

Specialist Programme on Artificial Intelligence for IT & ITES Industry

Recommender Systems (part2)

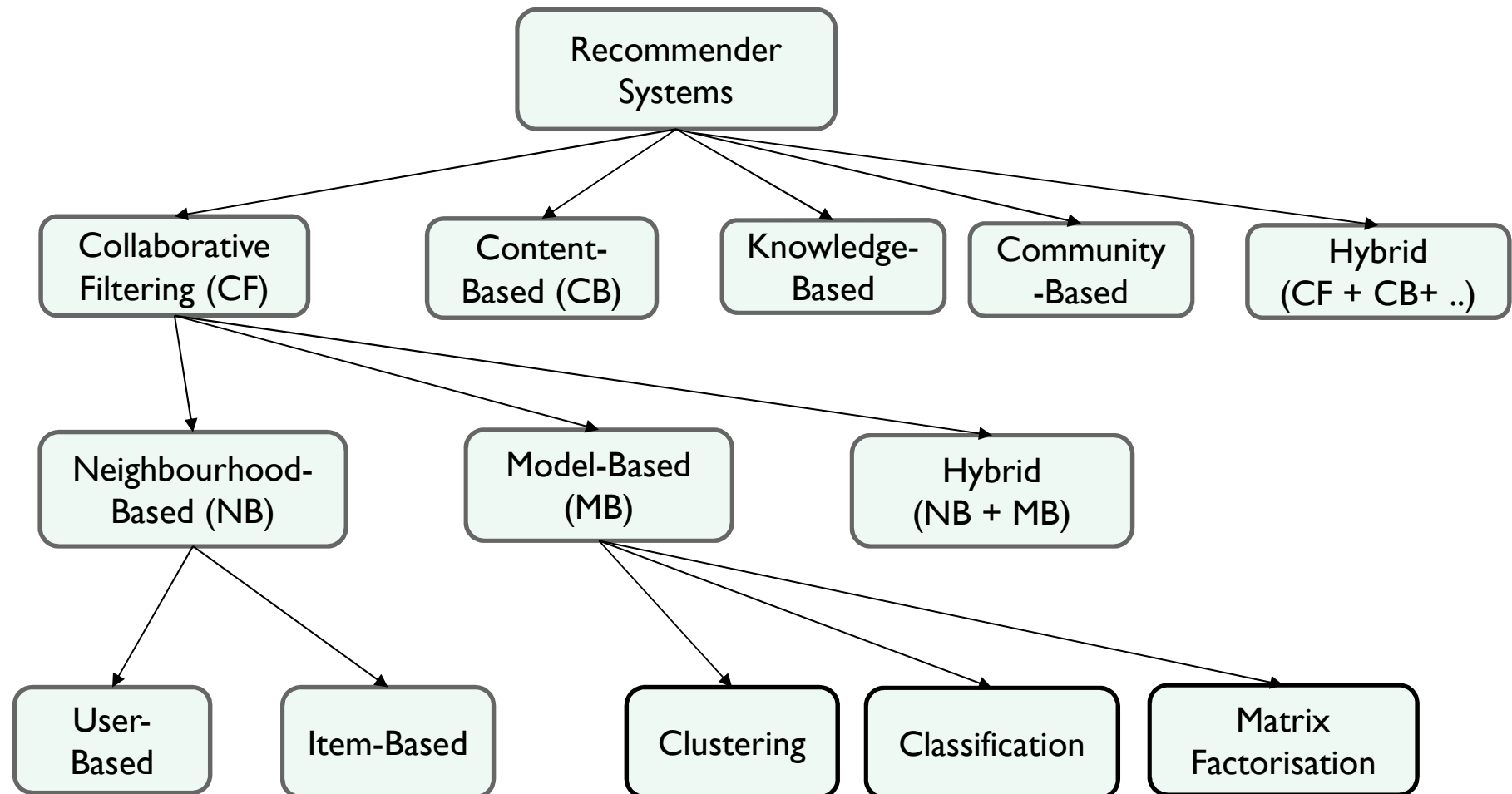
Dr Barry Shepherd
barryshepherd@nus.edu.sg

Singapore e-Government Leadership Centre
National University of Singapore

© 2020 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

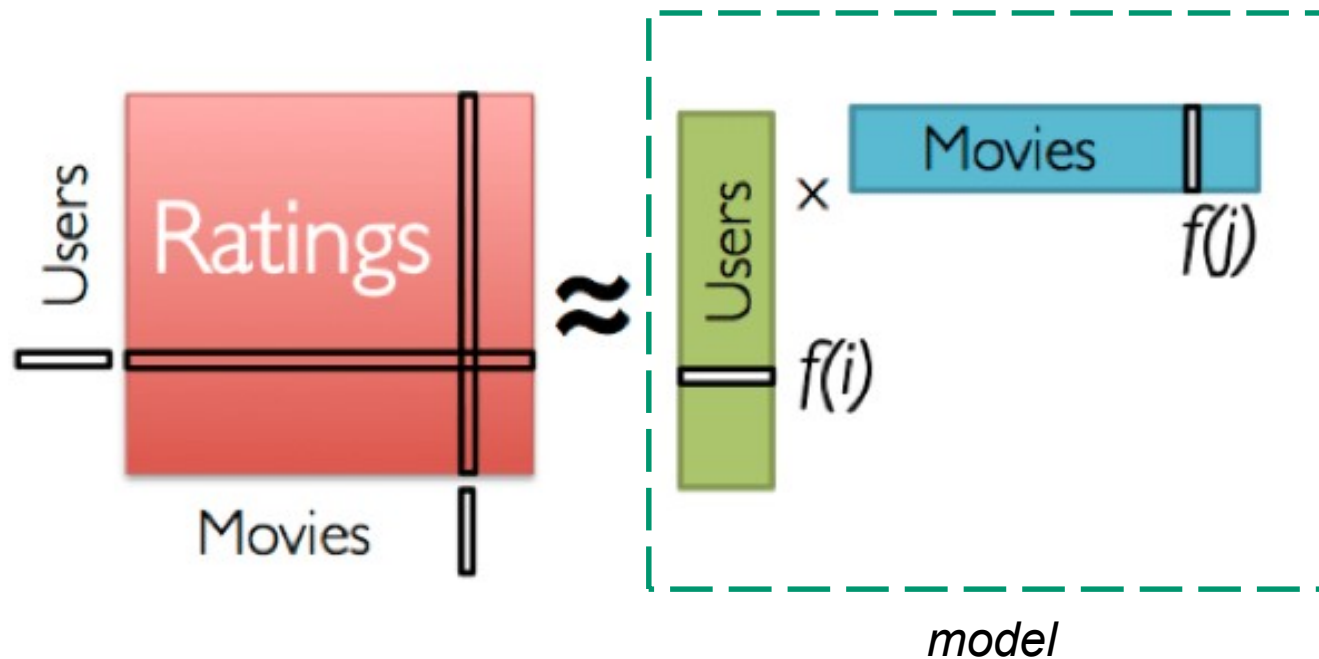
Inspire *Lead* *Transform*

Recommender System Approaches



Matrix Factorisation Methods

- Applies matrix factorisation methods to the ratings matrix
- Reduces it down to two smaller matrices which can be thought of as a model. Once we have done the factorisation we make rating predictions using the two factor matrices – not the original ratings matrix



Matrix Factorisation Concept

- Assume the items to be recommended have a set of properties.
E.g. Movies could have properties: type, actors, directors, film studios, location, length etc. Properties are numerical strengths, e.g. values 0 (low) to 10 (high)

	Length	Disney	Universal	Action	Comedy	SciFi	T. Hanks	B. Pitt
Movie1	4	10	0	2	8	0	8	2
Movie2	8	0	10	9	1	7	0	9

- Assume users express their preferences using the same properties

	Length	Disney	Universal	Action	Comedy	SciFi	T. Hanks	B. Pitt
User1	2	7	5	4	8	5	5	9
User2	9	3	7	9	1	7	2	5

Matrix Factorisation Concept

- The users rating for any movie can now be computed as a simple *dot* product of user preferences * movie properties
- E.g. User1's rating score for movie 2 is
 $= 2*8 + 7*0 + 5*10 + \dots$ (to normalise divide by the max score)

	Length	Disney	Universal	Action	Comedy	Sci.Fi	T. Hanks	B. Pitt
M1	4	10	0	2	8	0	8	2
M2	8	0	10	9	1	7	0	9

	Length	Disney	Universal	Action	Comedy	Sci.Fi	T. Hanks	B. Pitt
U1	2	7	5	4	8	5	5	9
U2	9	3	7	9	1	7	2	5

Matrix Factorisation Concept

- BUT we don't have the movie properties or user preferences, we don't even know what the properties should be. We only have a set of ratings from the users (the ratings matrix **R**). E.g.

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	MovieN
U1	-	3	-	-	8	0	3
U2	8	0	10	9	1	7	-
U3							6

- Solution ~ Assume the movies have K properties (e.g. say 100) and use matrix factorisation on **R** to derive the user preference matrix (**U**) and the movie properties matrix (**M**):

$$\mathbf{R} = \mathbf{U}^T \mathbf{M}$$

- We don't need to know what the K properties are, we just assume they exist, they are called **latent** (hidden) variables

Matrix Factorisation Concept

$$\begin{array}{ccc}
 \text{User ratings} & & \text{User preferences} \quad * \quad \text{Movies properties} \\
 \left[\begin{array}{ccc} 1 & \dots & M \\ \vdots & & \\ U & & \end{array} \right] & = & \left[\begin{array}{ccc} 1 & \dots & K \\ \vdots & & \\ U & & \end{array} \right] \\
 (500K * 17K = 8,500M) & & (500K * 100 = 50M)
 \end{array}$$

model

- The resulting matrices form the model ~ usually a much smaller dataset!
- 50 to 100 latent features are often all that is required for good results

Matrix Factorisation Concept

- Once the factorisation is done we can predict the ratings for all users and all items

		Item			
		W	X	Y	Z
User	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

$$R = U * I^T$$

Why are the known ratings not always predicted correctly!

Performing the Factorisation

- SVD (Singular Value Decomposition) is common method for matrix factorisation
- Issues ~ much of the ratings matrix is empty (very sparse matrix)
- Straight SVD doesn't work. Hard to find exact factors
- Instead the Netflix winner used incremental learning solution based on gradient descent by taking derivatives of the approximation error

E.g. Use stochastic gradient descent to minimise the squared error between the predicted rating and the actual rating:

$$\min_{x_*, y_*} \sum_{r_{u,i} \text{ is known}} (r_{ui} - x_u^T y_i)^2 + \lambda (\|x_u\|^2 + \|y_i\|^2)$$

Actual rating
Predicted rating
Regularisation Term

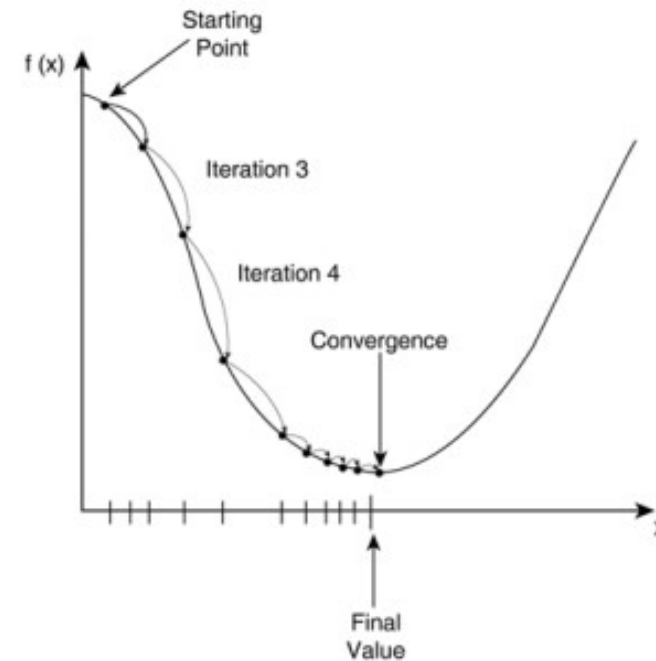
Factorising using Gradient Descent*

Procedure:

Present the training examples in turn and update the weights after each (similar to NN learning).

Continue until convergence (error stops reducing)

Training examples are triplets:
(user, movie, rating)



- An example update rule for the user preference matrix \mathbf{U} and the item properties matrix \mathbf{M} is:

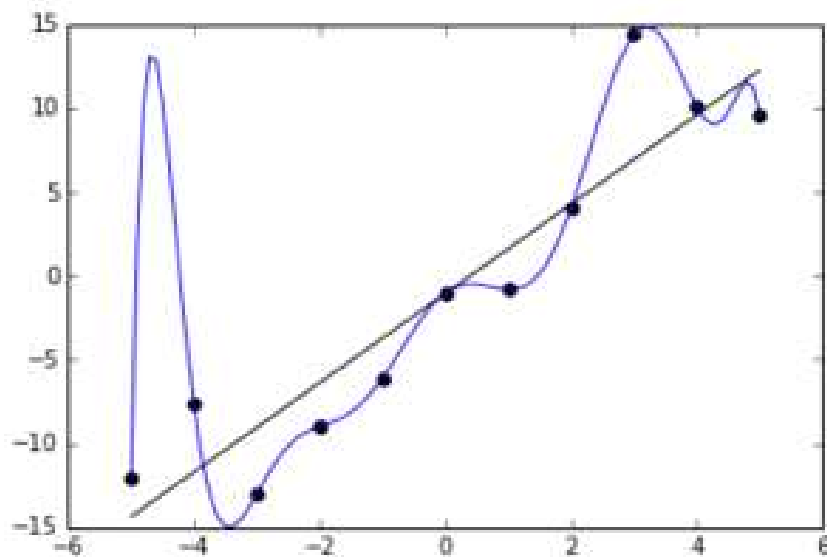
$$\begin{aligned} u_{ki}^{t+1} &= u_{ki}^t + 2\gamma (r_{ij} - p_{ij}) m_{kj}^t - \lambda u_{ki}^t \\ m_{kj}^{t+1} &= m_{kj}^t + 2\gamma (r_{ij} - p_{ij}) u_{ki}^t - \lambda m_{kj}^t \end{aligned}$$

u_{ki} = value of kth preference for user i
 m_{kj} = value of kth property for item j
 r_{ij} = rating of user i on item j
 p_{ij} = predicted rating of user i on item j
 γ = the learning rate
 λ = the regularisation constraint

*Method by Simon Funk (Netflix competitor)

Overfitting & Regularisation

- Given a high enough degree, a polynomial (or other model) can be tuned to exactly learn the training data (over-fit). This is bad, the model will not generalise well.
- Regularization tries to avoid overfitting by adding an additional penalty term into the error function. The additional term keeps the coefficients small so the model doesn't become too extreme



$$L(x, y) = \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2$$

$$\text{where } h_{\theta}x_i = \theta_0 + \theta_1x_1 + \theta_2x_2^2 + \theta_3x_3^3 + \theta_4x_4^4$$

$$L(x, y) \equiv \sum_{i=1}^n (y_i - h_{\theta}(x_i))^2 + \lambda \sum_{i=1}^n \theta_i^2$$

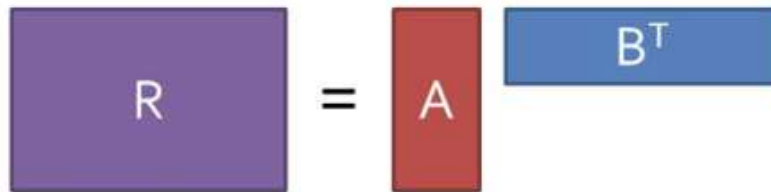


$$f(x_i) = h_{\theta}x = \theta_0 + \theta_1x_1 + \theta_2x_2^2 + \theta_3x_3^3 + \theta_4x_4^4$$

$$f(x_i) = h_{\theta}x = \theta_0 + \theta_1x_1 + \theta_2x_2^2$$

Alternating Least Squares Algorithm

Model R as product of user and movie feature matrices A and B of size $U \times K$ and $M \times K$



$$R = A B^T$$

We wish to find values for a_i and b_j that minimises the total squared error:

$$\text{cost} = \sum_{ij} (r_{ij} - a_i * b_j)^2$$

- If we fix B and optimize for A alone, the problem is simply reduced to the problem of linear regression.
- Recall that in linear regression we solve for β by minimizing the squared error $\|y - \beta X\|^2$ given X and y .
- The solution is given by the Ordinary Least Squares (OLS) formula $\beta = (X^T X)^{-1} X^T y$



Alternating Least Squares (ALS)

- » Start with random A & B
- » Optimize user vectors (A) based on movies
- » Optimize movie vectors (B) based on users
- » Repeat until converged

<https://datasciencemadesimpler.wordpress.com/tag/alternating-least-squares/>

Alternating Least Squares

$$R = A * B^T$$

	i1	i2	i3	i4	i5	i6	i7	i8
u1		4			5			
u2	1				2		5	
u3		3	4	3		5		
u4	2	2			5	5		3
u5			1			4		2
u6	3			5	4		3	

	f1	f2	f3	f4
u1	a1	a2	a3	a4
u2	a5	a6	a7	a8
u3	a9	a10	a11	a12
u4	a13	a14	a15	a16
u5	a17	a18	a19	a20
u6	a21	a22	a23	a24

	i1	i2	i3	i4	i5	i6	i7	i8
f1	b1	b2	b3	b4	b5	b6	b7	b8
f2	b9	b10	b11	b12	b13	b14	b15	b16
f3	b17	b18	b19	b20	b21	b22	b23	b24
f4	b25	b26	b27	b28	b29	b30	b31	b32

E.g. $4 = (a1 * b2 + a2 * b10 + a3 * b18 + a4 * b26)$

- Step1:** Assign random weights to A and B
(weights and ratings should be normalised to range 0 to 1)
- Step2 :** fix A and use linear regression* to estimate weights in B
will require building 8 (#items) regression models
- Step3:** fix B and use linear regression* to estimate weights in A
will require building 6 (#users) regression models
- Repeat** from step2 until convergence (or time limit etc)

* To help avoid overfitting we conceptually substitute the equivalent of:

- Lasso-regression, or
 - Ridge-regression
- (for L1 and L2 regularisation)

$$\sum_{i=1}^n (Y_i - \sum_{j=1}^p X_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p |\beta_j|$$

$$\sum_{i=1}^n (y_i - \sum_{j=1}^p x_{ij} \beta_j)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

Alternating Least Squares - Conceptual

$$R = A * B^T$$

	i1	i2	i3	i4	i5	i6	i7	i8
u1		4			5			
u2	1				2		5	
u3		3	4	3		5		
u4	2	2			5	5		3
u5			1			4		2
u6	3			5	4		3	

	f1	f2	f3	f4
u1	a1	a2	a3	a4
u2	a5	a6	a7	a8
u3	a9	a10	a11	a12
u4	a13	a14	a15	a16
u5	a17	a18	a19	a20
u6	a21	a22	a23	a24

	i1	i2	i3	i4	i5	i6	i7	i8
f1	b1	b2	b3	b4	b5	b6	b7	b8
f2	b9	b10	b11	b12	b13	b14	b15	b16
f3	b17	b18	b19	b20	b21	b22	b23	b24
f4	b25	b26	b27	b28	b29	b30	b31	b32

E.g. training data to learn the i2 weights in B:

x1	x2	x3	x4	Output (Y)
a1	a2	a3	a4	4
a9	a10	a11	a12	3
a13	a14	a15	a16	2



$$Y = MX + C$$

f1	f2	f3	f4
b2	b10	b18	b26

... training data to learn the i7 weights in B:

x1	x2	x3	x4	Output (Y)
a5	a6	a7	a8	5
a21	a22	a23	a24	3



f1	f2	f3	f4
b7	b15	b23	b31

Alternating Least Squares - Conceptual

$$R = A * B^T$$

	i1	i2	i3	i4	i5	i6	i7	i8
u1		4			5			
u2	1				2		5	
u3		3	4	3		5		
u4	2	2			5	5		3
u5			1			4		2
u6	3			5	4		3	

	f1	f2	f3	f4
u1	a1	a2	a3	a4
u2	a5	a6	a7	a8
u3	a9	a10	a11	a12
u4	a13	a14	a15	a16
u5	a17	a18	a19	a20
u6	a21	a22	a23	a24

	i1	i2	i3	i4	i5	i6	i7	i8
f1	b1	b2	b3	b4	b5	b6	b7	b8
f2	b9	b10	b11	b12	b13	b14	b15	b16
f3	b17	b18	b19	b20	b21	b22	b23	b24
f4	b25	b26	b27	b28	b29	b30	b31	b32

E.g. training data to learn the u2 weights in A:

x1	x2	x3	x4	Output (Y)
b1	b9	b17	b25	1
b5	b13	b21	b29	2
b7	b15	b23	b31	5



$$Y = MX + C$$

f1	f2	f3	f4
a5	a6	a7	a8

... training data to learn the u5 weights in A:

x1	x2	x3	x4	Output (Y)
				1
				4
				2



f1	f2	f3	f4
a17	a18	a19	a20

ALS: Handling New Users / Items

- No cold-start: Cannot predict for a user with no existing ratings (or a new item)
 - Ask the user for sample ratings, else wait until they have done some page-views, transactions etc
- Must re-factorise (which can take time)...

$$\begin{matrix} & \mathbf{R} & = & \mathbf{A} & * & \mathbf{B}^T \\
 \begin{array}{c|cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & i7 & i8 \\ \hline u1 & & 4 & & & 5 & & & \\ u2 & 1 & & & & 2 & & 5 & \\ u3 & & 3 & 4 & 3 & & 5 & & \\ u4 & 2 & 2 & & & 5 & 5 & & 3 \\ u5 & & & 1 & & & 4 & & 2 \\ u6 & 3 & & & 5 & 4 & & 3 & \\ u7 & & 2 & 4 & & & 1 & & \end{array} & & & \begin{array}{c|cccc} & f1 & f2 & f3 & f4 \\ \hline u1 & a1 & a2 & a3 & a4 \\ u2 & a5 & a6 & a7 & a8 \\ u3 & a9 & a10 & a11 & a12 \\ u4 & a13 & a14 & a15 & a16 \\ u5 & a17 & a18 & a19 & a20 \\ u6 & a21 & a22 & a23 & a24 \\ u7 & ? & ? & ? & ? \end{array} & & & \begin{array}{c|cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & i7 & i8 \\ \hline f1 & b1 & b2 & b3 & b4 & b5 & b6 & b7 & b8 \\ f2 & b9 & b10 & b11 & b12 & b13 & b14 & b15 & b16 \\ f3 & b17 & b18 & b19 & b20 & b21 & b22 & b23 & b24 \\ f4 & b25 & b26 & b27 & b28 & b29 & b30 & b31 & b32 \end{array}
 \end{matrix}$$

- Or factorise just for the new user (reuse the existing item factors matrix)

Other Popular Factorisation Methods

- Non-Negative Matrix Factorisation
 - https://en.wikipedia.org/wiki/Non-negative_matrix_factorization
- Logistic Matrix Factorisation
- SVD++
 - Incorporates Explicit and Implicit ratings
- Netflix and Parallel ALS
 - https://www.researchgate.net/publication/220788980_Large-Scale_Parallel_Collaborative_Filtering_for_the_Netflix_Prize

ALS with Implicit Feedback

- An implicit like ~ viewing a product page, clicking an item, purchase etc.
- Previously we treated implicit feedback as binary data, e.g.

IF #implicit likes $\geq N$ then rating = 1 else unknown ($N \sim 1$)

- How can we better utilise the #implicit likes, or page-view dwell time?
- How can we better handle items with no implicit likes?

Implicit Ratings matrix

	i1	i2	i3	i4	i5	i6	i7	i8
u1		4			5			
u2	1				2		5	
u3		3	4	3		5		
u4	2	2			5	5		3
u5			1			4		2



Binarised Implicit Ratings matrix

	i1	i2	i3	i4	i5	i6	i7	i8
u1		1			1			
u2	1				1		1	
u3		1	1	1		1		
u4	1	1			1	1		1
u5			1			1		1



Since there are no negative cases (no dislikes) then if we use the number directly then when we predict the users rating on an unseen item we are really predicting *how much* the user likes the item NOT if the user actually likes the item

ALS with Implicit Feedback *

<http://yifanhu.net/PUB/cf.pdf>

- Assume each implicit rating = number of views or buys for an item by a user

Implicit Ratings matrix

	i1	i2	i3	i4	i5	i6	i7	i8
u1		4			5			
u2	1				2		5	
u3		3	4	3		5		
u4	2	2			5	5		3
u5			1			4		2



Convert to preferences
 $P = 1$ if implicit rating > 0
 else 0

Preference matrix

	i1	i2	i3	i4	i5	i6	i7	i8
u1	0	1	0	0	1	0	0	0
u2	1	0	0	0	1	0	1	0
u3	0	1	1	1	0	1	0	0
u4	1	1	0	0	1	1	0	1
u5	0	0	1	0	0	1	0	1

Assign a confidence value to each preference:

$$C = 1 + \alpha * \text{rating} \quad (\alpha \sim 40)$$

e.g. more views \Rightarrow more confidence

Proceed as with ALS but with a new cost function:

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

where: $u \sim \text{users}$, $i \sim \text{items}$, x is the user factors matrix, y is the item factors matrix

Very popular alg, e.g. see Facebook system

<https://code.fb.com/core-data/recommending-items-to-more-than-a-billion-people/>

Libraries: Implicit (Python)

Implicit

Fast Python Collaborative Filtering for Implicit Datasets

This project provides fast Python implementations of several different popular recommendation algorithms for implicit feedback datasets:

- Alternating Least Squares as described in the papers [Collaborative Filtering for Implicit Feedback Datasets](#) and in [Applications of the Conjugate Gradient Method for Implicit Feedback Collaborative Filtering](#).
- [Bayesian Personalized Ranking](#)
- Item-Item Nearest Neighbour models, using Cosine, TFIDF or BM25 as a distance metric

All models have multi-threaded training routines, using Cython and OpenMP to fit the models in parallel among all available CPU cores. In addition, the ALS and BPR models both have custom CUDA kernels - enabling fitting on compatible GPU's. This library also supports using approximate nearest neighbours libraries such as [Annoy](#), [NMSLIB](#) and [Faiss](#) for [speeding up making recommendations](#).

Libraries: Surprise (Python)

 surprise

A Python scikit for
recommender systems.

Overview

Surprise is a Python `scikit` building and analyzing recommender systems that deal with explicit rating data.

Surprise was designed with the following purposes in mind:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on `documentation`, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of `Dataset handling`. Users can use both *built-in* datasets (`Movielens`, `Jester`), and their own *custom* datasets.
- Provide various ready-to-use `prediction algorithms` such as `baseline algorithms`, `neighborhood methods`, matrix factorization-based (`SVD`, `PMF`, `SVD++`, `NMF`), and *many others*. Also, various `similarity measures` (cosine, MSD, pearson...) are built-in.
- Make it easy to implement `new algorithm ideas`.
- Provide tools to `evaluate`, `analyse` and `compare` the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by `scikit-learn` excellent tools), as well as `exhaustive search over a set of parameters`.

The name *SurPRISE* (roughly :)) stands for Simple Python Recommendation System Engine.

Please note that surprise does not support implicit ratings or content-based information.

Libraries: RecoSystem (Python & R)

LIBMF and recosystem

LIBMF

is an open source C++ library for recommender system using parallel matrix factorization, developed by Dr. Chih-Jen Lin and his research group.
[3]

LIBMF is a parallelized library, meaning that users can take advantage of multi-core CPUs to speed up the computation. It also utilizes some advanced CPU features to further improve the performance.

LIBMF: A Matrix-factorization Library for Recommender Systems

Machine Learning Group at National Taiwan University

Version 2.01 released on February 20, 2016. LIBMF can solve more formulations than its previous versions :

Language	Description	Maintainers and Their Affiliation	Supported LIBMF version	Link
R	R interface to LIBMF	Yixuan Qiu from Purdue University	2.01	R LIBMF
Python	Python interface to LIBMF	Sam Royston	2.01	Python LIBMF

Libraries: softImpute (R)

`softImpute` is a package for matrix completion using nuclear norm regularization

Offers two algorithms:

1. `type="svd"` in the call to `softImpute`

This iteratively computes the soft-thresholded SVD of a filled in matrix - an algorithm described in [Mazumder et al \(2010\)](#).

2. `type="als"` in the call to `softImpute`

Performs alternating ridge regression, at each stage filling in the missing entries with the latest estimates. This we believe is the faster option.

The package can deal with both small and very large matrices; the latter are stored in sparse-matrix format using a new S4 class `"Incomplete"`. For example, `softImpute` can happily fit a rank 100 SVD to the netflix data (480,189 x 17,770, 99% missing) using a machine with about 32Gb of memory. For smaller matrices with missing data, the usual full matrix with `NA` suffices.

Libraries: Rrecsys (R)

rrecsys

This is a package in R that provides implementations of several baselines (Item/User Average and Most Popular Item Recommendation and other well-known recommendation algorithms). In particular, two main families of recommendation algorithms (i.e., Collaborative filtering and Matrix factorization) are implemented, as shown in the following:

1. Collaborative filtering

- Weighted Slope One
- User-based k-nearest neighbour
- Item-based k-nearest neighbour

1. Matrix factorization

- Simon Funk's SVD with stochastic gradient descent
- weighted Alternated Least Squares (wALS)
- Bayesian Personalized Ranking (BPR)

rrecsys addresses the two most common scenarios in Recommender Systems:

- Rating Prediction (e.g. on a scale of 1 to 5 stars), and
- Item Recommendations (e.g. a list of top-N recommended items).

<https://cran.r-project.org/web/packages/rrecsys/>

Pros & Cons of Matrix Factorisation

Pros

- Generally more accurate than User-based and Item-based CF

Cons

- They provide no explanation on the reasons behind a recommendation
- Factorisation must be redone to apply to new users / new items

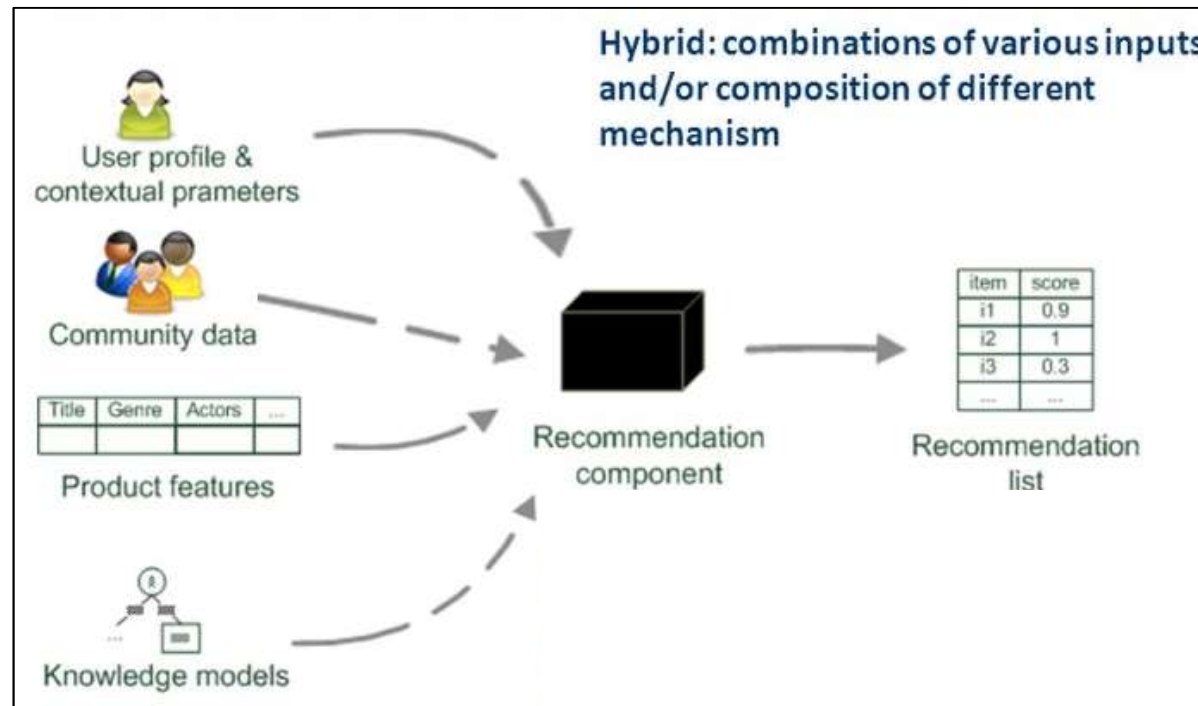
Hybrid Recommender Systems

- Often combining many different systems can generate better results
- E.g. Pandora Recommendation Engine
 - “The recommender uses about 70 different algorithms: 10 analyze content, 40 process collective intelligence, and then another 30 do personalized filtering.

Celma said, "This is challenging from an engineering point of view. We have the goal that when you thumb down a song, the recommendation for the next song occurs in less than 100 milliseconds. It is hard to do this in a way that scales across all users."

- E.g. Netflix Prize Winner
 - The winner of the Netflix prize was a combination of 107 algorithms

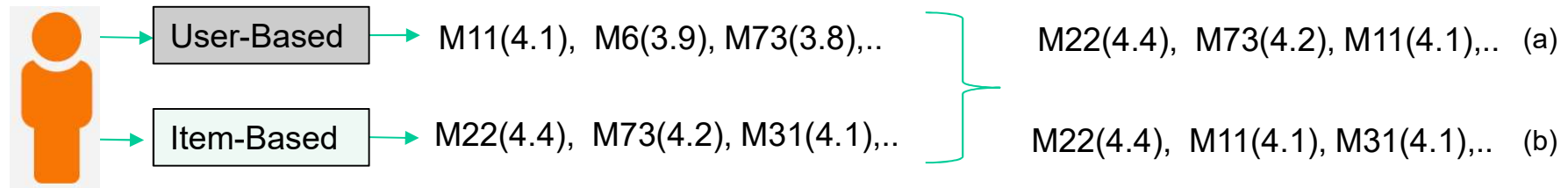
Hybrid Recommender Systems



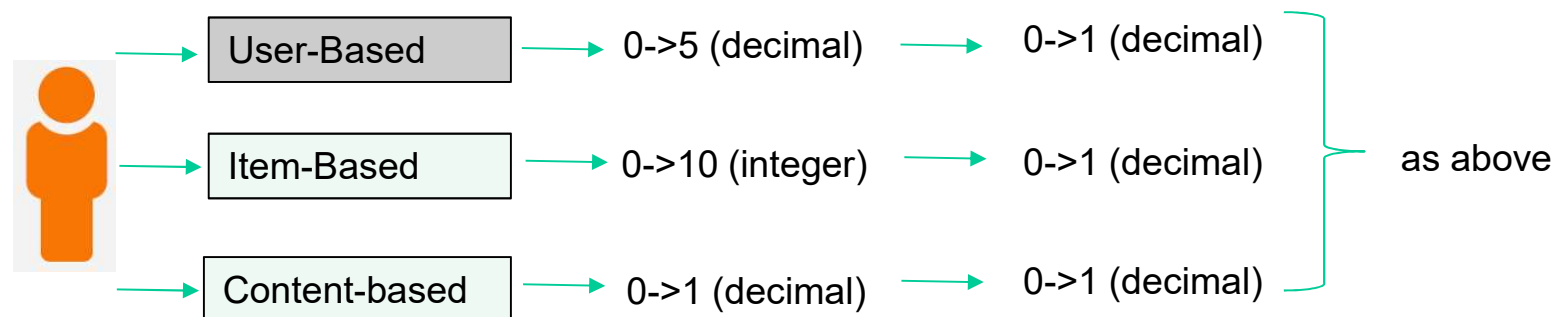
- **Mixed:** Present all recommender results together
- **Weighted:** Numerically combine the scores
- **Switching:** Switch between different recommendations according to user profile & context
- **Cascade:** Assign recommenders a priority, the low priority ones break ties between the higher ones
- **Feature Combination:** Combine features from different sources and input to a single recommender
- **Feature Augmentation:** One recommender generates features that form part of the input to another
- **Meta-level:** One recommendation technique is applied and produces a model, this is then the input to the next technique.

Numerical Combination of Scores

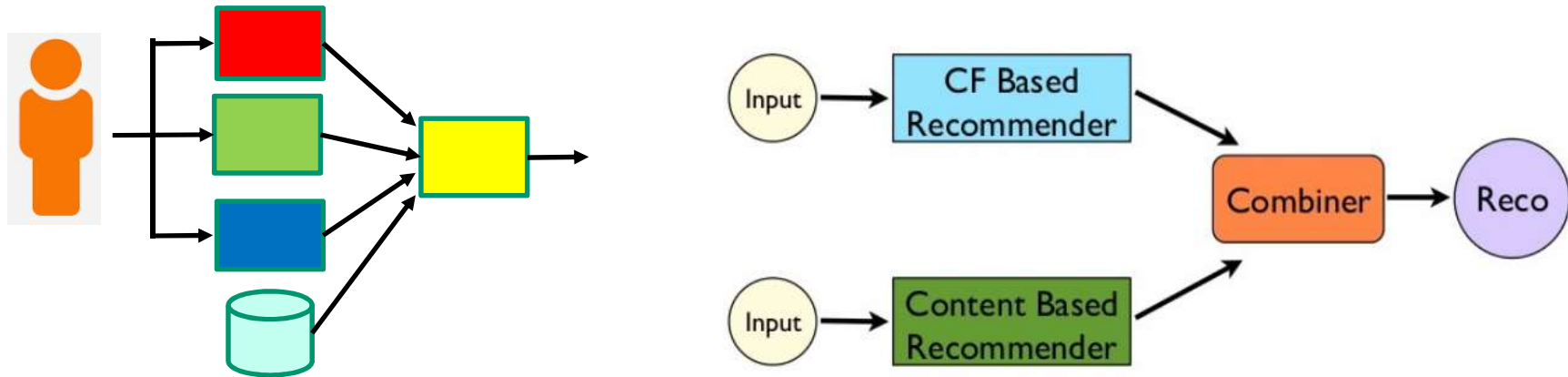
- If recommender outputs have the same range (e.g. predicted rating 1-5) then they can be numerically combined, e.g.
 - Pick the topN items with the highest predicted ratings, or
 - Average the predicted ratings for each item, pick the topN highest



- If the outputs have different ranges then can normalise all to be the same range



Weighted Combination



■ Feature Weighted Stacking

- Many different recommenders (models) are stacked up
- Another model is used to weight their votes
- Can select weights manually via experimentation
- Or, use a ML algorithm to learn the weights
 - ❖ E.g. linear regression
 - ❖ Training signal could be rating prediction error

Weighted Combination of Similarities

- Combining User Ratings with Content-Based Filtering (example)*

User Profile

~ characteristics of the items viewed (& liked)

<u>genre</u>	<u>synopsis</u>	<u>director</u>	<u>actress</u>	<u>actor</u>
(romance, scifi, action, comedy,..)	(word1, word2, word3,..)			
0.3 0.4 0.2 0.1	0.023 0.12 0.02			
Normalised counts of items viewed	TF-IDF's (across all viewed items)			

User Ratings

~ ratings given to viewed items

Cluster to get user cliques/groups (good for cold start)

$$\text{Sim}_{\text{userJ,userK}} = \text{sim}_{\text{userJ,userK}} * (1-c) + \text{sim}_{\text{groupJ,groupK}} * c$$

C ~ 0.3 was found to give best results

Collaborative Filtering

A new approach for combining content-based and collaborative filters

Byeong Man Kim, Qing Li, Chang Seok Park, Si Gwan Kim, Ju Yeon Kim

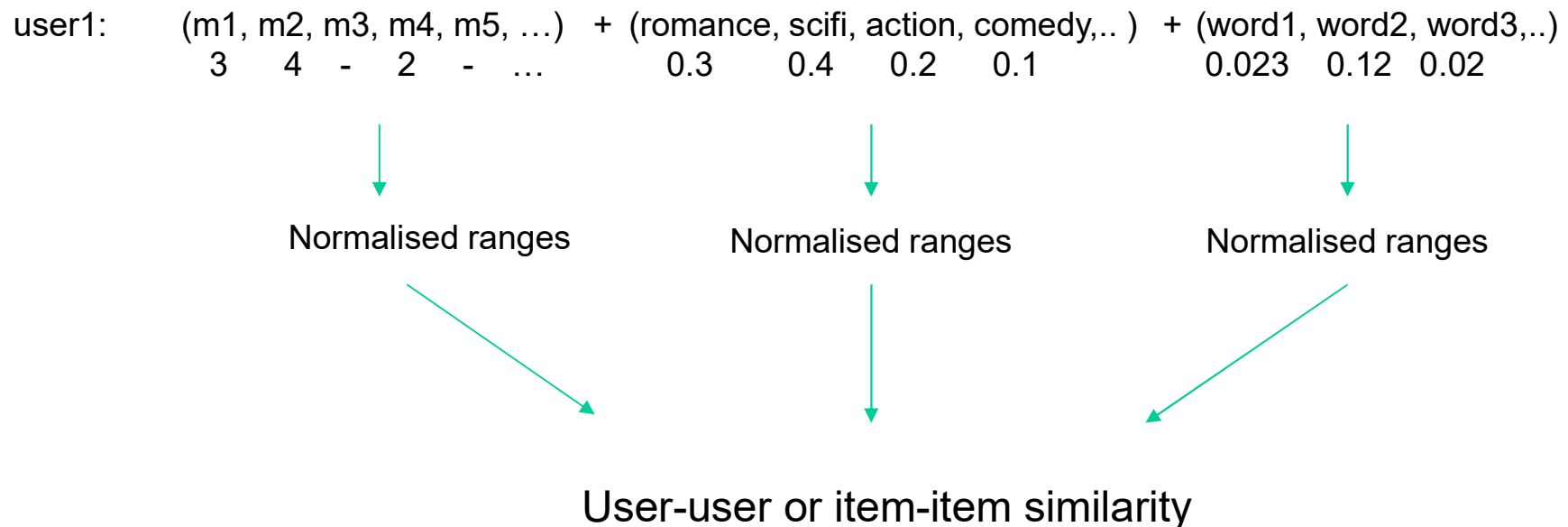
Journal of Intelligent Information Systems

July 2006, Volume 27, Issue 1, pp 79-91 | Cite as

*<https://link.springer.com/article/10.1007/s10844-006-8771-2>

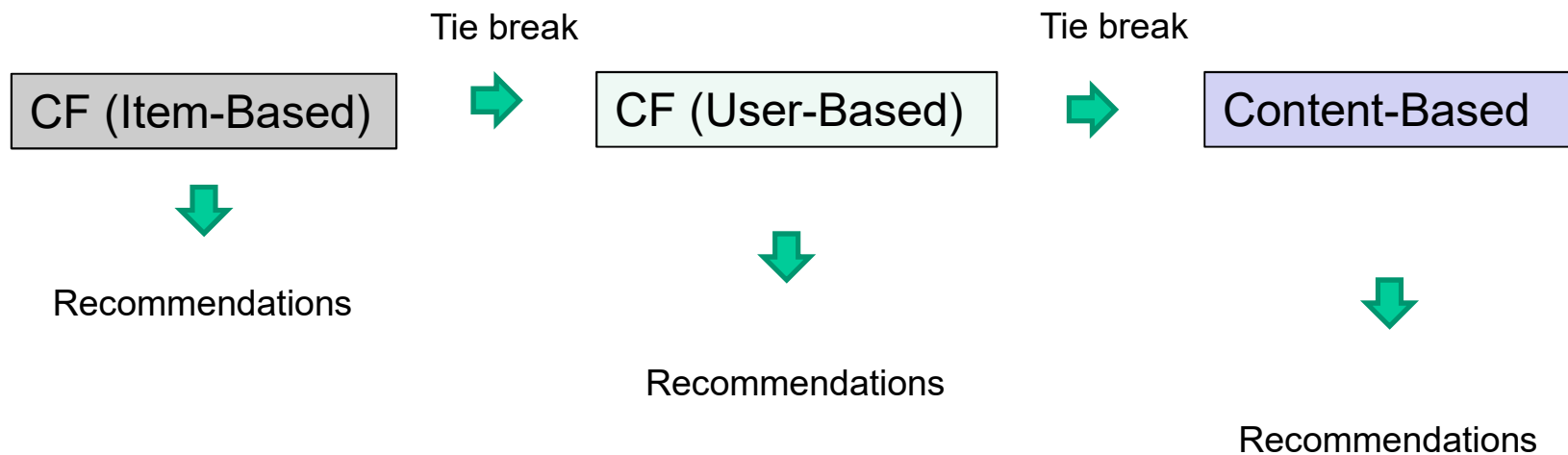
Feature Combination

- Combine the features together into one large user-record and input into a single recommender
- E.g.



Cascading Example

- Assign recommenders a priority, the low priority ones break ties between the higher ones
- E.g.

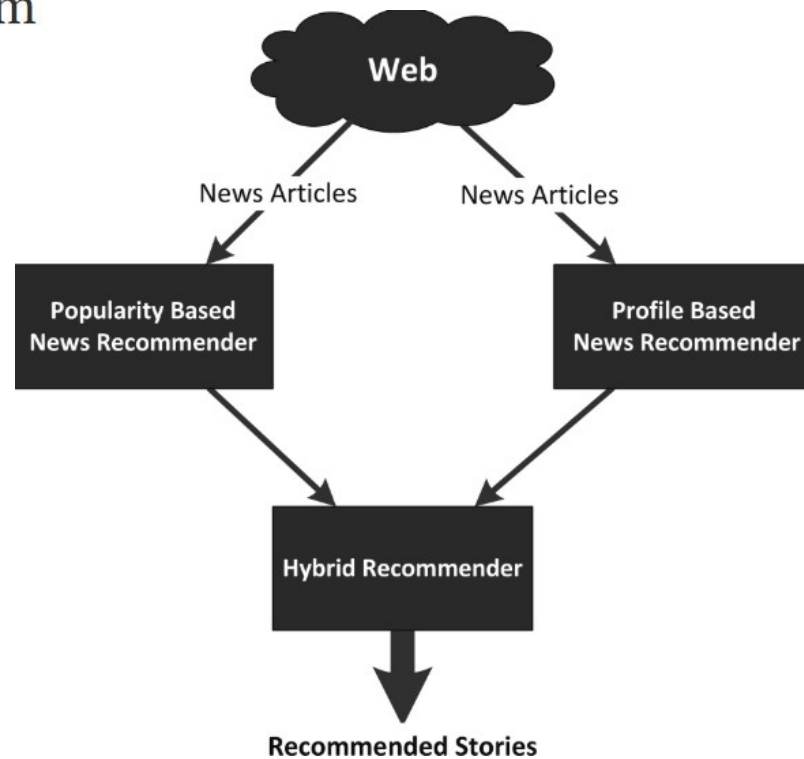


Example: News Recommendation (I)

Incorporating popularity in a personalized news recommender system

Nirmal Jonnalagedda, Susan Gauch, Kevin Labille, Sultan Alfarhood

<https://peerj.com/articles/cs-63/>



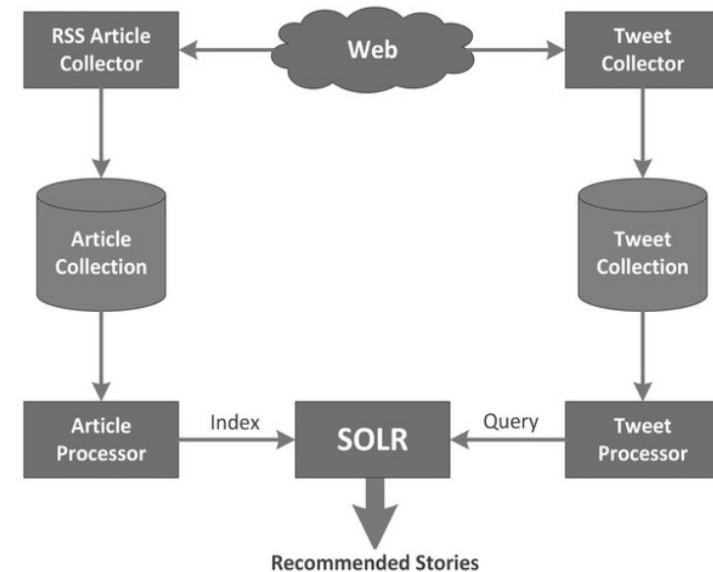
$$Hybrid_Wt_{ij} = \alpha * Popularity_Wt_j + (1 - \alpha) * Personal_Wt_{ij}$$

Example: News Recommendation (2)

Popularity-Based Recommender

Popularity score is based on #tweets that get mapped to an article over a time period.

$$Popularity_Wt_i = \sum_{t \in T} cosineSimilarity(Article_i, Tweet_t)$$



	Article B1	Article B2	Article B3	Article E4	Article S5	Article S6
Tweet 1	0.6					0.7
Tweet 2	0.3	0.1				
Tweet 3	0.5			0.9		
Tweet 4			0.5	0.4		
Tweet 5		0.2			0.2	
Tweet 6		0.1	0.1			
Tweet 7		0.1				0.3

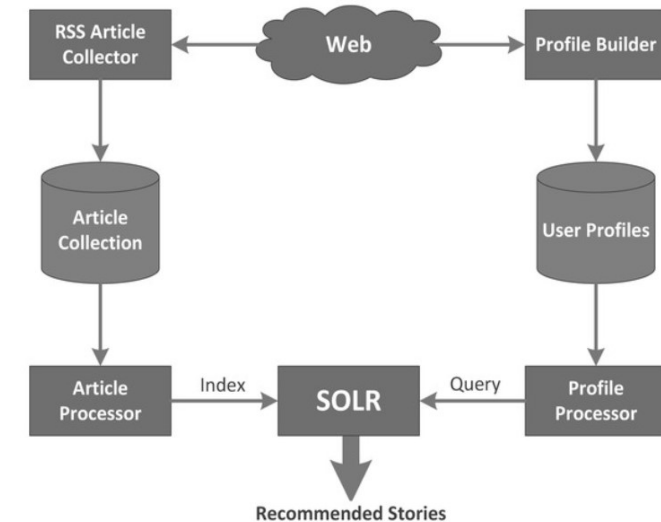


Article	Popularity_Wt
B1	1.4
E4	1.3
S6	1.0
B3	0.6
B2	0.5
S5	0.2

Example: News Recommendation (3)

Profile-based Recommender

Personal score is based on the similarity between users profiles (supplied by user by manually scoring articles) and the articles available.



$$Personal_Wt_{ij} = CosineSimilarity(ArticleProfile_i, UserProfile_j)$$

Category	Weight
Business	6
Entertainment	1
Sports	3

*

Articles	Business Wt	Entertainment Wt	Sports Wt
B1	0.3	0.2	0.0
B2	0.7	0.0	0.6
B3	0.4	0.7	0.0
E4	0.0	8.0	0.2
S5	0.6	0.1	0.0
S6	0.4	0.1	0.0



Article	Personal_Wt
B2	6.0
S5	3.7
B3	3.1
S6	2.5
B1	2.0
E4	1.4

Example: News Recommendation (4)

$$Hybrid_Wt_{ij} = \alpha * Popularity_Wt_j + (1 - \alpha) * Personal_Wt_{ij}$$

Article	Popularity_Wt	Normalized Popularity_Wt	Personal_Wt	Normalized Personal_Wt
B1	1.4	1.00	2	0.33
B2	0.5	0.36	6	1.00
B3	0.6	0.43	3.1	0.52
E4	1.3	0.93	1.4	0.23
S5	0.2	0.14	3.7	0.62
S6	1.0	0.71	2.5	0.42



Article	Hybrid_Wt
B2	0.36
B1	0.33
S6	0.30
B3	0.22
E4	0.22
S5	0.09

Example: Spotify

Combines 3 different Approaches:

Collaborative Filtering
using implicit feedback,
times played etc

+

Songs & Artist Models
uses web crawling to
get sentiment & buzz
about songs and artist

+

Audio Modelling

use CNN to convert raw audio into a feature set (tempo, liveliness, danceability etc).
Then match to users past listens

For each user, there are two listening histories we take into consideration: the set of all tracks a user listened to and the set of all artists a user listened to. Thus, we are able to compute a *artist similarity* (*artistSim*) and a *track similarity* (*trackSim*) as shown in Equations 2 and 3.

$$artistSim_{i,j} = \frac{|artists_i \cap artists_j|}{|artists_i \cup artists_j|} \quad (2)$$

$$trackSim_{i,j} = \frac{|tracks_i \cap tracks_j|}{|tracks_i \cup tracks_j|} \quad (3)$$

The final user similarity is computed using a weighted average of both, the *artistSim* and *trackSim* as depicted in Equation 4.

$$sim_{i,j} = w_a * artistSim_{i,j} + w_t * trackSim_{i,j} \quad (4)$$

http://ceur-ws.org/Vol-1313/paper_7.pdf

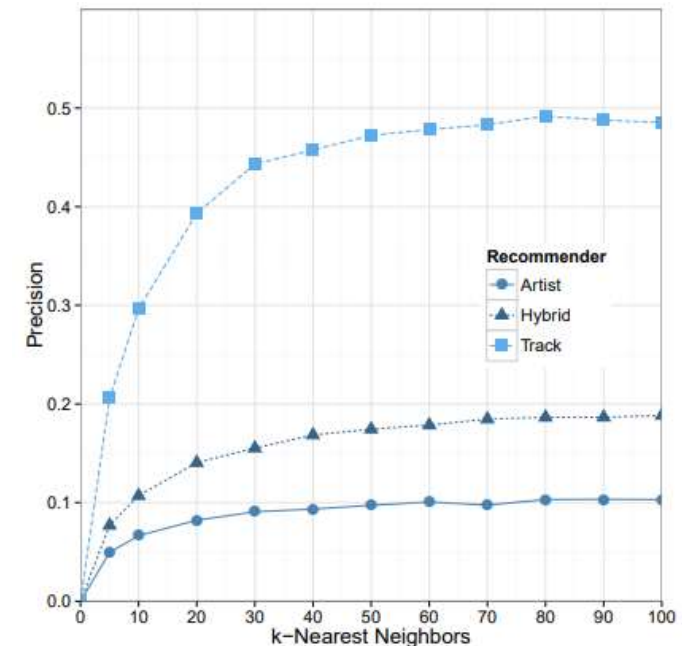
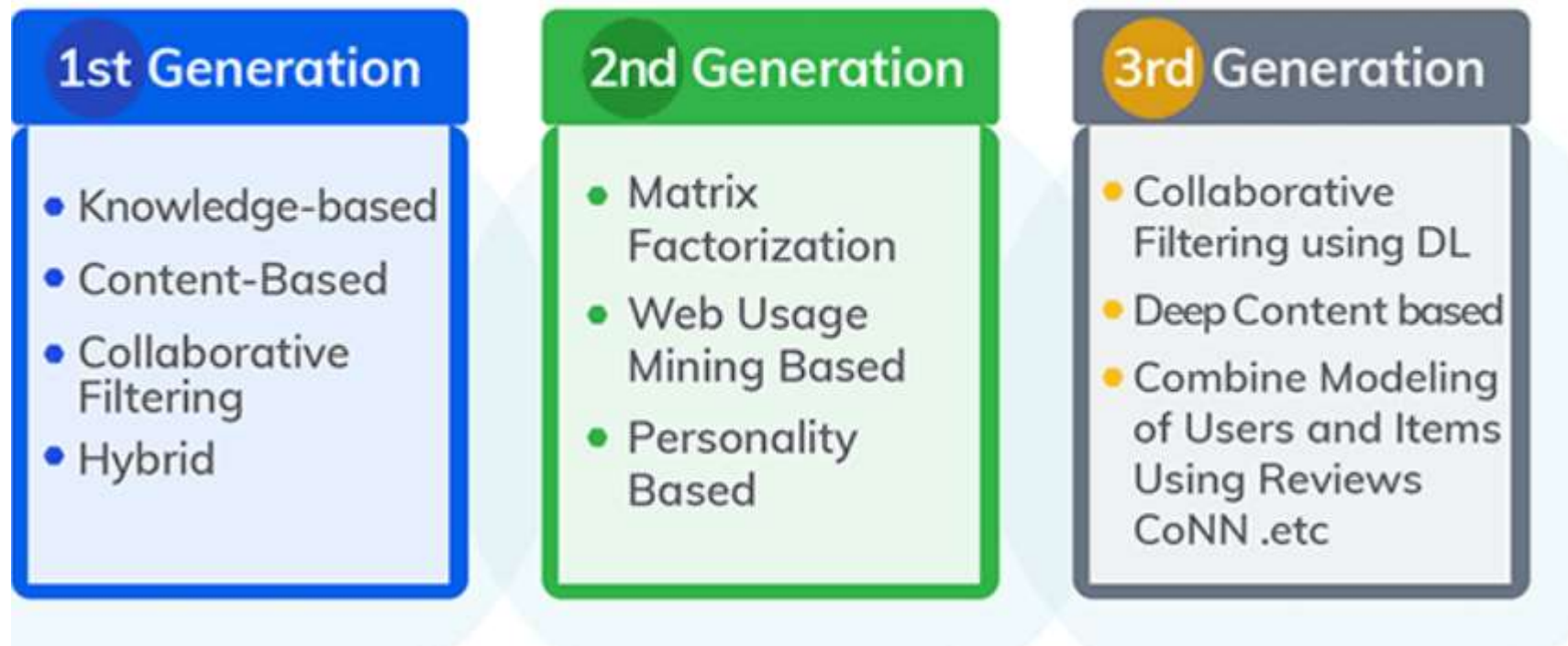


Figure 3: Precision and Recall of the Track-Based Recommender

<https://medium.com/s/story/spotify-s-discover-weekly-how-machine-learning-finds-your-new-music-19a41ab76efe>

Current Trends

Recommendation System Generations

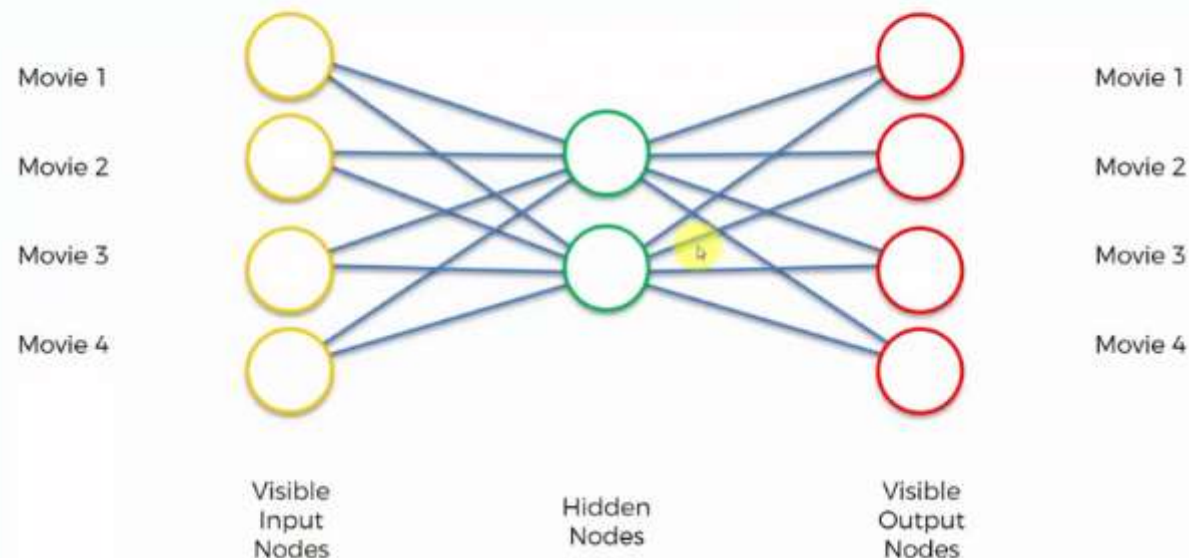


<https://www.xenonstack.com/blog/recommender-systems/>

Deep Learning & Collaborative Filtering

- Auto-encoders offer one way to learn how to predict ratings for unseen items
- The hidden (compressed) layer can be thought of as analogous to the latent features in MF approaches

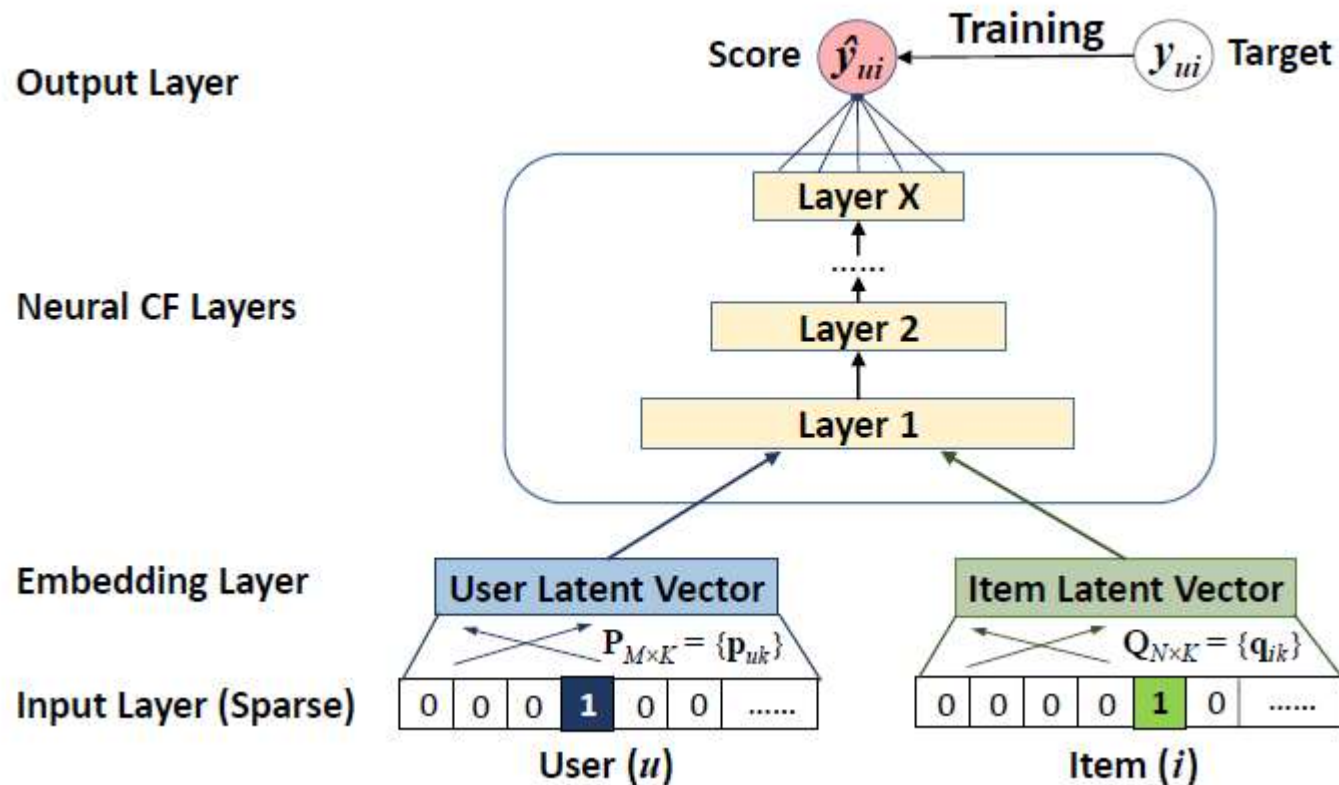
Auto Encoders



(Compressed layer)

Deep Learning & Collaborative Filtering

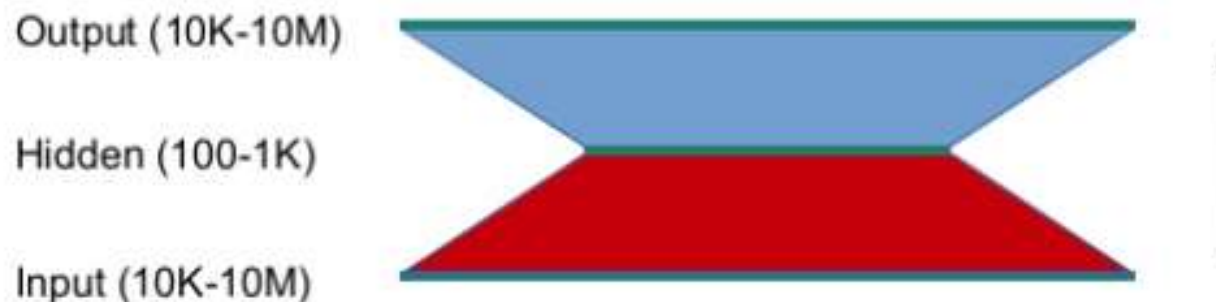
- In traditional matrix factorisation. Ratings are predicted by taking the inner (dot) product of the user latent features and the item latent features
- In the NCF* system, the inner product is replaced with a neural architecture that can learn an arbitrary function from data



*Neural Collaborative Filtering, see <https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf>

Recommendation Systems at Scale

- Amazon DSSTNE ~ Deep Scalable Sparse Tensor Neural Engine
- Based on Auto-Encoder neural network architecture



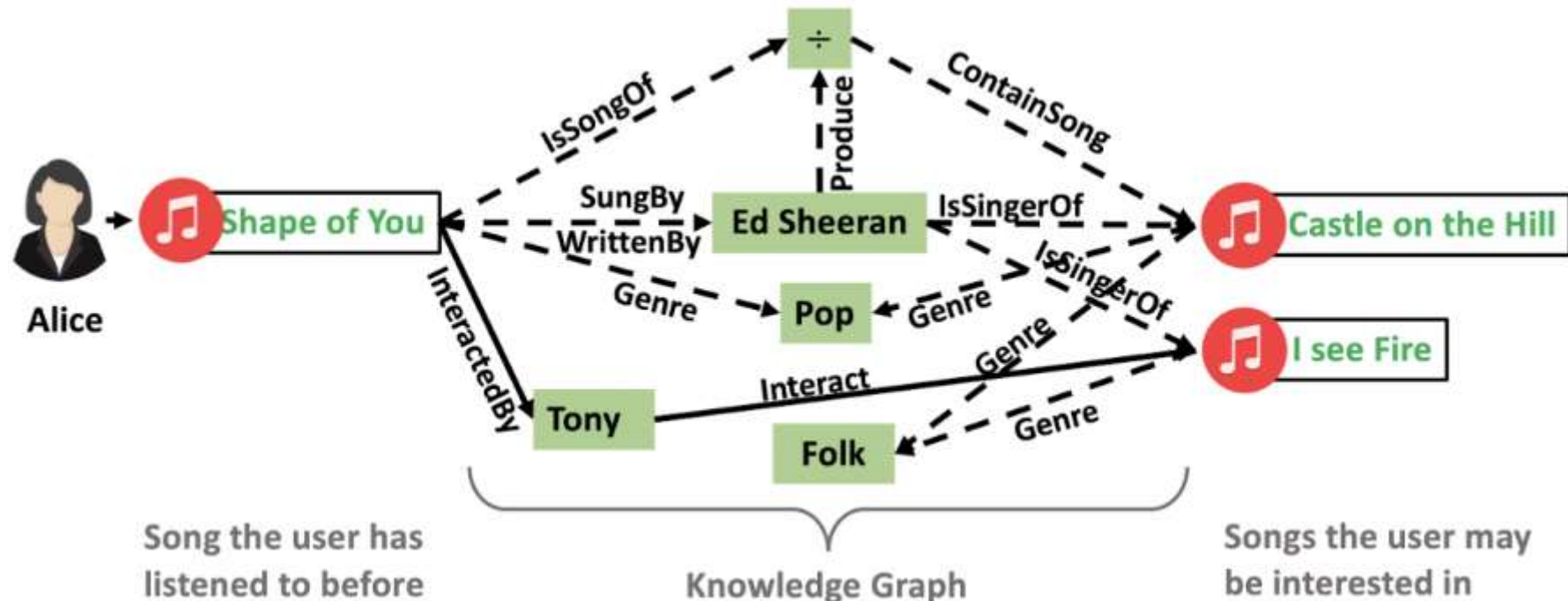
Our models often have hundreds of thousands of nodes in the input and output layers. At this scale, we can easily reach trillions of weights for a fully-connected network, even if it is shallow. Therefore, our models often do not fit the memory of a single GPU

In **model parallel training**, the model is distributed across N GPUs – the dataset (e.g., RDD) is replicated to all GPU nodes. Contrast this with *data parallel* training where each GPU only trains on a subset of the data, then shares the weights with each other using synchronization techniques such as a parameter server.

<https://aws.amazon.com/blogs/big-data/generating-recommendations-at-amazon-scale-with-apache-spark-and-amazon-dsstne/>

Recommendation as a Graph Problem

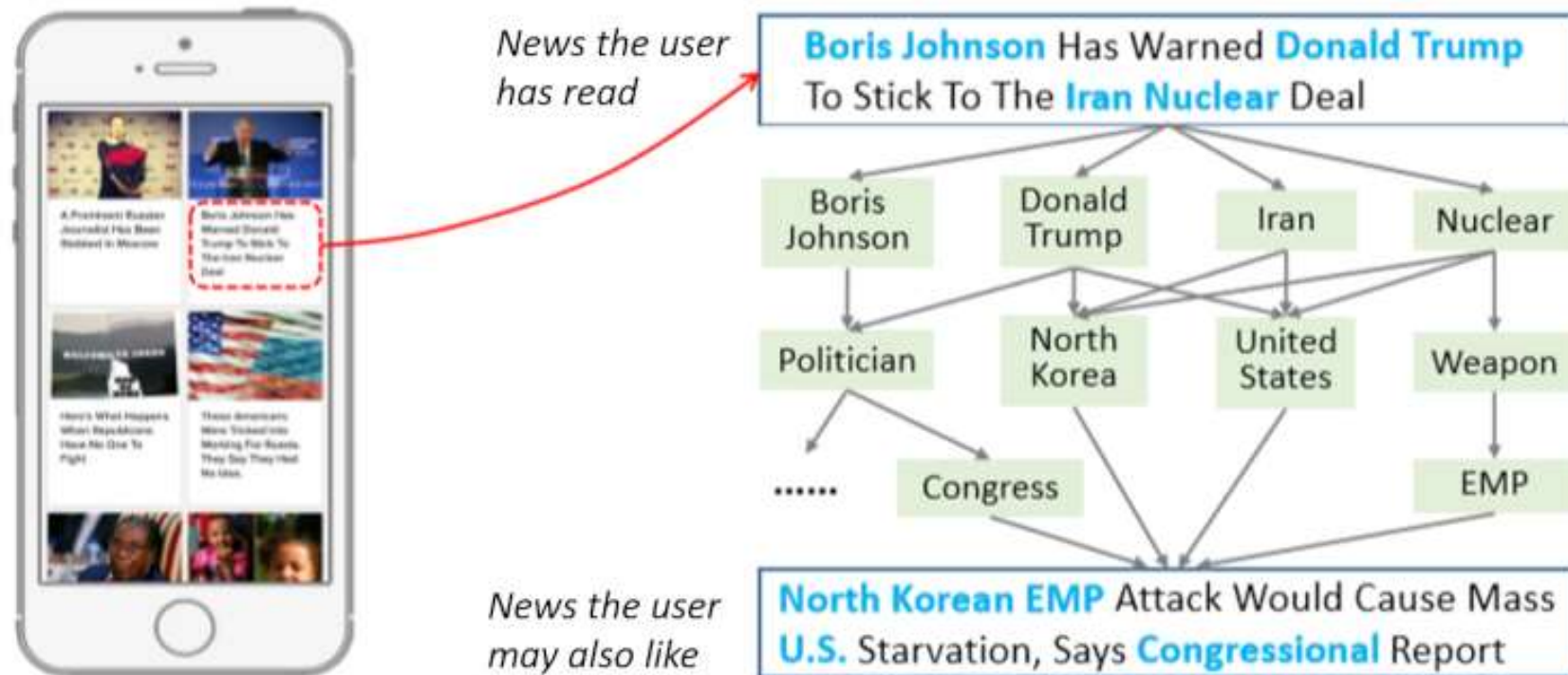
- By exploring the interlinks within a knowledge graph, the connectivity between users and items can be discovered as paths, which provide rich and complementary information to user-item interactions.



<https://tech.ebayinc.com/research/explainable-reasoning-over-knowledge-graphs-for-recommendation/>

Recommendation as a Graph Problem

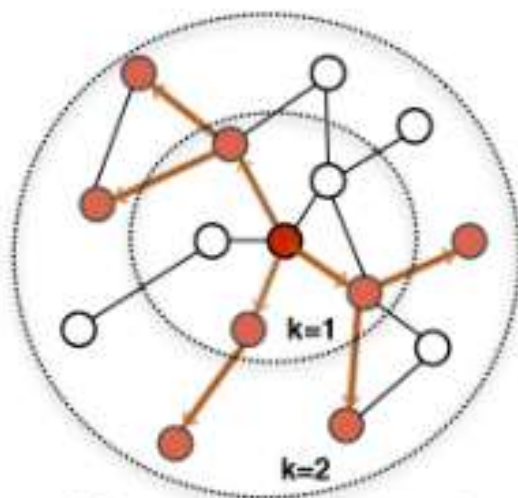
- The knowledge graph greatly expands the amount of information of each item and strengthens the connection between them, providing abundant reference values for a recommendation engine, which leads to additional diversity and explainability of the recommendation result



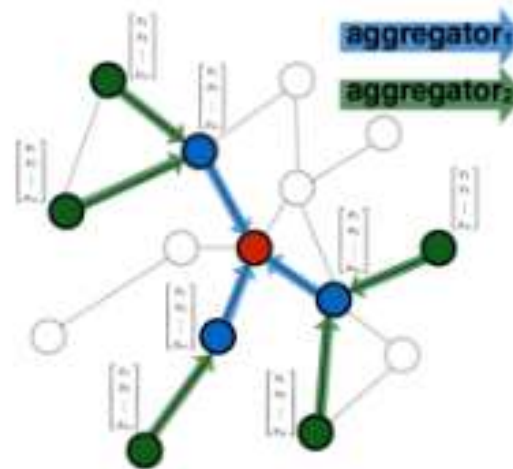
<https://www.microsoft.com/en-us/research/lab/microsoft-research-asia/articles/personalized-recommendation-systems/>

DL and Graph-Learning

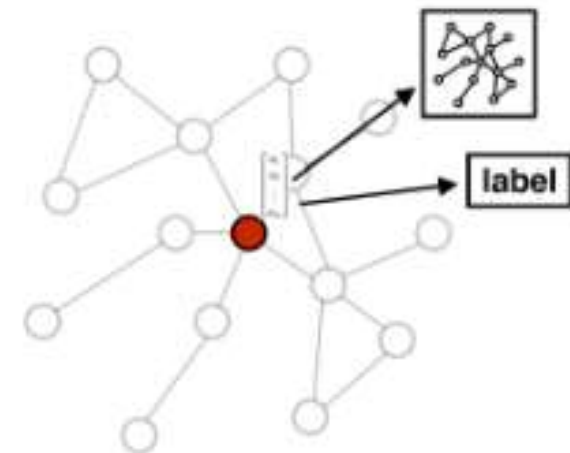
- Compared to a social network, a knowledge graph is a heterogeneous network; therefore, more sophisticated recommendation algorithms are required. In recent years, network representation learning has become one of the most popular research areas for addressing this.
- E.g. GraphSAGE is an inductive representation learning algorithm that is especially useful for graphs that grow over time.



1. Sample neighborhood



2. Aggregate feature information from neighbors



3. Predict graph context and label using aggregated information

Challenges and Issues – User Experience

The success of a recommender systems depends on more than just accuracy:

- **Diversity** - users tend to be more satisfied with recommendations when there is a higher diversity. There is unlikely to be a single best recommendation,
- **Serendipity** - how surprising are the recommendations?
- **Avoiding bad recommendations** - assigning a cost to them
- **Long versus short term recommendations.** How far ahead should the recommendation be?
- **Repeat Recommendations** - how often should we repeat a recommendation? Users may ignore a recommendation first time but still like the item, e.g. if short of time
- **Handling Sequences** – sometimes the sequence is important e.g. a compilation of musical tracks from slow to fast, episodes in Game of Thrones etc.

Challenges and Issues – Privacy

- **Privacy** - push-back by users if they feel the RS is collecting too much information about them.
 - The Netflix Challenge data was anonymised but in 2007 two researchers from the University of Texas were able to identify individual users by matching the data sets with film ratings on the Internet Movie Database.
 - As a result, in December 2009, an anonymous Netflix user sued Netflix in Doe v. Netflix.
 - This led in part to the cancellation of a second Netflix Prize competition in 2010.

RYAN SINGEL 03.12.18 02:48 PM

NetFlix Cancels Recommendation Contest After Privacy Lawsuit



Challenges and Issues - Trust

- Trust can be built by explaining how the recommendations are generated and why an item is being recommended
- Fake reviews and fake ratings erode trust



Yelp's fake review problem

by Daniel Roberts

@readDanwrite

SEPTEMBER 26, 2013, 3:05 PM EST

A New York sting operation caught businesses paying for positive ratings on recommendation websites.



Amazon shoppers misled by 'bundled' star-ratings and reviews

Guardian study finds inferior items appear highly praised, making ratings worthless

Fri 5 Apr 2019 06.00 BST



▲ The research found badly translated or updated Kindle versions of some literary classics appeared to have high ratings. Photograph: Alamy

Analysis by the Guardian shows products that have actually been given one-star ratings appear alongside rave reviews of better quality items, making it impossible for consumers to judge the true value of what they are about to buy.

Challenges and Issues - Computability

- **Scalability** – handling very large numbers of users and items
- **Cold-start** – making recommendations to a new user
- **Sparsity** - recommending items in **the long-tail** is hard since there are very few ratings / purchases for them

ARTICLE INTERNET, MEDIA

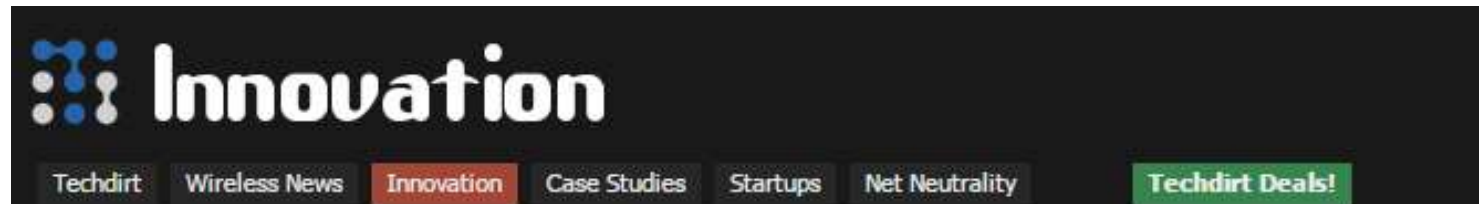
Do recommendation systems make the 'tail' longer or shorter?

By [Paul Belleflamme](#) 26 April 2012 39

(Updated March 2015)

<http://www.ipdigit.eu/2012/04/do-recommendation-systems-make-the-tail-longer-or-shorter/>

Challenges and Issues - Computability



Why Netflix Never Implemented The Algorithm That Won The Netflix \$1 Million Challenge

from the *times-change* dept

Innovation
by Mike Masnick
Fri, Apr 13th 2012
12:07am

- Despite all the plaudits and case studies, Netflix announced this week that despite paying \$1 million dollars to a winning team of multinational researchers in 2009, they never bothered to implement their solution.
- Why? Because, according to Netflix the “additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment.”

<https://medium.com/netflix-techblog/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429>