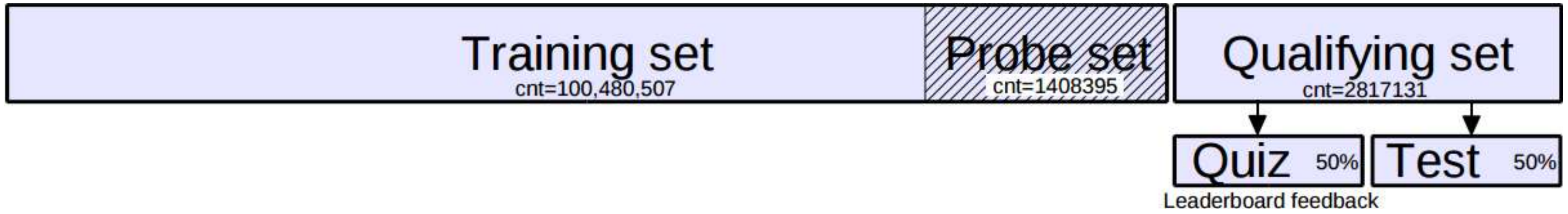
Recommenders

Collaborative Filtering Recommenders

“Everything is a Recommendation” (Netflix)

- recommenders have been popularized by Netflix; they are now extremely common and are utilized in a variety of areas
- recommenders seek to predict the ‘rating’ or ‘preference’ that a user would give to an item
- e.g., over 75% of what people watch on Netflix comes from their recommendation engine [“Netflix’s New ‘My List’ Feature Knows You Better Than You Know Yourself (Because Algorithms)”]

Netflix's $1M Prize



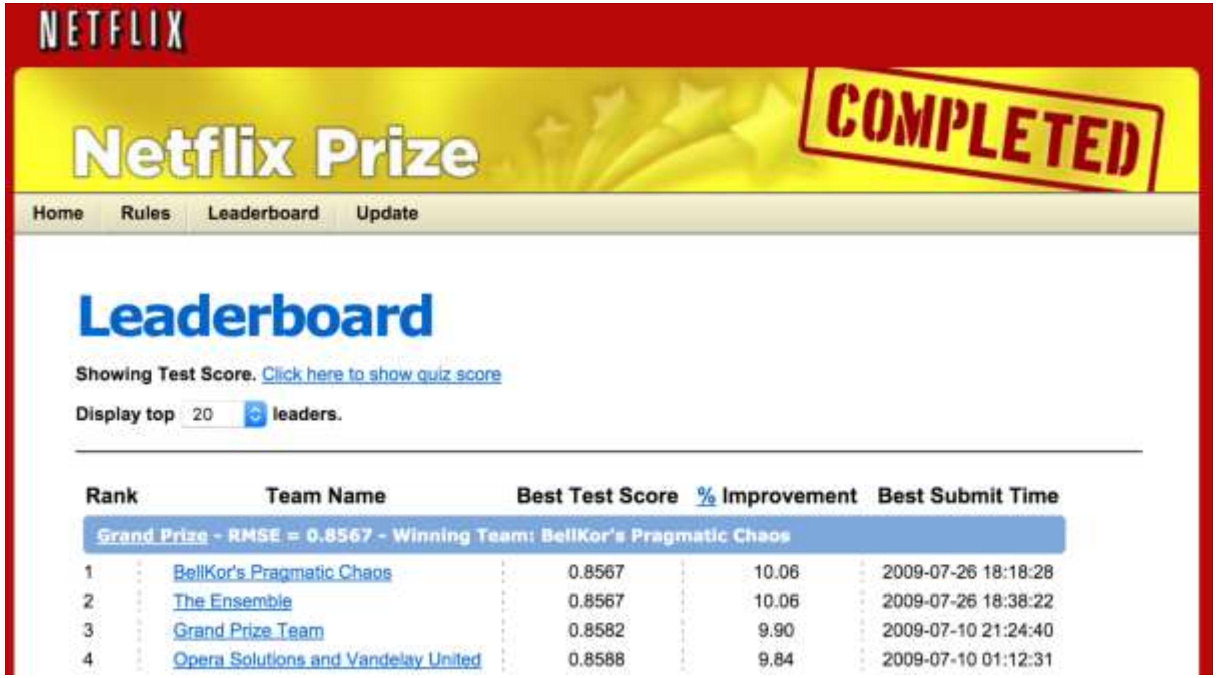
- training set of 100M ratings
  - that m = 480k users gave to n = 18k movies between 12/31/1999 and 12/31/2005
  - made of quadruplets of the form:
    - `user` *<integer>*, `movie` *<integer>*, `date of grade`, `grade` *<1-5>*
- hold-out set (4.2M) randomly split 3 ways into probe, quiz, and test subsets
  - probe set attached to the training set
  - competitors were required to predict ratings for the quiz and test sets
    - made of quadruplets of the form
      - `user` *<integer>*, `movie` *<integer>*, `date of grade`
    - prizemaster returns the root mean squared error (RMSE) achieved on the quiz set, then posted on the public leaderboard
    - prize winner is the one that scores best on the test set (Netflix never disclosed these scores)

Netflix’s $1M Prize (cont.)

- training set
  - average user rated over 200 movies
  - average movie was rated by over 5,000 users
- hold-out set (4.2M) randomly split 3 ways into probe, quiz, and test subsets
  - last nine movies rated by each user
  - compared with the training data, the hold-out set contains many more ratings by users that do not rate much and are therefore harder to predict
    - this represents real requirements for a collaborative filtering (CF) system, which needs to predict new ratings from older ones, and to equally address all users, not just the heavy raters

Netflix's $1M Prize (cont.)

- the winning team using gradient boosted decision trees to combine over 500 models



Netflix uses a collaborative filtering recommender system

- collaborative filtering
  - only consider past users behavior (yours and others)
    - “collaborative filtering” refers to the use of ratings from multiple users in a collaborative way to predict missing ratings
  - more examples:
    - Amazon recommendations
    - Google and Facebook ads
    - news feeds, trending news
    - …

There are other type of recommenders…

- popularity
  - make the **same** recommendation to **every** user based on the popularity of an item
    - e.g., Twitter Moments
- content-based (a.k.a., content filtering)
  - recommendations are made based on the properties/characteristics of an item and the past user behavior
    - e.g., Pandora Radio
      - users build up “stations” based on their musical interests
      - a user sets in each station one or more songs or artists that he or she likes
      - based on these preferences, Pandora plays similar songs that the user might also like
    - users can refine their station by giving a “thumbs up” (want to hear more similar music) or a “thumbs down” (don’t' want to hear this song again and is not interested in similar types of music) to songs
- matrix factorization methods (this afternoon)
  - find latent features (factors)

Data as a Utility Matrix

- typically, data is a utility/rating matrix which capture user well-being/preferences
- unrated items are coded as missing or 0
- matrix is sparse (as most items are unrated) *(e.g., 1.2% for the Netflix Prize matrix)*

		items				
		#1	#2	#3	#4	
users	A					
	B					
	C					…
	D					
	E					
		…				

Data can be Explicit

- user-provided ratings
  - e.g., 1 to 5 stars *(Netflix Prize, Amazon)*
  - e.g., like/non-like *(Facebook)*

items

users	#1	#2	#3	#4	…
A	1	4	2		
B		3		4	
C	1	5		5	
D	2		3		
E		3		3	
…					

or Implicit

- more common
  - infer user-item relationship from user behavior and actions
  - e.g., buy/non-buy *(Amazon)*
  - e.g., view/non-view *(Amazon)*

	items				
users	#1	#2	#3	#4	
A	1	1	1		
B		1		1	
C	1	1		1	…
D	1		1		
E		1		1	
		…			

User-User Similarities

- to predict Alice's rating of Titanic, we can
  - find a set of users "similar" to Alice who rated Titanic
  - then take the mean of their ratings of Titanic
- this is user-user similarity:
  - calculate the "similarity" of all user pairs (row vectors)
  - make predictions based on similarity between users

items

users	#1	#2	#3	**Titanic**	
Alice	**1**	**4**	**2**		**...**
B		3		4	
Caleb	**1**	**5**		**5**	**...**
D	2		3		
E		3		3	
		...			

Item-Item Similarities

- to predict Alice’s rating of Titanic, we can also
  - find a set of items similar to Titanic that Alice has also rated
  - take the (weighted) mean of Alice’s ratings on them
- this is item-item similarity:
  - calculate the “similarity” of all pair of items (columns vectors)
  - make predictions based on similarity between items

Enchanted

users

	#1	Enchanted	#3	Titanic	
Alice	1	**4**	2		
B		**3**		**4**	
C	1	**5**		**5**	…
D	2		3		
E		**3**		**3**	
		…		**…**	

User-User or Item-Item?

- let
  - m = |users| (e.g., 480k as in the Netflix challenge) and
  - n = |items| (e.g., 18k)
- user-user
  - m^2 user pairs and assuming a user-user similarity computation of O(n), we have a user-user computational complexity of $O(m^2 n)$
  - that's a whopping 4.1×10^{15} computations
- item-item
  - n^2 item pairs and assuming a item-item similarity computation of O(n), we have a user-user computational complexity of $O(m\,n^2)$
  - that's still a lot of computations (1.6×10^{14}) but it's also 26x less than in the user-user setting

User-User or Item-Item? (cont.)

- item-item
  - most popular as many businesses have a limited numbers of items for many more users
  - other pluses
    - item pairs have more stable similarities than user pairs (items have usually more ratings than users)

      (Netflix Prize: average movie was rated by over 5,000 users; average user rated over 200 movies)
    - item-item similarities are stable over time while users change preferences overtime

      (6 years span in Netflix Prize)

Similarity Metrix using Euclidean Distance

$$distance(u, v) = \|u - v\| = \sqrt{\sum_i (u_i - v_i)^2}$$

- a distance of 0 means the items are identical; the distance is also unbounded

$$similarity(u, v) = \frac{1}{1 + distance(u, v)}$$

[similarity ranges between 0 (totally dissimilar) to 1 (totally similar)]

- the Euclidean distance is the most intuitive of the distance metrics yet you'll probably never use it in this setting

Similarity Metrix using Pearson Correlation

- the Pearson correlation measures how much two vectors each deviate from their mean together:

$$pearson(u,v) = \frac{\sigma_{u,v}^2}{\sigma_u \sigma_v} = \frac{\sum_i (u_i - \bar{u})(v_i - \bar{v})}{\sqrt{\sum_i (u_i - \bar{u})^2}\sqrt{\sum_i (v_i - \bar{v})^2}}$$

$$similarity(u,v) = \frac{1}{2} + \frac{1}{2} pearson(u,v)$$

- this similarity isn’t sensitive to a user who consistently rates low or high
  - e.g., user #1 rates 3 items: 5, 5, and 3 and user #2 rates them as 3, 3, and 1: the similarity of these 2 users will be 1 (totally similar)

Similarity Metrix using Cosine Distance

- the Cosine distance measures the angle between two vectors

$$cos(\theta_{u,v}) = \frac{u \cdot v}{\|u\|\|v\|} = \frac{\sum_i u_i v_i}{\sqrt{\sum_i {u_i}^2}\sqrt{\sum_i {v_i}^2}}$$

$$similarity(u,v) = \frac{1}{2} + \frac{1}{2} cos(\theta_{u,v})$$

- equivalent to the Pearson Correlation when vectors are mean-centered

Similarity Metrix using Jaccard Similarity

$$similarity(u, v) = \frac{|U_u \cap U_v|}{|U_u \cup U_v|}$$

[U_k denotes the set of users who rated item k]

- the Jaccard similarity is a measure of the similarity of two sets
- here, we would like to measure if two items have been rated by the same users
- use this metric when you don't have ratings, just Boolean data

Similarity Matrix

pick a similarity metric then create the (e.g., item-item) similarity matrix:

items	#1	#2	#3	
#1	1	.3		
#2	.3	1	.7	…
#3		.7	1	
		…		

Predicting Ratings

- say user *u* hasn't rated item *i*
- we want to predict the rating that this user would give this item

$$p_{u,i} = \frac{\sum_{j \in I_u} similarity(i,j) \cdot r_{u,j}}{\sum_{j \in I_u} similarity(i,j)}$$

$$p_{u,i} = \text{user } u\text{'s predicted rating of item } i$$
$$I_u = \text{set of items rated by user } u$$
$$r_{u,j} = \text{user } u\text{'s true rating of item } j$$

Predicting Ratings (cont.)

- to improve performance, we can restrain our calculations to items most similar to *i*

$$p_{u,i} = \frac{\sum_{j \in I_u \cap \boldsymbol{N_i}} similarity(i,j) \cdot r_{u,j}}{\sum_{j \in I_u \cap \boldsymbol{N_i}} similarity(i,j)}$$

$$p_{u,i} = \text{user } u'\text{s predicted rating of item } i$$
$$I_u = \text{set of items rated by user } u$$
$$N_i = n \text{ items most similar to item } i$$
$$r_{u,j} = \text{user } u'\text{s true rating of item } j$$

Deploying the Recommender

- off-line (e.g., during the night)
  - compute similarity between all item pairs, $similarity(i, j)$
  - compute the neighborhood of each item, N_i
- online/at request time
  - predict scores for candidate items ($rating(u, i)$) and make recommendations

Validating a Recommender

- recommenders are inherently hard to validate
    - in practice, we would launch the recommender using A/B testing and see if it leads to more conversions
- beyond that, there isn't standard of how to evaluate a recommender
    - we can do a k-fold cross validation like we do with classification and regression and there are few different metrics we can use

Validating a Recommender

- Root Mean Squared Error (RMSE)
  - predict the rating for all user/movie pairs in the test set and calculate the RMSE between your predicted values and the true values

$$RMSE = \sqrt{\sum_{u;\, i \in I_u} \left(r_{u,i} - p_{u,i}\right)^2}$$

$$r_{u,i} = \text{user } u'\text{s true rating of item } i$$
$$p_{u,i} = \text{user } u'\text{s predicted rating of item } i$$

  - however this metric considers how far off you are with **<u>all</u>** of your ratings
  - in practice, we're trying to recommend the thing the user would want to see next; it's not about making rating predictions
  - a couple of other metrics that just consider this: precision and recall

Validating a Recommender (cont.)

- precision at n *(how many selected items are relevant)*

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{\{retrieved\ documents\}}$$

  - proportion of the top-n documents that are relevant (relevant = watched)

- recall at n *(how many relevant items are selected)*

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{\{relevant\ documents\}}$$

  - proportion of the relevant items are in the top n

Recommenders

Matrix Factorization for Recommenders

SVD vs. NMF

SVD:

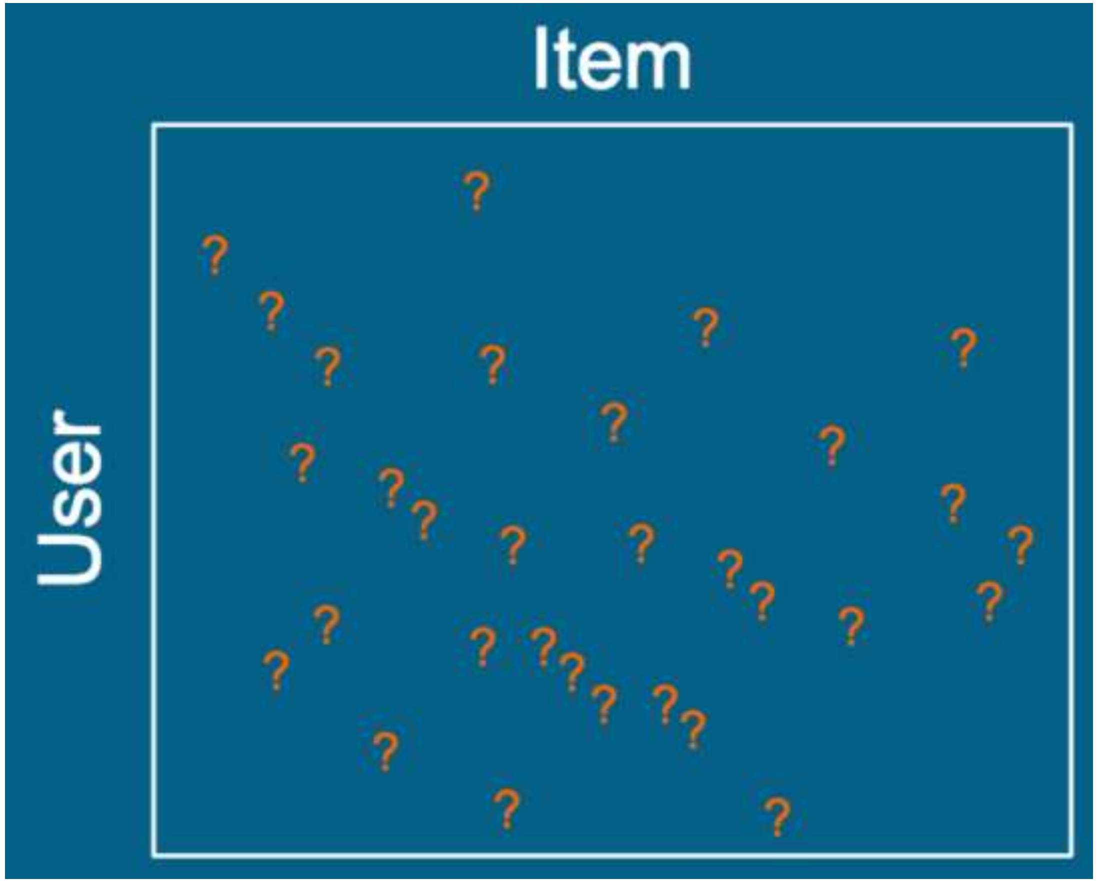
- $R = U\Sigma V^t$
- U and V are orthonormal matrices
- S is a diagonal matrix of decreasing positive “singular” values
- a SVD is not unique: the singular values are unique, but the singular vector matrices are not

NMF:

- $V \approx WH$
- all values of V must be non-negative
- W and H will not (likely) be orthonormal
- NMF is an approximate factorization and non-unique solutions
    - (local optima with non-convex RSS optimization)
- has a tunable parameter k

An explicit-rating utility matrix is usually VERY sparse…

- we’ve previously used SVD to find latent features
- is SVD be good for this sparse utility matrix?
- no!
  - you are forced to fill in missing values
  - the solution you’ll get would have to fit these fill-values (which is silly)



NMF on the other end is a great option for many recommender systems

- reason #1: no need to impute missing values

cost function:

$$\underset{W,H}{\operatorname{argmin}} \sum_{i,j \in \kappa} \left(v_{ij} - w_{i:} h_{:j}\right)^2 + \lambda \left(\|w_{i:}\|^2 + \|h_{:j}\|^2 \right)$$

not over all indices but just the set of indices of known ratings

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

Reason #2: NMF can accommodate biases terms to communicate prior knowledge

- in practice, much of the observed variation in rating values is due to item bias and user bias:
  - some items (e.g., movies) tend to be rated high, some low
  - some users tend to rate high, some low

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

overall bias of the rating by user *i* for item *j*

item j's average deviation from the overall average

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

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

user i's average deviation from the overall average

Biases (cont.)

- ratings are now estimated as:

prediction of how user i will interact with item j

prediction of user i rating item j

$$v_{ij} = \mu + b_i^* + b_j' + w_{i:}h_{:j}$$

average rating

user i's tendency to deviate from the average

item j's tendency to deviate from the average

- updated cost function:

$$\underset{W,H}{\operatorname{argmin}} \sum_{i,j \in \kappa} \left(v_{ij} - \mu - b_i^* - b_j' - w_{i:}h_{:j}\right)^2 + \lambda \left((b_i^*)^2 + (b_i^*)^2 + \|w_{i:}\|^2 + \left\|h_{:j}\right\|^2 \right)$$