summary

recommendation systems = rank options to help with decision making

assumption: users $\gg$ items $\gg$ ratings

types

- popularity-based $\longrightarrow$ what most other users like
  - data: community (not personal)
- demographic-based $\longrightarrow$ what most other users - similar to you - like
  - data: user, community
- user-user-cf $\longrightarrow$ what most other users - with similar taste to you - like
  - predicts your next rating, trained on other user's data
  - data: user, community
- item-item-cf $\longrightarrow$ what items are most similar to what you like - based on what others say
  - item similarity based on user ratings
  - data: user, community
- content-based $\longrightarrow$ what items are most similar to what you liked so far
  - data: user, product features
- knowledge-based $\longrightarrow$ what fits your needs
  - data: user, product features, knowledge model
- hybrid $\longrightarrow$ combined

components

- system owner:
  - content provider
- recommender system:
  - computes relevance, generates recommendation list
- U user:
  - receives recommendations
  - user profile = demographic data, feedback, preferences
  - user preferences = taste, intention towards items
  - user feedback = implicit (engagement), explicit (rating)
- I item:
  - thing to be recommended
- R rating
  - $r_{ui} \in R$
  - sparse matrix
- $s(u, i)$ score / predicted rating
  - $\hat{r}_{ui}$ for target-user and target-item
- k total number of features
  - this can be users, items, timestamps and any other data used to predict a missing rating from the rating matrix

feedback types

- explicit feedback:
  - = user-item rating matrix
  - less available, stronger signal

- implicit feedback:
    - = engagement (consumption data, clicks, page views, ...) in a binary matrix
    - more available, weaker signal (harder to interpret)
    - missing data: treat as 0s, then avoid predicting 0 for unknown values
    - mean centering: scale by how much feedback a user gives
    - needs customizable models like: weighted matrix factorization, bayesian personalized ranking

classification types

- a) memory-based
    - = store entire history of user-item interactions
    - examples: neighborhood methods, uu-cf, ii-cf
- b) model-based
    - = extract features from history and only store a representation that describes it
    - examples: matrix factorization, factorization machines, sparse linear method, neural networks, weighted matrix factorization, bayesian personalized ranking

user-user colaborative filtering

compare all users of target-item

- idea: predict rating of target item, by taking the average rating of all users similar to target user
- input:
    - user feedback, ratings in sparse user-item matrix
    - no content information
- pros: highly personalized recommendations, interesting results, can learn market segments
- cons: challenges with scalability, sparse data, new users without sufficient ratings.

predicted rating

- non personalized:
    - average rating of item, by all users (that have rated the item)
    - $U_i = \{u \in U | r_{ui} \in R\}$
    - $s(u,i) = \frac{\sum_{v \in U_i} r_{vi}}{|U_i|}$
- weighted by similarity:
    - weigh more similar users heigher
    - $w_{uv} \in [\text{-}1, 1]$
    - $s(u,i) = \frac{\sum_{v \in U} w_{uv} \cdot r_{vi}}{\sum_{v \in U} |w_{uv}|}$
- unbiased:
    - mean-centering, since ie. a 10/10 can be interpreted differently
    - $s(u,i) = \overline{r_u} + \frac{\sum_{v \in U} w_{uv} \cdot (r_{vi} - \overline{r_v})}{\sum_{v \in U} |w_{uv}|}$
- neighborhood:
    - only consider subset of all users
    - subset of users with some level of similarity to reduce noise in rating
    - $s(u,i) = \overline{r_u} + \frac{\sum_{v \in N(u)} w_{uv} \cdot (r_{vi} - \overline{r_v})}{\sum_{v \in N(u)} |w_{uv}|}$

user similarity: pearson correlation pcc

- users are similar, if they rate similarly → correlation increases with similar frequency and intensity of ratings
- mean centered, -1;1 normalized, sum of mutual ratings
- note: if you just iterate over mutually rated items, then users who have just 1 common item, are understood as identical. there are ways to fix this.

- $I_u = \{i \in I \mid r_{ui} \in R\}$
- $w_{uv} = \frac{\sum_{i \in I_u \cap I_v} (r_{ui} - \overline{r_u}) \cdot (r_{vi} - \overline{r_v})}{\sqrt{\sum_{i \in I_u} (r_{ui} - \overline{r_u})^2} \cdot \sqrt{\sum_{i \in I_v} (r_{vi} - \overline{r_v})^2}}$

user similarity: cosine similarity

- pearson correlation is identical to mean-centered cosine-similarity
- $\mathbf{u}$ = ratings vector for user u
- $w_{uv} = \cos(\mathbf{u}, \mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|} = \frac{\sum_{i=1}^{n} u_i v_i}{\sqrt{\sum_{i=1}^{n} u_i^2} \cdot \sqrt{\sum_{i=1}^{n} v_i^2}}$

runtime optimization

- $|U|=m$
- $|I|=n$
- runtime:
  - i) find user neighbourhood of size k
    - get similarity between 2 users: $O(n)$
    - get similarity between all users: $O(nm^2)$
    - optimization: filter by users with at least 1 mutually rated item
  - ii) predict rating of a single item: $O(k)$
    - optimization: filter by items that have rated the by at least 1 neighbor
  - iii) return highest ranked item: $O(n)$

item-item colaborative filtering

compare all items of target-user

- idea: items are similar if they have similar rating
- idea: predict rating of target item, by taking the average rating of all items similar to target item
- input:
  - item rankings
  - no content information
- pros: it is better than uu-cf
  - higher effectivity → items have more ratings, than users give ratings
  - higher efficiency → there are less items than users
  - higher stability → item-similarities are more stable than user-similarities (they change less frequently)
- cons: can lead to boring recommendations that are too similar to ones already rated

predicted rating

- non personalized:
  - average rating of all items, by target-user
  - $I_u = \{i \in I \mid r_{ui} \in R\}$
  - $s(u, i) = \frac{\sum_{j \in I_u} r_{uj}}{|I_u|}$
- neighborhood:
  - weights w_{ij} for how similar items rated by target-user are to the target-item we want to predict the rating of
  - $s(u, i) = \overline{r_i} + \frac{\sum_{j \in N(i)} w_{ij} (r_{uj} - \overline{r_j})}{\sum_{j \in N(i)} |w_{ij}|}$

item similarity: pearson correlation

- items are similar, if they get rated by users similarly
- $U_i = \{u \in U | r_{ui} \in R\}$

- $w_{ij} = \frac{\sum_{u \in U_i \cap U_j}(r_{ui} - \overline{r_i}) \cdot (r_{uj} - \overline{r_j})}{\sqrt{\sum_{u \in U_i}(r_{ui} - \overline{r_i})^2} \cdot \sqrt{\sum_{u \in U_j}(r_{uj} - \overline{r_j})^2}}$

runtime optimization

- $|U| = m$
- $|I| = n$
- runtime:
  - i) find item neighbourhood of size k
    - get similarity between 2 items: $O(m)$
    - get similarity between all items: $O(n^2 m)$
    - optimization: filter by items rated by at least 1 mutual user
  - ii) predict rating of a single item: $O(k)$
    - optimization: precompute and cache offline model of neighbours - works well because item ratings change more slowly
  - iii) return highest ranked item: $O(n)$

matrix factorization

generating representations / embeddings of R for improved effectivity (feature extraction) and efficiency (data compression).

SVD

- $M = U \Sigma V^T$
- = singular value decomposition
- can factorize any matrix

truncated SVD

- $\hat{M} = \hat{U} \hat{\Sigma} \hat{V}^T$
- = only keep the k largest values in each component
- = low-rank matrix approximation with minimal errors (under squared root)

when applied on the rating-matrix (many missing values of which we're trying to predict):

- $\hat{R} = \hat{U} \hat{\Sigma} \hat{V}^T$
- $\hat{r}_{ui} = \sum_f \hat{U}_{uf} \cdot \hat{\Sigma}_{uf} \cdot \hat{V}_{if}$
- where:
  - k = number of features / latent variables that describe user preferences (ie. comedy, drama, $\dots$)
  - $\hat{\Sigma}$ = significance of features (weights)
  - $\hat{U}$ = user features (interests)
  - $\hat{V}$ = item features (descriptions)

matrix factorization

- = truncated-SVD without the $\hat{\Sigma}$ matrix
- no longer an SVD, but the 2 factors can now be learned (ie. with alternating-least-squares ALS or stochastic-gradient-descent SGD)
- $\hat{R} = P^T \cdot Q$
- $\hat{r}_{ui} = p_u^T q_i \longrightarrow$ each matrix column is an user/item embedding
- where:
  - P = user features
  - Q = item features

SGD

- = stochastic gradient descent
- we want to learn P, Q
- non-convex optimization problem: minimize the total cost function J

cost function:

- $\theta \in \{P, Q\}$
- $J(\theta) = \frac{1}{|R|} \sum_{r_{ui} \in R} \underbrace{J^{(u,i)}(\theta)}$
  - average of the cost function for a single prediction
- $J(\theta) = \frac{1}{|R|} \sum_{r_{ui} \in R} \underbrace{e_{ui}^2} + \underbrace{\lambda(\|P\|^2 + \|Q\|^2)}$
- $J(\theta) = \frac{1}{|R|} \sum_{r_{ui} \in R} \underbrace{(r_{ui} - p_u^\intercal q_i)^2} + \underbrace{\lambda \left(\sum_u \sum_f p_{uf}^2 + \sum_i \sum_f q_{if}^2 \right)}$
  - squared error = actual rating - predicted rating
  - L2 or frobenius norm = limiting the vec magnitude by λ to avoid overfitting

gradients:

- $\frac{\partial J^{(u,i)}}{\partial p_u} = -2e_{ui} \cdot q_i + 2\lambda \cdot p_u$
- $\frac{\partial J^{(u,i)}}{\partial q_i} = -2e_{ui} \cdot p_u + 2\lambda \cdot q_i$

algorithm:

- i. randomly shuffle ratings R
- ii. randomly initialize P, Q
- iii. until convergence for each $r_{ui} \in R$
  - $e_{ui} := r_{ui} - p_u^\intercal q_i$
  - $\theta := \theta - \eta \cdot \frac{\partial J(\theta)}{\partial \theta}$ $\longrightarrow$ means subtracting gradient with learning rate η
    - $q_i := \frac{\partial J^{(u,i)}}{\partial q_i} = q_i + \eta \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$
    - $p_u := \frac{\partial J^{(u,i)}}{\partial p_u} = p_u + \eta \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$

SGD++

no longer just $\hat{r}_{ui} = p_u^T q_i$

$$\hat{r}_{ui} = \underbrace{\mu + b_u + b_i}_{\text{baseline estimate}} + \underbrace{q_i^\intercal}_{\text{item model}} \cdot \left(\underbrace{p_u}_{\text{user model}} + \underbrace{|N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j}_{\text{implicit feedback}} \right)$$

- baseline estimate / mean-centering:
  - $b_{ui} = \mu + b_i + b_u$
  - where:
    - μ = average rating, among all ratings
    - b_i = bias in item ratings
    - b_u = bias in user ratings
  - gradient descent should also learn b_i, b_u in addition to q_i, p_u:
    - $b_u := b_u + \eta \cdot (e_{ui} - \lambda \cdot b_u)$
    - $b_i := b_i + \eta \cdot (e_{ui} - \lambda \cdot b_i)$
- implicit feedback:
  - $|N(u)^{-\frac{1}{2}}| \cdot \sum_{j \in N(u)} y_j$
  - where:
    - $\sum_{j \in N(u)} y_j$ = implicit-feedback feature vector y_i for item i
    - $|N(u)^{-\frac{1}{2}}|$ = normalization
    - $N(u)$ = set of items user u has engaged with

weighted matrix factorization WMF

reduces ambiguity of implicit-feedback by using additional information to determine a confidence weights in loss function:

- $J(\theta) = \frac{1}{n \cdot m} \sum_{u,i} w_{ui} \cdot J^{(u,i)}(\theta)$
- $|R| \neq n \cdot m$
- observed feedback: $w_{ui} = 1 + \alpha \cdot c_{ui}$
  - measure frequency to avoid false positives (ie. engagement because of dislike)
  - α = significance
  - c_{ui} = interaction count
- missing feedback: $w_{ui} = w \cdot \frac{f_i^\alpha}{\Sigma_j f_j^\alpha}$

- measure exposure to avoid false negative (ie. no engagement because they don't know of existence)
- α = significance
- w = fixed constant
- f_i = popularity of item i

*bayesian personalized ranking BMF

improves ranking of implicit-feedback through pair-wise learning (instead of point-wise-learning) and preference of items with predicted implicit-feedback over those without:

- we want to learn all preferences of one item over the other
- $\hat{x}_{uij}$ = preferring i over j for user u
- $\hat{x}_{uij} = \hat{r}_{ui} - \hat{r}_{uj}$
- we can map this value to a probability-of-preference, by normalizing it to 0;1
- learn $\hat{x}_{uij}$ through matrix-factorization

factorization machines

factorization-machines

- higher-order linear-regression to learn weights and feature embeddings:
  - $\hat{r}_{ui} = \underbrace{\mu}_{(0)} + \underbrace{\sum_p w_p \cdot x_p^{ui}}_{(1)} + \underbrace{\sum_p \sum_{p' \neq p} w_{pp'} \cdot x_p^{ui} \cdot x_{p'}^{ui}}_{(2)}$
- where:
  - $x^{ui} \in X$ = embedding of all features (not just user, item but also timestamps etc.)
  - 0th order: μ
    - = bias, global average rating
  - 1st order: w_p
    - = weight to be learned for each feature p
  - 2nd order :$w_{pp'} = v_p^\top v_{p'}$
    - = interdependence weight to be learned for each pair of features $p, \ p'$
    - would take too long to learn, use dot product of feature embeddings v_p instead (factorization)
    - v_p has k dimensions

compared to matrix-factorization

- $\hat{r}_{ui} = \underbrace{\mu}_{(0)} + \underbrace{b_u + b_i}_{(1)} + \underbrace{p_u^T q_i}_{(2)}$
- $x^{ui} \in X$ only contains user, item as features
- learned with sgd, not linear regression
- not as flexible as SVD++ but can be extended as well

sparse linear methods

sparse linear methods SLIM

idea: learning something similar to the item-weights we multiplied to each user-rating in ii-cf

- assuming that weights are normalized, the prediction can be
  - $\hat{r}_{ui} = \sum_j w_{ij} \cdot r_{uj}$
  - $\hat{R} = RW$

- we can learn the weight by factorizing it
  - $W = Q^T T$
  - $\hat{R} = RQ^\intercal Y$
- therefore
  - $\hat{r}_{ui} = \sum_j r_{uj} \cdot q_i^T \cdot y_j = \underbrace{q_i^T}_{\text{item}} \cdot \underbrace{\sum_j r_{uj} \cdot y_j}_{\text{user}}$

neural networks

restricted boltzmann machines RBM

- similar to autoencoders: generate embeddings in hidden layer
- no longer relevant:
  - one input node for each item, as 1-hot-encoding of values [1;5]
  - hidden nodes are called stochastic-binary-units (binary value with some random probability)
  - trained with constrastive-divergence

autoencoders

- feed-forward neural net to generate embeddings
- usage:
  - autoRec: generates embeddings for users and items
  - collaborative-denoising-autoencoders cdae: denoises implicit feedback - ie. denoising and predicting next clicks of target user
- architecture:
  - input: data to be encoded
  - hidden layer: bottleneck with fewer nodes than input, to generate embedding (stored)
  - output: reconstruction of input embedding → should have minimal information loss $h_{W,b}(x) = \hat{x} \approx x$

matrix factorization as neural network

- assumption: no biases, linear activation
- input: x
  - as 1-hot encoding of item
  - passed to hidden layer with weights Q
- hidden: $z = Qx$
- output: $y = P^\intercal z = P^\intercal Qx$
  - $P_u^\intercal Q_i = y_u = \hat{r}_{ui}$
  - the u'th row in vector y is the predicted rating of item i for user u

content based recommenders

- based on information retrieval
- improved transparency, explainability
- no cold-start for items
- users can land in a filter-bubble and get repetitive recommendations
- input:
  - only user feedback history, no collaboration (user independence)
  - item content

vector space model VSM

sparse representation of document as vector (traditional way)

- tf-idf: calculate weight of a term in doc
- $w_{td} = tf_{td} \cdot idf_t$
  - $tf_{td} = \log(1 + f_{td}) \longrightarrow$ limit effect of frequency
  - $idf_t = \log(\frac{N}{df_t}) \longrightarrow$ rare terms should be more significant

similarity-weighted-average

- $\hat{r}_{uq} = \dfrac{\sum_{d \in N(q;u)} \cos(q,d) \cdot r_{ud}}{\sum_{d \in N(q;u)} \cos(q,d)}$
- finding neighborhood $N(q)$:
  - compare tf-idf of item from user-history d_i (document) to target-item q_i (query) through $\cos(\mathbf{q}, \mathbf{d})$

rocchio's method / relevance feedback

we can use the rating of a user to create a user-profile-vector that we then compare with item-vectors through cosine-similarity

- see: https://en.wikipedia.org/wiki/Rocchio_algorithm
- i. split feedback history in rel-docs D^+ or non-rel-docs D^-
- ii. compute user-profile-vector:
  - $u = \alpha \cdot \underbrace{u_0} + \beta \cdot \underbrace{\frac{1}{|D^+|} \sum_{d^+ \in D^+} d^+} - \gamma \cdot \underbrace{\frac{1}{|D^-|} \sum_{d^- \in D^-} d^-}$
  - where u_0 is some baseline vector for bias

evaluation metrics

prediction accuracy

- mean of absolute errors MAE
  - $e_{ui} = r_{ui} - \hat{r}_{ui}$
  - $\frac{1}{|R|} \sum_{r_{ui} \in R} |e_{ui}|$
- mean of squared errors MSE
  - $\frac{1}{|R|} \sum_{r_{ui} \in R} e_{ui}^2$
- root of mean of squared errors RMSE
  - $\sqrt{\frac{1}{|R|} \sum_{r_{ui} \in R} e_{ui}^2} \longrightarrow$ penalizes variations harder

classification accuracy

- precision P
  - $P = TP/(TP + FP)$
  - correctness = only recommending relevant items
- recall R
  - $R = TP/(TP + FN)$
  - completeness = not leaving out any relevant items
- F1
  - $F_1 = 2 \cdot \dfrac{P \cdot R}{P + R} = \dfrac{2}{\frac{1}{P} + \frac{1}{R}}$
  - harmonic mean of precision and recall

ranking accuracy

binary labels (rel vs. non-rel)

- precision P@k:
  - precision at rank cutoff

- recall R@k:
  - recall at rank cutoff
- PR-curve:
  - precision-recall graph at all cutoffs
- average precision AP@k:
  - average of the precision scores when recall increases at cutoffs
  - when averaged across all users it's called the mean-average-precision MAP

graded labels (rel scores)

- discounted cumulative gain DCG:
  - intuition: relevance score / logarithm of absolute position for each item
  - $DCG(D) = \sum_{j=1}^{k} \frac{r_{u,i_j}}{\log(j+1)}$
  - j = absolute position in ranking
  - discounted relevance means it normalizes score based on rank
- normalized discounted cumulative gain nDCG:
  - intuition: current dcg divided by the dcg of correctly sorted rel-docs
  - $nDCG(Q) = \frac{1}{|Q|} \sum_{q \in Q} \frac{DCG(q)}{DCG(\text{sorted}(rel(q)))}$
  - normalizes dcg again by best possible ranking per query $DCG(\text{sorted}(rel(q)))$ / ideal dcg IDCG - meaning the docs being in the correct order based on their relevance for the given query

k-fold cross validation (before evaluation)

- 80% train dataset:
  - dev/val – 1/k partition used to fine tune and find the right parameters
  - train – other partitions used to train model
- 20% test dataset:
  - test – used to evaluate model, has similar distribution to train set

online vs. offline

- offline = get metrics from datasets
- online = get metrics while system is running
- user-study = ui/ux research

qualitative user-study metrics

- trust = recommendation explainability, repeatable recommendations
- diversity = diversity within the recommendation list of users
- novelty/serendipity = diversity in the recommendation history of users
- all metrics [tintarev et. al]: trust, effectiveness, persuasiveness, efficiency, transparency, scrutability, satisfaction

coverage

- user coverage = fraction of users that can be recommended to (ie. smaller at cold-start)
- item coverage = fraction of items that can be recommended
- coverage inequality = skewed tail-heavy distribution (ie. measured by gini index)

significance testing

- choose significance-level/p-value to determine how likely it is that results are a coincidence

cold start

- new user:
  - non-personalized recommendations (popularity, demographics)
  - implicit feedback from random recommendations
- new item:
  - content based recommendations
  - recommend to users with broad taste first

- new system:
    - buy data
    - use knowledge based recommenders

sequence aware recommenders

session aware vs. session based

- long-term interests = general taste
- short-term interests = intention while using system (can be more significant)
- interaction log = sequence of actions taken by a specific user in a session (with unique id)
    - rich implicit feedback
    - arbitrary length, can be just the last action
    - lets us recommend alternatives, complements, continuations (next point-of-interest poi recommendation)
- a) session aware:
    - = user-account, cookies
    - access to short-term + long-term interests
- b) session based:
    - = anonymous user
    - access to short-term interests

making our algorithms session-aware

- treat each session like a user
- user-user cf:
    - = session-based kNN
    - $\hat{r}(s,i) = \sum_{s' \in N(s)} w_{s,s'} \cdot \mathbb{1}(i \in s')$
    - $N(s)$ = neighborhood of current session
    - $w_{s,s'}$ = session-session similarity
    - $\mathbb{1}(i \in s')$ = neighbors that contain target item
- item-item cf:
    - $\hat{r}(s,i) = \sum_{j \in s} w_{ij}$
    - sum of similarities of all current session items
    - $w_{i,j}$ = item-item similarity
- matrix factorization:
    - $w_{ij} = q_i^\intercal \cdot y_j$
    - $\hat{r}(s,i) = q_i^\intercal \sum_{j \in s} y_j = \sum_{j \in s} q_i^\intercal y_j$
    - must be trained for each new session
    - session is represented by features y of the items it contains

markov processes MP

- the state is a sequence:
    - $s = (i_i, \dots, i_k)$
- predicts the most likely future state, given present state (ignores the history, but you can encode it into the present state):
    - $P[\underbrace{S_{t+1} | S_t}_{\text{present}} = P[\underbrace{S_{t+1}}_{\text{future}} \mid \underbrace{S_1, \dots, S_t}_{\text{past}}]]$
- state transition probabilities:
    - $P_{s,s'} = P[S_{t+1} = s' | S_t = s]$
    - $P_{s,s'} = \frac{Pr[(i_1, \dots, i_{k+1})]}{Pr[(i_1, \dots, i_k)]}$

- probability of transition = how many times we see the transition happen $p_{s,s'}$ (s to s') divided by how many times we see the initial state (s)
- all transitions can be stored in multi-demensional matrix
  - state-space is too large
  - minimize sequence length
  - extract features, compress

recurrent neural networks

- ie. RNN, LSTM, GRU

further research topics

group recommenders

goal: combining preferences of multiple users into a single profile

types

- group type = established, occasional, random
- preferences of individual members = whether we have access to user histories
- recommendation consumption = how they're presented
- behavior of the group = passive / active negotiation and discussion of recommendations
- recommendation type = single item or a sequence of ranked items

aggregation strategies

- https://en.wikipedia.org/wiki/Social_choice_theory
- individual ratings can be aggregated by strategies from social-choice theory
- types:
  - majority-based strategies = use the most popular items (ie. plurality voting)
  - consensus-based strategies = consider preferences of all members (ie. average, average without misery, fairness)
  - borderline strategies = consider the preferences of a subset of group members (ie. dictatorship, least misery, most pleasure)
- heuristics-based methods make assumptions:
  - graph-based ranking = frequently visited paths
  - spearman-footrule ranking = least distance between preferences
  - nash-equilibrium strategy = tries to find nash equilibrium in non-cooperative game

psychology

- people can be studied through surveys or replies on examples shown to them (ie. a picture)
- metrics:
  - personal attributes = demographics, roles, personality, expertise, personal impact / cognitive certainty, ocean model, resolution strategy (collaborativeness vs. competitiveness)
  - group attributes = relationship strength, social trust, personal impact, conformity

metrics

- utility
- satisfaction
- fairness (based on participants, independent observers)
- privacy

generative recsys

new kind of recommender systems

- discriminative models → generative models:

- prediction tasks = top-k recommendation, rating prediction, …
    - generation tasks = conversational recommendation, explaination generation, …
- multi-stage traditional systems → single-stage ranking:
- single-task learning → multi-task learning:
    - p5 recommender paradigm = pretrain, personalized prompt, predict
    - tasks expressed as prompts/queries
    - needs fine-tuning
    - uses beam-search (best-first graph search) to prune search tree
    - can do few/zero-shot recommendations

risks

- can be biased and discriminating
- hallucinate

fairness

fairness

each stakeholder has a different understanding of fairness

- individual fairness = similar individuals have similar experiences
- group fairness = different groups have similar experiences
- anti-classification = specified attributes shouldn't be discriminatory
- anti-subordination = it should be attempted to undo past harm (ie. historic injustice)
- …

harm

- types:
    - distributional harm = systematically unfair distribution of resources
    - representational harm = inaccurate systematically unfair internal representation
- examples:
    - information asymmetry = not sharing critical information
    - matthew effect = accumulated advantage of already advantaged users
    - echo chambers = radicalization of users through constant reinforcement

source of bias

- training data (ie. gender bias)
- model architecture
- bad metrics / evaluation

mitigation

- responsibility
- transparency
- explainability

human-centered recsys

- psychology-informed: designed to respect user needs and values
- cognition-inspired, personality-aware, affect-aware - recommender systems

conversational recsys

conversational ai

- conversational recsys
- conversational search
- conversational qa
- social chatbot
- voice commanding

conversational recsys CRS

- help with decision making through conversation
- types:
  - single vs. multi-round interaction
  - short question-answering dialogue vs. prolonged chitchat
  - standalone service vs. embedded features into a larger service
  - communication medium (hand written, spoken, web form, …)
  - available data and knowledge (item catalogue, item content, user intent, interaction history, world knowledge, dialogue corpora, …)
  - architecture (RL agents, LLMs, …)
  - …

tasks

- request
- respond
- recommend
- explain
- …