INTRODUCING RECOMMENDATION SYSTEMS

Scott O'Hara

Metrowest Developers Machine Learning Group

04/28/2021

REFERENCES

The material for this talk is primarily drawn from the notes, slides and lectures of the courses and book below:

Machine Learning Foundations: A Case Study Approach

University of Washington, Prof. Emily Fox & Prof. Carlos Guestrin

https://www.coursera.org/learn/ml-foundations

CSE/STAT 416 - Intro to Machine Learning Spring 2018

University of Washington, Prof. Emily Fox, Spring 2018

https://courses.cs.washington.edu/courses/cse416/18sp/index.html

University of Washington, Sewoong Oh, Spring 2019

https://courses.cs.washington.edu/courses/cse416/19sp/index.html

University of Washington, Vinitra Swamy, Summer 2020

https://courses.cs.washington.edu/courses/cse416/20su/index.html

Why Recommender Systems?

Personalization

Personalization is transforming our experience of the world

YouTube

100 Hours a Minute

What do I care about?

Information overload

↓

Browsing is "history"

– Need new ways to discover content

Personalization: Connects *users & items*

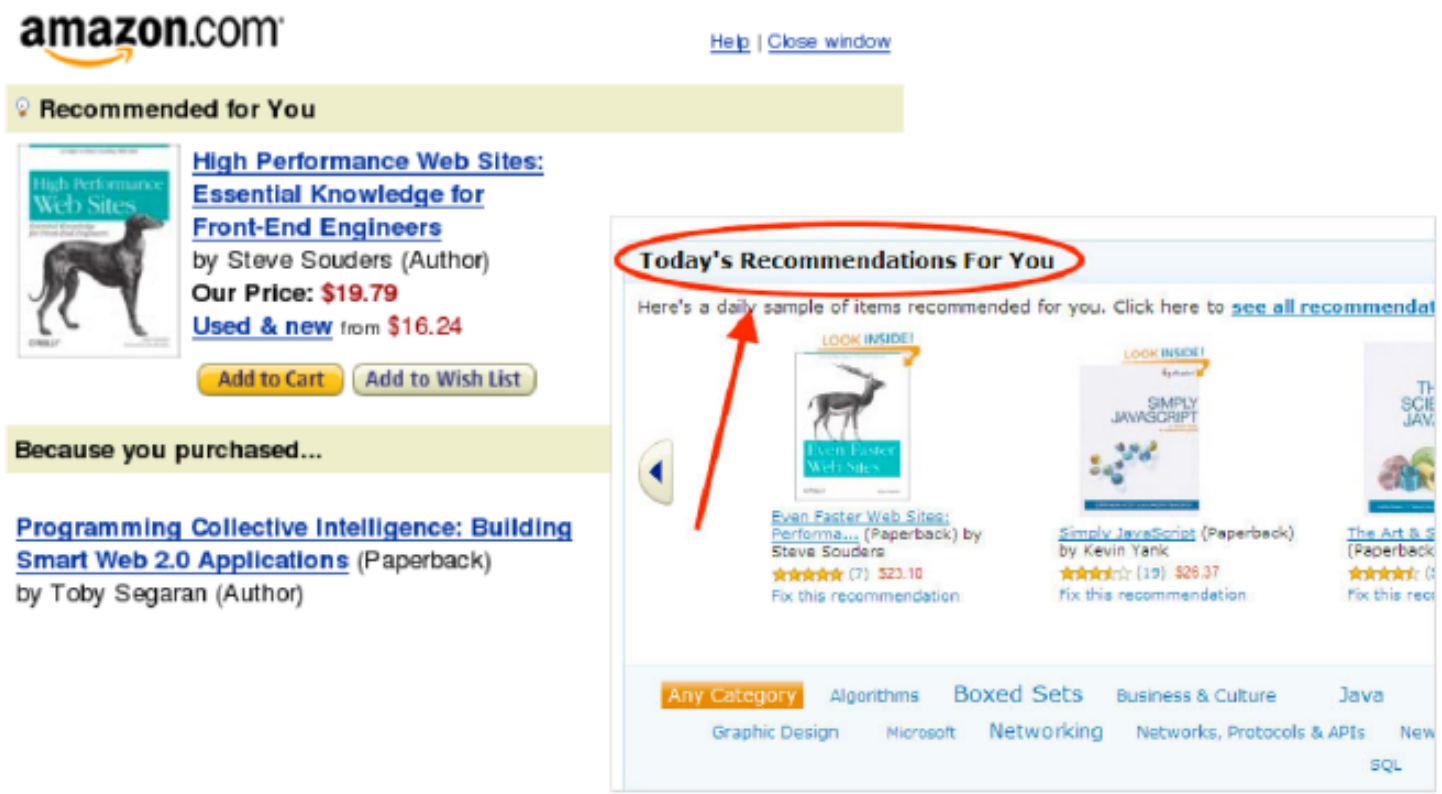
viewers

videos

Movie recommendations

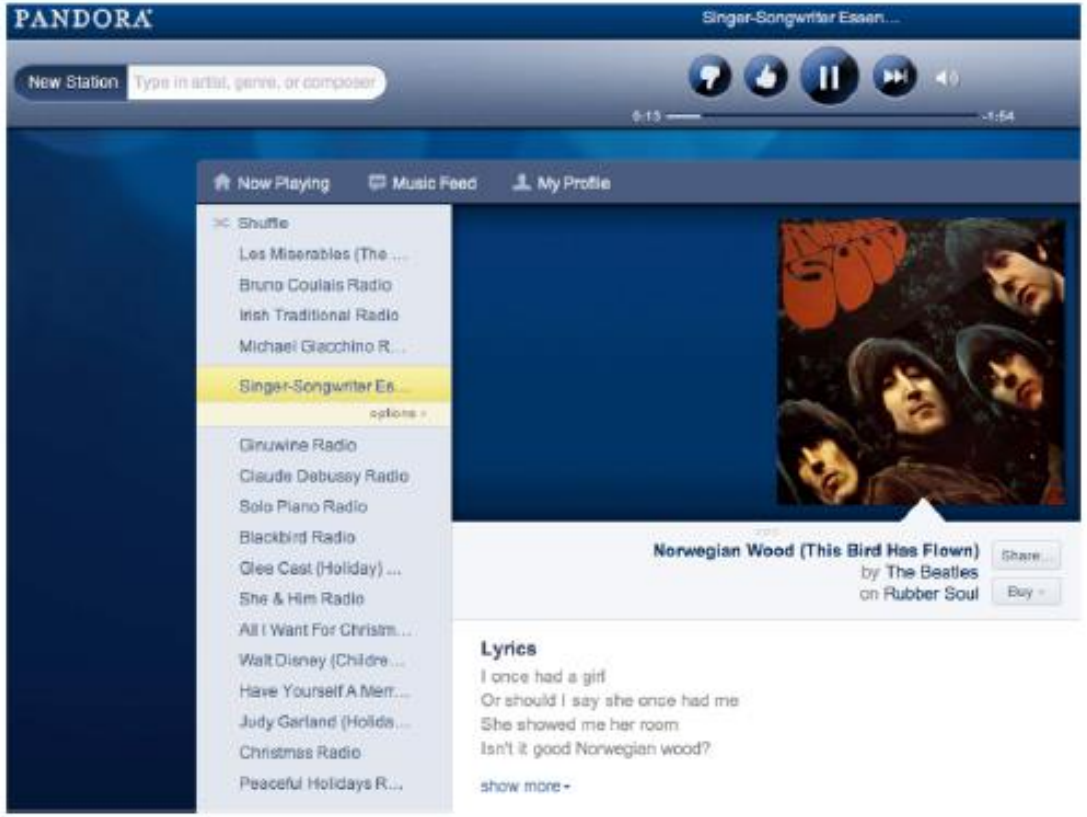
Connect users with movies they may want to watch

Product recommendations



Recommendations combine global & session interests

Music recommendations



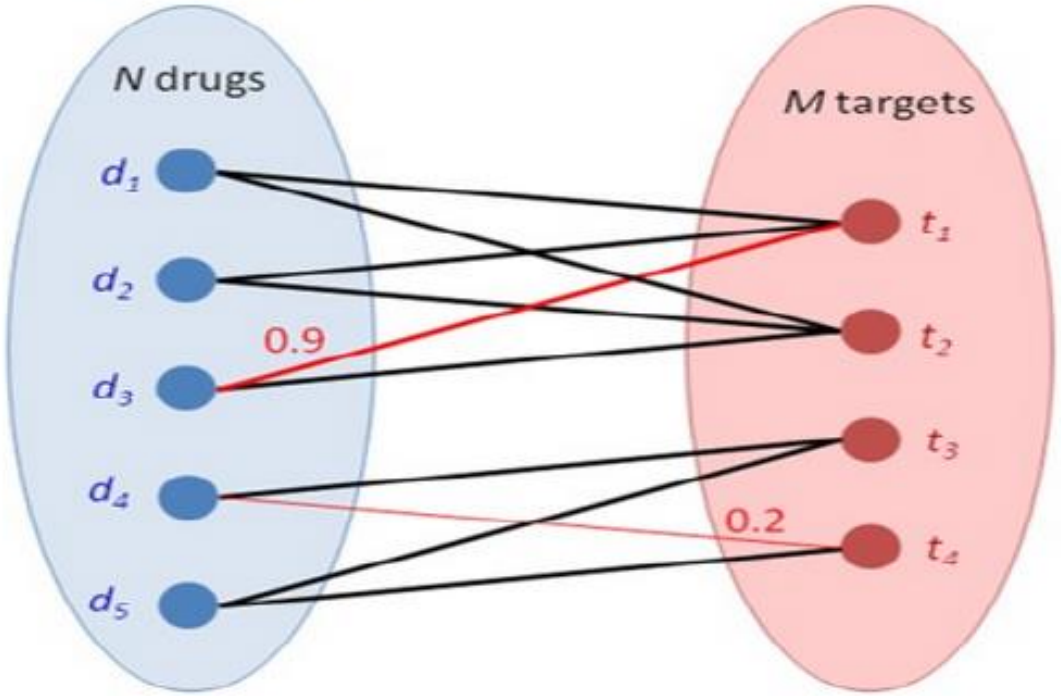
Recommendations form coherent & diverse sequence

Friend recommendations

Users and "items"
are of the same "type"

Drug-target interactions

Cobanoglu et al. '13



What drug should we "repurpose" for some disease?

What are the Challenges of Developing Recommendation Systems?

Type of feedback

- Explicit – user tells us what she likes

- Implicit – we try to infer what she likes from usage data

Top K versus diverse outputs

- Top K recommendations may be very redundant
  - *People who liked Rocky 1 also enjoyed Rocky 2, Rocky 3, Rocky 4, Rocky 5,…*

- Diverse recommendations
  - Users are multi-facetted & want to hedge our bets
  - Rocky 2, It's Always Sunny in Philadelphia, Gandhi

A new movies walks into a bar…

IN THEATERS

Cold-start problem: recommendations for new users or new movies

- Need side information about user/movie
  - A.K.A. features!

- Could also play 20-questions game…

That's so last year…

- Interests change over time…
  - Is it 1967?
  - Or 1977?
  - Or 1988?
  - Or 1998?
  - Or 2011?
- Models need flexibility to adapt to users
  - Macro scale
  - Micro scale
- And keep checking that system still accurate

macys.com

Scalability

For N users and M movies, some approaches take $O(N^3+M^3)$

- Not so good for billions of users…

Big focus has been on:

- Efficient implementations
- Fast exact & approximate methods as needed

Approaches to Building a Recommender System

Approaches to Building a Recommender System

- Solution 0: Popularity
- Solution 1: Content-based Filtering (Classification Model)
- Solution 2: Collaborative Filtering (People who bought this also bought …)
- Solution 3: Discovering Hidden Structure by Matrix Factorization
- Bringing it all together: Featurized Matrix Factorization

Solution 0: Popularity

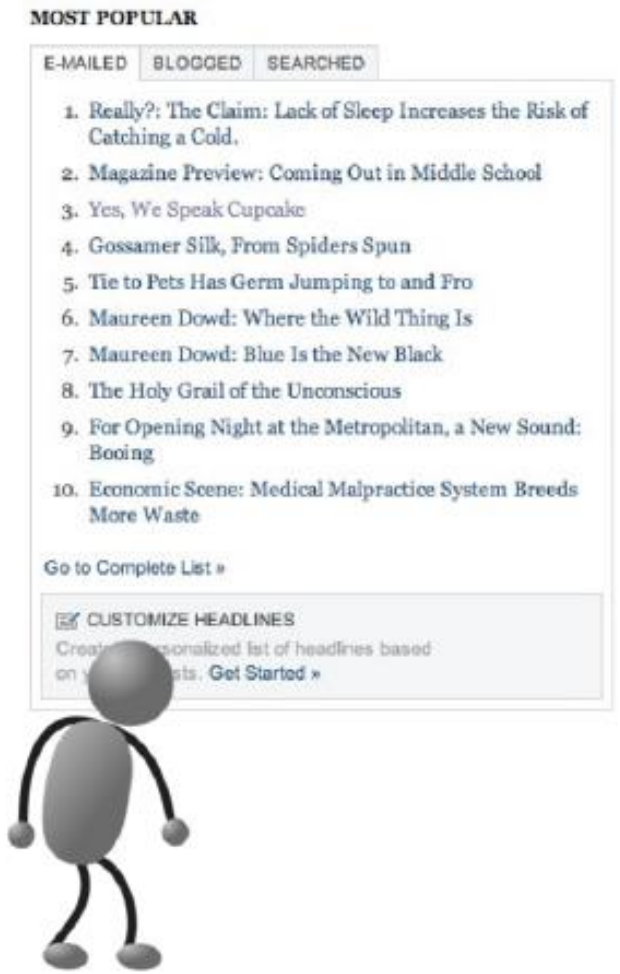
Simplest approach: Popularity

What are people viewing now?

- Rank by global popularity

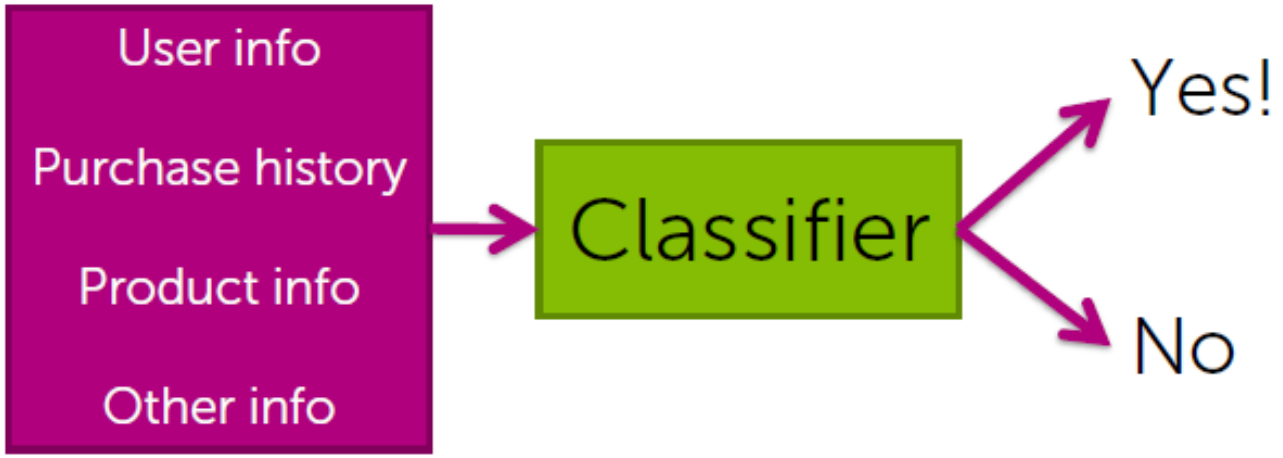
Limitation:

- No personalization



Solution 1: Content-based Filtering (Classification Model)

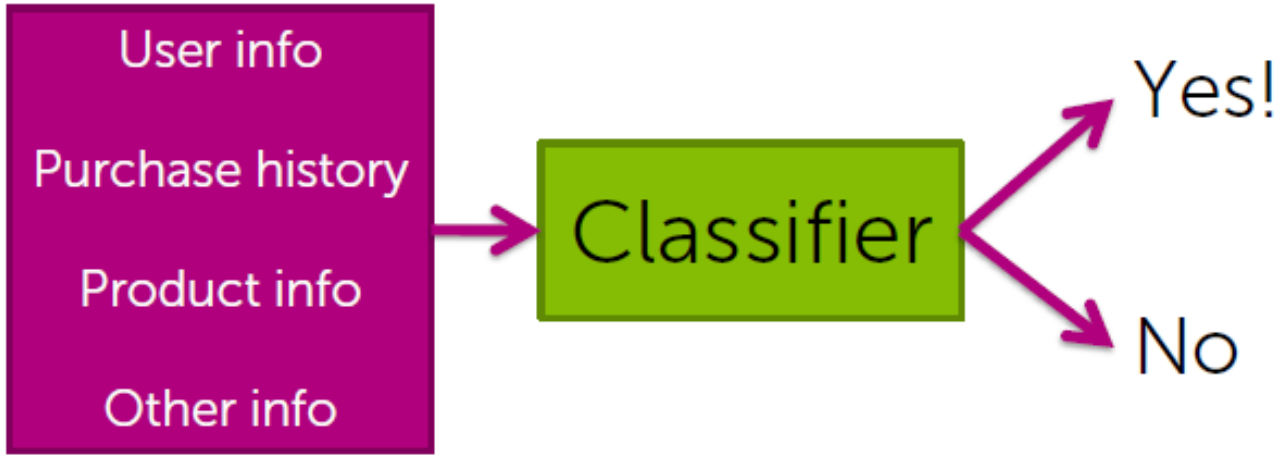
What's the probability I'll buy this product?



Pros:

- Personalized:
  Considers user info & purchase history
- Features can capture context:
  Time of the day, what I just saw,…
- Even handles limited user history: Age of user, …

Limitations of classification approach



- Features may not be available
- Often doesn't perform as well as collaborative filtering methods (next)

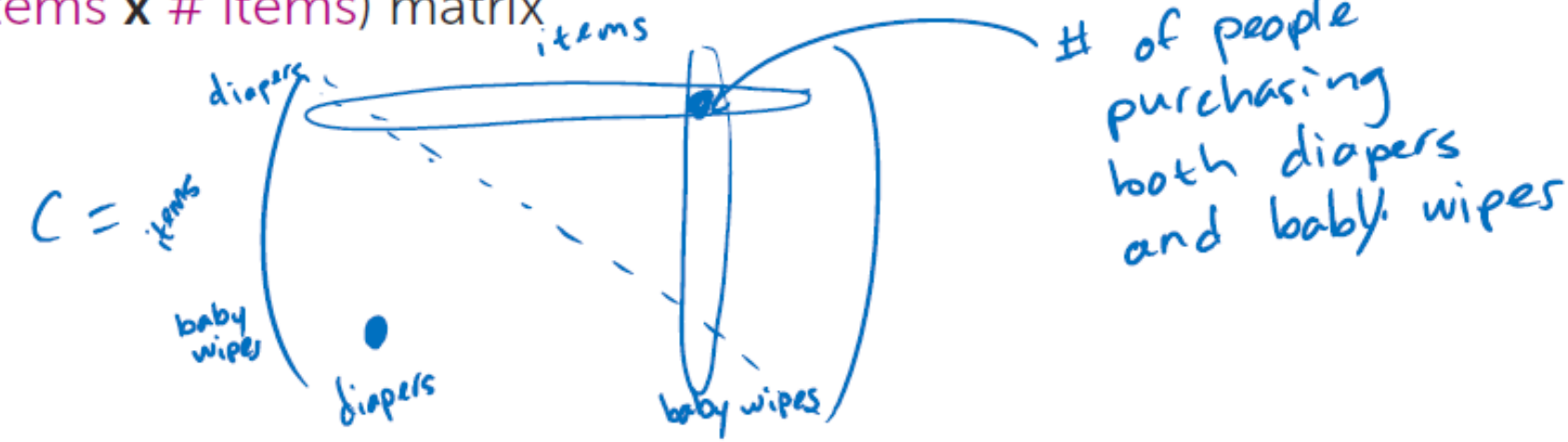
Solution 2: Collaborative Filtering

(People who bought this also bought …)

Co-occurrence matrix

- People who bought *diapers* also bought *baby wipes*

- **Matrix C:**
  store # users who bought both items *i & j*
  - (# items **x** # items) matrix



  - Symmetric: # purchasing *i & j* same as # for *j & i* ($C_{ij} = C_{ji}$)

Making recommendations using co-occurences

User  purchased *diapers*

1. Look at *diapers* row of matrix

2. Recommend other items with largest counts
   - *baby wipes*, *milk*, *baby food*,…

Co-occurrence matrix must be normalized

What if there are very popular items?

- Popular baby item:
  Pampers Swaddlers diapers

- For any baby item (e.g., ***i***=*Sophie giraffe* )
  large count C_{ij} for ***j***=*Pampers Swaddlers*

Result:

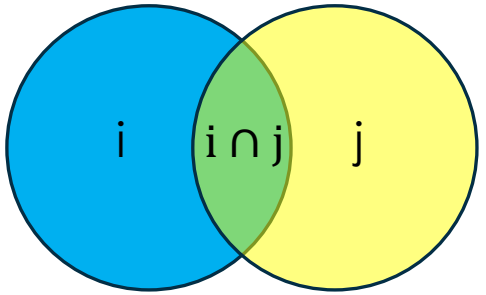
- Drowns out other effects
- Recommend based on popularity

Normalize co-occurrences: Similarity matrix

Jaccard similarity: normalizes by popularity

- Who purchased *i* and *j* *divided by* who purchased *i* or *j*

$$S_{ij} = \frac{\#\text{ purchased i} \cap \text{j}}{\#\text{ purchased i} \cup \text{j}}$$



i ∪ j

Many other similarity metrics possible, e.g., cosine similarity

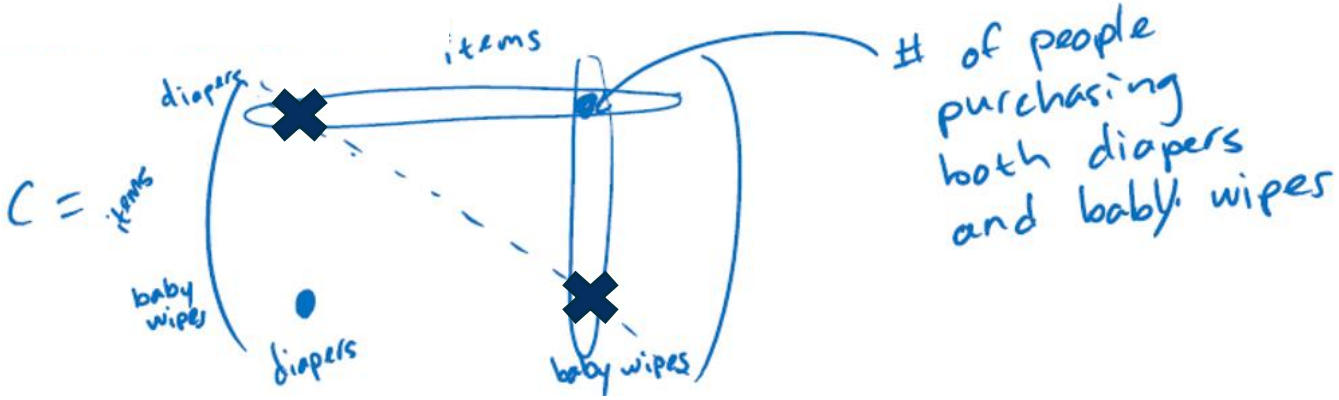
Normalize co-occurrences: Similarity matrix

Jaccard similarity: normalizes by popularity

- Who purchased *i* and *j* *divided by* who purchased *i* or *j*

$$S_{ij} = \frac{\#\text{ purchased i} \cap \text{j}}{\#\text{ purchased i} \cup \text{j}}$$

$$= \frac{C_{ij}}{C_{ii}+C_{jj}-C_{ij}}$$



Many other similarity metrics possible, e.g., cosine similarity

Limitations

- Only current page matters, no history
  - Recommend similar items to the one you bought
- What if you purchased many items?
  - Want recommendations based on purchase history

(Weighted) Average of purchased items

User  bought items {*diapers, milk*}

- Compute user-specific score for each item *j* in inventory by combining similarities:

  Score( *, baby wipes)* = ½ ($S_{baby\ wipes,\ diapers} + S_{baby\ wipes,\ milk}$)

- Could also weight recent purchases more

Sort *Score(*  *, j)* and find item *j* with highest similarity

Limitations

- Does **not** utilize:
  - context (e.g., time of day)
  - user features (e.g., age)
  - product features (e.g., baby vs. electronics)
- Scalability – similarity matrix M^2 size
- Cold start problem
  - What if a new user or product arrives?

Solution 3: Discovering Hidden Structure by Matrix Factorization

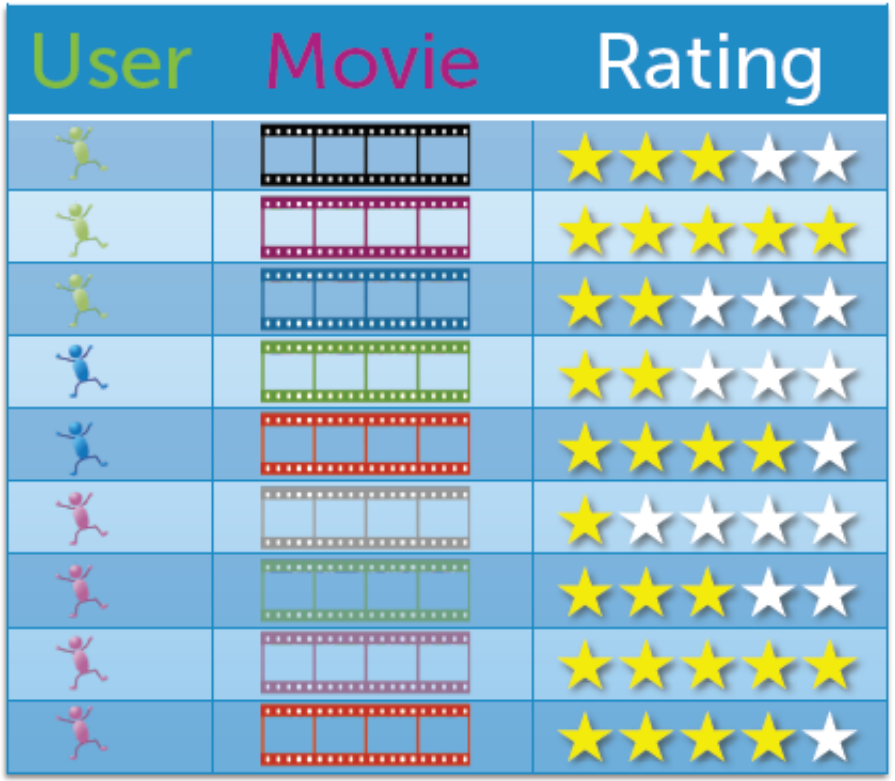
How Recommender Systems Work (Netflix/Amazon)

- Art of the Problem

https://www.youtube.com/watch?v=n3RKsY2H-NE

Movie recommendation

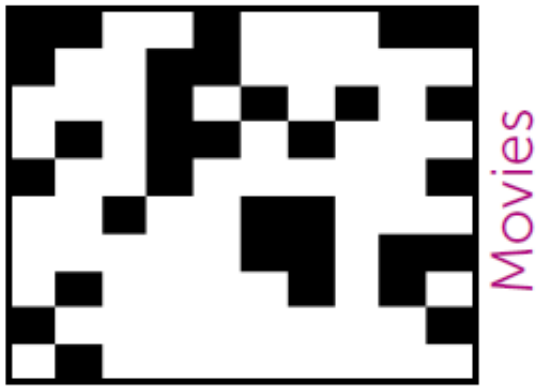
Users watch movies and rate them



Each user only watches a few of the available movies

Matrix completion problem

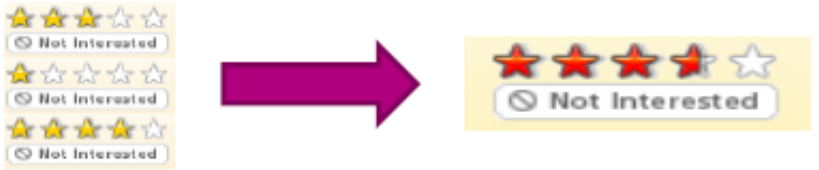
Rating =



Data: Users score some movies

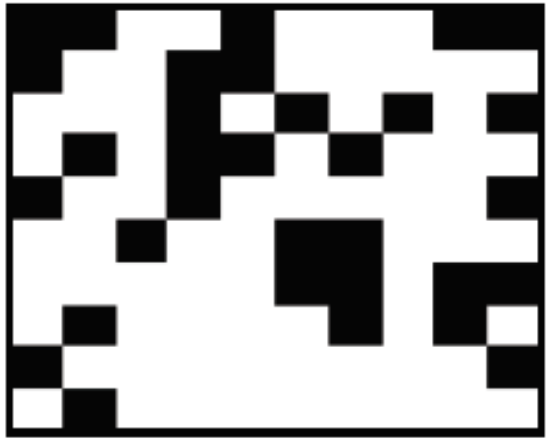
Rating(u,v) known for black cells
Rating(u,v) unknown for white cells

Goal: Filling missing data?



Matrix completion problem

Rating =



- Black cells indicate Rating(user,movie) known
- White cells indicate Rating(user,movie) unknown
- Each cell has values in {1,2,3,4,5}
- Goal: predict missing entries

Suppose we had *d* topics for each user & movie

- Describe movie *v*  with topics R_v
  - How much is it action, romance, drama,…
- Describe user *u*  with topics L_u
  - How much she likes action, romance, drama,…
- $\widehat{Rating}(u,v)$ is the product of the two vectors
- **Recommendations:** sort movies user hasn't watched by $\widehat{Rating}(u,v)$

Example: Predict a User's Rating:

- We can describe each movie **v** with feature vector $\mathbf{R_v}$
  - How much is the movie *action, romance, drama, …*
  - $\mathbf{R_v}$ = [0.3, 0.01, 1.5, …]
- We can describe each user **u** with feature vector $\mathbf{L_u}$
  - How much the user likes *action, romance, drama, …*
  - $\mathbf{L_u}$ = [2.3, 0.0, 0.7, …]
- Predict rating of movie user has not seen with **inner product**:

Rating(u,v) = 0.3*2.3 + 0.01*0.0 + 1.5*0.7 + …

Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Product recommendations

- Suppose the following features have been learned, which movie should we recommend to user #3?

User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Call this 4x2 matrix *L*

Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

Call this 3x2 matrix *R*

- Such prediction can be computed for all (user,movie) pairs
- And be written in a matrix form:

6	2	4
4	3	3
1	2	1
7	4	5

$=$

L

2	0
1	1
0	1
2	1

R^T

3	1	2
1	2	1

Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example

Suppose we have learned the following user and movie features

User ID	Feature
1	(2, 0)
2	(1, 1)
3	(0, 1)
4	(2, 1)

Movie ID	Feature vector
1	(3, 1)
2	(1, 2)
3	(2, 1)

Then we can predict what each user would rate each movie

L

2	0
1	1
0	1
2	1

R^T

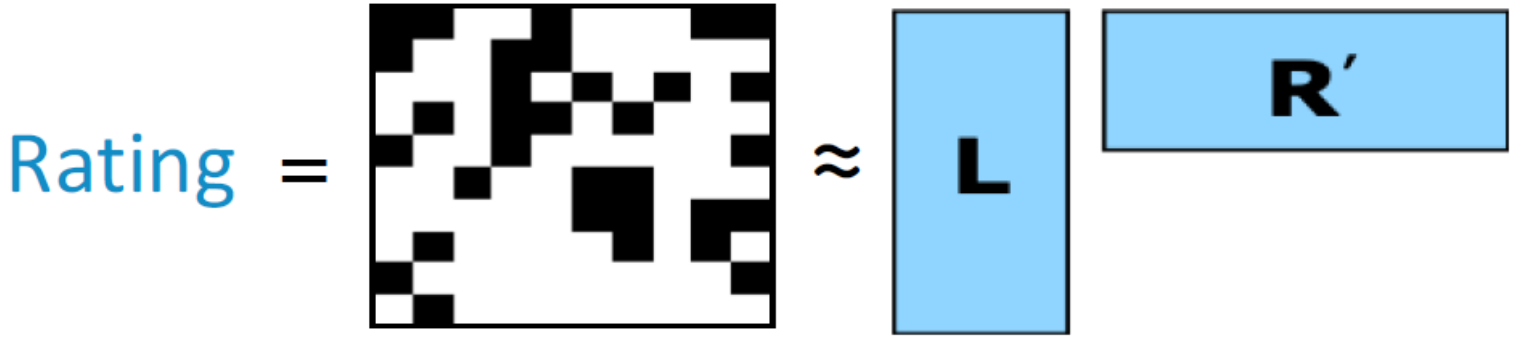
3	1	2
1	2	1

$=$

6	2	4
4	3	3
1	2	1
7	4	5

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Matrix Factorization



Find L and R that when multiplied, achieve predicted ratings that are close to the values that we have data for.

Our quality metric will be (could use others)

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \sum_{u,v:r_{uv}\neq ?} (L_u \cdot R_v - r_{uv})^2$$

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Unique Solution?

Is this problem well posed? Unfortunately, there is not a unique solution ☹

For example, assume we had a solution

6	2	4
4	3	3
1	2	1
7	4	5

$=$

L

2	0
1	1
0	1
2	1

R^T

3	1	2
1	2	1

Then doubling everything in L and halving everything in R is also a valid solution. The same is true for all constant multiples.

6	2	4
4	3	3
1	2	1
7	4	5

$=$

L

4	0
2	2
0	2
4	2

R^T

1.5	0.5	1.0
0.5	1.0	0.5

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Find $\hat{L}$ & $\hat{R}$

Remember, our quality metric is

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \sum_{u,v:r_{uv} \neq ?} (L_u \cdot R_v - r_{uv})^2$$

Gradient descent is not used much in practice to optimize this, since it is much easier to implement **coordinate descent** (i.e. Alternating Least Squares) to find $\hat{L}$ and $\hat{R}$

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Coordinate Descent

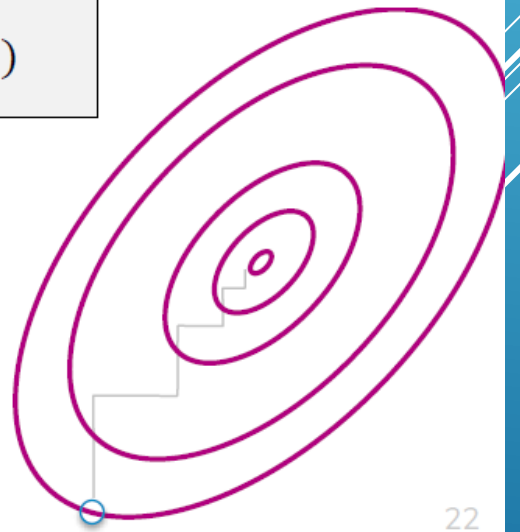
Goal: Minimize some function $g(w) = g(w_0, w_1, \ldots, w_D)$

Instead of finding optima for all coordinates, do it for one coordinate at a time!

Initialize $\hat{w} = 0$ *(or smartly)*
while not converged:
    pick a coordinate j
    $\hat{w}_j = \underset{w}{\operatorname{argmin}}\, g(\hat{w}_0, \ldots, \hat{w}_{j-1}, w, \hat{w}_{j+1}, \ldots, \hat{w}_D)$

To pick coordinate, can do round robin or pick at random each time.

Guaranteed to find an optimal solution under some constraints



Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Coordinate Descent for Matrix Factorization

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \sum_{u,v:r_{uv}\neq ?} (L_u \cdot R_v - r_{uv})^2$$

Fix movie factors R and optimize for L_u

First key insight:

Holding movies fixed, we can solve for each user separately!

For each user u

$$\hat{L}_u = \min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2$$

Second key insight:

Looks like linear regression!

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Overall Algorithm

Want to optimize

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \sum_{u,v:r_{uv}\neq ?} (L_u \cdot R_v - r_{uv})^2$$

Fix movie factors, and optimize for user factors separately

- Independent least squares for each user

$$\hat{L}_u = \min_{L_u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2$$

Fix user factors, and optimize for movie factors separately

- Independent least squares for each movie

$$\hat{R}_u = \min_{R_u} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2$$

System might be underdetermined: Use regularization

Converges to: local optima

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy

Using Results

Use movie factors $\hat{R}$ to discover "topics" for movie v: $\hat{R}_v$

Use user factors $\hat{L}$ to discover "topic preferences" for user u: $\hat{L}_u$

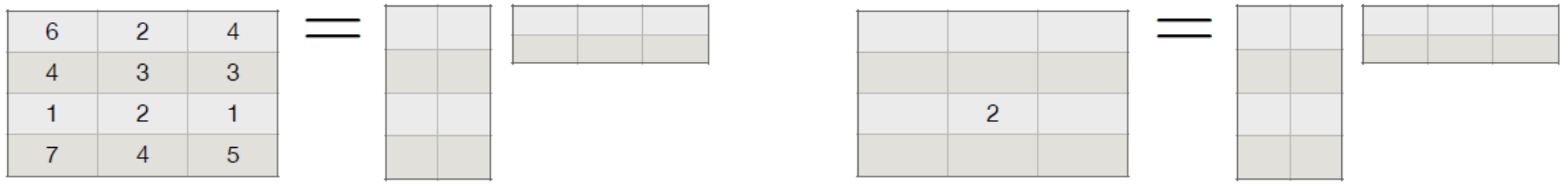
Predict how much a user u will like a movie v

$$\widehat{Rating}(u, v) = \hat{L}_u \cdot \hat{R}_v$$

Recommendations: Sort movies user hasn't watched by $\widehat{Rating}(u, v)$

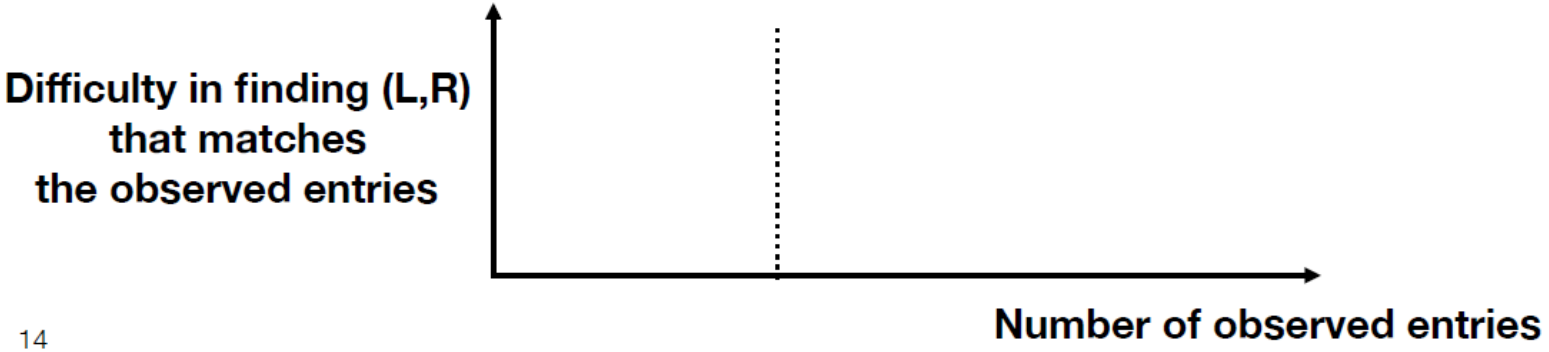
- Recommend movies with highest predicted rating

Reference: CSE/STAT 416 Intro to Machine Learning, Summer 2020, University of Washington, Instructor: Vinitra Swamy



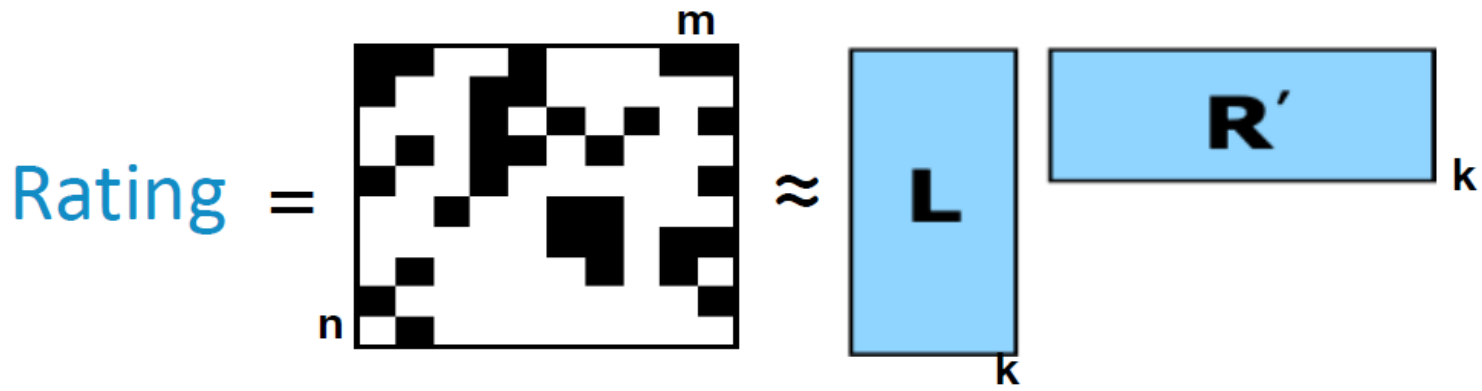
- But when can we solve this problem?
- That is how many entries do we need to see, in order for our prediction to be accurate?

- One extreme: suppose we observe all entries, then
  - Any factorization methods like singular value decomposition (SVD) will provide (one of the) correct factorizations
  - And, this correct, i.e. resulting $M = LR^T$
- Another extreme: suppose we observe one entry, then
  - It is easy to match the entry observed
  - But, most likely this is incorrect on the missing entries, i.e. $M \neq LR^T$



Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

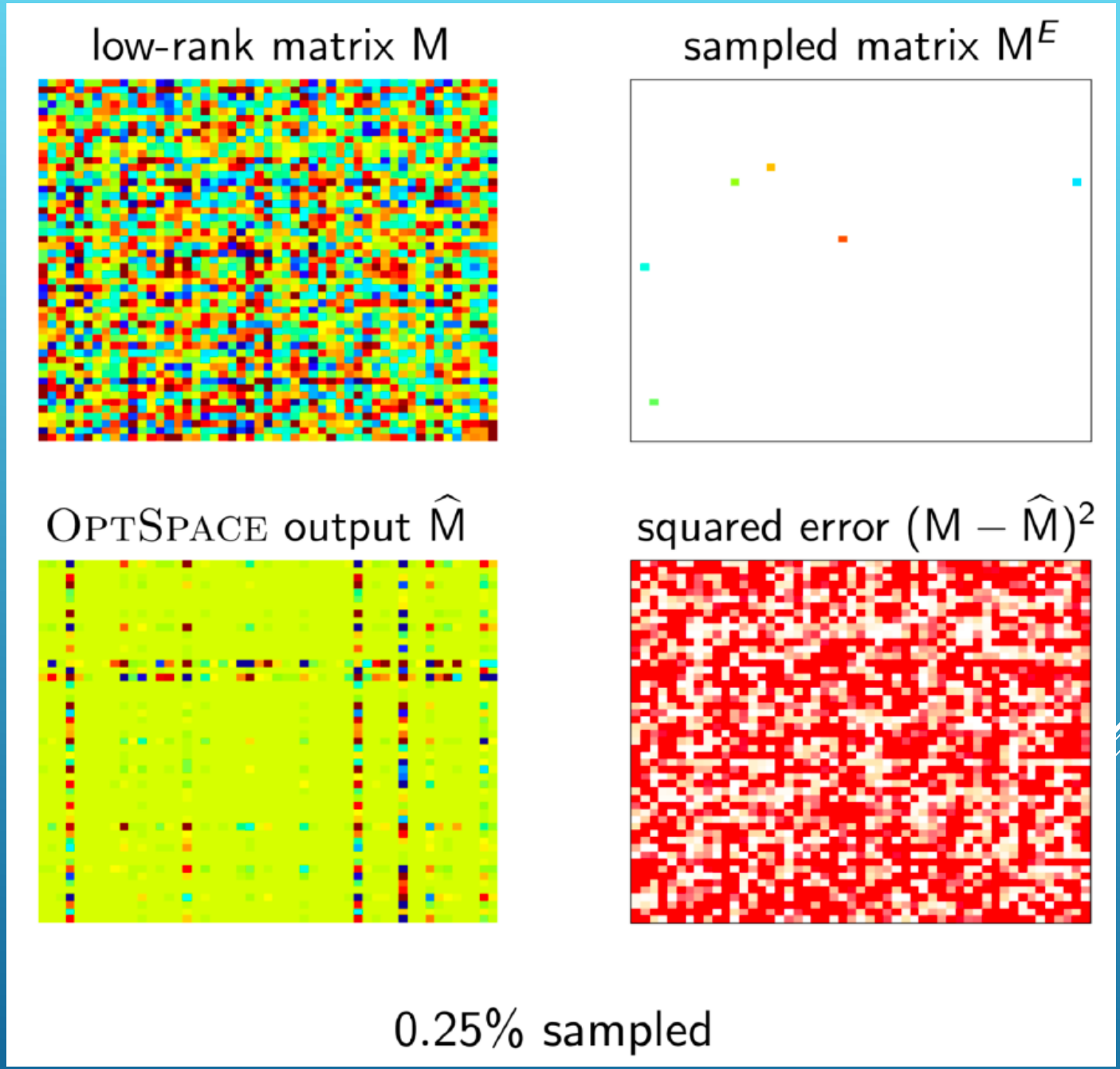
From factorization to Matrix completion



- If there are ***m*** movies and ***n*** users, and ***k*** topics, then how many parameters do we have in our factorization model ***L*** and ***R***?
  degrees-of-freedom = ***k*m+k*n**
- This is also sometimes called the degrees-of-freedom in the problem.
- How many entries do we observe if we have the full matrix?
- How many entries do you think we need, to accurately reconstruct the ground truth ***L*** and ***R*** that generated the data?

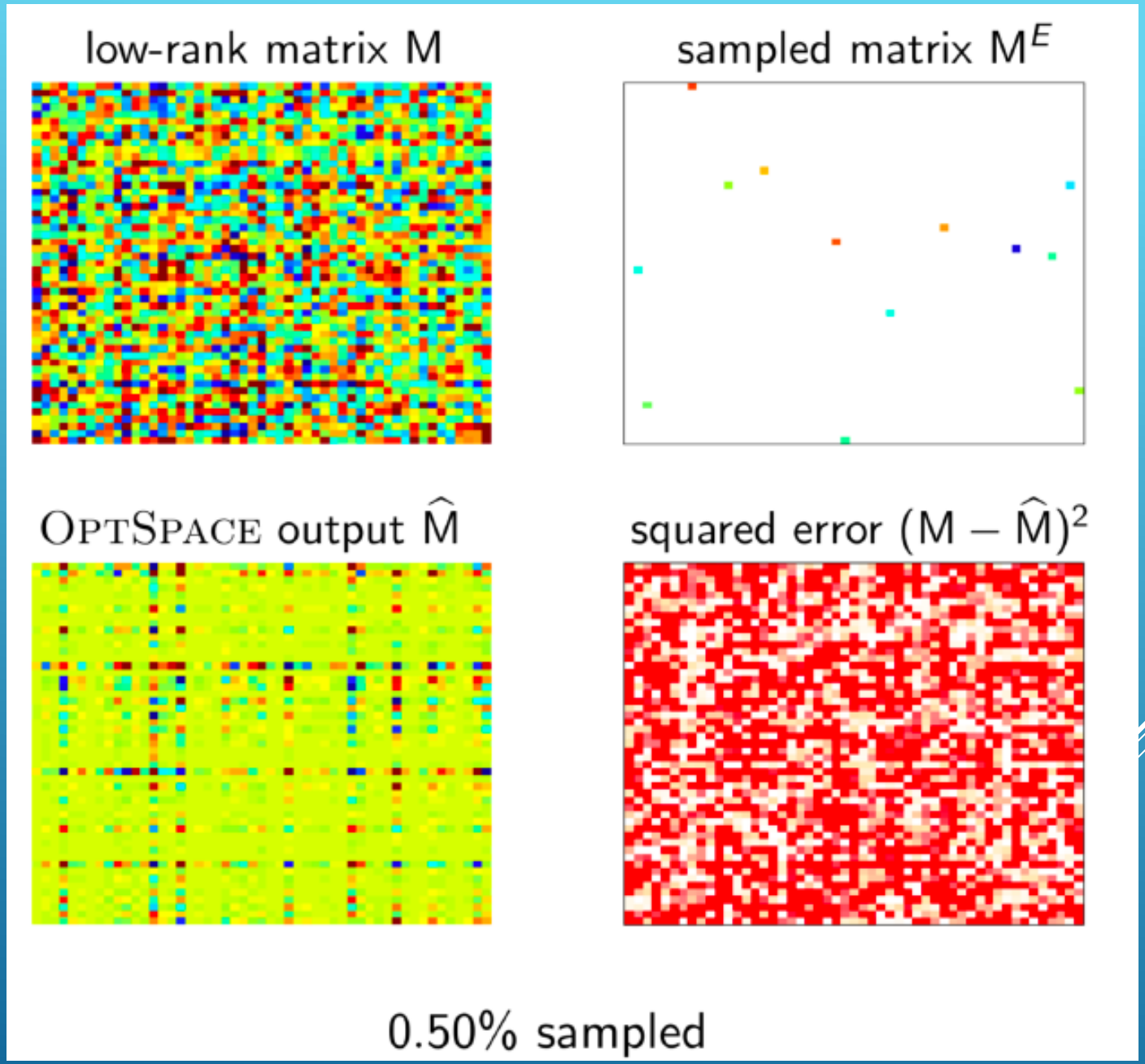
Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example: 2000 x 2000 rank-8 random matrix



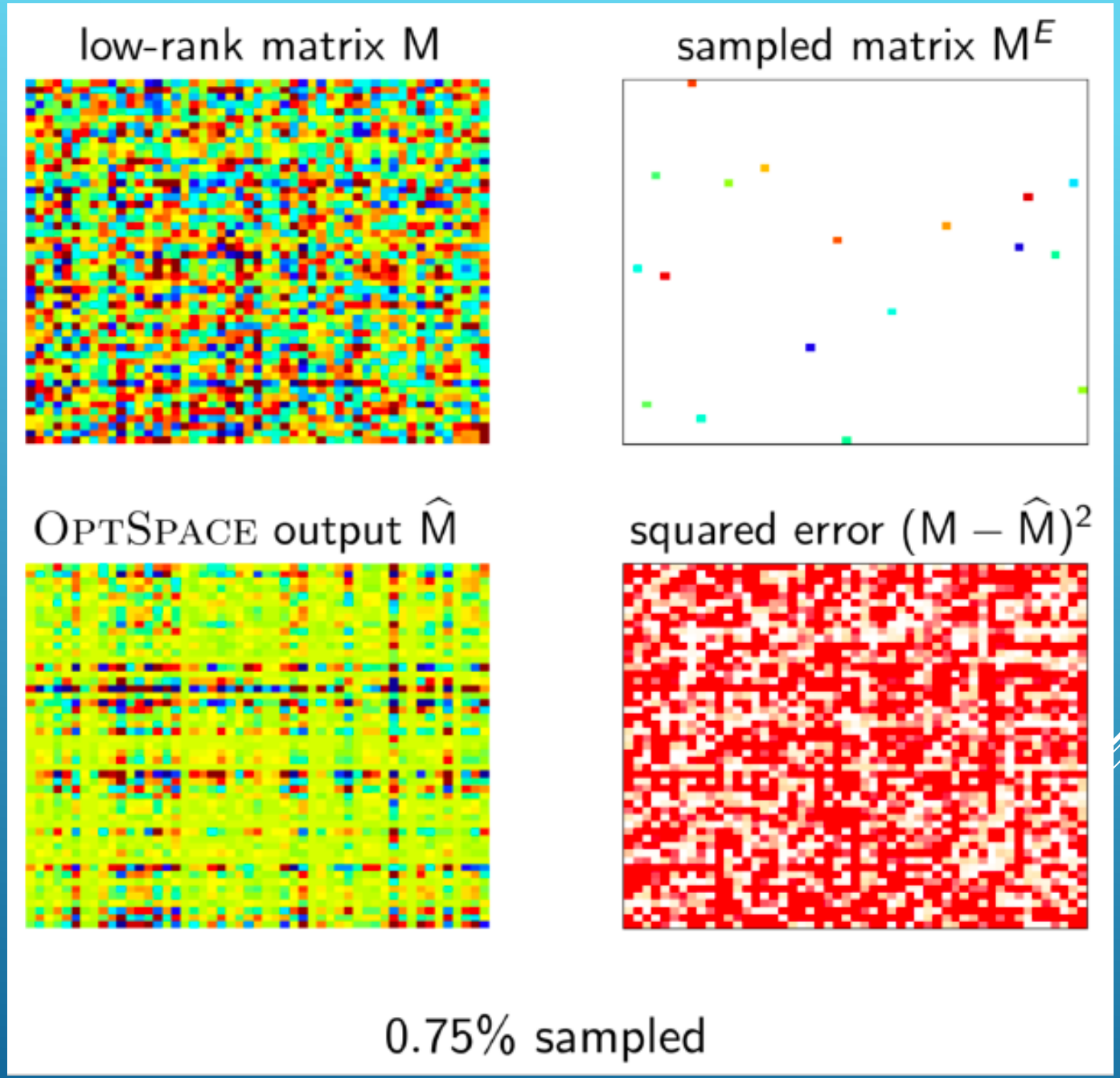
Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example: 2000 x 2000 rank-8 random matrix



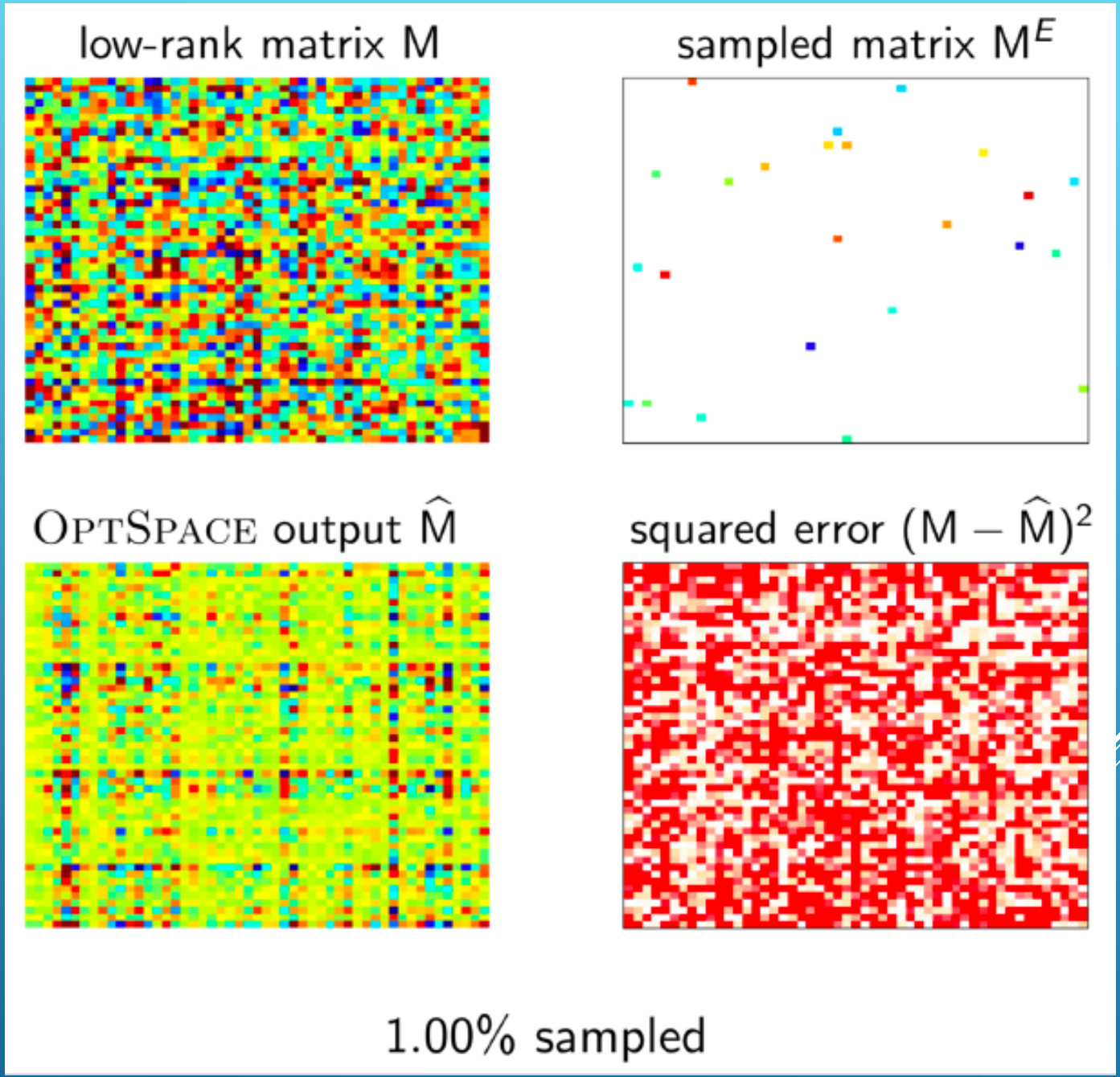
Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example: 2000 x 2000 rank-8 random matrix



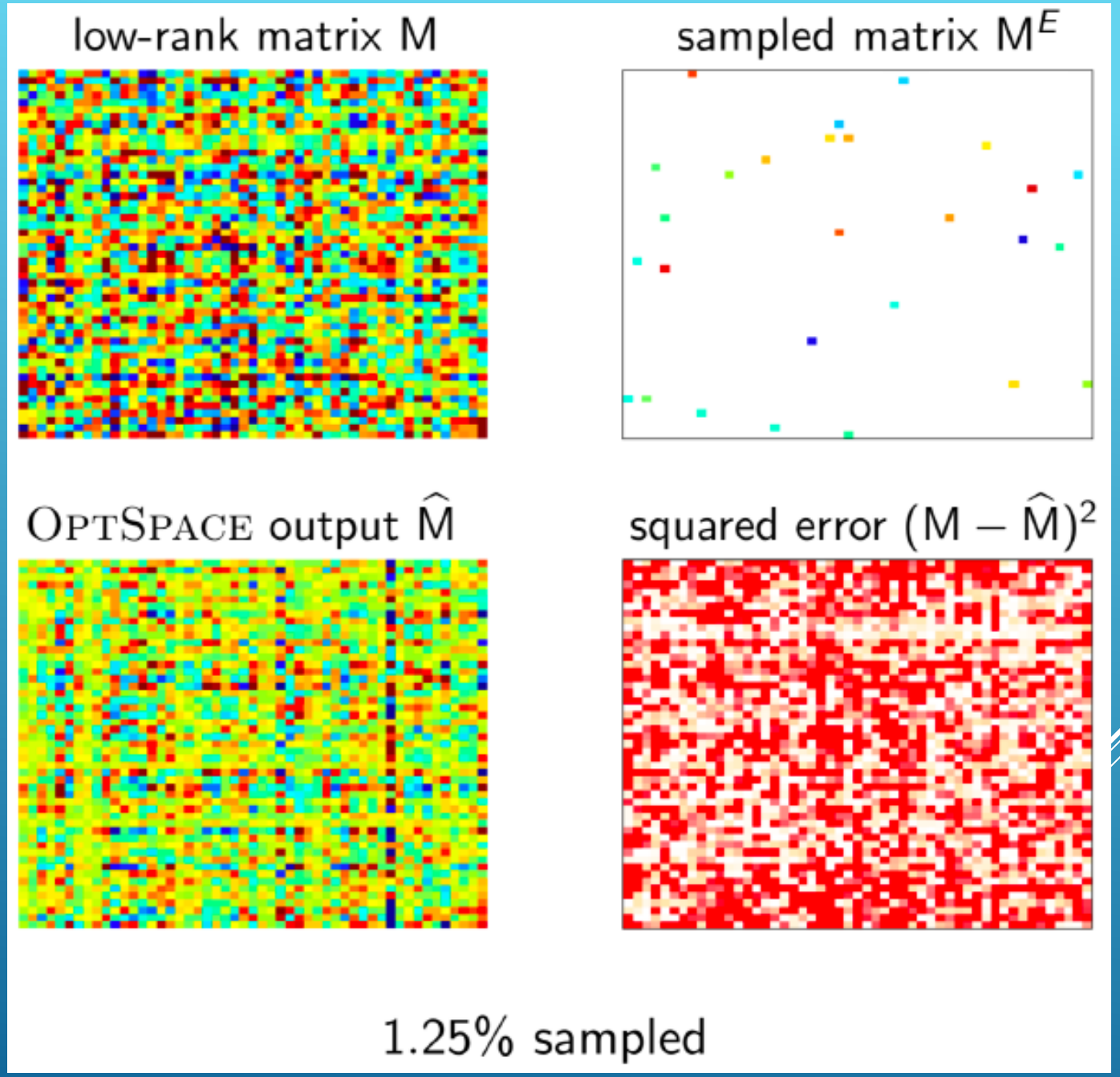
Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example: 2000 x 2000 rank-8 random matrix



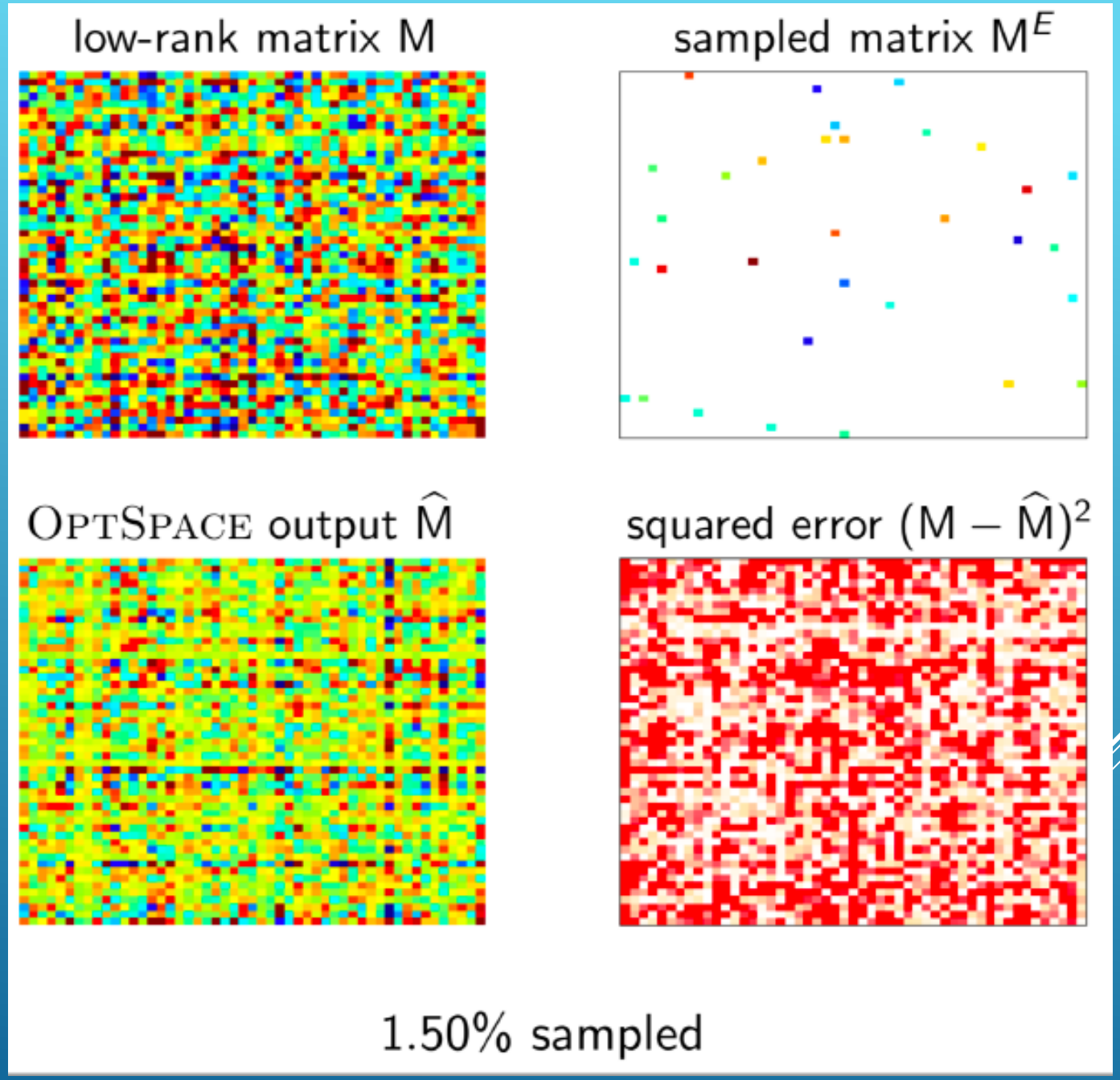
Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example:
2000 x 2000
rank-8 random
matrix



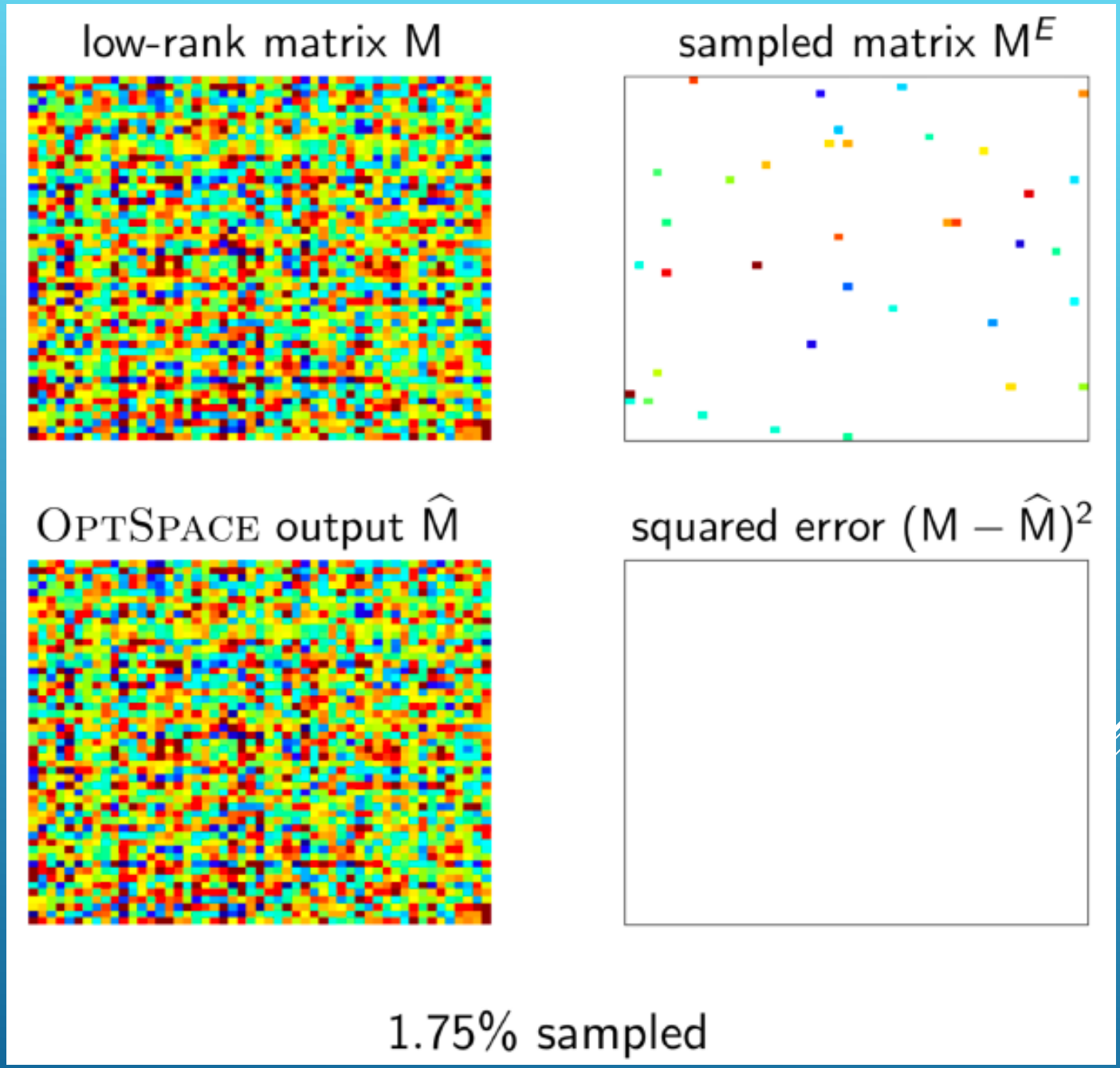
Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example:
2000 x 2000
rank-8 random
matrix



Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Example: 2000 x 2000 rank-8 random matrix

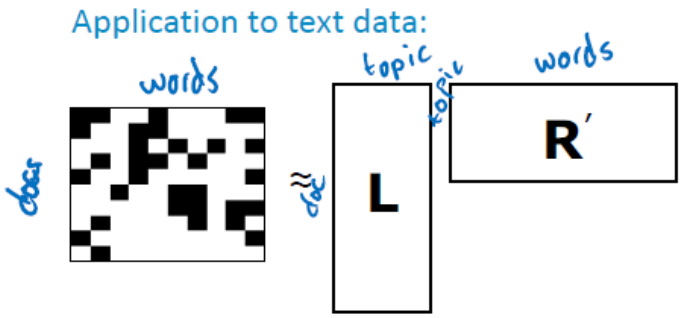


Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Application: recommendation systems

- Given partially observed ratings matrix
- Discover k topics, and k-dimensional user features L_u movie features R_v
- Predict how much a user will like a movie by $r_{uv}=L_u^T * R_v$
- Make recommendations based on the prediction

- Applied to Wikipedia

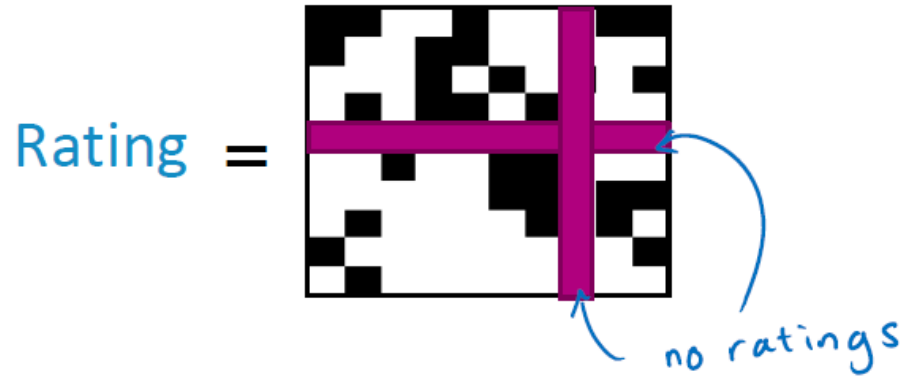


Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Bringing it all together: Featurized Matrix Factorization

Featured matrix factorization

- Limitations of matrix factorization
  - Cold-start problem
    - This model still cannot handle a new user or movie



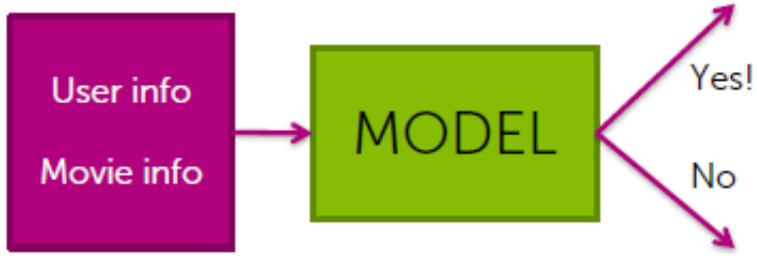
- As there is no observation for the entire row/column putting anything in that row has no penalty

$$\text{minimize}_{L,R} \sum_{u,v:r_{uv}\neq ?} \Big(\underbrace{(LR^T)_{uv}}_{L_u^T R_v} - r_{uv} \Big)^2$$

Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Combining features and discovered topics

- Features capture context
  - *Time of day, what I just saw, user info, past purchases,…*



- Discovered topics from matrix factorization capture groups of users who behave similarly
  - *Women from Seattle who teach and have a baby*

- **Combine** to mitigate cold-start problem
  - Ratings for a new user from features only
  - As more information about user is discovered, matrix factorization topics become more relevant

Collaborative filtering with specified features

- Create feature vector for each movie (often have this even for new movies):
- Define weights on these features for how much **all users** like each feature
- Fit linear model:
- Minimize:

Collaborative filtering with specified features

- Create feature vector for each movie (often have this even for new movies):

$\phi(v) = ($ 'action', 1994, Tarantino, ... $)$ — genre, year, director, ...

- Define weights on these features for how much **all users** like each feature

$w =$ vector of same length

- Fit linear model:

For all users, $r_{uv} \approx w \cdot \phi(v)$ — standard regression model

- Minimize:

$$\min_{w} \sum_{r_{uv}?} (w \cdot \phi(v) - r_{uv})^2 + \lambda_w \|w\| \leftarrow \text{LS, ridge, lasso}$$

Building in personalization

- Of course, users do *not* have identical preferences
- Include a user-specific deviation from the global set of user weights:

- If we don't have any observations about a user, use wisdom of the crowd

- As we gain more information about the user, forget the crowd

- Can add in user-specific features, and cross-features, too

Building in personalization

- Of course, users do *not* have identical preferences
- Include a user-specific deviation from the global set of user weights:

$r_{uv} \approx (w + w_u) \cdot \phi(v)$ — personalization to user u = deviation from crowd vector w

- If we don't have any observations about a user, use wisdom of the crowd

Initialize $w_u = 0 \Rightarrow r_{uv} \approx w \cdot \phi(v)$

- As we gain more information about the user, forget the crowd

w_u more informed (personalization)

- Can add in user-specific features, and cross-features, too

$\phi(u) = ($ age, gender, education $) = (25, F, MSc, \dots)$

$\phi(u,v) = (\dots \phi(u) \dots, \dots \phi(v) \dots,$ cross features $)$

Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

Featurized matrix factorization—
A combined approach

Feature-based approach:

- Feature representation of user and movies fixed
- Can address cold-start problem

Matrix factorization approach:

- Suffers from cold-start problem
- User & movie features are learned from data

A unified model:

$$r_{uv} \approx L_u \cdot R_v + (w + w_u) \cdot \phi(u, v)$$

Solve via coord. desc., grad. desc., etc.

Reference: CSE/STAT 416 Intro to Machine Learning, Spring 2019, University of Washington, Instructor: Sewoong Oh

INTRODUCING RECOMMENDATION SYSTEMS

Scott O'Hara

Metrowest Developers Machine Learning Group

03/31/2021