

Recommendation Systems

Arsen Yeghiazaryan

- Content based
- Collaborative filtering
 - Memory based
 - Item based
 - User based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

- Content based
- Collaborative filtering
 - Memory based
 - Item based
 - User based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

Content based RS highlights

- Product and user profiling is the key (e.g. movie profile - genre, actors, box office popularity, etc. User profile - demographic information, answers to questionnaire, etc.)
- Associate users with matching products based on profiles
- Alleviates the 'cold start' problem
- User independence
- Full interpretability of recommendations

Content based RS - Limitations

- Requires gathering external information
- Not all content is well represented by keywords, e.g. images
- Not sufficiently detailed profiles
- Over-specialization - 'serendipity' problem
- Users with thousands of purchases is a problem

- Content based
- Collaborative filtering
 - Memory based
 - Item based
 - User based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

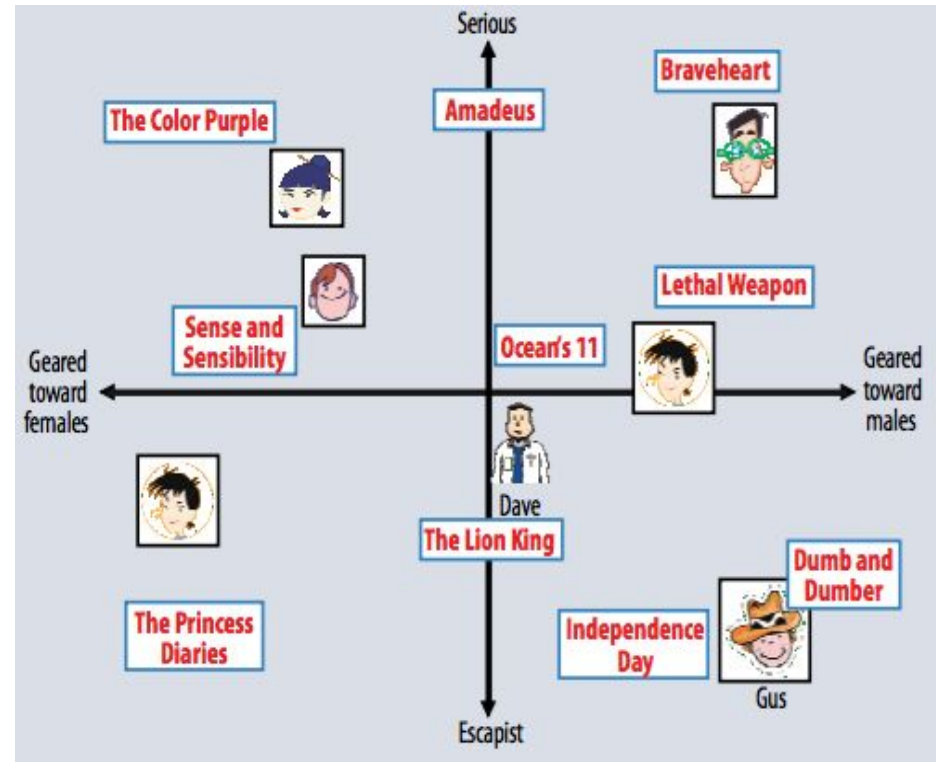
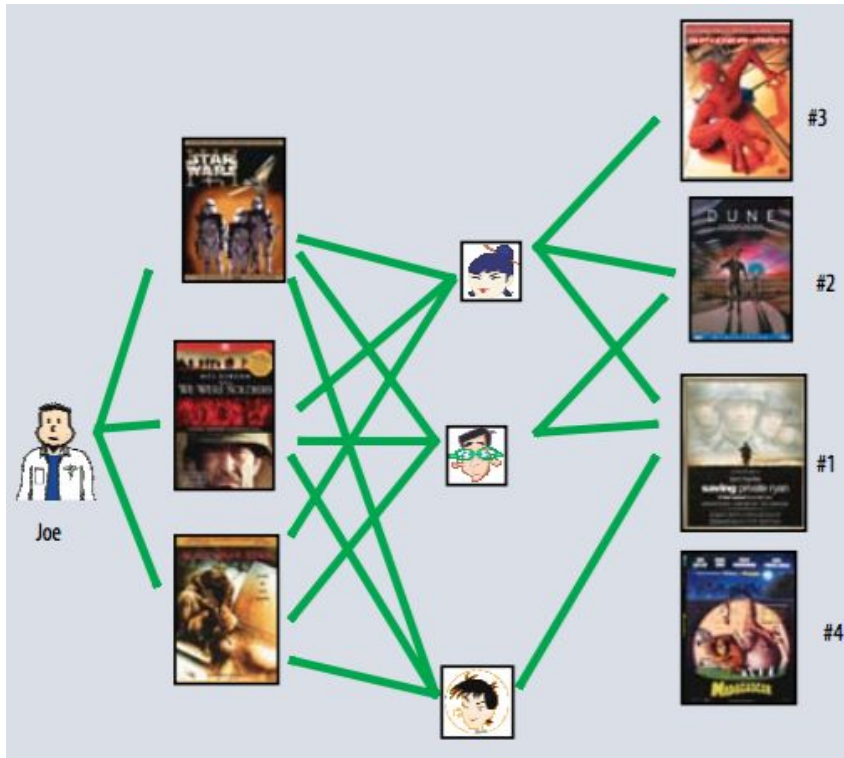
Collaborative filtering

- The most prominent approach to generate recommendations
 - used by large, commercial e-commerce sites
 - well-understood, various algorithms and variations exist
 - applicable in many domains (book, movies, DVDs, ..)
- Approach
 - use the "wisdom of the crowd" to recommend items
- Basic assumption and idea
 - Users give ratings to catalog items (implicitly or explicitly)
 - Customers who had similar tastes in the past, will have similar tastes in the future

Memory based

vs

Model based



- Content based
- Collaborative filtering
 - Memory based
 - User based
 - Item based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

User based nearest neighbor collaborative filtering

The basic technique:

- Given an "active user" (Alice) and an item / not yet seen by Alice
- The goal is to estimate Alice's rating for this item, e.g., by
 - find a set of users who liked the same items as Alice in the past and who have rated item /
 - use, e.g. the average of their ratings to predict, if Alice will like item /
 - do this for all items Alice has not seen and recommend the best-rated

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

User based nearest neighbor collaborative filtering

Questions:

- How do we measure similarity?
- How many neighbors should we consider?
- How do we generate a prediction from the neighbors' ratings?

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

User similarity

Metrics candidates:

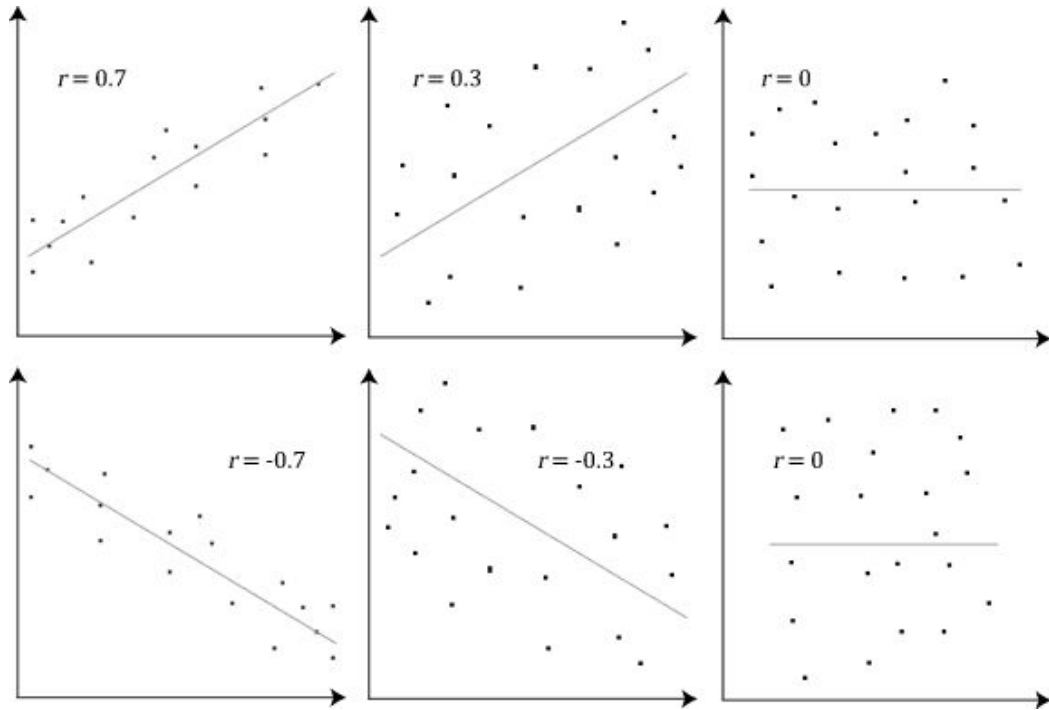
- Pearson correlation
- Cosine similarity
- Other

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1



sim = 0,85
sim = 0,00
sim = 0,70
sim = -0,79

Pearson correlation coefficient



$$r = \frac{\sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^n (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^n (Y_i - \bar{Y})^2}}$$

Making predictions: Naive

r_{ai} – rating of user a , item i

neighbors $N = k$ most similar users

prediction = average of neighbors' ratings

$$\text{pred}(a, i) = \frac{\sum_{b \in N} r_{bi}}{|N|}$$

Making predictions: Naive

r_{ai} – rating of user a , item i

neighbors $N = k$ most similar users

prediction = average of neighbors' ratings

$$\text{pred}(a, i) = \frac{\sum_{b \in N} r_{bi}}{|N|}$$

Problems with naive approach:

- a) All neighbors in N contribute equally.
- b) User bias - some user have higher ratings on average.

Making predictions: Naive

r_{ai} – rating of user a , item i

neighbors $N = k$ most similar users

prediction = average of neighbors' ratings

$$\text{pred}(a, i) = \frac{\sum_{b \in N} r_{bi}}{|N|}$$

Problems with naive approach:

- a) All neighbors in N contribute equally. Solution: weighted sum, weight $\text{sim}(a,b)$
- b) User bias - some user have higher ratings on average.

Making predictions: Naive

r_{ai} – rating of user a , item i

neighbors $N = k$ most similar users

prediction = average of neighbors' ratings

$$\text{pred}(a, i) = \frac{\sum_{b \in N} r_{bi}}{|N|}$$

Problems with naive approach:

- a) All neighbors in N contribute equally. Solution: weighted sum, weight $\text{sim}(a, b)$
- b) User bias - some user have higher ratings on average. Solution: use $(r_{bi} - \bar{r}_b)$ instead

Making predictions

Taking corrections a) and b) into account

$$pred(a, i) = \bar{r}_a + \frac{\sum_{b \in N} sim(a, b) \cdot (r_{bi} - \bar{r}_b)}{\sum_{b \in N} sim(a, b)}$$

r_{ai} – rating of user a , item i

\bar{r}_a, \bar{r}_b – user averages

Improving the metrics / Prediction function

- Not all neighbor ratings might be equally "valuable"
 - Agreement on commonly liked items is not so informative as agreement on controversial items
 - Possible solution: Give more weight to items that have a higher variance
- Value of number of co-rated items
 - Use "significance weighting", by e.g., linearly reducing the weight when the number of co-rated items is low
- Case amplification
 - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.
- Neighborhood selection
 - Use similarity threshold or fixed number of neighbors

- Content based
- Collaborative filtering
 - Memory based
 - User based
 - Item based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

Item based collaborative filtering

- Compute similarity between items
- Use this similarity to predict ratings
- More computationally efficient (often, number of items \ll number of users)
- Item similarities are supposed to be more stable than user similarities
- Practical advantage - more intuitive

Item based Nearest Neighbor CF: Problem statement

	Item1	Item2	Item3	Item4	Item5
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Similarity, Predictions

- (Adjusted) cosine similarity works slightly better for items
- Prediction

$$pred(u, p) = \frac{\sum_{i \in R} sim(i, p) r_{ui}}{\sum_{i \in R} sim(i, p)}$$

Similarity, Predictions

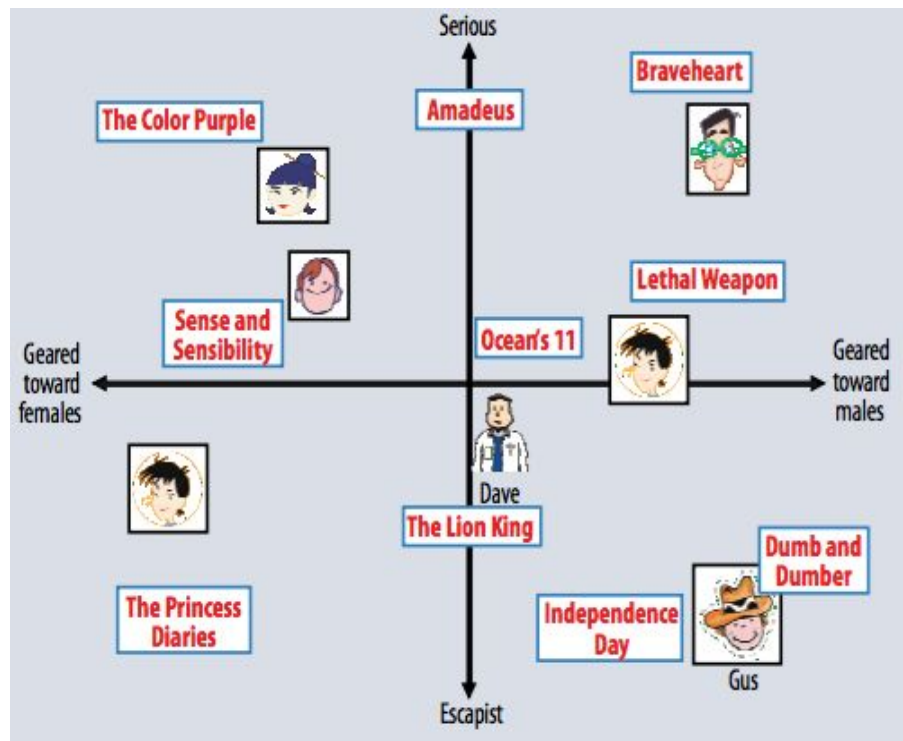
- Pearson correlation? Cosine similarity? Other?
- No fundamental reason for choice of one metric
- Based on practical experience and current application

Notes of Memory based CF methods

- Hard to scale! Item-based filtering does not solve the scalability problem itself
- Possible to calculate all pairwise similarities in advance
- Rely only on the rating matrix - users not always willing to rate many items (how to consider implicit feedback, i.e. clicks, page views, time spent on page, etc.)
- Cold start problem (ask/force users to rate a set of items, use content based methods in the initial phase)

- Content based
- Collaborative filtering
 - Memory based
 - User based
 - Item based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

Model based Collaborative Filtering: Matrix Factorization

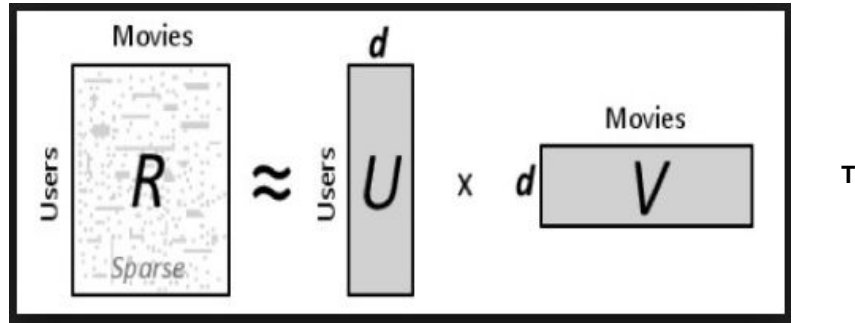


Matrix Factorization

R - ($N \times M$) sparse matrix of ratings

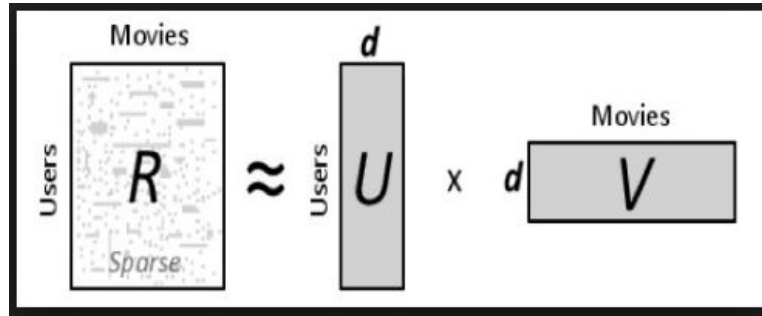
U - ($N \times d$) dense matrix of user latent factors

V - ($M \times d$) dense matrix of user latent factors



Matrix Factorization

- Latent factors of users and items
- Use these to predict ratings
- Related to singular value decomposition



T

Singular Value Decomposition

- singular value decomposition (SVD) – theorem in linear algebra
- in CF context the name “SVD” usually used for an approach only slightly related to SVD theorem
- related to “latent semantic analysis”

Singular Value Decomposition

$$X = USV^T$$

- U, V orthogonal matrices
- S diagonal matrix, diagonal entries \sim singular values

low-rank matrix approximation (use only top k singular values)

$$\begin{pmatrix} & X & \\ \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} & & \\ & m \times n & \end{pmatrix} = \begin{pmatrix} & U & \\ \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix} & & \\ & m \times r & \end{pmatrix} \begin{pmatrix} & S & \\ \begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix} & & \\ & r \times r & \end{pmatrix} \begin{pmatrix} & V^T & \\ \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix} & & \\ & r \times n & \end{pmatrix}$$

SVD - CF interpretation

$$X = USV^T$$

- X – matrix of ratings
- U – user-factors strengths
- V – item-factors strengths
- S – importance of factors

SVD vs Matrix Factorization

- matrix factorization techniques (SVD) work with full matrix
- ratings – sparse matrix
- solutions:
 - value imputation – expensive, imprecise
 - alternative algorithms (greedy, heuristic): gradient descent, alternating least squares

Statement of the Learning Problem

	Item			
	W	X	Y	Z
A		4.5	2.0	
B	4.0		3.5	
C		5.0		2.0
D		3.5	4.0	1.0

Rating Matrix

=

A	1.2	0.8
B	1.4	0.9
C	1.5	1.0
D	1.2	0.8

User
Matrix

X

W	X	Y	Z
1.5	1.2	1.0	0.8
1.7	0.6	1.1	0.4

Item
Matrix

Notation

- u – user, i – item
- r_{ui} – rating
- \hat{r}_{ui} – predicted rating
- b, b_u, b_i – bias
- q_i, p_u – latent factor vectors (length k)

Simple Baseline Predictors

Always use baseline methods!

- naive: $\hat{r}_{ui} = \mu$, μ is global mean

Simple Baseline Predictors

Always use baseline methods!

- naive: $\hat{r}_{ui} = \mu$, μ is global mean
- biases: $\hat{r}_{ui} = \mu + b_u + b_i$
 - b_u, b_i – biases, average deviations
 - some users/items – systematically larger/lower ratings

Latent Factors

(for a while assume centered data without bias)

$$\hat{r}_{ui} = q_i^T p_u$$

- vector multiplication
- user-item interaction via latent factors

illustration (3 factors):

- user (p_u): (0.5, 0.8, -0.3)
- item (q_i): (0.4, -0.1, -0.8)

Latent Factors

$$\hat{r}_{ui} = q_i^T p_u$$

- vector multiplication
- user-item interaction via latent factors

we need to find q_i , p_u from the data (cf content-based techniques)

note: finding q_i , p_u at the same time

Learning Latent Factors

- we want to minimize “squared errors” (related to RMSE, more details later)

$$\min_{q,p} \sum_{(u,i) \in T} (r_{ui} - q_i^T p_u)^2$$

- regularization to avoid overfitting (standard machine learning approach)

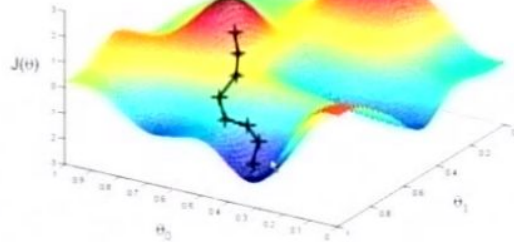
$$\min_{q,p} \sum_{(u,i) \in T} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

How to find the minimum?

Stochastic Gradient Descent

- standard technique in machine learning
- greedy, may find local minimum

Gradient Descent



Gradient Descent for MF

- prediction error $e_{ui} = r_{ui} - q_i^T p_u$
- update (in parallel):
 - $q_i := q_i + \gamma(e_{ui}p_u - \lambda q_i)$
 - $p_i := p_u + \gamma(e_{ui}q_i - \lambda p_u)$
- math behind equations – gradient = partial derivatives
- γ, λ – constants, set “pragmatically”
 - learning rate γ (0.005 for Netflix)
 - regularization λ (0.02 for Netflix)

Adding Bias

predictions:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

function to minimize:

$$\min_{q,p} \sum_{(u,i) \in T} (r_{ui} - \mu - b_u - b_i - q_i^T p_u)^2 + \lambda (||q_i||^2 + ||p_u||^2 + b_u^2 + b_i^2)$$

Improvements

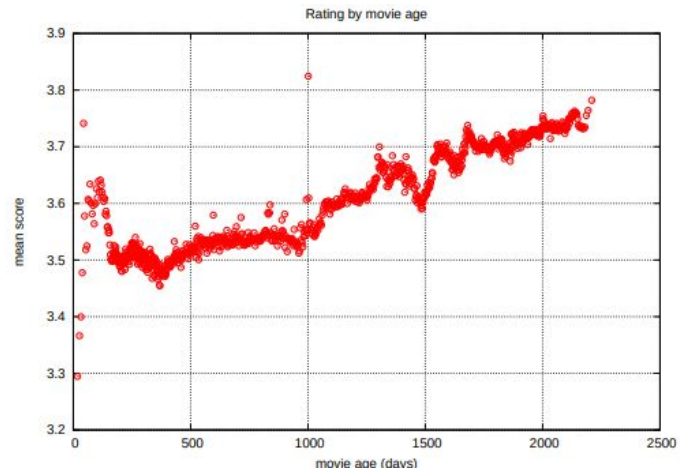
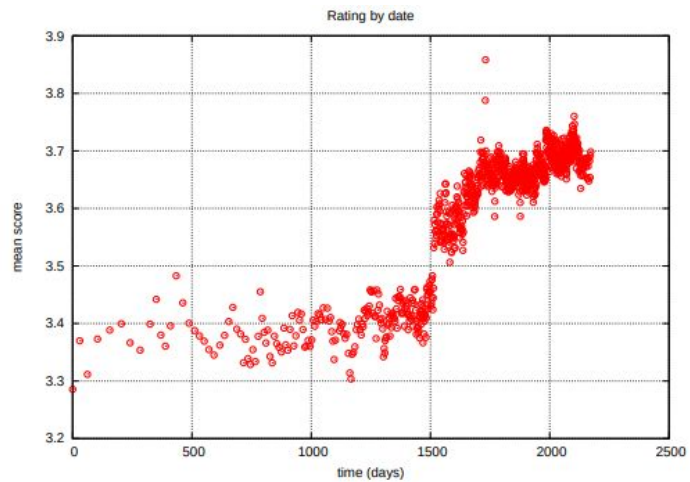
- additional data sources (implicit ratings)
- varying confidence level
- temporal dynamics

Incorporating Implicit feedback: SVD++

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$
$$b_{ui} = \mu + b_u + b_i$$

- $N(u)$ is the set of items for which user u has provided implicit feedback.
- Beats SVD (i.e. classical MF) on Netflix test set
- Possible to do other modifications (e.g. see Koren 2008 - Factorization meets the neighborhood- a multifaceted collaborative filtering model)

Temporal Dynamics



Temporal Dynamics

- SVD++

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$

- Modification

$$b_{ui}(t) = \mu + b_u(t) + b_i(t)$$

- Many different time dependence functions can be considered
- See Y.Koren - Collaborative Filtering with Temporal Dynamics
- Significantly outperforms both SVD and SVD++ on Netflix test data (RMSE)

Model	$f=10$	$f=20$	$f=50$	$f=100$	$f=200$
SVD	.9140	.9074	.9046	.9025	.9009
SVD++	.9131	.9032	.8952	.8924	.8911
timeSVD++	.8971	.8891	.8824	.8805	.8799

**WHAT IF WE ONLY
HAVE IMPLICIT FEEDBACK?**



Implicit vs Explicit feedback

- More abundant than explicit feedback (reluctance of users to give explicit feedback, limitations of the feedback collecting system)
- Types of implicit feedback - purchase history, browsing history, search patterns, mouse movements, etc.

Implicit feedback characteristics

- No negative feedback in case of implicit feedback!
- Implicit feedback is inherently noisy!
- Numerical value of E.F. indicates preference, numerical value of I.F. indicates confidence
- Evaluation requires appropriate metrics (RMSE isn't much suitable anymore)

Implicit Feedback: Problem Statement and Notations

- r_{ui} - observations for user actions, e.g. number of times u purchased item i , time u spent on website i , how many times u fully watched show i
- In the last example, $r_{ui}=0.7$ would indicate, that u watch 70% of the show
- In case of explicit feedback, r_{ui} values indicate preference. For implicit feedback we define preference p_{ui} as:

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

Implicit Feedback: Problem Statement and Notations

- Preference definition

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

- I.e., if u consumed i then preference is 1, if u never consumed i we believe no preference
- However, our beliefs are of greatly varying confidence levels! (e.g. 0 values are associated with low confidence level)
- As r_{ui} grows, we have stronger indication that the user indeed likes the item
- Let's denote the confidence level by c_{ui} . A plausible choice for r_{ui} could be:

$$c_{ui} = 1 + \alpha r_{ui}$$

- $\alpha=40$ was found to produce good results (Hu et al., Collaborative Filtering for Implicit Feedback Datasets)

Implicit Feedback: Problem Statement

Main differences from explicit feedback optimization problem

- Need to account for the varying confidence levels
- Optimization should account for all possible (user, item) pairs, not only those corresponding to observed data!

$$\min_{x_*, y_*} \sum_{u, i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

- Optimization contains (mxn) terms, which can reach a few billion. Makes SGD problematic!
- Optimize with Alternating Least Squares (ALS)


A small step back: Linear Regression

$$\hat{y}_i = f(x_i) = \hat{\beta}_0 + \sum_{j=1}^p x_{ij} \hat{\beta}_j$$

$$\hat{Y} = X\hat{\beta}$$


matrix representation

Minimize a loss function to find the β giving the “best fit”

$$\mathcal{L}(\beta) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2 + \lambda \sum_j \beta_j^2$$


penalty weight; tuning parameter

$\|\beta\|_2^2$

$$\hat{\beta} = X(X^T X + \lambda \mathbf{I}_p)^{-1} X^T Y$$

Alternating Least Squares

- Highly parallelizable substitute for SGD.
- Fix all item factors, solve for user factors. For each of m users:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

- C^u is a diagonal $(n \times n)$ matrix with $C^u_{ii} = c_{ui}$
- Next, fix all user factors, solve for item factors. For each of n items:

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i p(i)$$

- C^i is a diagonal $(m \times m)$ matrix with $C^i_{uu} = c_{ui}$
- Repeat until convergence

Alternating Least Squares: Complexity analysis

- One step of factor update for each of m users:

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u p(u)$$

- Computational bottleneck - $Y^T C^u Y$, naive computation requires $O(f^2 n)$ time. Should be calculated for each of m users
- Trick : $Y^T C^u Y = Y^T Y + Y^T (C^u - 1) Y$, where $(C^u - 1)$ contains only $n_u \ll n$ non-zero elements (n_u is the number of elements for which $r_{ui} > 0$).
- $Y^T Y$ precomputed once before each iteration for all users
- Overall time complexity for each user - $O(f^2 n_u + f^3)$, where $O(f^3)$ is the complexity of the matrix inversion operation (note: more efficient algorithms for inversion exist)
- Overall time complexity of one iteration over all users - $O(f^2 N + f^3 m)$, N is the number of all non-zero observations
- Linear in N !!

ALS vs SGD

SGD

- Easy to implement
- Fast running time (generally much faster than ALS)

ALS

- Allows massive parallelization (computes each user/item factors independently from others)
- More suitable for implicit feedback setups, where the preference matrix is not sparse
- Already implemented in Spark - *spark.mllib.recommendation.ALS*

Note: eALS, which is effectively an element-wise version of ALS, reduces ALS complexity even more (X.He et al 2017 - Fast Matrix Factorization for Online Recommendation with Implicit Feedback)

Evaluation metrics for Implicit feedback: Mean Percentage Ranking (MPR)

- For each user we predict a ranked list of items in I sorted by preference
- Let $rank_{ui}$ denote the percentile ranking of item i for user u
- $rank_{ui}=0\%$ signifies that i is predicted as the highest recommended item for u
- $rank_{ui}=100\%$ signifies that i is predicted as the lowest recommended item for u
- The percentile ranking is evenly distributed among the remaining items in the list by steps $(100\%/|I|)$
- MPR is defined as:

$$MPR = \frac{\sum_{ui} r_{ui}^t rank_{ui}}{\sum_{ui} r_{ui}^t}$$

- Lower values of MPR indicate better recommendation quality
- Other metric options: $HR@k$, $NDCG@k$

Small modification to MF: Logistic MF

- Let l_{ui} denote the event that user u has chose to interact with item i
- Let's model the probability of that event happening by a logistic function

$$p(l_{ui} | x_u, y_i, \beta_u, \beta_i) = \frac{\exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)}$$

- Likelihood of our observations R given the parameters X, Y and β

$$\mathcal{L}(R | X, Y, \beta) = \prod_{u,i} p(l_{ui} | x_u, y_i, \beta_u, \beta_i)^{\alpha r_{ui}} (1 - p(l_{ui} | x_u, y_i, \beta_u, \beta_i))$$

- Regularized log likelihood will look like

$$\log p(X, Y, \beta | R) = \sum_{u,i} \alpha r_{ui} (x_u y_i^T + \beta_u + \beta_i) - (1 + \alpha r_{ui}) \log(1 + \exp(x_u y_i^T + \beta_u + \beta_i)) - \frac{\lambda}{2} \|x_u\|^2 - \frac{\lambda}{2} \|y_i\|^2$$

- Optimize with gradient descent

Logistic MF: Optimization

- Gradients look like

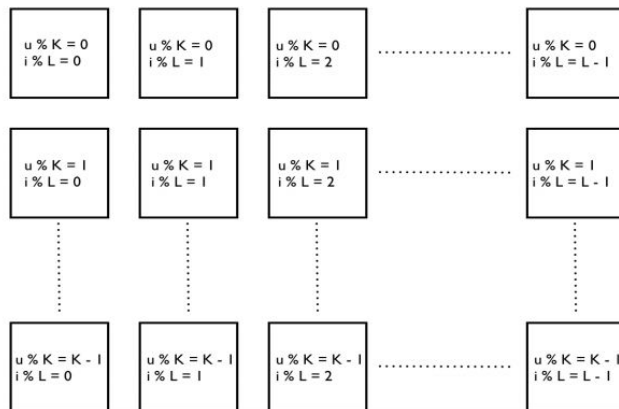
$$\frac{\partial}{\partial x_u} = \sum_i \alpha r_{ui} y_i - \frac{y_i (1 + \alpha r_{ui}) \exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)} - \lambda x_u$$

$$\frac{\partial}{\partial \beta_u} = \sum_i \alpha r_{ui} - \frac{(1 + \alpha r_{ui}) \exp(x_u y_i^T + \beta_u + \beta_i)}{1 + \exp(x_u y_i^T + \beta_u + \beta_i)}$$

- Naive approach will take too long!
- Solution - employ the Map-Reduce programming paradigm

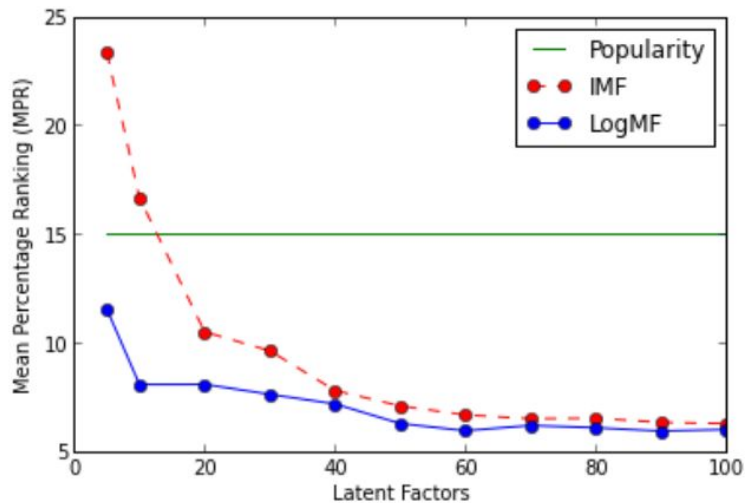
Logistic MF: Optimization

- Partition $(n \times m)$ matrix R into $(K \times L)$ blocks, where $K \ll n$, $L \ll m$



- Map phase: partition all observations, user vectors, and item vectors from the same block to the same mapper
- Reduce phase: key off u (or i if performing an item iteration) such that each v_{ui} and b_{ui} that map to the same user u (or item i if performing an item iteration) are sent to the same reducer
- Update the parameters and iterate til convergence

MF vs Logistic MF



Plot from C.Johnson - Logistic Matrix Factorization for Implicit Feedback Data

- Content based
- Collaborative filtering
 - Memory based
 - User based
 - Item based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

Generalization: Factorization Machines

Feature vector \mathbf{x}																	Target y					
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5	$y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3	$y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1	$y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4	$y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5	$y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1	$y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5	$y^{(6)}$
A B C ... User				TI NH SW ST ... Movie					TI NH SW ST ... Other Movies rated					Time	TI NH SW ST ... Last Movie rated							

Generalization: Factorization Machines

Feature vector \mathbf{x}																				Target y	
$\mathbf{x}^{(1)}$	1	0	0	...	1	0	0	0	...	0.3	0.3	0.3	0	...	13	0	0	0	0	...	5 $y^{(1)}$
$\mathbf{x}^{(2)}$	1	0	0	...	0	1	0	0	...	0.3	0.3	0.3	0	...	14	1	0	0	0	...	3 $y^{(2)}$
$\mathbf{x}^{(3)}$	1	0	0	...	0	0	1	0	...	0.3	0.3	0.3	0	...	16	0	1	0	0	...	1 $y^{(2)}$
$\mathbf{x}^{(4)}$	0	1	0	...	0	0	1	0	...	0	0	0.5	0.5	...	5	0	0	0	0	...	4 $y^{(3)}$
$\mathbf{x}^{(5)}$	0	1	0	...	0	0	0	1	...	0	0	0.5	0.5	...	8	0	0	1	0	...	5 $y^{(4)}$
$\mathbf{x}^{(6)}$	0	0	1	...	1	0	0	0	...	0.5	0	0.5	0	...	9	0	0	0	0	...	1 $y^{(5)}$
$\mathbf{x}^{(7)}$	0	0	1	...	0	0	1	0	...	0.5	0	0.5	0	...	12	1	0	0	0	...	5 $y^{(6)}$
	A	B	C	...	TI	NH	SW	ST	...	TI	NH	SW	ST	...	Time	TI	NH	SW	ST	...	
	User				Movie					Other Movies rated						Last Movie rated					

Allows to take into account for the final recommendation more complex information than just the user-item interaction!

1) *Model Equation*: The model equation for a factorization machine of degree $d = 2$ is defined as:

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{i=1}^n w_i x_i + \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \quad (1)$$

where the model parameters that have to be estimated are:

$$w_0 \in \mathbb{R}, \quad \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{V} \in \mathbb{R}^{n \times k} \quad (2)$$

And $\langle \cdot, \cdot \rangle$ is the dot product of two vectors of size k :

$$\langle \mathbf{v}_i, \mathbf{v}_j \rangle := \sum_{f=1}^k v_{i,f} \cdot v_{j,f} \quad (3)$$

A row \mathbf{v}_i within \mathbf{V} describes the i -th variable with k factors. $k \in \mathbb{N}_0^+$ is a hyperparameter that defines the dimensionality of the factorization.

A 2-way FM (degree $d = 2$) captures all single and pairwise interactions between variables:

- w_0 is the global bias.
- w_i models the strength of the i -th variable.
- $\hat{w}_{i,j} := \langle \mathbf{v}_i, \mathbf{v}_j \rangle$ models the interaction between the i -th and j -th variable. Instead of using an own model parameter $w_{i,j} \in \mathbb{R}$ for each interaction, the FM models the interaction by factorizing it. We will see later on, that this is the key point which allows high quality parameter estimates of higher-order interactions ($d \geq 2$) under sparsity.

Text from S. Rendle - Factorization machines

Factorization Machines: Computational considerations

- The complexity of straightforward computation of $eq(1)$ from previous slide is $O(kn^2)$, because all pairwise interactions should be computed
- Possible to compute in linear time!

$$\begin{aligned} & \sum_{i=1}^n \sum_{j=i+1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j \\ &= \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \langle \mathbf{v}_i, \mathbf{v}_j \rangle x_i x_j - \frac{1}{2} \sum_{i=1}^n \langle \mathbf{v}_i, \mathbf{v}_i \rangle x_i x_i \\ &= \frac{1}{2} \left(\sum_{i=1}^n \sum_{j=1}^n \sum_{f=1}^k v_{i,f} v_{j,f} x_i x_j - \sum_{i=1}^n \sum_{f=1}^k v_{i,f} v_{i,f} x_i x_i \right) \\ &= \frac{1}{2} \sum_{f=1}^k \left(\left(\sum_{i=1}^n v_{i,f} x_i \right) \left(\sum_{j=1}^n v_{j,f} x_j \right) - \sum_{i=1}^n v_{i,f}^2 x_i^2 \right) \\ &= \frac{1}{2} \sum_{f=1}^k \left(\left(\sum_{i=1}^n v_{i,f} x_i \right)^2 - \sum_{i=1}^n v_{i,f}^2 x_i^2 \right) \end{aligned}$$

Factorization Machines as Predictors

FM can be applied to variety of prediction tasks

- Regression - $y(x)$ can be used directly as the predictor, and the optimization criterion is e.g. the RMSE
- Binary classification - the sign of $y(x)$ is used and the parameters are optimized for hinge loss or logit loss
- Ranking - the vectors x are ordered by the score of $y(x)$ and optimization is done over pairs of instance vectors, with a pairwise classification loss

Optimization: Simple gradient descent, one iteration done in $O(kn)$ time (actually in $O(km(x))$, where $m(x)$ is the number of non-zero elements in feature matrix).

Factorization Machines

- The order d of FM can be taken >2 , to take into account interactions of higher orders
- Even in case of $d>2$ the computation complexity still can be reduced to $O(kn)$ with the same trick
- FM contain many other methods as special case! (SVD, MF, SVD++, Tensor Factorization, etc)

**LITERALLY NOBODY:
ML FOLKS: NEURAL NETS SHOULD WORK**



Adding Neural Networks to the Mix

Limitation of inner product space for MF

In this picture the ground truth similarity that MF should recover is Jaccard coefficient

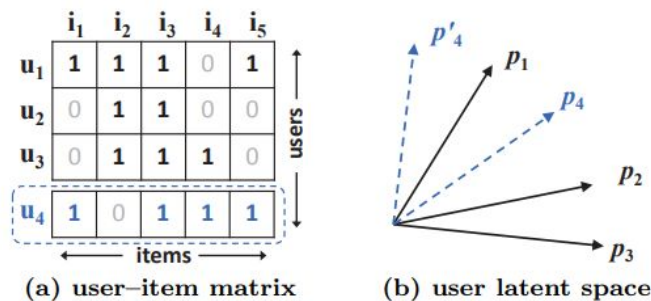


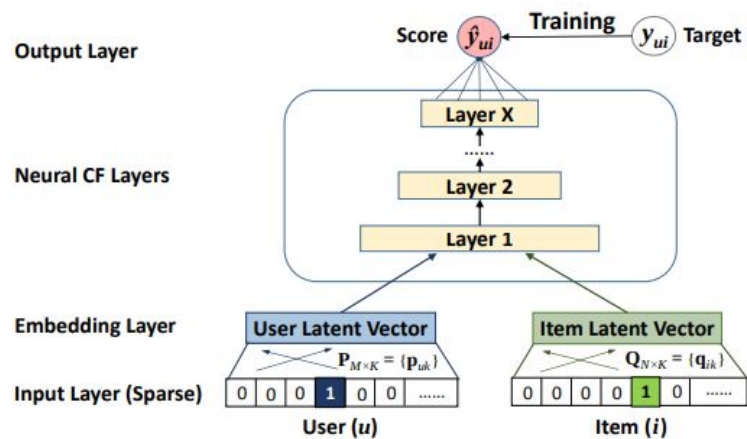
Figure 1: An example illustrates MF's limitation. From data matrix (a), u_4 is most similar to u_1 , followed by u_3 , and lastly u_2 . However in the latent space (b), placing p_4 closest to p_1 makes p_4 closer to p_2 than p_3 , incurring a large ranking loss.

Figure from X.HE et al 2017 - Neural Collaborative Filtering

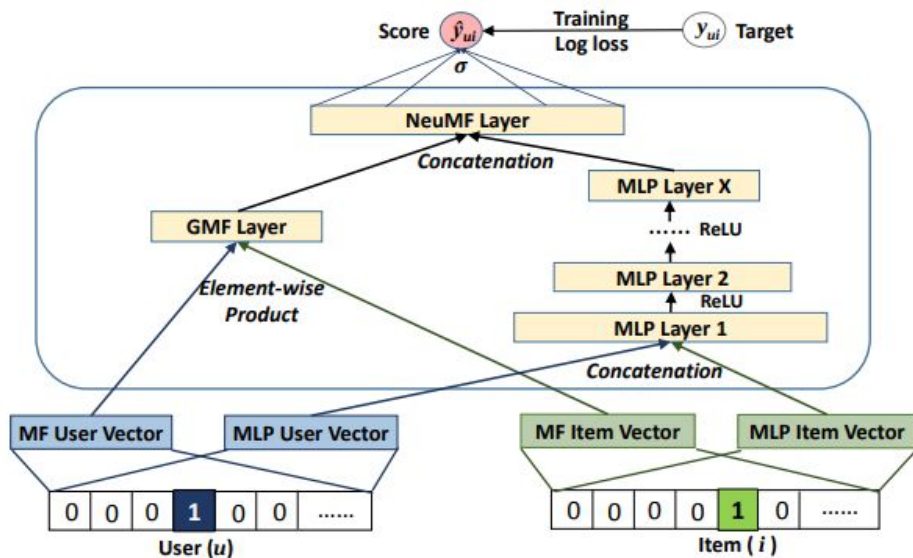
- Content based
- Collaborative filtering
 - Memory based
 - User based
 - Item based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

Neural Collaborative Filtering: Generalized MF

Includes MF as a special case!



Fusion of GMF and MLP



Neural Collaborative Filtering

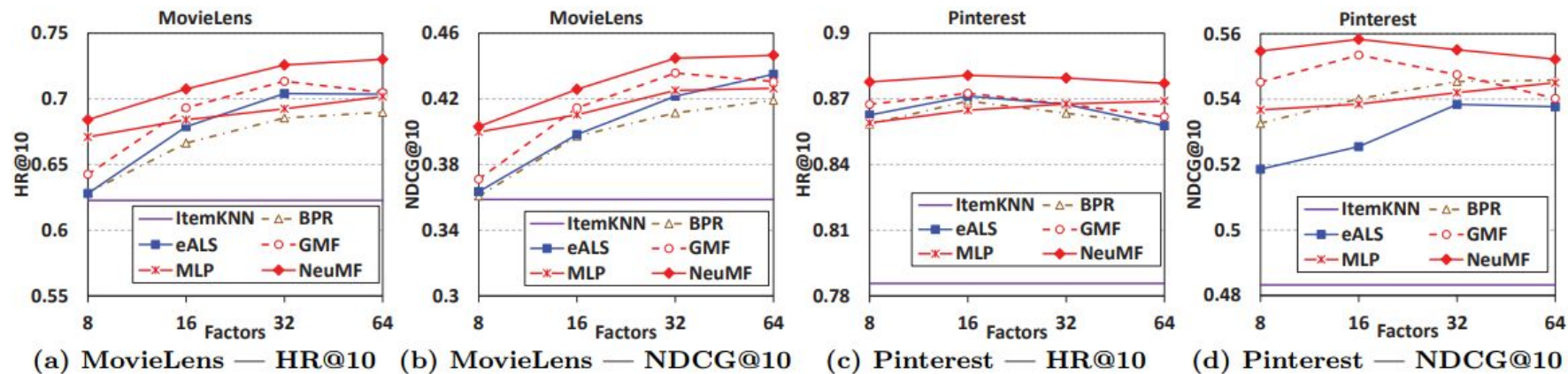


Figure 4: Performance of HR@10 and NDCG@10 *w.r.t.* the number of predictive factors on the two datasets.

Neural Collaborative Filtering

Going deeper helps, til it loses meaning

Table 3: HR@10 of MLP with different layers.

Factors	MLP-0	MLP-1	MLP-2	MLP-3	MLP-4
MovieLens					
8	0.452	0.628	0.655	0.671	0.678
16	0.454	0.663	0.674	0.684	0.690
32	0.453	0.682	0.687	0.692	0.699
64	0.453	0.687	0.696	0.702	0.707
Pinterest					
8	0.275	0.848	0.855	0.859	0.862
16	0.274	0.855	0.861	0.865	0.867
32	0.273	0.861	0.863	0.868	0.867
64	0.274	0.864	0.867	0.869	0.873

Figure from X.HE et al 2017 - Neural Collaborative Filtering

A real production example: Wide and Deep Network

- Wide set of cross-product features with simple linear model
 - Effective and interpretable
 - Requires good feature engineering for generalization
- Deep neural networks
 - Require less feature engineering
 - Good generalization
 - Sometimes overgeneralize, when user-item interactions are sparse and high-rank
- Why not to join these two methods in one learning framework?

A real production example: Wide and Deep Network

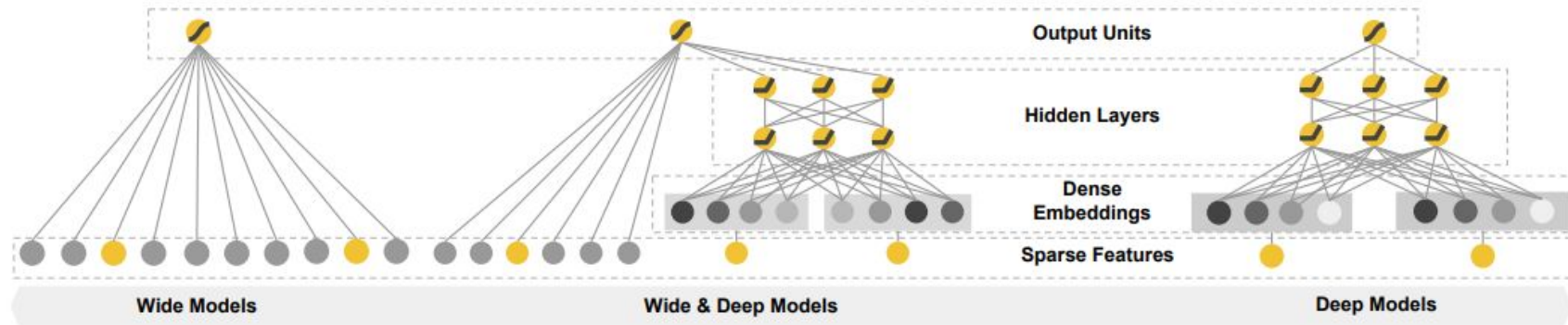
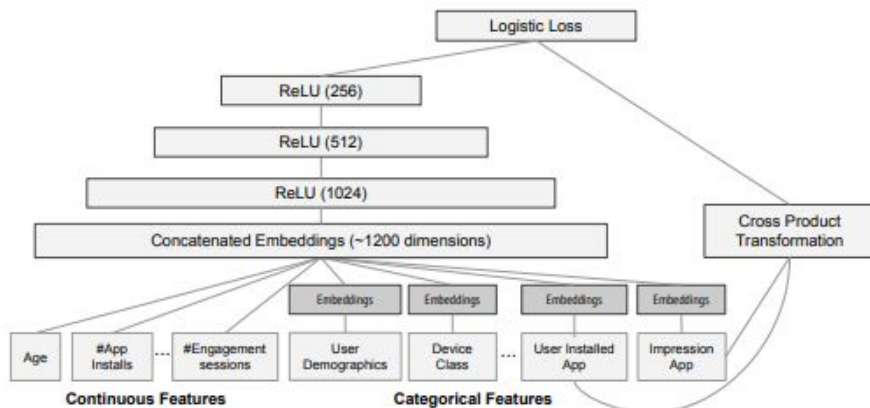


Figure 1: The spectrum of Wide & Deep models.

Figure from Cheng et al 2016 - Wide & Deep Learning for Recommender Systems

Wide and Deep Network: Implementation for Google Play



- Trained on over 500 billion examples
- Every time new set of training data arrives, model is fine-tuned
- Significantly outperforms wide-only and deep-only models
- TensorFlow implementation open-sourced by authors

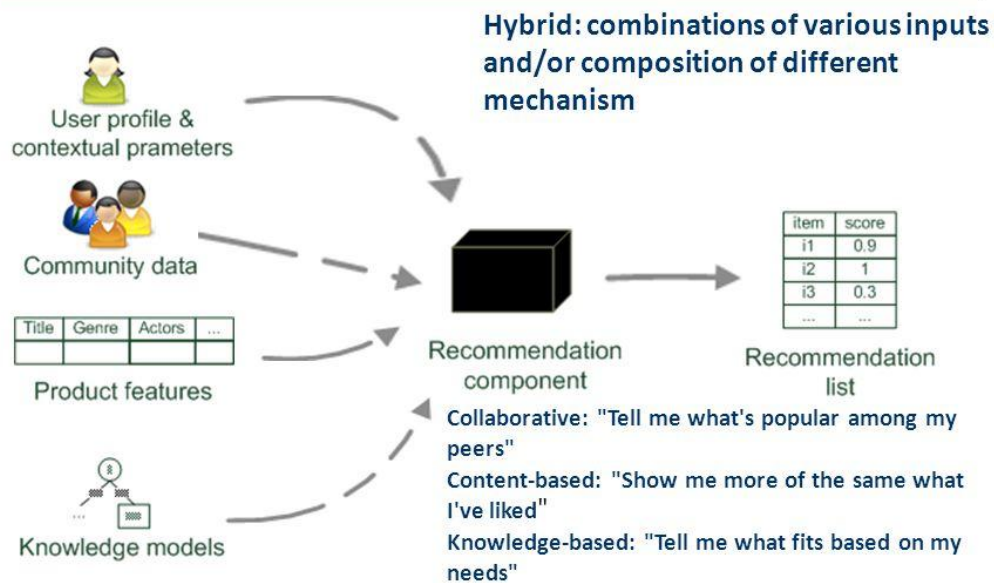
CF Summary

- Requires only ratings, widely applicable
- Neighborhood methods, latent factors
- Use of machine learning techniques
- Limitations
 - Cold start problem: how to recommend new items, what to recommend to new users
 - Popularity bias: difficult to recommend items from the long tail

- Content based
- Collaborative filtering
 - Memory based
 - User based
 - Item based
 - Model based
 - Matrix Factorization
 - Factorization machines
 - Neural Collaborative Filtering
- Hybrid RS

Hybrid RS

Hybrid recommender systems



Hybridization methods

Hybridization method	Description
Weighted	The scores (or votes) of several recommendation techniques are combined together to produce a single recommendation.
Switching	The system switches between recommendation techniques depending on the current situation.
Mixed	Recommendations from several different recommenders are presented at the same time
Feature combination	Features from different recommendation data sources are thrown together into a single recommendation algorithm.
Cascade	One recommender refines the recommendations given by another.
Feature augmentation	Output from one technique is used as an input feature to another.
Meta-level	The model learned by one recommender is used as input to another.

Table from R.Burke 2002 - Hybrid Recommender Systems: Survey and Experiments
(not much has changed since then in this regard)

Netflix Prize Winner: The BigChaos solution

Models used in the winning solution

- Neighborhood models (different versions of movie kNN)
- Latent Factor Models (SVD, NSVD, SVD++, time SVD, etc)
- Restricted Boltzmann Machines
- Regression
- Others. Full list of models https://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf

Ensembling of over 500 models done with Gradient Boosted Decision Trees (other ensembling methods were tried as well, like linear regression, neural networks)

This winning solution was never actually used in production as it is:

- Engineering costs weren't worth it
- Soon after the competition the data distribution of Netflix was changed

THANK YOU!