

# Recommenders

Mark Llorente

(heavily based on the slides/notes of Ryan Henning, Dan Becker, and Giovanna)



- What kind of data do we use for recommenders?
- High-level approaches to building recommenders:
  - Content-based
  - Collaborative filtering
  - Matrix factorization
- How do we evaluate our recommender models?
- How to deal with “cold start”?
- What are the computational performance concerns?
- Where have you seen or used recommenders?

# What does our dataset look like?

User	Item				
	A	B	C	D	...
	Al	1	?	2	?
	Bob	?	2	3	4
	Cat	3	?	1	5
	Dan	?	2	?	?
	Ed	2	?	?	1
	...				

We have explicit ratings, plus a bunch of missing values.

← Btw, we call this the *utility matrix*.

# What does our dataset look like?

User	Item				
	A	B	C	D	...
	Al	0	1	0	1
	Bob	0	0	1	0
	Cat	0	1	1	1
	Dan	1	0	0	1
	Ed	0	1	0	0
	...				

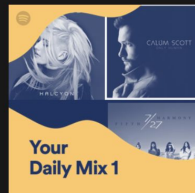
We have implicit feedback,  
and no missing values.

← Btw, we also call  
this the *utility  
matrix*.

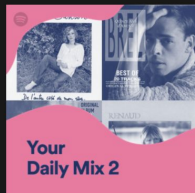
# Where are recommenders used?

## Your Daily Mixes

Play the music you love, without the effort. Packed with your favorites and new discoveries.



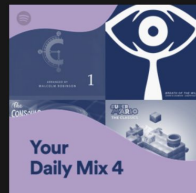
**Daily Mix 1**  
Ellie Goulding, Calum Scott,  
Cashmere Cat and more  
MADE FOR MARK



**Daily Mix 2**  
Véronique Sanson, Jacques  
Brel, Charles Aznavour and  
more  
MADE FOR MARK



**Daily Mix 3**  
Tennis, Metric, Stars and more  
MADE FOR MARK



**Daily Mix 4**  
Malcolm Robinson, User Youth,  
The Consouls and more  
MADE FOR MARK

## Recommended for you, Mark



## Our top pick for you



### Machine Learning A-Z™: Hands-On Python & R In Data Science

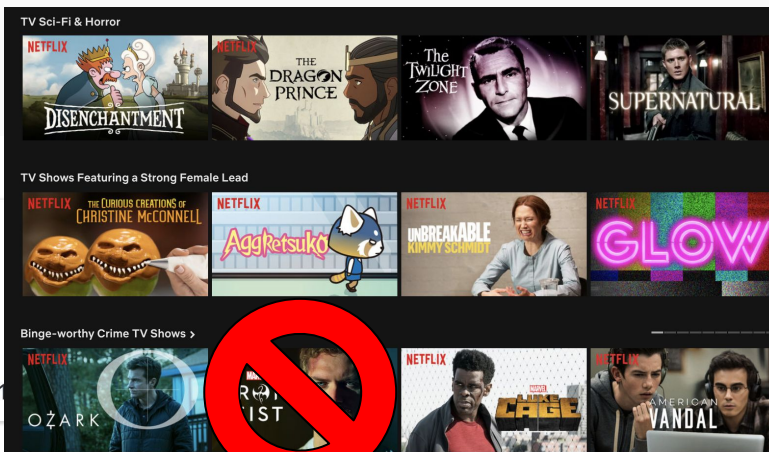
Last Updated October 2018

41 hours • 287 lectures • All Levels

★★★★★ 4.5 (57,599 ratings)

Explore Course

\$199.99 \$19.99



# Where are recommenders used?

They don't always get things right...

## Top Picks for Mark



# Business Goals:

What will the user **like**?

What will the user **buy**?

What will the user **click**?

Name a business that  
cares are each of these,  
and tell us why they care.

# Data Science Canon:

## Netflix's \$1,000,000 Prize (Oct. 2006 - July 2009)

**NETFLIX**

# Netflix Prize

**COMPLETED**

Home Rules Leaderboard Update

## 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	<a href="#">BellKor's Pragmatic Chaos</a>	0.8567	10.06	2009-07-26 18:18:28
2	<a href="#">The Ensemble</a>	0.8567	10.06	2009-07-26 18:38:22
3	<a href="#">Grand Prize Team</a>	0.8582	9.90	2009-07-10 21:24:40
4	<a href="#">Opera Solutions and Vandelay United</a>	0.8588	9.84	2009-07-10 01:12:31

Goal: Beat Netflix's own recommender by 10%.

Took almost 3 years.

The winning team used gradient boosted decision trees over the predictions of **500** other models.

Netflix never deployed the winning algorithm. (Why?)

# Let's learn recommenders.

Today we'll learn:

1. How to **build** a recommender,
2. How to **evaluate** your recommender, and
3. How to **deploy** your recommender.

... and very soon we'll have a case study that uses recommender systems.



# What are some guiding concepts around creating good recommenders?

- **Serendipity** - Did we find something unexpected and novel?
- **Personalization** - Did we make recommendations based on individual taste/want/need?
- **Diversity** - Did we not just recommend overly similar things? Next movie in a series?
- **Persistence** - How much do we recommend what we already know people like? Did how much they like or want it change over time?
- **Modality** - Context? Are there other people involved? Weather? Special Occasion?
- **Privacy** - Did we recommend something *too personal* based on a past purchase?
- **Motivation** - Best match? Best product value? Most profitable?
- **Trust** - Do customers trust the data and the predictions as being authentic?
- **Confidence** - Are the predictions valid?

## Popularity:

- Make the **same** recommendation to **every** user, based only on the popularity of an item.
- E.g. Twitter “Moments”

[What is this most like in other problems we’ve tackled?]

## Content-based (aka, Content filtering):

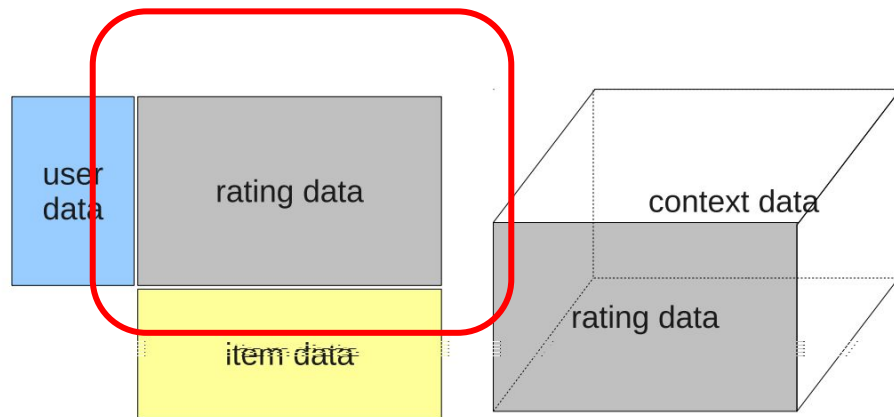
- Predictions are made based on the properties/characteristics of an item.
- User behavior is **not** considered.
- E.g. Pandora Radio

## Collaborative filtering:

- Only consider past user behavior. (**not** content properties...)
- User-User similarity: ...
- Item-Item similarity: ...
- E.g.
  - Netflix & Amazon Recommendations,
  - Google Ads,
  - Facebook Ads, Search, Friends Rec., News feed, Trending news, Rank Notifications, Rank Comments

## Matrix Factorization Methods:

- Find latent features (aka, factors)
- NMF’s fraternal twin



**Figure 1:** Attribute-aware methods can take additional information about the user or the item separately into account (left), whereas context-aware methods are more general and can analyze data that is simultaneously attached to all ‘modes’, i.e. the whole rating event (right).

# What does our dataset look like?

User	Item				
	A	B	C	D	...
	Al	1	?	2	?
	Bob	?	2	3	4
	Cat	3	?	1	5
	Dan	?	2	?	?
	Ed	2	?	?	1
	...				

We have explicit ratings, plus a bunch of missing values.

What company might have data like this?



Btw, we call this the *utility matrix*.

# What does our dataset look like?

User	Item				
	A	B	C	D	...
	Al	0	1	0	1
	Bob	0	0	1	0
	Cat	0	1	1	1
	Dan	1	0	0	1
	Ed	0	1	0	0
	...				

We have implicit feedback,  
and no missing values.

What company  
might have data  
like this?



Btw, we call this  
the *utility matrix*.



	Item				
	A	B	C	D	...
User	AI	1	?	2	?
	Bob	?	2	3	4
	Cat	3	?	1	5

We look at all pairs of users and calculate their similarity.

How can we calculate the similarity of these row vectors?  
(We'll get there.)

User	Item			
	A	B	C	D
	Al	1	?	2
	Bob	?	2	3
	Cat	3	?	1
	Dan	?	2	?
	Ed	2	?	?
	...			

We look at all pairs of items and calculate their similarity.

How can we calculate the similarity of these column vectors?  
(We'll get there.)

## User-User:

	Item				
	A	B	C	D	...
User	Al	1	?	2	?
	Bob	?	2	3	4
	Cat	3	?	1	5

**Let:** $m = \text{\#users,}$  $n = \text{\#items}$ 

We want to compute the similarity of all pairs.

What is the algorithmic efficiency of each approach?

## Item-Item:

User	Item				
	A	B	C	D	
	Al	1	?	2	?
	Bob	?	2	3	4
	Cat	3	?	1	5
	Dan	?	2	?	?
	Ed	2	?	?	1
...					


**User-User:**  $O(m^2n)$ **Item-Item:**  $O(mn^2)$ 

Which one is better?



# Similarity Metric using Euclidean Distance

(Not used often. Just a familiar example!)

What's the range? 


$$\text{dist}(a, b) = ||a - b|| = \sqrt{\sum_i (a_i - b_i)^2}$$

But we're interested in a **similarity**, so let's do this instead:

What's the range? 

$$\text{similarity}(a, b) = \frac{1}{1 + \text{dist}(a, b)}$$


# Similarity Metric using Pearson Correlation

What's the range? 

$$\text{pearson}(a, b) = \frac{\text{cov}(a, b)}{\text{std}(a) * \text{std}(b)} = \frac{\sum_i (a_i - \bar{a})(b_i - \bar{b})}{\sqrt{\sum_i (a_i - \bar{a})^2} \sqrt{\sum_i (b_i - \bar{b})^2}}$$


But we're interested in a **similarity**, so let's do this instead:

When use  
this?

What's the range? 

$$\text{similarity}(a, b) = 0.5 + 0.5 * \text{pearson}(a, b)$$


# Similarity Metric using Cosine Similarity

What's the range? 

$$\cos(\theta_{a,b}) = \frac{a \cdot b}{||a|| ||b||} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2} \sqrt{\sum_i b_i^2}}$$


But we're interested in a **standardized similarity**, so let's do this instead:

When use  
this?

What's the range? 

$$\text{similarity}(a, b) = 0.5 + 0.5 * \cos(\theta_{a,b})$$

# Similarity Metric using Jaccard Index

What's the range?   $\text{similarity}(a, b) = \frac{|U_a \cap U_b|}{|U_a \cup U_b|}$

$U_k$  denotes the set of users who rated item  $k$

When use this?

# The Similarity Matrix

Pick a similarity metric, create the similarity matrix:

	item 1	item 2	item 3	...
item 1	1	0.3	0.2	...
item 2	0.3	1	0.7	...
item 3	0.2	0.7	1	...
...	...	...	...	...

Say user  $u$  hasn't rated item  $i$ . We want to predict the rating that this user *would* give this item.

$$\text{rating}(u, i) = \frac{\sum_{j \in I_u} \text{similarity}(i, j) * r_{u,j}}{\sum_{j \in I_u} \text{similarity}(i, j)}$$

$I_u$  = set of items rated by user  $u$

$r_{u,j}$  = user  $u$ 's rating of item  $j$

We order by descending predicted rating for a single user, and recommend the top  $k$  items to the user.

This calculation of predicted ratings can be very costly. To mitigate this issue, we will only consider the  $n$  most similar items to an item when calculating the prediction.

$$\text{rating}(u, i) = \frac{\sum_{j \in I_u \cap N_i} \text{similarity}(i, j) * r_{u,j}}{\sum_{j \in I_u \cap N_i} \text{similarity}(i, j)}$$

$I_u$  = set of items rated by user  $u$

$r_{u,j}$  = user  $u$ 's rating of item  $j$

$N_i$  is the  $n$  items which are most similar to item  $i$

# Deploying the recommender

## **In the middle of the night:**

- Compute similarities between all pairs of items.
- Compute the neighborhood of each item.

## **At request time:**

- Predict scores for candidate items, and make a recommendation.

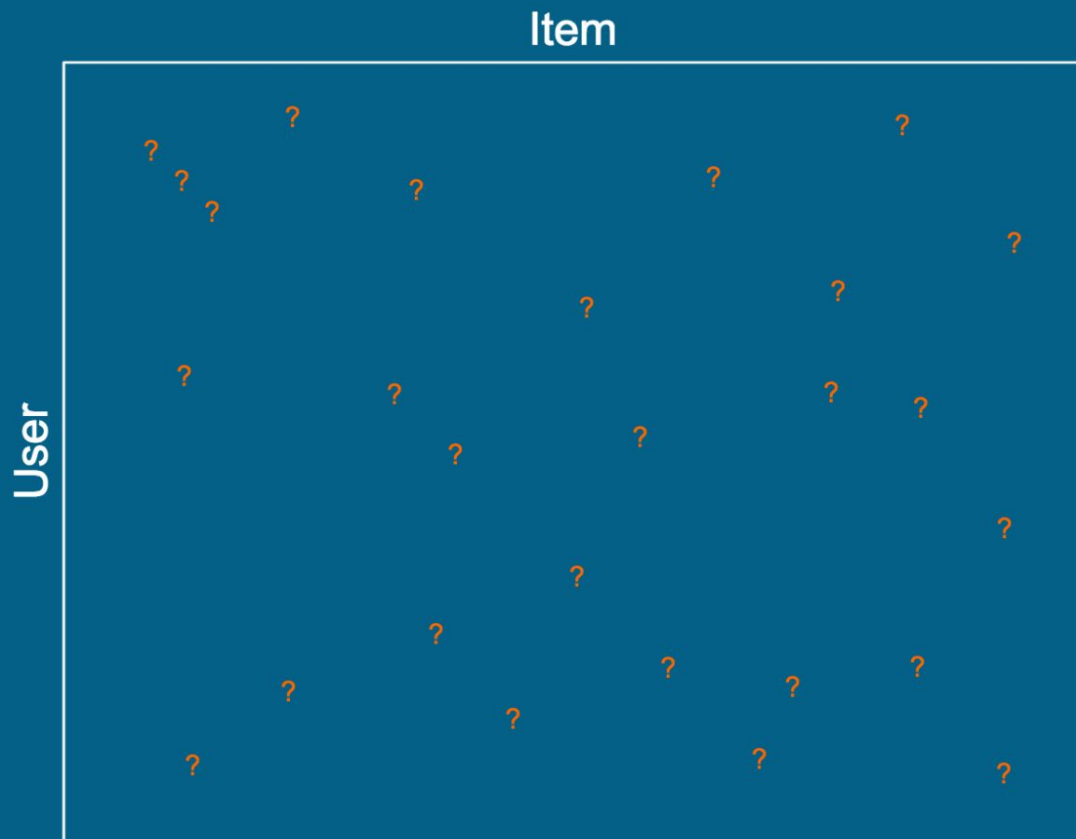


# How do we evaluate our recommender system?

Is it possible to do cross-validation like normal?

**Before we continue, let's review: Why do we perform cross-validation?**

Quick warning: Recommenders are inherently hard to validate. There is a lot of discussion in academia (research papers) and industry (here, Kaggle, Netflix, etc) about this. There is no ONE answer for all dataset.



For this slide, the question marks denote the holdout set (**not** missing values).

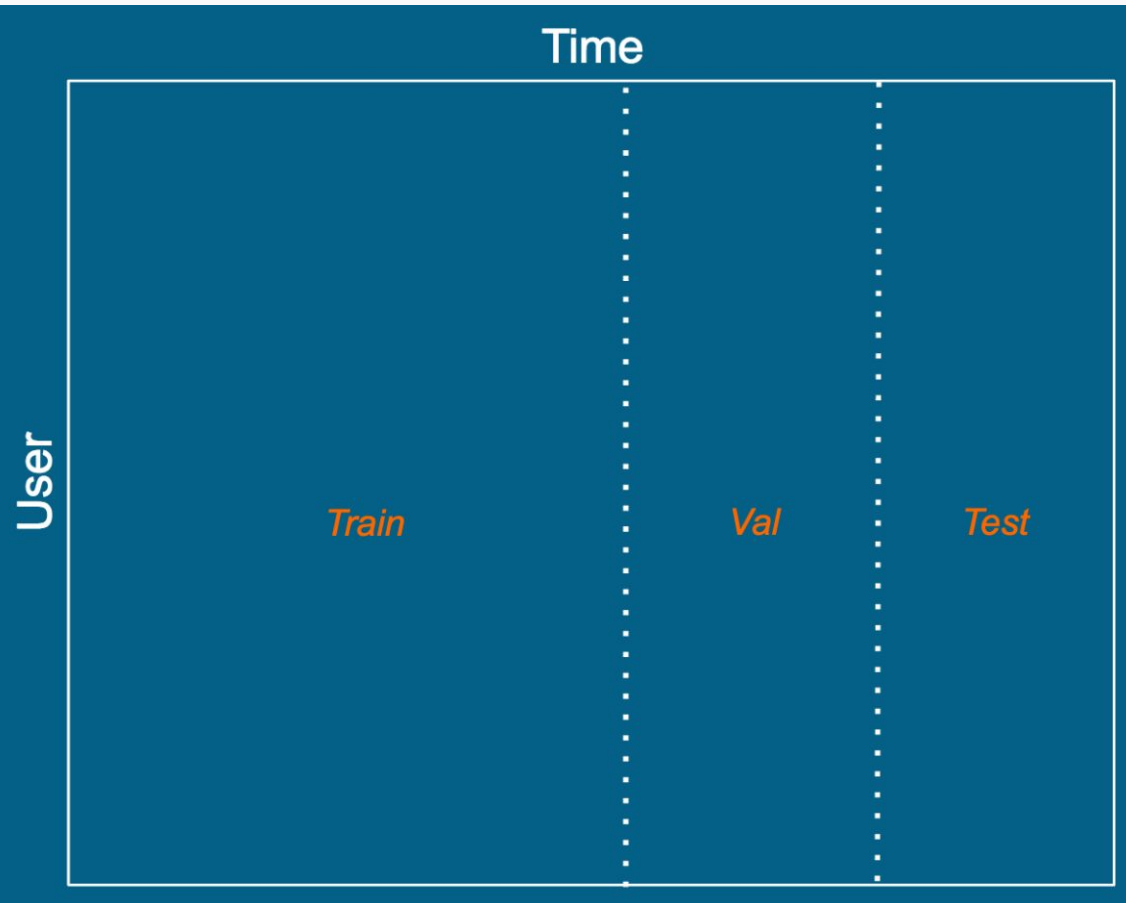
We can calculate MSE between the targets and our predictions over the holdout set.

(K-fold cross-validation is optional.)

Recall: Why do we perform cross-validation?

Why isn't the method above a true estimate of a recommender's performance in the field?

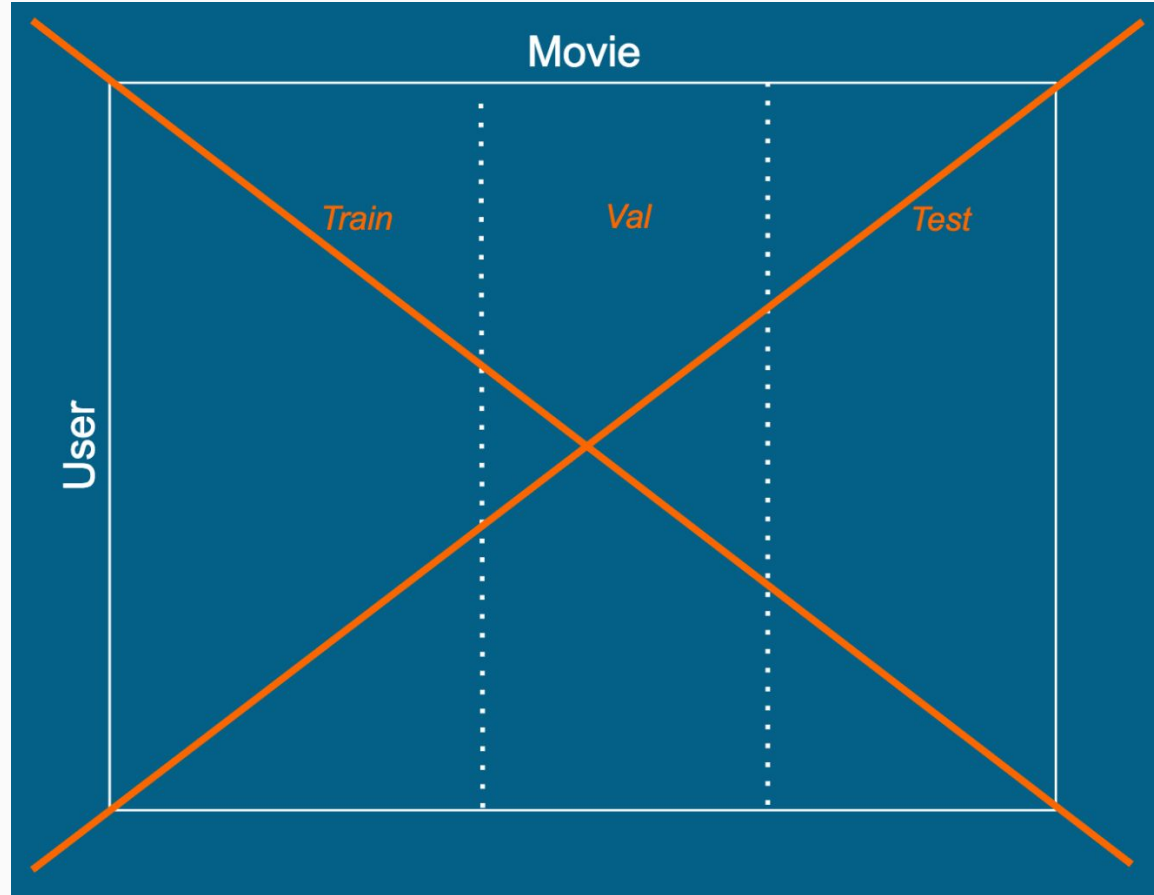
Why would A/B testing be better?



What's the deal with this?

I.e. Why might we prefer doing this instead of the more “normal” cross-validation from the previous slide?

DON'T DO THIS! Why?



# How to deal with “cold start”?

**Scenario:** A new user signs up.

What will our recommender do (assume we're using item-item similarities)?

**One strategy:** Force users to rate 5 items as part of the signup process. AND/OR Recommend popular items at first.

**Scenario:** A new item is introduced.

What will our recommender do (assume we're using item-item similarities)?

**One strategy:** Put it in the “new releases” section until enough users rate it AND/OR use item metadata if any exists.

# How to deal with “cold start”?

**Scenario:** A new user signs up.

What will our recommender do (assume we're Youtube and we're using item popularity to make recommendations)?

**This really isn't a problem...**

**Scenario:** A new item is introduced.

What will our recommender do (assume we're Youtube and we're using item popularity to make recommendations)?

**One strategy:** Don't use total number of views as the popularity metric (we'd have a *rich-get-richer* situation). Use something else...

# Matrix Factorization for Recommendation

Warning: There are a lot of acronyms  
in this lecture!

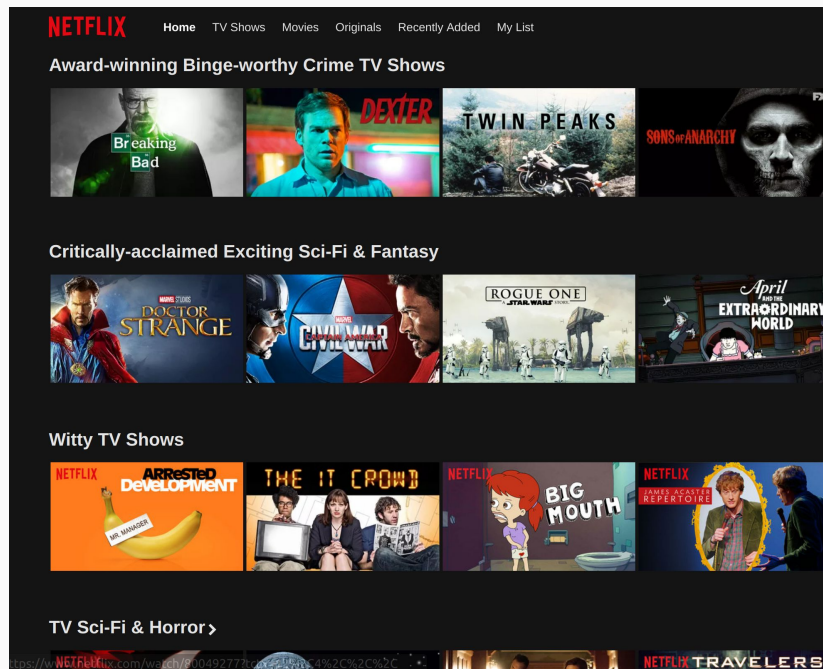
Mark Llorente  
Based off Ryan Henning's Slides



- UV Decomposition (UVD)
- SVD vs UVD
- UVD vs NMF
- UVD via Stochastic Gradient Descent (SGD)
- Matrix Factorization for Recommendation:
  - Basic system:
    - UVD + SGD... FTW
  - Intermediate topics:
    - regularization
    - accounting for biases

# “Topic” Based Recommendation

Where do the topics come from?



galvanize




$$R_{m \times n} \approx U_{m \times k} V_{k \times n}$$

$$r_{ij} \approx u_{i:} \cdot v_{:j}$$

- You choose  $k$ .
- $UV$  approximates  $R$  by necessity if  $k$  is less than the rank of  $R$ .
- Usually choose:  $k \ll \min(n, m)$
- Compute  $U$  and  $V$  such that:

$$\arg \min_{U, V} \sum_{i, j} (r_{ij} - u_{i:} \cdot v_{:j})^2$$

Least  
Squares!



# Single Value Decomposition\* vs UV Decomposition

## SVD:

- $R = USV^T$
- $U$  is an orthogonal matrix
- $S$  is a diagonal matrix of decreasing positive “singular” values
- $V$  is an orthogonal matrix
- Has a unique, exact solution

## UVD:

- $R \approx UV$
- $U$  and  $V$  will not (likely) be orthogonal
- Has many approximate, non-unique solutions:
  - non-convex optimization; has many local minima
- Has a tunable parameter  $k$

\*Note: There's a very good algorithm most akin to UVD that is called FunkSVD and has since been shortened by the community to just SVD even though it's not at all SVD as no singular values are found, and I want to cry.

# UVD vs Non-negative Matrix Factorization

UVD:

- By convention:  $R \approx UV$
- ... (see previous slides)

**NMF** is a specialization of **UVD**!

Both are approximate factorizations, and both optimize to reduce the RSS.

NMF:

- By convention:  $V \approx WH$
- Same as UVD, but with one extra constraint:  
**all values of  $V$ ,  $W$ , and  $H$  must be non-negative!**

## UVD vs NMF (*continued*)

UVD and NMF are both solved using either:

- Alternating Least Squares (ALS)
- Stochastic Gradient Descent (SGD)

# UV Decomposition Fitting Example!

**To the board!**

**(You will not have to solve any of this yourself manually. This is just showing how updating is possible.)**

# Alternating Least Squares vs SGD

## ALS:

- Parallelizes very well
- Available in Spark/MLlib

## SGD:

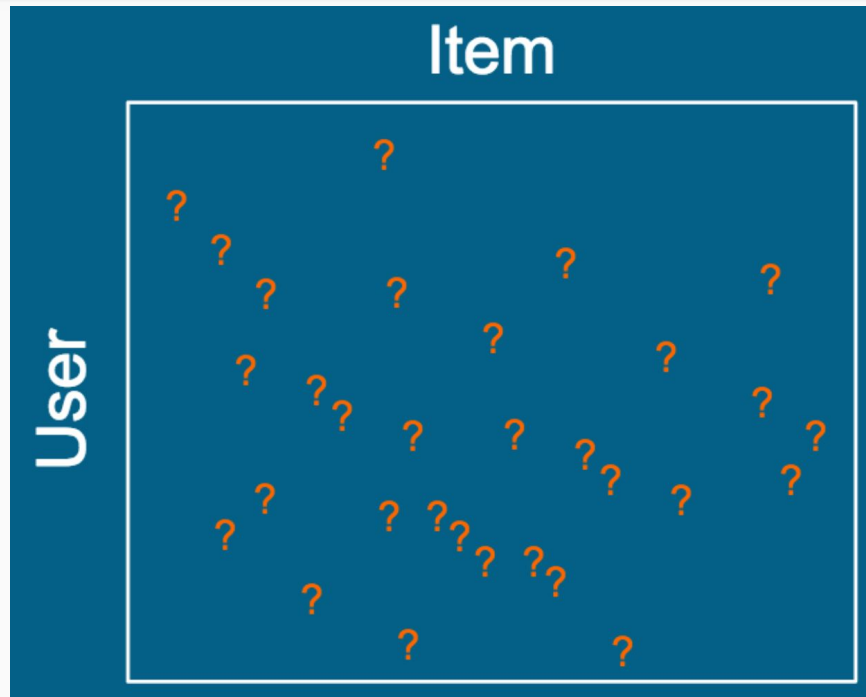
- Faster (if on single machine)
- Requires tuning learning rate
- Anecdotally better results...
- Works with missing values without special tricks (we'll call this a **sparse** matrix in this lecture)
- Easy to update when new ratings come in

# Matrix Factorization for Recommendation

Recall: An explicit-rating utility matrix is usually VERY sparse...

We've previously used SVD to find latent features (aka, factors)... Would SVD be good for this sparse utility matrix?  
(Hint: No!)

**What's the problem with using SVD on this sparse utility matrix?**



# SVD\*\*\* vs UVD (*revisited*)


## SVD (actual hand over my heart SVD):

- $R = USV^T$
- ...
- **Bad if  $R$  has missing values!**
  - You are forced to fill in missing values.
  - Solution fits these fill-values (which is silly).
  - Makes for a much larger memory footprint.
  - Slow to compute for large matrices.

## UVD:

- $R \approx UV$
- ...
- **Handles missing values when computed via SGD.**

$$\arg \min_{U,V} \sum_{i,j \in \mathcal{K}} (r_{ij} - u_{i:} \cdot v_{:j})^2$$

 Set of indices of known rating



# UVD + ALS or UVD + SGD!

Finding the best values through Alternating Least Squares or Stochastic Gradient Descent makes a lot of sense for recommender systems.

In fact, **UVD + SGD** is often considered '**best in class**' because:

- No need to impute missing values.
- Can use regularization to avoid overfitting.
- Optionally include bias terms to communicate prior knowledge.
- ***Can handle time-dynamics (e.g. change in user preference over time) without full retrain.***
- Used by the winning entry in the Netflix challenge.

Spark implements their current recommender using ALS, which is a clever feat of distributed calculations! (Default iterations is only 5, so make sure to tune your model!)

# Warning: Don't forget to regularize!

Since now we're fitting a large parameter set to sparse data, you'll most certainly need to regularize!

$$\arg \min_{U, V} \sum_{i, j \in \mathcal{K}} (r_{ij} - u_{i:} \cdot v_{:j})^2 + \lambda(||u_{i:}||^2 + ||v_{:j}||^2)$$

Tune lambda:  
the amount of  
regularization

In practice, much of the observed variation in rating values is due to item bias and user bias:

- Some items (e.g. movies) have a tendency to be rated high, some low.
- Some users have a tendency to rate high, some low.

We can capture this prior domain knowledge using a few bias terms:

$$b_{ij} = \mu + b_i^* + b'_j$$

The overall bias of the rating by user  $i$  for item  $j$

The overall average rating (i.e. the overall bias)

User  $i$ 's average deviation from the overall average

Item  $j$ 's average deviation from the overall average

# We added bias terms... now: The 4 parts of a prediction

$$r_{ij} \approx \mu + b_i^* + b'_j + u_{i:} \cdot v_{:j}$$

The prediction  
of user i rating  
item j

The average  
rating

User i's  
tendency to  
deviate from  
the average

Item j's  
tendency to  
deviate from  
the average

The prediction of  
how user i will  
interact with  
item j

Ratings are now estimated as:

$$r_{ij} \approx \mu + b_i^* + b'_j + u_{i:} \cdot v_{:j}$$

The new cost function, with the biases included:

$$\arg \min_{U, V, b^*, b'} \sum_{i, j \in \mathcal{K}} (r_{ij} - \mu - b_i^* - b'_j - u_{i:} \cdot v_{:j})^2 + \lambda_1 (\|u_{i:}\|^2 + \|v_{:j}\|^2) + \lambda_2 ((b_i^*)^2 + (b'_j)^2)$$

New part!

New part!

# FunkSVD (Most often called “SVD” even though it’s not true SVD from PCA/SVD day!)

Some packages that use some variation of FunkSVD algorithm:

Surprise: <http://surpriselib.com/>

I’ve seen projects that used this, but can’t speak to how easy it is to use. Lots of additional options for using this package.

LightFM: <https://github.com/lyst/lightfm>

Handles many many types of recommender algorithms and can handle using additional outside data to tweak the algorithm. A bit of a black box.

# Final Tips

(1 of 4) You can attempt to deploy many versions of matrix factorization models! The key things to think about are “where is the majority of my signal coming from and how do I make sure my model reflects that.

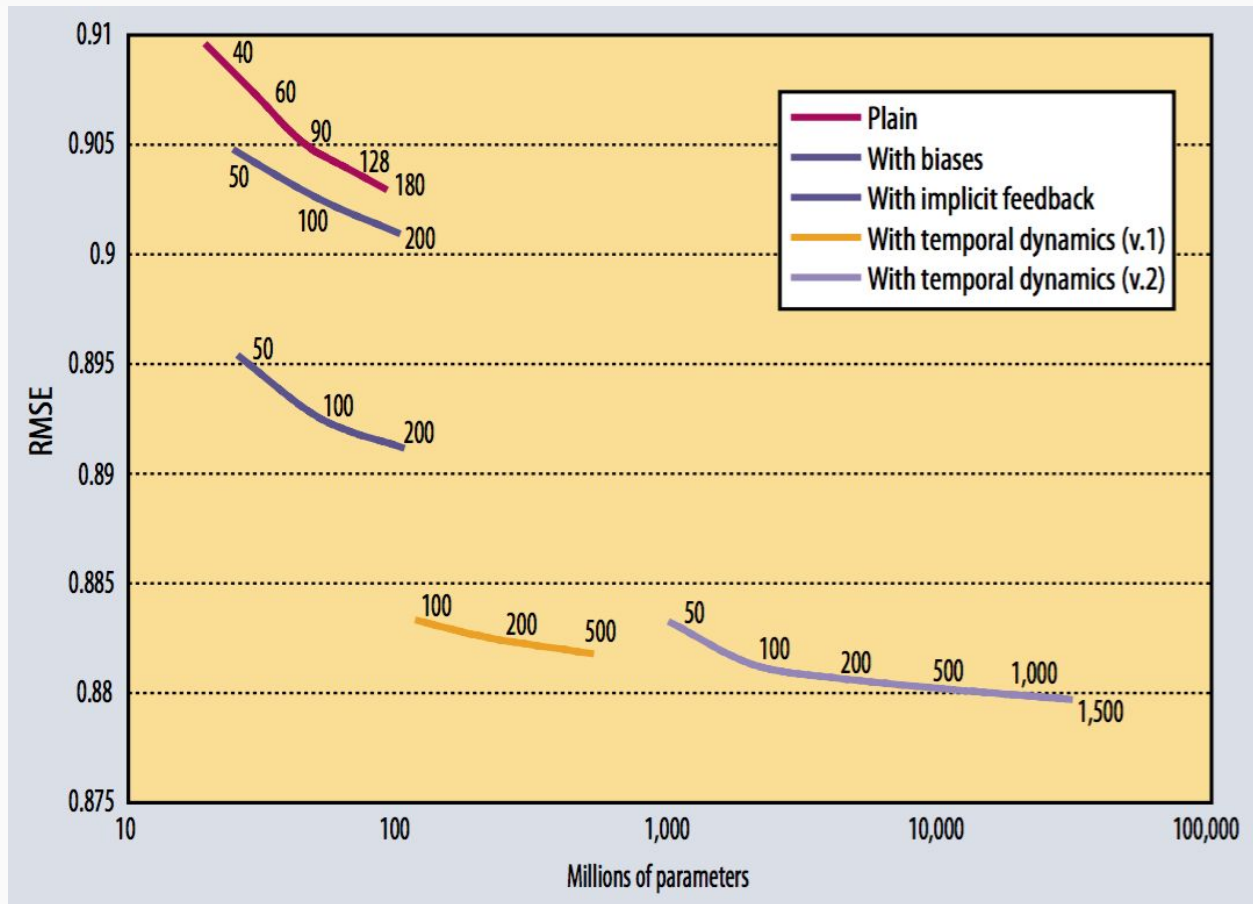
(2 of 4) Even if we use a very fancy tools, often the majority of signal can come from the bias *terms*: i.e. the entire ratings *mean*, the (leftover) per item means, the (leftoverleftover) per user means added together. The *residuals* from those predictions are what we will fit our fancy ALS/SGD models on!

# Final Tips

(3 of 4) The simple matrix factorization models like Spark's ALS recommender tend to overfit whether or not you've subtracted the bias terms from the data before fitting. Also, you can't remove the bias terms when using the NMF version. (Why?) Tweaking the regularization hyperparameters can help. FunkSVD based algorithms go one step further by figuring out the optimal amount of debiasing to perform on the data for matrix factorization.

(4 of 4) The only way to know for sure is to try things! That means cross validation across models and comparing scores.





Root mean square error over the Netflix dataset using various matrix factorization models.

Numbers on the chart denote each model's dimensionality (k).

The more refined models perform better (have lower error).

**Netflix's inhouse model performs at RMSE=0.9514 on this dataset**, so even the simple matrix factorization models are beating it!

Read the paper for details; it's a good read!

Thank you for your time!

galvanize

# FunkSVD (Most often “SVD” even though it’s not true SVD that we saw on Tuesday!!!)

A new cost function to account for bias terms!

$$\min_{p_*, q_*, b_*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda_3 (\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

svd++ cost function

Would UVD (or NMF) work better than SVD to find latent factors/topics when the utility matrix is sparse?

