  1. Sum over all non-zero entries in M, the square of the difference between that entry in M and the corresponding entry in P, i.e.,
     $$S = \sum_i \sum_j \begin{cases} (m_{ij} - p_{ij})^2 & \text{if } m_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$
  2. Take the mean (average) S_{mean} of these squares by dividing by the number of terms in the sum (i.e., the number of non-zero entries)
  3. Take the square root of the mean, i.e.,
     $$RMSE = \sqrt{S_{\text{mean}}}$$

Example: RMSE computation

$$\begin{pmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{pmatrix} \approx \begin{pmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \\ u_{41} & u_{42} \\ u_{51} & u_{52} \end{pmatrix} \cdot \begin{pmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \end{pmatrix}$$

- Assume all elements of U and V are 1

$1 \cdot 1 + 1 \cdot 1 = 2$

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

Example: RMSE Computation

RMSE between $\begin{pmatrix} 5 & 2 & 4 & 4 & 3 \\ 3 & 1 & 2 & 4 & 1 \\ 2 & & 3 & 1 & 4 \\ 2 & 5 & 4 & 3 & 5 \\ 4 & 4 & 5 & 4 & \end{pmatrix}$ and $\begin{pmatrix} 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 & 2 \end{pmatrix}$?

- 1st row: $(5-2)^2 + (2-2)^2 + (4-2)^2 + (4-2)^2 + (3-2)^2 = 9+4+4+1=18$
- 2nd row: $(1)^2 + (-1)^2 + (0)^2 + (2)^2 + (-1)^2 = 7$
- 3rd row: $(0)^2 + \qquad + (1)^2 + (-1)^2 + (2)^2 = 6$
- 4th row: $0 + 9 + 4 + 1 + 9 = 23$
- 5th row: $4 + 4 \; + 9 + 4 + \quad = 21$
- $S = 18 + 7 + 6 + 23 + 21 = 75$
- $S_{mean} = 75 /(25-2) \approx 3.26$
- $\text{RMSE} = \sqrt{3.26} \approx 1.8$

Note that if we minimize the sum of squares for row 1 is smaller, so is the RMSE because mean and square root does not change the order.

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?

- Note that if we minimize the sum of squares (Step 1 of the RMSE computation), we also minimize the RMSE. So, we don’t need to worry about the average (Step 2) or the square root (Step 3).

Example : Adjusting Element

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

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

- 1st row:
  $(5-(x+1))^2 + (2-(x+1))^2 + (4-(x+1))^2 + (4-(x+1))^2 + (3-(x+1))^2$
  $= (4 - x)^2 + (1 - x)^2 + (3 - x)^2 + (3 - x)^2 + (2 - x)^2$
- We want the value of x that minimizes the sum, so we take the derivative and set it equal to 0:
  $-2 \cdot (4 - x) - 2 \cdot (1 - x) - 2 \cdot (3 - x) - 2 \cdot (3 - x) - 2 \cdot (2 - x) = 0$
  $4 - x + 1 - x + 3 - x + 3 - x + 2 - x = 0$
  $13 = 5 \cdot x$
  $x = 13/5 = 2.6$
- 1st row: $(1.4)^2 + (-1.6)^2 + (0.4)^2 + (0.4)^2 + (-0.6)^2 = 5.2$ (old value=18)

Example : Adjusting Element

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

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

Empty for 2nd column

- 3rd row:
  $(2-(x+1))^2 + \boxed{} + (3-(x+1))^2 + (1-(x+1))^2 + (4-(x+1))^2$
  $= (1 - x)^2 + \boxed{} + (2 - x)^2 + (-x)^2 + (3 - x)^2$
- Derivative and set it equal to 0:
  $-2\cdot(1 - x) - 2\cdot(2 - x) - 2\cdot(-x) - 2\cdot(3 - x) = -2 \cdot (6 - 4 \cdot x) = 0$
  $x = 6/4 = 1.5$

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

Example : Adjusting Element

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

- 5rd column:
  $(3-(x+1))^2 + (1-(x+1)^2 + (4-(x+1))^2 + (5-(x+1))^2$
  $= (2 - x)^2 + (- x)^2 \quad + (3 - x)^2 \quad + (4 - x)^2$
- Derivative and set it equal to 0:
  $-2\cdot(2 - x - x + 3 - x + 4 - x) = -2\cdot(9 - 4 \cdot x) = 0$
  $x = 9/4 = 2.25$

Adjusting Arbitrary Element (1)

- Given: $M_{n \times m}, U_{n \times d}, V_{d \times m}$
- Suppose we want to vary u_{rs} (we call new value x)
- u_{rs} affects only elements in row r of product $P = U \cdot V$
- Elements in row r:

$$p_{rj} = \sum_{k} u_{rk} \cdot v_{kj} = \sum_{k \neq s} u_{rk} \cdot v_{kj} + x \cdot v_{sj}$$

- Error due to element p_{rj} (for nonblank entry m_{rj}):

$$(m_{rj} - p_{rj})^2 = (m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj} - x \cdot v_{sj})^2$$

- Sum of squares:

$$\sum_{j} (m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj} - x \cdot v_{sj})^2$$

where $\sum_j$ mean sum over nonblank entries of M

Adjusting Arbitrary Element (2)

- **Goal**: minimize sum of squares, i.e.,

$$\sum_j (m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj} - x \cdot v_{sj})^2$$

- Find minimum? Take derivative and set it $= 0$

$$\sum_j -2 \cdot v_{sj} \cdot \left(m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj} - x \cdot v_{sj}\right) = 0$$

$$\sum_j v_{sj} \cdot \left(m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj}\right) - \sum_j v_{sj} \cdot x \cdot v_{sj} = 0$$

$$\sum_j v_{sj} \cdot \left(m_{rj} - \sum_{k \neq s} u_{rk} \cdot v_{kj}\right) = \sum_j v_{sj} \cdot x \cdot v_{sj}$$

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

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

Adjusting Arbitrary Element (3)

- To update element u_{rs} use

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

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

- To update element v_{rs} use

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

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

Tasks (Part 2)

Compute how “good” your current estimates for U and V are.
Improve an element in U or V.

Task: multiplyMatrix

Task: RMSE

Task: updateUElem/updateVElem

Task: Multiply two Matrices

- Compute product of two given matrices
  - Input: two two-dimensional double arrays A and B
  - Return value: two-dimensional double array representing the matrix product $P = A \cdot B$, i.e., element $P_{ij} = \sum_x A_{ix} \cdot B_{xj}$
  - If A and B have different dimensions, return null.
- Signature:

```
public static double[][] multiplyMatrix(double[][] A,
                                        double[][] B)
```

- Example: see slide 34

Task: Compute RMSE

- Compute the Root-Mean-Square-Error between M and P (for all non-zero elements of M)
  - Input: two-dimensional double arrays M and P
  - Return value: double value representing the RMSE
  - If M and P have different dimensions, return value -1.
- Signature:

```
public static double rmse(double[][] M, double[][] P)
```

- Example: see slide 35

Task: Update Element in U or V

- Compute improve value of element in U (or V)
  - Input: two-dimensional arrays for M, U, V
  - Input: two indices r and s
  - Return value: new value of u_{rs} (or v_{rs}, respectively) that improves the RMSE between M and P (see slide 42)
- Signatures:

```
public static double updateUElem( double[][] M, double[][] U,
                                  double[][] V, int r, int s )
public static double updateVElem( double[][] M, double[][] U,
                                  double[][] V, int r, int s )
```

- Examples:
  - See slides 39-41 (show also intermediate results)
  - updateUElem(M, U, V, 0, 0) is in [5.999;6.001] and
  - updateVElem(M, U, V, 0, 0) is in [7.749;7.751] with
  - M={{ 0, 8, 9, 8, 7 }, { 18, 0, 18, 18, 18 },
    { 29, 28, 27, 0, 25 }, { 6, 6, 0, 6, 6 }, { 17, 16, 15, 14, 0 }};

Summary (Part 2)

- Key Problems and Main Approaches
- UV-Decomposition
  - Error computation (Root-Mean-Square-Error)
  - Updating a single element
- Tasks:
  - multiplyMatrix
  - RMSE
  - updateUElem/updateVElem

Outline (Part 3) – Next Week

- A complete UV-Decomposition
  - Issues with local minima
  - Initialization
  - Optimization
  - Stopping criteria
- Evaluation of (your) recommendations
- Tasks
  - optimizeU/optimizeV
  - recommend