Say hi or ask questions in chat before/during/after class

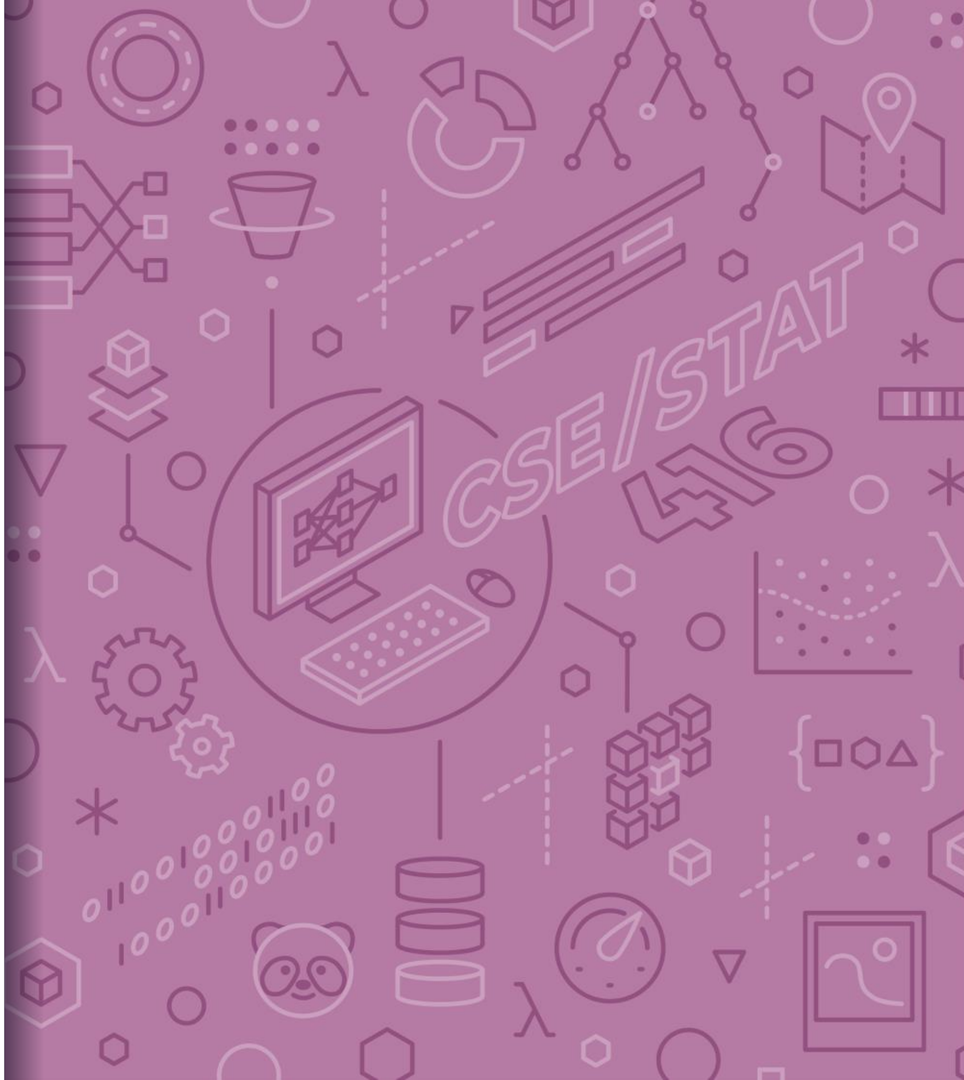
CSE/STAT 416

Recommender Systems

Hunter Schafer
Paul G. Allen School of Computer Science & Engineering
University of Washington

May 17, 2021

Music: HAIM



Last Time…

Co-occurrence Matrix

Solution 2

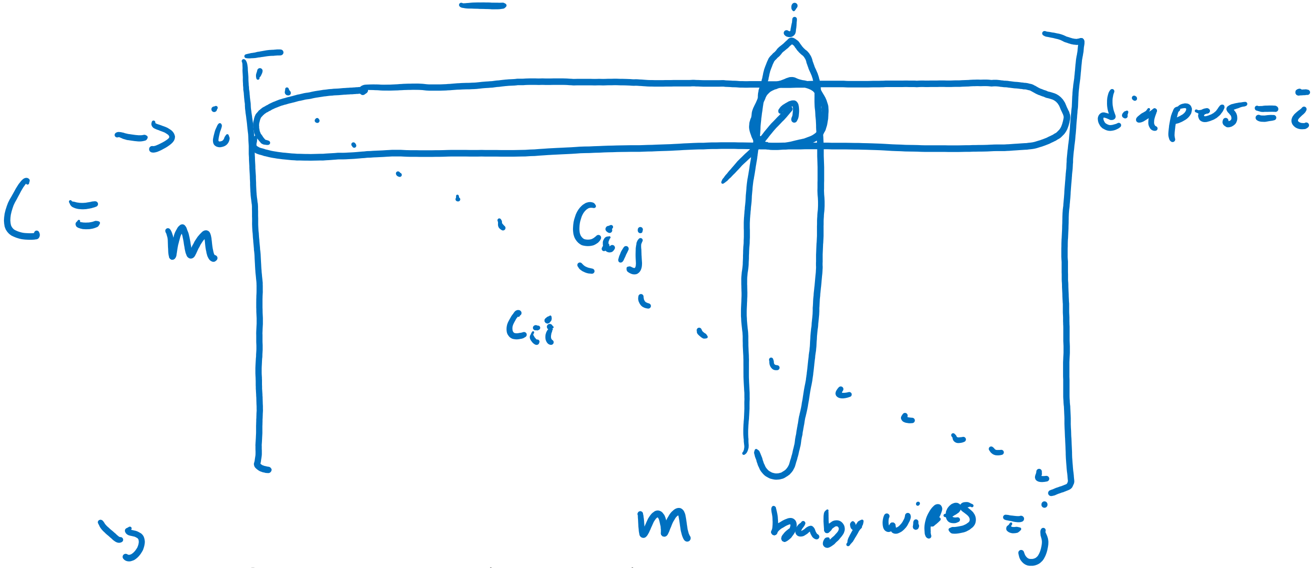
Co-occurrence Matrix

m = # items
n = # users

Idea: People who bought this, also bought …

- E.g. people who buy diapers also buy baby wipes

Make **co-occurrence matrix** $C \in \mathbb{R}^{m \times m}$ (m is the number of items) of item purchases. C_{ij} = # of users who bought both item i and j



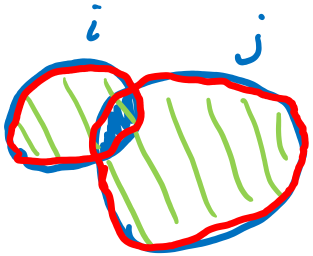
C will be symmetric ($C_{ij} = C_{ji}$)

Normalization

$C \rightarrow S$

The count matrix C needs to normalized, otherwise popular items will drown out others (will just reduce to popularity).

Normalize the counts by using the **Jaccard similarity** instead



$$S_{ij} = \frac{\#\ \text{purchased } i \textbf{ and } j}{\#\ \text{purchased } i \textbf{ or } j}$$

$$= \frac{C_{ij}}{T_i + T_j - C_{ij}}$$

Where T_K = # of people who bought item K

Could also use something like Cosine similarity, but Jaccard is popular

Analysis

Pros:

- It personalizes to the user

Cons

- Does **not** utilize
  - Context (e.g. time of day)
  - User features (e.g. age)
  - Product features (e.g. baby vs electronics)
- Scalability
  - Similarity is size m^2 where m is the number of items
- Cold start problem

Matrix Factorization

Solution 4

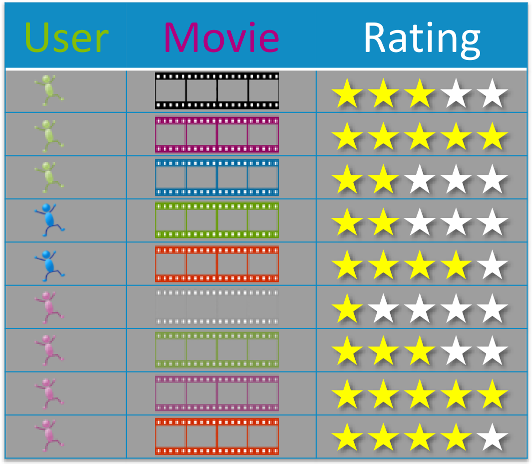
Matrix Completion

Want to recommend movies based on user ratings for movies.

Challenge: Users have rated relatively few of the entire catalog

Can think of this as a matrix of users and ratings with missing data!

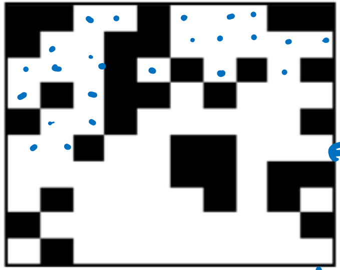
Input Data



User 1	5				3
User 2		2		4	
User 3			3		
User 4	1				
User 5			4		
User 6		5			2

=

user



movie

ratings ∈ [1, 5]

black = known rating

white = unknown rating (?)

Assumption

Matrix completion is an impossible task without some assumptions on data (unknowns could be anything otherwise).

Assume: There are k types of movies (e.g. action, romance, etc.) which users have various interests in.

This means we can describe a movie $\boldsymbol{v}$ with feature vector $\boldsymbol{R_v}$

- How much is the movie action, romance, drama, …
- Example: $\boldsymbol{R_v} = [0.3, \ 0.01, \ 1.5, \ \ldots] \in \mathbb{R}^k$

action mystery romance

We can describe each user $\boldsymbol{u}$ with a feature vector $\boldsymbol{L_u}$

- How much she likes action, romance, drama, ….
- Example: $L_u = [2.3, \ 0, \ 0.7, \ \ldots] \in \mathbb{R}^k$

action mystery romance

Estimate rating for user $\boldsymbol{u}$ and movie $\boldsymbol{v}$ as

$$\widehat{Rating}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{L_u} \cdot \boldsymbol{R_v} = 2.3 \cdot 0.3 + 0 \cdot 0.01 + 0.7 \cdot 1.5 + \ \ldots$$

Example

"factors"

Suppose we have learned the following user and movie features using 2 features $k=2$

L_u

	User ID	Feature
u_1	1	(2, 0)
u_2	2	(1, 1)
u_3	3	(0, 1)
u_4	4	(2, 1)

$L: 4 \times 2 \quad (n \times k)$

R_v

	Movie ID	Feature vector
v_1	1	(3, 1)
v_2	2	(1, 2)
v_3	3	(2, 1)

$R: 3 \times 2 \quad (m \times k)$

Then we can predict what each user would rate each movie

$$
\underset{n \times k}{\overset{L}{\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix}}}
\underset{k \times m}{\overset{R^T}{\begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}}}
=
\underset{n \times m}{\begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix}}
$$

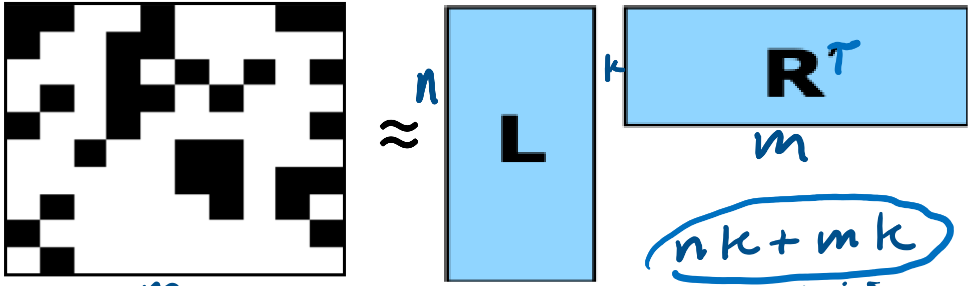
$\text{Rating}(u_i, v_j)$

5 4 4

Matrix Factorization

n users

Rating =



n · m entries total

m movies

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)

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

entries we have ratings for

predicted rating

actual rating

Unique Solution?

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

For example, assume we had a solution

$$\begin{array}{cccc} & & L & R^T \\ \begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix} & = & \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} & \begin{bmatrix} 3 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix} \end{array}$$

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

$$\begin{array}{cccc} & & L & R^T \\ \begin{bmatrix} 6 & 2 & 4 \\ 4 & 3 & 3 \\ 1 & 2 & 1 \\ 7 & 4 & 5 \end{bmatrix} & = & \begin{bmatrix} 4 & 0 \\ 2 & 2 \\ 0 & 2 \\ 4 & 2 \end{bmatrix} & \begin{bmatrix} 1.5 & 0.5 & 1.0 \\ 0.5 & 1.0 & 0.5 \end{bmatrix} \end{array}$$

Coordinate Descent

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

Remember, our quality metric is

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

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 $\hat{R}$

Coordinate Descent

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

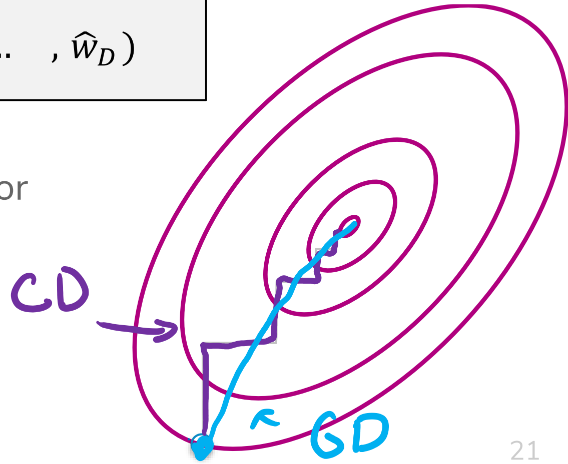
Instead of finding optima for all coordinates, do it for one coordinate at a time!

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

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

Guaranteed to find an optimal solution under some constraints

strongly convex, smooth



Coordinate Descent for Matrix Factorization

V_u = set of all movies rated by user u

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

Fix movie factors R and optimize for L_u

First key insight:

$$\underset{L_1,\ldots,L_n}{\operatorname{argmin}} \sum_{u,v: r_{uv} \neq ?} (L_u \cdot R_v - r_{uv})^2$$

$$= \underset{L_1,\ldots,L_n}{\operatorname{argmin}} \sum_{u} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2$$

Independent optimization for each user u!

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

Coordinate Descent for Matrix Factorization

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

For each user u

true data

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

"input" (fixed)

parameters

Second key insight:

Looks like linear regression!

$$\underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (w^T h(x_i) - y_i)^2$$

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 R, and optimize for user factors separately

- Independent least squares for each user

$$\hat{L}_u = \underset{L_u}{\operatorname{argmin}} \sum_{v \in V_u} (L_u \cdot R_v - r_{uv})^2 + \lambda_u \|L_u\|$$

Fix user factors, and optimize for movie factors separately

- Independent least squares for each movie

$$\hat{R}_v = \underset{R_v}{\operatorname{argmin}} \sum_{u \in U_v} (L_u \cdot R_v - r_{uv})^2 + \lambda_v \|L_u\|$$

Users who rated movie v

Repeatedly do these steps until convergence (to local optima)

System might be underdetermined: Use regularization

Poll Everywhere

Think

1.5 minutes

~~pollev.com/cs416~~

Consider we had the ratings matrix

	Movie 1	Movie 2
User 1	4	?
User 2	?	2

During one step of optimization, user and movie factors are

$k=3$

	User Factors
User 1	[1, 2, 1]
User 2	[1, 1, 0]

	Movie Factors
Movie 1	[2, 1, 0]
Movie 2	[0, 0, 2]

Two questions:

What is the current residual sum of squares loss? (number)

If the next step of coordinate descent updates the *user factors*, which factors would change?

- User 1
- User 2
- User 1 and 2
- None

Poll Everywhere

Think

3 minutes

pollev.com/cs416

Consider we had the ratings matrix

	Movie 1	Movie 2
User 1	4 4	? 2
User 2	? 3	2 0

$\widehat{Rating}(U_1, V_1) = L_1 \cdot R_1 = 1\cdot 2 + 2\cdot 1 + 1\cdot 0 = 4$

During one step of optimization, user and movie factors are

	User Factors
User 1	[1, 2, 1]
User 2	[1, 1, 0]

	Movie Factors
Movie 1	[2, 1, 0]
Movie 2	[0, 0, 2]

Two questions:

$RSS(L,R) = (4-4)^2 + (2-0)^2 = 4$

What is the current residual sum of squares loss? (number)

If the next step of coordinate descent updates the *user factors*, which factors would change?

- User 1 ✗
- User 2 ✓
- User 1 and 2
- None

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

Error is 0 for user 0,
no update

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

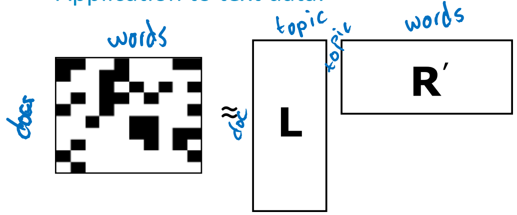
Brain Break

Topics

The “features” found by matrix factorization don’t always correspond to something meaningful (like film genre), but sometimes they do!

- Remember, the exact values are meaningless since we can scale them an infinite number of ways, but directions might mean something

Application to text data:



Poll Everywhere

Think

1 min

~~pollev.com/cs416~~

Which of the following are true about matrix factorization for recommendation systems?

A. Provides personalization ✓
B. Captures context (e.g. time of day) ✗
C. Solves the cold start problem ✗

1:00

Blending Models: Featurized Matrix Factorization

Final Solution

$||L||_F^2$ is the L2 norm equivalent for matrices

Cold Start Again

Consider a new user u' and we want to predict their ratings

No previous ratings for them so: for all v, $r_{u'v} = ?$

Objective

$$\hat{L}, \hat{R} = \underset{L,R}{\operatorname{argmin}} \cancelto{0}{\sum_{u,v:r_{uv} \neq ?} (L_u \cdot R_v - r_{uv})^2} + \lambda_U ||L||_F^2 + \lambda_V ||R||_F^2$$

Optimal user factor: $L_{u'} = 0$ because there is only penalty

Therefore, for all v, $\hat{r}_{u'v} = 0$ which seems like a problem

Blend Models

Idea: Learn a model to supplement the matrix factorization model!

Create a feature vector for each movie

$$h(v) = \begin{pmatrix} \text{genre} & \text{year} & \text{director} & \\ \text{'action'}, & 1994, & \text{'...'}, & \dots \end{pmatrix}$$

Define weights on these features for **all users** (global)

$$w_G \in \mathbb{R}^d$$

Fit linear model : $\hat{r}_{uv} = w_G^T h(v)$

$$\hat{w}_G = \underset{w}{\operatorname{argmin}} \sum_{u,v : r_{uv} \neq ?} \left(w_G^T h(v) - r_{uv}\right)^2 + \lambda ||w||$$

Add Personalization

Of course, not all users have same preferences.

Include a user-specific deviation from global model

$$\hat{r}_{uv} = (\hat{w}_G + \hat{w}_u)^T h(v)$$

New user $\Rightarrow \hat{w}_u = 0$ but can be updated over time

Can also add user specific features to model

$$h(u,v) = \begin{pmatrix} \text{genre} & \text{year} & & \text{gender} & \text{age} \\ \text{action}, & 1994, & \ldots, & F, & 25, \ldots \end{pmatrix}$$

Learn $\hat{w}_G, \hat{w}_u$ for $h(u,v)$

Featurized Matrix Factorization

Feature-based approach

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

Matrix factorization approach

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

A unified model

$$\hat{r}_{uv} = f\Big(\underbrace{\hat{L}_u \cdot \hat{R}_v}_{MF},\ \underbrace{(\hat{w}_G + \hat{w}_u)^T h(u,v)}_{FB}\Big)$$

f is some arbitrary method to combine results

Evaluating Recommendations

Accuracy?

Could we use classification accuracy to identify which recommender system is performing best?

- We don't really care to predict what a person **does not** like
- Instead, we want to find the relatively few items from the catalog that they will like
- Similar to a class imbalance

Instead, we want to look at our set of recommendations and ask:

- How many of our recommendations did the user like? Precision
- How many of the items that the user liked did we recommend? Recall

Sound familiar?

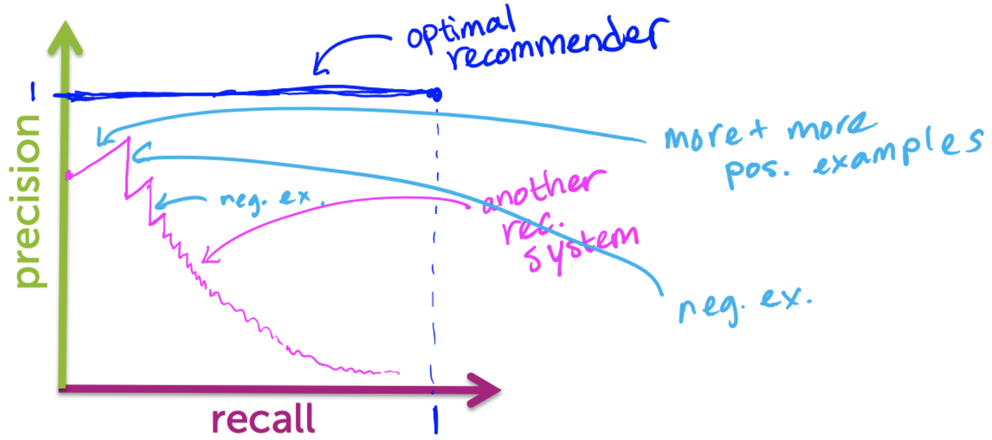
Precision - Recall

Precision and recall for recommender systems

$$precision = \frac{\#\ liked\ \&\ shown}{\#\ shown}$$

$$recall = \frac{\#liked\ \&\ shown}{\#liked}$$

For a given recommender system, plot precision and recall for different number of recommended items



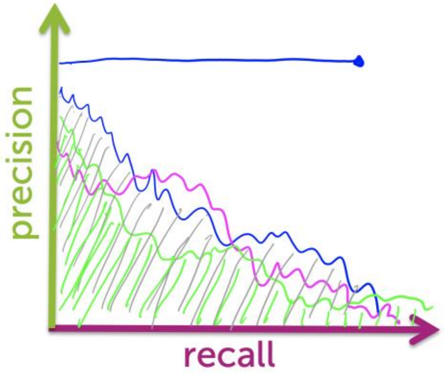
Comparing Algorithms

In general, it depends

- What is true always is that for a given precision, we want recall to be as large as possible (and vice versa)
- What target precision/recall depends on your application

One metric: area under the curve (**AUC**)

Another metric: Set desired recall and maximize precision (**precision at k**)



Optional: Offline Replay

Other approach: A/B Testing

Challenge: Deploying a new recommender system in-the-wild can be scary! Might not have confidence in how it will do.

One clever idea: use a simple model to start and record it's predictions and if the user interacted with the content (e.g., watched movie). Use this data of the model in deployment to test new models without needing to deploy them!

Test your new model on this logged data, if your model makes the same prediction as the simple model did originally, use that as a test example!

user pred. label

Deployed : (U_1, A, X) $(U_2, B, ✓)$ $(U_3, B, ✓)$

Experimental : ~~$(U_1, B, ✓)$~~ $(U_2, B, ✓)$ (U_3, B, X)

Recap

Now you can:

- Describe the goal of a recommender system
- Provide examples of applications where recommender systems are useful
- Implement a co-occurrence based recommender system
- Describe the input (observations, number of “topics”) and output (“topic” vectors, predicted values) of a matrix factorization model
- Implement a coordinate descent algorithm for optimizing the matrix factorization objective presented
- Exploit estimated “topic” vectors to make recommendations
- Describe the cold-start problem and ways to handle it (e.g., incorporating features)
- Analyze performance of various recommender systems in terms of precision and recall
- Use AUC or precision-at-k to select amongst candidate algorithms