

# week 10: Recommender systems

wbg231

January 2023

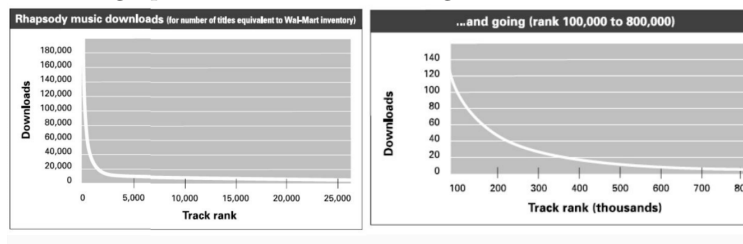
## 1 Introduction

### why the need for recommendation

- 1. physical objects occupy space
- 2. brick and mortar shops must satisfy physical constraints.
- 3. curators must look for some notion of utility
- 4. they chose to keep the items that satisfy the most customers
- digital items
  - on the web items take up no physical space so can have a lot of them
  - without algorithmic curation choice become overwhelming

### short head and long tail

- the concept is that popularity exponentially decays
- so there is a **short head** of item that are absurdly popular like Drake or what ever
- then there is a **long tail** ie many times more movies that have little to no popularity (but users may still enjoy)
- here is a graph of that for track ratings



- a retail store has limited space so focuses on the short head

## search and Recommender

- early Recommender systems relied on indexing and search
- users must describe what they want
- same query from different users will get the same results (so not personalized)
- textcolorredrecommendation = search + personalization that is a users history informs the ranking of results
- along with this is the idea that other users histories might be informative
- Recommender systems are the primary way users interact with large collections (like youtube Recommendations )

## personalization is critical

- traditional search methods model relevance of an item in response to a query
- personalized search methods do the same as above but model has a representation of each user
- to model relevance we need to record data known as feedback
- this works best in big data

## generic Recommendations

- given the diversity of user taste generic Recommendations do not work
- for our model to be good it must beat the popularity model (ie just recommending the top 100 most popular items )
- that model should always be the first thing you try to build a base line. that is rank each item by average utility defined as

$$utility(i) = \frac{\sum_u R[u, i]}{R[:, i]}$$

that is the sum of how all users rank that item over the number of users who ranked that item

- this produces the short head we were discussing earlier
- it is hard to improve on this because people have really different tastes that can be hard to learn
- there may not even be "logical corelation" like on a sonic level two songs may be similar but a user may have different feelings about the songs based on times they associate with those songs

## improving baseline with out personalization

- note that if only a few users rated a long, there average rating may not be representative of the whole population (1000 4 star ratings tells you more than 1 5 star rating)
- two ideas
- one just disregard items below a threshold level of interactions
- two use a prior
  1. so have your average utility set as

$$utility(i) = \frac{\sum_u R[u, i]}{|R[:, i]| + \beta}$$

where  $\beta > 0$  this is kinda like regularization in effect we are adding "more gave the item a low rating"

## global items and user bias

- another idea is to think of each interaction as a sum of global, item and user bias

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

so that is user u gave item i a certain rating that is equal to the sum of some global term  $\mu$  the bias of that item  $b[i]$  and bias of that user  $b[u]$  (where bias means offset)

- our global i teh average rating over all interactions

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

$\beta_g$  is a prior for the global

- the user bias vector has item i which is the average users difference from teh mean for item i

$$b[i] = \frac{\sum_u R[u, i] - \mu}{|R[:, i]| + \beta_i}$$

- the user item vector has users u which is the average users difference from teh mean for item u

$$b[u] = \frac{\sum_i R[u, i] - \mu}{|R[:, u]| + \beta_u}$$

- this model lets us make predictions for unseen interactions
- so we can predict for user u sorting items i by descending  $\mu + b[i] + b[u]$

- this is not more powerful than our base line but it is more interpretable. that is on average this model does not do much better than just predicting with the average popularity since we are in effect setting biases we dont know to zero

## implicit and explicit feedback

- explicit feedback is when users make a choice to rate a item (like a youtube video, sub to a channel, purchase an item)
- that gives us a really strong signal, but it takes user effort so it rare, and user can select into it so there is selection bias
- implicit feedback are things we can get by just observing the user (click rate, what they download , what songs they plan, do they finish a video)
- this is a weak or ambiguous signal does watching a video mean you like it? (not all the time )
- it is really common compared to explicit feedback though

## how is feedback obtained

- explicit feedback makes sense when the time and effort required to give it is low compared to the time it takes to consume and tem (like more people will like a video than write a movie review)
- implicit feedback is much more common approach

## Collaborative filtering

- the utility matrix is sparse ex: suppose zero here means disliked a video, one means liked a video blank means did not interact

		Items							
Users			1			1			
					0	0		1	
		1	1			1			
			1		0				

- our task is to predict the missing items

## neighborhood models

- two classes
- user based
  - the task is given user  $u$  find the most similar set of users  $\{u\}$
  - we are looking for similar rows in the utility matrix
  - similar rows of utility matrix need to be found
  - we predict items  $v$  that have high feedback and have been consumed by similar users and not yet consumed by user  $u$
- item based
  - we want to find the set of items  $\{v\}$  similar to those consumed by user  $u$
  - we are looking for similar columns in the utility matrix
  - we will predict those items which have not yet been consumed by user  $u$
- these two approaches are distinct but conceptually similar
- there are two questions to think about
  1. how do you define similarity between users and items
  2. how do you aggregate feedback over a neighborhood
- this can be tough to scale Depending on the type of feedback. this works really well for Jaccard similarity with min has and LSH
- this will not work for spatial data structures that can not deal with missing features

## Latent factor model

- this is a flexible framework for modeling feedback
- objective can be tuned to match some feedback mechanism (ratings, play count, purchase numbers)
- secondary objectives can be added (item bias, regularization, etc)
- Usually this model is easy to parallelize and scale up which is nice for this setting, can use Alternating least squares, and users are conditionally Independent given items and vice versa







- at the end you learn a low rank dense representation of the utility matrix. this Integrates will with spatial data structures (like knn), rank parameter (ie the rank of the matrix) that lets us control between complexity and expressivity
- the lf model is a bi linear model of user interactions that is given we are a interactions matrix  $R \in \mathbb{R}^{n \times d}$  we learn matrices  $U \in \mathbb{R}^{N \times 1}$  and  $V \in \mathbb{R}^{D \times 1} : R = UV^t$
- think of u and v as embeddings or representations of the users and items in a common vector space
- our objective function is called the the sum of least squared that is

$$J(U, V) = \sum_{(i,j) \in \Omega} (R_{i,j} - \langle u_i, V_j \rangle)^2$$

that is the sum of l2 distance between R and the inner product of U,V in what ever space they are in and

$$U, V = \min_{U, V} J(U, V)$$

- here is an example of Latent factors and how we can use them to reconstruct our ratings matrix

Reconstructing ratings by matrix multiplication				FC	H	J	S	T	Z
					5	1	2	1	5
					1	1	3	5	2
					2	5	3	1	1
FP				Rat	H	J	S	T	Z
A	0	1	0	A	1	1	3	5	2
B	0.75	0.5	0	B	4.25	1.25	3	3.25	4.75
C	0	0	1	C	2	5	3	1	1
D	0.75	0	0.5	D	4.75	3.25	3	1.25	4.25
E	0	0.75	0.75	E	2.25	4.5	4.5	4.5	2.25
F	0.25	0.25	0.25	F	2	1.75	2	1.75	2

- this is also nice for the purposes of compression as we can store our data using a lot less information

## Alternating least squares

- we solve our objective by doing the following
- Repeat
  1. fix user factors  $U$ , and optimize  $\min_V J(U, V)$
  2. fix item factors  $V$  and optimize user factors  $\min_U J(U, V)$
- when factors are fixed each item vectors  $V$  can be solved Independently from the item factors (allowing for parallelism)
- this problem is equivalent to alternative least squared regression

## modeling implicit feedback

- implicit feedback which is often count data (like views) is informative but hard predict
- instead we can try to predict binary interactions but use counts to weigh terms thus our optimization problem becomes

$$J(U, V) = \sum_{(i,j) \in \Omega} (1 + \alpha(R_{i,j}) < \mathbb{I}(R_{i,j} \geq 1) - u_i, V_j >)^2$$

where  $\alpha$  is some weighting term

- also could save space by dropping users or interactions with low counts
- could also compress values by setting  $R_{i,j} = \log(1 + R_{i,j}), \forall (i, j) \in \Omega \times \Omega$

## Handling new items

- the Latent factor model gives items with no interactions no representative, this means that a new item (which by nature has no interactions) will never be recommender with out adjustments
- this is known as a [cold start issue](#)
- we can solve the cold start problem by promoting new items, manual curation or complementing a Latent factor model with a content based model

## content based model

- suppose we have observed features  $x_j$  for each item  $j$  (these could be things like news source, song writer, genre) they are logical features of the item that we can explicitly specify instead of latent factors we learn
- each user  $i$  gets there own interactions model  $u_i$  and we model

$$R_{i,j} \approx \langle u_i, x_j \rangle$$

- like the lf model but the items factors are made explicit (that is we are only learning a representation of user factors)
- this can be limiting or over constraining
- an issue with this is as we noted above content features (like audio features in songs ) are not always predictive of preferences

## content cold start

- we train a LF model and learn  $U, V$  such that item  $j$  has factor  $v_j$
- we then regress item factors from features ie

$$V_i \approx f(x_i)$$

that is we use our explicit item features to learn how to predict item factors

- thus we a new item can map into our feature space using the learned mappings  $f(x_k)$  (what ever our regression model is)

## user warm start

- what happens when a new user enters the system?
- a lot of recommendation system ask for demographic data and examples of things you like. allowing new users to be quickly placed in the Collaborative filtering model before they start using the platform
- call this a warm start

## workflow in machine learning

1. obtain train and testing data
2. fit model to training data (optimize some objective function)
3. evaluate the model using your objective on the testing data.



## workflow in recommendation systems

- it is easy to think of observations  $(u,v)$  are Independent but that are not
- keep in mind what we are predicting and what we value
  - we care about satisfying a user
  - this should influence how we evaluate the model
- most interfaces provide many Recommendations at once
- this means we need to evaluate a collection of Recommendations per user
- then we can find the average across users to estimate system performance

## partitioning data

- recommendation models need some information on users to predict for them
- so when splitting data **partition each user's observations into train and validation set separately** this means each user will be in both sets but not all of there observations will be in both (this is a key detail )

## Evaluating recommendation systems

- our objective function is just a proxy for our main goal not the end goal it's self
- early recommender systems focused really hard on unitizing mean squared error
- but that is not everything, think about how Recommendations are delivered are they as a ranked list like in amazon netflix or youtube or are they one at a time like in pandora or spotify shuffle
- evaluations should reflect user behavior and how recomendations will be shown

## bipartite ranking evaluations

- the idea is to rank items by estimated relevance to users (or a query)
- using held out interactions we can determine which predictions were relevant (or positive) or irrelevant(negative)
- since we are predicting any irrelevant recommendation is a false positive
- we want all relevant documents to come before the irrelevant observations
- we assume all relevant documents are equally relevant

## ranking evaluations

- we predict an ordered list of items the ground truth is a held out list of interactions
- AUC area under the ROC
  - this tells us how often does a positive interaction come before a negative interaction
  - so if we had a list of predictions and the true labels of interactions on those predicted items were  $- + - + + - - \rightarrow \frac{3+2+2}{12} = \frac{7}{12}$
  - the ROC curve shows  $\frac{TP}{FP}$  at different classification thresholds
  - the AUC is the area under that ROC it is an aggregate measure of performance across all classification thresholds (it is in effect the likelihood that the model scores a negative example more highly than a positive example )
- average Precision (AP)
  - for each positive interactions what fraction of higher ranked items were also positive
  - so given we output a recommendation list with true labels  $- + - + + - -$  we would have  $AP \frac{1}{3}(\frac{1}{2} + \frac{1}{2} + \frac{3}{5})$  it is on average what proportion of the list at each step were positive predictions
- Reciprocal rank
  - the inverse position of the first positive interaction
  - so for the same list that is  $\frac{1}{2}$
  - this is good if we are recommending items one at a time

## in real life

- these models have a feedback loop in reality so they are hard to evaluate on observational data
- most real evaluation is done with A/B testing
- and more systems are shifting to RL and Causal approaches

## evaluation summary

- evaluation is hard to do well
- ranking is not everything
- think about what the model is used for and how it makes these recommendations

## Socio-cultural impact of recommender systems

- **filter bubble** diversity in what user's see falls over time as recommender systems rely on similarity
- this can either cause users to become bored and leave the site
- or become isolated and polarized in online echo chambers
- what information can you use for a recommendation system (gender, age, race, zipcode ,income ) all may cause bias issues
- the ethics are complex
- have to think about if the model is biased, is the user population biased, how does the model treat users outside of the typical user population
- who benefits from this recommender system, is personalization needed
- recommender systems are trained on past behavior so they can depend on the user population