Smallest font

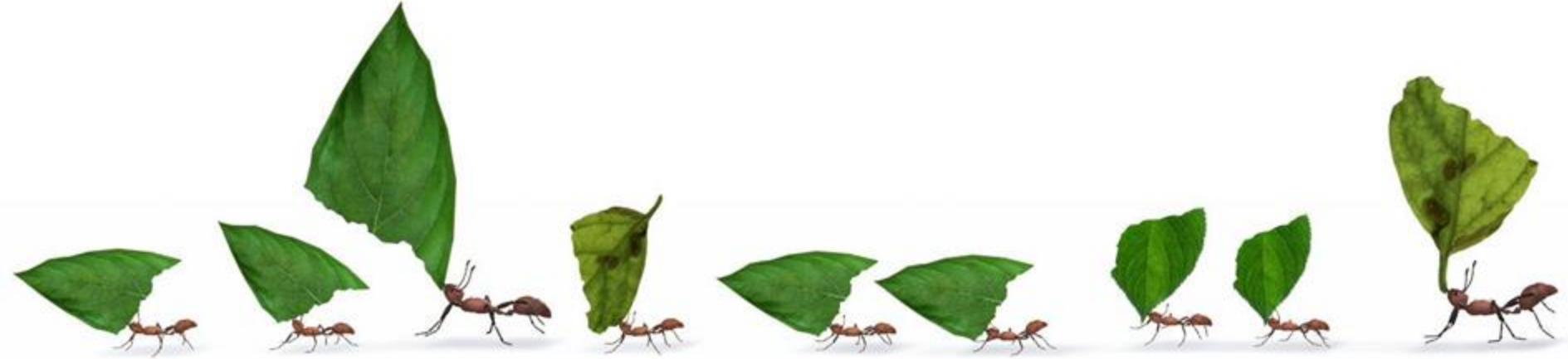
Please turn off and put away your cell phone

# Calibration slide

These slides are meant to help with note-taking
They are no substitute for lecture attendance

Smallest font

# Big Data



NYU Center for Data Science


# Recommender systems

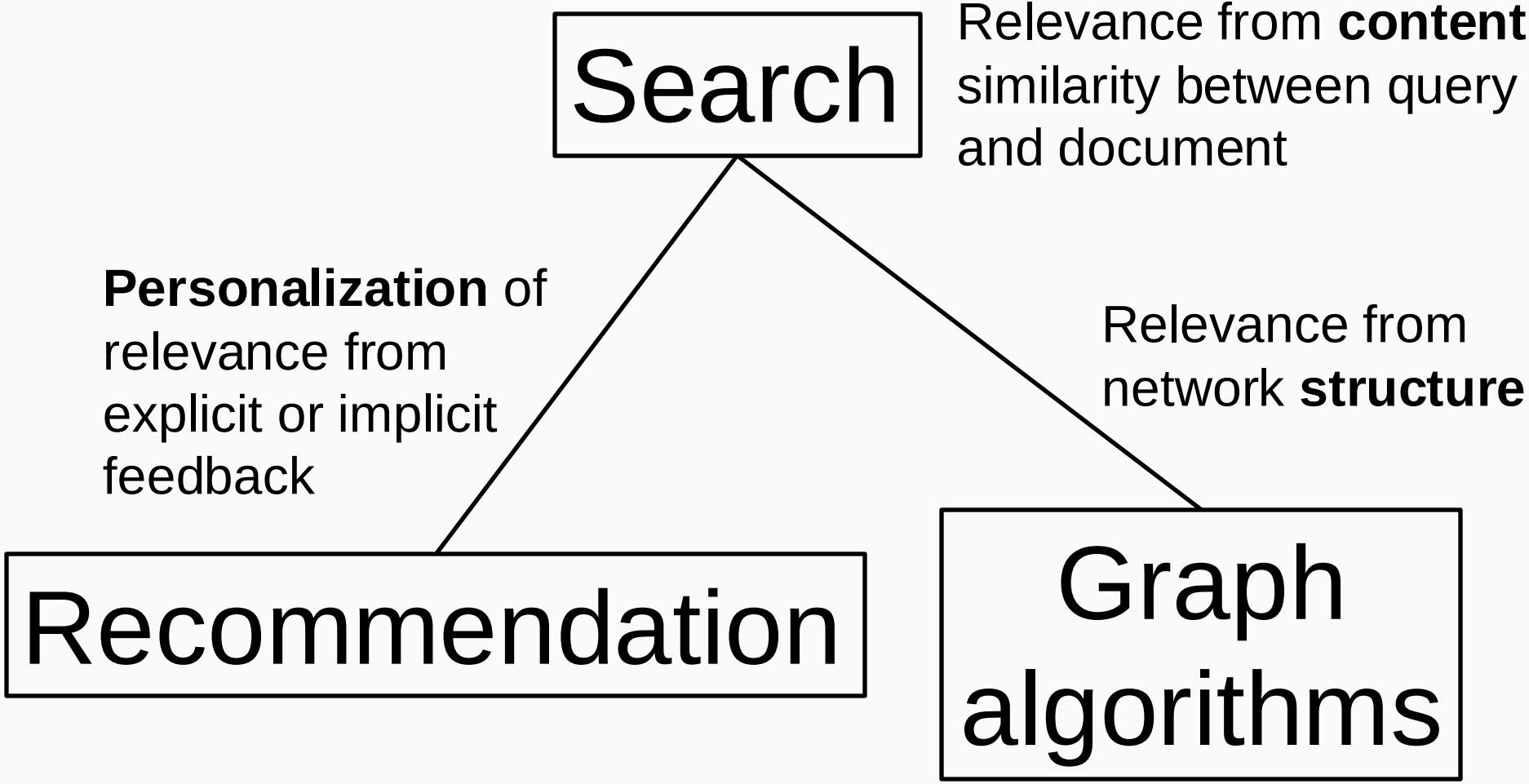
DS-GA 1004: Big Data

# This week

- Recommender systems
- Collaborative filtering algorithms
- Ranking and evaluation

# Confusion, Doubt, & Struggle: **Search**

- **Approximate minHash** (using hashes instead of permutations): The point of the permutations in minHash is to get a single integer number (the first time the concept is encountered in the document, given the permutation), then check for collisions.
- But if one is doing that, the ultimate purpose of the permutations is a single integer, which we can get more cheaply from a hash function (even doing a single permutation, if there is a large number of items is expensive). But the hash function doesn't mind a large input.
- **How do different outputs of different Hashes ($H_1$, $H_2$, etc.) of the same hash function H come from**? The hash functions used for this purpose (e.g. "MurmurHash") are families of functions that take parameters. In approximate minhash, we can use random coefficients for these parameters, e.g. hash(x) = (a*x + b) mod p, where **x** is the input, **a** and **b** are random coefficients and **p** is a large prime number. Those come from a seed, which is an input to MurmurHash (in addition to the input).
- If we look for similarity of documents, we have to reckon with "stop words" – words that are commonly used, which will leads to too many collisions for plain vanilla minHash. So precision will be low, defeating the purpose of doing approximate search in the first place.
- One solution: Instead of curating stop-words, use minHash with LSH. We can partition the rows into a number b of blocks and r of rows, but there are many tradeoffs between the number of blocks and the number of rows per block we use. If we ask for many rows (only considering it a collision if all rows in a block collide), the probability of random collisions will go way down, so precision will go up. If we increase the number of blocks (counting a collision in any block), recall will go up again.


Search
Relevance from **content** similarity between query and document
**Personalization** of relevance from explicit or implicit feedback
Relevance from network **structure**
Recommendation
Graph algorithms

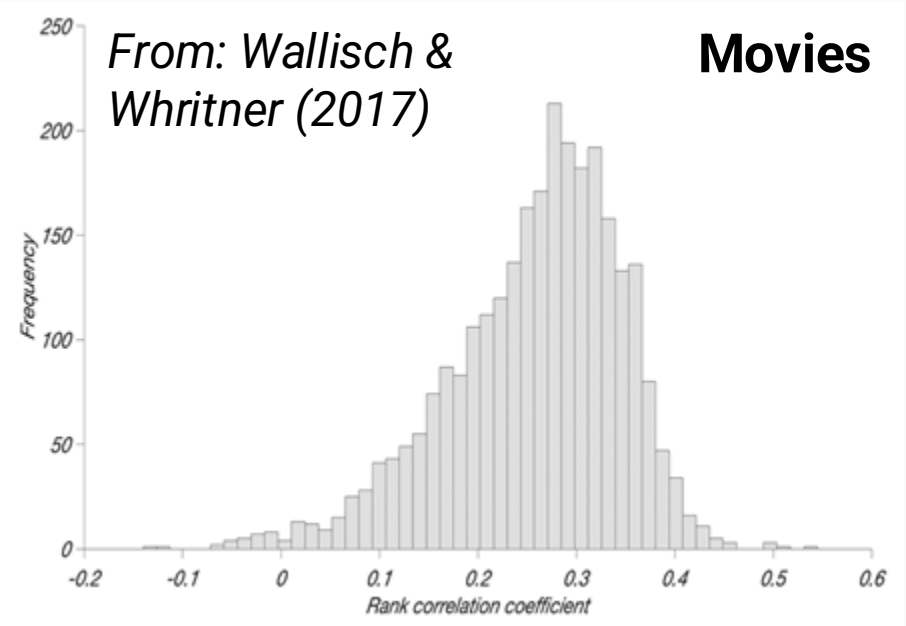
# Now: From Search to Recommendation

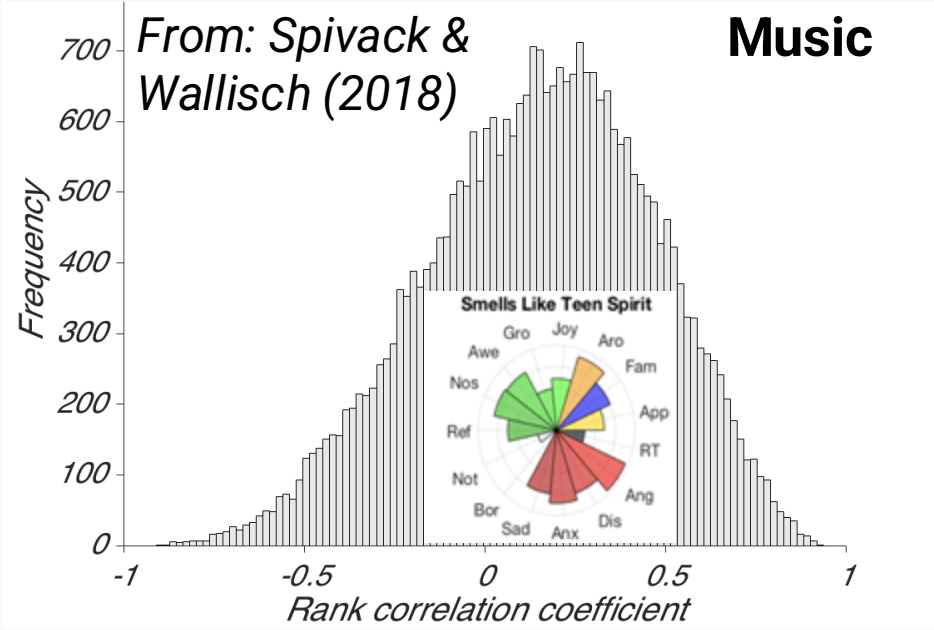
- Early solutions relied on **indexing** and **search**
  - Users must describe what they want
  - Same query by different users gets same results
- **Recommendation** = **search** + **personalization**
  - A user's past history informs the ranking of results
  - Not everyone has the same preferences
  - Relevance, but personalized…
  - **Other users' histories are also informative!**


Yahoo.com in April, 1997 [archive.org]

# Why the need for recommendation (personalization)?

Because people are complicated. There is genuine and dramatic diversity in preferences:

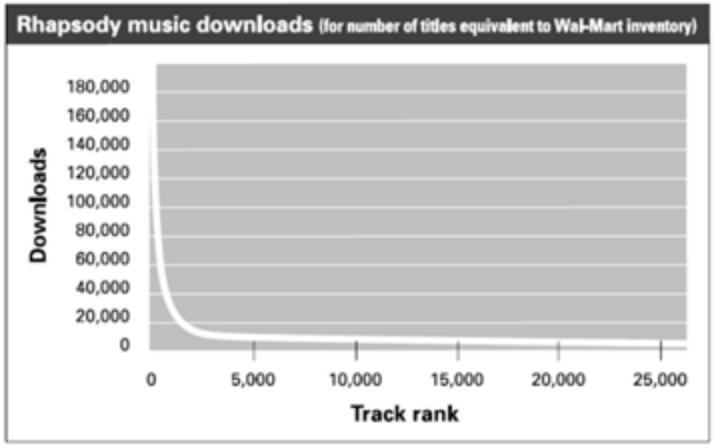


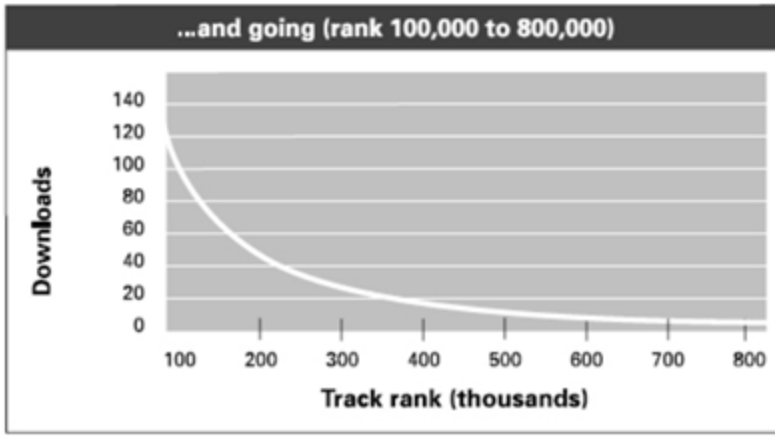



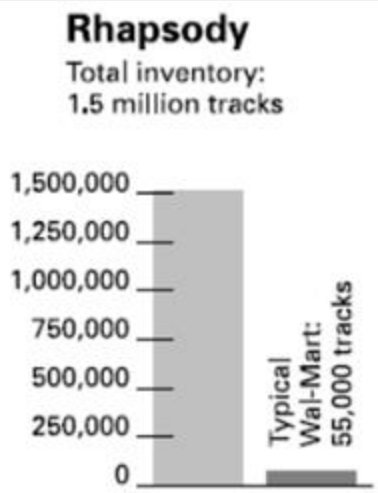
# **Good timing**:
# This need coincided with digital affordances in the 21st century


- **Physical objects occupy space**
  - Brick-and-mortar shops must satisfy physical constraints
  - Curators must prioritize for some notion of utility
  - Serving the most customers, maximizing sales/profit, etc.
- This is not true for **digital items**!
  - The web, e-books, news articles, movies, music, … take up no physical space
  - Without algorithmic curation, this quickly becomes overwhelming

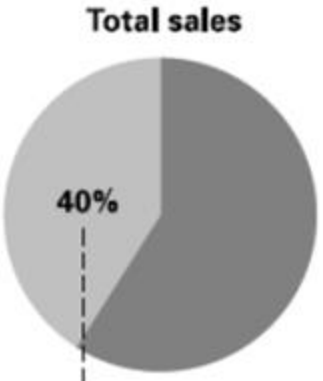
# Contrasting the brick/mortar “short head” with the digital “long tail”



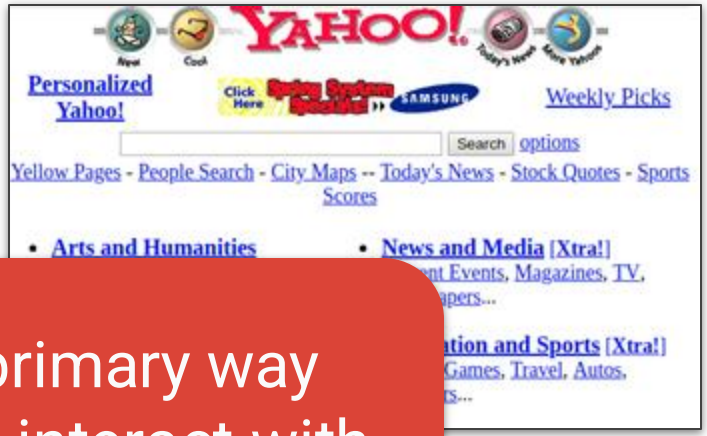

From: Chris Anderson:
“The long tail” (2006)



# Search ⇒ Recommendation

**At this point, >70% of clicks on Youtube and >80% on Netflix result from recommendations (not search or direct links)**

- Early solutions relied on **indexing** and **search**
  - Users must describe what they want
  - Same query by different users gets same results




Recommender systems are now the primary way that most people (often unknowingly) interact with large collections!

April, 1997 [archive.org]

# Personalization is critical to recommendation

- Traditional search / information retrieval:
  - Model the **relevance** of an **item** in response to a **query**
- Personalized search / recommendation
  - Same as above, but the model can access a description of the **user**
- To model **relevance** per user, we'll need to record data
  - This is known as **feedback**
- This approach works best if you have big data

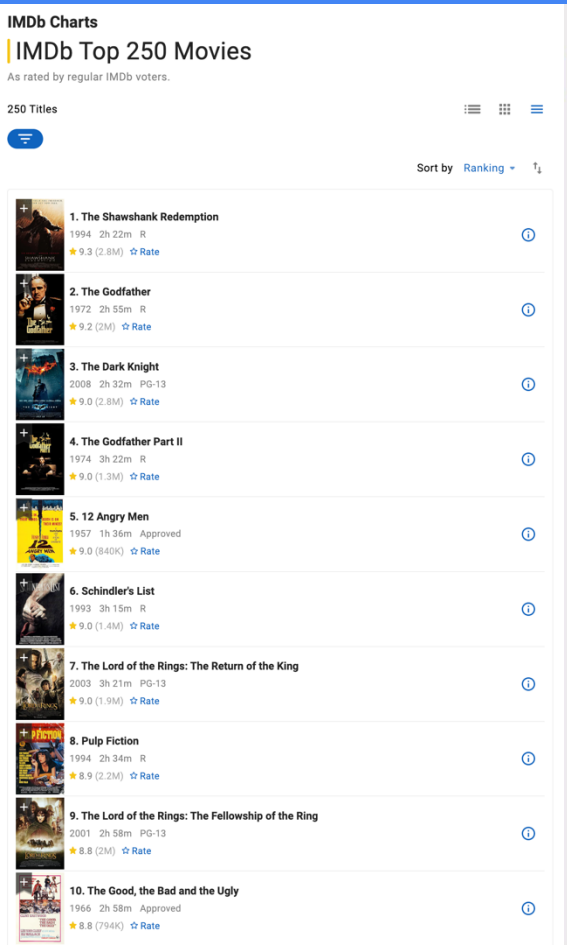
# And yet, there is a shared signal: The baseline tailored recommendation systems need to beat: The “popularity based model”

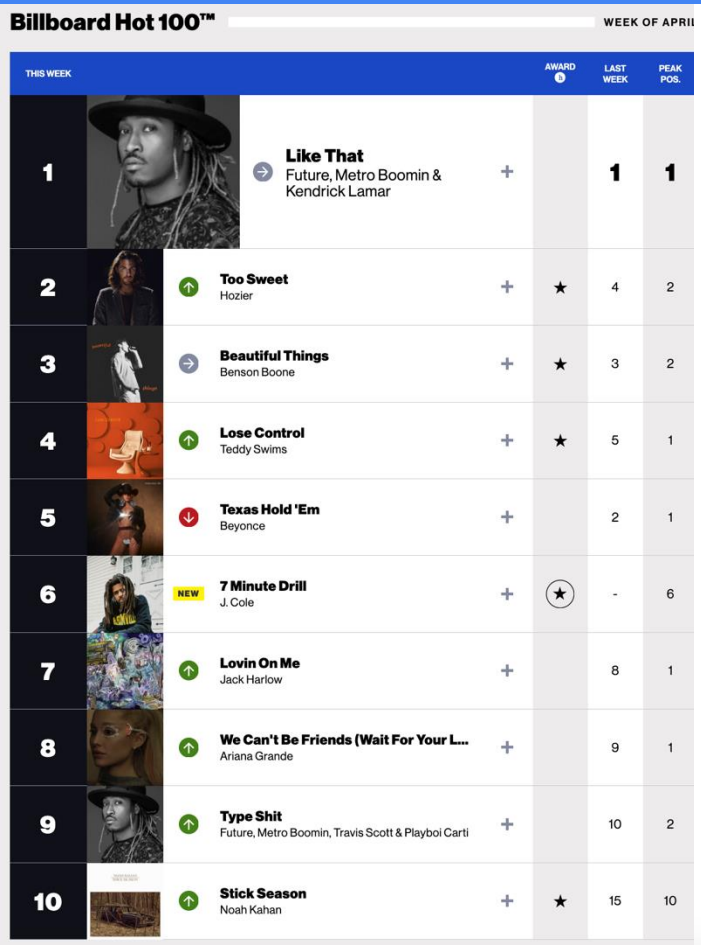

- **This should always be the first thing you try.**
  - 1. Compute the **average utility** for each item

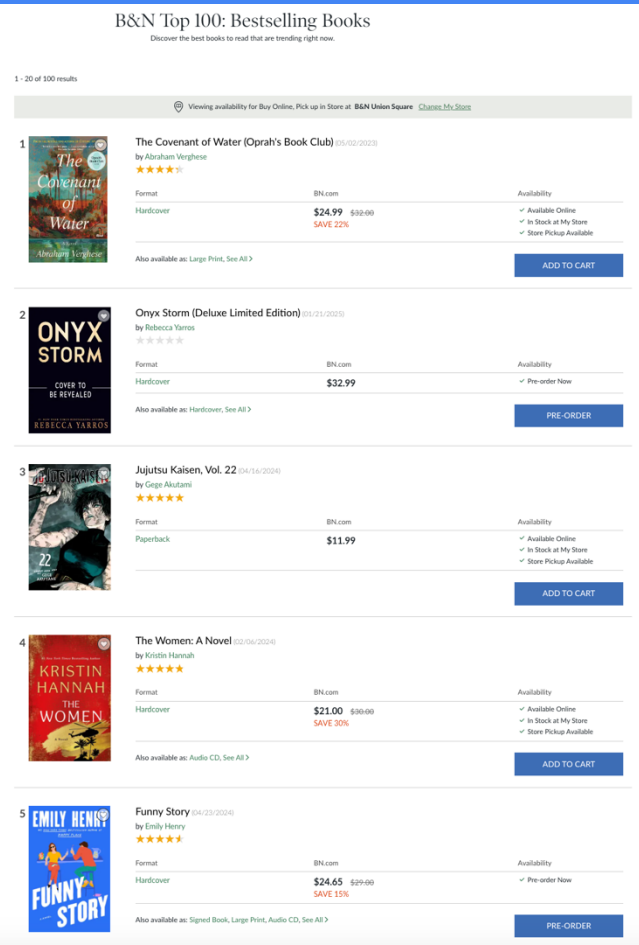
$$P[i] \leftarrow (\sum_u R[u, i]) \,/\, |R[:, i]|$$

  - 2. $f$ = list of items ranked by descending $P[i]$
- AKA: the “**short head**”; **greatest hits**; **Starbucks mix**, …
- This produces the **same ranking for all users**. **Personalization** should improve, but by how much?

# Some good examples



# Personalized recommendations can be based on implicit and explicit feedback

## Explicit feedback

- Examples:
  - **$$$** (Purchases)
  - ★★★☆☆
  - 👍👎
  - +Subscribe / Conversion

- **Strong signal** (+/-)

- **Relatively rare**: users will not always provide explicit feedback - concerns about selection bias.

# Explicit vs. implicit feedback: Tradeoffs

## Explicit feedback

- Examples:
  - $$$ (Purchases)
  - ★★★☆☆
  - 👍👎
  - +Subscribe / Conversion
- **Strong signal** (+/-)
- **Relatively rare**: users will not always provide explicit feedback - concerns about selection bias.

## Implicit feedback

- Examples:
  - Click-through
  - Downloads
  - Play counts
  - Abandonment / skipping
- **Weak, ambiguous signal** (+ only, usually)
- **Much more abundant** than explicit feedback

# How is feedback obtained?

- **Explicit feedback** makes sense when

  **Time/effort to rate item is much less than time to consume the item!**

  - **Examples**: movies, books, restaurants, courses, …
- **Implicit feedback** is currently more commonly used → Big Data

  - **Examples**: Songs, news articles, short videos, search engine results, …

# We can/will use both implicit and explicit feedback to personalize recommendations. But first: Improving the popularity baseline without tailoring

- Which would you rather have?
  - Item 1: **1000 ratings**, avg = ★★★★☆
  - Item 2: **1 rating**, avg = ★★★★★
- Few **interactions** $|R[:, i]|$ $\Rightarrow$ **unstable estimates** $P[i]$
- **Quick fix 1**: discard items with too few ratings
- **Quick fix 2**: use a **prior**:

$$P[i] \leftarrow (\sum_u R[u, i]) \,/\, (|R[:, i]| + \beta)$$

  - $\beta$: A hyperparameter that "**damps**" all ratings, but impacts low n averages more
  - AKA "**pseudo-counts**" (equivalent to adding extra observations with R=0)

# Global, item, and user “bias”

- Model each interaction as a combination of **global**, **item**, and **user** terms:

$$R[u, i] \approx \mu + b[i] + b[u]$$

# Global, item, and user “bias”

- Model each interaction as a combination of **global**, **item**, and **user** terms:

$$R[u, i] \approx \mu + b[i] + b[u]$$

- The average rating over all interactions:

$$\mu = (\sum_{u,i} R[u, i])/(|R| + \beta_g)$$

# Global, item, and user “bias”

- Model each interaction as a combination of **global**, **item**, and **user** terms:

$$R[u, i] \approx \mu + b[i] + b[u]$$

- The average rating over all interactions:

$$\mu = (\sum_{u,i} R[u, i])/(|R| + \beta_g)$$

- Average (users) difference from $\mu$ for item $i$:

$$b[i] = (\sum_{u} R[u, i] - \mu)/(|R[:, i]| + \beta_i)$$

# Global, item, and user “bias”

- Model each interaction as a combination of **global**, **item**, and **user** terms:

$$R[u, i] \approx \mu + b[i] + b[u]$$

- The average rating over all interactions:

$$\mu = (\sum_{u,i} R[u, i])/(|R| + \beta_g)$$

- Average (users) difference from $\mu$ for item $i$:

$$b[i] = (\sum_{u} R[u, i] - \mu)/(|R[:, i]| + \beta_i)$$

- Average (items) difference from $\mu+b[i]$ for user $u$:

$$b[u] = (\sum_{i} R[u, i] - \mu - b[i])/(|R[u]| + \beta_u)$$

# Recommending with the “bias model”

- $R[u, i] \approx \mu + b[i] + b[u]$ lets us estimate all interactions
- Predictions for user $u$: sort items $i$ by descending $\mu + b[i] + b[u]$
- $\mu$ and $b[u]$ are constant in this prediction, since $u$ is fixed.
- Equivalently, we can sort by the item bias $b[i]$
- This isn’t any more powerful than the original popularity model (every user gets the same item bias list)
  - but it is a lot more interpretable (pure item (”bias”)!

# Going beyond popularity baselines and bias models with “Collaborative filtering”

# Collaborative filtering: The task / challenge

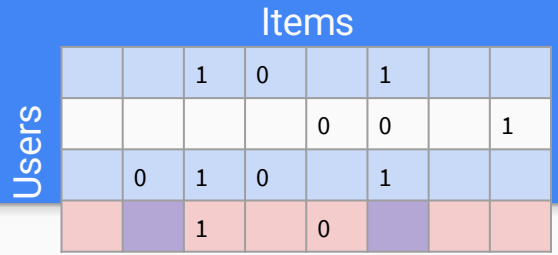
- ***Utility matrix* (R)**: feedback for sparsely observed interactions

Items

| Users | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | | | 1 | | |
| | | | | | 0 | 0 | | 1 |
| | | 1 | 1 | | | 1 | | |
| | | | 1 | | 0 | | | |

- **Task**: predict the missing entries $\rightarrow R_{ui} \approx f(\text{User} = u, \text{Item} = i)$
- **Evaluation**: depends on the feedback mechanism

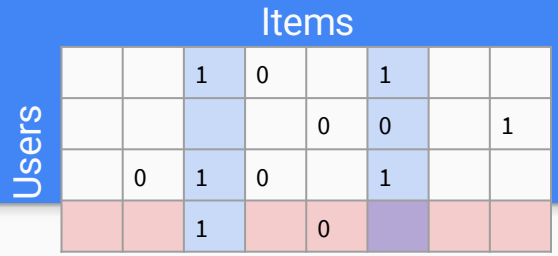
# Neighborhood models




- **User-based** model:
  - Given a user *u*, find the most similar users {*u'*}
  - (Similar rows of the utility matrix)
  - Predict items *v* with high feedback by similar users, not yet consumed by *u*

*Su and Khoshgoftaar, 2009*

# Neighborhood models

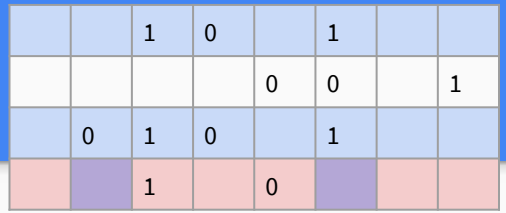



- **User-based** model:
  - Given a user *u*, find the most similar users {*u'*}
  - (Similar rows of the utility matrix)
  - Predict items *v* with high feedback by similar users, not yet consumed by *u*

- **Item-based** model:
  - Find items *v'* similar to those consumed by *u*
  - (Similar columns of the utility matrix)
  - Predict those which have not yet been consumed by *u*

*Su and Khoshgoftaar, 2009*

# Neighborhood models: Issues

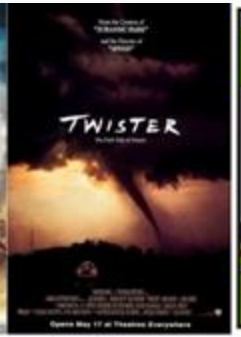
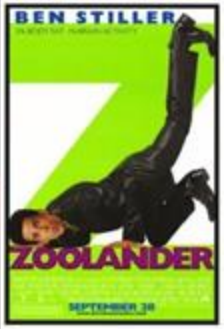



- Conceptually simple, but the devil is in the details!
  - How do you define ***similarity*** between users? Between items? By which metric?
  - How do you **aggregate feedback** over the neighborhood (from multiple similar users)?
  - User-based: Scaling issues (large user base, user turnover)
  - Item-based: Usually scales better and more stable, but might miss changes / trends

- Depending on feedback type, this can be difficult to scale with big data
  - Binary feedback ⇒ Jaccard similarity, MinHash+LSH will work
  - Otherwise … ? Most spatial data structures are not robust to missing data!

# In real life (big data) situations, data will be large (both rows and columns) and most data will be missing: We need to work **smarter**! → Latent factor models

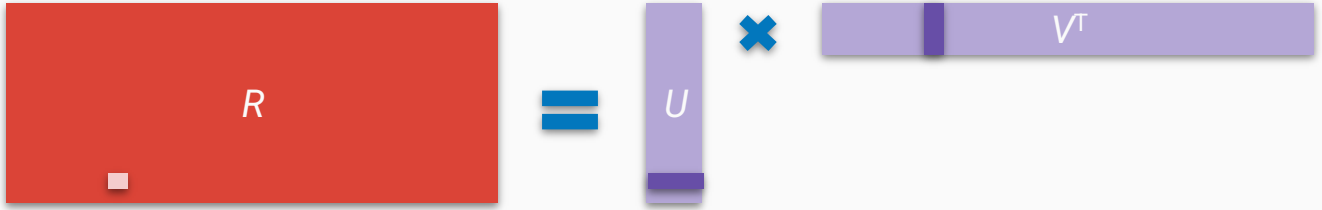
- **Latent factor models**
- Flexible framework for feedback modeling
  - Objective can be **tuned to match feedback** mechanism (e.g., ★★★☆☆ vs play counts)
  - Secondary objectives can be added (item bias, regularization, etc.)
- Usually easy to **parallelize** and scale up training
  - E.g.: alternating least squares.
  - Users are independent (conditional on items), and vice versa
- Learned representation is **low-rank** and dense
  - Integrates well with spatial data structures
  - Rank parameter provides control on complexity ⇔ expressivity

| Ratings matrix<br>Movie Ratings |  |  |  |  |  |
|---|---|---|---|---|---|
| Alex | 1 | 1 | 3 | 5 | 2 |
| Brett | 4.25 | 1.25 | 3 | 3.25 | 4.75 |
| Casey | 2 | 5 | 3 | 1 | 1 |
| Drew | 4.75 | 3.25 | 3 | 1.25 | 4.25 |
| Emory | 2.25 | 4.5 | 4.5 | 4.5 | 2.25 |
| Frankie | 2 | 1.75 | 2 | 1.75 | 2 |

# Latent factor models

- Bi-linear model of user-item interactions:

$$\min_{U,V} \sum_{(i,j)\in\Omega} (R_{ij} - \langle U_i, V_j \rangle)^2$$




- *U* and *V* are embeddings of users and items into a common vector space
- Model parameters (*U, V*) are learned to predict sparse observations *R*

*Reconstructing ratings by matrix multiplication*

| FP |  |  |  |
|---|---|---|---|
| A | 0 | 1 | 0 |
| B | 0.75 | 0.5 | 0 |
| C | 0 | 0 | 1 |
| D | 0.75 | 0 | 0.5 |
| E | 0 | 0.75 | 0.75 |
| F | 0.25 | 0.25 | 0.25 |

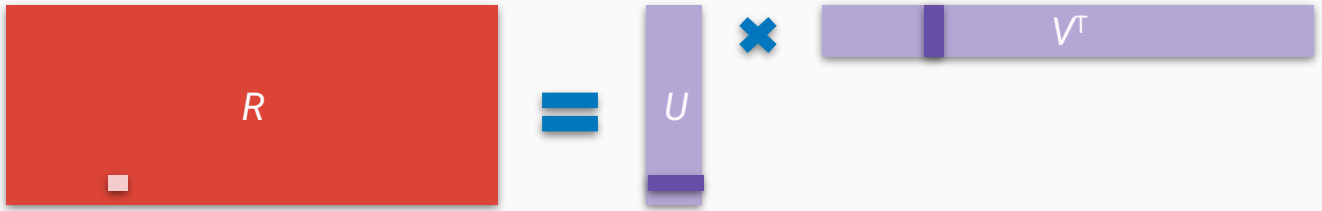
| FC | H | J | S | T | Z |
|---|---|---|---|---|---|
|  | 5 | 1 | 2 | 1 | 5 |
|  | 1 | 1 | 3 | 5 | 2 |
|  | 2 | 5 | 3 | 1 | 1 |

| Rat | H | J | S | T | Z |
|---|---|---|---|---|---|
| A | 1 | 1 | 3 | 5 | 2 |
| B | 4.25 | 1.25 | 3 | 3.25 | 4.75 |
| C | 2 | 5 | 3 | 1 | 1 |
| D | 4.75 | 3.25 | 3 | 1.25 | 4.25 |
| E | 2.25 | 4.5 | 4.5 | 4.5 | 2.25 |
| F | 2 | 1.75 | 2 | 1.75 | 2 |

# Alternating least squares

$$\min_{U,V} \sum_{(i,j)\in\Omega} (R_{ij} - \langle U_i, V_j \rangle)^2$$

- Repeat:
  - Fix user factors $U$, solve for item factors $V$
  - Fix item factors $V$, solve for user factors $U$




- When user factors are fixed:
  - Each item vector $V_i$ can be solved independently of the others $V_j$ ← ***parallelism***!
  - Optimization problem is equivalent to ordinary least squares regression

# Summary: **ALS via OLS**

- **Problem**: Neighborhood models have problems with sparse feedback matrices (most data is missing), as well as with integration of feedback.
- **Solution**: We decompose the feedback matrix (e.g. a ratings matrix R) into two smaller matrices, U (representing the user preferences for particular features) and V (representing feature content of the items).
- **Approach**: We initialize U and V with random values (or with values from a SVD), then using OLS to alternate between steps where we treat the entries in the V matrix as the parameters and the U matrix as the data and a step where we treat the entries in the U matrix as the parameters and the entries in the V matrix as the data.
- **OLS**: We do this with OLS (iteratively / alternatively).
- Until **convergence**: RMSE between the between the dot product of U and $V^T$ and the actually existing ratings in the R matrix converges.

# ALS step by step

- We start with a feedback matrix **R** (dimensionality n x m).
- We pick the hyperparameter **k** as the number of features to consider (or get this from SVD as the number of non-neglible singular values).
- We initialize a matrix **U** as n x k and a matrix **V** as m x k.

OLS normal equation:

$$\boldsymbol{\beta} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\mathbf{y}$$

- We now compute (per user i from 1 to n for all existing ratings r):

$$U_i = (\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{V})^{-1}\boldsymbol{V}^{\mathrm{T}}\mathbf{R_r}$$

- Then, we compute (per item j from 1 to m for all existing ratings r):

$$V_i = (\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{U})^{-1}\boldsymbol{U}^{\mathrm{T}}\mathbf{R_r}$$

- Then compute RMSE($R_r$,$UV^T$). If this term doesn't change or drops under some threshold, stop. Otherwise repeat the last 2 steps.

# More tricky: Modeling implicit feedback

- Count data (e.g. number of times listened to some song) is hard to predict
  - And we don't really care about that anyway!

- Instead, predict **binary interaction**, but use **counts to weight** terms!

$R_{ij}$: Raw interaction count

α: hyperparameter (how much repeated interactions matter)

$$\min_{U,V} \sum_{(i,j)\in\Omega} c_{ij}(p_{ij} - \langle U_i, V_j \rangle)^2$$

$$p_{ij} = \begin{cases} 1 & R_{ij} > 0 \\ 0 & R_{ij} = 0 \end{cases}$$

$$c_{ij} = 1 + \alpha R_{ij}$$

User showed interest

User did not

$C_{ij}$ = Confidence

[Hu, Koren, Volinsky 2008]

# Modeling implicit feedback for recommendations

- Count data (e.g. number of times listened to some song) is hard to predict
  - And we don't really care about that anyway!

- Instead, predict **binary interaction**, but use **counts to weight** terms!

$R_{ij}$: Raw interaction count

α: hyperparameter (how much repeated interactions matter)

$$\min_{U,V} \sum_{(i,j)\in\Omega} c_{ij}(p_{ij} - \langle U_i, V_j \rangle)^2$$

$$p_{ij} = \begin{cases} 1 & R_{ij} > 0 \\ 0 & R_{ij} = 0 \end{cases}$$

$$c_{ij} = 1 + \alpha R_{ij}$$

User showed interest

User did not

$C_{ij}$ = Confidence

- You don't have to use R directly. There are other implementations:
  - Simply drop low counts
  - Compress large values $R_{ij} \rightarrow \log(1 + R_{ij})$

[Hu, Koren, Volinsky 2008]

# Another practical issue of recommender systems with latent factor models: Handling new items

- Beware: This approach gives **no representation** to items with **no interactions**
- ⇒ A new item will never be recommended until it has representation!
- This is known as the ***cold-start*** problem
- Solutions typically involve
  - Active promotion / manual curation
  - Complementing this with content-based modeling

# Content-based models

Suppose you have observed features $x_j$ for each item

- News ⇒ topics, location, source
- Movies ⇒ genre, year, length, language, director, actors, …
- Music ⇒ metadata + acoustic attributes

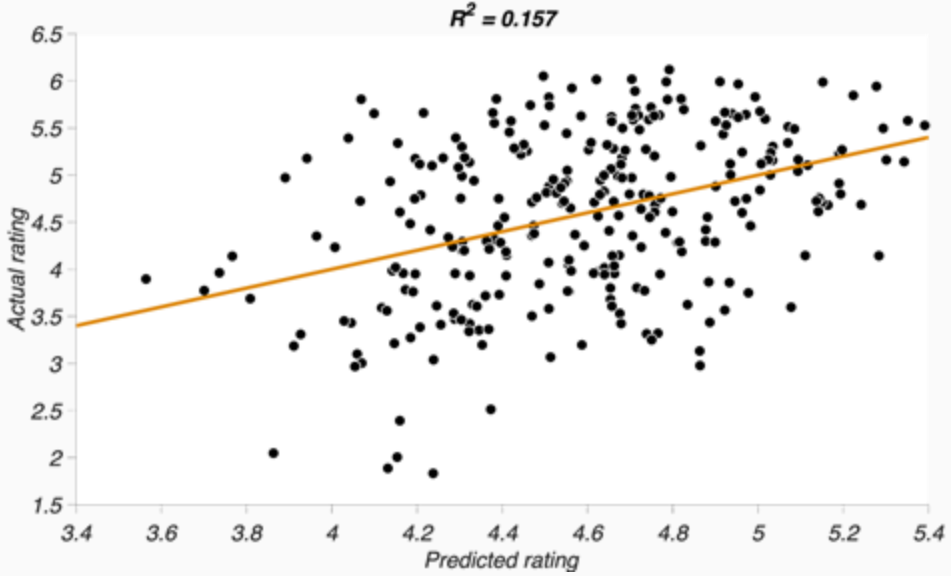
- **Content-based model**
  - Each user *i* gets their own interaction model $u_i$

    $$R_{ij} \approx \langle u_i, x_j \rangle$$

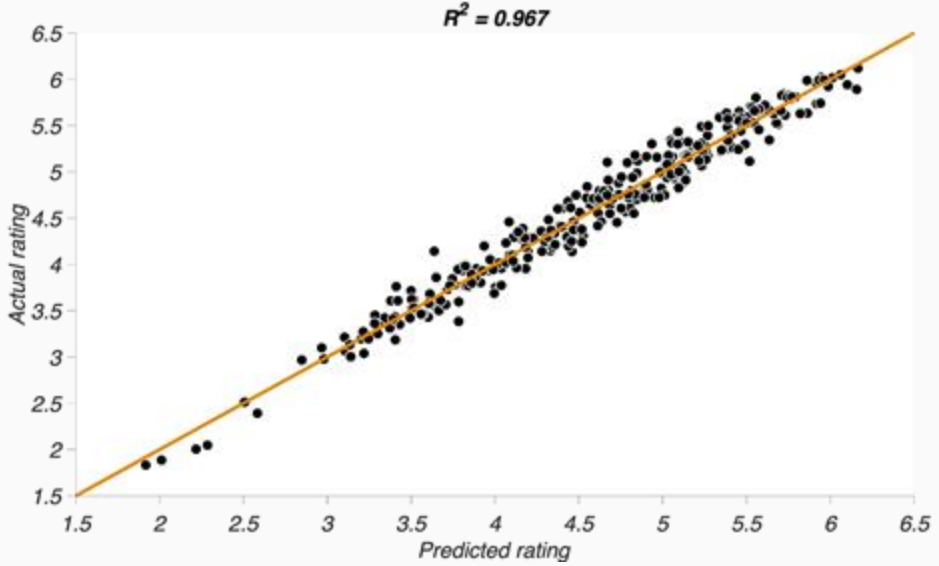
  - Like LF model, but the item factors are **explicit**
  - Can be limiting / over-constrained

# Challenge: What if the content features are not that predictive?

## Prediction from audio features




## Prediction from evoked emotions



# Content-based models

Suppose you have observed features $\boldsymbol{x}_j$ for each item

- News ⇒ topics, location, source
- Movies ⇒ genre, year, length, language, director, actors, …
- Music ⇒ metadata + acoustic attributes

- **Content-based model**
  - Each user $i$ gets their own interaction model $u_i$

    $$R_{ij} \approx \langle u_i, x_j \rangle$$

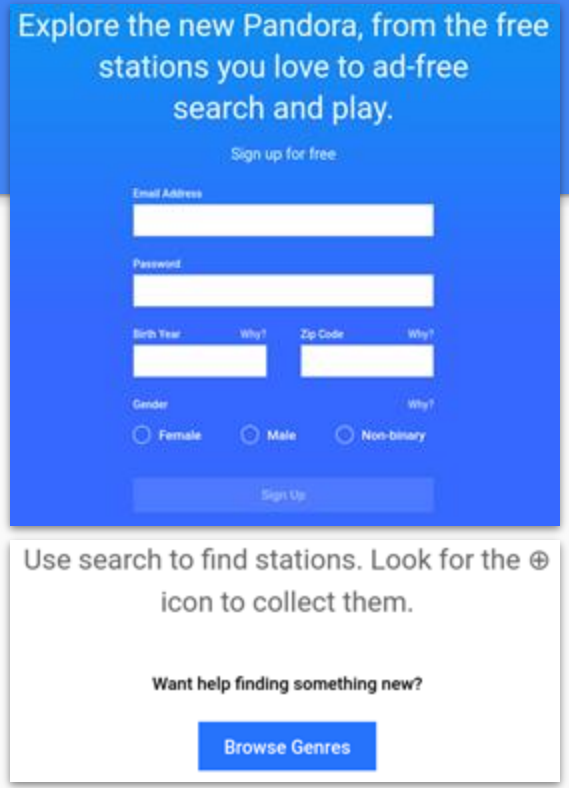
  - Like LF model, but the item factors are **explicit**
  - Can be limiting / over-constrained
- **Content cold-start**
  - Train a LF model as before, item $j \rightarrow V_j$
  - Then regress item factors from features

    $$V_j \approx \boldsymbol{f}(x_j)$$

  - A new item can map into embedding space by learned mapping $\boldsymbol{f}(x_k)$ (e.g. linear regression)

# User warm-start

- What happens when a new user enters a system?
- Typical systems request some **demographic data**, and ask for **examples of things you like**
- These data are used to position you in the collaborative filter




Pandora.com, 2022

# Evaluation of recommender systems

- $R_{ij} \approx \langle u_i, v_j \rangle$
- $R_{ij} \approx b_j$
- $R_{qj} \approx \text{Jaccard}(q, v_j)$

# Task

- Predict a ranked list of items
  - Could be from collaborative filter…
  - Could be from popularity…
  - Could even be from a search query…
- What is the workflow for recommender systems modeling?
- How should we evaluate a model?

# Working with recommender data

- It's tempting to think of observations (u, v) as independent, but they aren't!
  - Different items for the same user ⇒ not independent
  - Different users for the same item ⇒ not independent
- What are we actually **predicting**, and what do we **value**?
  - Typically we care about satisfying a **user** – given the list of recommendations
  - This should influence our evaluation criteria (beyond RMSE) – unseen relevant?
- Most interfaces provide several recommendations at once
  - Evaluate the **collection of recommendations** per user
  - **Average across users** to estimate system performance

# **Partitioning** data for recommender systems

- We often need to partition training data for validation / parameter tuning
- It's tempting to randomly split interactions, but this can fail badly, as recommendations are not necessarily independent of users/items.
- The model needs at least some history for each user that it will predict upon, and sequence could matter
- **⇒ Partition each user's observations into train / val separately**

# Evaluating recommender systems

- Modeling objectives are usually just a proxy for our main goal

- Early RecSys work focused on **mean squared error** (MSE) of star ratings
  - $\text{loss}(u_i) = \mathbf{E}_j[(\ R_{ij}\ -\ \langle u_i\ ,\ v_j \rangle\ )^2]$
  - (Thanks, NetFlix)

# Evaluating recommender systems

- Modeling objectives are usually just a proxy for our main goal
- Early RecSys work focused on **mean squared error** (MSE) of star ratings
  - $\text{loss}(u_i) = \mathbf{E}_j[(\ R_{ij}\ -\ \langle u_i\ ,\ v_j \rangle\ )^2]$
  - (Thanks, NetFlix)
- Instead, think about how recommendations are ***delivered***
  - Ranked list? (NetFlix, Google search results, Amazon)
  - One at a time? (Pandora, streaming radio, YouTube autoplay)
- ***Evaluation should reflect user behavior!***

# Bipartite ranking evaluation

- **Idea**: rank items in order of decreasing estimated relevance for the user
  - Item 5, Item 17, Item 209, Item 444, Item 682, Item 999, Item 128, …
- Using **held-out interactions**, determine which items were **relevant** (**+**) or **irrelevant** (**-**):
  - {17, 444, 682,…} ⇒ **-**, **+**, **-**, **+**, **+**, **-**, **-**, **…**
- Score the sequence of **+** and **-**
  - Ideally, all **+** come before all **-**
  - All **+**'s assumed to be equally good, all **-**'s equally bad ⇒ "Bipartite ranking"
  - Particularly useful for implicit feedback (clicks, plays)

**Ranking evaluations**: Metrics to assess recommendation models that focus on the order in which items are presented, with the assumption that the order is relevant for the user experience

**Prediction**: ordered list of items, **reference data**: held-out interactions

- **AUC** (area under ROC curve)
  - How often does a **+ interaction** rank ahead of a **- interaction**?
  - **-+-++--** ⇒ (3 + 2 + 2) / (3 * 4) = 7/12 = 0.583

**Ranking evaluations**: Metrics to assess recommendation models that focus on the order in which items are presented, with the assumption that the order is relevant for the user experience

**Prediction**: ordered list of items, **reference data**: held-out interactions

- **AUC** (area under ROC curve)
  - How often does a **+ interaction** rank ahead of a **- interaction**?
  - **-+-++--** ⇒ (3 + 2 + 2) / (3 * 4) = 7/12 = 0.583

- **Average Precision** (AP)
  - For each **+ interaction**, what fraction of higher-ranked items were also **positive**?
  - **-+-++--** ⇒ ⅓ * (½ + ½ + ⅗) = 0.533 [the lead 1/3 = 1/n yields the average)

**Ranking evaluations**: Metrics to assess recommendation models that focus on the order in which items are presented, with the assumption that the order is relevant for the user experience

**Prediction**: ordered list of items, **reference data**: held-out interactions

- **AUC** (area under ROC curve)
  - How often does a **+ interaction** rank ahead of a **- interaction**?
  - **-+-++--** ⇒ (3 + 2 + 2) / (3 * 4) = 7/12 = 0.583
- **Average Precision** (AP)
  - For each **+ interaction**, what fraction of higher-ranked items were also **positive**?
  - **-+-++--** ⇒ ⅓ * (½ + ½ + ⅗) = 0.533 [the lead 1/3 = 1/n yields the average)
- **Reciprocal rank** (MRR)
  - The inverse rank position of first **+ interaction**
  - **-+-++--** ⇒ ½

**Ranking evaluations**: Metrics to assess recommendation models that focus on the order in which items are presented, with the assumption that the order is relevant for the user experience

**Prediction**: ordered list of items, **reference data**: held-out interactions

- **AUC** (area under ROC curve)
  - How often does a **+ interaction** rank ahead of a **- interaction**?
  - **-+-++--** ⇒ (3 + 2 + 2) / (3 * 4) = 7/12 = 0.583
- **Average Precision** (AP)
  - For each **+ interaction**, what fraction of higher-ranked items were also **positive**?
  - **-+-++--** ⇒ ⅓ * (½ + ½ + ⅗) = 0.533 [the lead 1/3 = 1/n yields the average)
- **Reciprocal rank** (MRR)
  - The inverse rank position of first **+ interaction**
  - **-+-++--** ⇒ ½

Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR): Users are assumed to be independent, and scores are averaged across users at evaluation time.

# Beyond bipartite ranking metrics

Bipartite ranking metrics work well when the feedback *is* binary (clicks, likes, views, listens)

There is a space for a ranking evaluation metric (result is a ranked list) that takes graded relevance (e.g. star ratings) into account.

Most commonly used in this scenario:
Normalized Discounted Cumulative Gain (**NDCG**)

NDCG allows to model graded relevance, while also considering the ranking of the results.

Philosophy: "Reward the model for ranking good items higher, especially the best ones."

# How NDCG works (mathematically)

| Movie | Stars |
|---|---|
| Avatar (2009) | 2 |
| Big Short (2015) | 5 |
| Cats (2019) | 1 |
| Dune (2021) | 4 |
| Elysium (2013) | 3 |

Key idea: There is an ideal order – NDCG represents how well the ranking our model produces compares to that ideal (with a metric ranging from 0 to 1)

$$NDCG@k = \frac{DCG@k}{IDCG@k}$$

$$CG@k = \sum_{i=1}^{k} rel_i$$

Ideal order: B, D, E, A, C

CG@3 = 12

How does a given recommended order compare to that? D,C,B,E,A CG@3 = 10

In addition to the sum, the presentation order should matter too: It is better to serve higher value items earlier. Items later in the list should matter less.

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}$$

This is the linear case. If you want to reward bringing high-relevance items more, use exponential gain:

$$DCG@k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

# In real life, things are even more complicated…

- In practice, recommender systems exist in a **feedback loop**
- These are extremely difficult to measure offline from **observational data**
  - Interpret ranking metrics with healthy skepticism!
- Competing models are often evaluated by A/B testing, and measuring some dependent variable
  - Engagement, sales, etc.
- Recently, focus is shifting to reinforcement learning and causal models

# Evaluation summary

- Properly evaluating a recommender system is not easy!
- Ranking metrics are a start, but there's much more to it than "accuracy"
  - Diversity? Novelty? Serendipity?
  - Efficiency? Ease of use?
  - Explainability / transparency?
  - Adverse effects on users?

# Next week

- Graph algorithms (esp. page rank)
- Spatial search (vector databases)

# After that

- Ethics (filter bubbles, polarization)
- Privacy

# Q&R