

## Graduate Certificate in Intelligent Reasoning Systems

# Reasoning and Knowledge Discovery from (large) Datasets

Model-based and Hybrid  
Recommender Systems

Dr. Barry Shepherd  
Institute of Systems Science  
National University of Singapore  
Email: [barryshepherd@nus.edu.sg](mailto:barryshepherd@nus.edu.sg)



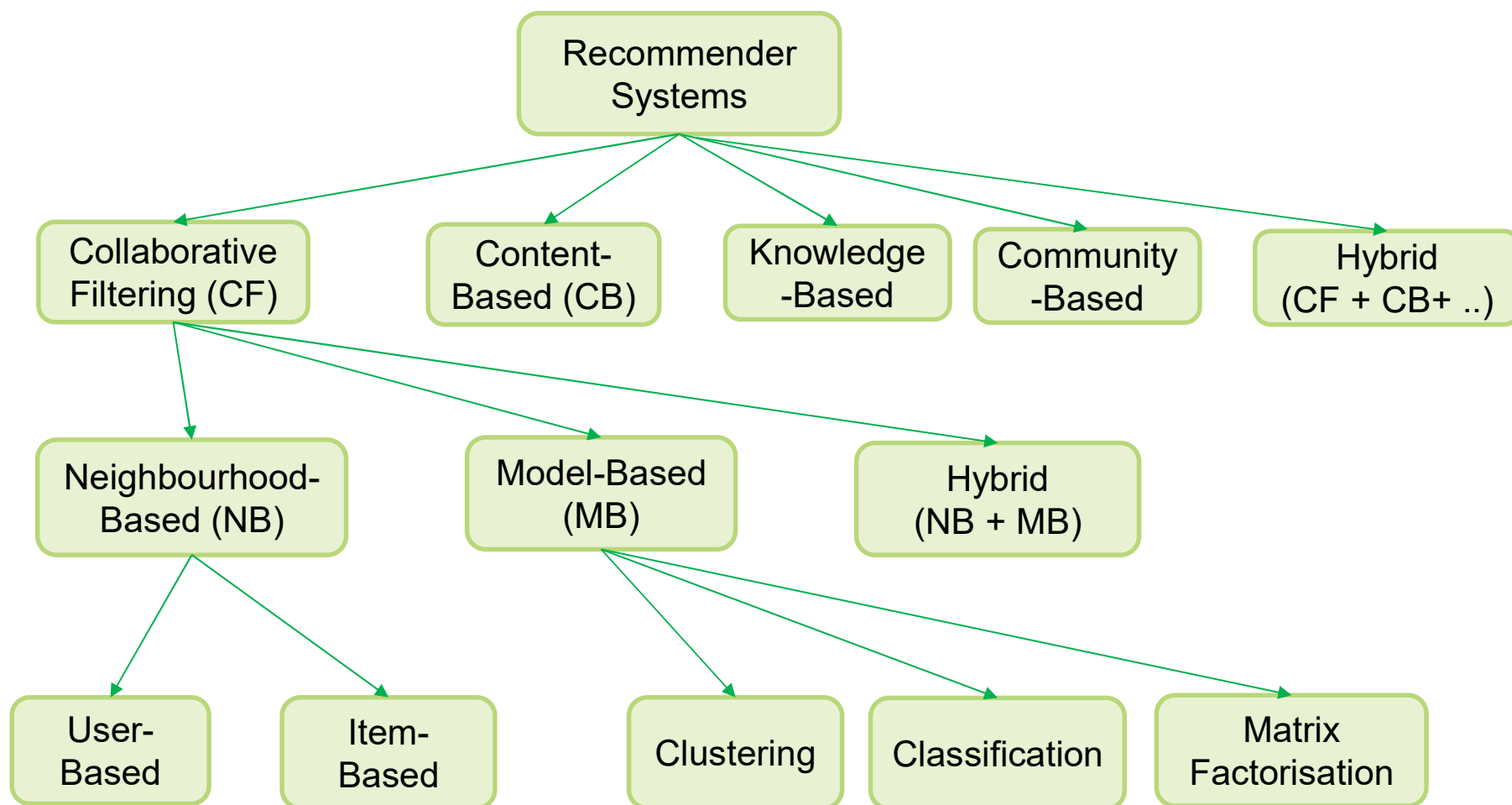
© 2021 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.

# Agenda



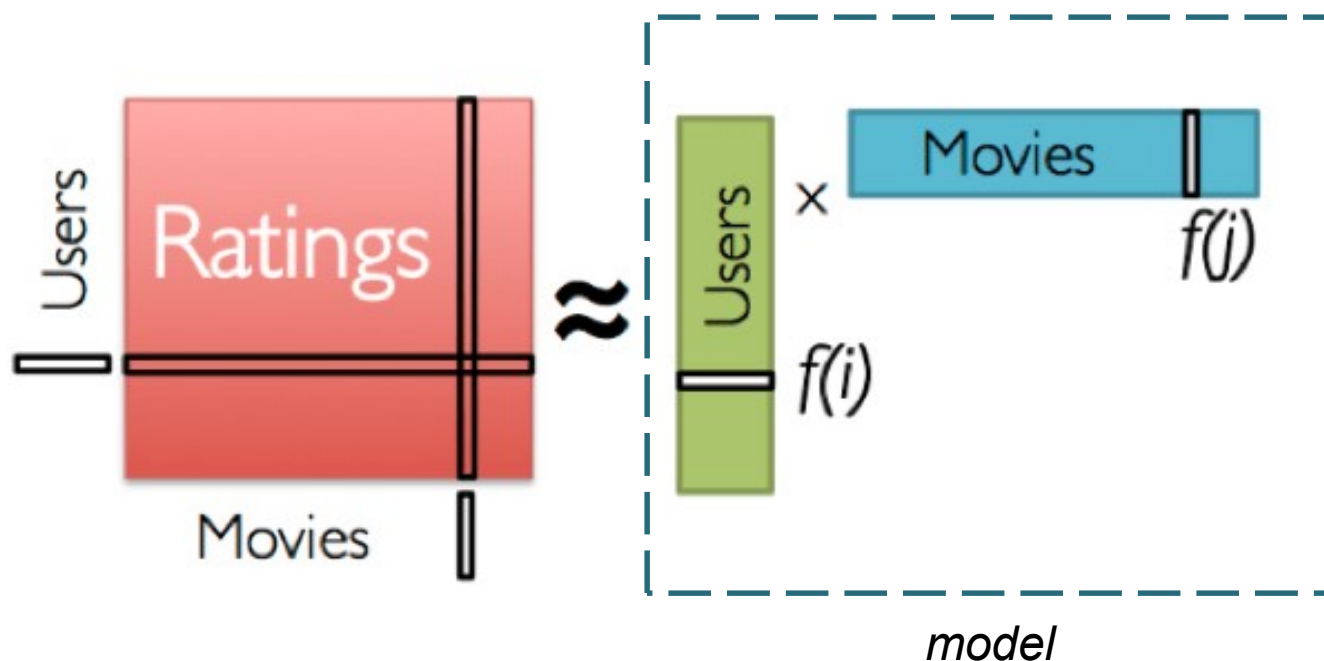
Day	Topic
Day4 (am)	Introduction – basic concepts overview
Day4 (am)	Market Basket Analysis and Recommender Systems
Day4 (pm)	Similarity-Based Recommender Systems
Day5 (am)	Model-Based Recommender Systems
Day5 (pm)	Hybrid and Advanced Recommender Systems
Day5 (pm)	Course Assessment Quiz

# Recommender System Approaches



# Recommending using Matrix Factorisation

- Matrix factorisation can be applied to the ratings matrix
- Converts it into 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  

$$= 2*8 + 7*0 + 5*10 + \dots \quad (\text{to normalise divide by the max score})$$

	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

$$\begin{array}{ccc}
 \begin{array}{c} \text{User ratings} \\ \left[ \begin{array}{ccc} 1 & \dots & M \\ \vdots & & \\ U & & \end{array} \right] \\ (500K * 17K = 8,500M) \end{array} & = & \begin{array}{c} \text{User preferences} \\ \left[ \begin{array}{ccc} 1 & \dots & K \\ \vdots & & \\ U & & \end{array} \right] \\ (500K * 100 = 50M) \end{array} * \begin{array}{c} \text{Movies properties} \\ \left[ \begin{array}{ccc} 1 & \dots & M \\ \vdots & & \\ K & & \end{array} \right] \\ (17K * 100 = 1.7M) \\ \text{model} \end{array}
 \end{array}$$

- 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 Example

- Once 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 well with sparse data. 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

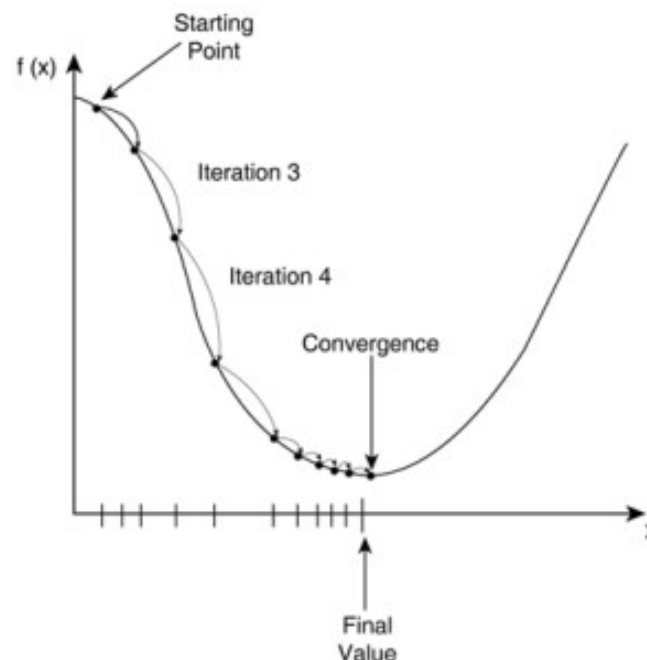
# Factorising using Gradient Descent\*

## 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:

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

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

$u_{ki}$  = value of  $k$ th preference for user  $i$   
 $m_{kj}$  = value of  $k$ th property for item  $j$   
 $r_{ij}$  = rating of user  $i$  on item  $j$   
 $p_{ij}$  = predicted rating of user  $i$  on item  $j$   
 $\gamma$  = the learning rate  
 $\lambda$  = the regularisation constraint

\*Method by Simon Funk (Netflix competitor)

# Overfitting & Regularisation

- Given a high enough degree, a polynomial (or any 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 encourages the coefficients to be small so that the model doesn't become over-complex

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

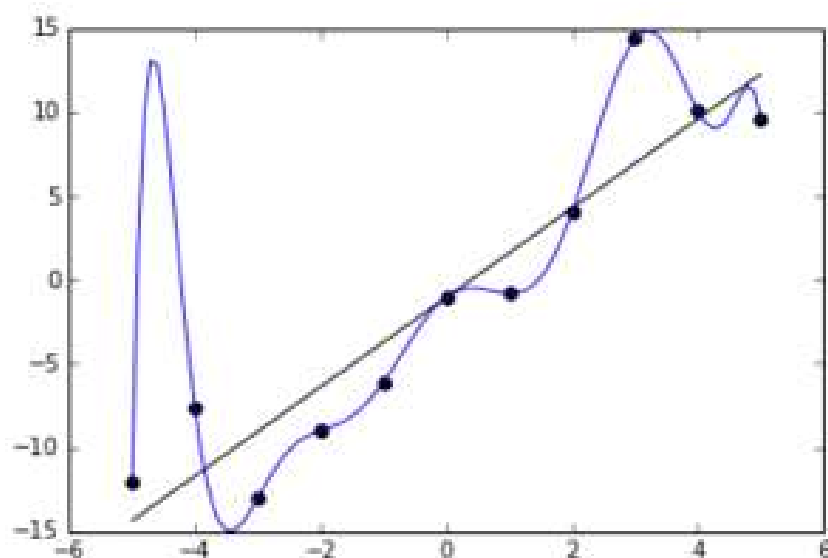
$$\text{where } h_{\theta}x_i = \theta_0 + \theta_1x_1 + \theta_2x_2^2 + \theta_3x_3^3 + \theta_4x_4^4$$

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



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

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



*We want the model to represent the training data, not to reproduce the training data!*

For MF, keeping the coefficients small means keeping the values in the factorized matrix small – this will be more crucial if the number of latent features is set as a large number (more potential to overfit)

The optimal number of latent features is usually determined by trial-and-error using a test data set, but starting with a rule-of-thumb initial value (e.g. 50→100)

# Alternating Least Squares Algorithm

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

$$R = AB^T$$

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

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

- If we fix  $B$  and optimize for  $A$  alone, the problem is reduced to the problem of linear regression:

$$y = \beta X \quad \} \quad y = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots$$

- The matrix  $B$  corresponds to the training data (the input variables  $X$ ), the matrix  $R$  corresponds to the output variables ( $y$ ), and  $A$  corresponds to the coefficients  $\beta$

## 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

- In linear regression, we solve for the coefficients  $\beta$  by minimizing the squared error:  $\|y - \beta X\|^2$
- The solution is given by the Ordinary Least Squares formula:  $\beta = (X^T X)^{-1} X^T y$

<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 can 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)...

$$R = A * B^T$$

	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	
u7		2	4			1		

	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
u7	?	?	?	?

	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

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

Matrix Factorisation at scale, e.g. Facebook system

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

# Other Popular Factorisation Methods

- Non-Negative Matrix Factorisation
  - [https://en.wikipedia.org/wiki/Non-negative\\_matrix\\_factorization](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](https://www.researchgate.net/publication/220788980_Large-Scale_Parallel_Collaborative_Filtering_for_the_Netflix_Prize)

# Workshop3:

## Introduction to Surprise Library (3A)

## Matrix Factorisation (3B)



# Working with Implicit User Feedback

- Issues with Explicit Ratings
  - Very sparse - users tend to be lazy, often don't bother to rate
  - Users may not always tell the truth? E.g. Influenced by peer/friend opinions (all of my friends liked that movie so maybe it was good after all!)
  - May not be available at all if no system in place to collect them
  - Watching what the user actually does (e.g. what they view or buy) may be more reliable / accurate than ratings

## Examples of Implicit Ratings

- Buying a product
- Viewing a (product) page
- Clicking on a link
- Time spend looking at a page
- Repeat visits
- Referring a page to others

## Issues with Implicit Ratings

- Buying something doesn't always mean liking something
- Could be a mistake buy or impulse buy or mistaken page-click etc.
- I may be buying for someone else using my own account



# Working with Implicit User Feedback

- Example: Count the number of visits to a product webpage or the time spent on the page and normalize to (say) 1  $\rightarrow$  5

User (or Session)	page1	page2	page3	page4	page5	page6
1	2	5	4		3	1
2		3		5	3	1
3			5	3		

*Implicit ratings matrix*

- This looks similar to the explicit ratings matrix. So can we proceed as before?
  - We can try – evaluate by testing
  - But, since all users who visit a product page must like the product to some degree (else they would not have visited?), if we predict the implicit rating we are predicting the amount of like and not predicting like versus dislike



# Implicit ratings are often treated as Binary

- Assume ANY page view (or similar) is a like, else don't know

User (or Session)	page1	page2	page3	page4	page5	page6
1	2	5	4		3	1
2		3		5	3	1
3			5	3		



User (or Session)	page1	page2	page3	page4	page5	page6
1	1	1	1		1	1
2		1		1	1	1
3			1	1		

- But now we can't use Cosine Similarity (or similar) since result will always be 1

$$\text{Cosine Similarity} = (1*1 + 1*1 + \dots) / \sqrt{(1^2+1^2 + \dots)*(1^2+1^2 + \dots)} = N/N = 1$$

$$\text{Euclidean Similarity} = 1 / (1 + \sqrt{(1-1)^2 + (1-1)^2 + \dots}) = 1/1 = 1$$

- To apply Cosine or Euclidean we must assume the NA's => 0 (don't like)
  - Can often work well, e.g. recommenderLab (R library) makes this assumption

# Binary Ratings: Jaccard Similarity

- Measures the similarity between two sets
- Makes no assumptions about the missing values

$$\text{Sim}_{\text{jaccard}}(A,B) = |A \cap B| / |A \cup B|$$

User	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
U1	1	1	1				1			
U2		1			1	1	1	1		

What is the Jaccard similarity for (u1,u2)?

# ALS with Implicit Feedback

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

- Recall that an implicit like = viewing a product page, clicking an item, purchase etc.
- BUT... there is no dislike signal (unlike explicit ratings)

*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 to each preference:  
 $C = 1 + \alpha \cdot \text{rating}$  ( $\alpha \sim 40$ )  
 e.g. more views => 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

\*Note that the preference matrix has no missing values, so optimisation process is different and more time consuming.



# Workshop4: Introduction to the Implicit Library



Fast Python Collaborative Filtering for Implicit Datasets.

# Agenda



Day	Topic
Day4 (am)	Introduction – basic concepts overview
Day4 (am)	Market Basket Analysis and Recommender Systems
Day4 (pm)	Similarity-Based Recommender Systems
Day5 (am)	Model-Based Recommender Systems
Day5 (pm)	Hybrid Recommender Systems
Day5 (pm)	Advanced Methods Overview
Day5 (pm)	Course Assessment Quiz

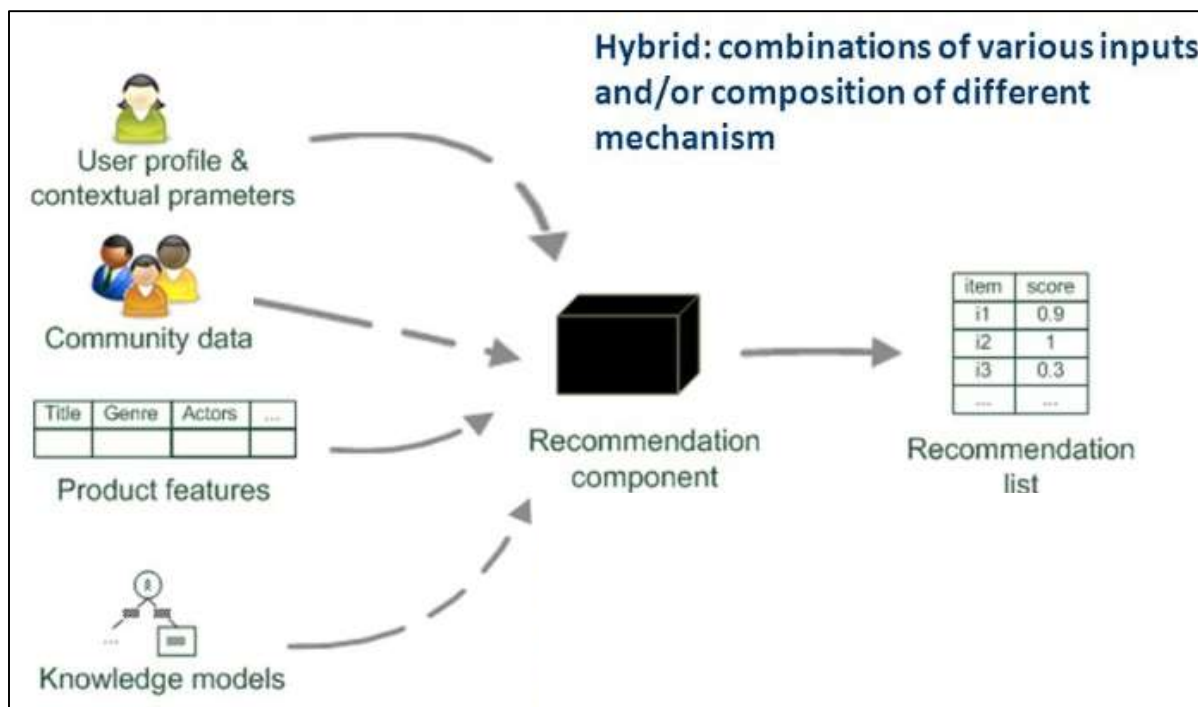
# Hybrid Recommender Systems

- 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

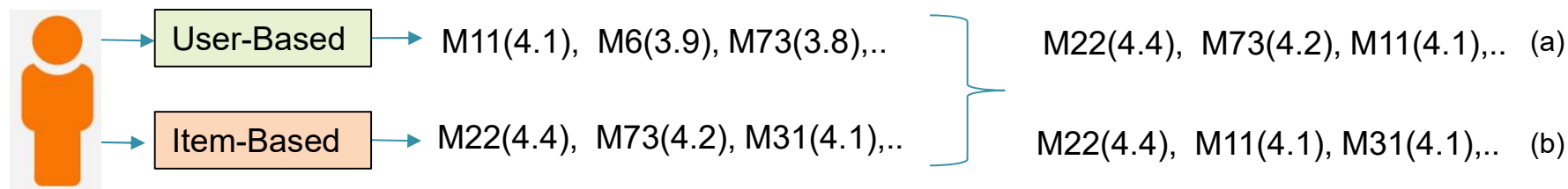


- **Mixed:** Present all recommender results together
- **Weighted:** Numerically combine the scores
- **Switching:** Switch between different recommendations according to user profile & context
- **Cascade:** Assign recommenders a priority, the low priority ones break ties between the higher ones
- **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.

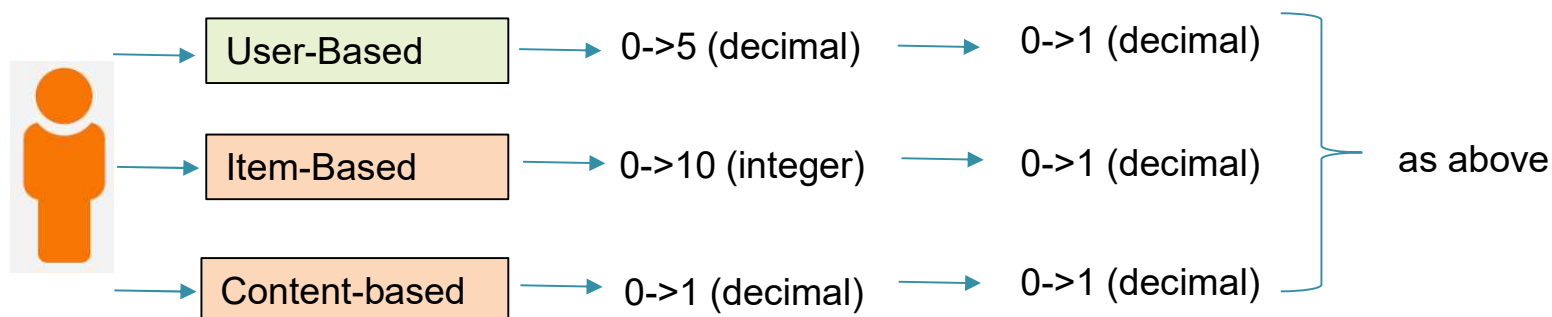
Hybridization techniques from Wikipedia

# 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

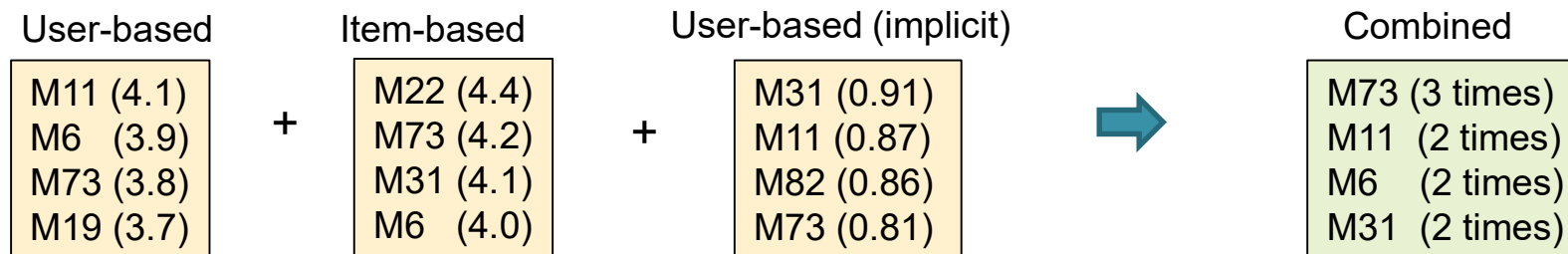


- If the outputs have different ranges then can normalise all to be the same range



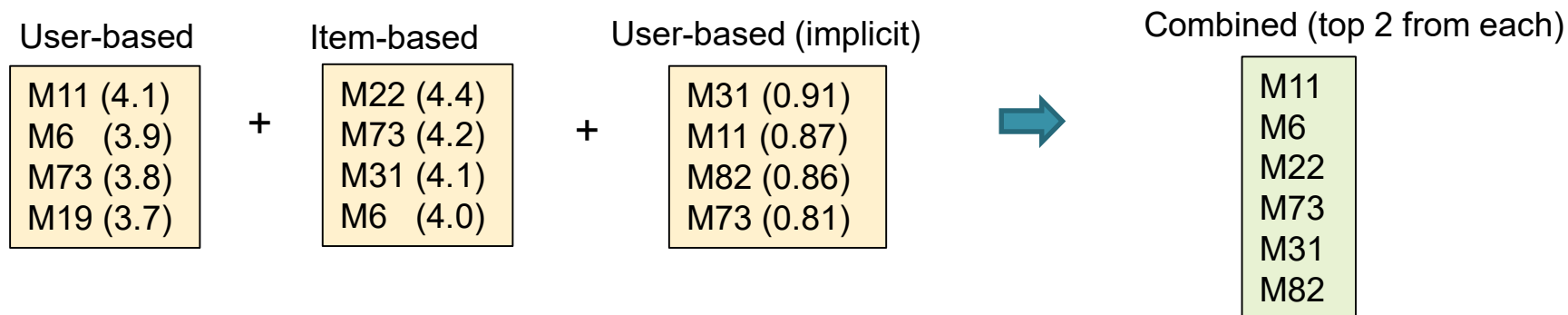
# Other Combination Methods

- Pick the most frequent items in the topN , with tie break:

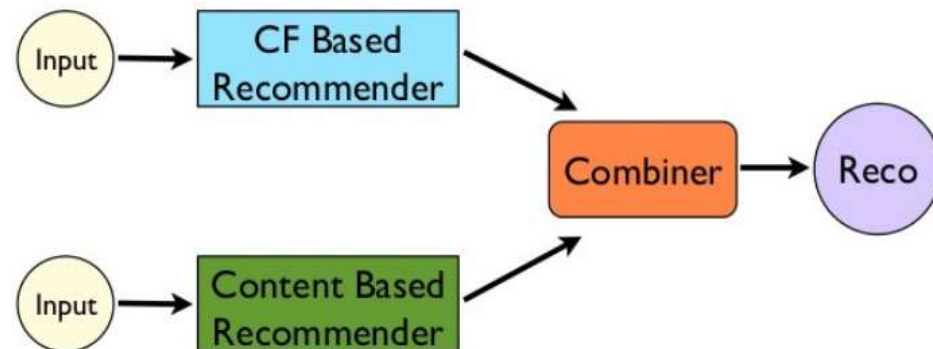
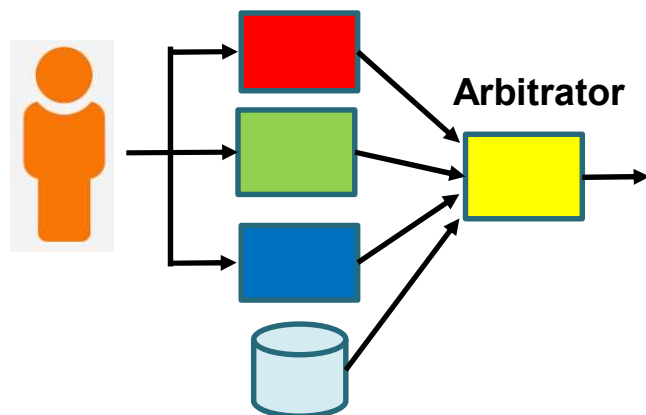


- For small N and many items its possible (likely) than the top items are all different hence one solution is to make N larger and then select the best K items ( $K < N$ )

- Or, pick a selection from the topN of each recommender:



# 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

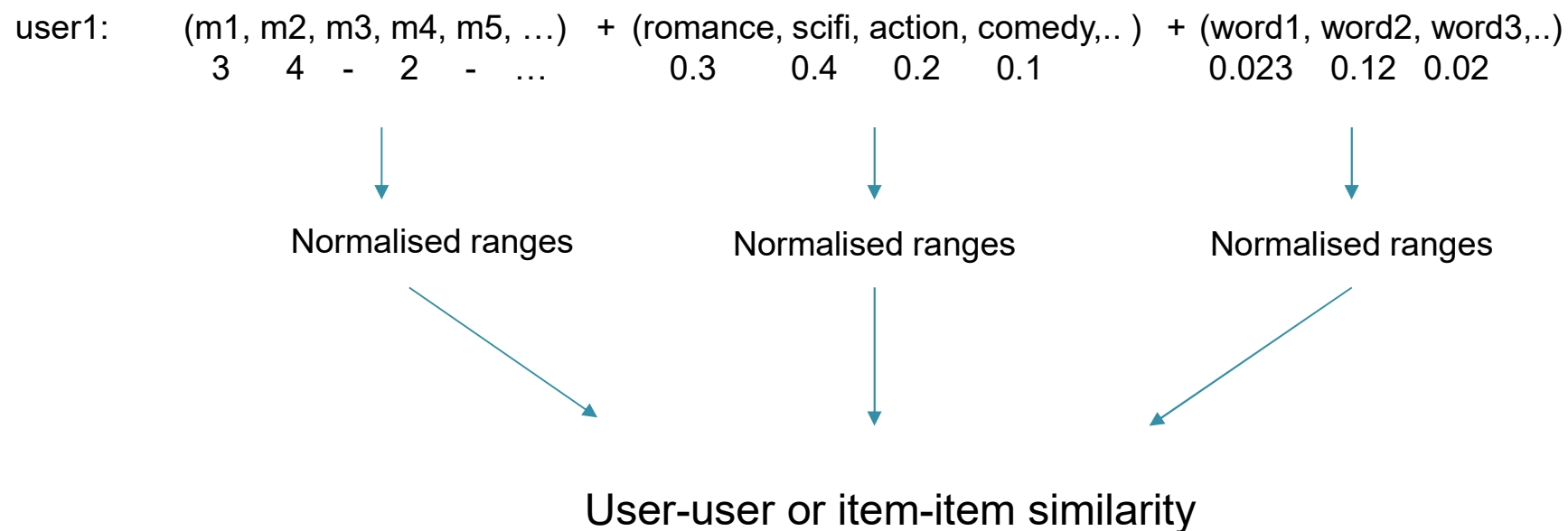
July 2006, Volume 27, Issue 1, pp 79-91 | Cite as

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



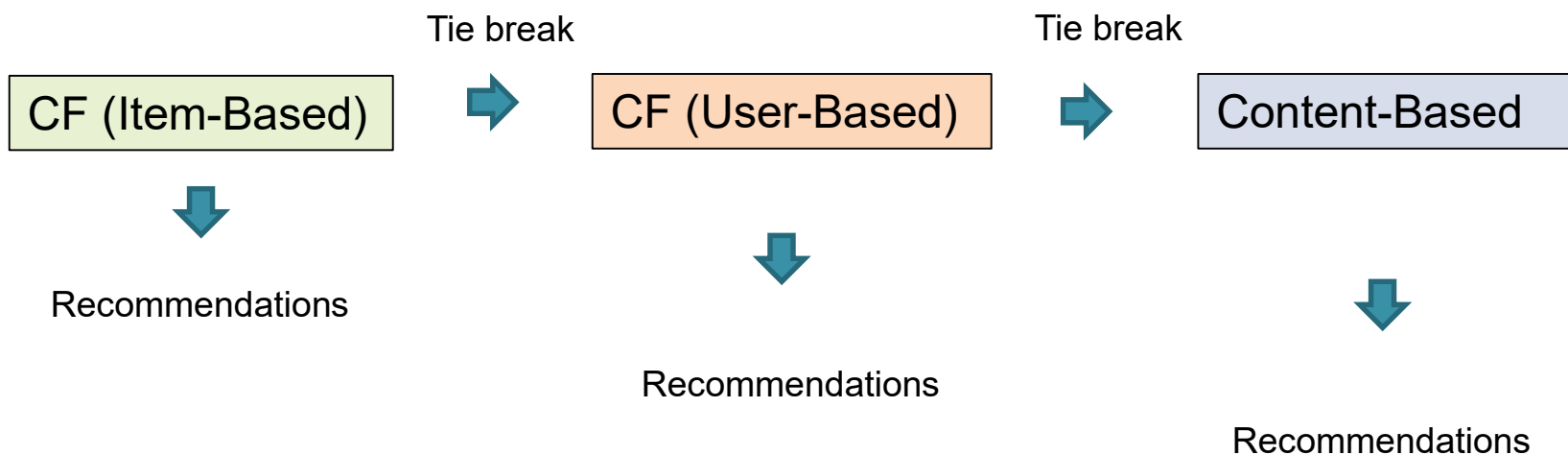
# Feature Combination

- 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.

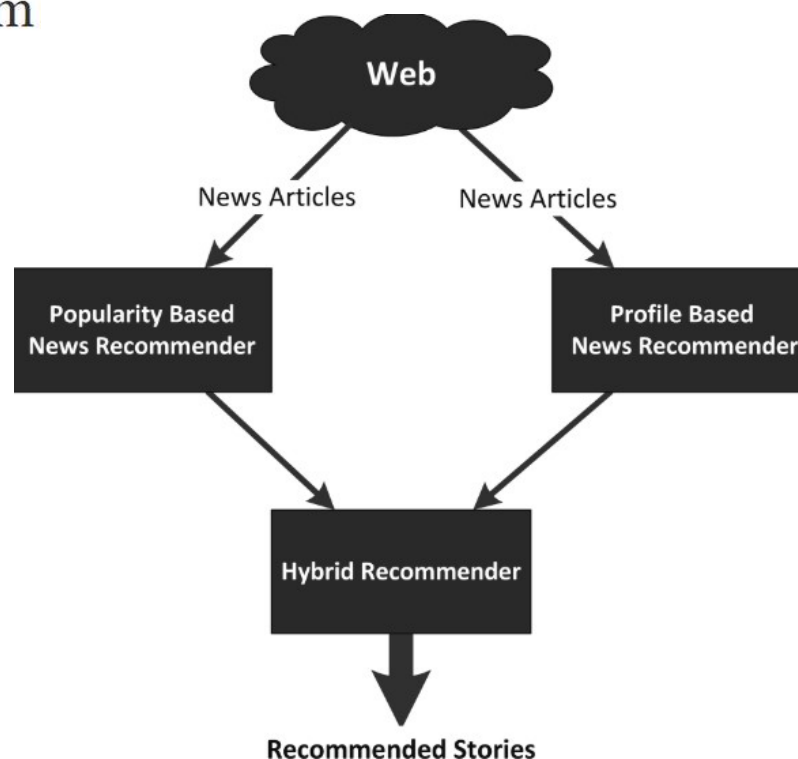


# Example: News Recommendation (1)

Incorporating popularity in a personalized news recommender system

Nirmal Jonnalagedda, Susan Gauch, Kevin Labille, Sultan Alfarhood

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



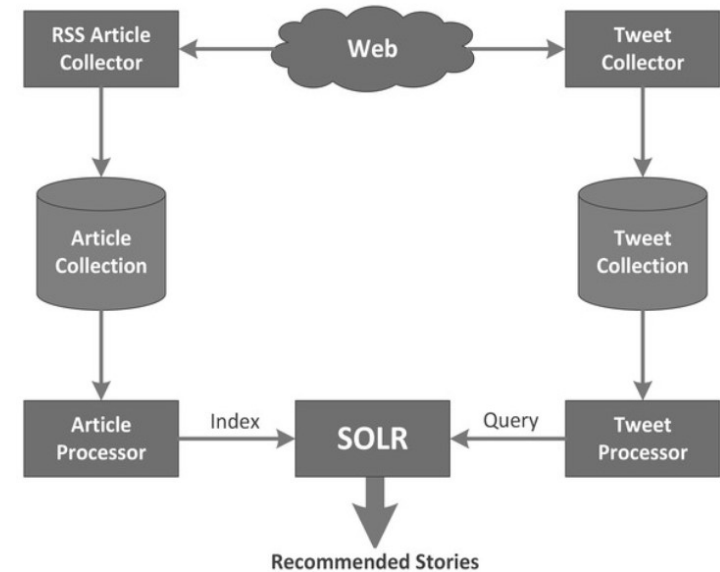
$$Hybrid\_Wt_{ij} = \alpha * Popularity\_Wt_j + (1 - \alpha) * Personal\_Wt_{ij}$$

## 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

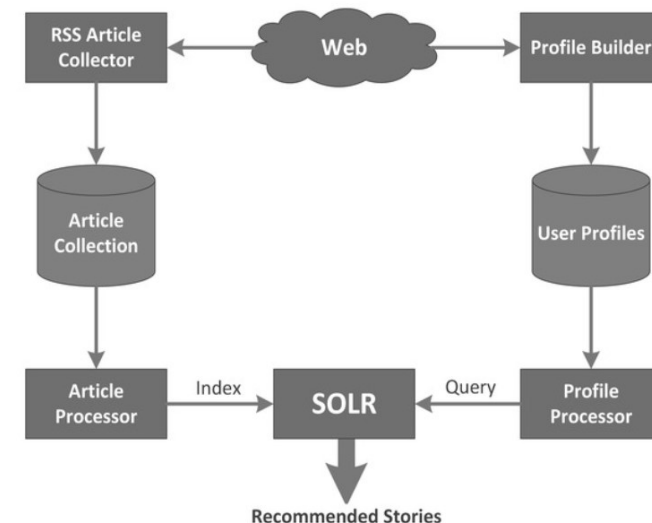


Article	Popularity_Wt
B1	1.4
E4	1.3
S6	1.0
B3	0.6
B2	0.5
S5	0.2

# 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](http://ceur-ws.org/Vol-1313/paper_7.pdf)

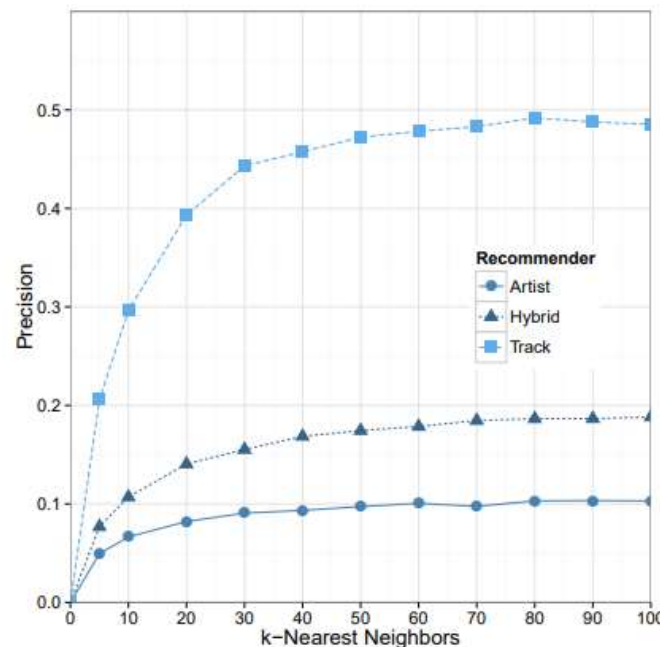


Figure 3: Precision and Recall of the Track-Based Recommender

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

# Agenda

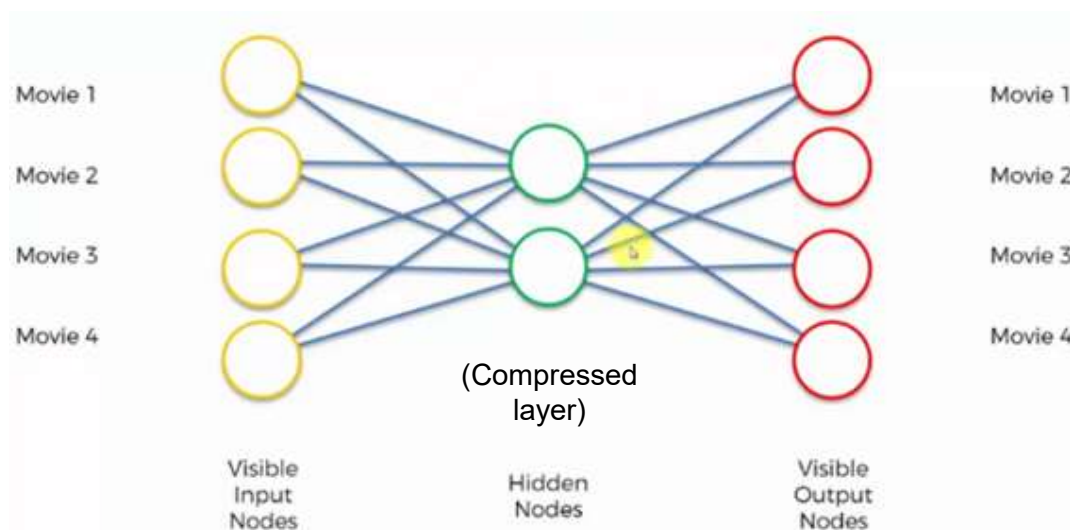


Day	Topic
Day4 (am)	Introduction – basic concepts overview
Day4 (am)	Market Basket Analysis and Recommender Systems
Day4 (pm)	Similarity-Based Recommender Systems
Day5 (am)	Model-Based Recommender Systems
Day5 (pm)	Hybrid Recommender Systems
Day5 (pm)	Advanced Methods Overview
Day5 (pm)	Course Assessment Quiz



# Recommendations using Deep Neural Networks

- **Auto-Encoders** offer one way to learn how to predict ratings for unseen items
- In Auto-Encoders, the input and output layers are identical. During training the hidden layer weights are adjusted so that the output matches the input (with minimum error).
- Often used for image compression and image reconstruction. For recommender systems the input is the user ratings vector
- The hidden (compressed) layer can be thought of as analogous to the latent features in MF



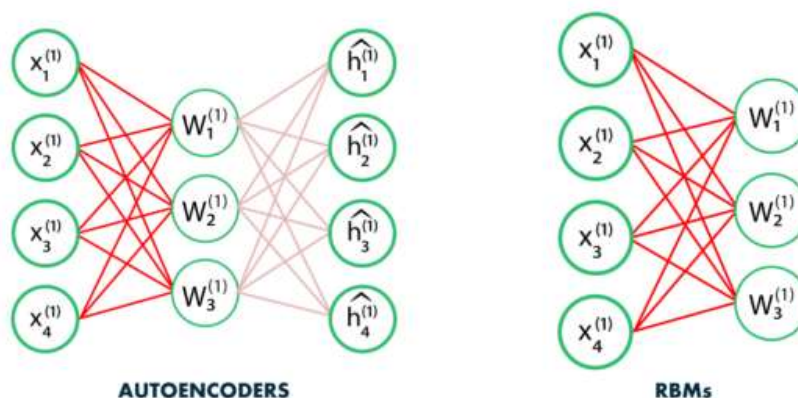
- After training, we can input a user with a few ratings and the output will contain the predicted ratings for that user for all of the movies

# RBM's and the Netflix Winning Solution

- Apart from Matrix Factorisation, the Netflix winner also used a Restricted Boltzman Machine

## Difference between Autoencoders & RBMs

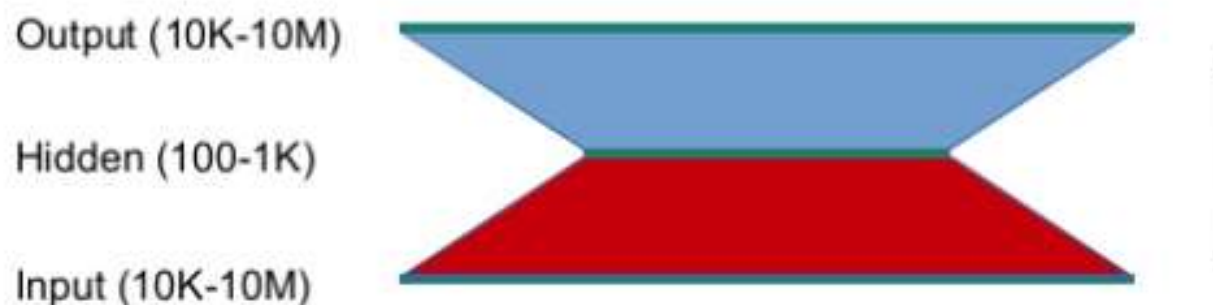
**Autoencoder** is a simple 3-layer neural network where output units are directly connected back to input units. Typically, the number of hidden units is much less than the number of visible ones. The task of training is to minimize an error or reconstruction, i.e. find the most efficient compact representation for input data.



**RBM** shares a similar idea, but it uses stochastic units with particular distribution instead of deterministic distribution. The task of training is to find out how these two sets of variables are actually connected to each other.

# Recommendation Systems at Scale

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



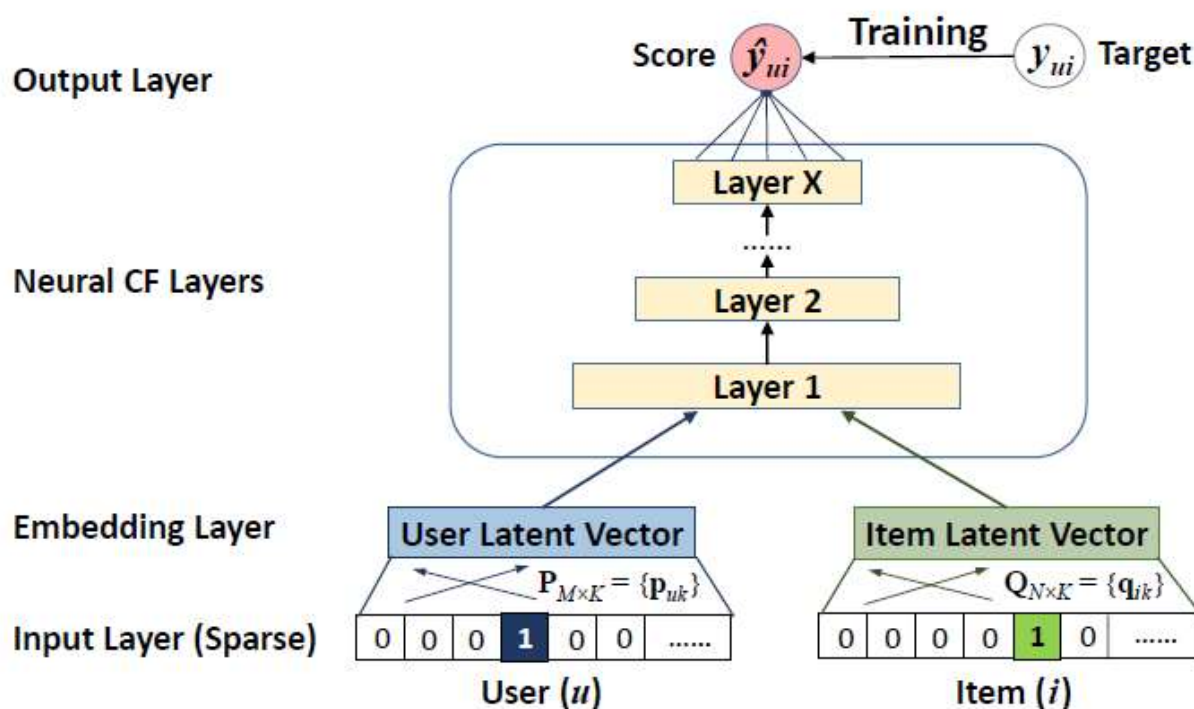
*“Our models often have hundreds of thousands of nodes in the input and output layers. At this scale, we can easily reach trillions of weights for a fully-connected network, even if it is shallow. Therefore, our models often do not fit the memory of a single GPU”*

In **model parallel training**, the model is distributed across  $N$  GPUs – the dataset 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/>

# Recommendations using Deep Neural Networks

- When using 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

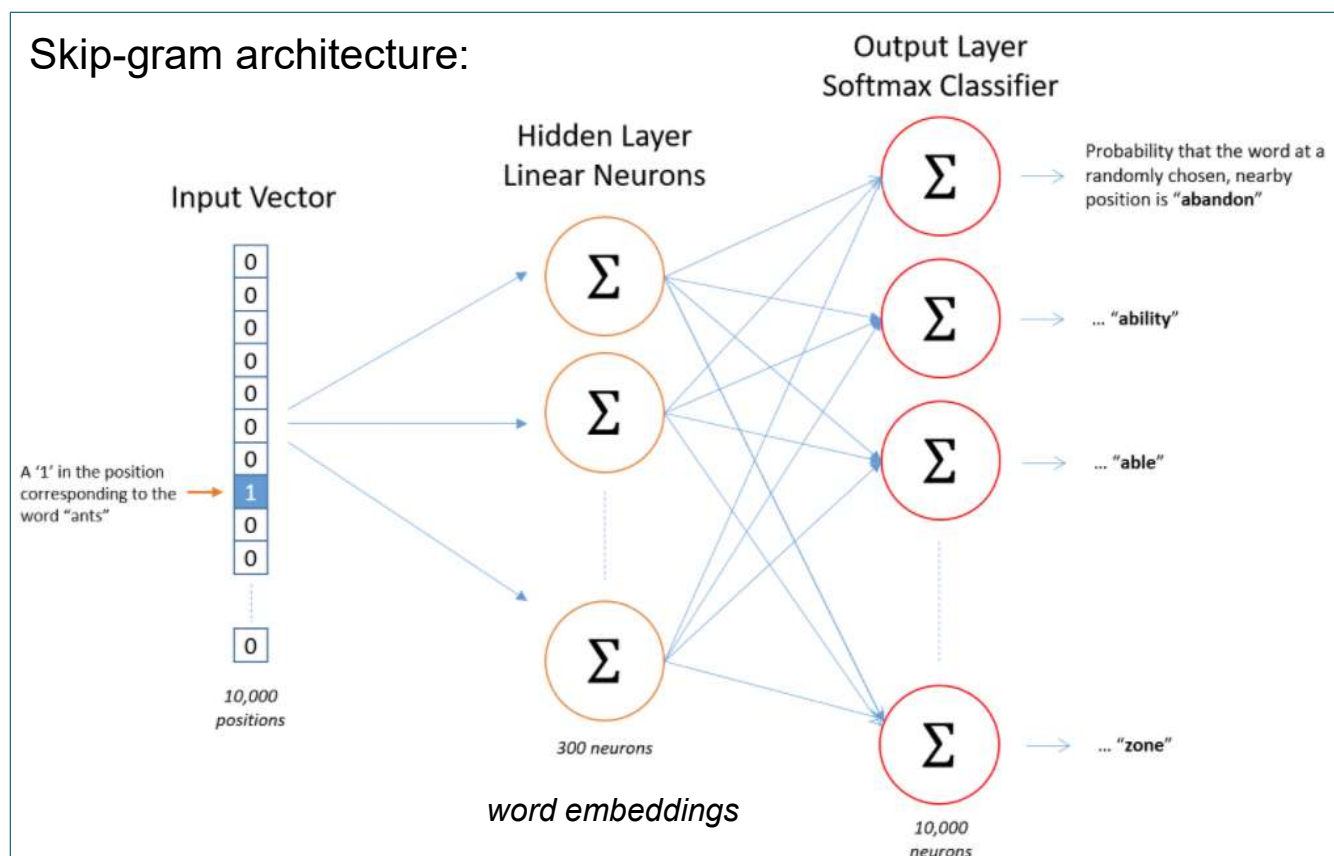


\*Neural Collaborative Filtering, see <https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf>

# Leveraging NLP methods: Prod2Vec

- Many NLP methods exist for mapping words into vector space such that similar words are near each other. The word vectors (embeddings) represent how a word interacts with other words in a corpus
- Word2Vec** (Google) trains words against other words that neighbour them in the input corpus. Skip-Gram is one architecture. Can be used to predict the next word in a sentence (language model)

Skip-gram architecture:



**Prod2Vec:** Treats products as word tokens and uses user sequences of products (analogical to sentences) to learn product embeddings. The user sequence of products is the sequence of placing items into a basket

After training, supply a product as input to get likelihoods of other products being nearby.

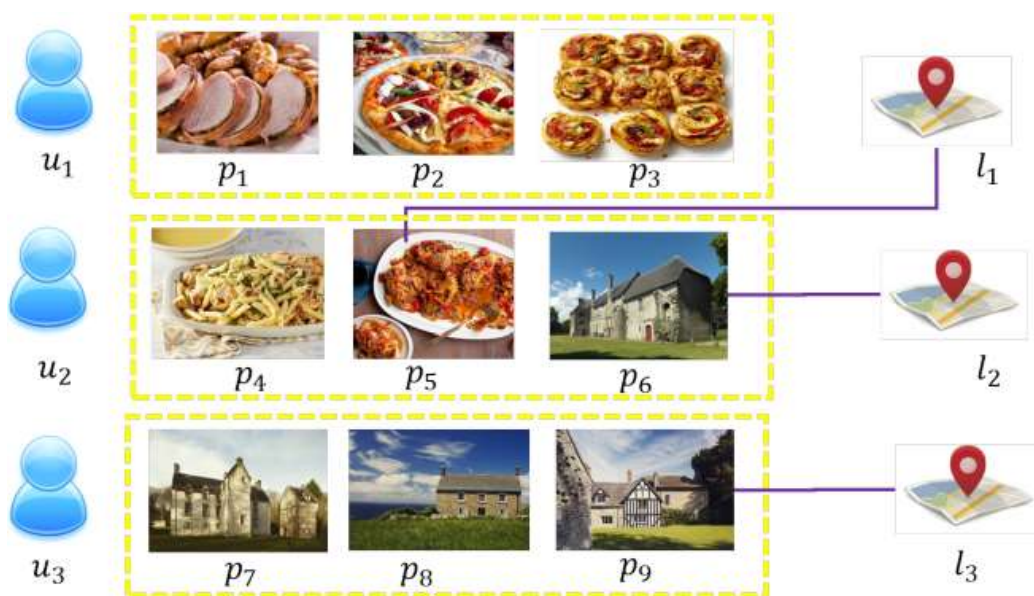
Or use the vectors (embeddings) to generate a product-product similarity matrix for content-based filtering

<http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/>



# Visual ‘Likes’: Handling Product Images

- E.g. Point of Interest Recommendation
- With location-based social networks such as Yelp and Instagram, users can upload photos of Points of Interest. These reflect user interests and also provide informative descriptions about locations. E.g. a user who posts architecture photos is more likely to visit famous landmarks; a user who posts images about food is more likely to visit restaurants.



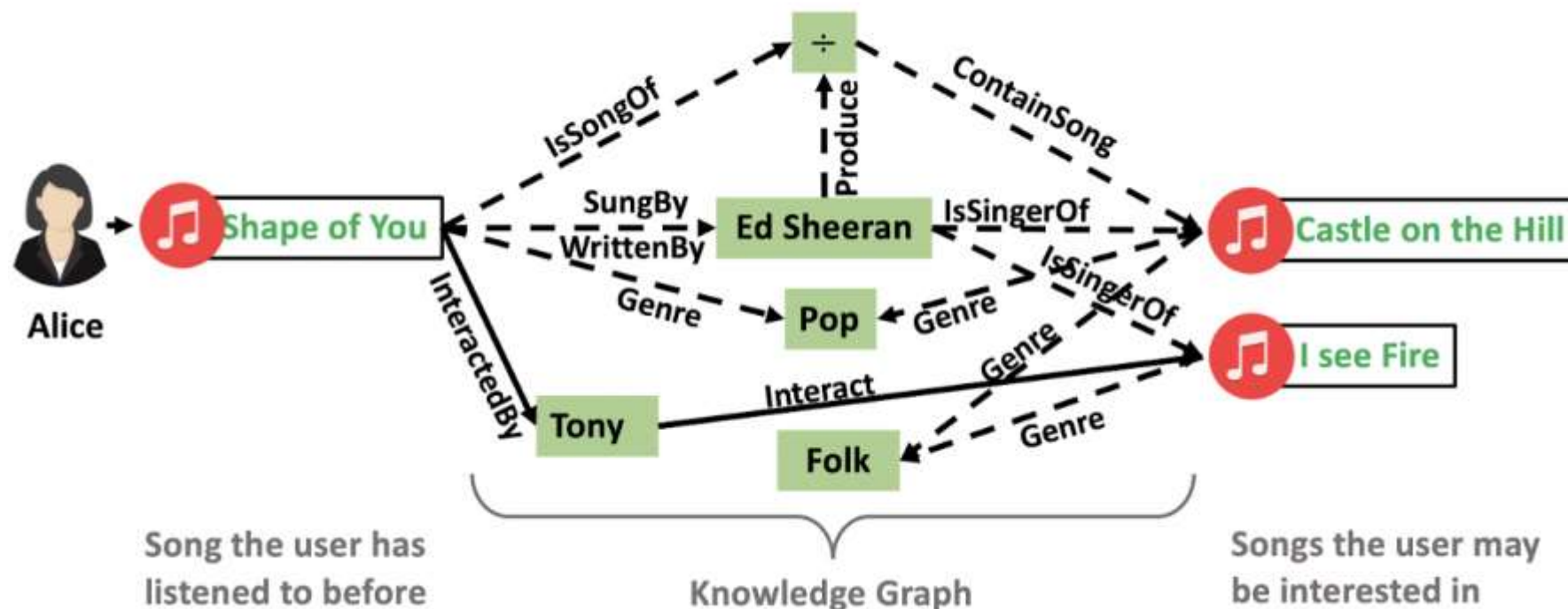
A CNN (Convolutional Neural Network) is used to extract features from the uploaded images, these are then used as inputs by a probabilistic matrix factorisation engine

[http://www.public.asu.edu/~skai2/files/WWW\\_suhang.pdf](http://www.public.asu.edu/~skai2/files/WWW_suhang.pdf)

**Example of Images Posted by Users.**

# 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.

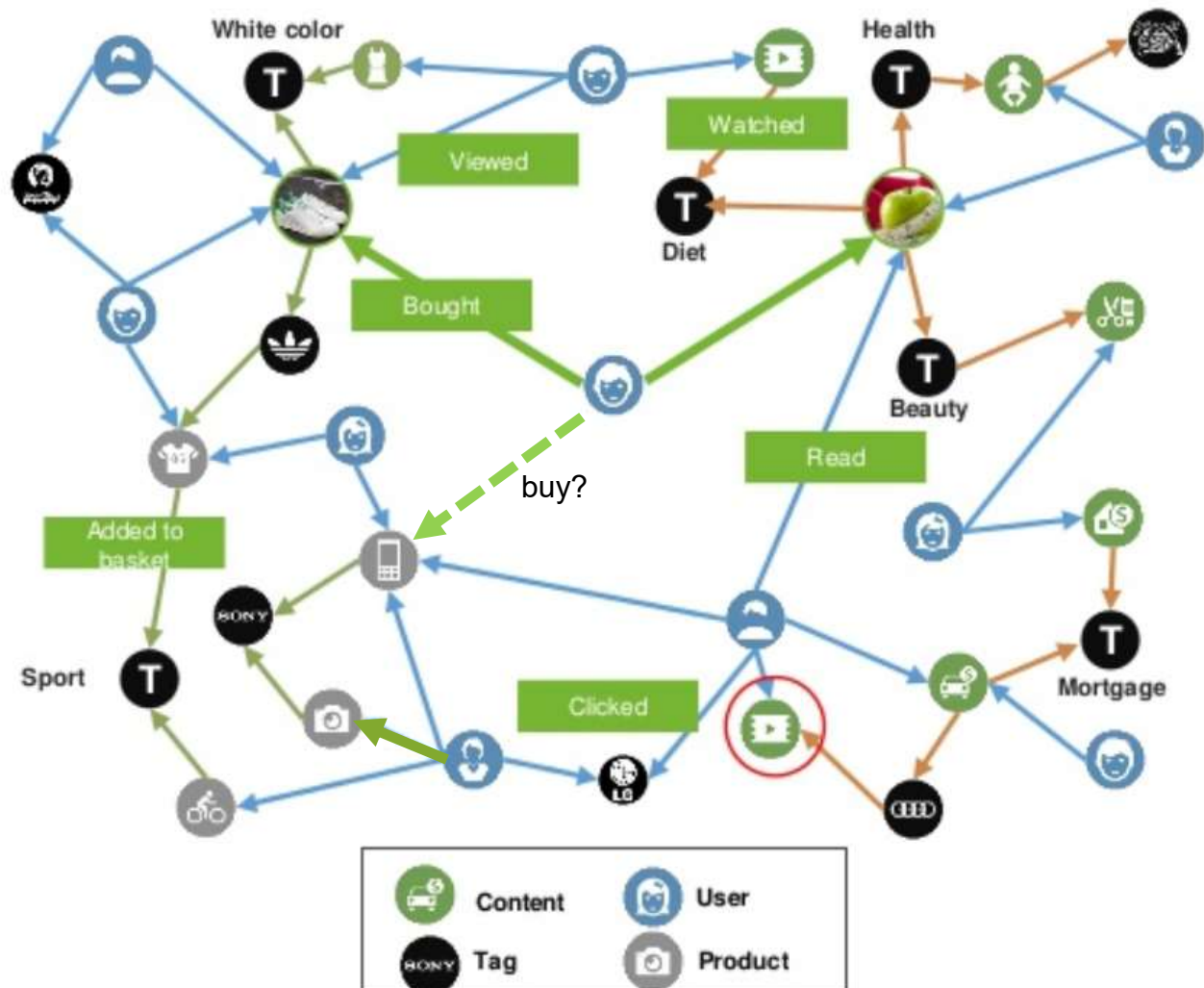


<https://tech.ebayinc.com/research/explainable-reasoning-over-knowledge-graphs-for-recommendation/>

# Recommendation using Link Prediction

- Is a link likely to exist between a user node and an item node?
- For example, user interactions with an e-retail site

- bought, added-to-basket  
→ viewed, watched, read...  
→ tagged-as





# Recommendation using Link Prediction

- Predictive Modelling Approach
  - Generate training examples by considering all possible links  $\sim (a,b)$  pairs in the graph
  - Model inputs = node properties and graph topology (relationship) features
  - Model output = T/F does a link exist

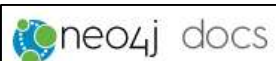
node1					node2					Topology ??				link
sex	age	income	status	...	sex	age	income	status	...					
M	45	12000	single	...	M	23	5000	married	...					Y
M	45	12000	single		F	51	7200	married						N

Model inputs

Model output

- Instead of explicitly creating/selecting the features (model inputs) , we can use ML to learn a lower dimensional vector representation of the graph nodes and topology – these representations are called embeddings

# Example: Predicting Road Links



## Node embeddings

*This chapter provides explanations and examples for the node embedding algorithms in the Neo4j Graph Data Science library.*

Node embedding algorithms compute low-dimensional vector representations of nodes in a graph. These vectors, also called embeddings, can be used for machine learning. The Neo4j Graph Data Science library contains the following node embedding algorithms:

- Production-quality
  - FastRP
- Beta
  - GraphSAGE
- Alpha
  - Node2Vec

<https://neo4j.com/developer/graph-data-science/link-prediction/>

Road	Origin cntry	Origin place	Destn cntry	Destn place	Dist.	Water crossing
E01	GB	Larne	GB	Belfast	36	FALSE
E01	GB	Belfast	IRL	Dublin	165	FALSE
E01	IRL	Dublin	IRL	Wexford	140	FALSE
E01	IRL	Wexford	IRL	Rosslare	19	FALSE

place	embedding
"Larne"	[2.1327764987945557, 1.0994384288787842, 0.49434158205986023, 2.040088176727295, 1.0031780004501343, 0.873113751411438, -2.413508176803589, -2.3154311180114746, 0.9832170009613037, -2.1525325775146484]
"Belfast"	[2.593827962875366, 0.7630942463874817, 0.7039147019386292, 2.3797214031219482, 0.7776350378990173, 0.5327662825584412, -2.3597097396850586, -1.9403077363967896, 1.2757759094238281, -1.681670904159546]
"Dublin"	[1.9023665189743042, 0.795767068862915, 0.722218930721283, 2.71242094039917, 0.21453168988227844, 0.3935716152191162, -2.1960527896881104, -2.6851491928100586, 1.0708848237991333, -0.6451507806777954]
"Wexford"	[2.0736780166625977, 1.1650514602661133, 0.4161956012248993, 2.8500888347625732, -0.12804043292999268, 0.355782151222229, -2.719728946685791, -2.6983509063720703, 0.5993242859840393, 0.265157550573349]
"Rosslare"	[1.8750609159469604, 1.2445515394210815, 0.630532443523407, 2.4588329792022705, 0.1135958656668663, 0.007978626526892185, -2.7186481952667236, -2.418004035949707, 0.3507269024848938, 0.9638977646827698]

road	Node1 (origin) embedding					Node2 (destination) embedding					Link
E01											Y
E07											Y
---											N

# 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, e.g. items from different artists. There is unlikely to be a single best recommendation, allow the user to treat the RS as a knowledge discovery tool
- **Serendipity** - how surprising are the recommendations?
- **Avoiding bad recommendations** - assigning a cost to them
- **Long versus short term recommendations.** How far ahead can the recommendation be? e.g. associations found between items over longer time
- **Repeat Recommendations** - sometimes it may be more effective to re-show recommendations or let users re-rate items, than showing new items. Users may ignore a recommendation first time but still like the item, e.g. if short of time
- **Recommending 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 03.12.10 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

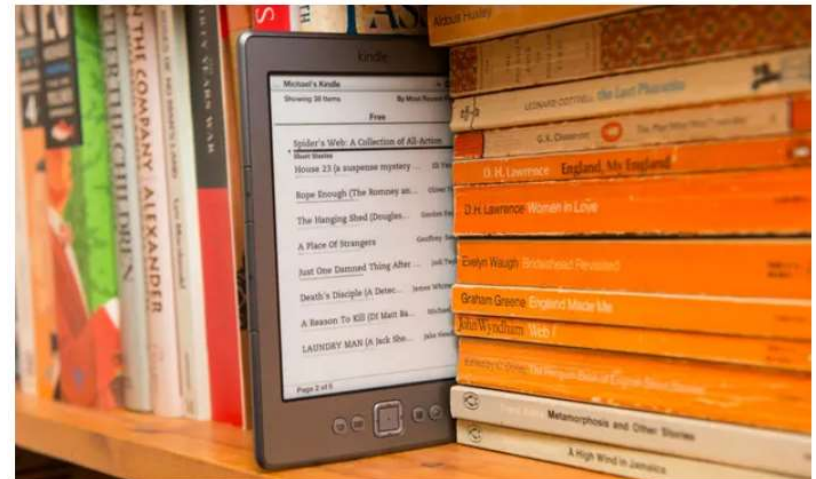
**A New York sting operation caught businesses paying for positive ratings on recommendation websites.**



## 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?

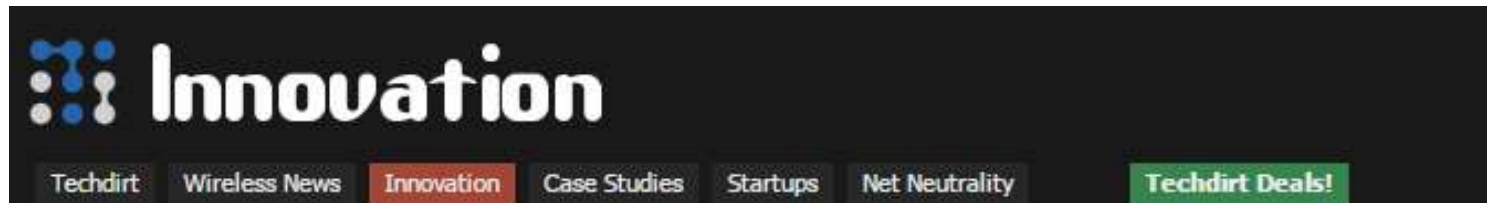
By [Paul Belleflamme](#) 26 April 2012 39

(Updated March 2015)

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



# Challenges and Issues - Computability



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

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

from the *times-change* dept

- 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.”

## Instead

...they gave us the source code. We looked at the two algorithms with the best performance in the ensemble: *Matrix Factorization* (generally called SVD, *Singular Value Decomposition*) and *Restricted Boltzmann Machines* (RBM). To put these to use, we had to overcome some limitations, for instance that they were built to handle 100 million ratings, instead of the 5 billion+ that we have, and they were not built to adapt as members added more ratings. Once we overcame those challenges, we put the two algorithms into production, where they are still used as part of our recommendation engine.

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

# Agenda



Day	Topic
Day4 (am)	Introduction – basic concepts overview
Day4 (am)	Market Basket Analysis and Recommender Systems
Day4 (pm)	Similarity-Based Recommender Systems
Day5 (am)	Model-Based Recommender Systems
Day5 (pm)	Hybrid and Advanced Recommender Systems
Day5 (pm)	Course Assessment Quiz